NetScaler® イングレスコントローラー

Copyright © 1997–2026. Citrix Systems, Inc. All Rights Reserved.

Contents

概要 30
KubernetesにおけるIngress Controllerとは 30
ネットスケーラー イングレス コントローラーとは 30
NetScaler Ingress Controllerを選ぶ理由 30
TCPおよびUDPトラフィックのサポート 30
高度なトラフィック管理ポリシー 31
柔軟なデプロイメントトポロジー 31
East-Westトラフィックに対するレイヤー7ロードバランシングのサポート 31
ベアメタルクラスター上のLoadBalancerタイプのサービス 31
NetScaler Ingress Controllerのデプロイ 32
Kubernetesベースのマイクロサービス向けNetScaler®クラウドネイティブソリューション 32
クラウドネイティブへの移行において、Kubernetesはどのように役立ちますか? 32
NetScalerクラウドネイティブソリューションとは何ですか? 33
NetScaler クラウドネイティブソリューションのコンポーネント 35
Kubernetes向けNetScaler® イングレスコントローラー 36
サポートされているプラットフォームとデプロイ 37
Kubernetesプラットフォーム 37
NetScalerとの互換性 37
NetScalerクラウドネイティブコンポーネントとのNetScaler Operatorの互換性 39
クラウドネイティブコンポーネントとNetScalerの互換性 39
リリースノート 40
NetScaler Ingress Controller (NSIC)/NetScaler GSLB コントローラー 40
IPAMコントローラー 41
参照 41
はじめに 42
始める前に 42

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Ingress Controllerの使用を開始する 42
NetScaler Ingress Controllerを使用してNetScaler CPXをデプロイする 42
サンプルアプリケーションをデプロイする 43
サンプルアプリケーションのIngressをデプロイする 43
次のステップ 44
KubernetesクラスターにNetScaler® Ingress Controllerを展開する 45
前提条件 45
KubernetesクラスターにNetScaler Ingress Controllerを展開する方法 45
次に 47
KubernetesクラスターにNetScaler® Ingress Controllerを使用してCPXを展開する 47
前提条件 47
KubernetesクラスターにNetScaler Ingress Controllerを展開する方法 47
次のステップ 48
デプロイメントトポロジー 49
シングルティアトポロジー 49
デュアルティアトポロジー 49
クラウドトポロジー 50
サービスメッシュライト 50
LoadBalancerタイプのサービス 51
NodePortタイプのサービス 52
トポロジーを選択するためのガイドライン 52
シングルティア (統合イングレス) 52
デュアルティア 52
Helmチャートを使用したデプロイ 53
NetScaler ingress controller を展開する 54
統合イングレス 54
KubernetesクラスターにNetScaler Ingress Controllerをスタンドアロンポッドとして展開する 54

© 1997–2026 Citrix Systems, Inc. All rights reserved.

前提条件 55
KubernetesクラスターにNetScaler Ingress Controllerをスタンドアロンポッドとして展開する 57
NetScaler MPXまたはNetScaler VPXでのルート追加 58
デュアルティアイングレス 59
デュアルティアイングレス展開 60
前提条件 60
デュアルティアイングレス展開 60
マルチクラスターイングレス 67
はじめに 67
デプロイメントトポロジー 68
リスナー 69
イングレス 69
マルチクラスターイングレスのデプロイ 69
前提条件 70
マルチクラスターイングレスソリューションをデプロイする手順 71
高度なユースケース 79
ポリシーバインディング 81
異なるアプリケーションを使用したマルチクラスターイングレスのセットアップ 84
Rancherで管理されるKubernetesクラスターにNetScaler Ingress Controllerを展開する 86
前提条件 86
展開オプション 86
NetScaler CPXのサイドカーとしてNetScaler Ingress Controllerを展開する 87
NetScaler Ingress Controller をスタンドアロンポッドとしてデプロイする 88
PKS管理のKubernetesクラスターにNetScaler Ingress Controllerを展開する 89
前提条件 89
展開オプション 89
NetScaler Ingress Controllerをポッドとして展開する 90

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler CPXでNetScaler Ingress Controllerをサイドカーとして展開する 90
ネットワーク構成 90
デプロイメントソリューション 91
NetScaler統合型カナリアデプロイメントソリューションの展開 91
Canary CRDを使用したカナリアの展開 91
NetScaler統合カナリアデプロイメントソリューションのためのSpinnakerパイプラインのワークフロー 93
Google Cloud PlatformでNetScaler統合カナリアデプロイメントソリューションをデプロイする 94
トラブルシューティング 104
サンプルJSONファイル 105
自動カナリア構成のサンプルJSONファイル 120
Ingressアノテーションを使用した簡素化されたカナリアデプロイ 123
重みに基づくカナリアデプロイ 123
HTTPリクエストヘッダーに基づくカナリアデプロイメント 124
HTTPリクエストヘッダー値に基づくカナリアデプロイメント 124
Ingressアノテーションを使用したカナリアデプロイメントの構成 124
NetScaler IPAMコントローラーをデプロイする 127
前提条件 127
IPAMコントローラーをデプロイする 128
IPアドレスの割り当て 128
IPAMコントローラーにおける環境変数 129
参照 130
IPAMコントローラーおよびイングレスコントローラーとのBIND DNS統合 130
前提条件 130
BIND DNSを使用したIPAMコントローラーのデプロイ 130
ユースケース 131
イングレス 131

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サービス 134
NetScaler GSLB コントローラー 136
GSLB の概要と展開トポロジ 136
概要 136
展開トポロジ 137
GSLBメソッドとサポートされる展開タイプ 139
分散Kubernetesクラスターにデプロイされたアプリケーション向けNetScaler GSLBコントローラーを構成するためのCRD 140
NetScaler® GSLBコントローラーの展開 145
GSLB構成を同期する 149
グローバル・トラフィック・ポリシー (GTP) 展開の例 150
例 6: 送信元IPの永続性 157
グローバルサービスエントリ (GSE) の例 158
GSLBのマルチモニターサポート 159
NetScaler® GSLBコントローラー（シングルサイト用） 161
概要 161
NetScaler GSLBコントローラーの展開 162
サービスメッシュライト 183
kube-proxyによる東西通信 184
サービスメッシュライトアーキテクチャにおけるNetScaler CPXによる東西通信 184
ステップ1：コーヒーサービス定義をNetScaler CPXを指すように変更する 184
ステップ2：コーヒーマイクロサービスPod用に `coffee-headless` という名前のヘッドレスサービスを作成する 185
ステップ3： `coffee-headless` サービス用のルールを含むIngressリソースを作成する 186
Service Mesh liteアーキテクチャにおけるNetScaler MPXまたはVPXとのEast-west通信 186
ステップ1: coffeeホスト名をNetScaler MPX/VPX IPアドレスに解決する外部サービスを作成する 186
ステップ2: マイクロサービスPod用のヘッドレスサービスを作成する 188

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ステップ3: Ingressリソースを作成する 188
Service Mesh lite でのアプリケーションの自動展開 189
NetScaler® オペレーター 190
リリースノート 190
NetScaler オペレーター 190
NetScaler® Operator リリースノート 191
バージョン 3.6.0 191
NetScaler® Operator リリースノート 197
バージョン 3.5.0 197
NetScaler® Operator リリースノート 197
バージョン 3.4.1 197
NetScaler® Operator リリースノート 198
バージョン 3.3.0 198
NetScaler® Operator リリースノート 199
バージョン 3.2.0 199
NetScaler® Operator リリースノート 199
バージョン 3.1.2 199
NetScaler® Operator のアップグレード 200
NetScaler Operator をデプロイする 202
NetScaler Operator を使用して OpenShift に NetScaler Ingress Controller を展開する 204
前提条件 204
NetScaler Operator を使用して NetScaler Ingress Controller をスタンドアロンポッドとして展開する 205
参照 228
NetScaler Operator を使用して OpenShift に CPX と共に NetScaler Ingress Controller (NSIC) を展開する 228
前提条件 228
NetScaler CPX と共にサイドカーとして NetScaler Ingress Controller を展開する 229

© 1997–2026 Citrix Systems, Inc. All rights reserved.

参考資料 255
NetScaler Operator を使用して NetScaler Cloud Controller をデプロイする 255
はじめに 255
前提条件 255
サービスプリンシパルを作成してロールを割り当てる 256
ケース I: ルートを作成するサブネットが ARO クラスターの一部ではない場合 256
ケースII: ルートが作成されるサブネットがAROクラスターの一部である場合 259
NetScaler Cloud Controllerインスタンスをデプロイする手順 261
NetScaler Operator を使用して OpenShift に NetScaler GSLB Controller をデプロイする 265
NetScaler® GSLB Controller をデプロイする 265
前提条件 265
NetScaler Operator を使用して NetScaler GSLB Controller をデプロイする手順 265
参照 271
NetScaler Operator を使用して、NetScaler® Ingress Controller をスタンドアロンポッドとしてデプロイする 271
前提条件 272
デプロイ手順 272
NetScaler IPAM Controllerのパラメータ 276
必須およびオプションのパラメータ 276
ローカルIPAMパラメータ 277
Infobloxパラメータ 277
NetScaler® Observability Exporter (NSOE) を NetScaler Operator を使用してデプロイする 278
前提条件 278
NetScaler Operator を使用して NetScaler Observability Exporter をデプロイする 278
プラットフォーム統合 286
OpenShift 286
NetScaler Kubernetes Ingress Controller を OpenShift ルータープラグインとして展開する 286
サポートされているルートタイプ 287

© 1997–2026 Citrix Systems, Inc. All rights reserved.

展開オプション 287
OpenShiftでサポートされているCitrixコンポーネント 289
例 289
OpenShiftルーターシャーディングサポート付きNetScaler Ingress Controllerのデプロイ 299
OpenShiftルーターシャーディングでNetScaler CPXを展開する 301
OpenShiftルーターシャーディングサポート付きNetScaler Ingress Controllerルータープラグインを展開する 301
例: OpenShiftルートを作成し、NetScaler VPXでルート構成を確認する 301
クラウド 304
Azure Kubernetes ServiceクラスターにNetScaler CPXをIngressデバイスとして展開する304
AKSクラスターにNetScaler CPXをIngressデバイスとして展開する 305
クイックデプロイ 307
NetScaler Ingress ControllerをNetScaler VPXと共にAzure Kubernetes Serviceクラスターに展開する 308
前提条件 308
トポロジ 308
Azure MarketplaceからNetScaler VPXインスタンスを取得する 309
Azure マーケットプレイスからNetScaler Ingress Controllerを取得する 309
NetScaler Ingress Controllerのデプロイ 309
サンプルアプリケーションを使用してデプロイを検証する 311
Google Cloud Platform に Ingress デバイスとして NetScaler CPX を展開する 312
前提条件 312
Google Cloud Platform に Ingress デバイスとして NetScaler CPX を展開する 312
クイックデプロイ 314
Anthos に NetScaler Ingress Controller を展開する 315
Anthos GKE オンプレミスで NetScaler CPX を `LoadBalancer` タイプのサービスとして公開する 315
デュアルティアイングレス展開 320

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Amazon ELBとNetScaler Ingress Controllerを使用して、EKS環境でNetScaler VPXをアクティブ-アクティブ高可用性で展開する 324

ソリューションの概要 324

展開プロセス 325

トラブルシューティング 333

AWSでLoadBalancerタイプのサービス向けCitrix®ソリューションを展開する 334

前提条件 335

Helmチャートを使用してAWSでLoadBalancerタイプのサービス向けCitrixソリューションを展開する 335

YAMLを使用してAWSでLoadBalancerタイプのサービス向けCitrixソリューションをデプロイする 336

Amazon EKSおよびMicrosoft AKSクラスターを使用したマルチクラウドおよびGSLBソリューション 338

展開トポロジ 338

前提条件 338

AWSへのNetScaler VPXのデプロイ 339

AzureでのNetScaler VPXの展開 341

AWS および Azure に展開された NetScaler VPX で ADNS サービスを構成する 342

AWSおよびAzureにデプロイされたNetScaler VPXでGSLBサービスを設定する 343

AWSおよびAzure KubernetesクラスターにGTPおよびGSE CRDを適用する 344

GSLBコントローラーの展開 344

Amazon EKSおよびMicrosoft AKSクラスターのイングレスおよびGSLBデバイスとしてのNetScaler VPX 352

VMware クラウドファンデーション上の vSphere Kubernetes サービスにおけるネットスケーラー 356

vSphere Kubernetes Service の概要 356

Kubernetes 向け NetScaler 356

ソリューションアーキテクチャ 357

ソリューションの検証 358

© 1997–2026 Citrix Systems, Inc. All rights reserved.

VKS に NetScaler をデプロイする 359
前提条件 359
パートI: ゲートウェイAPI — ネットスケーラー Kubernetes ゲートウェイコントローラー 360
第II部: イングレスAPI — NetScalerイングレスコントローラー 368
トラブルシューティング 376
結論 377
リソース 378
設定 379
アノテーション 380
Ingressアノテーション 380
サービスアノテーション 412
HTTP、TCP、またはSSLプロファイル用のスマートアノテーション 425
Ingressのスマートアノテーション 427
スマートアノテーションを使用したIngress YAMLの例 428
ルートのスマートアノテーション 430
スマートアノテーションを使用したルートマニフェストの例 431
安全でないトラフィックをリダイレクトするためのサービスアノテーションを含むYAMLの例 433
サービス用スマートアノテーション 434
サービス用のスマートアノテーションを含むIngress YAMLの例 436
ServicetypeLB: スマートアノテーションのイベント変更 439
例 440
不安全なサービスタイプアノテーションにおけるSIP_UDPサポートのIngress YAMLの例 440
アノテーションキー `ingress.citrix.com/default-response-code` 441
respondwith (デフォルト) 443
リダイレクト 443
レスポンドウィズHTMLページ 443
例とターゲットURLルール 444

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Ingress Controller の ConfigMap サポート 446
NetScaler Ingress Controller でサポートされている環境変数 447
NetScaler Ingress ControllerのConfigMapサポートの構成 448
Ingress の構成 452
ホスト名ベースのルーティング 452
パスベースのルーティング 454
ワイルドカードホストルーティング 455
正確なパスの一致 456
非ホスト名ルーティング 458
デフォルトのバックエンド 460
Ingressクラスのサポート 461
Ingressクラスとは? 461
ネットスケーラー イングレス コントローラー と イングレス クラス 461
Ingressクラスを含むYAML設定の例 461
Ingress V1とIngressClassのサポート 463
指定されたIPアドレスを持つIngressリソースのIngressステータスを更新する 466
サイドカーデプロイメントのIngressステータス更新 466
Ingressステータス更新後のIngress出力例 467
LoadBalancer タイプのサービス用サービス クラス 467
サービス クラスを含む YAML 設定の例 468
NetScalerでHTTP、TCP、またはSSLプロファイルを構成する 470
Kubernetes環境におけるNetScaler構成を理解する 470
HTTPプロファイル 471
TCPプロファイル 472
SSLプロファイル 472
前提条件 472
アノテーションを使用したフロントエンドプロファイルの構成 474
ConfigMap変数を使用したグローバルフロントエンドプロファイル構成 477

© 1997–2026 Citrix Systems, Inc. All rights reserved.

FRONTEND_HTTP_PROFILEを使用した構成 477
FRONTEND_TCP_PROFILEを使用した構成 478
FRONTEND_SSL_PROFILEを使用した構成 480
バックエンド構成 482
TLS構成なしのバックエンドイングレスマニフェストの例 482
TLS構成ありのバックエンドイングレスマニフェストの例 483
Ingress NetScalerで組み込みまたは既存のユーザー定義プロファイルを使用する 485
HTTPプロファイルの例 486
TCPプロファイルの例 486
SSLプロファイルの例 486
HTTP、SSL、およびTCPプロファイルを適用する例 486
例: SNI証明書をSSL仮想サーバーに追加する 490
例: SSL暗号グループのバインド 494
ログレベル 495
ログレベルの変更 497
Kubernetesシークレットリソースを介したLoadBalancerタイプのサービス用SSL証明書 498
NetScaler Ingress Controllerのデフォルト証明書を使用する 498
Kubernetesシークレットとしての SSL証明書のためのサービスアノテーション 498
例: フロントエンドシークレットとフロントエンドCAシークレット 500
例: バックエンドシークレットとバックエンドCAシークレット 501
NetScaler CPXを使用したLoadBalancerタイプのサービスおよびIngressの外部IPアドレスのBGPアドバタイズメント 502
LoadBalancerタイプのサービス向けNetScaler CPXソリューションをデプロイする 504
前提条件 504
BGP構成 509
ノードごとに異なるネイバー 512
サービス構成 513
外部IPアドレス構成 513

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サービスアノテーション構成 514
外部トラフィックポリシー構成 514
Ingressリソースの使用 515
Helmのインストール 516
トラブルシューティング 516
オンプレミスKubernetesクラスター向けMetalLBレイヤー2モードでのNetScaler CPX統合 517
MetalLBをインストールして構成する 518
レイヤー2のMetalLB構成 519
NetScaler CPX サービスインストール 520
サンプルアプリケーションをデプロイする 522
その他のリファレンス 523
Kubernetes IngressのHTTPRoute CRDを使用した高度なコンテンツルーティング 523
HTTPRoute CRDを使用してKubernetes Ingressの高度なコンテンツルーティングを設定する 524
前提条件 524
Ingressリソースをデプロイする 524
HTTPRouteリソースをデプロイする 526
Listener CRDのプロファイルサポート 528
HTTPプロファイル 528
TCPプロファイル 530
SSLプロファイル 532
SSL暗号 534
アナリティクスプロファイル 537
IPAMコントローラーを使用したIPアドレス管理 540
IPAMコントローラーとInfobloxの統合 541
IPAMコントローラーを使用してIPアドレスを割り当てる 542
ステップ1: VIP CRDを展開する 542
ステップ 2: NetScaler Ingress Controller を展開する 542

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ステップ 3: IPAM コントローラーを展開する 544
ステップ4：Ingressリソースを展開する 548
IPアドレスの割り当て 552
複数のIPアドレス割り当て 552
アノテーションを介してCRDを適用する 553
CRDを参照するためのIngressアノテーション 553
RewriteおよびResponder CRDを参照するためのLoadBalancerタイプサービスのアノテーション 555
アノテーションによるIngressのリスナーCRDサポート 555
制限事項 556
Ingress 用の Listener CRD リソースのデプロイ 556
証明書の管理 560
Ingressリソースによる証明書の管理 560
Listenerリソースによる証明書の管理 562
リスナーにおけるSSLパススルーのサポート 564
リスナーCRDの例 565
イングレス YAML 565
NetScaler Ingress Controller を使用した一貫性ハッシュアルゴリズムの構成 566
ハッシュアルゴリズムの構成 567
NetScaler Ingress Controller を使用して DNS レコードを追加する 568
IngressリソースのDNSレコードを追加する 568
LoadBalancerタイプのサービスのDNSレコードを追加する 568
NetScaler を使用した Kubernetes の Open Policy Agent サポート 570
使用例 571
例1：クライアントの送信元IPアドレスに基づいてリソースへのアクセスを許可または拒否する 571
例2：認証後のユーザーグループに基づいてアクセスを許可または拒否する 573
例3：認証中に取得された認証属性に基づいてアクセスを許可または拒否する 579

© 1997–2026 Citrix Systems, Inc. All rights reserved.

メトリクスをPrometheusに直接エクスポートする 584

NetScaler CPXからPrometheusへのメトリクスの直接エクスポートを構成する 585

NetScaler VPX または NetScaler MPX から Prometheus へのメトリクスの直接エクスポートの構成 587

管理パーティションを持つNetScaler向けNetScaler Ingress Controllerを展開する 588

前提条件 589

管理パーティションを持つNetScaler向けNetScaler Ingress Controllerを展開するには 589

ユースケース: NetScaler Consoleのパーティションを使用してマルチテナントのマイクロサービスベースアプリケーションを安全に提供する方法 592

例 593

名前空間を作成する 593

名前空間 `ns1` での構成 593

名前空間 `ns2` での構成 597

ネットワーク 606

NetScaler VPX または MPX で静的ルートを構成する 606

NetScaler でルートを手動で構成する 607

NetScalerでルートを自動的に構成する 607

ノードコントローラーを使用してKubernetesノードとIngress NetScaler間のネットワークを確立する 608

自己修復ルーターポッドとネイティブヘルスチェック 609

前提条件 610

NetScalerノードコントローラーを展開する 611

クラスターデプロイメントの検証 614

NetScaler Kubernetes ノードコントローラーの削除 614

インターネットアクセスなしで NetScaler ノードコントローラーを実行する 615

内部リポジリのイメージを使用するように NetScaler ノードコントローラーを設定する 615

同じクラスターで複数のNetScalerノードコントローラーを実行する 615

NetScalerノードコントローラーを使用したOpenShift向けOVN-CNIサポート 615

© 1997–2026 Citrix Systems, Inc. All rights reserved.

OVN-Kubernetesを使用してOpenShiftをデプロイする 616
クリーンアップ / アンインストール 619
制限事項 619
Ingress を使用して NodePort タイプのサービスを公開する 619
展開例 619
Calico を使用した Pod 間通信の構成 623
Pod 間通信 623
カリコ 624
Kubernetes と Calico の統合 624
トラブルシューティング 629
NetScaler Ingress ControllerにおけるLoadBalancerタイプのKubernetesサービスサポートの機能強化 630
NetScalerでのBGP RHIの自動構成のサポート 630
ポッドの可用性に基づいてVIPをアドバタイズおよびリコールする 631
NetScaler Ingress Controllerを使用したNetScalerでのBGP RHIの構成 631
証明書管理 639
NetScaler Ingress ControllerにおけるTLS証明書の処理 639
前提条件 640
NetScaler Ingress Controllerのデフォルト証明書 641
事前設定された証明書 654
ingress YAML の TLS セクション 655
注意点 656
NetScalerにおけるTLSクライアント認証のサポート 657
TLSクライアント認証の構成 657
NetScaler Ingress Controller を使用した NetScaler での TLS サーバー認証サポート 658
TLS サーバー認証の構成 659
NetScaler Ingress Controller を使用して NetScaler 上で証明書をインストール、リンク、更新する 660

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Kubernetesでの証明書のリンク 661

NetScaler Ingress Controller を使用した NetScaler® の証明書キーバンドル 663

NetScaler Ingress ControllerでのSSL CA証明書バンドルのサポート 664

NetScaler Ingress Controller を介して NetScaler で作成された事前設定済み証明書バンドルのサポート 667

参照 667

Kubernetes Ingress を使用した SSL パススルーの構成 668

cert-manager を使用した自動証明書管理 670

NetScaler Ingress Controllerとcert-managerを使用して、Kubernetes上にHTTPSウェブアプリケーションをLet`s Encryptでデプロイする 670

前提条件 671

cert-managerをインストールする 672

サンプルWebアプリケーションをデプロイする 672

HTTPチャレンジを使用したACME証明書の発行を構成する 678

DNSチャレンジを使用してACME証明書を発行する 687

NetScaler で証明書を検証する 693

本番環境に移行する 697

トラブルシューティング 699

cert-managerを使用して、NetScaler Ingress ControllerとHashiCorp VaultでKubernetes上にHTTPSウェブアプリケーションをデプロイする 702

前提条件 703

マニフェストファイルを使用してcert-managerを展開する 704

サンプルWebアプリケーションを展開する 706

HashiCorp Vaultを認証局として構成する 712

AppRoleベースの認証を作成する 715

Kubernetesでの証明書発行を構成する 717

NetScaler Ingress Controller で NetScaler 証明書検証を有効にする 730

前提条件 731

© 1997–2026 Citrix Systems, Inc. All rights reserved.

証明書検証のために NetScaler Ingress Controller を構成する 731
APIサーバー証明書の検証を無効にする 734
NetScaler Ingress ControllerまたはGSLBイングレスでAPIサーバー証明書の検証を無効にする 734
自己署名証明書を作成し、Kubernetesシークレットにリンクする 735
自己署名証明書を作成する 735
証明書をKubernetesシークレットにリンクする 736
Ingressをデプロイする 736
CRD 737
OWASP CRD を使用して OWASP Top 10 保護ポリシーを構成する 738
OWASP CRD 定義 739
OWASP CRD 構造 739
ボット保護 740
WAF保護 747
OWASP CRD をデプロイする 755
完全な例: 組み合わせたWAFおよびボットポリシー 756
NetScaler を使用した Kubernetes の認証および認可ポリシー 762
Auth CRD の定義 763
Auth CRD の属性 763
Auth CRD のデプロイ 773
認証ポリシーと認可ポリシーの記述方法 774
NetScalerを使用したKubernetesでのレート制限 793
レート制限CRDの定義 794
レート制限CRDの属性 794
レート制限CRDを展開する 799
レートベースのポリシー構成の記述方法 800
Kubernetesでリライトおよびレスポンダーポリシーを使用する 803
NetScaler®リライトおよびレスポンダーCRDを展開する 804

© 1997–2026 Citrix Systems, Inc. All rights reserved.

リライトおよびレスポンダー CRD 属性 804
ポリシー構成の記述方法 807
ポリシー設定の例 811
使用例 816
関連資料 832
ICAPを使用したリモートコンテンツ検査またはコンテンツ変換サービス 833
NetScaler® ICAP CRDのデプロイ 834
ICAP CRD属性 834
バックエンドサービスとイングレスクラスの指定 835
ポリシー構成の記述方法 843
ポリシー設定の例 847
VIPカスタムリソース定義 (CRD) 853
参照 853
HTTPルート 854
HTTP CRD オブジェクトの例 855
HTTPRoute.spec 857
HTTPRoute.rules 858
HTTPRoute.rules.match 859
HTTPRoute.rules.match.path 860
HTTPRoute.rules.match.headers 862
HTTPRoute.rules.match.headers.headerName 863
HTTPRoute.rules.match.headers.headerValue 865
HTTPRoute.rules.match.cookies 868
HTTPRoute.rules.match.queryParams 871
HTTPRoute.rules.action 873
HTTPRoute.rules.action.backend 874
HTTPRoute.rules.action.backend.kube 874
BackendConfig 875

© 1997–2026 Citrix Systems, Inc. All rights reserved.

HTTPRoute.rules.action.redirect 877
NetScalerによるKubernetes向け高度なコンテンツルーティング 879
リスナー CRD 879
Listener CRDオブジェクトの記述方法 880
HTTPRoute カスタムリソース定義 884
HTTPRoute CRD を展開する 884
HTTPRoute CRD オブジェクトの記述方法 885
HTTPRoute CRDオブジェクトをListener CRDオブジェクトにアタッチする 887
リスナー 888
リスナーCRDオブジェクトの例 888
Listener.spec 891
Listener.certificates 895
Listener.certificates.Secret 897
Listener.routes 898
Listener.action 900
Listener.action.backend 900
Listener.action.backend.kube 901
BackendConfig 902
Listener.action.redirect 903
Listener.policies 905
Listener.policies.tcpprofile 907
Listener.policies.httpprofile 909
Listener.policies.sslprofile 910
Listener.policies.analyticsprofile 912
Listener.policies.analyticsprofile.config 913
NetScaler Ingress Controller を使用して Web アプリケーションファイアウォールポリシーを構成する 914
一般的な保護 915

© 1997–2026 Citrix Systems, Inc. All rights reserved.

HTML 保護 915
JSON保護 915
XML保護 915
WAF CRDの定義 916
WAF CRD属性 916
WAF CRDを展開する 918
WAF設定の記述方法 918
NetScaler Ingress Controller を使用してボット管理ポリシーを構成する 942
ボット CRD の定義 943
Bot CRDの属性 943
Bot CRDを展開する 944
Bot構成の記述方法 944
NetScaler Ingress Controller を使用してクロスオリジンリソース共有ポリシーを構成する 953
CORS とは 954
CORS プリフライト 954
CORS CRD の定義 954
CORS CRDを展開する 955
NetScaler Ingress Controller 用の AppQoE を使用したリクエスト再試行機能の有効化 957
AppQoE CRD の定義 958
AppQoE CRD を展開する 958
NetScaler Ingress Controller を介したワイルドカード DNS ドメインの設定 960
使用ガイドラインと制限事項 960
ワイルドカード DNS CRD の定義 960
Wildcard DNS CRD をデプロイする 961
Wildcard DNS 構成ポリシーの記述方法 961
メトリクス 963
PrometheusとGrafanaを使用してNetScalerのメトリクスを表示する 963
NetScaler メトリクス エクスポーター 963

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Prometheus Operatorを起動する 963
NetScaler Metrics Exporter の構成 966
分析と可観測性 978
NetScaler Ingress Controller YAMLファイルでアノテーションを使用して分析を有効にする 978
NetScaler Consoleを使用した分析 981
オープンソースツールによる分析 982
ConfigMap を使用した分析構成のサポート 982
NS_ANALYTICS_CONFIG のスキーマ 982
分析構成用の ConfigMap の作成 988
ConfigMap を使用した分析構成のサポートされている環境変数 989
ユースケース 992
IPAMコントローラーを使用したIPアドレス管理 993
LoadBalancerタイプのサービスの概要 993
LoadBalancerタイプのサービス向けNetScaler IPAMソリューション 993
IPAMコントローラーを使用したIPアドレス管理 994
IPAMコントローラーによって割り当てられたIPアドレスを持つLoadBalancerタイプのサービスを公開する 994
NetScaler Ingress Controllerを展開する 995
IPAMコントローラーのデプロイ 995
サンプルアプリケーションのデプロイ 996
LoadBalancerタイプのサービスを使用してサンプルアプリケーションを公開する 997
サービスへのアクセス 999
Ingress の保護 999
TLS v1.3プロトコルを有効にする 1000
HTTP厳格トランスポートセキュリティ (HSTS) 1001
OCSPステープリング 1002
クライアント認証を必須に設定する 1002

© 1997–2026 Citrix Systems, Inc. All rights reserved.

TLSセッションチケット拡張 1003
SSLセッションの再利用 1004
暗号グループの使用 1004
暗号リダイレクトの使用 1005
TCPのユースケース 1006
TCPイングレストラフィックを負荷分散する方法 1006
TCPベースのイングレストラフィック 1006
LoadBalancerタイプのサービスに対するTCPベースのトラフィック 1008
SSL over TCPに基づいたイングレストラフィックの負荷分散 1009
TCPベースのアプリケーションのパフォーマンスを監視し、改善する 1010
NetScaler CPXサービスで非標準HTTPポートを公開する方法 1012
Helmチャートのデプロイの場合 1012
OpenShift Operatorを使用するデプロイの場合 1012
TCPプロファイルのサポート 1013
LoadBalancerタイプのサービスに対するTCPプロファイルのサポート 1013
Ingressアノテーションを使用したTCPプロファイルの構成 1020
HTTPユースケース 1034
LoadBalancerタイプのサービスに対するHTTPプロファイルのサポート 1034
ユーザー定義HTTPプロファイル 1034
組み込みHTTPプロファイル 1035
例: HTTPプロファイル構成を持つLoadBalancerタイプのサービス 1035
Ingressアノテーションを使用してHTTPプロファイルを構成する 1041
HTTP/2の構成 1043
HTTP/2アップグレード 1044
直接HTTP/2 1044
代替サービス (ALT-SVC) を使用した直接HTTP/2 1045
HTTPセッションタイムアウトの処理 1045
reqTimeout と reqTimeoutAction 1045

© 1997–2026 Citrix Systems, Inc. All rights reserved.

adptTimeout 1046

reusePoolTimeout 1046

Ingressアノテーションを使用したHTTPプロファイルの適用例 1047

リライトおよびレスポンダーポリシーによるHTTPコールアウト 1051

リライトおよびレスポンダーCRDを使用して、クライアントIPアドレスがブロックリストに登録されているかどうかを検証する 1054

rewrite and responder CRDを使用して、クライアントが要求した有効なパスでURLを更新する 1056

Ingress NetScaler でセッションアフィニティまたはパーシステンスを設定する 1058

Cookie に基づくパーシステンスを設定する 1058

送信元IPアドレス永続性 1059

SSLセッションID永続性 1060

宛先IPアドレスベースの永続性 1060

送信元および宛先IPアドレスベースの永続性 1060

IPアドレスの許可リスト登録または拒否リスト登録 1061

IPアドレスの許可リスト登録 1061

IPアドレスを許可リストに登録し、許可リストにないクライアントからのリクエストに403応答を送信する 1063

CIDRを許可リストに登録する 1064

IPアドレスをブロックリストに登録する 1065

CIDRをブロックリストに登録する 1066

CIDRを許可リストに登録し、IPアドレスをブロックリストに登録する 1067

CIDRをブロックリストに登録し、IPアドレスを許可リストに登録する 1068

ExternalDNSとの相互運用性 1069

NetScaler Ingress ControllerのためにVaultサーバーに保存されたNetScaler資格情報を使用する 1070

前提条件 1070

NetScaler Ingress ControllerのためにVaultサーバーからNetScaler資格情報を使用する 1071

Kubernetes認証用のサービスアカウントを作成する 1071

© 1997–2026 Citrix Systems, Inc. All rights reserved.

キーボールトシークレットを作成し、VaultサーバーでKubernetes認証を設定する 1072
NetScaler Ingress ControllerでVaultエージェントの自動認証を活用する 1074
NetScalerの資格情報を保存するためにKubernetesシークレットを使用する方法 1081
Kubernetesシークレットを作成する 1082
NetScaler Ingress ControllerのPod仕様でシークレットデータを環境変数として使用する 1083
NetScaler Ingress Controller に資格情報を渡すためにシークレットボリュームマウントを使用する 1085
Hashicorp Vault サーバーに保存されている NetScaler 資格情報を使用する 1087
UDPベースのアプリケーションへのイングレストラフィックをロードバランスする方法 1088
UDPベースのアプリケーションのパフォーマンスを監視および改善する 1090
NetScaler CPXサービスで非標準HTTPポートを公開する方法 1091
Helmチャートの展開の場合 1092
OpenShift Operatorを使用する展開の場合 1092
デュアルティア展開のセットアップ方法 1093
セットアッププロセス 1094
ワンステップデプロイメントマニフェストファイルを使用したデュアルティアデプロイメントのセットアップ 1098
NetScaler CPXのカスタムメトリックによる水平Podオートスケーラー 1100
NetScaler CPX HPAソリューション 1101
NetScaler CPX HPA ソリューションのデプロイ 1102
ダイレクトサーバーリターンを展開する 1104
NetScalerを使用したKubernetes向けDSRネットワークトポロジ 1105
NetScalerを使用したクラウドネイティブアプリケーション向けDSRの展開 1105
NetScaler CPXをTier-2 ADCとして展開する 1105
Kubernetesクラスターへのサンプルアプリケーションの展開 1106
Tier-1とTier-2 ADC間のネットワーク接続を確立する 1107
Tier-1 ADC 用のNetScaler Ingress Controllerを展開し、NetScaler CPXをサービスとして公開する 1107

© 1997–2026 Citrix Systems, Inc. All rights reserved.

DSR展開をテストする 1108
トラブルシューティング 1108
アドミッションコントローラーWebフックのサポート 1109
アドミッションコントローラーの適用方法 1110
アドミッションコントローラーWebhookの記述とデプロイ 1110
前提条件 1110
ミューティングアドミッションWebhook構成 1111
Gatekeeperを使用したバリデーションアドミッションWebhook構成 1111
サンプルポリシーの展開 1112
その他のサンプルユースケース 1114
NetScaler Ingress Controller を使用して gRPC サポートを有効にする 1114
gRPC サポートを有効にする 1114
レート制限CRDを検証する 1121
gRPCでリライトおよびレスポンダーCRDを検証する 1122
複数のKubernetesクラスターに対するポリシーベースルーティング 1125
NetScaler Ingress Controllerを使用したPBRの構成 1125
NetScaler Ingress Controller展開後のNetScalerでのPBR構成の検証 1126
NetScalerノードコントローラーを使用してPBRを構成する 1128
NetScalerノードコントローラーのデプロイ後にNetScalerでのPBR構成を検証する 1129
MongoDB向け単一階層NetScaler® Ingressソリューション 1131
前提条件 1132
MongoDB向けNetScaler Ingressソリューションのデプロイ 1133
NetScaler Ingressソリューションのデプロイを検証する 1139
MongoDBトラフィックの可観測性を検証する 1139
テレメトリデータをPrometheusにエクスポートする 1140
KubernetesベースのアプリケーションNetScaler VPXとAzure Pipelinesを使用したカナリアおよびブルー/グリーンデプロイメント 1142

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Kubernetesベースのアプリケーション向けNetScaler VPXとAzure Pipelinesを使用したカナリアデプロイメント 1143
カナリアデプロイメントの利点 1143
NetScalerを使用したカナリアデプロイメント 1143
NetScaler VPXとAzureパイプラインを使用したカナリアデプロイ 1144
Azureパイプラインでカナリアにサンプルアプリケーションをデプロイする 1145
Kubernetesベースのアプリケーション向けNetScaler VPXとAzureパイプラインを使用したブルー/グリーンデプロイ 1146
外部サービス向けのトラフィック管理 1147
外部サービスに到達するようにNetScalerを構成する 1147
KubernetesクラスターからNetScalerへの到達可能性のためのサービスを構成する 1147
NetScaler Ingress Controllerを使用してNetScalerをドメイン名リゾルバーとして構成する 1148
NetScaler CPXを使用したトラフィック管理 1148
名前空間をまたがるExternalNameサービスのサポート 1150
ExternalNameサービスの例 1150
アップグレードとトラブルシューティング 1153
アップグレード中のNetScaler CPX構成の永続性 1153
利点 1153
構成の永続化を有効にする 1154
Helmのインストール 1154
NetScaler Ingress Controller のトラブルシューティング 1154
イベントベースのデバッグ 1154
ログベースのデバッグ 1156
NetScaler Kubernetes kubectl プラグイン 1162
curl を使用したインストール 1163
Krewを使用したインストール 1164
Kubectlプラグインのサブコマンド使用例 1164

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サブコマンドの使用例 1165
NSIC診断ツール 1170
NetScalerノードコントローラーの問題のトラブルシューティング 1171
ルーターポッドが起動しない 1171
サービスステータスがDOWN 1171
サービスステータスは稼働中だが、NetScalerからのpingが機能しない 1172
ポッドエンドポイントまたはVIPへのcURLが機能しない 1173
カスタマーサポート 1173
NetScaler Ingress Controller をアップグレードする 1174
前提条件 1175
YAML ファイルを使用してデプロイされた NSIC をアップグレードする 1175
Helmチャートを使用して展開されたNSICをアップグレードする 1177
リファレンス 1180
エンティティ名の変更 1180
コメントの形式 1180
CRDのエンティティ名の変更 1182
アップグレードシナリオ 1188
NetScaler Ingress Controllerをバージョン4.0.xからバージョン3.x.xにダウングレードする 1188
ライセンス 1190
fix 1193

© 1997–2026 Citrix Systems, Inc. All rights reserved.

概要

KubernetesにおけるIngress Controllerとは

Kubernetesクラスター内でアプリケーションを実行している場合、外部ユーザーがKubernetesクラスター外からアプリケーションにアクセスする方法を提供する必要があります。Kubernetesは、Ingressと呼ばれるオブジェクトを提供しており、これによりKubernetesクラスター内のサービスにアクセスするためのルールを定義できます。これは、安定したIPアドレスを使用して、クラスター内で実行されている複数のサービスに外部からアクセスするための最も効果的な方法を提供します。

Ingress Controllerは、Ingressで定義されたルールを解釈する、クラスター内にデプロイされたアプリケーションです。Ingress Controllerは、Ingressルールを、クラスターと統合されたロードバランシングアプリケーションの構成指示に変換します。ロードバランサーは、Kubernetesクラスター内で実行されるソフトウェアアプリケーションでも、クラスター外で実行されるハードウェアアプライアンスでも構いません。

ネットスケーラー イングレス コントローラーとは

NetScalerは、NetScaler (NetScaler CPX、VPX、またはMPX) を使用して、Kubernetesクラスターへのトラフィックを管理およびルーティングするためのKubernetes Ingress Controllerの実装を提供します。

NetScaler Ingress Controllerを使用すると、Ingressルールに従ってNetScaler CPX、VPX、またはMPXを構成し、NetScalerをKubernetes環境と統合できます。

NetScaler Ingress Controllerを選ぶ理由

このトピックでは、NetScaler Ingress Controllerを使用してNetScalerをKubernetesクラスターと統合する主な利点について説明します。

TCPおよびUDPトラフィックのサポート

標準的なKubernetes Ingressソリューションは、レイヤー7（HTTPまたはHTTPSトラフィック）でのみロードバランシングを提供します。場合によっては、TCPまたはUDPアプリケーションに依存する多くのレガシーアプリケーションを公開し、それらのアプリケーションをロードバランシングする方法が必要になります。NetScaler Ingress Controllerを使用したNetScaler Ingress

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Controllerソリューションは、標準のHTTPまたはHTTPS Ingressとは別に、TCP、TCP-SSL、およびUDPトラフィックのサポートを提供します。また、複数のクラウドまたはオンプレミスデータセンター間でシームレスに動作します。

高度なトラフィック管理ポリシー

NetScalerは、レイヤー7でトラフィックを効率的にロードバランシングするための、リライトポリシーやレスポンダーポリシーのようなエンタープライズグレードのトラフィック管理ポリシーを提供します。しかし、Kubernetes Ingressには、このようなエンタープライズグレードのトラフィック管理ポリシーがありません。CitrixのKubernetes Ingressソリューションを使用すると、NetScalerが提供するCRDを使用して、Kubernetes環境のアプリケーショントラフィックにリライトポリシーとレスポンダーポリシーを適用できます。

柔軟なデプロイメントトポロジー

NetScalerは、NetScalerとKubernetes環境をどのように管理したいかに応じて、Single-TierやDual-Tierのような柔軟で強力なトポロジーを提供します。デプロイメントトポロジーの詳細については、 デプロイメントトポロジーページを参照してください。

East-Westトラフィックに対するレイヤー7ロードバランシングのサポート

Kubernetesクラスター内のマイクロサービス間のトラフィック（East-Westトラフィック）の場合、Kubernetesはネイティブで限定的なレイヤー4ロードバランシングのみを提供します。Ingressコントローラーとともに NetScaler CPXを使用することで、East-Westトラフィックに対する高度なレイヤー7ロードバランシングを実現できます。

ベアメタルクラスター上のLoadBalancerタイプのサービス

パブリッククラウドにデプロイするのではなく、Kubernetesクラスターをベアメタルまたはオンプレミスにデプロイしたい状況がいくつかあるかもしれません。ベアメタルKubernetesクラスターでアプリケーションを実行している場合、Ingressを使用するよりも、`LoadBalancer`タイプのサービスを使用してTCPまたはUDPトラフィックをルーティングする方がはるかに簡単です。HTTPトラフィックの場合でも、Ingressよりも便利な場合があります。しかし、ベアメタルKubernetesクラスターには、ネイティブで利用可能なロードバランサーの実装がありません。NetScalerは、IngressコントローラーとNetScalerを使用して、そのようなサービスをロードバランシングする方法を提供します。詳細については、Expose services of type LoadBalancerを参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Ingress Controllerのデプロイ

NetScaler Ingress Controllerは、以下のデプロイモードでデプロイできます。

1. スタンドアロンポッドとして: このモードは、Kubernetesクラスターの外部にあるNetScaler MPXやVPXなどのADCを管理する場合に使用されます。
2. 同じポッド内のNetScaler CPXとともに、ポッド内のサイドカーとして: コントローラーは、同じポッド内にあるNetScaler CPXのみを担当します。

NetScalerが提供するIngressコントローラーは、Helmチャートを使用してデプロイできます。詳細については、Deploy NetScaler Ingress Controller using Helm chartsを参照してください。

Kubernetesベースのマイクロサービス向けNetScaler®クラウドネイティブソリューション

企業がより迅速に革新し、顧客との距離を縮めるために変革するにつれて、社内プロセスを再構築し、組織内の境界を打ち破っています。彼らはサイロを排除し、適切なスキルセットを同じチームに集めています。目標の1つは、速度、俊敏性、効率性をもってソフトウェアアプリケーションを作成し、提供することです。この点に関して、マイクロサービスに基づいた最新のアプリケーションアーキテクチャが、ますます多くの企業に採用されています。 マイクロサービスアーキテクチャを使用すると、疎結合されたサービスのセットとしてアプリケーションを作成でき、これらは個別にデプロイ、更新、スケーリングできます。

クラウドネイティブは、以下の主要な属性を持つアプリケーションを構築およびデプロイするために、マイクロサービスアーキテクチャに依存するアプローチです。

- アプリケーションを疎結合されたマイクロサービスまたはコンテナとしてデプロイする
- 非常に高度な自動化を伴う
- アジャイルなDevOpsプロセスと継続的デリバリーワークフローを実装する
- 相互作用とコラボレーションのためのAPIを中心に据える

クラウドネイティブへの移行において、Kubernetesはどのように役立ちますか?

望ましいレベルの俊敏性と安定性を提供するために、クラウドネイティブアプリケーションには、高レベルのインフラストラクチャ自動化、セキュリティ、ネットワーキング、および監視が必要です。大規模なコンテナを効率的に管理できるコンテナオーケストレーションシステムが必要です。Kubernetesは、コンテナのデプロイとオーケストレーションのための最も人気のあるプラットフォームとして登場しました。Kubernetesは、コンテナの実行、デプロイ、管理という複雑な

© 1997–2026 Citrix Systems, Inc. All rights reserved.

タスクを開発者やオペレーターから抽象化し、ノードのクラスター間でコンテナを自動的にスケジュールします。Kubernetesとクラウドネイティブコンピューティング財団（CNCF）エコシステムは、クラウドネイティブソリューションのためのプラットフォームを構築するのに役立ちます。

Kubernetesを使用する主な利点のいくつか：

- オンプレミス、ハイブリッド、またはパブリッククラウドインフラストラクチャのいずれにおいても、アプリケーションのデプロイを簡素化する
- アプリケーションの開発とデプロイを加速する
- アプリケーションの俊敏性、柔軟性、スケーラビリティを向上させる

NetScalerクラウドネイティブソリューションとは何ですか?

Kubernetesを本番環境で使用するメリットを最大化するには、Kubernetesをいくつかのツール、ベンダー提供のコンポーネント、およびオープンソースコンポーネントと統合する必要があります。クラウドネイティブアプリケーションのプロダクショングレードの信頼性とセキュリティを確保することは、多くの組織が直面する課題です。NetScalerは、Kubernetes本番環境における課題に対処するためのNetScalerクラウドネイティブソリューションを提供します。

NetScalerクラウドネイティブソリューションは、NetScalerの高度なトラフィック管理、可観測性、および包括的なセキュリティ機能を活用し、エンタープライズグレードの信頼性とセキュリティを確保します。Kubernetes環境におけるアプリケーショントラフィックへの完全な可視性を提供し、即座にフィードバックを返し、アプリケーションのパフォーマンスに関する有意義な洞察を得るのに役立ちます。

以下の表は、Ingressソリューションを実装する際のさまざまな関係者の主要な要件をリストしています。

関係者	職務	ニーズ
プラットフォーム管理者	Kubernetesクラスターの可用性を確保する	複数のクラスターにデプロイされたアプリケーションの管理、運用、およびプラットフォームのライフサイクル管理をよりシンプルにする方法
DevOps	アプリケーションの本番環境へのデプロイを加速する	CI/CDパイプラインとの統合、Canaryやブルーグリーンなどのデプロイ技術のサポートによる迅速なデプロイ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

関係者	職務	ニーズ
開発者	マイクロサービスの開発とテスト	Kubernetesクラスターへのトラフィック導入方法、トレースとデバッグ、アプリケーションのレート制限、およびアプリケーションの認証
SRE	サービスレベルアグリーメントを満たすためのアプリケーションの可用性を確保する	アプリケーションとインフラストラクチャのための高度なテレメトリ
SecOPs	セキュリティコンプライアンスを確保する	Kubernetesクラスター内のマイクロサービス間の安全な通信のためのIngressトラフィックの保護、API保護、サービスメッシュ

次の図は、NetScalerクラウドネイティブソリューションと、それがステークホルダーのクラウドネイティブジャーニーで直面するさまざまな課題にどのように対処するかを説明しています。

NetScalerクラウドネイティブソリューション](/en-us/netscaler-k8s-ingress-controller/media/citrix-cloud-native.png)

NetScalerクラウドネイティブソリューションは、以下の主要な利点を提供します。

- 開発者、SRE、DevOps、およびネットワーク管理者またはクラスター管理者のニーズに応える高度なKubernetes Ingressソリューションを提供します。
- TCPまたはUDPトラフィックに基づくレガシーアプリケーションをKubernetes環境に移行する際に、それらを書き換える必要をなくします。
- Kubernetes APIとして公開されるNetScalerポリシーでアプリケーションを保護します。
- North-SouthトラフィックおよびEast-Westトラフィック向けに高性能なマイクロサービスをデプロイするのに役立ちます。
- NetScaler Consoleサービスグラフを使用して、すべてのマイクロサービスを統合的に表示します。
- TCP、UDP、HTTP、HTTPS、SSLを含むさまざまな種類のトラフィックにおけるマイクロサービスのトラブルシューティングを高速化します。
- APIを保護します。
- Canary デプロイメントのための CI/CD パイプラインを自動化します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- CNCF オープンソースツールとのすぐに使える統合を提供します。

NetScaler が提供するさまざまなクラウドネイティブソリューションの詳細については、以下のリンクを参照してください。

- Kubernetes Ingress ソリューション
- サービスメッシュ
- 可観測性ソリューション

NetScaler クラウドネイティブソリューションのコンポーネント

次の表は、NetScaler クラウドネイティブソリューションの主要コンポーネントについて説明しています。

コンポーネント	説明
NetScaler イングレスコントローラー	このコンテナは、NetScaler ADC (NetScaler CPX、BLX、VPX、または MPX) を使用して Kubernetes クラスターへのトラフィックを管理およびルーティングするための Kubernetes Ingress Controller の実装です。NetScaler Ingress Controller を使用すると、Ingress ルールに従って NetScaler CPX、BLX、VPX、または MPX を構成し、NetScaler ADC を Kubernetes 環境と統合できます。
NetScaler オブザーバビリティエクスポーター	NetScaler Observability Exporter は、NetScaler ADC からメトリクスとトランザクションを収集し、サポートされているエンドポイントに適した形式 (JSON、AVRO など) に変換するコンテナです。NetScaler Observability Exporter によって収集されたデータを目的のエンドポイントにエクスポートできます。エンドポイントにエクスポートされたデータを分析することで、NetScaler ADC によってプロキシされたアプリケーションのマイクロサービスレベルで貴重な洞察を得ることができます。
NetScaler xDS アダプター	NetScaler xDSアダプターは、xDS API (Istio、Consulなど) に基づくサービスメッシュコントロールプレーンの実装とNetScalerを統合するためのコンテナです。これは、サービスメッ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

コンポーネント	説明
	シュコントロールプレーンと通信し、コントロールプレーンAPIサーバーへのgRPCクライアントとして機能することで更新をリッスンします。コントロールプレーンからの更新に基づいて、NetScaler xDS-Adaptorは同等のNetScaler構成を生成します。
NetScaler CPX(https://docs.citrix.com/ja-jp/citrix-adc-cpx/current-release.html)	NetScaler CPXは、Dockerホストにプロビジョニングできるコンテナベースのアプリケーションデリバリーコントローラーです。NetScaler CPXを使用すると、顧客はDockerエンジンの機能を活用し、コンテナベースのアプリケーション向けにNetScalerの負荷分散およびトラフィック管理機能を使用できます。1つまたは複数のNetScaler CPXインスタンスを、Dockerホスト上にスタンドアロンインスタンスとして展開できます。

Kubernetes向けNetScaler® イングレスコントローラー

Kubernetesクラスター内でアプリケーションを実行している場合、外部ユーザーがKubernetesクラスター外からアプリケーションにアクセスする方法を提供する必要があります。KubernetesはIngressと呼ばれるオブジェクトを提供しており、これによりKubernetesクラスター内のサービスにアクセスするためのルールを定義できます。これは、安定したIPアドレスを使用して、クラスター内で実行されている複数のサービスに外部からアクセスするための最も効果的な方法を提供します。

Ingressコントローラーは、クラスター内にデプロイされるアプリケーションであり、Ingressで定義されたルールを解釈します。Ingressコントローラーは、Ingressルールを、クラスターと統合されたロードバランシングアプリケーションの構成指示に変換します。ロードバランサーは、Kubernetesクラスター内で実行されるソフトウェアアプリケーションである場合もあれば、クラスター外で実行されるハードウェアアプライアンスである場合もあります。

NetScalerは、NetScaler ADC (NetScaler CPX、BLX、VPX、またはMPX) を使用して、Kubernetesクラスターへのトラフィックを管理およびルーティングするためのKubernetes Ingress Controllerの実装を提供します。

NetScaler Ingress Controllerを使用すると、Ingressルールに従ってNetScaler CPX、BLX、VPX、またはMPXを構成し、NetScaler ADCをKubernetes環境と統合できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サポートされているプラットフォームとデプロイ

このトピックでは、サポートされているKubernetesプラットフォームと、クラウドネイティブコンポーネントのNetScalerとの互換性について詳しく説明します。

Kubernetesプラットフォーム

NetScaler Ingress Controller (NSIC) は、以下のプラットフォームでサポートされています。

- ベアメタルまたはAWS、GCP、Azureなどのパブリッククラウドでセルフホストされている Kubernetes v1.21 (以降)。
- グーグル・クバネティス・エンジン (GKE)
- エラスティック・クバネティス・サービス (EKS)
- アジュール・クバネティス・サービス (AKS)
- Red Hat OpenShift バージョン 3.11 以降
- ピボタル・コンテナ・サービス (PKS)
- ディアマンティ・エンタープライズ・クバネティス・プラットフォーム
- ミランティス・クバネティス・エンジン
- ヴイエムウェア タンズ
- ランチャー

NetScalerとの互換性

以下の表は、任意のNetScaler Ingress Controllerリリースと互換性のあるNetScalerのバージョンを示しています。

NSIC/GSLBC	NETSCALERのバージョン
4.2.26	VPX/MPX/CPX 14.1-73.x 以降
4.1.17	VPX/MPX/CPX 14.1-66.59
4.0.16	VPX/MPX/CPX 14.1-60.57
3.6.0	VPX/MPX/CPX 14.1-66.59
3.5.0	VPX/MPX/CPX 14.1-66.59

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NSIC/GSLBC	NETSCALERのバージョン
3.4.4	VPX/MPX/CPX 14.1-60.57
3.3.2	VPX/MPX/CPX 14.1-47.48
3.2.22	VPX/MPX/CPX 14.1-47.48
3.1.34	VPX/MPX/CPX 14.1-43.50
3.0.5	VPX/MPX/CPX 14.1-38.53
2.3.15	VPX/MPX/CPX 14.1-38.53
2.2.10	VPX/MPX 14.1-25.56CPX 14.1-25.111
2.1.4	VPX/MPX 14.1-25.56CPX 14.1-25.111
2.0.6	VPX/MPX 14.1-25.56CPX 14.1-25.111
1.43.7	VPX/MPX 14.1-25.56 以降 CPX 14.1-25.109 以降
1.42.12	VPX/MPX 14.1-17.38 以降 CPX 14.17.101 以降
1.41.5	VPX/MPX 14.1-17.38 以降 CPX 14.17.101 以降
1.40.12	14.1-12.35 以降
1.39.6	14.1-12.35 以降
1.38.27	13.1-49.15 以降
1.37.5	13.1-49.15 以降
1.36.5	13.1-49.15 以降
1.35.6	13.1-49.15 以降

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScalerクラウドネイティブコンポーネントとのNetScaler Operatorの互換性

この表は、任意のNetScaler Operatorリリースを使用して展開できるNetScalerクラウドネイティブコンポーネントのリリースを一覧表示します。

OPENSHIFT バージョン	NETSCALER OPERATOR バージョン	NSIC バージョン	GSLB CONTROLLER バージョン	NSOE バージョン	CLOUD CONTROLLER バージョン	NETSCALER IPAM コントローラー	NETSCALER GATEWAY コントローラー
4.11 以降	3.6.0	4.1.17	4.1.17	1.10.1	1.2.0	2.3.1	2.1.0
4.11 以降	3.5.0	3.4.4	3.4.4	1.10.1	1.2.0	2.1.2	1.4.0
4.11 以降	3.4.1	3.3.4	3.3.4	1.10.1	1.2.0	2.1.2	1.4.0
4.11 以降	3.3.0	3.3.2	3.3.2	1.10.1	1.2.0	2.1.2	1.3.0
4.11 以降	3.2.0	2.3.15	2.3.15	1.10.1	1.2.0	2.1.2	--
4.11 以降	3.1.2	2.2.10	2.2.10	1.10.1	1.2.0	--	--

クラウドネイティブコンポーネントとNetScalerの互換性

以下のクラウドネイティブコンポーネントは、NetScalerバージョン13.1以降と互換性があります。

- ネットスケーラー Kubernetes ゲートウェイコントローラー
- ネットスケーラー ノードコントローラー
- ネットスケーラー オブザーバビリティエクスポーター
- ネットスケーラー コンソールサービス
- ネットスケーラー コンソール オンプレミス
- ネットスケーラー xDSアダプター

© 1997–2026 Citrix Systems, Inc. All rights reserved.

リリースノート

NetScaler Ingress Controller (NSIC)/NetScaler GSLB コントローラー

a. NSIC/GSLB リリース

- 4.2.26
- 4.1.17
- 4.0.16
- 3.6.0
- 3.5.0
- 3.4.4
- 3.3.2
- 3.2.22
- 3.1.34
- 3.0.5
- 2.3.15
- 2.2.10
- 2.1.4
- 2.0.6
- 1.43.7

b. Helmチャートのリリース

- 4.2.26
- 4.1.17
- 4.0.16
- 3.6.0
- 3.5.0
- 3.4.4
- 3.3.2
- 3.2.22
- 3.1.34

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- 3.0.6
- 3.0.5
- 2.3.16
- 2.3.15
- 2.2.11
- 2.2.10
- 2.1.4
- 2.0.6

c. オペレーターのリリース

- 3.6.0
- 3.5.0
- 3.4.1
- 3.3.0
- 3.2.0
- 3.1.2

IPAMコントローラー

- 2.4.0
- 2.3.1
- 2.2.1
- 2.1.2
- 2.0.2
- 2.0.1
- 1.2.0

参照

- Helmチャートを使用したNSICのアップグレードに関する情報については、NetScaler Ingress Controllerのアップグレードを参照してください。
- NetScaler Operatorのアップグレードに関する情報については、NetScaler Operatorのアップグレードを参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

はじめに

NetScaler Ingress Controllerを初めて使用する場合、このガイドはKubernetes用NetScaler Ingress Controllerを迅速に評価するのに役立ちます。上級ユーザーの場合は、次は何ですかを参照してください。

始める前に

Minikubeクラスターをインストールして設定済みであることを確認してください。

NetScaler Ingress Controllerの使用を開始する

この手順では、以下のステップを実行します。

- NetScaler CPX（NetScalerのコンテナ化バージョン）をイングレスコントローラーとともにデプロイする
- サンプルアプリケーションである`Guestbook`をデプロイする
- `Guestbook`アプリケーションにトラフィックをルーティングするためのイングレスルールをデプロイする
- アプリケーションにトラフィックを送信し、検証する

NetScaler Ingress Controllerを使用してNetScaler CPXをデプロイする

NetScaler Ingress Controllerを使用してNetScaler CPXをデプロイするには、以下を実行します。

1. MinikubeクラスターにNetScaler CPXをイングレスプロキシとしてデプロイします。

```
kubectl create -f https://raw.githubusercontent.com/citrix/cloud-native-getting-started/master/beginners-guide/manifest/cpx.yaml
```

2. 次のコマンドを使用してインストールを検証します。

```
kubectl get pods -l app=cpx-ingress
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サンプルアプリケーションをデプロイする

このステップでは、Redisを使用する多層PHPベースのWebアプリケーションである`Guestbook`をデプロイします。

1. Minikubeに`Guestbook`アプリケーションをデプロイします。

```
kubectl create -f https://raw.githubusercontent.com/citrix/cloud-native-getting-started/master/beginners-guide/manifest/guestbook-app.yaml
```

2. 以下を使用してインストールを確認します。

```
kubectl get pods -l 'app in (guestbook, redis)'
```

サンプルアプリケーションのIngressをデプロイする

サンプルアプリケーションのIngressルールをデプロイし、機能を検証するには、以下の手順を実行します。

1. `Guestbook`アプリケーション(http://www.guestbook.com)にトラフィックを送信するIngressルールをデプロイします。

```
kubectl create -f https://raw.githubusercontent.com/citrix/cloud-native-getting-started/master/beginners-guide/manifest/guestbook-ingress.yaml
```

2. 次のコマンドを使用してIngressのデプロイを確認します。

```
kubectl get ingress
```

3. 次のコマンドを使用してサービスに関する情報を表示します。

```
kubectl get svc cpx-service



# kubectl get service cpx-service
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
NAME           TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
               AGE



cpx-service    NodePort    10.106.123.144   &lt;none>        80:30592/
TCP,443:31338/TCP   43m
```

この例からNodePort情報を取得できます。30592と31338はNodePortです。

4. `Guestbook`マイクロサービスアプリケーションにトラフィックを送信し、このURLへのトラフィックが`Guestbook`ページを取得することを確認します。

```
curl -s -H "Host: www.guestbook.com" http://&lt;MiniKube-IP-
address>:&lt;NodePort> | grep Guestbook
```

期待される出力:

```
&lt;title>Guestbook&lt;/title>

  <h2>Guestbook</h2>
```

注:

`minikube ip`コマンドを使用してMinikubeのIPアドレスを取得できます。また、`minikube service cpx-service --url`コマンドを使用して、cURLコマンドで使用されるURLを直接取得することもできます。

次のステップ

この入門セクションは、初心者がNetScaler Ingress Controllerを迅速に評価するのに役立ち、インストールは基本的な機能のみをカバーしています。 NetScaler Ingress Controllerのデプロイに関する包括的な情報については、以下のトピックを参照し、それに応じてインストールをカスタマイズできます。

- 「展開トポロジー」(/ja-jp/netscaler-k8s-ingress-controller/deployment-topologies.html): NetScaler Ingress Controllerでサポートされているさまざまなトポロジーに関する情報を提供します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- 「サポートされているプラットフォーム」(/ja-jp/netscaler-k8s-ingress-controller/support-matrix.html): ベアメタルおよびクラウドプラットフォームを含む、サポートされているさまざまなプラットフォームに関する情報を提供します。
- 「NetScaler Ingress Controllerの展開」(/ja-jp/netscaler-k8s-ingress-controller/deploy/unified-ingress-deployment.html): NetScaler CPX、VPX、MPXなど、さまざまなNetScalerのフレーバーにNetScaler Ingress Controllerを展開する方法に関する情報を提供します。

KubernetesクラスターにNetScaler® Ingress Controllerを展開する

このセクションでは、NetScaler (MPX/SDX/VPX/BLX) およびKubernetes環境にアクセスできるネットワーク管理者が、KubernetesにNetScaler Ingress Controllerを展開するのに役立ちます。

前提条件

- Kubernetesクラスターをインストールしてセットアップしていること。
- Helmバージョン3.x以降をインストールしていること。Helmのインストールについては、Helmチャートのインストールを参照してください。
- NetScalerにシステムユーザーアカウントがあること。NetScaler Ingress Controllerはこのアカウントを使用してNetScalerに構成をプッシュします。NetScalerでNetScaler Ingress Controllerのシステムユーザーアカウントを作成するを参照してください。
- NetScalerユーザーアカウントの資格情報を保存するためのKubernetesシークレットがあること。NetScaler資格情報を保存するためにKubernetesシークレットを使用する方法を参照してください。

KubernetesクラスターにNetScaler Ingress Controllerを展開する方法

1. 次のコマンドを使用して、NetScaler Helmチャートリポジトリをローカルレジストリに追加します。

   ```
   helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/
   ```

   NetScaler Helmチャートリポジトリがすでにローカルレジストリに追加されている場合は、次のコマンドを使用してリポジトリを更新します。

   ```
   helm repo update netscaler
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

2. 次のコマンドを使用してNetScaler Ingress Controllerをインストールします。

   注:

   インストール中に構成できるすべてのパラメーターについては、構成を参照してください。

   ```
   helm install netscaler-ingress-controller netscaler/netscaler-ingress-controller --set license.accept=yes --set adcCredentialSecret=<<K8s secret for login NS> --set nsIP=<management IP>
   ```

3. インストールを確認します。

   注:

   カスタムHelmリリース名が指定されている場合、ラベルはapp=`<releasename>-netscaler-ingress-controller`です。

   ```
   kubectl get pods -l app=netscaler-ingress-controller
   ```

   ☐

4. (オプション) NetScaler Ingress Controllerポッドの概要を表示します。

   ```
   kubectl describe pod <pod name>
   ```

   例:

   ```
   kubectl describe pod netscaler-ingress-controller-8474d866d9b-nqp6p
   ```

5. (オプション) NetScaler Ingress Controllerのログを確認します。

   ```
   kubectl logs -f <pod name>
   ```

   例:

   ```
   kubectl logs -f netscaler-ingress-controller-8474d866d9b-nqp6p
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

6. (オプション) インストールを削除します。

```
helm uninstall netscaler-ingress-controller
```

次に

NetScaler Ingress Controllerのデプロイに関する包括的な情報と、それに応じたインストールのカスタマイズについては、以下のトピックを参照してください。

- デプロイメントトポロジ
- 統合インプレス

KubernetesクラスターにNetScaler® Ingress Controllerを使用してCPXを展開する

このセクションでは、Kubernetes環境にアクセスできるプラットフォーム管理者が、KubernetesにNetScaler Ingress Controllerを使用してCPXを展開するのに役立ちます。

前提条件

- Kubernetesクラスターをインストールし、セットアップしていること。
- Helmバージョン3.x以降をインストール済みであること。Helmのインストールについては、こちらを参照してください。

KubernetesクラスターにNetScaler Ingress Controllerを展開する方法

1. 次のコマンドを使用して、NetScaler Helmチャートリポジトリをローカルレジストリに追加します。

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/
```

NetScaler Helmチャートリポジトリがすでにローカルレジストリに追加されている場合は、次のコマンドを使用してリポジトリを更新します。

```
helm repo update netscaler
```

2. 次のコマンドを使用して、NSICでNetScaler CPXをインストールします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
helm install netscaler-cpx-with-ingress-controller netscaler/netscaler-cpx-with-ingress-controller --set license.accept=yes,serviceType.nodePort.enabled=True
```

3. インストールを確認します。

```
kubectl get pods -l app=netscaler-cpx-with-ingress-controller
```

☐

`READY`列の2/2で強調表示されているように、同じポッドで2つのコンテナが実行されています。1つはNetScaler CPXプロキシ用で、もう1つはNetScaler Ingress Controller用です。

4. 両方のコンテナの詳細を表示します。

```
kubectl describe pod $(kubectl get pods -l app=netscaler-cpx-with-ingress-controller | awk '{print $1}' | grep netscaler-cpx-with-ingress-controller)
```

☐

次のステップ

NetScaler Ingress Controller のデプロイに関する包括的な情報と、それに応じたインストールのカスタマイズについては、以下のトピックを参照してください。

- デプロイメントトポロジ
- デュアルティアイングレス

© 1997–2026 Citrix Systems, Inc. All rights reserved.

デプロイメントトポロジー

NetScalerは、組織の境界を補完する強力で柔軟なトポロジーで組み合わせることができます。デュアルティアデプロイメントでは、第1層に大容量のハードウェアまたは仮想化されたNetScaler（NetScaler MPXおよびVPX）を採用し、セキュリティ機能をオフロードし、比較的静的な組織ポリシーを実装しながら、ネットワークオペレーターとKubernetesオペレーターの間で制御を分割します。

デュアルティアデプロイメントでは、第2層はKubernetesクラスター内（NetScaler CPXを使用）にあり、サービスオーナーの管理下にあります。この設定は、ネットワークオペレーターに安定性を提供しつつ、Kubernetesユーザーが高頻度の変更を実装できるようにします。シングルティアトポロジーは、高頻度の変更を処理する必要がある組織に適しています。

シングルティアトポロジー

シングルティアトポロジーでは、NetScaler MPXまたはVPXデバイスがクライアントからクラスター内のマイクロサービスへの（南北）トラフィックをプロキシします。NetScaler Ingress Controllerは、Kubernetesクラスター内にスタンドアロンポッドとしてデプロイされます。このコントローラーは、マイクロサービスまたはIngressリソースへの変更に基づいて、NetScaler（MPXまたはVPX）の設定を自動化します。

シングルティア(/en-us/netscaler-k8s-ingress-controller/media/singletopology.png)

デュアルティアトポロジー

デュアルティアトポロジーでは、Tier-1のNetScaler MPXまたはVPXデバイスが、クライアントからTier-2のNetScaler CPXへの（南北）トラフィックをプロキシします。その後、Tier-2のNetScaler CPXがKubernetesクラスター内のマイクロサービスにトラフィックをルーティングします。スタンドアロンポッドとしてデプロイされたNetScaler Ingress ControllerがTier-1デバイスを設定します。また、1つ以上のNetScaler CPXポッド内のサイドカーコントローラーが、同じポッド内の関連するNetScaler CPXを設定します。

デュアルティア(/en-us/netscaler-k8s-ingress-controller/media/dualtier.png)

© 1997–2026 Citrix Systems, Inc. All rights reserved.

クラウドトポロジー

Amazon Web Services (AWS)、 Google Cloud、 Microsoft Azureなどのパブリッククラウドの Kubernetesクラスターは、 AWS Elastic Load Balancing、 Google Cloud Load Balancing、 Microsoft Azure NLBなどのネイティブロードバランシングサービスを、NetScaler CPXの第2層へのロードバランシングの最初の（比較的静的な）層として使用できます。NetScaler CPXは、サイドカーIngressコントローラーとともにKubernetesクラスター内で動作します。Kubernetesクラスターは、NetScaler CPXをIngressとして使用しながら、セルフホスト型またはクラウドプロバイダーによって管理されているもの（例えば、 AWS EKS、 Google GKE、 Azure AKS）のいずれかです。クラウドベースのKubernetesクラスターがセルフホスト型またはセルフマネージド型の場合、NetScaler VPXをデュアルティアトポロジーの最初の層として使用できます。

Tier-1にNetScaler (VPX) を使用したクラウドデプロイメント: Tier-1にVPXを使用したクラウドデプロイメント(/en-us/netscaler-k8s-ingress-controller/media/cloud-deploy-vpx-tier-1.png)

Tier-1にCloud LBを使用したクラウドデプロイメント: Tier-1にCLBを使用したクラウドデプロイメント(/en-us/netscaler-k8s-ingress-controller/media/cloud-deploy-clb-tier-1.png)

サービスメッシュライト

Ingressソリューションは通常、クライアントからKubernetesクラスター内のマイクロサービスへのトラフィック（南北トラフィック）に対してレイヤー7プロキシ機能を実行します。サービスメッシュライトアーキテクチャは、同じIngressソリューションを使用して、Kubernetesクラスター内のサービス間のトラフィック（東西トラフィック）も管理します。通常、サービスメッシュソリューションは東西トラフィックの管理に使用されますが、より重く、管理が複雑です。サービスメッシュライトソリューションは、サービスメッシュアーキテクチャの簡素化されたバージョンであり、南北および東西トラフィック管理の両方を管理する必要がある場合に理想的です。サービスメッシュでは、アプリケーションの数と同じ数のサイドカープロキシが存在します。しかし、サービスメッシュライトアーキテクチャでは、プロキシは複数の東西接続を管理するスタンドアロンプロキシとしてデプロイされます。したがって、サービスメッシュライトソリューションは、サービスメッシュと比較して軽量です。 なぜなら、必要なプロキシの数が少ないためです。

標準的なKubernetesデプロイメントでは、East-Westトラフィックは各ノードにデプロイされた組み込みのkube-proxyを通過します。Kube-proxyはL4プロキシであるため、L7プロキシの利点なしにTCP/UDPベースのロードバランシングしかできません。

NetScaler (MPX、VPX、またはCPX) は、East-Westトラフィックに対して次のような利点を提供できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- 相互TLSまたはSSLオフロード
- HTTPまたはHTTPSヘッダーパラメーターに基づくコンテンツベースのルーティング、トラフィックの許可またはブロック
- 高度なロードバランシングアルゴリズム（例：最小接続数、最小応答時間など）
- ゴールデンシグナル（エラー、遅延、飽和、トラフィック量）を測定することによるEast-Westトラフィックの可観測性。(/ja-jp/citrix-application-delivery-management-service.html) Service Graphは、マイクロサービスを監視およびデバッグするための可観測性ソリューションです。

詳細については、(/ja-jp/netscaler-k8s-ingress-controller/deploy/service-mesh-lite.html)を参照してください。

(/en-us/netscaler-k8s-ingress-controller/media/dual-tier-topology-with-hairpin-E-W.png)

サービスメッシュライトトポロジが推奨されるシナリオを以下に示します。

- マイクロサービスに対してNorth-SouthおよびEast-Westの両方のトラフィック管理が必要な場合。
- マイクロサービスへのサイドカープロキシとしてではなく、スタンドアロンプロキシとしてデプロイされたプロキシを介したEast-Westトラフィック管理が必要な場合。
- Kubernetesクラスター内のプロキシがNorth-SouthおよびEast-Westの両方のトラフィック管理を実行する必要がある場合。
- サービスメッシュの利点が必要だが、より軽量でシンプルなソリューションを望む場合。

LoadBalancerタイプのサービス

Kubernetesの`LoadBalancer`タイプのサービスを使用すると、イングレスリソースを使用せずにサービスを外部に直接公開できます。これは、独自のネイティブクラウドロードバランサーを起動し、サービスがアクセスされる外部IPアドレスを割り当てるクラウドプロバイダーによってのみ提供されます。これにより、マイクロサービスを簡単にデプロイし、Kubernetesクラスターの外部に公開できます。

ベアメタルKubernetesクラスターでは、デフォルトで、`LoadBalancer`タイプのサービスは、サービスに対して`NodePorts`を公開するだけです。そして、外部ロードバランサーは構成しません。

NetScaler Ingress Controllerは、`LoadBalancer`タイプのサービスをサポートしています。`LoadBalancer`タイプのサービスを作成し、Tier-1のイングレスNetScalerを使用して公開できます。イングレスNetScalerはサービス用のロードバランサーをプロビジョニングし、外部IPアドレ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

スがサービスに割り当てられます。NetScaler Ingress Controllerは、NetScaler IPAMコントローラーを使用してIPアドレスを割り当てます。

詳細については、LoadBalancerタイプのサービスを公開するを参照してください。

NodePortタイプのサービス

デフォルトでは、Kubernetesサービスはクラスター IPアドレスを使用してアクセスできます。クラスターIPアドレスは、Kubernetesクラスター内でアクセスできる内部IPアドレスです。Kubernetesクラスターの外部からサービスにアクセスできるようにするには、`NodePort`タイプのサービスを作成できます。

NetScaler Ingress Controllerは、`NodePort`タイプのサービスをサポートしています。イングレスNetScalerとNetScaler Ingress Controllerを使用すると、`NodePort`タイプのサービスを外部に公開できます。

詳細については、NodePortタイプのサービスを公開するを参照してください。

トポロジーを選択するためのガイドライン

以下の情報は、ニーズに基づいて、シングルティアとデュアルティアのトポロジーの中から適切なデプロイメントを選択するのに役立ちます。

シングルティア (統合イングレス)

シングルティア (統合イングレス) トポロジーが推奨されるシナリオと利点を以下に示します。

- 既存のNetScaler®をKubernetesクラスターの前面にあるイングレスプロキシとして使用できるため、開始と導入が簡単です。
- ネットワークチームがNetScalerとKubernetesデプロイメントの両方を管理する場合。
- マイクロサービスとして実行されているワークロードが少なく、Kubernetesクラスター内のKubernetesプロキシは不要です。
- ノースサウストラフィックのデプロイメントにより適しています。

デュアルティア

デュアルティアイングレストポロジが推奨されるシナリオと利点の一部を以下に示します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- マイクロサービスとして実行されているワークロードが大量にある場合、Kubernetesクラスター内にプロキシが必要になります。
- 外部プロキシ（ネットワークチームが管理）とKubernetesプロキシ（プラットフォームチームが管理）が2つの異なるチームによって管理されている場合。
- Kubernetes外部のプロキシとKubernetes内部のプロキシで機能の分離が必要です。例えば、外部NetScalerでのWAFとSSLオフロード、およびKubernetesプロキシでのポリシー適用とレート制限などです。
- Kubernetesクラスター内のプロキシは、ノースサウストラフィック管理のみを実行します。

Helmチャートを使用したデプロイ

NetScalerクラウドネイティブトポロジをデプロイするには、YAMLとHelmチャートを使用したさまざまなオプションがあります。Helmチャートは、Kubernetes環境でのデプロイに最も簡単な方法の1つです。Helmチャートを使用してデプロイする場合、各パラメータを引数として提供する代わりに、設定可能なパラメータの値を指定するために`values.yaml`ファイルを使用できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler ingress controller を展開する

要件に応じて、さまざまな展開オプションを使用してNetScaler Ingress Controllerを展開できます。詳細については、以下のトピックを参照してください。

- 統合インプレス
- デュアルティアインプレス
- マルチクラスターインプレス
- ランチャーで
- PKS で

統合イングレス

統合イングレス展開は、North-Southトラフィックのロードバランシングを実行するために使用されます。North-Southトラフィックとは、Kubernetesクラスターに出入りするトラフィック、つまりクライアントとフロントエンドのマイクロサービス間を流れるトラフィックのことです。この展開では、NetScaler Ingress ControllerがNetScalerの設定を自動化し、Kubernetes環境内のマイクロサービスへのトラフィックをロードバランスします。NetScaler Ingress Controllerは、Kubernetes APIサーバーを監視し、NetScaler MPXまたはNetScaler VPXを設定するポッドとして実行されます。

KubernetesクラスターにNetScaler Ingress Controllerをスタンドアロンポッドとして展開する

netscaler-ingress-controller Helmチャートを使用して、NetScaler Ingress ControllerをKubernetesクラスターにポッドとして展開します。

インストールには、NetScaler Ingress Controllerポッドの作成、クラスターロールの作成、クラスターロールバインディング、およびサービスアカウントの作成が含まれます。

Helmチャートを使用してNSICを展開する場合、各パラメーターを引数として提供する代わりに、`values.yaml`ファイルを使用して設定可能なパラメーターの値を指定できます。

注:

NSICはRBACロールでも動作します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

前提条件

- NetScaler Ingress ControllerがNetScalerと通信するために使用するNS_IP (NetScaler IP) アドレスを特定します。IPアドレスは、NetScaler展開のタイプに応じて、次のいずれかのアドレスである可能性があります。
  - NSIP (スタンドアロンアプライアンスの場合) - スタンドアロンNetScalerアプライアンスの管理IPアドレス。詳細については、IP Addressing in NetScalerを参照してください。
  - SNIP (高可用性モードのアプライアンスの場合) - サブネットIPアドレス。詳細については、IP Addressing in NetScalerを参照してください。
  - CLIP (クラスターモードのアプライアンスの場合) - クラスターNetScaler展開のクラスター管理IP (CLIP) アドレス。詳細については、IP addressing for a clusterを参照してください。
- Ingressデバイスとして使用されるNetScaler MPXまたはNetScaler VPXのシステムユーザーアカウントのユーザー名とパスワード。NetScaler Ingress ControllerがNetScaler MPXまたはNetScaler VPXを設定できるように、NetScalerには特定の権限を持つシステムユーザーアカウント (非デフォルト) が必要です。NetScalerでシステムユーザーアカウントを作成する手順については、Create a system user account for NetScaler Ingress Controller in NetScalerを参照してください。

  ユーザー名とパスワードを直接渡すか、Kubernetesシークレットを使用できます。Kubernetesシークレットを使用する場合は、次のコマンドを使用してユーザー名とパスワードのシークレットを作成します。

  ```
  kubectl create secret  generic nslogin --from-
  literal=username=&lt;username> --from-literal=password=&lt;password>
  ```

- Kubernetes環境ではバージョン1.21以降。
- OpenShiftプラットフォームではバージョン4.8以降。
- Helmバージョン3.x以降がインストールされていることを確認してください。Helmのインストールについては、Helmチャートのインストールを参照してください。

NetScalerでNetScaler® Ingress Controllerのシステムユーザーアカウントを作成する

NetScaler Ingress Controllerは、NetScalerのシステムユーザーアカウントを使用してNetScalerを構成します。システムユーザーアカウントには、NetScaler Ingress ControllerがNetScalerで以下を構成するための特定の権限が必要です。

- コンテンツスイッチング (CS) 仮想サーバーの追加、削除、または表示

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- CSポリシーとアクションの構成
- 負荷分散 (LB) 仮想サーバーの構成
- サービスグループの構成
- SSL証明書キーの構成
- ルートの構成
- ユーザーモニターの構成
- システムファイルの追加 (KubernetesからSSL証明書キーをアップロードするため)
- 仮想IPアドレス (VIP) の構成
- NetScalerアプライアンスのステータスの確認

システムユーザーアカウントを作成するには、以下を実行します。

1. NetScalerにログオンします。以下の手順を実行します。
   a. PuTTyなどのSSHクライアントを使用して、NetScalerへのSSH接続を開きます。
   b. 管理者資格情報を使用してNetScalerにログオンします。
2. 次のコマンドを使用してシステムユーザーアカウントを作成します。

   ```
   add system user &lt;username> &lt;password>
   ```

   例：

   ```
   add system user nsic mypassword
   ```

3. システムユーザーアカウントに必要な権限を付与するポリシーを作成します。次のコマンドを使用します。

   ```
   add cmdpolicy nsic-policy ALLOW '\^(\?!shell)(\?!sftp)(\?!scp)(\?!batch)(\?!source)(\?!.*superuser)(\?!.*nsroot)(\?!install)(\?!show\s+system\s+(user|cmdPolicy|file))(\?!(set|add|rm|create|export|kill)\s+system)(\?!(unbind|bind)\s+system\s+(user|group))(\?!diff\s+ns\s+config)(\?!(set|unset|add|rm|bind|unbind|switch)\s+ns\s+partition).*|(\^install\s*(wi|wf))|(\^\S+\s+system\s+file)\^(\?!shell)(\?!sftp)(\?!scp)(\?!batch)(\?!source)(\?!.*superuser)(\?!.*nsroot)(\?!install)(\?!show\s+system\s+(user|cmdPolicy|file))(\?!(set|add|rm|create|export|kill)\s+system)(\?!(unbind|bind)\s+system\s+(user|group))(\?!diff\s+ns\s+config)(\?!(set|unset|add|rm|bind|
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

`unbind|switch)\s+ns\s+partition).*|(\^install\s*(wi|wf))|(\^\S+\s+system\s+file)'`

> **注：**
>
> システムユーザーアカウントには、定義したコマンドポリシーに基づいて権限が付与されます。

ステップ3で述べられているコマンドポリシーは、組み込みの`sysAdmin`コマンドポリシーに似ており、ファイルアップロードの追加権限が付与されています。

提供されているコマンドポリシーの仕様では、エスケープする必要がある特殊文字は、NetScalerコマンドラインに簡単にコピー＆ペーストできるように、すでに省略されています。

NetScaler設定ウィザード（GUI）からコマンドポリシーを設定するには、次のコマンドポリシー仕様を使用します。

```
\^(?!shell)(?!sftp)(?!scp)(?!batch)(?!source)(?!.*superuser)(?!.*nsroot)(?!install)(?!show\s+system\s+(user|cmdPolicy|file))(?!(set|add|rm|create|export|kill)\s+system)(?!(unbind|bind)\s+system\s+(user|group))(?!diff\s+ns\s+config)(?!(set|unset|add|rm|bind|unbind|switch)\s+ns\s+partition).*|(\^install\s*(wi|wf))|(\^\S+\s+system\s+file)\^(?!shell)(?!sftp)(?!scp)(?!batch)(?!source)(?!.*superuser)(?!.*nsroot)(?!install)(?!show\s+system\s+(user|cmdPolicy|file))(?!(set|add|rm|create|export|kill)\s+system)(?!(unbind|bind)\s+system\s+(user|group))(?!diff\s+ns\s+config)(?!(set|unset|add|rm|bind|unbind|switch)\s+ns\s+partition).*|(\^install\s*(wi|wf))|(\^\S+\s+system\s+file)
```

4. 次のコマンドを使用して、ポリシーをシステムユーザーアカウントにバインドします。

```
bind system user nsic nsic-policy 0
```

KubernetesクラスターにNetScaler Ingress Controllerをスタンドアロンポッドとして展開する

NetScaler Ingress Controllerをスタンドアロンポッドとして展開するには：

1. 次のコマンドを使用して、NetScaler Helmチャートリポジトリをローカルレジストリに追加します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/
```

2. 次のコマンドを使用して、NetScaler Ingress Controllerをインストールします。

```
helm install nsic netscaler/netscaler-ingress-controller --set nsIP=<NSIP>,license.accept=yes,adcCredentialSecret=<Secret-for-NetScaler-credentials>,entityPrefix="<provideprefix>"
```

注：

各パラメータをコマンドライン引数として指定する代わりに、`helm install`コマンドで`values.yaml`ファイルを使用して、設定可能なパラメータの値を指定することもできます。例：`helm install nsic netscaler/netscaler-ingress-controller -f values.yaml`。

NSICのインストール中に設定できる必須およびオプションのパラメータについては、構成を参照してください。

警告：

KubernetesクラスターでNSICポッドのレプリカを作成しないでください。`replicatset>1`を指定してNSICポッドのレプリカを作成すると、同じAPP_PREFIXがNSICポッド間で共有され、各インスタンスが同じリソースを構成しようとするため、NSIC/VPXで予期しない動作が発生します。

NetScaler MPXまたはNetScaler VPXでのルート追加

Kubernetesクラスターに展開されたサービスがシームレスに機能するためには、NetScalerがポッドが実行されている基盤となるオーバーレイネットワークに到達できる必要があります。NetScaler Ingress Controllerの`feature-node-watch`引数を使用すると、ポッドネットワークへのNetScaler上の自動ルート構成が可能です。詳細については、静的ルーティングを参照してください。デフォルトでは、`feature-node-watch`はfalseに設定されています。自動ルート構成を有効にするには、trueに設定します。

複数のk8sクラスター間で負荷分散するために単一のNetScalerを使用する展開では、CNIサブネットが重複する可能性があり、ルート競合により静的ルーティングが失敗する可能性があります。このような展開では、ポリシーベースルーティング（PBR）を使用できます。PBRでは、環境変数

© 1997–2026 Citrix Systems, Inc. All rights reserved.

またはConfigMapを使用して、各Kubernetesクラスターに固有の1つ以上のサブネットIPアドレスを提供する必要があります。PBRサポートを参照してください。

NetScalerでポリシーベースルート（PBR）を構成するために、サブネットIPアドレス（SNIP）を提供するには、次のコマンドを使用します。

```
helm install my-release netscaler/netscaler-ingress-controller --set
nsIP=<NSIP>,license.accept=yes,adcCredentialSecret=<Secret-for-NetScaler-
credentials>,nsSNIPS='[<NS_SNIP1>\, <NS_SNIP2>\, ...]'
```

注：

PBRは、NetScaler Node Controllerを展開することでも実現できます。NetScaler Node Controllerは、デフォルトでVXLANトンネルの作成時に静的ルートを追加します。静的ルートの競合を避けるためにポリシーベースルーティング（PBR）を使用するには、NetScaler Node ControllerとNetScaler Ingress Controllerの両方が連携して動作し、特定の引数で起動する必要があります。詳細については、NetScalerノードコントローラーを使用したPBRの構成を参照してください。

NSIC の helm install コマンドで `nsncPbr=<True/False>` パラメータを使用して、NetScaler Node Controller が NetScaler 上でポリシーベースルート (PBR) を構成していることを NSIC に通知します。

```
helm install my-release netscaler/netscaler-ingress-controller --set
nsIP=<NSIP>,license.accept=yes,adcCredentialSecret=<Secret-for-NetScaler-
credentials>,clusterName=<unique-cluster-identifier>,nsncPbr=<True/False>
```

デュアルティアイングレス

デュアルティア展開では、NetScaler VPX または NetScaler MPX が Kubernetes クラスターの外部（ティア1）に展開され、NetScaler CPX は Kubernetes クラスターの内部（ティア2）に展開されます。

ティア1のNetScaler MPXまたはNetScaler VPXデバイスは、クライアントからのトラフィック（南北）をティア2のNetScaler CPXにプロキシします。ティア2のNetScaler CPXは、そのトラフィックをKubernetesクラスター内のマイクロサービスにルーティングします。スタンドアロンポッドとして展開されたNetScaler Ingress Controllerは、ティア1のNetScalerを設定します。そし

© 1997–2026 Citrix Systems, Inc. All rights reserved.

て、1つ以上のNetScaler CPXポッド内のサイドカーNetScaler Ingress Controllerは、同じポッド内の関連するNetScaler CPXを設定します。

デュアルティア展開は、ベアメタル環境のKubernetes、またはAWS、GCP、Azureなどのパブリッククラウドにセットアップできます。

デュアルティアイングレス展開

前提条件

- クラウドまたはオンプレミスにKubernetesクラスターを作成します。クラウド内のKubernetesクラスターは、マネージドKubernetes（例：GKE、EKS、AKS）またはカスタム作成されたKubernetes展開のいずれかです。

デュアルティアイングレス展開

1. Kubernetesクラスターの外部で、NetScaler MPXまたはNetScaler VPXをマルチNIC展開モードで展開します。
   - NetScaler MPXの展開手順については、NetScaler documentationを参照してください。
   - NetScaler VPXの展開手順については、Deploy a NetScaler VPX instanceを参照してください。
2. スタンドアロンのNetScaler Ingress Controllerを展開して、NetScaler VPXを設定し、クライアント要求をKubernetesクラスター内のCPXサービスに転送するようにします。

   スタンドアロンのNetScaler Ingress Controllerの展開方法については、Unified Ingressを参照してください。ステップ8でNetScaler VPXを設定するために展開されたイングレスリソースが同じイングレスクラスを使用するため、このNSICインスタンスの`ingressClass`を`nsic-vpx`として設定します。
3. クラウド展開では、ティア1のNetScaler VPXでMAC-Based Forwarding modeを有効にします。NetScaler VPXはマルチNICモードで展開されているため、POD CNIネットワークまたはクライアントネットワークに到達するための戻りルートがありません。したがって、このシナリオを処理するために、ティア1のNetScaler VPXでMAC-Based Forwardingモードを有効にする必要があります。
4. 次のコマンドを使用して、サンプルマイクロサービスをデプロイします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl apply -f - <<EOF

apiVersion: apps/v1

kind: Deployment

metadata:

 name: cnn-website

 labels:

   name: cnn-website

   app: cnn-website

spec:

 selector:

   matchLabels:

     app: cnn-website

 replicas: 2

 template:

   metadata:

     labels:

       name: cnn-website

       app: cnn-website

   spec:

     containers:

     - name: cnn-website

       image: quay.io/sample-apps/cnn-website:v1.0.0
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      ports:
      - name: http-80
        containerPort: 80
      - name: https-443
        containerPort: 443
---
apiVersion: v1
kind: Service
metadata:
 name: cnn-website
 labels:
   app: cnn-website
spec:
 type: NodePort
 ports:
 - name: http-80
   port: 80
   targetPort: 80
 - name: https-443
   port: 443
   targetPort: 443
 selector:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
   name: cnn-website

EOF
```

5. NetScaler CPX をティア2イングressとして、NetScaler Ingress Controller をサイドカーとしてデプロイします。

   a. 次のコマンドを使用して、NetScaler Helm チャートリポジトリをローカルレジストリに追加します。

   ```
   helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/
   ```

   b. 次のコマンドを使用して、NetScaler Ingress Controller とともに NetScaler CPX をインストールします。

   ```
   helm install netscaler-cpx-with-ingress-controller netscaler/netscaler-cpx-with-ingress-controller --set license.accept=yes,serviceType.nodePort.enabled=True,ingressClass[0]=cpx
   ```

   注:

   各パラメータをコマンドライン引数として指定する代わりに、`helm install` コマンドで `values.yaml` ファイルを使用して、設定可能なパラメータの値を指定することもできます。例えば、,

   ```
   helm install netscaler-cpx-with-ingress-controller netscaler/netscaler-cpx-with-ingress-controller -f values.yaml
   ```

   NSIC インストール時のCPXで設定できる必須およびオプションのパラメータについては、構成を参照してください。

6. 証明書を使用してKubernetesシークレットを作成し、ingressオブジェクトがデプロイされたときに証明書をNetScalerにコピーします。CPXとVPXの両方のingressリソースで同じシークレットを使用できます。この手順では、`tls-secret` を使用します。Kubernetesシークレットの生成方法については、Kubernetesシークレットの生成を参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

7. 次のコマンドを使用して、Kubernetesクラスター内のNetScaler VPXまたはNetScaler MPXからNetScaler CPXにトラフィックをルーティングするingressオブジェクトを作成します。

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  annotations:

    ingress.citrix.com/frontend-ip: <NSVIP> # vserver IP

    ingress.citrix.com/secure_backend: "True"

  name: vpxingress

  spec:

    tls:

      - secretName: tls-secret

    ingressClassName: nsic-vpx

    rules:

    - host: cnnwebsite.com

      http:

        paths:

        - backend:

            service:

              name: netscaler-cpx-with-ingress-controller-cpx-service

              port:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
                number: 443

          path: /

          pathType: Prefix

---

apiVersion: networking.k8s.io/v1

kind: IngressClass # If Helm chart is used to deploy NSIC, you need not
create IngressClass

metadata:

  name: nsic-vpx

spec:

  controller: citrix.com/ingress-controller
```

8. 次のコマンドを使用して、ティア2のNetScaler CPX用のingressオブジェクトを作成し、マイクロサービスアプリケーションにトラフィックをルーティングします。

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

apiVersion: networking.k8s.io/v1

metadata:

  annotations:

  name: cpxingress

spec:

  tls:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    - secretName: tls-secret
  ingressClassName: cpx
  rules:
    - host: cnnwebsite.com
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: cnn-website
                port:
                  number: 80
---
apiVersion: networking.k8s.io/v1
kind: IngressClass # If Helm chart is used to deploy CPX with NSIC, you need not create IngressClass
metadata:
  name: cpx
spec:
  controller: citrix.com/ingress-controller
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
EOF
```

9. `cnn-website` アプリケーションにトラフィックを送信するには、次の手順を実行します。

   Windowsの場合:

   a. 管理者としてメモ帳を開きます。

   b. `c:\Windows\System32\Drivers\etc\hosts` に `VIP cnnwebsite.com` エントリを追加します。

   MACまたはLinuxの場合:

   a. 次のコマンドを使用して、ファイルエディターでホストファイルを開きます: `sudo nano /etc/hosts`。

   b. `VIP cnnwebsite.com`エントリを追加します。

10. 次のコマンドを使用してトラフィックを送信します:

```
curl  -kv https://cnn-website
```

警告:

KubernetesクラスターにNSICポッドのレプリカを作成しないでください。`replicatset>1`を指定してNSICポッドのレプリカを作成すると、同じAPP_PREFIXがNSICポッド間で共有され、各インスタンスが同じリソースを構成しようとするため、NSIC/VPXで予期しない動作が発生します。

マルチクラスターイングレス

はじめに

マルチクラスターKubernetesソリューションは、複数のクラスターにワークロードを分散するのに理想的です。NetScalerマルチクラスターイングレスソリューションは、単一のフロントエンドIPアドレスを使用して、クラスター全体に分散されたアプリケーションをNetScalerがロードバランシングできるようにします。ロードバランシングされたアプリケーションは、同じアプリケーション、同じドメインの異なるアプリケーション、またはまったく異なるアプリケーションのいずれかになります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

以前は、複数のクラスターでアプリケーションをロードバランシングするには、クラスターで実行されているNetScaler Ingress Controller (NSIC) の各インスタンスに対して、NetScaler上に専用のコンテンツスイッチング仮想サーバーが必要でした。NetScalerマルチクラスターイングレスソリューションを使用すると、複数のイングレスコントローラーがコンテンツスイッチング仮想サーバーを共有できます。したがって、クラスター全体にデプロイされたアプリケーションは、同じコンテンツスイッチング仮想サーバーIP (VIP) アドレスを使用してロードバランシングできます。

要約すると、マルチクラスターイングレスソリューションは、ロードバランシングリソースの使用を最適化し、それによって運用コストを削減します。

注:

マルチクラスターイングレスソリューションは、NSICバージョン2.0.6以降でサポートされています。

デプロイメントトポロジー

次の図は、データセンター/サイトにおける2つのKubernetesクラスターのマルチクラスターイングレスデプロイメントトポロジーを示しています。ここでは、NetScalerが単一のフロントエンドIPアドレスを使用して、クラスター全体に分散されたアプリケーションをロードバランシングします。

- 両方のクラスターにNSICがデプロイされます。両方のNSICインスタンスは同じNetScalerを構成します。
- クラスターにデプロイされるアプリケーションの性質に応じて、通常、次のユースケースがあります。
  - 各Kubernetesクラスターにデプロイされた同じドメイン (company.website.com) の異なるアプリケーション: cluster1の`App-B`とcluster2の`App-C`。

    ここでは、NetScaler上のコンテンツスイッチング仮想サーバーが`App-B`と`App-C`間でトラフィックをロードバランシングします。各アプリケーションに対して個別のコンテンツスイッチングポリシーが構成されます。
  - 両方のKubernetesクラスターにデプロイされた同じアプリケーション。

    ここでは、NetScaler上のコンテンツスイッチング仮想サーバーが、両方のクラスターにデプロイされた同じアプリケーション`App-A`のトラフィックをロードバランスします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

`App-A` 用に作成されるコンテンツスイッチングポリシーは1つだけで、これはアプリケーションの実行中のインスタンス（エンドポイント）の両方で使用されます。

以下の図で、同じフロントエンドIPアドレスがcluster1とcluster2にデプロイされたアプリケーション `App-B` と `App-c` へのトラフィックをロードバランスするためにどのように使用されるかを理解しましょう。

次のリソースを構成すると、マルチクラスターイングレスソリューションが有効になります: リスナー と イングレス。

リスナー

- クラスターにデプロイされたリスナーリソースは、リスナーYAMLで指定されたVIPを使用してNetScaler上にコンテンツスイッチング仮想サーバーを作成します。マルチクラスターイングレス設定では、同じリスナーリソースがすべてのクラスターにデプロイされます。

  マルチクラスターイングレスソリューションのリスナーリソースは、フロントエンドのトラフィック管理を処理します。コンテンツスイッチング仮想サーバーに必要なすべてのシークレット、暗号、およびフロントエンドプロファイル構成を提供する必要があります。リスナーCRDの詳細については、リスナー を参照してください。マルチクラスターイングレス設定用のリスナーリソースのデプロイに関する詳細は、マルチクラスターイングレスソリューションをデプロイする手順 セクションで入手できます。

イングレス

- マルチクラスター設定でアプリケーションにトラフィックをルーティングするために各クラスターにデプロイされたイングレスリソースは、共有VIPアドレスを持つ同じリスナーリソースを参照します。したがって、NSICは既存の同じコンテンツ仮想サーバーを参照し、NetScaler上にコンテンツスイッチングポリシーのみを作成し、これらは後でコンテンツスイッチング仮想サーバーにバインドされます。

この場合、各アプリケーションのコンテンツスイッチングポリシー、つまりCSPOL-App-BとCSPOL-App-Cは、同じコンテンツスイッチング仮想サーバーにバインドされます。例えば、`company.website/App-B` のリクエストがコンテンツスイッチング仮想サーバーのIP（VIP）アドレスに送信されると、コンテンツスイッチング仮想サーバーはCSPOL-App-Bポリシーを適用し、リクエストを `LBvserver-App-B` にルーティングし、それがApp-Bサービスに送信します。

マルチクラスターイングレスのデプロイ

この手順では、同じHTTPSアプリケーションを2つのクラスターにデプロイし、両方のクラスターに必要なリスナーリソースとイングレスリソースをデプロイします。これにより、NetScaler VPX

© 1997–2026 Citrix Systems, Inc. All rights reserved.

またはNetScaler MPXが単一のフロントエンドIPアドレスを使用してこれらのアプリケーションをロードバランスします。

前提条件

- クラウドまたはオンプレミスでホストされているKubernetesクラスターへのアクセス。クラウド内のKubernetesクラスターは、マネージドKubernetes（例：GKE、EKS、AKS）またはカスタム作成されたKubernetesデプロイメントのいずれかです。KubernetesまたはOpenShiftは適切に構成され、運用可能である必要があります。NetScalerとクラスターのPodネットワーク間のネットワーク接続が確保されていることを確認してください。
- Kubernetesデプロイメントの場合、バージョン1.21以降を実行しているKubernetesクラスターへのアクセスが必要です。
- OpenShiftのデプロイメントには、バージョン4.11以降を実行しているOpenShiftクラスターへのアクセスが必要です。
- Helmバージョン3.x以降がインストールされていること。Helmのインストールについては、こちらを参照してください。
- NetScaler MPX/VPXは、特定のニーズに応じて、スタンドアロン、HA、またはクラスター構成でデプロイされます。
    - NetScaler MPXのデプロイ手順については、NetScaler documentationを参照してください。
    - NetScaler VPXのデプロイ手順については、Deploy a NetScaler VPX instanceを参照してください。
- NetScaler Ingress ControllerがNetScalerと通信するために使用するNSIP (NetScaler IP) アドレスを決定します。IPアドレスは、NetScalerのデプロイの種類に応じて、以下のいずれかになります。
    - NSIP (スタンドアロンアプライアンスの場合) - スタンドアロンNetScalerアプライアンスの管理IPアドレス。詳細については、IP Addressing in NetScalerを参照してください。
    - SNIP (高可用性モードのアプライアンスの場合) - サブネットIPアドレス。詳細については、IP Addressing in NetScalerを参照してください。
    - CLIP (クラスターモードのアプライアンスの場合) - クラスターNetScalerデプロイメントのクラスター管理IP (CLIP) アドレス。詳細については、IP addressing for a clusterを参照してください。
- NetScaler VPXまたはNetScaler MPXのユーザーアカウント。NetScaler Ingress Controllerは、NetScaler MPXまたはNetScaler VPXを構成するために、NetScalerのシステムユーザーアカウ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ントを使用します。NetScalerでシステムユーザーアカウントを作成する手順については、Create System User Account for NetScaler Ingress Controller in NetScalerを参照してください。

ユーザー名とパスワードを直接渡すか、Kubernetesシークレットを使用できます。Kubernetesシークレットを使用する場合は、次のコマンドを使用してユーザー名とパスワードのシークレットを作成します。

```
kubectl create secret generic nslogin --from-
literal=username=<username> --from-literal=password=<password>
```

マルチクラスターイングレスソリューションをデプロイする手順

NetScalerと共有フロントエンドIPアドレスを使用して、2つのKubernetesクラスターにデプロイされたCloud Native Networking (CNN) アプリケーションを公開するマルチクラスターイングレスソリューションをデプロイするには、次の手順に従います。

注記:

マルチクラスターイングレスソリューションをセットアップするには、両方のクラスターで次の手順を繰り返します。特定のクラスターで何を行う必要があるかを強調する例外がいくつかの手順で追加されています。それに応じて手順を実行してください。

1. 以下のYAMLを使用してNetScaler CNNアプリケーションをデプロイします。

   CNNアプリケーションは、NetScaler®クラウドネイティブポートフォリオで提供されるソリューションを一覧表示するHTTPベースのアプリケーションです。

```
kubectl apply -f - <<EOF

apiVersion: apps/v1

kind: Deployment

metadata:

  name: cnn-website

  labels:

    name: cnn-website
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    app: cnn-website
spec:
  selector:
    matchLabels:
      app: cnn-website
  replicas: 2
  template:
    metadata:
      labels:
        name: cnn-website
        app: cnn-website
    spec:
      containers:
      - name: cnn-website
        image: quay.io/sample-apps/cnn-website:v1.0.0
        ports:
        - name: http-80
          containerPort: 80
        - name: https-443
          containerPort: 443
---
apiVersion: v1
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kind: Service

metadata:

  name: cnn-website

  labels:

    app: cnn-website

spec:

  type: NodePort

  ports:

  - name: http-80

    port: 80

    targetPort: 80

  - name: https-443

    port: 443

    targetPort: 443

  selector:

    name: cnn-website

EOF
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

OpenShiftデプロイメントの場合、サービスアカウントに特定のSecurity Context Constraint (SCC) を付与するには、次のコマンド `oc adm policy add-scc-to-user anyuid system:serviceaccount:<namespace>:default` を実行します。`<namespace>` を、CNNアプリケーションをデプロイした実際の名前空間に置き換えてください。

2. 次のコマンドを使用して、NetScaler Helmチャートリポジトリをローカルレジストリに追加します。

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/
```

NetScaler Helmチャートリポジトリがすでにローカルレジストリに追加されている場合は、次のコマンドを使用してリポジトリを更新します。

```
helm repo update netscaler
```

3. NetScaler Ingress Controllerを次のように構成するには、`values.yaml` を更新します。

cluster1の例 `values.yaml`

```
license:

  accept: yes

adcCredentialSecret: nslogin # K8s Secret Name

nsIP: <x.x.x> # CLIP (for appliances in Cluster mode), SNIP (for appliances in High Availability mode) , NSIP (for standalone appliances)

openshift: false # set to true for OpenShift deployments

entityPrefix: cluster1 # unique for each NSIC instance.

clusterName: cluster1

ingressClass: ['nsic-vpx'] # ingress class used in the ingress
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
resources

multiClusterPrefix: mc # Multi-cluster prefix for the NSIC instance.
Same value must be specified for a set of NSIC instances configuring
NetScaler in multi-cluster setup.

# serviceClass- To use service type LB, specify the service class
```

cluster2の例 `values.yaml`

```
license:

  accept: yes

adcCredentialSecret: nslogin # K8s Secret Name

nsIP: <x.x.x> # CLIP (for appliances in Cluster mode), SNIP (for
appliances in High Availability mode) , NSIP (for standalone
appliances)

openshift: false # set to true for OpenShift deployments

entityPrefix: cluster2 # unique for each NSIC instance.

clusterName: cluster2

ingressClass: ['nsic-vpx'] # ingress class used in the ingress
resources

multiClusterPrefix: mc # Multi-cluster prefix for the NSIC instance.
Same value must be specified for a set of NSIC instances configuring
NetScaler in multi-cluster setup.

# serviceClass- To use service type LB, specify the service class
```

注:

OpenShiftデプロイメントの場合、`values.yaml` で `openshift` パラメータを `true` に設定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NSICのインストール中に構成できる必須およびオプションのパラメータについては、構成 を参照してください。

4. 変更された `values.yaml` を使用してNetScaler Ingress Controllerをデプロイします。

```
helm install nsic netscaler/netscaler-ingress-controller -f values.yaml
```

注:

以前のバージョンのNSICがすでにクラスターにデプロイされている場合は、次のコマンドを使用してリスナーCRD仕様をデプロイします: `kubectl apply -f https://raw.githubusercontent.com/netscaler/netscaler-k8s-ingress-controller/master/crd/contentrouting/Listener.yaml`。

5. 以下のリスナーCRDリソースをデプロイします。

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1

kind: Listener

metadata:

  name: mc-listener

spec:

  multicluster: True

  ingressclass: nsic-vpx

  protocol: 'https'

  vip: <Provide the shared front-end IP address>

  certificates:

  #  you need to specify either secret name or pre-configured cert-
keyname
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    - secret:

        name: <Provide k8s secret name>  # provide k8s secret
containing cert-key of the application

      default: true

    - preconfigured: <ADC-certkeyname> # provide pre-configured cert-
key from ADC.

EOF
```

- `vip`を、両方のクラスターにデプロイされたアプリケーションを公開するために使用される仮想IPアドレスで更新します。
- `multicluster`は`True`として設定する必要があります。
- NetScalerで作成されたSSL証明書キー名で事前設定されたシークレットを更新するか、`secret.name`セクションでKubernetes TLSシークレットの名前を更新します。Listener.certificatesを参照してください。

この例では、リスナーリソース`mc-listener`はVIPやシークレットなどのフロントエンド構成を指定します。NetScalerにHTTPSトラフィック用のポート443でコンテンツスイッチング仮想サーバーを作成します。

6. CNNアプリケーションを公開するために、以下のイングレスリソースをデプロイします。

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: frontend-ingress

  annotations:

    ingress.citrix.com/listener: mc-listener

spec:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  ingressClassName: nsic-vpx
  rules:
    - http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: cnn-website
                port:
                  number: 80
---
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: nsic-vpx
spec:
  controller: citrix.com/ingress-controller
EOF
```

- 前のステップで作成されたリスナーリソースは、`ingress.citrix.com/listener`アノテーションを使用して渡す必要があります。 この例では、イングレスリソース`frontend-ingress`はリスナーリソース`mc-listener`（ステップ5で作成）を参照しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- イングレスクラスは`nsic-vpx`として言及されています。

Ingressは、コンテンツスイッチングポリシー、ロードバランシング仮想サーバー、サービスグループを作成し、アプリケーションポッドのIPアドレスをサービスグループメンバーとしてバインドします。

注記：

- マルチクラスターイングレス設定で異なるアプリケーションをロードバランシングする場合、各アプリケーションに対して個別のコンテンツスイッチングポリシーが作成されます。このような場合、ポリシーバインディングに特定の順序が必要な場合は、`ingress.citrix.com/multicluster-policy-priority-order`アノテーションを使用してコンテンツスイッチングポリシーに優先度番号を割り当てる必要があります。詳細については、ポリシーバインディングを参照してください。
- マルチクラスターイングレス設定で異なるアプリケーションをロードバランシングする場合のリスナーおよびイングレスリソースに関する情報については、異なるアプリケーションを使用したマルチクラスターイングレス設定を参照してください。

7. 構成を検証するには、`<VIP>`をステップ5のリスナーリソースで提供されている仮想IPアドレスに置き換えます。

```
curl -kv https://<VIP>
```

高度なユースケース

同じアプリケーションが複数のクラスターにデプロイされている場合、トラフィック分散要件が異なる多様なデプロイシナリオが発生します。`Active-Active`として知られるデフォルトの動作では、ロードバランシング方法に基づいて、アプリケーションのすべてのインスタンスがトラフィックを受信します。ここでは、同じアプリケーションが2つのクラスターにデプロイされている場合の高度なユースケースを見てみましょう。

アクティブ-バックアップモード

このモードでは、1つのアプリケーションが常にアクティブであり、常にトラフィックを受信します。他のアプリケーションは、このクラスターまたはアプリケーションがダウンしている場合にのみトラフィックを受信します。`ingress.citrix.com/multicluster-backup-order`イングレスアノテーションを使用して、どのアプリケーションをバックアップとして機能させるかを定義できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

同じリスナーリソースを両方のクラスターにデプロイする必要があります。リスナーリソースの例については、マルチクラスターイングレスソリューションをデプロイする手順 セクションのステップ5を参照してください。

	クラスター1 (アクティブ)	クラスター2 (バックアップ)
イングレスアノテーション	ingress.citrix.com/ multicluster-backup-order: "1"	ingress.citrix.com/ multicluster-backup-order: "2"
情報	デフォルトはアクティブ (1) です。アノテーションはスキップできます。	アプリケーションをバックアップと見なすには、アノテーションが必須です。

- クラスター1にデプロイされたアプリケーションはアクティブにトラフィックを受信し、クラスター2はバックアップとして機能します。クラスター2のアプリケーションは、クラスター1またはクラスター1のアプリケーションがダウンしている場合にのみトラフィックを受信します。

カナリアモード

注:

以下のデプロイメントでは、各クラスターに同じリスナーリソースをデプロイする必要があります。リスナーリソースの例については、「マルチクラスターIngressソリューションをデプロイする手順」セクションのステップ5を参照してください。

カナリアデプロイメントは、「重みによるカナリア」と「ヘッダーによるカナリア」という2つの異なる戦略をサポートしています。

重みによるカナリアデプロイメント

アプリケーションのインスタンスに異なる重みを割り当てることで、カナリアデプロイメントを実装できます。たとえば、2つのクラスターがあり、アプリケーションの新しいバージョンがロールアウトされている場合、トラフィックの特定の割合を新しいバージョン（カナリア）に割り当て、残りのトラフィックを既存のバージョンに割り当てることができます。このカナリアモードにより、新しいバージョンのパフォーマンスを制御された方法で段階的にロールアウトし、監視することができます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

	クラスター1 (バージョン1)	クラスター2 (バージョン2)
Ingressアノテーション	NA	ingress.citrix.com/ multicluster-canary-weight: "10"
情報	アノテーションは不要です	アノテーションは必須です。このアプリケーションはクライアントトラフィックの10%を受け取ります。

ヘッダーによるカナリアデプロイメント

特定のHTTPヘッダーに基づいてカナリアデプロイメントを実装できます。たとえば、「X-Canary-Version」のようなHTTPリクエスト内の特定のヘッダーを使用して、トラフィックのルーティングを制御できます。この場合、「X-Canary-Version」ヘッダーを持つリクエストはアプリのカナリアバージョンに転送され、それ以外のリクエストは安定版にルーティングされます。このカナリアモードは、カナリアデプロイメントをよりきめ細かく制御し、テスト目的で特定のユーザーまたはリクエストのサブセットをターゲットにすることを可能にします。

	クラスター1 (バージョン1)	クラスター2 (バージョン2)
アノテーション	NA	ingress.citrix.com/ multicluster-canary-by-header: "version2"
情報	アノテーションは不要です	アノテーションはカナリアアプリケーションに必須です。`version2` ヘッダーを持つすべてのクライアントトラフィックはクラスター2に送信されます。

ポリシーバインディング

ポリシーバインディングに特定のシーケンスが必要な場合は、`ingress.citrix.com/multicluster-policy-priority-order` アノテーションを使用してコンテンツスイッチングポリシーに優先度番号を割り当てる必要があります。数値が小さいほど、優先度が高くなります。

ポリシーバインディングを理解するために、次のイングレスリソースの例を考えてみましょう。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: mc-ing

  annotations:

    ingress.citrix.com/listener: mc-listener

    ingress.citrix.com/multicluster-policy-priority-order: '{"frontend":
{"80": "3", "9443": "1"}, "backend": "2"}'

spec:

  ingressClassName: nsic-vpx

  rules:

    - http:

        paths:

          - path: /

            pathType: Prefix

            backend:

              service:

                name: frontend

                port:

                  number: 80

          - path: /abc
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            pathType: Prefix
            backend:
              service:
                name: frontend
                port:
                  number: 9443
          - path: /xyz
            pathType: Prefix
            backend:
              service:
                name: backend
                port:
                  number: 80
EOF
```

このリソースの例には、コンテンツスイッチングポリシーの優先順位を定義する `ingress.citrix.com/multicluster-policy-priority-order` アノテーションが含まれています。NetScaler Ingress Controller は、優先度をランダムに割り当てるのではなく、提供された優先度値を使用してコンテンツスイッチングポリシーをコンテンツスイッチング仮想サーバーにバインドします。

ingress.citrix.com/multicluster-policy-priority-order: '{"Front end": {"80": "3", "9443": "1"}, "back-end": "2"}' アノテーションの場合、優先度は次のように割り当てられます。

- `frontend:80` サービスに関連付けられたコンテンツスイッチングポリシーは、優先度3でコンテンツスイッチング仮想サーバーにバインドされます。
- `frontend:9443` サービスに関連付けられたコンテンツスイッチングポリシーは、優先度1でコンテンツスイッチング仮想サーバーにバインドされます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- `backend:80` サービスに関連付けられたコンテンツスイッチングポリシーは、優先度2でコンテンツスイッチング仮想サーバーにバインドされます。

警告:

イングレスリソースで単一のサービスが異なるポートで言及されている場合、各ポートに優先度番号を明示的に指定する必要があります。そうしないと、ランダムな優先度が割り当てられます。

異なるアプリケーションを使用したマルチクラスターイングレスのセットアップ

異なるアプリケーションの負荷分散のためのマルチクラスターイングレス設定では、各クラスターにデプロイされるリスナーリソースは同じである必要があります。各クラスターのイングレスリソースは同じリスナーを参照する必要があります。デプロイメントトポロジセクションで説明されている例では、`mc-listener`はcluster1とcluster2の両方にデプロイされたリスナーリソースです。cluster1とcluster2にデプロイされた以下のサンプルイングレスリソースは`mc-listener`を参照します。

cluster1で`App-B`を公開するためのイングレスリソース:

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

name: app-b

annotations:

  ingress.citrix.com/listener: mc-listener

spec:

ingressClassName: cluster1

rules:

- http:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    paths:
    - path: /App-B
      pathType: Prefix
      backend:
        service:
          name: app-b-svc
          port:
            number: 80
EOF
```

cluster2で`App-C`を公開するためのイングレスリソース:

```
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
name: app-c
annotations:
  ingress.citrix.com/listener: mc-listener
spec:
ingressClassName: cluster2
rules:
- http:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    paths:
    - path: /App-C
      pathType: Prefix
      backend:
        service:
          name: app-c-svc
          port:
            number: 80
EOF
```

Rancherで管理されるKubernetesクラスターにNetScaler Ingress Controllerを展開する

Rancherは、Kubernetesクラスターを簡単に展開および管理できる直感的なユーザーインターフェイスを備えたオープンソースプラットフォームです。Rancherは、クラウドまたはオンプレミス展開のいずれのインフラストラクチャでもKubernetesクラスターをサポートします。Rancherを使用すると、組織全体で実行されている複数のクラスターを一元的に管理することもできます。

NetScaler Ingress ControllerはKubernetesのIngressを中心に構築されており、Ingressリソース構成に基づいて1つ以上のNetScalerを自動的に構成できます。NetScaler Ingress ControllerをRancherで管理されるKubernetesクラスターに展開して、NetScalerの高度なロードバランシングおよびトラフィック管理機能をクラスターに拡張できます。

前提条件

Kubernetesクラスターを作成し、Rancherプラットフォームにクラスターをインポートする必要があります。

展開オプション

NetScaler CPXをクラスター内のポッドとして展開するか、NetScaler MPXまたはVPXアプライアンスをKubernetesクラスターの外部に展開することができます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScalerの使用方法に応じて、Rancherプラットフォーム上のKubernetesクラスターにNetScaler Ingress Controllerを展開する方法は2つあります。

- 同じポッド内のNetScaler CPXと並行してサイドカーコンテナとして展開する: このモードでは、NetScaler Ingress ControllerがNetScaler CPXを構成します。
- Kubernetesクラスター内のスタンドアロンポッドとして展開する: このモードでは、クラスターの外部に展開されたNetScaler MPXまたはVPXアプライアンスを制御できます。

NetScaler CPXのサイドカーとしてNetScaler Ingress Controllerを展開する

この展開では、NetScaler CPXインスタンスを使用して、Kubernetesクラスター内のマイクロサービスへの南北トラフィックのロードバランシングを行うことができます。NetScaler Ingress Controllerは、`citrix-k8s-cpx-ingress.yaml`ファイルを使用して、NetScaler CPXコンテナと並行して同じポッド内にサイドカーとして展開されます。

RancherプラットフォームでNetScaler CPXのサイドカーとしてNetScaler Ingress Controllerを展開するには、次の手順を実行します。

1. 次のコマンドを使用して、`citrix-k8s-cpx-ingress.yaml`ファイルをダウンロードします。

   ```
   wget  https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/baremetal/citrix-k8s-cpx-ingress.yml
   ```

2. Rancher GUIのクラスターページで、グローバルビューからを選択します。
3. 「クラスター」ページから、アクセスしたいクラスターを開きます。
4. `Launch kubectl` をクリックして、Kubernetes クラスターと対話するためのターミナルを開きます。
5. 起動したターミナルで `cpx.yaml` という名前のファイルを作成し、変更した `citrix-k8s-cpx-ingress.yaml` ファイルの内容を `cpx.yaml` ファイルにコピーします。
6. 次のコマンドを使用して、新しく作成した YAML ファイルをデプロイします。

   ```
   kubectl create -f cpx.yaml
   ```

7. 次のコマンドを使用して、NetScaler Ingress Controller が正常にデプロイされていることを確認します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl get pods --all-namespaces
```

NetScaler Ingress Controller をスタンドアロンポッドとしてデプロイする

このデプロイでは、スタンドアロンポッドとして実行される NetScaler Ingress Controller を使用して、Kubernetes クラスターから NetScaler MPX または VPX アプライアンスを制御できます。このデプロイには `citrix-k8s-ingress-controller.yaml` ファイルを使用できます。

開始する前に: NetScaler Ingress Controller をデプロイするために必要なすべての 前提条件 を満たしていることを確認してください。

Rancher プラットフォームに NetScaler Ingress Controller をスタンドアロンポッドとしてデプロイするには:**

1. 次のコマンドを使用して `citrix-k8s-ingress-controller.yaml` ファイルをダウンロードします。

   ```
   wget https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/baremetal/citrix-k8s-ingress-controller.yaml
   ```

2. `citrix-k8s-ingress-controller.yaml` ファイルを編集し、NetScaler Ingress Controller をポッドとしてデプロイする の情報を使用して環境変数の値を入力します。

   注: NetScaler Ingress Controller によって管理される Ingress リソースの `Status.LoadBalancer.Ingress` フィールドを割り当てられた IP アドレスで更新するには、NetScaler Ingress Controller を起動するときにコマンドライン引数 `--update-ingress-status yes` を指定する必要があります。詳細については、指定された IP アドレスを持つ Ingress リソースの Ingress ステータスを更新する を参照してください。

3. Rancher GUI のクラスターページで、グローバルビューから「クラスター」を選択します。
4. 「クラスター」ページから、アクセスしたいクラスターを開きます。
5. `Launch kubectl` をクリックして、Kubernetes クラスターと対話するためのターミナルを開きます。
6. 起動したターミナルで`cic.yaml`という名前のファイルを作成し、変更された`citrix-k8s-ingress-controller.yaml`ファイルの内容を`cic.yaml`にコピーします。
7. 次のコマンドを使用して、`cic.yaml`ファイルをデプロイします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl create -f cic.yaml
```

8. 次のコマンドを使用して、NetScaler Ingress Controllerが正常にデプロイされているかを確認します。

```
kubectl get pods --all-namespaces
```

PKS管理のKubernetesクラスターにNetScaler Ingress Controllerを展開する

Pivotal Container Service (PKS)は、BOSHとPivotal Ops Managerを使用して、オペレーターがエンタープライズグレードのKubernetesクラスターをプロビジョニング、運用、管理することを可能にします。

NetScaler Ingress ControllerはKubernetesのIngressを中心に構築されており、Ingressリソース構成に基づいて1つ以上のNetScalerを自動的に構成できます。NetScaler Ingress ControllerをPKS管理のKubernetesクラスターに展開することで、NetScalerの高度なロードバランシングおよびトラフィック管理機能をクラスターに拡張できます。

前提条件

PKSを使用してKubernetesクラスターを作成する前に。Pivotal Ops Managerで利用可能なすべてのプランについて、以下のオプションが設定されていることを確認してください。

- 特権コンテナを有効にする
- DenyEscalatingExecを無効にする

PKSフレームワークおよびその他のドキュメントの詳細については、Pivotal Container Service ドキュメントを参照してください。

必要なオプションを設定したら、PKS CLIフレームワークを使用してKubernetesクラスターを作成し、作成したクラスターのコンテキストを設定します。

展開オプション

NetScaler CPXをクラスター内のポッドとして展開するか、NetScaler MPXまたはVPXアプライアンスをKubernetesクラスターの外部に展開することができます。

NetScalerの使用方法に応じて、PKS上のKubernetesクラスターにNetScaler Ingress Controllerを展開する方法は2つあります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- 同じポッド内のNetScaler CPXと並行してサイドカーコンテナとして: このモードでは、NetScaler Ingress ControllerがNetScaler CPXを構成します。
- Kubernetesクラスター内のスタンドアロンポッドとして: このモードでは、クラスターの外部に展開されたNetScaler MPXまたはVPXアプライアンスを制御できます。

NetScaler Ingress Controllerをポッドとして展開する

トピックに記載されている手順に従ってください: NetScaler MPXまたはVPXアプライアンス用のKubernetesクラスターにスタンドアロンポッドとしてNetScaler Ingress Controllerを展開する。

NetScaler CPXでNetScaler Ingress Controllerをサイドカーとして展開する

トピックに記載されている手順に従ってください: NetScaler CPXでNetScaler Ingress Controllerをサイドカーとして展開する。

ネットワーク構成

Kubernetesクラスターに展開されたサービスがシームレスに機能するためには、Ingress NetScalerデバイスがPodが実行されている基盤となるオーバーレイネットワークに到達できることが不可欠です。NetScaler Ingress Controllerを使用すると、静的ルーティング、ノードコントローラー、NodePortタイプのサービス、またはLoadBalancerタイプのサービスを使用して、NetScalerデバイスとサービス間のネットワーク接続を構成できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

デプロイメントソリューション

デプロイメントソリューション:

- NetScaler 統合カナリアデプロイメントソリューション
- NetScaler IPAMコントローラーのデプロイ
- IPAMコントローラーおよびイングレスコントローラーとのBIND DNS統合
- NetScaler GSLBコントローラー
- サービスメッシュライト

NetScaler統合型カナリアデプロイメントソリューションの展開

カナリアリリースとは、新しいソフトウェアバージョンを本番環境に導入する際のリスクを軽減するために、まず変更を少数のユーザーに展開する手法です。ユーザーによる検証後、アプリケーションはより多くのユーザーに展開されます。

NetScalerは、NetScaler Ingress Controllerを使用してカナリアデプロイメントを行うための以下のオプションを提供します。

- Canary CRDを使用したカナリアの展開
- Ingressアノテーションを使用したカナリアの展開

Canary CRDを使用したデプロイメントでは、カナリア構成はKubernetes CRDを使用して適用されます。Citrix®は、Ingressアノテーションを使用したカナリアデプロイメントのよりシンプルなオプションもサポートしています。

Canary CRDを使用したカナリアの展開

このセクションでは、Canary CRDを使用してカナリアデプロイメントを実行する方法について説明します。

NetScaler統合型カナリアデプロイメントソリューションは、継続的デリバリー (CD) のすべてのコンポーネントを連携させ、アプリケーション開発者にとってカナリアデプロイメントをより簡単にします。このソリューションは、継続的デリバリープラットフォームとしてSpinnakerを、カナリア分析用のSpinnakerプラグインとしてKayentaを使用します。Kayentaは、ユーザーが設定し

© 1997–2026 Citrix Systems, Inc. All rights reserved.

たメトリクスをソースから取得し、統計テストを実行して、カナリアの集計スコアを提供するオープンソースのカナリア分析サービスです。統計テストとカウンターからのスコアは、成功基準とともにカナリアを昇格または失敗させるために使用されます。

NetScalerは、豊富なアプリケーション中心の構成モジュールを備えており、アプリケーションのトラフィックとアプリケーションインスタンスの健全性に対する完全な可視性を提供します。NetScalerの正確なパフォーマンス統計を生成する機能は、カナリア分析に活用され、カナリアデプロイメントに関するより良い決定を下すことができます。このソリューションでは、NetScalerはSpinnakerプラットフォームと統合され、Kayentaを使用したカナリアデプロイメント分析のための正確なメトリクスを提供するソースとして機能します。

NetScaler Metrics Exporterは、アプリケーションのパフォーマンスメトリクスをオープンソースの監視システムPrometheusにエクスポートし、Kayentaを設定してカナリアデプロイメント用のメトリクスを取得できます。カナリアバージョンへのトラフィック分散は、NetScalerポリシーインフラストラクチャを使用して調整できます。特定の種類のトラフィックを本番環境からベースラインとカナリアに迂回させたい場合は、豊富なNetScalerポリシーインフラストラクチャを活用して、一致式を使用してトラフィックをベースラインとカナリアにリダイレクトできます。

たとえば、一致式 HTTP.REQ.URL.CONTAINS("citrix india") を使用して、本番環境からカナリアおよびベースラインにトラフィックを迂回させることができます。この式に一致するトラフィックはカナリアおよびベースラインに迂回され、残りのトラフィックは本番環境に送られます。

Citrix統合型カナリアデプロイメントソリューションの一部であるコンポーネントとその機能は次のとおりです。

- GitHub: GitHubは、Gitが提供するすべての分散バージョン管理およびソースコード管理機能を提供し、追加機能も備えています。 GitHubには、Docker HubやSpinnakerなど、CI/CDパイプラインの一部を構成する他のツールと統合するための多くのユーティリティが用意されています。
- Docker Hub: Docker Hubは、Dockerが提供するクラウドベースのリポジトリサービスで、Dockerイメージの共有と検索に使用されます。GitHubとDocker Hubを統合して、GitHubのソースコードからイメージを自動的にビルドし、ビルドされたイメージをDocker Hubにプッシュできます。
- Spinnaker: Spinnakerは、ソフトウェアの変更を高速かつ確実にリリースするためのオープンソースのマルチクラウド継続的デリバリープラットフォームです。Spinnakerのアプリケーションデプロイ機能を使用して、継続的デリバリーワークフローを構築および管理できます。Spinnakerにおける主要なデプロイ管理の構成要素はパイプラインとして知られています。Spinnakerのパイプラインは、ステージとして知られる一連のアクションで構成されています。Spinnakerは、アプリケーションのデプロイ、スクリプトの実行、カナリア分析の実行、

© 1997–2026 Citrix Systems, Inc. All rights reserved.

デプロイの削除など、さまざまなステージを提供します。Spinnakerを多くのサードパーティツールと統合して、多くの追加機能をサポートできます。

- Prometheus: Prometheusは、オープンソースのシステム監視およびアラートツールキットです。Prometheusは、時系列データベースに大量のデータを保持できる監視システムです。NetScaler Metricsは、Prometheusを介してパフォーマンスメトリックをSpinnakerに公開します。
- Jenkins: Jenkinsは、ソフトウェアのビルド、テスト、デリバリー、デプロイに関連するあらゆる種類のタスクを自動化するのに役立つオープンソースの自動化サーバーです。Jenkinsは、デプロイサイクルの一部としてカスタムスクリプトの実行もサポートしています。
- NetScaler Ingress Controller NetScalerは、ベアメタルおよびクラウドデプロイメント向けに、NetScaler MPX（ハードウェア）、NetScaler VPX（仮想化）、およびNetScaler CPX（コンテナ化）用のIngress Controllerを提供します。NetScaler Ingress ControllerはKubernetes Ingressを中心に構築されており、Ingressリソース構成に基づいて1つ以上のNetScalerを自動的に構成します。

Citrix統合カナリアデプロイメントソリューションには、以下のNetScaler®ソフトウェアバージョンが必要です。

- ネットスケーラー イングレス コントローラーのビルド/バージョン: `quay.io/citrix/citrix-k8s-ingress-controller:1.29.5`。
- NetScaler CPXバージョン: `quay.io/citrix/citrix-k8s-cpx-ingress:13.0-83.27`。
- NetScaler Metrics Exporterバージョン: `quay.io/citrix/netscaler-metrics-exporter:1.4.0`。

NetScaler統合カナリアデプロイメントソリューションのためのSpinnakerパイプラインのワークフロー

次の図は、NetScaler統合カナリアデプロイメントソリューションのためのSpinnakerパイプラインのワークフローを示しています。

次の手順は、図に示されているワークフローを説明しています。

1. 開発者はGitHubでソースコードを管理し、必要に応じて変更を加え、その変更をGitHubにコミットします。
2. ソースコードの変更をリッスンするために、GitHubにWebhookが構成されています。ソースコードがGitHubにチェックインされるたびに、Webhookがトリガーされ、Docker Hubに新しいソースコードでイメージをビルドするように通知します。Dockerイメージが作成されると、Docker Hubに構成された別のWebhookがSpinnakerパイプラインをトリガーします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

3. Spinnakerパイプラインがトリガーされると、イメージのカナリアバージョンとベースラインバージョンがデプロイされます。
4. カナリアバージョンとベースラインバージョンがデプロイされると、本番環境からのトラフィックの一部がカナリアバージョンとベースラインバージョンに転送されます。NetScalerはパフォーマンス統計を収集し、NetScaler Metrics Exporterの助けを借りてPrometheusに統計をエクスポートします。Prometheusはこれらの統計をKayentaに提供し、カナリア分析を行います。
5. Kayentaはパフォーマンス統計に基づいてカナリア分析を実行し、スコアを生成します。このスコアに基づいて、カナリアデプロイメントは成功または失敗と判断され、イメージはロールアウトまたはロールバックされます。

Google Cloud PlatformでNetScaler統合カナリアデプロイメントソリューションをデプロイする

このセクションには、Spinnakerのセットアップ方法、Spinnakerパイプラインの作成方法、およびカナリアデプロイメントのサンプルに関する情報が含まれています。

Google Cloud PlatformでSpinnakerをデプロイする

このトピックには、Spinnakerのデプロイ方法と、Google Cloud Platform (GCP) でのカナリアデプロイメントのためにプラグインをSpinnakerと統合する方法に関する情報が含まれています。

GCPでSpinnakerをデプロイし、プラグインを統合するには、以下の手順を実行します。

1. 以下のコマンドを使用して、環境をセットアップし、GKEクラスターを作成します。

```
export GOOGLE_CLOUD_PROJECT=[PROJECT_ID]

gcloud config set project $GOOGLE_CLOUD_PROJECT

gcloud config set compute/zone us-central1-f

gcloud services enable container.googleapis.com

gcloud beta container clusters create kayenta-tutorial

--machine-type=n1-standard-2 --enable-stackdriver-kubernetes
```

2. 以下のコマンドを使用して、PrometheusをStackdriverと統合するためのプラグインをインストールします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl apply --as=admin --as-group=system:masters -f \

https://storage.googleapis.com/stackdriver-prometheus-documentation/
rbac-setup.yml

curl -sS "https://storage.googleapis.com/stackdriver-prometheus-
documentation/prometheus-service.yml" |

\sed "s/_stackdriver_project_id:.*/_stackdriver_project_id:
$GOOGLE_CLOUD_PROJECT/" |

\sed "s/_kubernetes_cluster_name:.*/_kubernetes_cluster_name: kayenta-
tutorial/" |

\sed "s/_kubernetes_location:.*/_kubernetes_location: us-central1-f/" |

\kubectl apply -f -
```

3. 以下の手順を使用して、GKEクラスターにSpinnakerをデプロイします。

   a. Spinnakerウェブサイトから Spinnaker用の `quick-install.yml` ファイルをダウンロードします。

   b. Docker Hubから始まるさまざまなコンポーネントを統合するために、`quick-install.yml` ファイルを更新します。SpinnakerをDocker Hubと統合するには、`quick-install.yml` ファイル内のConfigMapの下にあるアドレス、ユーザー名、パスワード、メール、およびリポジトリの値を更新します。

```
dockerRegistry:

            enabled: true

            accounts:

            - name: my-docker-registry

            requiredGroupMembership: []

            providerVersion: V1

            permissions: {}
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            address: https://index.docker.io

            username: &lt;username>

            password: &lt;password>

            email: &lt;mail-id>

            cacheIntervalSeconds: 30

            clientTimeoutMillis: 60000

            cacheThreads: 1

            paginateSize: 100

            sortTagsByDate: false

            trackDigests: false

            insecureRegistry: false

            repositories:- &lt;repository-name>

            primaryAccount: my-docker-registry
```

c. (オプション) Jenkinsをセットアップするには、以下の手順を実行します。

```
sudo apt-get update

sudo apt-get upgrade

sudo apt-get install openjdk-8-jdk

wget -q -O - https://jenkins-ci.org/debian/jenkins-ci.org.key |

sudo apt-key add -

sudo sh -c 'echo deb http://pkg.jenkins-ci.org/debian binary/ > /
etc/apt/sources.list.d/jenkins.list'

sudo apt-get update
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
sudo apt-get install jenkins git

sudo apt-get install software-properties-common python-software-
properties apt-transport-https

sudo add-apt-repository https://dl.bintray.com/spinnaker-releases/
debians
```

注:

JenkinsがKubernetesのノードのいずれかにインストールされている場合、パブリックアクセス用にそのノードのファイアウォールルールを更新する必要があります。

d. JenkinsをSpinnakerと統合するために、`quick-install.yml`ファイルで以下の値を更新します。

```
    data:igor.yml: |

enabled: true

skipLifeCycleManagement: false

    ci:jenkins:

enabled: true

  masters:

- name: master

      address: &lt;endpoint>

    username: &lt;username>

  password: &lt;password>
```

e. PrometheusとGrafanaをセットアップするには、NetScaler Metrics Exporterの「PrometheusとGrafanaの統合」セクションを参照し、手順を実行します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

f. PrometheusをSpinnakerと統合するには、`quick-install.yml`ファイルで以下の値を更新します。

```
  data:
  config: |
    deploymentConfigurations:
    canary:
    enabled: true
serviceIntegrations:
      - name: prometheus
  enabled: true
accounts:
  - name: my-prometheus
endpoint:
      baseUrl: prometheus-endpoint
    supportedTypes:
- METRICS_STORE
  data:
  config: |
    deploymentConfigurations:
    metricStores:
      prometheus:
      enabled: true
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      add_source_metalabels: true

      stackdriver:

  enabled: true

    period: 30

    enabled: true
```

g. 通知のためにSlackをSpinnakerと統合するには、`quick-install.yml`ファイルで以下の値を更新します。

```
data:

 config: |

  deploymentConfigurations:

   notifications:

    slack:

     enabled: true

      botName: &lt;BotName>

      token: &lt;token>
```

h. 必要なすべてのコンポーネントが統合されたら、次の手順を実行してSpinnakerをデプロイします。

```
kubectl apply -f quick-install.yaml
```

i. 次のコマンドを使用してデプロイの進行状況を確認します。デプロイが完了すると、このコマンドはすべてのポッドをReady x/xとして出力します。

```
watch kubectl -n spinnaker get pods
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

4. Spinnakerをデプロイしたら、次の手順を使用してデプロイをテストできます。

   a. 次のコマンドを使用して、ローカルポートをSpinnakerのdeckコンポーネントに転送することで、Spinnakerへのアクセスを有効にします。

   ```
   DECK_POD=$(kubectl -n spinnaker get pods -l \

   cluster=spin-deck,app=spin \

   -o=jsonpath='{.items[0].metadata.name}')

   kubectl -n spinnaker port-forward $DECK_POD 8080:9000 >/dev/null &
   ```

   b. Spinnakerにアクセスするには、Cloud Shellで**Web Previewアイコン**をクリックし、**Preview on port 8080**を選択します。

   注:

   Spinnakerには安全に、またはHTTP経由でアクセスできます。Spinnakerを安全に公開するには、spin-ingress-ssl.yamlファイルを使用してIngressをデプロイします。Spinnakerアプリケーションが公開されたら、Spinnakerに割り当てられたドメインまたはIngressのIPアドレスを使用してアクセスできます。

Spinnakerパイプラインを作成し、自動カナリアデプロイメントを構成する

Spinnakerをデプロイしたら、アプリケーション用のSpinnakerパイプラインを作成し、自動カナリアデプロイメントを構成します。

1. Spinnakerでアプリケーションを作成します。
2. Spinnakerパイプラインを作成します。Sample JSON filesで提供されているサンプルファイルを使用して、パイプラインをJSONファイルとして編集できます。
3. Spinnakerで自動カナリア分析のための自動カナリア設定を作成します。JSONファイルで提供されている設定を、自動カナリア設定のサンプルとして使用できますサンプルJSONファイル。

カナリア用のサンプルアプリケーションをデプロイする

この例では、NetScaler統合カナリアデプロイメントソリューションを使用して、サンプルアプリケーションのカナリアデプロイメントを実行する方法を示します。この例では、NetScaler CPX、

© 1997–2026 Citrix Systems, Inc. All rights reserved.

MPX、またはVPXがGKEクラスターのIngressデバイスとしてデプロイされます。NetScalerは、カナリア分析に必要なパフォーマンスメトリックを生成します。

前提条件として、サンプルアプリケーションをデプロイする前に、次の手順を完了する必要があります。

- Google Cloud PlatformにSpinnakerをデプロイするを使用して、Google Cloud PlatformにSpinnakerと必要なプラグインをインストールします。

サンプルアプリケーションをデプロイする

カナリアリリースとしてサンプルアプリケーションをデプロイするには、次の手順を実行します。

1. rbac.yamlファイルをデプロイして、NetScalerに必要なRBACルールを作成します。

   ```
   kubectl apply -f rbac.yaml
   ```

2. NetScaler Ingress Controllerは、NetScaler CPXのサイドカーとしてデプロイすることも、NetScaler VPXまたはMPXを制御するスタンドアロンポッドとしてデプロイすることもできます。

   cpx-with-cic-sidecar.ymlファイルを使用して、NetScaler Ingress ControllerをNetScaler CPXのサイドカーとしてデプロイします。これにより、NetScaler Metrics Exporterも同じポッドにデプロイされます。

   ```
   kubectl apply -f cpx-with-cic-sidecar.yml
   ```

   NetScaler Ingress ControllerをNetScaler VPXまたはMPX用のスタンドアロンポッドとしてデプロイするには、cic-vpx.yamlファイルを使用します。このデプロイでは、NetScaler Metrics Exporterをデプロイするためにexporter.yamlファイルを使用する必要があります。

   ```
   kubectl apply -f cic-vpx.yaml

   kubectl apply -f exporter.yaml
   ```

   注:

   NetScaler Ingress Controllerのデプロイ方法に応じて、NetScaler Ingress Controllerデプロイメント用のYAMLファイルを編集し、NetScaler Ingress Controllerのデプロイに記載されている環境変数の値を変更する必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

3. spin-ingress-ssl.yamlファイルを使用して、Spinnakerを安全に公開するためのIngressをデプロイします。

```
kubectl apply -f spin-ingress-ssl.yaml
```

注:

Ingress 用の TLS 証明書の作成に関する詳細については、NetScaler Ingress Controller の TLS 証明書 を参照してください。

4. NetScaler を使用して Spinnaker が公開されたら、Spinnaker にアクセスし、まだ手順が完了していない場合は、Spinnaker パイプラインを作成し、自動カナリアデプロイメントを構成する の手順を実行します。
5. production.yaml ファイルを使用して、アプリケーションのプロダクションバージョンをデプロイします。

```
kubectl apply -f production.yaml
```

6. ingress.yaml ファイルを使用して、クラスター外からのトラフィックをクラスター内のサービスに公開する Ingress リソースルールを作成します。

```
kubectl apply -f ingress.yaml
```

7. service.yaml ファイルを使用して、カナリアデプロイメントが必要なアプリケーションの Kubernetes サービスを作成します。

```
kubectl apply -f service.yaml
```

8. canary-crd-class.yaml ファイルを使用して、カナリア構成を定義するカナリア CRD をデプロイします。

```
kubectl apply -f canary-crd-class.yaml
```

注:

CRD を作成したら、CRD オブジェクトを適用する前に 10 秒間待機します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

9. カナリア構成をカスタマイズするために、カナリア CRD に基づいて CRD オブジェクト canary-crd-object.yaml を作成します。

```
kubectl apply -f canary-crd-object.yaml
```

次の表は、カナリア CRD オブジェクトのフィールドについて説明しています。

フィールド	説明
serviceNames	この CRD を適用する必要があるサービスの一覧
デプロイメント	デプロイ戦略をKayentaとして指定します。
パーセンテージ	プロダクションからベースラインおよびカナリアに転送されるトラフィックの割合を指定します。
matchExpression (オプション)	カナリアバージョンとベースラインバージョンに誘導されるユーザーのサブセットを定義するために使用できる、NetScalerがサポートする任意のポリシー。トラフィックのx パーセンテージが設定されている場合、matchExpressionに一致するユーザーのサブセット内から、xパーセンテージのユーザーのみがベースラインとカナリアに転送されます。残りのユーザーはプロダクションに転送されます。
スピナッカー	サービスに適用するSpinnakerパイプライン構成を指定します。
ドメイン	SpinnakerゲートのIPアドレスまたはドメイン名。
ポート	Spinnakerゲートのポート番号。
applicationName	Spinnaker内のアプリケーション名。
pipelineName	Spinnakerアプリケーション下のパイプライン名。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明
serviceName	Spinnaker構成を適用するサービスの名前を指定します。

10. アプリケーションのカナリアバージョンとベースラインバージョンをデプロイします。

> **注:**
>
> カナリアデプロイメントを完全に自動化している場合は、Spinnakerパイプラインのデプロイ (マニフェスト) ステージを使用してカナリアバージョンとベースラインバージョンをデプロイするため、この手順を実行する必要はありません。

カナリアバージョンとベースラインバージョンを手動でデプロイするには、canary.yamlとbaseline.yamlファイルを使用します。

```
kubectl apply -f canary.yaml

kubectl apply -f baseline.yaml
```

トラブルシューティング

デプロイメントのトラブルシューティングを行うには、次の手順を実行します。

1. Spinnaker、Prometheus、Kayenta、NetScaler CPX、NetScaler Metrics Exporter、NetScaler Ingress Controllerなどの各コンポーネントのPodログを確認します。
2. NetScalerプロキシの構成中に、NetScaler Ingress ControllerのPodログで構成関連のエラーがないか確認します。
3. 問題を絞り込むには、NetScaler Ingress ControllerのPodログで`exception/Exception`キーワードを検索します。
4. 検索より前のログを確認します。失敗して問題を引き起こした構成を確認します。
5. 構成中の失敗の原因を確認します。
6. 誤った構成が原因で障害が発生した場合は、構成を修正します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サンプルJSONファイル

このトピックには、Spinnakerパイプライン構成と自動カナリア構成のサンプルJSONファイルが含まれています。これらのファイルは、Spinnakerパイプラインと自動カナリア構成を作成する際の参照として使用できます。

Spinnakerパイプライン構成用のサンプルJSONファイル**

```
{

  "appConfig": {},

  "description": "This pipeline deploys a canary version of the
application, and a baseline (identical to production) version.\nIt
compares them, and if the canary is OK, it triggers the production
deployment pipeline.",

  "executionEngine": "v2",

  "expectedArtifacts": [

    {

      "defaultArtifact": {

        "kind": "custom"

      },

      "id": "ac842617-988f-48dc-a7a4-7f020d93cc42",

      "matchArtifact": {

        "kind": "docker",

        "name": "index.docker.io/sample/demo",

        "type": "docker/image"

      },

      "useDefaultArtifact": false,

      "usePriorExecution": false
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    }
  ],
  "keepWaitingPipelines": false,
  "lastModifiedBy": "anonymous",
  "limitConcurrent": true,
  "parallel": true,
  "parameterConfig": [],
  "stages": [
    {
      "account": "my-kubernetes-account",
      "cloudProvider": "kubernetes",
      "kinds": [
        "Deployment",
        "ConfigMap"
      ],
      "labelSelectors": {
        "selectors": [
          {
            "key": "version",
            "kind": "EQUALS",
            "values": [
              "canary"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            ]
          }
        ]
      },
      "location": "default",
      "name": "Delete Canary",
      "options": {
        "cascading": true
      },
      "refId": "12",
      "requisiteStageRefIds": [
        "19",
        "26"
      ],
      "type": "deleteManifest"
    },
    {
      "account": "my-kubernetes-account",
      "cloudProvider": "kubernetes",
      "kinds": [
        "Deployment"
      ],
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      "labelSelectors": {
        "selectors": [
          {
            "key": "version",
            "kind": "EQUALS",
            "values": [
              "baseline"
            ]
          }
        ]
      },
      "location": "default",
      "name": "Delete Baseline",
      "options": {
        "cascading": true
      },
      "refId": "13",
      "requisiteStageRefIds": [
        "19",
        "26"
      ],
      "type": "deleteManifest"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    },
    {
      "name": "Successful deployment",
      "preconditions": [],
      "refId": "14",
      "requisiteStageRefIds": [
        "12",
        "13"
      ],
      "type": "checkPreconditions"
    },
    {
      "application": "sampleapplicaion",
      "expectedArtifacts": [
        {
          "defaultArtifact": {
            "kind": "custom"
          },
          "id": "9185c756-c6cd-49bc-beee-e3f7118f3412",
          "matchArtifact": {
            "kind": "docker",
            "name": "index.docker.io/sample/demo",
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            "type": "docker/image"
          },
          "useDefaultArtifact": false,
          "usePriorExecution": false
        }
      ],
      "failPipeline": true,
      "name": "Deploy to Production",
      "pipeline": "7048e5ac-2464-4557-a05a-bec8bdf868fc",
      "refId": "19",
      "requisiteStageRefIds": [
        "25"
      ],
      "stageEnabled": {
        "expression": "\"${ #stage('Canary Analysis')['status'].toString()
== 'SUCCEEDED'}\"",
        "type": "expression"
      },
      "type": "pipeline",
      "waitForCompletion": true
    },
    {
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
"account": "my-kubernetes-account",
"cloudProvider": "kubernetes",
"manifestArtifactAccount": "embedded-artifact",
"manifests": [
  {
    "apiVersion": "apps/v1",
    "kind": "Deployment",
    "metadata": {
      "labels": {
        "name": "sampleapplicaion-prod",
        "version": "baseline"
      },
      "name": "sampleapplicaion-baseline-deployment",
      "namespace": "default"
    },
    "spec": {
      "replicas": 4,
      "strategy": {
        "rollingUpdate": {
          "maxSurge": 10,
          "maxUnavailable": 10
        },
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            "type": "RollingUpdate"
          },
          "template": {
            "metadata": {
              "labels": {
                "name": "sampleapplicaion-prod"
              }
            },
            "spec": {
              "containers": [
                {
                  "image": "index.docker.io/sample/demo:v1",
                  "imagePullPolicy": "Always",
                  "name": "sampleapplicaion-prod",
                  "ports": [
                    {
                      "containerPort": 8080,
                      "name": "port-8080"
                    }
                  ]
                }
              ]
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            }
          }
        }
      }
    ],
    "moniker": {
      "app": "sampleapplicaion"
    },
    "name": "Deploy Baseline",
    "refId": "20",
    "relationships": {
      "loadBalancers": [],
      "securityGroups": []
    },
    "requisiteStageRefIds": [],
    "source": "text",
    "type": "deployManifest"
  },
  {
    "account": "my-kubernetes-account",
    "cloudProvider": "kubernetes",
    "manifestArtifactAccount": "embedded-artifact",
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
"manifests": [
  {
    "apiVersion": "apps/v1",
    "kind": "Deployment",
    "metadata": {
      "labels": {
        "name": "sampleapplicaion-prod",
        "version": "canary"
      },
      "name": "sampleapplicaion-canary-deployment",
      "namespace": "default"
    },
    "spec": {
      "replicas": 4,
      "strategy": {
        "rollingUpdate": {
          "maxSurge": 10,
          "maxUnavailable": 10
        },
        "type": "RollingUpdate"
      },
      "template": {
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            "metadata": {
              "labels": {
                "name": "sampleapplicaion-prod"
              }
            },
            "spec": {
              "containers": [
                {
                  "image": "index.docker.io/sample/demo",
                  "imagePullPolicy": "Always",
                  "name": "sampleapplicaion-prod",
                  "ports": [
                    {
                      "containerPort": 8080,
                      "name": "port-8080"
                    }
                  ]
                }
              ]
            }
          }
        }
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        }
      ],
      "moniker": {
        "app": "sampleapplicaion"
      },
      "name": "Deploy Canary",
      "refId": "21",
      "relationships": {
        "loadBalancers": [],
        "securityGroups": []
      },
      "requiredArtifactIds": [
        "ac842617-988f-48dc-a7a4-7f020d93cc42"
      ],
      "requisiteStageRefIds": [],
      "source": "text",
      "type": "deployManifest"
    },
    {
      "analysisType": "realTime",
      "canaryConfig": {
        "beginCanaryAnalysisAfterMins": "2",
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        "canaryAnalysisIntervalMins": "",
        "canaryConfigId": "7bdb4ab4-f933-4a41-865f-6d3e9c786351",
        "combinedCanaryResultStrategy": "LOWEST",
        "lifetimeDuration": "PT0H5M",
        "metricsAccountName": "my-prometheus",
        "scopes": [
          {
            "controlLocation": "default",
            "controlScope": "k8s-sampleapplicaion.default.80.k8s-
sampleapplicaion.default.8080.svc-baseline",
            "experimentLocation": "default",
            "experimentScope": "k8s-sampleapplicaion.default.80.k8s-
sampleapplicaion.default.8080.svc-canary",
            "extendedScopeParams": {},
            "scopeName": "default"
          }
        ],
        "scoreThresholds": {
          "marginal": "0",
          "pass": "70"
        },
        "storageAccountName": "kayenta-minio"
      },
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      "name": "Canary Analysis",
      "refId": "25",
      "requisiteStageRefIds": [
        "20",
        "21"
      ],
      "type": "kayentaCanary"
    },
    {
      "continuePipeline": false,
      "failPipeline": true,
      "job": "NJob",
      "master": "master",
      "name": "Auto Cleanup:  GCR Image and code revert",
      "parameters": {},
      "refId": "26",
      "requisiteStageRefIds": [
        "25"
      ],
      "stageEnabled": {
        "type": "expression"
      },
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      "type": "jenkins"
    }
  ],
  "triggers": [
    {
      "account": "my-docker-registry",
      "enabled": true,
      "expectedArtifactIds": [
        "ac842617-988f-48dc-a7a4-7f020d93cc42"
      ],
      "organization": "sample",
      "payloadConstraints": {},
      "registry": "index.docker.io",
      "repository": "sample/demo",
      "source": "dockerhub",
      "type": "webhook"
    }
  ],
  "updateTs": "1553144362000"
}
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

自動カナリア構成のサンプルJSONファイル

以下は、自動カナリア構成のサンプルJSONファイルです。

```
{
  "applications": [
    "sampleapplicaion"
  ],
  "classifier": {
    "groupWeights": {
      "Group 1": 70,
      "Group 2": 30
    },
    "scoreThresholds": {
      "marginal": 75,
      "pass": 95
    }
  },
  "configVersion": "1",
  "createdTimestamp": 1552650414234,
  "createdTimestampIso": "2019-03-15T11:46:54.234Z",
  "description": "Canary Config",
  "judge": {
    "judgeConfigurations": {},
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    "name": "NetflixACAJudge-v1.0"
  },
  "metrics": [
    {
      "analysisConfigurations": {
        "canary": {
          "direction": "increase"
        }
      },
      "groups": [
        "Group 1"
      ],
      "name": "Server Response Errors - 5XX",
      "query": {
        "customFilterTemplate": "tot_requests",
        "metricName": "netscaler_lb_vserver_svr_busy_err_rate",
        "serviceType": "prometheus",
        "type": "prometheus"
      },
      "scopeName": "default"
    },
    {
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    "analysisConfigurations": {
      "canary": {
        "direction": "either",
        "nanStrategy": "replace"
      }
    },
    "groups": [
      "Group 2"
    ],
    "name": "Server Response Latency - TTFB",
    "query": {
      "customFilterTemplate": "ttfb",
      "metricName": "netscaler_lb_vserver_hits_total",
      "serviceType": "prometheus",
      "type": "prometheus"
    },
    "scopeName": "default"
  }
],
"name": "canary-config",
"templates": {
  "tot_requests": "lb_vserver_name = \"${scope}\"",
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    "ttfb": "lb_vserver_name = \"${scope}\""
  },
  "updatedTimestamp": 1553098513495,
  "updatedTimestampIso": "2019-03-20T16:15:13.495Z"
}
```

Ingressアノテーションを使用した簡素化されたカナリアデプロイ

このトピックでは、Ingressアノテーションを使用した簡素化されたカナリアデプロイメントに関する情報を提供します。NetScalerはカナリアデプロイメントをサポートするための複数のオプションを提供していますが、これはよりシンプルなタイプのカナリアデプロイメントです。

Ingressアノテーションを使用したカナリアは、ルールベースのカナリアデプロイメントです。このアプローチでは、アプリケーションリクエストがルールベースのカナリアデプロイメント戦略に基づいて処理される必要があることを示すために、特定の注釈を持つ追加のIngressオブジェクトを定義する必要があります。Citrixソリューションでは、Ingressレベルでのカナリアベースのトラフィックルーティングは、次のようにさまざまなルールセットを定義することで実現できます。

- 重みに基づいてカナリアルールを適用する
- HTTPリクエストヘッダーに基づいてカナリアルールを適用する
- HTTPヘッダー値に基づいてカナリアルールを適用する

カナリアルールの優先順位は次のとおりです。

HTTPリクエストヘッダー値によるカナリア --> HTTPリクエストヘッダーによるカナリア --> 重みによるカナリア

重みに基づくカナリアデプロイ

重みベースのカナリアデプロイメントは、広く使用されているカナリアデプロイメントのアプローチです。このアプローチでは、カナリアバージョンとアプリケーションのプロダクションバージョンに振り分けられるトラフィックの割合を決定する重みを0から100の範囲で設定できます。

重みベースのカナリアデプロイメントのワークフローは次のとおりです。

- 最初は重みをゼロに設定できます。これは、トラフィックがカナリアバージョンに転送されないことを示します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- カナリアデプロイメントを開始することを決定したら、トラフィックがカナリアバージョンにも向けられるように、重みを必要なパーセンテージに変更します。
- 最後に、カナリアバージョンがリリース準備完了であると判断したら、すべてのトラフィックがカナリアバージョンに転送されるように、重みを100に変更します。

NetScaler Ingress Controller を使用して重みベースのカナリアをデプロイするには、カナリアアノテーション `ingress.citrix.com/canary-weight:` を持つ新しい Ingress を作成し、カナリアバージョンに転送するトラフィックの割合を指定します。

HTTPリクエストヘッダーに基づくカナリアデプロイメント

クライアントによって制御されるHTTPリクエストヘッダーに基づいてカナリアデプロイメントを構成できます。リクエストヘッダーは、カナリアIngressで指定されたサービスにリクエストをルーティングするようにIngressに通知します。リクエストヘッダーにIngressアノテーション `ingress.citrix.com/canary-by-header:` で指定された値が含まれている場合、リクエストはカナリアIngressで指定されたサービスにルーティングされます。

HTTPリクエストヘッダー値に基づくカナリアデプロイメント

ヘッダーによるカナリアの拡張であるHTTPリクエストヘッダーの値に基づいてカナリアデプロイメントを構成することもできます。このデプロイメントでは、`ingress.citrix.com/canary-by-header:` アノテーションに加えて、`ingress.citrix.com/canary-by-header-value:` アノテーションも指定します。リクエストヘッダー値がIngressアノテーション `ingress.citrix.com/canary-by-header-value:` で指定された値と一致する場合、リクエストはカナリアIngressで指定されたサービスにルーティングされます。複数のヘッダー値を文字列のリストとして指定できます。

HTTPリクエストヘッダー値に基づくカナリアデプロイメントのサンプルアノテーションは次のとおりです。

ingress.citrix.com/canary-by-header-value: ''

Ingressアノテーションを使用したカナリアデプロイメントの構成

サンプルアプリケーションをカナリアリリースとしてデプロイするには、次の手順を実行します。

1. NetScaler Ingress Controller をデプロイする の手順に従って、NetScaler Ingress Controller をデプロイします。NetScaler Ingress Controller は、NetScaler CPX とともにサイドカーとしてデプロイすることも、NetScaler VPX または MPX を制御するスタンドアロンポッドとしてデプロイすることもできます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

2. guestbook-deploy.yaml ファイルを使用して、`Guestbook` アプリケーションをデプロイします。

```
kubectl apply -f guestbook-deploy.yaml
```

3. guestbook-service.yaml ファイルを使用して、`Guestbook` アプリケーションを公開するサービスをデプロイします。

```
kubectl apply -f guestbook-service.yaml
```

4. `Guestbook`アプリケーション用のIngressオブジェクトをguestbook-ingress.yamlファイルを使用してデプロイします。

```
kubectl apply -f  guestbook-ingress.yaml
```

5. `Guestbook`アプリケーションのカナリアバージョンをcanary-deployment.yamlファイルを使用してデプロイします。

```
kubectl apply –f canary-deployment.yaml
```

6. `Guestbook`アプリケーションのカナリアバージョンを公開するサービスをcanary-service.yamlファイルを使用してデプロイします。

```
kubectl apply –f canary-service.yaml
```

7. `Guestbook`アプリケーションのカナリアバージョン用のアノテーション付きIngressオブジェクトをcanary-ingress.yamlファイルを使用してデプロイします。

```
    kubectl apply –f canary-ingress.yaml









      apiVersion: networking.k8s.io/v1
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kind: Ingress

metadata:

  annotations:

      ingress.citrix.com/canary-weight: "10"

  name: canary-by-weight

spec:

  ingressClassName: citrix

  rules:

  - host: webapp.com

    http:

      paths:

      - backend:

          service:

            name: guestbook-canary

            port:

              number: 80

        path: /

        pathType: Prefix
```

ここで、アノテーション`ingress.citrix.com/canary-weight: “10”`は、重みベースのカナリア用のアノテーションです。このアノテーションは、`webapp.com`宛ての総リクエストの10パーセントが`guestbook-canary`サービスに送信されるようにNetScalerを構成するようNetScaler Ingress Controllerに指定します。これは、`Guestbook`アプリケーションのカナリアバージョン用のサービスです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Ingress Controllerを使用してHTTPヘッダーベースのカナリアをデプロイするには、`canary-ingress.yaml`ファイル内のカナリアアノテーション`ingress.citrix.com/canary-weight:`を`ingress.citrix.com/canary-by-header:`アノテーションに置き換えます。

NetScaler Ingress Controllerを使用してHTTPヘッダー値ベースのカナリアをデプロイするには、`canary-ingress.yaml`ファイル内の`ingress.citrix.com/canary-weight:`アノテーションを`ingress.citrix.com/canary-by-header:`と`ingress.citrix.com/canary-by-header-value:`アノテーションに置き換えます。

注:

ヘッダーベースのカナリアとヘッダー値ベースのカナリアを実現するためのCanary example YAMLsを参照してください。

NetScaler IPAMコントローラーをデプロイする

NetScalerは、IPアドレス管理用のIPAMコントローラーを提供します。NetScaler IPAMコントローラーは、Kubernetesクラスター内でNetScaler Ingress Controllerと並行して動作します。NetScaler IPAMコントローラーは、指定されたIPアドレス範囲から、LoadBalancerタイプのサービスおよびイングレスリソースにIPアドレスを割り当てます。

NetScaler IPAMコントローラーには、NetScalerのVIPカスタムリソース定義 (CRD) が必要です。VIP CRDは、NetScaler Ingress ControllerとNetScaler IPAMコントローラー間の内部通信に使用されます。

前提条件

- Kubernetesクラスターと、クラスターと通信するためのkubectlコマンドラインツール。
- リソースを分離するために、`netscaler`という名前空間を作成します。名前空間を作成するには、次のコマンドを実行します。

  ```
  kubectl create namespace netscaler
  ```

- 次のHelmコマンドを使用して、NetScaler VPXまたはNetScaler MPX用にNetScaler Ingress Controllerをインストールします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

次のコマンドを実行する前に、NetScaler VPXまたはNetScaler MPXの資格情報を使用してシークレットを作成してください。

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/



helm install NetScaler-ingress-controller netscaler/NetScaler-ingress-controller --set nsIP=<NSIP of MPX/VPX>,license.accept=yes,adcCredentialSecret=<Secret-for-ADC-credentials>,ingressClass[0]=netscaler,serviceClass[0]=netscaler,ipam=true,crds.install=true -n netscaler
```

Helmチャートを使用してNetScaler Ingress Controllerをデプロイおよび構成する方法の詳細については、Helmチャートリポジトリを参照してください。

IPAMコントローラーをデプロイする

1. 次のコマンドを使用して、NetScaler Helmチャートリポジトリをローカルレジストリに追加します。

   ```
   helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/
   ```

2. 次のコマンドを使用して、NetScaler IPAMコントローラーをインストールします。

   ```
   helm install netscaler-ipam-controller netscaler/netscaler-ipam-controller --set vipRange='[{"<VIP-range-key>": ["<ip-range>"]}]' -n netscaler
   ```

Helmチャートを使用してIPAMコントローラーをインストールする際のすべての構成可能なパラメーターに関する情報については、Helmチャートリポジトリを参照してください。

IPアドレスの割り当て

- LoadBalancerタイプのサービスの場合、VIP範囲から各サービスに一意のIPアドレスが割り当てられます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- イングレスリソースの場合、指定されたIP範囲内のIPアドレスが割り当てられます。複数のイングレスリソースが同じVIP範囲を参照する場合、最初のイングレスリソースに割り当てられたIPアドレスが他のすべてのイングレスリソースに割り当てられます。
- LoadBalancerタイプのサービスとイングレスリソースの両方が、同時にNetScaler IPAMコントローラーをIPアドレスの割り当てに使用できます。いずれかのリソースタイプにIPアドレスが割り当てられている場合、他のリソースタイプでは利用できません。ただし、同じIPアドレスを複数のイングレスリソースに割り当てることができます。

IPAMコントローラーにおける環境変数

このセクションでは、NetScaler IPAMコントローラーの環境変数に関する情報を提供します。

VIP_RANGE

`VIP_RANGE` 環境変数を使用すると、IPアドレス範囲を定義できます。IPアドレス範囲を定義するか、一意の名前に関連付けられたIPアドレス範囲を定義することができます。

IPアドレス範囲

サブネットまたは複数のサブネットからIPアドレス範囲を定義できます。また、`-` 文字を使用してIPアドレス範囲を定義することもできます。IPAMコントローラーは、このIPアドレス範囲からサービスにIPアドレスを割り当てます。

一意の名前に関連付けられたIPアドレス範囲

IPアドレス範囲に一意の名前を割り当て、`VIP_RANGE` 環境変数でその範囲を定義できます。このIPアドレス範囲の割り当て方法により、IPアドレス範囲を区別できます。`LoadBalancer` タイプのサービスを作成する際、サービス定義で `service.citrix.com/ipam-range` アノテーションを使用して、IPアドレス割り当てに使用するIPアドレス範囲を指定できます。

reuseIngressVip

`reuseIngressVip` 変数を使用すると、IPAMコントローラーを設定して、同じVIP範囲を参照する各イングレスリソースに同じIPアドレスまたは異なるIPアドレスのいずれかを割り当てることができます。

REUSEINGRESSVIP の値	説明
true (デフォルト)	各イングレスリソースに同じIPアドレスを割り当てる
偽	各イングレスリソースに異なるIPアドレスを割り当てる

© 1997–2026 Citrix Systems, Inc. All rights reserved.

参照

- IPAMコントローラーによって割り当てられたIPアドレスを持つLoadBalancerタイプのサービスを公開する方法については、このセクションを参照してください。

IPAMコントローラーおよびイングレスコントローラーとのBIND DNS統合

BIND 9は、DNSプロトコルの完全な実装です。BIND 9は、権威ネームサーバー、リゾルバー、およびサポートされているホストではスタブリゾルバーとして構成できます。詳細については、Bind 9を参照してください。

NetScaler Ingress Controllerは、`ipam-range`アノテーションを使用して、Ingressおよび`LoadBalancer`タイプのサービスにIPアドレスを割り当てることをサポートしています。IPAMコントローラーは、NetScaler Ingress Controllerとの内部通信にVIP CustomResourceDefinition (CRD)を使用します。VIP CRDは、IPAMコントローラーのHelmインストールの一部として自動的にインストールされ、現在、VIPに関連付けられた`hostname`フィールドをサポートしています。

前提条件

1. IPAMをtrueとして有効にし、サービスクラスとイングレスクラスを設定したhelm値を使用してイングレスコントローラーをデプロイします。次の例では、helmチャートを使用して、NetScaler Ingress Controllerを目的の構成でデプロイしています。

   ```
   helm install nsic netscaler/netscaler-ingress-controller --set nsIP=

   <NSIP>,license.accept=yes,adcCredentialSecret=nslogin,ingressClass[0]=c
   ic-vpx,serviceClass[0]=cic-vpx,ipam=true,entityPrefix=demo
   ```

2. ドキュメントTSIGに従って、TSIG (Transaction SIGnatures) を設定し、binddnsでアクセス制御を提供します。TSIG KEYとTSIG Secretは後で使用するために控えておいてください。

BIND DNSを使用したIPAMコントローラーのデプロイ

BIND DNSを有効にするには、以下の構成でIPAMコントローラーをデプロイします。

1. KubernetesシークレットとしてBIND DNS TSIG KEYとTSIG SECRETを作成します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl create secret generic binddns-tsig-secret --from-
literal=tsigKey=<TSIG Key Name> --from-literal=tsigSecret=<TSIG Secret>
-n <namespace>
```

2. 次のコマンドでIPAMコントローラーをインストールします。

```
helm install my-release netscaler/netscaler-ipam-controller \

  --set vipRange=<IP_RANGE> \

  --set dns.enabled=true \

  --set dns.type=binddns \

  --set dns.binddns.server=<BIND_SERVER_IP> \

  --set dns.binddns.zone=example.com \

  --set dns.binddns.tsigKeySecret=binddns-tsig-secret
```

ユースケース

現在、ホスト名がサポートされる方法は2つあります。

- イングレス
- サービスタイプ `LoadBalancer`

イングレス

イングレスの場合、目的のイングレスクラスを指定して、次のアノテーションでIPAM範囲を提供する必要があります。

```
ingress.citrix.com/ipam-range: prod
```

次の例で詳細を示します。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  annotations:
    ingress.citrix.com/ipam-range: prod
  name: web-ingress
spec:
  ingressClassName: cic-vpx
  defaultBackend:
    service:
      name: frontend
      port:
        number: 80
  rules:
    - host: web-frontend.example.com
      http:
        paths:
          - backend:
              service:
                name: frontend
                port:
                  number: 80
            path: /
            pathType: ImplementationSpecific
    - host: web-backend.example.com
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    http:
      paths:
        - backend:
            service:
              name: backend
              port:
                number: 80
          path: /
          pathType: ImplementationSpecific
```

イングレスが作成されると、各ルールに対して`spec.rules[].host`フィールドで指定されたホスト名でそれぞれのVIPオブジェクトが作成されます。

```
apiVersion: citrix.com/v1
kind: vip
metadata:
  name: ingress-web-ingress
  namespace: default
spec:
  description: VIP for the web-ingress Service
  hostname:
  - web-frontend.example.com
  - web-backend.example.com
  ipaddress:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  kind: ingress

  name: web-ingress

  range-name: prod
```

IPAMコントローラーはこのVIPを読み取り、利用可能な場合はIP範囲からIPを割り当てます。IP割り当てが完了すると、IPAMはBIND DNSを呼び出し、ホスト名のリストと、BIND DNSでホスト名にAレコードとして割り当てられるIPアドレスを渡します。

サービス

`LoadBalancer`タイプのサービスの場合、次のアノテーションでIPAM範囲を提供する必要があります。

```
service.citrix.com/ipam-range: 'prod'
```

次のアノテーションのホスト名:

```
service.citrix.com/external-hostname: 'svc.lb.example.com'
```

サービスYAML内のサービス クラス。

```
apiVersion: v1

kind: Service

metadata:

  name: apache

  annotations:

    service.citrix.com/ipam-range: 'prod'

    service.citrix.com/external-hostname: 'svc.lb.example.com'

    service.citrix.com/class: 'cic-vpx'

  labels:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    name: apache
spec:
  externalTrafficPolicy: Local
  type: LoadBalancer
  selector:
    name: apache
  ports:
  - name: http
    port: 80
    targetPort: http
```

これにより、`external-hostname`アノテーションで指定されたホスト名を持つVIPオブジェクトが作成されます。

```
apiVersion: citrix.com/v1
kind: vip
metadata:
  name: service-apache
  namespace: default
spec:
  description: VIP for the apache Service
  hostname:
  - svc.lb.example.com
  ipaddress:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kind: service

name: apache

range-name: prod
```

IPAMコントローラーはこのVIPを読み取り、設定されたIP範囲から利用可能なIPアドレスを割り当てます。IPアドレスが割り当てられた後、IPAMコントローラーは、各ホスト名を割り当てられたIPアドレスにマッピングするAレコードをBIND DNSに作成します。

NetScaler GSLB コントローラー

NetScaler GSLB コントローラーに関する以下のトピックが利用可能です。

- GSLB の概要と展開トポロジ
- 複数のサイトに NetScaler GSLB コントローラーを展開する
- 単一サイトに NetScaler GSLB コントローラーを展開する

GSLB の概要と展開トポロジ

概要

高可用性、近接性ベースの負荷分散、およびスケーラビリティを確保するには、複数の分散Kubernetesクラスターにアプリケーションを展開する必要があります。地理的に分散した場所にまたがる複数のKubernetesクラスターにアプリケーションが展開されている場合、アプリケーションインスタンス間でトラフィックを分散するために負荷分散の決定を行う必要があります。

NetScaler GSLBコントローラーは、地理的に分散した場所間でサービスを負荷分散するようにNetScaler（GSLBデバイス）を構成します。GSLBソリューションは、イングレスまたはサービスタイプLoadBalancerを使用して公開されるKubernetesサービスに対して、より優れたパフォーマンスと信頼性を保証します。GSLBトポロジでは、各リージョンにGSLBデバイスが展開されます。GSLBデバイスの1つがプライマリADCとして機能し、その他はセカンダリADCとして機能します。GSLBプライマリADCは、サイト全体に展開された各クラスターに展開されたGSLBコントローラーによって構成されます。このGSLBデバイスは、複数のクラスターにわたって展開されたサービスを負荷分散します。

GSLBの詳細については、「グローバルサーバー負荷分散」を参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

NetScaler GSLBコントローラーイメージは、NetScaler Ingress Controllerのイメージと同じです。

展開トポロジ

GSLB展開トポロジのコンポーネントを以下に示します。

- **GSLBデバイス**: NetScaler MPXまたはNetScaler VPXは、グローバルサーバー負荷分散（GSLB）デバイスとして使用されます。各データセンターにGSLBデバイスが構成されます。各GSLBデバイスでは、1つのサイトがローカルデータセンターを表すローカルサイトとして構成されます。他のサイトはリモートサイトとして構成されます。GSLBデバイスとして使用されるNetScaler MPXまたはNetScaler VPXは、NetScaler Ingress Controllerとともにイングレスデバイスとしても使用できます。
- **イングレスロードバランサー**: NetScaler CPXまたはサードパーティ製アプリケーションは、各Kubernetesクラスターにイングレスロードバランサーとして展開されます。
- **イングレスコントローラー**: イングレスコントローラーは、NetScaler Ingress Controllerまたはサードパーティ製イングレスコントローラーのいずれかです。
- **GSLBコントローラー**: 展開内の各クラスターは、GSLBコントローラーインスタンスを実行します。各GSLBコントローラーは、それぞれのクラスターに展開されたアプリケーションのGSLBプライマリADCを構成します。グローバルサーバー負荷分散（GSLB）構成同期オプションは、プライマリサイトのGSLB構成をGSLBセットアップ内のすべてのGSLBサイトにコピーするために使用されます。GSLB同期を構成するNetScalerはプライマリサイトと呼ばれ、構成がコピーされるサイトはセカンダリサイトと呼ばれます。

次の展開図は、NetScaler GSLBコントローラーのサンプル構成を示しています。各サンプル構成には、異なるリージョンに2つのデータセンター（サイト）が含まれており、各データセンターにはKubernetesクラスターが含まれています。

GSLBサイトで使用されるイングレスコントローラーのタイプに基づいて、2つのサンプル展開トポロジを検討してみましょう。

- 両方のGSLBサイトにNetScaler® Ingress Controller(#netscaler-ingress-controller-in-both-gslb-sites)
- 両方のGSLBサイトにサードパーティのイングレスコントローラー(#any-third-party-ingress-controller-in-both-gslb-sites)

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

GSLBコントローラーの展開は、一方のGSLBサイトにNetScaler Ingress Controller、もう一方のGSLBサイトにサードパーティのイングレスコントローラーがある場合もサポートされます。

両方のGSLBサイトにNetScaler Ingress Controller

注:

GSLBおよびイングレスまたはサービスタイプのLoadBalancerに使用されるNetScaler MPXまたはNetScaler VPXは、同じでも異なっていても構いません。以下の例では、GSLBおよびイングレスまたはサービスタイプのLoadBalancerに同じNetScalerが使用されています。

次の図は、NetScaler Ingress Controllerが存在する場合のNetScaler GSLBコントローラーの展開トポロジを示しています。

以下の手順の番号は、前の図の番号に対応しています。

1. 各クラスターで、NetScaler Ingress Controllerは、`ingress`を使用するか、`service of type LoadBalancer`構成を使用してNetScalerを構成します。
2. 各クラスターで、NetScaler GSLBコントローラーは、プライマリサイトのGSLBデバイスをGSLB構成で構成します。
3. GSLB構成は、異なるGSLBサイトのGSLBデバイス間で自動的に同期されます。
4. アプリケーションのホスト名またはFQDNに対するDNSクエリは、NetScalerで構成されたGSLB仮想サーバーに送信されます。GSLB仮想サーバーでのDNS解決は、構成されたグローバル・トラフィック・ポリシー (GTP) に基づいて、いずれかのクラスター上のIPアドレスに解決されます。
5. DNS解決に基づいて、データトラフィックは、いずれかのクラスターのイングレスフロントエンドIPアドレスまたはサービスタイプLoadBalancer IPアドレスのいずれかに到達します。
6. 必要なアプリケーションはGSLBデバイスを介してアクセスされます。

両方のGSLBサイトにおけるサードパーティ製Ingressコントローラー

次の図は、サードパーティ製Ingressコントローラーが存在する場合のNetScaler GSLBコントローラーの展開トポロジを説明しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

次の項目は、前の図を説明しています。

1. NetScaler GSLBコントローラーは、プライマリサイトのGSLBプライマリADCにGSLB構成を設定します。
2. GSLB構成は、異なるGSLBサイトのGSLBデバイス間で自動的に同期されます。
3. アプリケーションのホスト名またはFQDNに対するDNSクエリは、NetScalerで構成されたGSLB仮想サーバーに送信されます。GSLB仮想サーバーでのDNS解決は、構成されたグローバル・トラフィック・ポリシー（GTP）に基づいて、いずれかのクラスター上のIPアドレスに解決されます。
4. DNS解決に基づいて、データトラフィックは、IngressフロントエンドIPアドレスまたはサービスタイプLoadBalancerフロントエンドIPアドレスのいずれかのクラスターに着信します。
5. 必要なアプリケーションはプロキシを介してアクセスされます。

GSLBメソッドとサポートされる展開タイプ

次のグローバル負荷分散メソッドがサポートされています。

- ラウンドトリップタイム (RTT)
- 静的近接
- ラウンドロビン (RR)

以下のデプロイタイプがサポートされています。

- ローカル優先: ローカル優先デプロイメントでは、あるアプリケーションが別のアプリケーションと通信したい場合、同じクラスター内のローカルアプリケーションを優先します。アプリケーションがローカルで利用できない場合、リクエストは他のクラスターまたはリージョンに転送されます。
- カナリア: カナリアリリースは、新しいソフトウェアバージョンを本番環境に導入する際のリスクを軽減するための手法で、まず変更を少数のユーザーに展開します。このソリューションでは、アプリケーションの新しいバージョンを本番環境に移行する前に、選択したクラスターに展開したい場合にカナリアデプロイメントを使用できます。
- フェイルオーバー: フェイルオーバーデプロイメントは、アプリケーションをアクティブ/アクティブモードでデプロイできない場合に、アクティブ/パッシブ構成でデプロイしたいときに使用されます。
- ラウンドトリップタイム (RTT): RTTデプロイメントでは、ネットワークのリアルタイムステータスが監視され、クライアントのリクエストは最も低いRTT値を持つデータセンターに動的に転送されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- 静的近接性: 静的近接性デプロイメントでは、IPアドレスベースの静的近接性データベースを使用して、クライアントのローカルDNSサーバーとGSLBサイト間の近接性を判断します。リクエストは、近接性基準に最も一致するサイトに送信されます。
- ラウンドロビン: ラウンドロビンデプロイメントでは、GSLBデバイスはそれにバインドされているサービスリストを継続的にローテーションします。リクエストを受信すると、リストの最初のサービスに接続を割り当て、そのサービスをリストの最後に移動します。

注:

現在、IPv6はサポートされていません。

分散Kubernetesクラスターにデプロイされたアプリケーション向けNetScaler GSLBコントローラーを構成するためのCRD

KubernetesアプリケーションのGSLBを実行するためのNetScaler構成をサポートするために、以下のCRDが導入されています。

- グローバル・トラフィック・ポリシー (GTP)
- グローバル・サービス・エントリー (GSE)

GTP CRD

GTP CRDは、デプロイタイプ（カナリア、フェイルオーバー）、GSLBドメイン、イングレスのヘルスモニター、サービスタイプなど、NetScalerでGSLBを構成するためのパラメーターを受け入れます。

GTP CRD の仕様は `<https://github.com/netscaler/netscaler-k8s-ingress-controller/blob/master/gslb/Manifest/gtp-crd.yaml>` で入手できます。

注:

GTP CRD は、特定のドメインのすべてのクラスターで同じです。

次の表は、GTP CRD 属性について説明しています。

フィールド	説明
`ipType`	DNS レコードタイプを A または AAAA として指定します。現在、`A` レコードタイプのみがサポートされています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明
`serviceType`	GSLB サポートが適用されるプロトコルを指定します。
`host`	GSLB サポートが適用されるドメインを指定します。
`trafficPolicy`	GSLB 展開でサポートされるトラフィック分散ポリシーを指定します。
`sourceIpPersistenceId`	一意の送信元IP永続性IDを指定します。この属性により、受信パケットの送信元IPアドレスに基づいて永続性が有効になります。`sourceIpPersistenceId`属性は100の倍数であり、一意である必要があります。
`secLbMethod`	ローカルファースト、カナリア、またはフェイルオーバーにおけるグループ内のクラスター間でサポートされるトラフィック分散ポリシーを指定します。
`destination`	各クラスターのIngressまたはLoadBalancerサービスエンドポイントを指定します。宛先名はGSEの名前と一致する必要があります。
`weight`	クラスター間で分散されるトラフィックの割合を指定します。カナリアデプロイメントの場合、割合はパーセンテージで指定されます。
`CIDR`	ローカルファーストで使用されるCIDRを指定して、ローカリティのスコープを決定します。
`primary`	フェイルオーバーデプロイメントにおいて、宛先がプライマリクラスターであるかバックアップクラスターであるかを指定します。
`monType`	GSLBエンドポイントの健全性を判断するためのプローブのタイプを指定します。モニタータイプがHTTPSの場合、TLSハンドシェイク中にSNIがデフォルトで有効になります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明
`uri`	HTTPおよびHTTPSプロトコル用のGSLBエンドポイントの健全性をプローブするパスを指定します。
`respCode`	HTTPおよびHTTPSプロトコルでGSLBエンドポイントを正常とマークするために期待される応答コードを指定します。
`customHeader`	GSLBエンドポイントの監視トラフィックに追加するカスタムヘッダーを指定します。
`ingressHostName`	GTPがIngressルールと照合する必要がある正確なホスト値を指定します。GSLB FQDNがクラスター内のIngressホスト名と異なる場合、このオプションを使用します。
`multipleIPResponse`	このオプションが有効な場合、GSLB DNSは1つのバックエンドIPアドレスではなく、すべての正常なバックエンドIPアドレスで応答します。
`emptyDownResponse`	このオプションが有効な場合、すべてのメンバーがダウンしているときに、GSLBは空のDNS応答（NXDOMAINではなく）を返します。

GSE CRD

GSE CRDは、各クラスターのエンドポイント情報（クラスターにトラフィックをルーティングするKubernetesオブジェクト）を決定します。

GSE CRDの仕様は、NetScaler Ingress ControllerのGitHubリポジトリ（gse-crd.yaml）で入手できます。

次の表は、**GSE CRD**の属性について説明しています。

フィールド	説明
IPv4アドレス	ローカルクラスターイングレスIPv4アドレス、または`LoadBalancer`タイプのサービスエンドポイントIPv4アドレス

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明
domainName	ローカルクラスターイングレスドメイン名、または`LoadBalancer`タイプのサービスドメイン名
monitorPort	ローカルクラスターイングレスのリスニングポート、または`LoadBalancer`タイプのサービスのリスニングポート

注:

- GSE CRDはクラスター固有であり、各クラスターに対して個別のGSE CRDが作成されます。
- Ingressを使用したGSE CRDの自動生成の場合、ホスト名はGTP CRDインスタンスで指定されたホスト名と一致する必要があります。
- Ingressと`LoadBalancer`タイプのサービスの両方について、GSE CRDは、指定された最初のポートに対してのみ生成されます。
- GSE CRDは、対応するサービスまたはIngressに`status.loadBalancer.ingress.ip/hostname`フィールドが入力されている場合にのみ自動生成されます。

ローカルサイトの優先設定用ConfigMap

GSLBデバイスを構成する際、効率的なトラフィックルーティングと遅延の最小化のために、ADNS IPアドレスに地理的に最も近いKubernetesまたはOpenShiftクラスターにデプロイされた仮想IPアドレスとアプリケーションを選択することが重要です。

サービスが利用可能な場合、GSLBデバイスのローカルサイト優先設定を使用して、ローカル仮想IPアドレスに解決できます。このシナリオは、ロードバランシングポリシーコマンドを使用してGSLBサービスの優先順位を設定することで実現できます。詳細については、「ロードバランシングポリシーコマンドを使用してGSLBサービスの優先順位を構成する」を参照してください。

GSLBデバイス内のサービスまたはサービスグループの優先順位は、効果的なネットワークトラフィック管理と最適化を可能にします。これは通常、サービスグループをGSLB仮想サーバーにバインドする際に構成されます。特定のサービスまたはグループを優先することで、ロードバランシング中に、特に需要が高い場合や低遅延接続が必要な場合に、重要なアプリケーションが優先されるようになります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ローカルサイトの優先設定を有効にするために、`Netscaler-gslb-controller` Helmチャートに`localSiteSelection`パラメーターが追加されます。このパラメーターをtrueに設定すると、GSLBデバイスに構成が自動的に追加されます。このパラメーターは、GSLBコントローラー用のConfigMapを作成し、構成のオンザフライでの追加、変更、削除をサポートします。

注記:

- ConfigMapを介したローカルサイト優先設定の構成（追加、変更、削除のいずれか）では、サービスグループの優先順位を構成または非構成にする必要があるため、トラフィックに影響が出ます。
- ConfigMapはすべてのクラスターで構成する必要があります。

GSLBデバイスのローカルサイト優先設定のサンプル：

```
add lb action <site1-act> -type SELECTIONORDER -value 1 2 3

add lb action <site2-act> -type SELECTIONORDER -value 2 3 1

add lb action <site3-act> -type SELECTIONORDER -value 3 1 2

add lb policy <site1-pol> -rule "CLIENT.IP.DST.EQ<site1_ip>" -action
<site1-act>

add lb policy <site2-pol> -rule "CLIENT.IP.DST.EQ<site2_ip>" -action
<site2-act>

add lb policy <site3-pol> -rule "CLIENT.IP.DST.EQ<site3_ip>" -action
<site3-act>

bind gslb vserver <gslb_name> -serviceGroupName <sg_name_1> -order 1

bind gslb vserver <gslb_name> -serviceGroupName <sg_name_2> -order 2

bind gslb vserver <gslb_name> -serviceGroupName <sg_name_3> -order 3

bind gslb vserver <gslb_name> -policyName <site1-pol> -priority 1

bind gslb vserver <gslb_name> -policyName <site2-pol> -priority 2
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
bind gslb vserver <gslb_name> -policyName <site3-pol> -priority 3
```

Helmチャートを使用してConfigMapを構成する方法については、`<https://github.com/netscaler/netscaler-helm-charts/tree/master/netscaler-gslb-controller>`を参照してください。

OpenShiftオペレーターを使用してConfigMapを構成する方法については、`<https://docs.netscaler.com/ja-jp/netscaler-k8s-ingress-controller/deploy/gslb-openshift-operator>`を参照してください。

NetScaler® GSLBコントローラーの展開

以下の手順では、クラスターにGSLBコントローラーを展開する方法について説明します。

注記：

他のクラスターにGSLBコントローラーを展開するには、手順1から5を繰り返します。

1. GSLBコントローラーがGSLBデバイスに接続し、GSLBコントローラーから構成をプッシュするために必要なシークレットを作成します。

```
kubectl create secret generic secret-1  --from-
literal=username=<username for gslb device1>  --from-
literal=password=<password for gslb device1> --from-
literal=sitesyncpassword=<password1 for secure site-to-site
communication>
```

```
kubectl create secret generic secret-2  --from-
literal=username=<username for gslb device2>  --from-
literal=password=<password for gslb device2> --from-
literal=sitesyncpassword=<password2 for secure site-to-site
communication>
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注記：

- これらのシークレットは、各サイトの`helm install`コマンドを使用してGSLBコントローラーをインストールする際のパラメーターとして提供されます。コマンド内の`username`と`password`は、NetScaler GSLBデバイスユーザーの資格情報を指定します。NetScalerでシステムユーザーアカウントを作成する方法については、「NetScalerでNetScaler Ingress Controllerのシステムユーザーアカウントを作成する」を参照してください。
- GSLBコントローラーバージョン2.3.15以降、GSLBサイト間の通信のセキュリティを強化するために、上記のシークレットを作成する際に、追加のキー`sitesyncpassword`がサポートされます。`sitesyncpassword`キーが提供されない場合、GSLBサイト間の通信にはデフォルトの`password`キーが使用されます。

2. 次のコマンドを使用して、NetScaler Helmチャートリポジトリをローカルのhelmレジストリに追加します。

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/
```

NetScaler Helmチャートリポジトリがすでにローカルレジストリに追加されている場合は、次のコマンドを使用してリポジトリを更新します。

```
helm repo update netscaler
```

3. 次のコマンドを実行して、Helmチャートを使用してクラスターにGSLBコントローラーをインストールします。

注記：

NetScaler Operatorを使用してGSLBコントローラーをインストールする方法については、「NetScaler Operatorを使用してOpenShiftにNetScaler GSLBコントローラーを展開する」を参照してください。

```
helm install gslb-release netscaler/netscaler-gslb-controller -f values.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注記：

このチャートは、推奨されるRBACロールとロールバインディングをデフォルトでインストールします。

例 `values.yaml` ファイル:

```
license:

  accept: yes

localRegion: "east"

localCluster: "cluster1"

openshift: false # set to true for OpenShift deployments

entityPrefix: "k8s"



sitedata:

- siteName: "site1"

  siteIp: "x.x.x.x"

  siteMask: "y.y.y.y"

  sitePublicIp: "z.z.z.z"

  secretName: "secret-1"

  siteRegion: "east"

- siteName: "site2"

  siteIp: "x.x.x.x"

  siteMask: "y.y.y.y"

  sitePublicIp: "z.z.z.z"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  secretName: "secret-2"

  siteRegion: "west"
```

values.yml ファイルで以下のパラメータを指定します。

パラメータ	説明
LocalRegion	GSLB コントローラーが展開されているローカルリージョン。
LocalCluster	GSLB コントローラーが展開されているクラスターの名前。この値は、各 Kubernetes クラスターで一意です。
entityPrefix	NetScaler VPX/MPX 上のリソースのプレフィックス。**注:** entityPrefix はすべてのクラスターで同じである必要があります。
サイトデータ.サイト名	GSLB デバイスで構成されている最初の GSLB サイトの名前。
サイトデータ.サイトIP	最初の GSLB サイトの IP アドレス。site1 の NetScaler の IP アドレスを *sitedata.siteIp* として追加します。
サイトデータ.サイトマスク	最初のGSLBサイトIPアドレスのネットマスク。
サイトデータ.サイトパブリックIP	最初のGSLBサイトのパブリックIPアドレス。
サイトデータ.シークレット名	最初のGSLBサイトのログイン資格情報を含むシークレットの名前。
サイトデータ.サイトリージョン	最初のサイトのリージョン。
サイトデータ.サイト名	GSLBデバイスで構成された2番目のGSLBサイトの名前。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	説明
サイトデータ.サイトIP	2番目のGSLBサイトのIPアドレス。site2のNetScalerのIPアドレスを*sitedata.siteIp*として追加します。
サイトデータ.サイトマスク	2番目のGSLBサイトIPアドレスのネットマスク。
サイトデータ.サイトパブリックIP	2番目のGSLBサイトのパブリックIPアドレス。
サイトデータ.シークレット名	2番目のサイトのログイン資格情報を含むシークレット。
サイトデータ.サイトリージョン	2番目のサイトのリージョン。

注:

GSLBサイト情報の順序は、すべてのクラスターで同じである必要があります。順序の最初のサイトは、構成をプッシュするためのプライマリサイトと見なされます。そのプライマリサイトがダウンすると、リスト内の次のサイトが新しいプライマリになります。たとえば、cluster1でサイトの順序が`site1`の後に`site2`が続く場合、他のすべてのクラスターも同じ順序である必要があります。

4. 次のコマンドを使用してインストールを確認します: `kubectl get pods -l app=gslb-release-netscaler-gslb-controller`。

   各クラスターにGSLBコントローラーが正常にインストールされると、ADNSサービスが構成され、両方のGSLBデバイスで管理アクセスが有効になります。

GSLB構成を同期する

プライマリNetScaler GSLBデバイスで次のコマンドを同じ順序で実行し、プライマリGSLBデバイスとセカンダリGSLBデバイス間でGSLB構成の自動同期を有効にします。

```
set gslb parameter -automaticconfigsync enable

sync gslb config -debug
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

グローバル・トラフィック・ポリシー (GTP) 展開の例

GTP構成は、すべてのクラスターで同じである必要があります。

以下の例では、アプリケーションapp1が東リージョンのcluster1のデフォルト名前空間と、西リージョンのcluster2のデフォルト名前空間に展開されています。

注:

GTP yamlの宛先情報は、`servicename.namespace.region.cluster`の形式である必要があります。ここで、servicenameとnamespaceは、Service種類のKubernetesオブジェクトとその名前空間に対応します。

カナリアおよびフェイルオーバー展開の負荷分散方法を指定できます。
NSICリリース2.1.4以降、GSLBセットアップのサービスに対して複数のモニターを構成できます。詳細については、「GSLBのマルチモニターサポート」を参照してください。

例1：ラウンドロビン展開

この展開を使用して、クラスター全体にトラフィックを均等に分散します。次の例では、ラウンドロビン展開のGTPを構成します。

`weight`フィールドを使用して、より多くのクライアント要求をグループ内の特定のクラスターに転送します。`monitor`パラメーターの下に`customHeader`引数を追加して、GSLBエンドポイント監視トラフィックに追加するカスタムヘッダーを指定します。

```
kubectl apply -f - <<EOF

apiVersion: "citrix.com/v1beta1"

kind: globaltrafficpolicy

metadata:

  name: gtp1

  namespace: default

spec:

  serviceType: 'HTTP'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  hosts:

  - host: 'app1.com'

    policy:

      trafficPolicy: 'ROUNDROBIN'

      targets:

      - destination: 'app1.default.east.cluster1'

        weight: 2

      - destination: 'app1.default.west.cluster2'

        weight: 5

      monitor:

      - monType: http

        uri: ''

        customHeader: "Host: <custom hostname>\r\n x-b3-traceid:
afc38bae00096a96\r\n\r\n"

        respCode: 200

EOF
```

例2：フェイルオーバー展開

このポリシーを使用して、アプリケーションをアクティブ/パッシブモードで構成します。フェイルオーバー展開では、アプリケーションは複数のクラスターに展開されます。フェイルオーバーは、GTPポリシーでそれらのターゲット宛先に割り当てられた重みに基づいて、異なるクラスター内のアプリケーションインスタンス（ターゲット宛先）間で実現されます。

次の例は、フェイルオーバーのGTP構成例を示しています。プライマリフィールドを使用して、アクティブなターゲット宛先とパッシブなターゲット宛先を指定できます。プライマリフィールドの

© 1997–2026 Citrix Systems, Inc. All rights reserved.

デフォルト値は`True`で、ターゲット宛先がアクティブであることを示します。各クラスターのエンドポイントにモニターをバインドして、そのヘルスをプローブします。

```
kubectl apply -f - <<EOF

apiVersion: "citrix.com/v1beta1"

kind: globaltrafficpolicy

metadata:

  name: gtp1

  namespace: default

spec:

  serviceType: 'HTTP'

  hosts:

  - host: 'app1.com'

    policy:

      trafficPolicy: 'FAILOVER'

      secLbMethod: 'ROUNDROBIN'

      targets:

      - destination: 'app1.default.east.cluster1'

        weight: 1

      - destination: 'app1.default.west.cluster2'

        primary: false

        weight: 1

      monitor:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      - monType: http
        uri: ''
        respCode: 200
EOF
```

例3：RTT展開

このポリシーを使用して、ネットワークのリアルタイムステータスを監視し、クライアント要求を最も低いRTT値を持つターゲット宛先に動的に転送します。

以下は、ラウンドトリップタイム展開のグローバルなトラフィックポリシーの例です。

```
kubectl apply -f - <<EOF
apiVersion: "citrix.com/v1beta1"
kind: globaltrafficpolicy
metadata:
  name: gtp1
  namespace: default
spec:
  serviceType: 'HTTP'
  hosts:
  - host: 'app1.com'
    policy:
      trafficPolicy: 'RTT'
      targets:
      - destination: 'app1.default.east.cluster1'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      - destination: 'app1.default.west.cluster2'

      monitor:

      - monType: tcp

EOF
```

例4：カナリア展開

アプリケーションの新しいバージョンを本番環境に移行する前に、選択したクラスターに展開する場合は、カナリア展開を使用します。

このセクションでは、カナリア展開を使用したグローバルなトラフィックポリシーの例について説明します。このポリシーでは、アプリケーションの新しいバージョンを本番環境に展開する前にロールアウトする必要があります。

この例では、アプリケーションは`west`リージョンのクラスター`cluster2`に展開されています。アプリケーションの新しいバージョンは、`east`リージョンの`cluster1`に展開されています。`weight`フィールドを使用して、各クラスターにリダイレクトされるトラフィックの量を指定できます。ここでは、`weight`が40パーセントと指定されています。したがって、トラフィックの40パーセントのみが新しいバージョンに転送されます。宛先に`weight`フィールドが指定されていない場合、それは大部分のトラフィックを受け取る本番環境の一部と見なされます。アプリケーションの新しいバージョンが安定していると判断された場合、新しいバージョンは他のクラスターにもロールアウトできます。

```
kubectl apply -f - <<EOF

apiVersion: "citrix.com/v1beta1"

kind: globaltrafficpolicy

metadata:

  name: gtp1

  namespace: default

spec:

  serviceType: 'HTTP'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  hosts:
  - host: 'app1.com'
    policy:
      trafficPolicy: 'CANARY'
      secLbMethod: 'ROUNDROBIN'
      targets:
      - destination: 'app1.default.east.cluster1'
        weight: 40
      - destination: 'app1.default.west.cluster2'
      monitor:
      - monType: http
        uri: ''
        respCode: 200
EOF
```

例 5: 静的近接性

このポリシーを使用して、近接基準に最も一致するサービスを選択します。

以下のGTPは、静的近接性デプロイメントの例です。

注:

静的近接性の場合、すべてのGSLBデバイスにロケーションデータベースを手動で適用する必要があります: `add locationfile /var/netscaler/inbuilt_db/Citrix_Netscaler_InBuilt_GeoIP_DB_IPv4`。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl apply -f - <<EOF

apiVersion: "citrix.com/v1beta1"

kind: globaltrafficpolicy

metadata:

  name: gtp1

  namespace: default

spec:

  serviceType: 'HTTP'

  hosts:

  - host: 'app1.com'

    policy:

      trafficPolicy: 'STATICPROXIMITY'

      targets:

      - destination: 'app1.default.east.cluster1'

      - destination: 'app1.default.west.cluster2'

      monitor:

      - monType: http

        uri: ''

        respCode: 200

EOF
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

例 6: 送信元IPの永続性

以下のトラフィックポリシーは、パラメータ `sourceIpPersistenceId` を指定して送信元IPの永続性を有効にする例です。

```
kubectl apply -f - <<EOF

apiVersion: "citrix.com/v1beta1"

kind: globaltrafficpolicy

metadata:

  name: gtp1

  namespace: default

spec:

  serviceType: 'HTTP'

  hosts:

  - host: 'app1.com'

    policy:

      trafficPolicy: 'ROUNDROBIN'

      sourceIpPersistenceId: 300

      targets:

      - destination: 'app1.default.east.cluster1'

        weight: 2

      - destination: 'app1.default.west.cluster2'

        weight: 5

      monitor:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      - monType: tcp

        uri: ''

        respCode: 200

EOF
```

グローバルサービスエントリ (GSE) の例

GSE構成は、クラスタエンドポイント情報に基づいて特定のクラスタに適用されます。GSE名は、グローバルなトラフィックポリシーのターゲット宛先名と同じである必要があります。

注:

GSEの作成はオプションです。GSEが作成されていない場合、NetScaler Ingress Controllerは、`<svcname>.<namespace>.<region>.<cluster>` 形式に一致するホストを持つ一致するイングレスを探します。

前のセクションで述べたグローバルなトラフィックポリシーの場合、cluster1のグローバルサービスエントリは次のとおりです。この例では、グローバルサービスエントリ名 `app1.default.east.cluster1` は、作成されたグローバルなトラフィックポリシーのターゲット宛先名の1つです。

```
kubectl apply -f - <<EOF

apiVersion: "citrix.com/v1beta1"

kind: globalserviceentry

metadata:

  name: 'app1.default.east.cluster1'

  namespace: default

spec:

  endpoint:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    ipv4address: 10.102.217.70

    monitorPort: 33036

EOF
```

GSLBのマルチモニターサポート

GSLBセットアップでは、同じホストのサービスを監視するために複数のモニターを構成できます。モニターは、サービスのヘルスチェックに使用されるリクエストプロトコルに応じて、異なるタイプにすることができます。例えば、HTTP、HTTPS、TCPなどです。

複数のモニターを構成することに加えて、モニターに対して以下の追加パラメータを定義できます。

- `Destination port`: モニターがヘルスチェックに使用するサービスポート。
- `SNI`: このモニターのSNIステータス。指定可能な値は`True`と`False`です。
- `CN`: SNIリクエストで使用される共通名 (CN)。共通名は通常、監視対象サーバーのドメイン名です。

要件に応じて、各モニターのパラメーターの組み合わせを定義することもできます。マルチモニターをサポートするGTPでは、各サービスを監視でき、いずれかのサービスがダウンした場合、サイトはダウンとマークされ、トラフィックは別のサイトに転送されます。次のGTP YAML例には、複数のモニターが含まれています。

```
kubectl apply -f - <<EOF

apiVersion: "citrix.com/v1beta1"

kind: globaltrafficpolicy

metadata:

  name: gtp1

  namespace: default

spec:

  serviceType: 'HTTP'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
hosts:
- host: 'app1.com'
  policy:
    trafficPolicy: 'STATICPROXIMITY'
    targets:
    - destination: 'app1.default.east.cluster1'
    - destination: 'app1.default.west.cluster2'
    monitor:
    - monType: HTTPS
      uri: ''
      respCode: '200,300,400'
      destinationPort: 1000
    - monType: HTTP
      uri: ''
      respCode: '200,300,400'
      destinationPort: 3000
    - monType: HTTPS
      uri: ''
      respCode: '200,300,400'
      destinationPort: 4000
    - monType: TCP
      uri: ''
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        destinationPort: 5000
        respCode: '200'
      - monType: HTTPS
        uri: 'test.com'
        sni: True
        respCode: '200,300,400'
        destinationPort: 6000
      - monType: HTTPS
        uri: 'test.com'
        sni: True
        commonName: 'testabc.com'
        respCode: '200,300,400'
        destinationPort: 7000
  status:
    {}
EOF
```

NetScaler® GSLBコントローラー（シングルサイト用）

概要

高可用性、近接性ベースの負荷分散、およびスケーラビリティを確保するには、複数のKubernetesクラスターにアプリケーションをデプロイする必要があります。GSLBソリューションは、Ingressを使用して公開されるKubernetesサービスのパフォーマンスと信頼性を向上させます。NetScaler GSLBコントローラーは、地理的に分散した場所間でサービスを負荷分散するようにNetScaler（GSLBデバイス）を構成します。シングルサイトGSLBソリューションでは、データ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

センター内のGSLBデバイスは、データセンターの各Kubernetesクラスターにデプロイされた GSLBコントローラーによって構成されます。このGSLBデバイスは、データセンターの複数のクラスターにデプロイされたサービスを負荷分散します。

次の図は、2つのKubernetesクラスターと単一のGSLBサイトを持つデータセンターにおける NetScaler GSLBコントローラーの展開トポロジを示しています。

注:

GSLBとIngressに使用されるNetScaler（MPXまたはVPX）は、同じでも異なっていてもかまいません。次の図では、GSLBとIngressに同じNetScalerが使用されています。

以下の手順の番号は、前の図の番号に対応しています。

1. 各クラスターでは、NetScaler Ingress Controllerが `Ingress` を使用してNetScalerを構成します。
2. 各クラスターでは、NetScaler GSLBコントローラーがGSLB構成でGSLBデバイスを構成します。
3. アプリケーションURLのDNSクエリは、NetScalerで構成されたGSLB仮想サーバーに送信されます。GSLB仮想サーバーでのDNS解決は、構成されたグローバル・トラフィック・ポリシー（GTP）に基づいて、いずれかのクラスター上のIPアドレスに解決されます。
4. DNS解決に基づいて、データトラフィックは、IngressフロントエンドIPアドレスまたはいずれかのクラスターのコンテンツスイッチング仮想サーバーIPアドレスのいずれかに到達します。
5. 必要なアプリケーションは、GSLBデバイスを介してアクセスされます。

NetScaler GSLBコントローラーの展開

前提条件

- クラウドまたはオンプレミスでホストされているKubernetesクラスターへのアクセス。クラウド内のKubernetesクラスターは、マネージドKubernetes（例：GKE、EKS、AKS）またはカスタム作成されたKubernetesデプロイメントのいずれかです。Kubernetesクラスターは適切に構成され、稼働している必要があります。NetScalerとクラスターのPodネットワーク間のネットワーク接続が確保されていることを確認してください。
- Kubernetesデプロイメントの場合、バージョン1.21以降を実行しているKubernetesクラスターへのアクセスが必要です。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- OpenShiftデプロイメントの場合、バージョン4.11以降を実行しているOpenShiftクラスターへのアクセスが必要です。
- Helmバージョン3.x以降がインストールされていること。Helmのインストールについては、こちらを参照してください。
- NetScaler MPX/VPXは、特定のニーズに応じて、スタンドアロン、HA、またはクラスター構成でデプロイされます。
  - NetScaler MPXのデプロイ手順については、NetScalerドキュメントを参照してください。
  - NetScaler VPXのデプロイ手順については、NetScaler VPXインスタンスの展開を参照してください。
- NetScaler Ingress ControllerがNetScalerと通信するために使用するNSIP (NetScaler IP) アドレスを決定します。IPアドレスは、NetScalerデプロイメントのタイプに応じて、以下のいずれかになります。
  - NSIP (スタンドアロンアプライアンスの場合) - スタンドアロンNetScalerアプライアンスの管理IPアドレスです。詳細については、NetScalerでのIPアドレス指定を参照してください。
  - SNIP (高可用性モードのアプライアンスの場合) - サブネットIPアドレスです。詳細については、NetScalerでのIPアドレス指定を参照してください。
  - CLIP (クラスターモードのアプライアンスの場合) - クラスターNetScaler展開のクラスター管理IP (CLIP) アドレスです。詳細については、クラスターのIPアドレス指定を参照してください。
- NetScaler VPXまたはNetScaler MPXのユーザーアカウント。NetScaler Ingress Controllerは、NetScaler内のシステムユーザーアカウントを使用してNetScaler MPXまたはNetScaler VPXを構成します。NetScalerでシステムユーザーアカウントを作成する手順については、NetScaler Ingress Controller 用のNetScalerシステムユーザーアカウントの作成を参照してください。

  ユーザー名とパスワードを直接渡すか、Kubernetesシークレットを使用できます。Kubernetesシークレットを使用する場合は、次のコマンドを使用してユーザー名とパスワードのシークレットを作成します。

  ```
  kubectl create secret generic nslogin --from-
  literal=username=<username> --from-literal=password=<password>
  ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

シングルサイト向けGSLBコントローラーのデプロイ手順

この手順では、サイトの2つのクラスターに同じHTTPSアプリケーションをデプロイし、各クラスターにGSLBコントローラーをデプロイします。これにより、GSLBコントローラーはGSLBデバイス（NetScaler VPXまたはNetScaler MPX）を構成し、クラスター全体にデプロイされたサービスの負荷分散を行います。

注記:

- シングルサイトGSLBソリューションをセットアップするには、両方のクラスターで以下の手順を繰り返します。特定クラスターで実行すべきことを強調する例外が一部の手順に追加されていますので、それに応じて手順を実行してください。
- アプリケーションの名前空間にイングレスリソースをデプロイする必要があります。
- GTPおよびGSEリソースは同じ名前空間にデプロイする必要があります。これらのリソースはアプリケーションの名前空間にデプロイすることをお勧めします。

1. NetScaler VPXまたはNetScaler MPXで、証明書署名要求 (CSR) の共通名として`cnn.com`を使用して証明書-キーペアを作成します。証明書-キーペアの作成については、証明書の作成を参照してください。
2. 次のコマンドを使用してCNNアプリケーションをデプロイします。

```
kubectl apply -f - <<EOF

apiVersion: apps/v1

kind: Deployment

metadata:

name: cnn-website

labels:

  name: cnn-website

  app: cnn-website

spec:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
selector:
  matchLabels:
    app: cnn-website
replicas: 2
template:
  metadata:
    labels:
      name: cnn-website
      app: cnn-website
  spec:
    containers:
    - name: cnn-website
      image: quay.io/sample-apps/cnn-website:v1.0.0
      ports:
      - name: http-80
        containerPort: 80
      - name: https-443
        containerPort: 443
---
apiVersion: v1
kind: Service
metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
name: cnn-website

labels:

  app: cnn-website

spec:

type: NodePort

ports:

- name: http-80

  port: 80

  targetPort: 80

- name: https-443

  port: 443

  targetPort: 443

selector:

  name: cnn-website

EOF
```

注記:

OpenShiftのデプロイでは、次のコマンド`oc adm policy add-scc-to-user anyuid system:serviceaccount:<namespace>:default`を実行して、サービスアカウントに特定のセキュリティコンテキスト制約 (SCC) を付与します。`<namespace>`をCNNアプリケーションをデプロイした実際の名前空間に置き換えてください。

3. NSICをデプロイします。

   a. 次のコマンドを使用して、NetScaler Helmチャートリポジトリをローカルの Helmレジストリに追加します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/
```

NetScaler Helmチャートリポジトリがすでにローカルレジストリに追加されている場合は、次のコマンドを使用してリポジトリを更新します。

```
helm repo update netscaler
```

b. `values.yaml`を更新して、以下に示すようにNetScaler Ingress Controllerを構成します。

cluster1の例 `values.yaml`

```
license:

  accept: yes

adcCredentialSecret: nslogin # K8s Secret created as part of prerequisiste

nsIP: <x.x.x> # CLIP (for appliances in Cluster mode), SNIP (for appliances in High Availability mode) , NSIP (for standalone appliances)

openshift: false # set to true for OpenShift deployments

entityPrefix: cluster1 # unique for each NSIC instance.

clusterName: cluster1

ingressClass: ['nsic-vpx'] # ingress class used in the ingress resources

 # serviceClass- To use service type LB, specify the service class
```

cluster2の例 `values.yaml`

```
license:

  accept: yes

adcCredentialSecret: nslogin # K8s Secret created as part of
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
prerequisiste

nsIP: <x.x.x> # CLIP (for appliances in Cluster mode), SNIP (for
appliances in High Availability mode) , NSIP (for standalone
appliances)

openshift: false # set to true for OpenShift deployments

entityPrefix: cluster2 # unique for each NSIC instance.

clusterName: cluster2

ingressClass: ['nsic-vpx'] # ingress class used in the ingress
resources

# serviceClass- To use service type LB, specify the service class
```

c. 次のコマンドを実行して、Helmチャートを使用してNSICをインストールします。

```
helm install nsic netscaler/netscaler-ingress-controller -f
values.yaml
```

NSICインストール中に設定できる必須およびオプションのパラメータについては、設定を参照してください。

注:

以前のバージョンのNSICがすでにクラスターにデプロイされている場合は、次のコマンドを使用してCRD仕様をデプロイします: `kubectl apply -f https://raw.githubusercontent.com/netscaler/netscaler-helm-charts/refs/heads/master/netscaler-ingress-controller/crds/crds.yaml`。

4. 次のイングレスリソースをデプロイします。

cluster1のイングレスリソース。

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
metadata:

 name: frontend-ingress

 annotations:

   ingress.citrix.com/frontend-ip: <NSVIP1> # vserver IP1

   ingress.citrix.com/preconfigured-certkey: '{"certs": [{"name":
"singlesite", "type": "default"}]}' # provide the certificate key
created in step 1

spec:

 tls:

 - {}

 ingressClassName: nsic-vpx

 rules:

   - host: cnn.com

     http:

       paths:

         - path: /

           pathType: Prefix

           backend:

             service:

               name: cnn-website

               port:

                 number: 80
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
EOF
```

cluster2のイングレスリソース。

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: frontend-ingress

  annotations:

    ingress.citrix.com/frontend-ip: <NSVIP2> # vserver IP2

    ingress.citrix.com/preconfigured-certkey: '{"certs": [{"name":
"singlesite", "type": "default"}]}' # provide the name of the
certificate key created in step 1

spec:

  tls:

    - {}

  ingressClassName: nsic-vpx

  rules:

    - host: cnn.com

      http:

        paths:

          - path: /

            pathType: Prefix
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            backend:
              service:
                name: cnn-website
                port:
                  number: 80
EOF
```

5. GSLBコントローラーがGSLBデバイスに接続し、GSLBコントローラーから構成をプッシュするために必要なシークレットを作成します。

```
kubectl create secret generic secret--from-literal=username=<username
for gslb device>--from-literal=password=<password for gslb device>
```

注:

- このシークレットは、GSLBコントローラーの`helm install`コマンドで、それぞれのサイトのパラメータとして提供されます。コマンドでNetScaler (GSLBデバイス) ユーザーの資格情報を`username`および`password`として指定します。
- この場合、GSLBデバイスとイングレスデバイスが同じであるため、前提条件セクションで作成された同じシークレットを使用できます。

6. 次のコマンドを実行して、Helmチャートを使用してGSLBコントローラーをインストールします。

```
helm install my-release netscaler/netscaler-gslb-controller -f
values.yaml
```

注:

このチャートは、推奨されるRBACロールとロールバインディングをデフォルトでインストールします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

例の`values.yaml`ファイル:

```
license:

  accept: yes

localRegion: "east"

localCluster: "cluster1"  # use cluster2 when deploying GSLB controller
in cluster2

entityPrefix: "gslb" # should be same for GSLB controller in both
clusters

nsIP: "x.x.x.x"

openshift: false # set to true for OpenShift deployments

adcCredentialSecret: <Secret-for-NetScaler-credentials>

sitedata:

  - siteName: "site1"

    siteIp: "x.x.x.x"

    siteMask:

    sitePublicip:

    secretName: "secret"

    siteRegion: "east"
```

YAMLファイルで次のパラメータを指定します。

パラメータ	説明
LocalRegion	GSLBコントローラーがデプロイされているローカルリージョン。この値は、すべてのクラスターにわたるGSLBコントローラーのデプロイメントで同じです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	説明
LocalCluster	GSLBコントローラーがデプロイされているクラスターの名前。この値は、Kubernetesクラスターごとに一意です。
sitedata.サイト名	GSLBサイトの名前。
sitedata.サイトIP	GSLBサイトのIPアドレス。サイト1のNetScalerのIPアドレスを*sitedata.siteIp*として追加します。
sitedata.サイトマスク	GSLBサイトIPアドレスのネットマスク。
sitedata.サイトパブリックIP	GSLBサイトのサイトパブリックIPアドレス。
sitedata.シークレット名	GSLBサイトのログイン資格情報を含むシークレットの名前。
サイトデータ.サイトリージョン	GSLBサイトのリージョン。
NSIP	GSLBデバイスのSNIP (サブネットIPアドレス)。NetScalerに*sitedata.siteIp*をSNIPとして追加します。
adcCredentialSecret	NetScaler VPXまたはMPXのログイン資格情報を含むKubernetesシークレット。

各クラスターにGSLBコントローラーが正常にインストールされると、GSLBサイトとADNSサービスが構成され、GSLBサイトのIPアドレスで管理アクセスが有効になります。

注:

以前のバージョンのGSLBコントローラーがクラスターにすでにデプロイされている場合は、次のコマンドを使用してGTPおよびGSE CRD仕様をデプロイします: `kubectl apply -f https://raw.githubusercontent.com/netscaler/netscaler-k8s-ingress-controller/master/gslb/Manifest/gtp-crd.yaml` および `kubectl apply -f https://raw.githubusercontent.com/netscaler/netscaler-k8s-ingress-controller/master/gslb/Manifest/gse-crd.yaml`。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

7. 要件に基づいてグローバル・トラフィック・ポリシーをデプロイします。

注:

- GTP構成がすべてのクラスターで同じであることを確認してください。GTP CRDと許可される値の詳細については、GTP CRDを参照してください。
- GTP YAMLの宛先情報は、`servicename.namespace.region.cluster`の形式である必要があります。ここで、サービス名と名前空間は、それぞれServiceタイプのKubernetesオブジェクトとその名前空間に対応します。

カナリアおよびフェイルオーバーデプロイメントのロードバランシング方法を指定できます。

- **例1：ラウンドロビン展開**

  この展開を使用して、トラフィックをクラスター全体に均等に分散します。次の例では、ラウンドロビン展開用にGTPを構成します。 `weight`フィールドを使用して、より多くのクライアント要求をグループ内の特定のクラスターに誘導できます。

```
kubectl apply -f - <<EOF

apiVersion: "citrix.com/v1beta1"

kind: globaltrafficpolicy

metadata:

  name: gtp1

spec:

  serviceType: 'HTTP'

  hosts:

  - host: 'cnn.com'

    policy:

      trafficPolicy: 'ROUNDROBIN'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      targets:
      - destination: 'cnn-website.default.east.cluster1'
        weight: 2
      - destination: 'cnn-website.default.east.cluster2'
        weight: 5
      monitor:
      - monType: http
        uri: ''
        respCode: 200
EOF
```

- **例2：フェイルオーバー展開**

このポリシーを使用して、アプリケーションをアクティブ/パッシブモードで構成します。フェイルオーバー展開では、アプリケーションは複数のクラスターに展開されます。フェイルオーバーは、GTPポリシーでターゲット宛先に割り当てられた重みに基づいて、ターゲット宛先のインスタンス間で実現されます。

次の例は、フェイルオーバーのGTP構成例を示しています。`primary`フィールドを使用して、どのターゲット宛先がアクティブで、どのターゲット宛先がパッシブであるかを指定できます。`primary`フィールドのデフォルト値は`True`で、ターゲット宛先がアクティブであることを示します。各クラスターのエンドポイントにモニターをバインドして、そのヘルスをプローブします。

```
kubectl apply -f - <<EOF
apiVersion: "citrix.com/v1beta1"
kind: globaltrafficpolicy
metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: gtp1
spec:
  serviceType: 'HTTP'
  hosts:
  - host: 'cnn.com'
    policy:
      trafficPolicy: 'FAILOVER'
      secLbMethod: 'ROUNDROBIN'
      targets:
      - destination: 'cnn-website.default.east.cluster1'
        weight: 1
      - destination: 'cnn-website.default.east.cluster2'
        primary: false
        weight: 1
      monitor:
      - monType: http
        uri: ''
        respCode: 200
EOF
```

- **例3：RTT展開**

  このポリシーを使用して、ネットワークのリアルタイムステータスを監視し、クライアント要求を最も低いRTT値を持つターゲット宛先に動的に誘導します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

次の例では、RTT展開用にGTPを構成します。

```
kubectl apply -f - <<EOF

apiVersion: "citrix.com/v1beta1"

kind: globaltrafficpolicy

metadata:

  name: gtp1

spec:

  serviceType: 'HTTP'

  hosts:

  - host: 'cnn.com'

    policy:

      trafficPolicy: 'RTT'

      targets:

      - destination: 'cnn-website.default.east.cluster1'

      - destination: 'cnn-website.default.east.cluster2'

      monitor:

      - monType: http

        uri: ''

        respCode: 200

EOF
```

- **例4：カナリア展開**

© 1997–2026 Citrix Systems, Inc. All rights reserved.

カナリア展開は、アプリケーションの新しいバージョンを本番環境に移行する前に、選択したクラスターにロールアウトする場合に使用します。

このセクションでは、カナリア展開を使用したグローバル・トラフィック・ポリシーの例について説明します。このポリシーでは、アプリケーションの新しいバージョンを本番環境に展開する前に、段階的にロールアウトする必要があります。

この例では、アプリケーションの安定版が`cluster2`に展開されます。アプリケーションの新しいバージョンは`cluster1`に展開されます。`weight`フィールドを使用して、各クラスターにリダイレクトされるトラフィックの量を指定します。ここでは、`weight`が40パーセントと指定されています。したがって、トラフィックの40パーセントのみが新しいバージョンに誘導されます。宛先に`weight`フィールドが指定されていない場合、それは大部分のトラフィックを受け取る本番環境の一部と見なされます。アプリケーションの新しいバージョンが安定したら、新しいバージョンを他のクラスターにロールアウトできます。

```
kubectl apply -f - <<EOF

apiVersion: "citrix.com/v1beta1"

kind: globaltrafficpolicy

metadata:

  name: gtp1

  namespace: default

spec:

  serviceType: 'HTTP'

  hosts:

  - host: 'cnn.com'

    policy:

      trafficPolicy: 'CANARY'

      secLbMethod: 'ROUNDROBIN'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      targets:
      - destination: 'cnn-website.default.east.cluster1'
        weight: 40
      - destination: 'cnn-website.default.east.cluster2'
      monitor:
      - monType: http
        uri: ''
        respCode: 200
EOF
```

- **例5：静的近接性**

このポリシーを使用して、近接性基準に最も一致するサービスを選択します。次のトラフィックポリシーは、静的近接性展開の例です。

注：

静的近接性の場合、ロケーションデータベースを手動で適用する必要があります: `add locationfile /var/netscaler/inbuilt_db/Citrix_Netscaler_InBuilt_GeoIP_DB_IPv4`。

```
kubectl apply -f - <<EOF
apiVersion: "citrix.com/v1beta1"
kind: globaltrafficpolicy
metadata:
  name: gtp1
  namespace: default
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:
  serviceType: 'HTTP'
  hosts:
  - host: 'cnn.com'
    policy:
      trafficPolicy: 'STATICPROXIMITY'
      targets:
      - destination: 'cnn-website.default.east.cluster1'
      - destination: 'cnn-website.default.east.cluster2'
      monitor:
      - monType: http
        uri: ''
        respCode: 200
EOF
```

- **例 6: ソースIP永続性**

次のトラフィックポリシーは、パラメータ `sourceIpPersistenceId` を指定してソースIP永続性を有効にする例です。

```
kubectl apply -f - <<EOF
apiVersion: "citrix.com/v1beta1"
kind: globaltrafficpolicy
metadata:
  name: gtp1
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  namespace: default
spec:
  serviceType: 'HTTP'
  hosts:
  - host: 'cnn.com'
    policy:
      trafficPolicy: 'ROUNDROBIN'
      sourceIpPersistenceId: 300
      targets:
      - destination: 'cnn-website.default.east.cluster1'
        weight: 2
      - destination: 'cnn-website.default.east.cluster2'
        weight: 5
      monitor:
      - monType: http
        uri: ''
        respCode: 200
EOF
```

8. GSEリソースをデプロイします。

   GSE構成は、クラスターエンドポイント情報に基づいて特定のクラスターに適用されます。GSE名は、グローバルなトラフィックポリシーにおけるターゲット宛先名と同じである必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

GSEの作成はオプションです。GSEが作成されていない場合、NetScaler Ingress Controllerはホストが `<svcname>.<namespace>.<region>.<cluster>` 形式に一致するイングレスを探します。

以前の例で述べたグローバルなトラフィックポリシーの場合、次のYAMLはcluster1のグローバルサービスエントリです。この例では、グローバルサービスエントリ名 `cnn.default.east.cluster1` は、グローバルなトラフィックポリシーにおけるターゲット宛先名の一つです。

```
kubectl apply -f - <<EOF

apiVersion: "citrix.com/v1beta1"

kind: globalserviceentry

metadata:

  name: 'cnn-website.default.east.cluster1'

spec:

  endpoint:

    ipv4address: NSVIP1 # vserver IP1

    monitorPort: 80

EOF
```

以前の例で述べたグローバルなトラフィックポリシーの場合、次のYAMLはcluster2のグローバルサービスエントリです。この例では、グローバルサービスエントリ名 `cnn.default.east.cluster2` は、グローバルなトラフィックポリシーにおけるターゲット宛先名の一つです。

```
kubectl apply -f - <<EOF

apiVersion: "citrix.com/v1beta1"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kind: globalserviceentry

metadata:

  name: 'cnn-website.default.east.cluster2'

spec:

  endpoint:

    ipv4address: NSVIP2 # vserver IP2

    monitorPort: 80

EOF
```

9. DNSクエリを送信するには、次のコマンドを使用します。

```
dig@siteIP cnn.com
```

サービスメッシュライト

Ingressソリューション（ハードウェア、仮想化、またはコンテナ化のいずれか）は通常、南北（N-S）トラフィックに対してL7プロキシ機能を実行します。サービスメッシュライトアーキテクチャは、同じIngressソリューションを使用して東西トラフィックも管理します。

標準的なKubernetesデプロイメントでは、東西（E-W）トラフィックは各ノードにデプロイされた組み込みのkube-proxyを通過します。Kube-proxyはL4プロキシであり、TCP/UDPベースの負荷分散しか実行できず、L7プロキシが提供する利点を提供できません。

NetScaler（MPX、VPX、またはCPX）は、E-Wトラフィックに対してL7プロキシの利点を提供できます。例えば、以下のようなものです。

- 相互TLSおよびSSLオフロード。
- コンテンツベースのルーティング、HTTPおよびHTTPSヘッダーパラメータに基づくトラフィックの許可またはブロック。
- 高度な負荷分散アルゴリズム（最小接続数または最小応答時間）。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- ゴールデンシグナル（エラー、遅延、飽和、トラフィック量）の測定による東西トラフィックの可観測性。NetScaler Console Service Graphは、マイクロサービスを監視およびデバッグするための可観測性ソリューションです。

サービスメッシュアーキテクチャ（IstioやLinkerDなど）は管理が複雑です。サービスメッシュライトアーキテクチャは軽量版であり、同じ要件を達成するために開始するのがはるかに簡単です。

サービスメッシュライトアーキテクチャでNetScaler CPXを使用して東西通信を構成するには、まずkube-proxyが東西トラフィックを管理するためにどのように構成されているかを理解する必要があります。

kube-proxyによる東西通信

マイクロサービス用にKubernetesデプロイメントを作成すると、Kubernetesはレプリカ数に基づいて一連のPodをデプロイします。これらのPodにアクセスするには、それらのPodにアクセスするための抽象化を提供するKubernetesサービスを作成します。この抽象化は、サービスにクラスターIPアドレスを割り当てることによって提供されます。

Kubernetes DNSには、サービス名をクラスターIPアドレスにマッピングするアドレスレコードが設定されます。そのため、例えば`tea`というアプリケーションが`coffee`というマイクロサービスにアクセスしたい場合、DNSは`coffee`サービスのクラスターIPアドレスを`tea`アプリケーションに返します。`tea`アプリケーションは接続を開始し、その接続はkube-proxyによって傍受され、一連の`coffee` Podに負荷分散されます。

サービスメッシュライトアーキテクチャにおけるNetScaler CPXによる東西通信

目標は、NetScaler CPXを東西パスに挿入し、Ingressルールを使用してこのトラフィックを制御することです。

NetScaler CPXとの東西通信を構成するには、次の手順を実行します。

ステップ1：コーヒーサービス定義をNetScaler CPXを指すように変更する

NetScaler CPXが東西トラフィックを管理するには、マイクロサービス（例: `coffee`）のFQDNが、ターゲットマイクロサービス（`coffee`）のクラスターIPではなく、NetScaler CPXのIPアドレスを指す必要があります。（このNetScaler CPXの展開は、Ingress NetScaler CPXデバイスと同じにすることができます。）この変更後、Kubernetesクラスター内のPodがコーヒーサービスのFQDNを解決すると、NetScaler CPXのIPアドレスが返されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注記：

オブザーバビリティのためにNetScaler Consoleでサービスグラフを起動するためにサービスメッシュライトを展開している場合は、サービスを変更した後、NetScaler CPX IPアドレスを指すすべてのアプリケーションサービスにラベル `citrix-adc: cpx` を追加する必要があります。

ステップ2：コーヒーマイクロサービスPod用に `coffee-headless` という名前のヘッドレスサービスを作成する

`coffee` サービスをNetScaler CPXを指すように変更したため、コーヒーマイクロサービスの展開を表すサービスをもう1つ作成する必要があります。

以下は、ヘッドレスサービスリソースのサンプルです。

```
apiVersion: v1

kind: Service

metadata:

  name: coffee-headless

spec:

#headless Service

  clusterIP: None

  ports:

  - name: coffee-443

    port: 443

    targetPort: 443

  selector:

    name: coffee-deployment
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ステップ3： `coffee-headless` サービス用のルールを含むIngressリソースを作成する

前の手順での変更により、NetScaler CPXを構成してコーヒーマイクロサービスPodへの東西トラフィックを制御するIngressオブジェクトを作成する準備が整いました。

以下は、Ingressリソースのサンプルです。

これらの変更を伴う通常のIngressロードバランシング手法を使用すると、NetScaler CPXは東西トラフィックをロードバランスできるようになります。次の図は、NetScaler CPXサービスメッシュライトアーキテクチャがIngressルールを使用して `tea` と `coffee` マイクロサービス間の東西通信にL7プロキシを提供する方法を示しています。

サンプル(/en-us/netscaler-k8s-ingress-controller/media/coffee-micro-summary.png)

Service Mesh liteアーキテクチャにおけるNetScaler MPXまたはVPXとのEast-west通信

Ingressとして機能するNetScaler MPXまたはVPXは、前のセクションで述べたのと同様の方法で、わずかな変更を加えてEast-westマイクロサービス通信を負荷分散することもできます。以下の手順で、これを実現する方法を示します。

ステップ1: coffeeホスト名をNetScaler MPX/VPX IPアドレスに解決する外部サービスを作成する

これには2つの方法があります。ホスト名をマッピングする外部サービスを追加するか、IPアドレスを使用することができます。

ホスト名によるマッピング (CNAME)

- NetScaler MPXまたはVPXでIngressエンドポイントIPアドレス（コンテンツスイッチング仮想サーバーIPアドレス）のドメイン名を作成し（例: `myadc-instance1.us-east-1.mydomain.com`）、それをDNSサーバーで更新します。
- `coffee` のKubernetesサービスを `externalName` を `myadc-instance1.us-east-1.mydomain.com` として作成します。
- これで、任意のPodが `coffee` マイクロサービスを検索すると、 `CNAME` (`myadc-instance1.us-east-1.mydomain.com`)が返されます。

```
kind: Service

apiVersion: v1
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
metadata:

name: coffee

spec:

type: ExternalName

externalName: myadc-instance1.us-east-1.mydomain.com
```

ホスト名をIPアドレスにマッピングする

アプリケーションでホスト名`coffee`を使用し、それがNetScaler MPXまたはVPXでホストされている仮想IPアドレスにリダイレクトされるようにしたい場合は、以下を作成できます。

```
---

kind: "Service"

apiVersion: "v1"

metadata:

  name: "coffee"

spec:

  ports:

    -

      name: "coffee"

      protocol: "TCP"

      port: 80

---

kind: "Endpoints"

apiVersion: "v1"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
metadata:

  name: "coffee"

subsets:

  -

    addresses:

      -

        ip: "1.1.1.1" # Ingress IP in MPX

    ports:

      -

        port: 80

        name: "coffee"
```

ステップ2: マイクロサービスPod用のヘッドレスサービスを作成する

coffeeサービスをNetScaler MPXを指すように変更したため、coffeeマイクロサービスデプロイメントを表すもう1つのサービスを作成する必要があります。

ステップ3: Ingressリソースを作成する

`ingress.citrix.com/frontend-ip`アノテーションを使用してIngressリソースを作成します。このアノテーションの値は、NetScaler MPXまたはVPXのIngressエンドポイントIPアドレスと一致する必要があります。

これで、NetScaler MPX または VPX を構成して、コーヒーマイクロサービスポッドへの東西トラフィックを制御する Ingress オブジェクトを作成できます。

以下は、Ingress リソースのサンプルです。

サンプル(/en-us/netscaler-k8s-ingress-controller/media/coffee-headless-ingress.png)

これらの変更により、通常の Ingress ロードバランシング手法を使用することで、NetScaler MPX は東西トラフィックをロードバランスできるようになります。次の図は、Ingress ルールを使用して N-S および E-W プロキシとして構成された NetScaler MPX または VPX を示しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サンプル(/en-us/netscaler-k8s-ingress-controller/media/image006.png)

Service Mesh lite でのアプリケーションの自動展開

Service Mesh lite アーキテクチャでアプリケーションを展開するには、複数のタスクを手動で実行する必要があります。ただし、複数のマイクロサービスで構成される複数のアプリケーションを展開したい場合は、Service Mesh lite アーキテクチャでサービスを展開するより簡単な方法があります。NetScaler® は、展開準備ができた YAML を自動的に生成する方法を提供します。

このドキュメント は、NetScaler が提供するスクリプトを使用して、既存の YAML から Service Mesh lite 展開に必要なすべての YAML を生成する方法に関する情報を提供します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler® オペレーター

オペレーターは、OpenShiftネイティブアプリケーションを効果的、自動化され、スケーラブルな方法でパッケージ化、デプロイ、管理するように設計されたオープンソースツールキットです。

NetScaler Operatorのデプロイに関する情報については、NetScaler Operatorのデプロイを参照してください。

NetScaler Operatorを使用して、次のコントローラーをデプロイできます。

- NetScaler Ingress Controllerのデプロイ
- CPXを使用したNetScaler Ingress Controllerのデプロイ
- NetScaler Cloud Controllerのデプロイ
- NetScaler GSLB Controllerのデプロイ
- NetScaler IPAM Controllerのデプロイ
- NetScaler Observability Exporterのデプロイ

リリースノート

NetScaler オペレーター

- 3.6.0
- 3.5.0
- 3.4.1
- 3.3.0
- 3.2.0
- 3.1.2

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler® Operator リリースノート

バージョン 3.6.0

新機能

注:

古い設定から新しい設定への調整中に、設定変更により Citrix CRD の設定でダウンタイムが発生する可能性があります。

NETSCALER IPAM CONTROLLER のサポート

NetScaler Kubernetes Gateway Controller は、独自の NetScaler IPAM コントローラーを使用して、Kubernetes Gateway リソースの IP アドレス割り当てをサポートします。IPAM が有効な場合、`spec.addresses` で静的 IP アドレスが設定されていないゲートウェイは、IPAM コントローラーから仮想 IP (VIP) を自動的に受け取ります。これにより、各ゲートウェイの IP アドレスを手動で割り当てて追跡する必要がなくなります。

詳細については、`<https://docs.netscaler.com/ja-jp/netscaler-kubernetes-gateway-controller/ipam-controller>` を参照してください。

GATEWAY および TCPROUTE/UDPROUTE CRD を使用した TCP/UDP アプリケーションの公開

NetScaler Kubernetes Gateway Controller は、NetScaler とともに Kubernetes Gateway API を使用して TCP/UDP ベースのアプリケーションを公開することをサポートします。

詳細については、NetScaler Kubernetes Gateway Controller の展開 および NetScaler CPX Kubernetes Gateway Controller の展開 を参照してください。

アップグレード時の NETSCALER CPX の構成永続性

NetScaler CPX の構成永続性により、Pod のシャットダウン時に実行中の構成を Kubernetes ConfigMap に保存することで、新しい Pod が稼働するのに必要な時間が大幅に短縮されます。この機能強化により、Helm アップグレード、ローリングアップデート、またはその他の置き換えによって代替 Pod が起動した場合でも、構成が自動的に復元されます。新しい Pod はほとんどの設定が事前にロードされた状態で初期化されるため、トラフィックを処理する準備が整うまでの遅延が最小限に抑えられます。

以前は、NetScaler CPX Pod が再起動するたびに、イングレスおよびサービスリソースを通じて蓄積された実行中の構成が失われていました。このため、NetScaler Ingress Controller (NSIC) は、NITRO API を介してすべてのリソースを最初から再プッシュする必要がありました。大規模

© 1997–2026 Citrix Systems, Inc. All rights reserved.

なイングレス展開では、この調整プロセスにより、NetScaler CPX がトラフィックを管理する準備が完全に整うまでに長い遅延が発生していました。

詳細については、`<https://docs.netscaler.com/ja-jp/netscaler-k8s-ingress-controller/accelerate-cpx-upgrade-process>` を参照してください。

OWASP CRDを使用してOWASP TOP 10保護ポリシーを構成する

NetScaler Ingress Controllerは、OWASP CRDを使用してOWASP Top 10保護ポリシーの構成をサポートするようになりました。OWASP Top 10:2025は、Webアプリケーションに対する最も重大なセキュリティリスクを表しています。OWASP CRD (owasppolicy) を使用すると、単一のKubernetesカスタムリソースで、OWASP Top 10カテゴリに合わせた包括的なWAFおよびボット管理保護を構成できます。

詳細については、`<https://docs.netscaler.com/ja-jp/netscaler-k8s-ingress-controller/crds/owasp-top-10-protection-policies>`を参照してください。

IPAMコントローラーおよびイングレスコントローラーとのBIND DNS統合

BIND 9は、DNSプロトコルの完全な実装です。BIND 9は、権威ネームサーバー、リゾルバー、およびサポートされているホストではスタブリゾルバーとして構成できます。詳細については、Bind 9を参照してください。

NetScaler Ingress Controllerは、ipam-rangeアノテーションを使用して、IngressおよびLoadBalancerタイプのServiceにIPを割り当てることをサポートしています。IPAMコントローラーには、NetScalerが提供するVIP CustomResourceDefinition (CRD) が必要です。VIP CRDは、NetScaler Ingress ControllerとIPAMコントローラー間の内部通信に使用されます。このリリースから、VIP CRDに変更があります。VIP CRDは、VIPに関連付けられているhostnameフィールドをサポートするようになりました。

詳細については、`<https://docs.netscaler.com/ja-jp/netscaler-k8s-ingress-controller/bind-dns-integration>`を参照してください。

NETSCALER INGRESS CONTROLLERの再起動中のNETSCALERリソース構成の最適化

NetScaler Ingress Controller (NSIC) は、運用上の遅延を最小限に抑えるために最適化された起動シーケンスをサポートするようになりました。この機能により、再起動またはアップグレード時のNetScalerリソースの構成が高速化され、コントローラーがより迅速に実行状態に到達できるようになります。

NetScalerノードコントローラーを使用したOpenShiftのOVN-CNIサポート ノードコントローラーとルーターポッドは、OpenShift 4.xクラスターのデフォルトCNIであるOVN-Kubernetesをサポー

© 1997–2026 Citrix Systems, Inc. All rights reserved.

トしています。OpenShiftノードとCitrix ADCの間で、ovn-k8s-mp0インターフェイスを使用してVXLANオーバーレイネットワークが確立されます。

詳細については、`<https://docs.netscaler.com/ja-jp/netscaler-k8s-ingress-controller/network/ovn-kubernetes-cni-support>`を参照してください。

GTP CRDでの`INGRESSHOSTNAME`、`MULTIPLEIPRESPONSE`、および`EMPTYDOWNRESPONSE`のサポート

- `ingressHostName` — globaltrafficpolicy.spec.hostsの下にある新しいオプションフィールドを使用すると、GTPのホスト (DNS) フィールドとは独立して、Ingressをそのルールホスト名で照合できるため、単一のGSLBドメインを異なるホスト名を公開するIngressでバックアップできます。
- `multipleIPResponse` (MIR) および `emptyDownResponse` (EDR) — 新しいオプションの[ポリシーフィールド multipleIPResponse (MIR) および `emptyDownResponse (EDR) は、自動作成されたGSLB仮想サーバー上の対応する-MIR/-EDRパラメーターに伝播され、CRDから直接マルチIP DNS応答と空ダウン動作を有効にします。

詳細については、`<https://docs.netscaler.com/ja-jp/netscaler-k8s-ingress-controller/gslb/gslb#gtp-crd>`を参照してください。

CRDのエンティティ名の変更

NetScalerカスタムリソース定義 (CRD) インスタンスが作成されると、NetScaler Ingress ControllerはそのCRDインスタンスに関連付けられた複数のNetScalerエンティティを生成します。NetScaler Ingress Controllerは、CRDインスタンスとの関連付けを維持するために、各エンティティに一意の名前を保持します。エンティティの命名はCRD名に直接基づいているため、一部のNetScalerエンティティ名が最大文字数制限を超えていました。

NetScaler Ingress Controller 4.0.16以降、CRD作成時に短いエンティティ名を生成するために、以下の方法を使用して命名規則が最適化されました。

- ハッシュ化された命名: エンティティ名の一部がハッシュ化され、全体の長さが短縮されます。
- 保持される情報: NetScalerがエンティティコメントをサポートしている場合、必要なKubernetes関連のメタデータはエンティティのコメントフィールドに保持されます。
- 互換性の向上: 名前はNetScalerの文字制限に準拠しつつ、完全なトレーサビリティを維持します。詳細については、`<https://docs.netscaler.com/ja-jp/netscaler-k8s-ingress-controller/entity-name-change-for-crds>`を参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

GSLBコントローラーの改善

GSLBコントローラーは、以下の改善を含むように強化されました。

- TCP、UDP、SSLなど、すべてのIngressリソースタイプに対するGSE自動作成のサポートが追加されました。
- GslbConfigSyncMonitorは、GSLBサイトの監視効率を向上させるため、マスターGSLBノードでデフォルトで有効になりました。詳細については、`<https://docs.netscaler.com/ja-jp/netscaler-k8s-ingress-controller/gslb/gslb>`を参照してください。

SERVICETYPELB: スマートアノテーションのイベント変更

NetScaler Ingress Controllerリリース4.0.16以降、ServiceTypeLBで以下のいずれかのアノテーションを変更した場合、NetScaler Ingress ControllerはNetScalerで設定を削除して再作成するのではなく、設定を変更します。

```
"service.citrix.com/lbvserver",

"service.citrix.com/csvserver",

"service.citrix.com/servicegroup",

"service.citrix.com/monitor",

"service.citrix.com/analyticsprofile",

"service.citrix.com/insecure-redirect",

"service.citrix.com/secret",

"service.citrix.com/preconfigured-certkey",

"service.citrix.com/ca-secret",

"service.citrix.com/preconfigured-ca-certkey",

"service.citrix.com/backend-secret",

"service.citrix.com/preconfigured-backend-certkey",

"service.citrix.com/backend-ca-secret",

"service.citrix.com/preconfigured-backend-ca-certkey"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
'service.citrix.com/ssl-termination-',

'service.citrix.com/frontend-tcpprofile-',

'service.citrix.com/backend-tcpprofile-',

'service.citrix.com/frontend-httpprofile-',

'service.citrix.com/backend-httpprofile-',

'service.citrix.com/frontend-sslprofile-',

'service.citrix.com/backend-sslprofile-'
```

NETSCALERマルチクラスターINGRESSデプロイメントにおけるSSLパススルーのサポート

SSLパススルー機能を使用すると、受信するセキュアソケットレイヤー (SSL) リクエストを、ロードバランサーを使用して復号化するのではなく、復号化のためにサーバーに直接渡すことができます。SSLパススルーはWebアプリケーションのセキュリティに広く使用されており、TCPモードを使用して暗号化されたデータをサーバーに渡します。

NetScaler Ingress Controller 4.0.16以降、NetScalerマルチクラスターIngressデプロイメントでSSLパススルー機能がサポートされます。詳細については、`<https://docs.netscaler.com/ja-jp/netscaler-k8s-ingress-controller/configure/ssl-passthrough-multicluster>`を参照してください。

修正された問題 - NetScaler Kubernetes Gateway Controller

WebSocketはデフォルトで無効になっています。このユースケースには、`HTTPProfile` CRDを使用できるようになりました。

プロファイルで`GatewayClass`が提供された場合に、`HTTPProfile` CRDコントローラーで問題が確認されました。プロファイルで提供された`GatewayClass`名が、属性'ref_manager'がないというエラーを引き起こす可能性があります。

ホスト名が同じ親ドメインを共有している場合、ワイルドカードよりも正しい優先順位が適用されません。この問題の修正として、同じコンテンツスイッチング仮想サーバー上の重複するホスト名のルーティング優先度が強化されました。

`RequestHeaderModifier.set`は、ヘッダーが既に存在する場合にのみ機能します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

`URLRewrite.replacePrefixMatch`は二重スラッシュを生成するため、パスの正規化が正しく処理されません。

名前が20文字まで同じ2つのHTTPルートが、2つではなく1つのロードバランシング仮想サーバーを作成します。ロードバランシング仮想サーバーの命名に競合が確認されました。

コントローラーの再起動後、またはHTTPルートの後にサービスイベントが受信された後、ロードバランシング仮想サーバーのサービスグループバインディングが失われます。

Kubernetes APIへのGET呼び出し中に、`429 errors`が確認されました。

修正された問題 - NetScaler Ingress Controller

- イングレスに複数のサービスが記述されている場合、カスタムモニターの作成は失敗し、モニターはそのうちの1つに対してのみ作成されます。
- CRUD操作中に、ボットCRD内の複数のボットプロファイルパラメーターが見落とされます。
- エラーのログ記録中に、不適切な例外処理が確認されました。
- NetScaler Ingress Controllerポッドが再起動すると、コントローラーは、コントローラーがダウンしているときに削除されたイングレスの構成を削除できません。
- 名前が20文字まで同じ2つのHTTPルートが、2つではなく1つのロードバランシング仮想サーバーを作成します。
- NetScaler Ingress Controllerは、IPAMがコンテンツスイッチング仮想サーバーのIPアドレスを提供する前であっても、存在しないサービスグループにサービスグループメンバーをバインドしようとします。
- NetScaler Ingress Controllerは、IPAMがコンテンツスイッチング仮想サーバーのIPアドレスを提供する前であっても、存在しないサービスグループにサービスグループメンバーをバインドしようとします。
- 複数のPodが同じPod IPアドレスを共有している場合、1つのPodを削除すると、NetScaler上のアクティブなPodのエンドポイント状態が削除される可能性があります。場合によっては、これによりNetScaler Kubernetes Ingress Controllerがキーエラーでクラッシュする可能性があります。
- ハイフン (-) を含むKubernetesエンティティ (名前空間、サービスなど) は、authpolicyやpatsetなどのNetScalerエンティティの構成の失敗を引き起こしました。これは、これらのエンティティ名がNetScaler上でハイフンを許可しないためです。
- NSIC/GSLBコントローラーは、負荷がかかっているKubernetes APIサーバーからHTTP 429 (Too Many Requests) エラーを受信する可能性があり、その結果、一部の監視イベントが見逃されることになりました。Helmチャートには、専用のFlowSchemaと

© 1997–2026 Citrix Systems, Inc. All rights reserved.

PriorityLevelConfiguration (API Priority and Fairness) が同梱されるようになり、NSICのAPIトラフィックには予約された同時実行共有が与えられ、他のクライアントと並んでレート制限されることがなくなります。

NetScaler® Operator リリースノート

バージョン 3.5.0

新機能

修正された問題

- NetScaler Ingress Controller の構成における SSL、HTTP、および TCP の不適切なデフォルトプロファイル形式を解決しました。
- NetScaler IPAM Controller における Infoblox グリッドホスト設定を修正しました。

NetScaler® Operator リリースノート

バージョン 3.4.1

新機能

LAS ライセンスサポート

以下のパラメータを設定することで、Kubernetess Gateway Controller を使用する CPX および Ingress Controller を使用する NetScaler CPX で、NetScaler CPX 用に LAS ライセンス方式を使用できるようになりました。

```
spec:

  admSettings:

    admIP: ''

    bandWidth: 1000

    licenseEdition: PLATINUM

    licensingMode: las

    loginSecret: ''
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NETSCALER INGRESS CONTROLLER/GSLB CONTROLLER バージョン 3.4.4 のサポート

NetScaler Operator バージョン 3.4.1 は、NetScaler Ingress Controller/GSLB Controller バージョン 3.4.4 をサポートしています。

NETSCALER KUBERNETES GATEWAY CONTROLLER バージョン 1.4.0 のサポート

NetScaler Operator バージョン 3.4.1 は、NetScaler Kubernetes Gateway Controller バージョン 1.4.0 をサポートしています。

NetScaler® Operator リリースノート

バージョン 3.3.0

新機能

NETSCALER キューバネティス ゲートウェイ コントローラー のサポート

NetScaler Operator を使用して NetScaler Kubernetes Gateway Controller を展開できるようになりました。詳細については、Deploy NetScaler Kubernetes Gateway Controller in OpenShift using NetScaler Operator を参照してください。

NETSCALER KUBERNETES GATEWAY CONTROLLER と併用した NETSCALER CPX のサポート

NetScaler Operator を使用して、NetScaler Kubernetes Gateway Controller と併用する NetScaler CPX を展開できるようになりました。詳細については、Deploy NetScaler Kubernetes Gateway Controller with CPX in OpenShift using NetScaler Operator を参照してください。

カスタムサービスアカウント

オペレーターを使用して展開された NetScaler Ingress Controller/GSLB Controller および NetScaler Kubernetes Gateway Controller で、`serviceAccount.name` 変数を使用してカスタムサービスアカウント名を指定できるようになりました。サービスアカウントを自動的に作成するには、`serviceAccount.create` を `true` に設定します。`serviceAccount.create` が `false` に設定されている場合は、指定された `serviceAccount.name` が適切な `clusterRole` にバインドされていることを確認してください。

`createClusterRoleAndBinding` 変数が導入され、これを `true` に設定すると、`ClusterRole` と `ClusterRoleBinding` が自動的に作成されます。変数が `false` に設定されている場合、`ClusterRole` と `ClusterRoleBinding` は作成されません。

NETSCALER INGRESS CONTROLLER を使用した NETSCALER における証明書キーバンドルのサポート

NetScaler Ingress Controller/NetScaler Kubernetes Gateway Controller は、NetScaler のリリース 14.1 ビルド 21.x 以降でサポートされている証明書バンドル (`certkeybundle`) 機能をサポート

© 1997–2026 Citrix Systems, Inc. All rights reserved.

するようになりました。この機能により、証明書チェーンの問題や、2つの証明書が中間 CA を共有する場合に必要となる追加の処理が解決されます。NetScaler における証明書キーバンドルのサポートの詳細については、Support for SSL certificate key bundle を参照してください。

この機能セットを有効にするために、Ingress Controller、NetScaler Kubernetes Gateway Controller、NetScaler Kubernetes Gateway Controller と併用する NetScaler CPX、および Ingress Controller と併用する NetScaler CPX のオペレーターに `certBundle` が追加されました。

NetScaler® Operator リリースノート

バージョン 3.2.0

NetScaler IPAM Controller のサポート

NetScaler Operator を使用して、NetScaler IPAM Controller を展開できるようになりました。詳細については、「OpenShift に NetScaler Operator を使用して NetScaler IPAM Controller を展開する」を参照してください。

NetScaler GSLB Controller を使用した GSLB サイト間の通信のセキュリティ強化

NetScaler GSLB Controller の展開では、GSLB Controller が GSLB デバイスに接続して構成をプッシュするために使用するシークレットを作成する際に、追加のキー `sitesyncpassword` がサポートされます。このキーは、GSLB サイト間の通信のセキュリティを強化します。詳細については、「NetScaler GSLB Controller を展開する」を参照してください。

`UpgradeImageAlways` パラメータのサポート

NetScaler Operator は `UpgradeImageAlways` パラメータをサポートするようになりました。このパラメータが `True` に設定されている場合、オペレーターの更新中にコントローラーのイメージがアップグレードされます。ユーザーがオペレーターによって提供される現在のバージョンとは異なるイメージを使用したい場合は、このパラメータを `False` に設定する必要があります。

NetScaler® Operator リリースノート

バージョン 3.1.2

新機能

GSLB CONTROLLER の展開のサポート

NetScaler Operator を使用して GSLB Controller を展開できるようになりました。詳細については、「Deploy NetScaler GSLB Controller in OpenShift using NetScaler Operator」を参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

`MULTICLUSTERPREFIX` パラメーターの構成のサポート

NetScaler Ingress Controller の展開 YAML で `multiClusterPrefix` パラメーターを構成できるようになりました。マルチクラスターイングレスソリューションの設定方法については、「Multi-cluster ingress」を参照してください。

NETSCALER INGRESS CONTROLLER での NETSCALER 証明書の検証

NetScaler Ingress Controller で NetScaler 証明書の検証を有効にできるようになりました。詳細については、「Enable NetScaler certificate validation in NetScaler Ingress Controller」を参照してください。

インターネットコンテンツ適応プロトコル (ICAP) のサポート

ICAP を使用してリモートコンテンツ検査を構成できるようになりました。

詳細については、「Remote content inspection or content transformation service using ICAP」を参照してください。

イングレスリソースの `ADDRESS` フィールドが更新されます

NetScaler CPX が ClusterIP、NodePort、または LoadBalancer タイプのサービスを使用して公開される NSIC サイドカー展開では、展開 YAML で `updateIngressStatus` を `true` として指定すると、イングレスの Address (`Status.LoadBalancer.IP`) フィールドが更新されます。

詳細については、「ingress status update for sidecar deployments」を参照してください。

修正された問題

- Azure KubernetesでKubernetes API認証のために生成されたサービスアカウントトークンは、1時間ごとに期限切れになります。その結果、コントローラーは定期的に再起動し、新しく生成されたトークンを取得して認証を継続します。

NetScaler® Operator のアップグレード

NetScaler Operator を以前のバージョンからバージョン 3.1.2 にアップグレードするには、以下の手順を実行します。

1. 以下のコマンドを使用して、NSIC、GSLB、および NSOE など、インストールされているすべてのコントローラーの YAML をバックアップします。

```
for crd in $(oc get crds -o jsonpath='{.items[*].metadata.name}' | tr '
' '\n' | grep 'netscaler.com'); do echo "Getting instances for CRD:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
$crd"; oc get $crd --all-namespaces -o yaml >>
"all_controller_crd_instances.yaml"; done
```

2. 以下のコマンドを使用して、NSIC および GSLB コントローラーによって管理されている `citrix.com` CRD インスタンスを YAML ファイルにバックアップします。

```
for crd in $(oc get crds -o jsonpath='{.items[*].metadata.name}' | tr '
' '\n' | grep 'citrix.com'); do   echo "Getting instances for CRD:
$crd";   kubectl get $crd --all-namespaces -o yaml >>
"all_citrix_crd_instances.yaml"; done
```

3. 以下のコマンドを使用して、`citrix.com` CRD を削除します。

```
$(oc get crds -o jsonpath='{.items[*].metadata.name}' | tr ' ' '\n' |
grep 'citrix.com'); do oc delete crd $crd; done
```

4. NetScaler Operator 内のすべてのコントローラーインスタンスを削除します。

重要

オペレーター削除ワークフローの一部として表示される **このオペレーターのすべてのオペランドインスタンスを削除** オプションは、場合によっては機能しないことがあり、一部の残存コンポーネントが削除されない場合があります。競合を避けるために、ここに記載されている手順に従ってください。

a. **Operators > Installed Operators** に移動し、**NetScaler Operator** をクリックします。

b. **すべてのインスタンス** をクリックして、すべてのコントローラーインスタンスを一覧表示します。

c. 各コントローラーインスタンスを削除するには、コントローラーの横にある省略記号アイコンをクリックし、次に **削除** をクリックします。

以下の例では、NetScaler Ingress Controller インスタンスを持つ NetScaler CPX が削除されます。

d. **削除** をクリックして削除を確定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

5. 次のコマンドを使用して、NetScaler Operator (NSIC、NSIC+CPX、およびNSOE) の **netscaler.com** CRDを削除します。

```
for crd in $(oc get crds -o jsonpath='{.items[*].metadata.name}' | tr '
' '\n' | grep 'netscaler.com'); do echo "Deleting CRD: $crd" oc delete
crd $crd done
```

6. NetScaler Operatorをアンインストールするには、**Operators > Installed Operators**に移動し、NetScaler Ingress Controller Operatorの横にある省略記号アイコンをクリックし、**Uninstall Operator**をクリックします。

7. **Uninstall**をクリックしてアンインストールを確認します。

8. ステップ1でバックアップしたコントローラーYAMLを、NetScaler Operator 3.1.2で導入された新しいスキーマに変換します。そのためには、コントローラーYAMLの各パラメーターについて、新しいスキーマで対応する変更されたパラメーターを確認し、それに応じてYAMLを更新します。NetScaler Operatorバージョン3.1.2のパラメーターについては、NetScaler Operatorを参照してください。
9. NetScaler Operatorを展開します。NetScaler Operatorの展開については、Deploy NetScaler Operatorを参照してください。
10. ステップ8で更新したYAMLを使用して、コントローラーを順次インストールします。詳細については、NetScaler Operatorを参照してください。
11. 次のコマンドを使用して、ステップ2でバックアップしたCRDをインストールします。

```
oc apply -f all_citrix_crd_instances.yaml
```

NetScaler Operator をデプロイする

次の手順を実行します。

1. OpenShift クラスターコンソールにログインします。
2. **Operators > OperatorHub** に移動し、左側のパネルで **Certified** ソースを選択し、**NetScaler® Operator** を選択し、次に **Install** をクリックします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Operator をインストールする(/en-us/netscaler-k8s-ingress-controller/media/install-netscaler-operator.png)

3. NetScaler Operator をサブスクライブするには、次のいずれかのオプションを選択します。
   - **クラスター上のすべての名前空間 (デフォルト)**: NetScaler Operator は、OpenShift クラスター上のすべての名前空間で利用できます。したがって、このオプションを使用すると、クラスター上の任意の名前空間から NetScaler インスタンスを起動できます。
   - **クラスター上の特定の名前空間**: NetScaler Operator は、OpenShift クラスター上の選択された名前空間で利用できます。したがって、このオプションを使用すると、選択された名前空間でのみ NetScaler Operator インスタンスを起動できます。
4. この場合、**クラスター上の特定の名前空間** を選択し、次に **Install** をクリックします。

   NetScaler Operator のサブスクリプション(/en-us/netscaler-k8s-ingress-controller/media/netscaler-operator-subscription.png)

   NetScaler Operator が正常にサブスクライブされるまで待ちます。

   NetScaler Operator のインストール確認(/en-us/netscaler-k8s-ingress-controller/media/netscaler-operator-confirm.png)
5. (オプション) すべての名前空間に NetScaler Operator をデプロイするには、インストールモードとして `All namespaces on the cluster (default)` が選択され、インストールされた名前空間として `openshift-operators` が選択されていることを確認し、次に **Install** をクリックします。

   注記:

   すべてのネームスペースにNetScaler Operatorをデプロイする前に、既存のNetScaler Operatorインスタンスをすべて削除してください。

   - [ ] NetScaler Operatorが正常にサブスクライブされるまで待ちます。
6. **Workloads > Pods**セクションに移動し、**netscaler-Operator-controller-manager**ポッドが稼働していることを確認します。

- [ ] NetScaler Operatorを使用して、以下のコントローラーをデプロイできます。

- NetScaler Ingress Controllerをデプロイする

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- CPXでNetScaler Ingress Controllerをデプロイする
- NetScaler Cloud Controllerをデプロイする
- NetScaler GSLB Controllerをデプロイする
- NetScaler Observability Exporterをデプロイする

NetScaler Operator を使用して OpenShift に NetScaler Ingress Controller を展開する

このセットアップでは、NetScaler Ingress Controller は OpenShift クラスターの外部にある NetScaler MPX または NetScaler VPX を構成します。

前提条件

- Red Hat OpenShift Cluster (バージョン 4.1 以降)。
- NetScaler Operator を展開します。NetScaler Operator の展開方法については、NetScaler Operator の展開 を参照してください。
- NetScaler Ingress Controller が NetScaler と通信するために必要な IP アドレスを特定します。この IP アドレスは、NetScaler の展開タイプに応じて、次のいずれかの IP アドレスである可能性があります。
  - NSIP (スタンドアロンアプライアンスの場合) - スタンドアロン NetScaler アプライアンスの管理 IP アドレスです。詳細については、NetScaler の IP アドレス指定 を参照してください。
  - SNIP (高可用性モードのアプライアンスの場合) - サブネット IP アドレスです。詳細については、NetScaler の IP アドレス指定 を参照してください。
  - CLIP (クラスターモードのアプライアンスの場合) - クラスター化された NetScaler 展開のクラスター管理 IP (CLIP) アドレスです。詳細については、クラスターの IP アドレス指定 を参照してください。
- イングレスデバイスとして使用される NetScaler VPX または NetScaler MPX のユーザー名とパスワード。NetScaler Ingress Controller が NetScaler VPX または NetScaler MPX を構成できるように、NetScaler には特定の権限を持つシステムユーザーアカウント (非デフォルト) が必要です。NetScaler でシステムユーザーアカウントを作成する手順については、NetScaler で NetScaler Ingress Controller 用のシステムユーザーアカウントを作成する を参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ユーザー名とパスワードを環境変数としてコントローラーに直接渡すか、OpenShift シークレットを使用できます (推奨)。次のコマンドを使用してユーザー名とパスワードのシークレットを作成するには、`<username>` と `<password>` を必要な値に変更します。

```
oc create secret  generic nslogin --from-literal=username=<username> --
from-literal=password=<password>
```

- 任意のサービスモニターにエクスポートするには、次のコマンドを使用してサービスモニター CRD をインストールします。

```
oc create -f https://raw.githubusercontent.com/prometheus-operator/
prometheus-operator/refs/heads/main/example/prometheus-operator-crd/
monitoring.coreos.com_servicemonitors.yaml
```

NetScaler Operator を使用して NetScaler Ingress Controller をスタンドアロンポッドとして展開する

NetScaler Operator を使用すると、NetScaler Ingress Controller を OpenShift クラスター内のスタンドアロンポッドとして展開できます。NetScaler Ingress Controller は、OpenShift クラスターで実行されているアプリケーションのイングレスデバイスまたはルーターとして展開される NetScaler MPX または NetScaler VPX を構成します。次の図はトポロジを示しています。

次の手順を実行します。

1. OpenShift 4.x クラスターコンソールにログインします。
2. コンソールを使用して Apache アプリケーションを展開します。

   a. **Workloads > Deployments > Create Deployment** に移動し、次の YAML ファイルを使用してデプロイを作成します。

   注：

   Apache アプリケーションはデモンストレーションのみを目的としています。要件に基づいて YAML ファイルを変更できます。

```
---

apiVersion: apps/v1
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kind: Deployment
metadata:
  name: apache
  labels:
      name: apache
spec:
  selector:
    matchLabels:
      app: apache
  replicas: 2
  template:
    metadata:
      labels:
        app: apache
    spec:
      containers:
      - name: apache
        image: httpd:latest
        ports:
        - containerPort: 80
    ---
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
```

b. **Workloads > Pods** セクションに移動し、Apache アプリケーションのポッドが稼働していることを確認します。

3. Apache アプリケーションのサービスを作成します。**Networking > Services > Create Service** に移動し、次の YAML ファイルを使用します。

```
apiVersion: v1

kind: Service

metadata:

  name: apache

spec:

  ports:

  - port: 80

    targetPort: 80

  selector:

    app: apache
```

4. Apache アプリケーションのイングレスを作成します。**Networking > Ingress > Create Ingress** に移動し、次の YAML を使用してイングレスを作成します。クラスターに適用する前に、イングレス YAML で NetScaler VPX の VIP アドレスを更新してください。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
annotations:
  ingress.citrix.com/frontend-ip: <NSVIP>
name: vpx-ingress
spec:
  ingressClassName: nsic-vpx
  rules:
  - host: citrix-ingress-operator.com
    http:
      paths:
      - backend:
          service:
            name: apache
            port:
              number: 80
        path: /
        pathType: Prefix
```

5. **Operators > Installed Operators** に移動し、**NetScaler® Operator** をクリックします。
6. **NetScaler NetScaler Ingress Controller** タブをクリックし、**Create NetScalerIngressController** を選択します。

   NetScaler NetScaler Ingress Controller の作成(/en-us/netscaler-k8s-ingress-controller/media/create-cic-instance.png)

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler NetScaler Ingress Controller の YAML 定義が表示されます。必要に応じて、`Form view` ボタンを選択し、フォームでパラメータを更新できます。

パラメータリスト(/en-us/netscaler-k8s-ingress-controller/media/nic-instance-yaml.png)

7. インストール中に構成できる必須およびオプションのパラメータを一覧表示する以下の表を参照してください。

注記：

- `acceptLicense` パラメータが `Yes` に設定されていることを確認してください。
- `entityPrefix` に一意の値を指定します。
- `nsIP` パラメータには NetScaler VPX インスタンスの IP アドレスを、`adcCredentialSecret` パラメータには NetScaler VPX 資格情報を使用して作成された Kubernetes シークレットを指定します。
- NetScaler Ingress Controller をこの手順で指定されたイングレスリソースに関連付けるには、`ingressClass` パラメータを使用してイングレスクラス名を `ingressClass: [‘NSIC-vpx’]` として指定します。

注記：

必要に応じて、`form view` をクリックして、必要に応じてパラメータを設定できます。

ユースケースに応じて、他の利用可能なパラメータを構成できます。

パラメータ	必須またはオプション	デフォルト値	説明
acceptLicense	必須	いいえ	NetScaler Ingress Controller EULA に同意するには、この値を `Yes` に設定します。
アフィニティ	オプション	N/A	Pod の割り当て用アフィニティラベル。
extraVolumes	オプション	N/A	マウントする追加の VolumeMounts。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須またはオプション	デフォルト値	説明
imagePullSecrets	オプション	N/A	プライベートDockerレジストリまたはリポジトリからイメージをプルするために使用されるKubernetesシークレットのリスト。このシークレットの作成方法の詳細については、「プライベートレジストリからイメージをプルする」を参照してください。
nodeSelector	オプション	N/A	NetScaler Ingress Controllerのデプロイでで nodeSelectorオプションに使用されるノードラベルのキーと値のペア。
podAnnotations	オプション	N/A	ポッドに追加するアノテーションのマップ。
トレランス	オプション	N/A	NetScaler Ingress Controllerのデプロイに対するtolerations。

パラメータ	必須/任意	デフォルト値	説明
analyticsConfig			
distributedTracing.enable	任意	フォールス	NetScalerでOpenTracingを有効にするには、この値を `true` に設定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
distributedTracing.samplingrate	任意	100	OpenTracingのサンプリングレート（パーセンテージ）。
必須	必須	フォールス	NetScalerがメトリクスとトランザクションレコードをアナリティクスサーバーに送信するように設定する場合、この値を `true` に設定します。
endpoint.metrics.service	オプション	N/A	アナリティクスサーバーのIPアドレスまたはDNSアドレス。形式: `servicename.namespace`, `servicename.namespace.svc.cluster.local`, `namespace/servicename`。
endpoint.transactions.service	オプション	N/A	Kubernetes環境にデプロイされたアナリティクスサービスのネームスペースを持つIPアドレスまたはサービス名。形式: `namespace/servicename`。
timeseries.port	オプション	30002	時系列エンドポイントのために、クラスター外にアナリティクスサービスを公開

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			するために使用されるポート。
timeseries.auditlogs.enable	オプション	いいえ	NetScalerから監査ログデータをエクスポートするには、この値を`true`に設定します。
timeseries.events.enable	任意	いいえ	NetScalerからイベントをエクスポートするには、この値を`true`に設定します。
timeseries.metrics.enable	任意	いいえ	NetScalerからメトリクスを送信できるようにするには、この値を`true`に設定します。
timeseries.metrics.mode	任意	アヴロ	メトリックエンドポイントのモード。
timeseries.metrics.exportFrequency	任意	30	時系列データのエクスポート時間間隔。設定可能な値は30秒から300秒です。
timeseries.metrics.schemaFile	任意	schema.json	メトリクスコレクターに追加および設定する必要があるNetScalerカウンターを含むスキーマファイルの名前。サポートされているすべてのカウンターを含む参照スキーマファイル`reference_schema.json`は、パス`/`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			`var/metrics_conf/` にもあります。このスキーマファイルは、カスタムカウンターリストを作成するための参照として使用できます。
timeseries.metrics.enableNativeScrape	任意	フォールス	メトリクスのネイティブエクスポートには、この値を `true` に設定します。
transactions.port	任意	30001	トランザクションエンドポイントのために、クラスター外にアナリティクスサービスを公開するために使用されるポート。
transactions.enable	任意	フォールス	NetScalerからトランザクションをエクスポートするには、この値を `true` に設定します。

パラメータ	必須/任意	デフォルト値	説明
エクスポーター			
extraVolumeMounts	任意	該当なし	エクスポーターコンテナにマウントする追加のVolumeMounts。
イメージ	任意	quay.io/netscaler/netscaler-adc-metrics-	Quay.ioでホストされているメトリクスエ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
		exporter@sha`<sha value of the latest release>`	クスポーターイメージ。
必須	オプション	フォールス	`true`に設定すると、NetScaler MPXまたはNetScaler VPXのメトリクスをプルするために、NetScaler Ingress Controllerとともに Exporter for NetScaler Statsが実行されます。
pullPolicy	オプション	IfNotPresent	エクスポーターイメージのプルポリシー。
ports.containerPort	オプション	8888	エクスポーターコンテナポート。
リソース	任意	該当なし	メトリクスエクスポーターコンテナのCPU/メモリリソース。
serviceMonitorExtraLabels	任意	該当なし	エクスポーターが有効な場合のサービスモニターの追加ラベル。

パラメータ	必須/任意	デフォルト値	説明
ingressController			
clusterName	任意	該当なし	NetScaler Ingress Controllerがデプロイされている Kubernetesクラス

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			ターの一意の識別子。
defaultSSLCertSecret	オプション	該当なし	NetScalerでデフォルトの非SNI証明書として使用する必要があるKubernetesシークレット名。
defaultSSLSNICertSecret	オプション	該当なし	NetScalerでデフォルトのSNI証明書として使用する必要があるKubernetesシークレット名。
disableAPIServerCertVerify	オプション	フォールス	APIサーバー証明書の検証を無効にするには、このパラメーターを`True`に設定します。
disableOpenshiftRoutes	オプション	いいえ	デフォルトでは、OpenShiftルートはOpenShift環境で処理されます。このパラメータを使用して、NetScaler Ingress ControllerがOpenShiftルートを処理するのを無効にできます。
enableLivenessProbe	オプション	はい	NetScaler Ingress Controllerのライブネスプローブ設定を無効にするには、`false`に設定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
enableReadinessProbe	オプション	はい	NetScaler Ingress Controllerのレディネスプローブ設定を無効にするには、`false`に設定します。
entityPrefix	オプション	N/A	NetScaler MPXまたはNetScaler VPX上のリソースのプレフィックス。
ignoreNodeExternalIP	任意	偽	LoadBalancerサービスまたはNodePortサービスのサービスグループメンバーとしてノードIPアドレスを追加する場合、NetScaler Ingress Controllerには選択基準があり、利用可能な場合は`Node ExternalIP`を、`Node ExternalIP`が利用できない場合は`Node InternalIP`を選択します。しかし、一部のユーザーは、Node ExternalIPが存在する場合でも、Node ExternalIPよりもNode InternalIPを使用したいと考えるかもしれません。この変数が`true`に設定されている場合、NSICはNode ExternalIPが存在する場合でも、サービスグループメンバー

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			に使用するNode InternalIPを優先します。
イメージ	必須	`quay.io/netscaler/netscaler-k8s-ingress-controller@sha value of latest release`	Quay.ioでホストされているNetScaler Ingress Controllerイメージ。
ingressClass	任意	該当なし	特定のイングレスリソースをNetScaler Ingress Controllerに関連付けるためのパラメータ。イングレスクラスの詳細については、「Ingress class support」を参照してください。Kubernetesバージョン1.19以降の場合、このパラメータはここで指定された名前でIngressClassオブジェクトを作成します。
extraVolumeMounts	任意	該当なし	NetScaler Ingress Controllerコンテナにマウントされる追加のVolumeMounts。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
IPAM	任意	フォールス	IPAMコントローラーを使用してLoadBalancerタイプのサービスにIPアドレスを自動的に割り当てるには、このパラメーターを`true`に設定します。
jsonLog	任意	フォールス	ログメッセージがJSON形式で必要な場合は、この引数をtrueに設定します。
kubernetesURL	任意	N/A	NetScaler Ingress Controllerがイベントを登録するために使用するkube-apiserverのURL。値が指定されていない場合、NetScaler Ingress Controllerは内部kube-apiserver IPアドレスを使用します。
livenessProbe	任意	N/A	NetScaler Ingress Controller のライブネスプローブ設定。
logLevel	オプション	INFO	NetScaler Ingress Controller によって生成されるログを制御するログレベル。サポートされているログレベルは、`CRITICAL`、`ERROR`、`WARNING`、`INFO`、`DEBUG`、`TRACE`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			、および `NONE` です。
logProxy	オプション	N/A	NetScaler 可観測性エクスポーター用のElasticsearch、Kafka、またはZipkin エンドポイント。
multiClusterPrefix	オプション	mc	マルチクラスターイングレス機能のために、NetScaler MPX または NetScaler VPX 上の共有リソースに設定するプレフィックス。コンテンツスイッチング仮想サーバーのIPアドレスを共同で共有するNetScaler Ingress Controllerは、multiclusterPrefixに同じ値を設定する必要があります。詳細については、こちらを参照してください。
namespaceLabels	オプション	N/A	OpenShiftクラスターでrouteShardingのためにNetScaler Ingress Controllerが使用する名前空間ラベルセレクターを指定します。
nodeLabels	任意	""	ノードラベルを持つノードにPodがある場合、NetScaler Ingress Controllerは

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			NetScalerを設定して、BGPを使用してVIPをアドバタイズします。
nodeWatch	任意	フォールス	NetScaler MPXまたはNetScaler VPXからKubernetesクラスター内のPodへのネットワークルートを自動的に構成したい場合は、パラメーターを`true`に設定します。詳細については、「NetScalerインスタンスでルートを自動的に構成する」を参照してください。
nsncPbr	任意	フォールス	NetScaler Node ControllerがNetScalerでポリシーベースルーティング (PBR) を構成していることをNetScaler Ingress Controllerに通知するには、このパラメーターを`true`に設定します。詳細については、「NSNC-PBR-SUPPORT」を参照してください。
openShift	オプション	真	OpenShift環境が使用されていない場合、このパラメータを`false`に設定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
optimizeEndpointBinding	オプション	偽	バックエンドエンドポイントを単一のAPI呼び出しでサービスグループにバインドできるようにするには、このパラメータを`true`に設定します。アプリケーションあたりのエンドポイント（ポッド）の数が多い場合に推奨されます。NetScalerバージョン13.0-45.7以降にのみ適用されます。
podIPsforServiceGroupMembers	オプション	偽	デフォルトでは、NetScaler Ingress Controllerは、LoadBalancerサービスおよびNodePortサービスタイプを設定する際に、ノードIPアドレスとノードポートをサービスグループメンバーとして追加します。`True`に設定すると、NetScaler Ingress Controllerはノードの代わりにポッドIPアドレスとポッドポートを追加します。feature-node-watch引数を使用するか、NetScaler Node Controllerを使用していずれかの方法で、NetScalerと

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			K8sクラスターの内部ポッド間にルートがある場合、このパラメータを`true`に設定できます。
profileHttpFrontend	オプション	N/A	フロントエンドHTTPプロファイルは、設定によるインライン、または既存の事前設定済みプロファイルによって、2つの方法のいずれかで構成できます。空の値も許可されます。詳細については、FRONTEND_HTTP_PROFILEを使用した構成を参照してください。
profileSslFrontend	オプション	N/A	フロントエンドSSLプロファイルは、設定を介したインライン、または既存の事前設定済みプロファイルによって、2つの方法のいずれかで構成できます。空の値も許可されます。`profileSslFrontend.bindings.ciphers`は設定でのみ使用できます。詳細については、「FRONTEND_SSL_PROFILEを使用した構成」を参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
profileTcpFrontend	オプション	N/A	フロントエンドTCPプロファイルは、設定を介したインライン、または既存の事前設定済みプロファイルによって、2つの方法のいずれかで構成できます。空の値も許可されます。詳細については、「FRONTEND_TCP_PROFILEを使用した構成」を参照してください。
pullPolicy	必須	IfNotPresent	NetScaler Ingress Controllerのイメージプルポリシー。
rbacRole	オプション	偽	このパラメータを `true` に設定すると、NetScaler Ingress ControllerがRBACロールセットrbacRole=trueでデプロイされます。デフォルトでは、NetScaler Ingress ControllerはRBAC ClusterRole(rbacRole=false)でインストールされます。
readinessProbe	オプション	該当なし	NetScaler Ingress ControllerのReadinessプローブ設定。
リソース	オプション	該当なし	NetScaler Ingress Controllerコンテナの

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			CPUとメモリの資源。
routeLabels	オプション	該当なし	OpenShiftクラスターでrouteShardingのためにNetScaler Ingress Controllerが使用するルートラベルセレクター。
serviceClass	オプション	N/A	複数のイングレスコントローラーがデプロイされている場合に、特定のサービスのみをNetScaler Ingress Controllerに関連付けるために使用されるパラメーターです。サービスクラスの詳細については、サービス クラスのサポートを参照してください。
setAsDefaultIngressClass	オプション	偽	IngressClassオブジェクトをデフォルトのイングレスクラスとして設定します。「ingressClassName」フィールドが指定されていない新しいイングレスには、このイングレスクラスが割り当てられます。このパラメーターは、Kubernetesバージョン1.19以降にのみ適用されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
updateIngressStatus	オプション	真	NetScaler Ingress Controllerによって管理されるイングレスリソースの `Status.LoadBalancer.Ingress` フィールドを割り当てられたIPアドレスで更新する必要がある場合に、このパラメーターを設定します。詳細については、こちらを参照してください。

パラメーター	必須/オプション	デフォルト値	説明
ネットスケーラー			
adcCredentialSecret	必須	NA	NetScaler Ingress ControllerがNetScalerに接続するために必要なSecret。
nitroReadTimeout	任意	20	nitroの読み取りタイムアウト（秒単位）。
nsConfigDnsRec	任意	いいえ	イングレス経由でNetScalerでのDNSアドレスレコードの追加を有効にするには、 `true` に設定します。
nsCookieVersion	任意	0	永続性クッキーのバージョン (0 または 1)。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須/オプション	デフォルト値	説明
nsDnsNameserver	任意	該当なし	NetScaler の DNS ネームサーバー。
nsHTTP2ServerSide	任意	OFF	NetScaler サービスグループ構成で HTTP2 を有効にするには、このパラメーターを `ON` に設定します。
nsIP	必須	該当なし	NetScaler IP アドレス。
nsLbHashAlgo.required	任意	無効	ロードバランシングの一貫性ハッシュアルゴリズム。
nsLbHashAlgo.hashFingers	任意	256	ハッシュアルゴリズムに使用するフィンガーの数。指定可能な値は1から1024です。
nsLbHashAlgo.hashAlgorithm	任意	既定	サポートされているアルゴリズム。サポートされているアルゴリズムは、`default`、`jarh`、および`prac`です。
nsPort	任意	443	NetScaler Ingress ControllerがNetScalerと通信するために使用するポート。HTTPにはポート80を使用できます。
nsProtocol	任意	HTTPS	NetScaler Ingress Controllerが

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須/オプション	デフォルト値	説明
			NetScalerと通信するために使用するプロトコルです。ポート80でHTTPを使用できます。
nsSNIPS	任意	N/A	NetScaler上のサブネットIPアドレスのリスト。静的ルートの代わりにPBRルートを作成するために使用されます PBR support
nsSvcLbDnsRec	任意	フォールス	このパラメータを `true` に設定すると、LoadBalancerサービスタイプを介したNetScalerでのDNSアドレスレコードの追加が有効になります。
nsVIP	任意	N/A	NetScaler上の仮想IPアドレス。

1. 必要なパラメータの値を更新したら、**作成**をクリックします。

   NetScaler Ingress Controllerが正常にデプロイされ、初期化されていることを確認します。

   ☐

2. **ワークロード > Pod**セクションに移動し、NetScaler NetScaler Ingress Controller Podが稼働していることを確認します。

   ☐

3. トラフィックを送信してデプロイを検証します。

   ```
   curl http://citrix-ingress-Operator.com --resolve citrix-ingress-Operator.com:80:<VIP>
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

前のcurlコマンドは次を返すはずです。

```
<html><body><h1>It works!</h1></body></html>
```

注記:

- アプリケーションにClusterIPタイプのサービスを使用している場合は、OpenShiftクラスター内のPodネットワークがNetScaler VPXまたはNetScaler MPXから到達可能であることを確認してください。NetScaler Ingress Controllerを使用して静的ルートを自動的に構成するには、「静的ルートの構成」を参照してください。

参照

- NetScaler Operatorを使用してNetScaler Observability Exporterをデプロイする方法については、「NetScaler Operatorを使用してNetScaler Observability Exporterをデプロイする」を参照してください。
- NetScaler Agent Operatorをデプロイする方法については、「OpenShiftコンソールを使用してNetScalerエージェントオペレーターをインストールする」を参照してください。
- または、Helmチャートを使用してNetScaler Ingress Controllerをデプロイすることもできます。「Helmチャートを使用してNetScaler NetScaler Ingress Controllerをデプロイする」を参照してください。

NetScaler Operator を使用して OpenShift に CPX と共に NetScaler Ingress Controller (NSIC) を展開する

このセットアップでは、NetScaler Ingress Controller が OpenShift クラスターに展開された NetScaler CPX を構成します。

前提条件

- Red Hat OpenShift Cluster (バージョン 4.1 以降) へのアクセス。
- NetScaler Operator を展開します。NetScaler Operator の展開方法については、NetScaler Operator の展開 を参照してください。
- 任意のサービスモニターにエクスポートするには、次のコマンドを使用してサービスモニター CRD をインストールします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
oc create -f https://raw.githubusercontent.com/prometheus-operator/
prometheus-operator/refs/heads/main/example/prometheus-operator-crd/
monitoring.coreos.com_servicemonitors.yaml
```

- direct Prometheus export を介して収集された NetScaler CPX のメトリクスを表示する場合は、Prometheus Operator をインストールします。

NetScaler CPX と共にサイドカーとして NetScaler Ingress Controller を展開する

NetScaler Operator を使用して、NetScaler Ingress Controller をサイドカーとして NetScaler CPX を展開できます。NetScaler Ingress Controller は、OpenShift クラスターで実行されているアプリケーションのイングレスまたはルーターとして展開される NetScaler CPX を構成します。次の図はトポロジを説明しています。

次の手順を実行します。

1. OpenShift 4.x クラスターコンソールにログインします。
2. コンソールを使用して Apache アプリケーションを展開します。
   a. **Workloads > Deployments > Create Deployment** に移動し、次の YAML を使用して展開を作成します。

```
---

apiVersion: apps/v1

kind: Deployment

metadata:

  name: apache

  labels:

     name: apache

spec:

  selector:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    matchLabels:
      app: apache
  replicas: 2
  template:
    metadata:
      labels:
        app: apache
    spec:
      containers:
      - name: apache
        image: httpd:latest
        ports:
        - containerPort: 80
    ---
```

注記:

Apacheアプリケーションはデモンストレーション目的のみです。要件に基づいてYAMLファイルを変更できます。

b. **Workloads > Pods**セクションに移動し、ApacheアプリケーションのPodが稼働していることを確認します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

3. Apacheアプリケーションのサービスを作成します。**Networking > Services > Create Service** に移動し、以下のYAMLを使用してサービスを作成します。

```
apiVersion: v1

kind: Service

metadata:

  name: apache

spec:

  ports:

  - port: 80

    targetPort: 80

  selector:

    app: apache
```

4. ApacheアプリケーションのIngressを作成します。**Networking > Ingress > Create Ingress**に移動し、以下のYAMLを使用してIngressを作成します。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

 name: cpx-ingress

spec:

 rules:

 - host: citrix-ingress-operator.com

   http:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    paths:
    - backend:
        service:
          name: apache
          port:
            number: 80
      path: /
      pathType: Prefix
```

5. **Operators > Installed Operators**に移動し、**NetScaler® Operator**を選択します。
6. **NetScaler CPX™ with Ingress Controller**タブをクリックします。
7. **Create NetScalerCpxWithIngressController**をクリックします。

   NetScaler Ingress Controllerを含むNetScaler CPXのYAML定義が表示されます。

注:

- `acceptLicense` パラメータが `Yes` に設定されていることを確認してください。
- この手順で指定されたイングレスリソースにNSICを関連付けるには、`ingressClass` パラメータを使用してイングレスクラス名を `ingressClass: ['nsic-cpx]` として指定します。

ユースケースに応じて、他の利用可能なパラメータを設定するには、以下の表を参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須または任意	デフォルト値	説明
acceptLicense	必須	no	`Yes` を設定して NetScaler Ingress Controller EULA に同意します。
アフィニティ	任意	N/A	ポッド割り当て用のアフィニティラベル。
nodeSelector	オプション	該当なし	NetScaler Ingress Controllerのデプロイでで使用されるノードラベルのキーと値のペア。
podAnnotations	オプション	該当なし	ポッドに追加するアノテーションのマップ。
replicaCount	オプション	1	デプロイされるCPX-NSICポッドの数。 `cpxBgpRouter : true` の場合、CPXはDaemonSetとしてデプロイされるため、replicaCountは1です。
容認	オプション	N/A	CPX-NSICデプロイメントのトレランスを指定します。

パラメータ	必須/任意	デフォルト値	説明
: **admSettings**:			
ADMIP	任意	N/A	NetScaler Console (旧NetScaler

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			Console) のIPアドレス。
bandWidth	任意	1000	NetScaler CPXの希望する帯域幅容量（Mbps）。
bandWidthLicense	任意	フォールス	NetScaler CPXで帯域幅ベースのライセンスを使用したい場合は、このパラメータを`true`に設定します。
cpxCores	任意	1	NetScaler CPXに必要なvCPUの数。
licenseEdition	任意	PLATINUM	指定可能な値は、`Standard`、`Platinum`、および`Enterprise`です。
licenseServerIP	任意	N/A	NetScaler CPXをライセンスするためのNetScaler Console (旧称 NetScaler Console) IPアドレス。詳細については、「ライセンス」(https://docs.netscaler.com/ja-jp/citrix-k8s-ingress-controller/licensing/)を参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
licenseServerPort	任意	27000	デフォルト以外のポートが使用されている場合のNetScaler Console (旧称 NetScaler Console) ポート。
loginSecret	任意	N/A	NetScaler Consoleにログインするためのシークレットキー。
プラットフォーム	任意	いいえ	CP1000プラットフォームライセンスを有効にするには、`true`に設定します。
vCPULicense	任意	N/A	NetScaler CPXでvCPUベースのライセンスを使用する場合は、値を`true`に設定します。
licensingMode	オプション	プール	ライセンスモードをpoolまたはlasに設定します。licensingModeがlasの場合、nADMIP、netscalerCpx.loginSecret、bandWidth、およびlicenseEditionを設定します。poolライセンスの場合、licenseServerIPを設定し、bandWidthLicenseまたはvCPULicenseのい

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			ずれかを設定します。

パラメータ	必須/オプション	デフォルト値	説明
: **analyticsConfig**:			
distributedTracing.enable	オプション	いいえ	NetScalerでOpenTracingを有効にするには、値を `true` に設定します。
distributedTracing.samplingrate	任意	100	OpenTracingのサンプリングレートをパーセンテージで指定します。
endpoint.metrics.service	任意	該当なし	この値をアナリティクスサーバーのIPアドレスまたはDNSアドレスとして設定します。形式: `servicename.namespace.svc.cluster.local`, `namespace/servicename`。
endpoint.transactions.service	任意	該当なし	このパラメーターを、Kubernetes環境にデプロイされたアナリティクスサービスのIPアドレスまたは名前空間付きサービス名として設定します。形式: `namespace/servicename`。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/オプション	デフォルト値	説明
必須	必須	いいえ	NetScalerがメトリクスとトランザクションレコードをアナリティクスサービスに送信するように構成したい場合は、このパラメータを `true` に設定します。
timeseries.port	任意	5563	時系列エンドポイントのアナリティクスサービスを公開するために使用されるポート。
timeseries.metrics.enable	任意	いいえ	NetScalerからのメトリクス送信を有効にするには、この値を `true` に設定します。
timeseries.metrics.mode	任意	アヴロ	メトリックエンドポイントのモード。
timeseries.metrics.exportFrequency	任意	30	時系列データのエクスポート時間間隔。設定可能な値は30秒から300秒です。
timeseries.metrics.schemaFile	オプション	schema.json	メトリクスコレクターに追加および設定する必要があるNetScalerカウンターを含むスキーマファイルの名前。サポートされているすべてのカウンターを含む参照スキーマファイル `reference_schem`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/オプション	デフォルト値	説明
			`a.json`は、パス`/var/metrics_conf/`でも利用できます。このスキーマファイルは、カスタムカウンターリストを作成するための参照として使用できます。
timeseries.metrics.enableNativeScrape	オプション	フォールス	メトリクスのネイティブエクスポートには、このパラメーターを`true`に設定します。
timeseries.auditlogs.enable	オプション	フォールス	NetScalerから監査ログデータをエクスポートするには、このパラメーターを`true`に設定します。
timeseries.events.enable	任意	フォールス	NetScalerからイベントをエクスポートするには、このパラメーターを`true`に設定します。
transactions.enable	任意	フォールス	NetScalerからトランザクションをエクスポートするには、このパラメーターを`true`に設定します。
transactions.port	任意	5557	トランザクションエンドポイントの分析サービスを公開するために使用されるポート。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須/任意	デフォルト値	説明
: **エクスポーター**:			
必須	オプション	いいえ	このパラメータを `true` に設定すると、NetScaler Ingress Controller とともに Exporter for NetScaler Stats を実行して、NetScaler CPX のメトリクスをプルできます。
イメージ	オプション	`quay.io/netscaler/netscaler-adc-metrics-exporter@sha256:3d54b9151c742c00117be74e1ffd54430358e4cf628fe1f02d503088583f678`	Quay.io でホストされているメトリクスエクスポーターイメージ。
pullPolicy	オプション	IfNotPresent	エクスポーターイメージのプルポリシー。
リソース	オプション	{}	メトリクスエクスポーターコンテナの CPU/メモリリソース。
ports.containerPort	オプション	8888	エクスポーターコンテナポート。
extraVolumes	オプション	N/A	追加の volumeMounts用の追加ボリューム。
imagePullSecrets	オプション	N/A	プライベート Dockerレジストリ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須/任意	デフォルト値	説明
			またはリポジトリからイメージをプルするために使用されるKubernetesシークレットのリスト。このシークレットの作成方法の詳細については、「プライベートレジストリからイメージをプルする」を参照してください。

パラメータ	必須/任意	デフォルト値	説明
: **ingressController**:			
defaultSSLCertSecret	任意	N/A	NetScalerでデフォルトの非SNI証明書として使用する必要があるKubernetesシークレット名。
defaultSSLSNICertSecret	任意	N/A	NetScalerでデフォルトのSNI証明書として使用する必要があるKubernetesシークレット名。
disableAPIServerCertVerify	任意	偽	APIサーバー証明書の検証を無効にするには、このパラメーターを`true`に設定します。
disableOpenshiftRoutes	オプション	偽	デフォルトでは、OpenShift環境でOpenShiftルートが処理されます。NetScaler Ingress Controllerによる

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			OpenShiftルートの処理を無効にするには、このパラメーターを`true`に設定します。
enableLivenessProbe	オプション	真	NetScaler Ingress Controllerの活性プローブ設定を無効にするには、このパラメーターを`false`に設定します。
enableReadinessProbe	オプション	真	NetScaler Ingress Controllerの準備プローブ設定。
entityPrefix	オプション	N/A	NetScaler CPX上のリソースのプレフィックス。
extraVolumeMounts	オプション	N/A	
イメージ	必須	`quay.io/netscaler/netscaler-k8s-ingress-controller@sha256:2ff38654476234bfe2c3908932702b90c8ec6ed3cfb7c7f467dc6181df0f1adf`	Quay.ioでホストされているNetScaler Ingress Controllerイメージ。
ingressClass	オプション	N/A	特定のイングレスリソースをNetScaler Ingress Controllerに関連付けるためのパラメータ。イングレスクラスの詳細については、「Ingress

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			class support」を参照してください。Kubernetesバージョン1.19以降の場合、このパラメータはここで指定された名前でIngressClassオブジェクトを作成します。
IPAM	オプション	偽	IPAMコントローラーを使用してIPアドレス管理を有効にするには、このパラメーターを`true`に設定します。
jsonLog	オプション	偽	ログメッセージをJSON形式で出力する必要がある場合は、このパラメーターを`true`に設定します。
kubernetesURL	オプション	N/A	NetScaler Ingress Controllerがイベントを登録するために使用するkube-apiserverのURLです。値が指定されていない場合、NetScaler Ingress Controllerは内部kube-apiserver IPアドレスを使用します。
livenessProbe.initialDelaySeconds	オプション	30	Kubernetesが最初の活性チェックを実行する前に待機する

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			時間（秒単位）です。
livenessProbe.periodSeconds	任意	60	初期遅延の後、Kubernetes は 60 秒ごとに活性チェックを実行します。
logLevel	任意	INFO	NSIC によって生成されるログを制御するログレベルです。サポートされているログレベルは、`CRITICAL`、`ERROR`、`WARNING`、`INFO`、`DEBUG`、`TRACE`、および `NONE` です。
logProxy	任意	N/A	NetScaler 可観測性エクスポーター用の Elasticsearch、Kafka、または Zipkin エンドポイント。
namespaceLabels	任意	N/A	OpenShift クラスターで routeSharding のために NetScaler Ingress Controller が使用するネームスペースラベルセレクター。
OpenShift	オプション	真	OpenShift環境を使用していない場合は、このパラメータを`false`に設定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
optimizeEndpointBinding	オプション	偽	バックエンドエンドポイントを単一のAPI呼び出しでサービスグループにバインドできるようにするには、`true`に設定します。アプリケーションあたりのエンドポイント（ポッド）の数が多い場合は、このパラメータをtrueに設定することをお勧めします。NetScalerバージョン13.0-45.7以降にのみ適用されます。
profileHttpFrontend	オプション	N/A	フロントエンドHTTPプロファイル。詳細については、FRONTEND_HTTP_PROFILEを使用した構成を参照してください。
profileSslFrontend	オプション	N/A	フロントエンドSSLプロファイル。詳細については、FRONTEND_SSL_PROFILEを使用した構成を参照してください。
profileTcpFrontend	オプション	該当なし	フロントエンドTCPプロファイル。詳細については、FRONTEND_TCP_PROFILEを使用し

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			た構成を参照してください。
pullPolicy	必須	IfNotPresent	NetScaler Ingress Controllerのイメージプルポリシー。
rbacRole	オプション	偽	RBACロールでNSICを展開するには、`rbacRole=true`を設定します。デフォルトでは、NSICはClusterRole (rbacRole=false) でインストールされます。
readinessProbe	オプション	N/A	Readinessプローブの設定。
リソース	オプション	{}	NetScalerのイングレスコントローラーコンテナのCPU/メモリリソース。
routeLabels	オプション	N/A	このパラメータを使用して、OpenShiftクラスターでNetScaler Ingress Controllerがルーティングに使用するルートラベルセレクターを指定できます。
setAsDefaultIngressClass	オプション	偽	IngressClassオブジェクトをデフォルトのイングレスクラスとして設定します。「

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			ingressClassName」フィールドが指定されていない新しいイングレスには、このイングレスクラスが割り当てられます。このパラメータは、Kubernetes バージョン1.19以降にのみ適用されます。
updateIngressStatus	任意	はい	NetScaler Ingress Controller によって管理されるイングレスリソースの `Status.LoadBalancer.Ingress` フィールドを割り当てられた IP アドレスで更新する必要がある場合に、このパラメーターを設定します。詳細については、こちらを参照してください。

パラメーター	必須/任意	デフォルト値	説明
: **netscalerCpx**:			
bgpSettings.bgpConfig	任意	N/A	このパラメーターは、YAML 形式の BGP 設定を表します。個々のフィールドの説明については、ドキュメントを参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須/任意	デフォルト値	説明
cpxBgpRouter	任意	いいえ	このパラメータがtrueに設定されている場合、CPXはBGPコントローラーモードのデーモンセットとしてデプロイされ、BGPアドバタイズメントはKubernetesクラスターへの外部トラフィックを誘引するために行われます。
cpxCommands	オプション	該当なし	CPXがインスタンス化されるときに適用される、ユーザー提供のNetScaler起動設定。この設定は動的な設定ではなく、configmapへのその後の変更は、ポッドが再起動されない限りCPX設定に反映されません。詳細については、(https://docs.netscaler.com/ja-jp/citrix-adc-cpx/current-release/configure-cpx-kubernetes-using-configmaps.html)を参照してください。
cpxLicenseAggregator	オプション	該当なし	CPXのライセンスに使用されている場合、CPXライセンスアグリゲーターのIPアドレス/FQDN。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須/任意	デフォルト値	説明
cpxShellCommands	オプション	該当なし	CPXがインスタンス化されるときに適用される、ユーザー提供の起動設定。これは動的な設定ではなく、configmapへのその後の変更は、ポッドが再起動されない限りCPX設定に反映されないことに注意してください。詳細については、(https://docs.netscaler.com/ja-jp/citrix-adc-cpx/current-release/configure-cpx-kubernetes-using-configmaps.html)を参照してください。
daemonSet	オプション	いいえ	NetScaler CPXをDaemonSetとしてデプロイする必要がある場合は、このパラメータを`true`に設定します。
enableLivenessProbe	オプション	はい	NetScaler CPXのliveness probe設定を無効にするには、`false`に設定します。
enableStartupProbe	オプション	はい	NetScaler CPXのstartupProbe設定を無効にするには、`false`に設定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須/任意	デフォルト値	説明
extraVolumeMounts	オプション	N/A	
hostName	オプション	N/A	このエンティティは、CPXのホスト名を設定するために使用されます。
イメージ	必須	quay.io/netscaler/netscaler-cpx@sha256:04defadf6311259d6ada5380498ed7afe8035b11ee3c1d7b456faa94472b5550	Quay.ioでホストされているNetScaler CPXイメージ。
ingressIP	任意	N/A	値がイングレスリソースの `ingress.com/frontend-ip` アノテーションによって上書きされない場合、イングレスリソースによって使用される外部IPアドレス。このIPアドレスは、cpxBgpRouterがtrueに設定されている場合、外部ルーターにもアドバタイズされます。
livenessProbe	任意	N/A	NetScaler CPXのlivenessProbe設定。
mgmtHttpPort	オプション	9080	NodePort CPXサービスに使用されるHTTPポート。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須/任意	デフォルト値	説明
mgmtHttpsPort	オプション	9443	NodePort CPXサービスに使用されるHTTPSポート。
nitroReadTimeout	オプション	20	nitro読み取りタイムアウト（秒単位）。
nsConfigDnsRec	オプション	フォールス	Ingressを介したNetScalerでのDNSアドレスレコードの追加を有効/無効にするため。
nsCookieVersion	任意	0	永続性クッキーのバージョン。指定可能な値は0と1です。
nsDnsNameserver	任意	N/A	NetScalerのDNSネームサーバー。
nsEnableLabels	任意	トゥルー	NetScaler CPXのラベルを有効にします。
nsGateway	任意	192.168.1.1	CPXがホストモードで実行されている場合、つまりcpxBgpRouterがtrueに設定されている場合に、内部通信に使用するゲートウェイ。このパラメータが指定されていない場合、nsIPネットワークの最初のIPアドレスがゲートウェイとして使用されます。nsIPと同じネットワーク内にある必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須/任意	デフォルト値	説明
nsHTTP2ServerSide	オプション	オフ	NetScalerサービスグループ構成でHTTP2を有効にするには、このパラメータを`ON`に設定します。
nsIP	オプション	192.168.1.2	CPXがホストモードで実行されている場合、つまりcpxBgpRouterが`true`に設定されている場合に、NetScaler CPXが内部通信に使用するNetScaler IPアドレス。このIP範囲に/24の内部ネットワークが作成され、ネットワーク名前空間内の内部通信に使用されます。
nsLbHashAlgo.required	オプション	いいえ	ロードバランシングの一貫性ハッシュアルゴリズムを設定するには、このパラメータを`true`に設定します。
nsLbHashAlgo.hashFingers	オプション	256	ハッシュアルゴリズムに使用するフィンガーの数。指定可能な値は1から1024までです。
nsLbHashAlgo.hashAlgorithm	任意	'デフォルト'	サポートされているアルゴリズムは、`default`、`jarh`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須/任意	デフォルト値	説明
			、および `prac` です。
nsProtocol	任意	エイチティーティーピー	NetScaler CPXとNetScaler Ingress Controller間の通信に使用されるプロトコル。
nsSvcLbDnsRec	任意	いいえ	このパラメータを `true` に設定すると、タイプLoadBalancerサービスを介したNetScalerでのDNSアドレスレコードの追加が有効になります。
prometheusCredentialSecret	任意	該当なし	Prometheus を使用したメトリクスのネイティブエクスポート用の読み取り専用ユーザーを作成するために必要なシークレットキー。
pullPolicy	必須	IfNotPresent	NetScaler CPX イメージのプルポリシー。
service.annotations	任意	該当なし	CPX サービスで使用されるアノテーションの辞書。例: こちらを参照してください。
service.spec	任意	該当なし	NetScaler CPX の仕様設定。CPX サービスを NodePort として公開するには、 `service.spec.ty`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須/任意	デフォルト値	説明
			`pe` を `NodePort` として指定し、`service.annotations` の下の `service.citrix.com/secret` アノテーションを使用してフロントエンドサーバー証明書のシークレットリソース名を指定します。例: `service.citrix.com/secret: tls-secret`。この表の後の注記では、CPX サービスを NodePort として公開するために、デプロイメント YAML で必要なフィールドを指定する方法の例を示します。
startupProbe	オプション	N/A	NetScaler CPXのstartupProbe設定。

注

CPXサービスをNodePortとして公開するには、サービスタイプを`NodePort`として指定し、CPX-with-NSICデプロイメントYAMLでフロントエンドサーバー証明書のシークレットリソース名を次のように指定します。

```
service:

  annotations:

    service.citrix.com/secret: tls-secret
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  spec:

    type: NodePort
```

8. 必要なパラメーターの値を更新したら、**作成**をクリックします。Ingress Controllerが搭載されたNetScaler CPXが正常にデプロイされ、初期化されていることを確認してください。

9. NetScaler CPXのサービスアカウントに特権セキュリティコンテキスト制約をアタッチします（特権ポッドとして実行されるため）。次のコマンドを使用します。

   - NetScaler CPXがデプロイされているネームスペースで、次のコマンドを使用してNetScaler CPXが使用するサービスアカウント名を取得します: `oc get sa`
   - NetScaler CPXのサービスアカウントに特権SCCをアタッチします。

```
oc adm policy add-scc-to-user privileged
system:serviceaccount:<namespace>:<CPX-ServiceAccount-Name retrieved in
the previous step>
```

10. **ワークロード > ポッド**セクションに移動し、`netscaler-cpx-with-ingress-controller`ポッドが稼働中であることを確認します。

11. トラフィックを送信してデプロイメントを確認します。

    a. 次のコマンドを使用してNodePortの詳細を取得します。

```
oc get svc
```

    b. 次のコマンドに示すように、`cpx-service` NodePort を使用してトラフィックを送信します。

```
curl http://citrix-ingress-Operator.com:<NodePort> --resolve citrix-
ingress-Operator.com:<NodePort>:<Master-Node-IP>
```

    上記の curl コマンドは、次の出力を返すはずです。

```
<html><body><h1>It works!</h1></body></html>
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

参考資料

- NetScaler Operator を使用して NetScaler Observability Exporter を展開する方法については、NetScaler Operator を使用して NetScaler Observability Exporter を展開する を参照してください。
- NetScaler エージェントオペレーターを展開する方法については、OpenShift コンソールを使用して NetScaler エージェントオペレーターをインストールする を参照してください。
- または、Helm チャートを使用して NetScaler Ingress Controller を展開することもできます。Helm チャートを使用して NetScaler Ingress Controller を展開する を参照してください。

NetScaler Operator を使用して NetScaler Cloud Controller をデプロイする

はじめに

NetScaler Cloud Controller は、適切な Azure ルートテーブルにポッドネットワークのルートエントリを作成および維持します。

> **注:**
>
> NetScaler Cloud Controller は、NetScaler と、同じリソースグループ内にある Azure Red Hat OpenShift (ARO) クラスター間の Azure ルートのみを作成します。

このセクションでは、Azure および ARO 環境における NetScaler Cloud Controller のデプロイ、構成、および管理に関する情報を提供します。

前提条件

1. アクティブな Azure サブスクリプション。

   サブスクリプション ID をメモします。
2. リソースを作成および管理するための十分な権限を持つ Azure アカウント。リソースグループ内でリソースを作成し、サービスプリンシパルを管理するためのアクセス権が必要です。
3. Azure アカウントの **TENANT_ID**、**CLIENT_ID**、および **CLIENT_SECRET** をメモします。
4. Azure で実行中の ARO クラスター (バージョン 4.11 以降)。
5. ARO クラスターと、ルートを確立する必要があるサブネットは、同じ VNet 内にある必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

リソースグループ名、VNet 名、サブネット名、および ARO クラスターと NetScaler® が配置されている Azure の場所をメモします。

6. NetScaler Cloud Controller をデプロイする名前空間で、説明されているようにシークレットを作成します。

```
oc create secret generic azsecret --from-literal=clientid='<CLIENT_ID>'
--from-literal=clientsecret='<CLIENT_SECRET>' -n <namespace>
```

7. サービスプリンシパルを作成し、カスタムロールを割り当てます。詳細については、サービスプリンシパルを作成してロールを割り当てるを参照してください。
8. NetScaler Operator を展開します。NetScaler Operator の展開方法については、NetScaler Operator を展開するを参照してください。

サービスプリンシパルを作成してロールを割り当てる

NetScaler Cloud Controller が Azure リソースを管理するための認証情報を設定するには、サービスプリンシパルを作成してロールを割り当てます。サービスプリンシパルを作成してロールを割り当てる際に考慮すべき2つのシナリオを以下に示します。

ケース I: ルートを作成するサブネットが ARO クラスターの一部ではない場合

サービスプリンシパルを目的のロールで作成するか、既存のサービスプリンシパルにロールを追加するには、以下の手順に従います。

注:

サービスプリンシパルが既に存在する場合は、手順9に進んでサービスプリンシパルにロールを割り当ててください。

1. **Microsoft Azure Portal** にログインします。
2. **App registrations** サービスに移動します。
3. **New Registration** をクリックします。

4. 名前やサポートされているアカウントの種類などの詳細を入力します。
5. **Register** をクリックしてサービスプリンシパルを作成します。

   サービスプリンシパルの詳細から **clientID** をメモします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

クライアントID(/en-us/netscaler-k8s-ingress-controller/media/clientid.png)

6. 左ペインで**証明書とシークレット**をクリックし、**新しいクライアントシークレット**をクリックします。

   クライアントシークレット(/en-us/netscaler-k8s-ingress-controller/media/client-secret.png)

7. 説明を入力し、有効期限を選択します。

   クライアントシークレットの詳細(/en-us/netscaler-k8s-ingress-controller/media/client-secret-details.png)

8. シークレット値をメモします。この値が**clientSecret**です。

   クライアントシークレット値(/en-us/netscaler-k8s-ingress-controller/media/client-secret-value.png)

9. **リソースグループ → `<Your Resource Group>` → アクセス制御 (IAM)**に移動します。

   AROクラスターとVNetが存在するリソースグループを選択します。

10. **追加**をクリックし、**カスタムロールの追加**を選択します。

    カスタムロールの追加(/en-us/netscaler-k8s-ingress-controller/media/add-custom-role.png)

11. 次のcustom.JSONサンプルを追加します。

```
{

"properties": {

    "roleName": "NetScaler Cloud Controller Role",

    "description": "Allows managing cloud controller role",

    "assignableScopes": [

        "/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/
<RESOURCE_GROUP_NAME>"

    ],

    "permissions": [

        {

            "actions": [
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
                "Microsoft.Network/routeTables/routes/write",
                "Microsoft.Network/routeTables/routes/read",
                "Microsoft.Network/routeTables/routes/delete",
                "Microsoft.Network/routeTables/delete",
                "Microsoft.Network/routeTables/write",
                "Microsoft.Network/routeTables/read",
                "Microsoft.Network/routeTables/join/action",
                "Microsoft.Network/networkSecurityGroups/join/action",
                "Microsoft.Network/virtualNetworks/subnets/read",
                "Microsoft.Network/virtualNetworks/subnets/write"
            ],
            "notActions": [],
            "dataActions": [],
            "notDataActions": []
        }
    ]
}
}
```

12. 希望する設定に従って`custom_role.json`ファイルを変更します。

    JSONファイルに、ご自身の**リソースグループ**と**サブスクリプションID**を追加する必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

13. **ベースライン権限**で**JSONから開始**を選択し、変更した`custom_role.json`ファイルをアップロードします。

    「カスタムロールの追加」(/en-us/netscaler-k8s-ingress-controller/media/add-custom-role-json.png)

14. **「レビュー + 作成」**をクリックします。
15. **「ロールの割り当ての追加」**に移動します。

    「ロールの割り当ての追加」(/en-us/netscaler-k8s-ingress-controller/media/add-custom-role-assignment.png)

16. **ステップ14**で作成したカスタムロールを検索します。
17. ロールを選択し、**「次へ」**を押します。

    「カスタムロールの検索」(/en-us/netscaler-k8s-ingress-controller/media/select-custom-role.png)

18. **「メンバー」**の下で、**ステップ5**で作成したサービスプリンシパルを選択します。

    「ロールの割り当ての追加」(/en-us/netscaler-k8s-ingress-controller/media/add-role-assignment.png)

19. **「レビュー + 割り当て」**をクリックします。

ケースII: ルートが作成されるサブネットがAROクラスターの一部である場合

この場合、OpenShiftのインストール中に作成されたサービスプリンシパルへのアクセスが必要です。AROのインストール中に自動的に作成されるサービスプリンシパルには、**aro-app-**というプレフィックスが付いています。

1. AROクラスターとVNetが存在する**リソースグループ → `<Your Resource Group>`**に移動します。**リソースグループ → `<Your Resource Group>` → アクセス制御 (IAM)**に移動します。
2. `Add`をクリックし、`Add custom role`を選択します。

    「カスタムロールの追加」(/en-us/netscaler-k8s-ingress-controller/media/add-custom-role.png)

3. 以下を`custom_role.json.`という名前のJSONファイルにコピーします。

```
{

"properties": {
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    "roleName": "NetScaler Cloud Controller Role",

    "description": "Allows managing cloud controller role",

    "assignableScopes": [

        "/subscriptions/<SUBSCRIPTION_ID>/resourceGroups/
<RESOURCE_GROUP_NAME>"

    ],

    "permissions": [

        {

            "actions": [

                "Microsoft.Network/routeTables/routes/write",

                "Microsoft.Network/routeTables/routes/read",

                "Microsoft.Network/routeTables/routes/delete",

                "Microsoft.Network/routeTables/delete",

                "Microsoft.Network/routeTables/write",

                "Microsoft.Network/routeTables/read",

                "Microsoft.Network/routeTables/join/action",

                "Microsoft.Network/networkSecurityGroups/join/action",

                "Microsoft.Network/virtualNetworks/subnets/read",

                "Microsoft.Network/virtualNetworks/subnets/write"

            ],

            "notActions": [],

            "dataActions": [],
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            "notDataActions": []

        }

    ]

}

}
```

4. `custom_role.json`ファイルを希望の設定に合わせて変更します。

   JSONファイルに**リソースグループ**と**サブスクリプションID**を追加する必要があります。

5. **ベースライン権限**で**JSONから開始**を選択し、変更した`custom_role.json`ファイルをアップロードします。

   カスタムロールを追加(/en-us/netscaler-k8s-ingress-controller/media/add-custom-role-json.png)

6. **レビュー + 作成**をクリックします。
7. **ロールの割り当てを追加**をクリックします。

   ロールの割り当てを追加(/en-us/netscaler-k8s-ingress-controller/media/add-custom-role-assignment.png)

8. ステップ4で作成したカスタムロールを検索します。ロールを選択し、**次へ**をクリックします。

   カスタムロールを検索(/en-us/netscaler-k8s-ingress-controller/media/select-custom-role.png)

9. **メンバー**セクションで、OpenShiftクラスターを管理しているサービスプリンシパルを選択します。

   ロールの割り当てを追加(/en-us/netscaler-k8s-ingress-controller/media/add-role-assignment.png)

10. **レビュー + 割り当て**をクリックします。

NetScaler Cloud Controllerインスタンスをデプロイする手順

1. 4.x OpenShiftコンソールにログインします。
2. 「**Operators > Installed Operators**」に移動し、「**NetScaler Operator**」を選択します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Ingress Controller オペレーター(/en-us/netscaler-k8s-ingress-controller/media/ns-installed-Operators.png)

1. NetScaler Operator に移動し、`NetScaler Cloud Controller` タブをクリックしてから、「**Create NetScaler Cloud Controller**」をクリックします。

   クラウドコントローラーの展開(/en-us/netscaler-k8s-ingress-controller/media/deploy-cloud-controller.png)

   NetScaler Cloud Controller CRD の YAML が表示されます。オプションで、`Form view` ボタンを選択し、フォームでパラメータを更新できます。
2. .YAML ファイルを編集して、前提条件セクションで取得した以下の値を追加します。

   クラウドコントローラーの作成(/en-us/netscaler-k8s-ingress-controller/media/create-cloud-controller.png)

   表示される YAML のフィールドの説明については、以下の表を参照してください。

パラメータ	必須/オプション	デフォルト値	説明
名前	必須	ネットスケーラークラウドコントローラー-サンプル	NetScaler Cloud Controllerインスタンスの名前。
acceptLicense	必須	いいえ	NetScaler Cloud Controller EULAに同意するには、この値を`Yes`に設定します。
親和性	任意	該当なし	ポッド割り当てのアフィニティラベル。
nodeSelector	任意	該当なし	NetScaler Cloud Controllerの展開におけるnodeSelectorオプションに使用されるノードラベルのキーと値のペア。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/オプション	デフォルト値	説明
podAnnotations	任意	N/A	ポッドに追加するアノテーションのマップ。
トレレーション	任意	N/A	NetScaler Cloud Controllerデプロイメントの tolerations。

パラメータ	必須/任意	デフォルト値	説明
: **アジュール**:			
クライアントシークレット	必須	N/A	前提条件セクションで作成されたシークレット名。
イメージ	必須	`quay.io/netscaler/netscaler-cloud-controller@shavalue of latest release`	Quay.ioでホストされているNetScaler Cloud Controllerイメージ。
場所	必須	N/A	Azureの場所。
pullPolicy	任意	IfNotPresent	NetScaler Cloud Controllerイメージのプルポリシー。
リソースグループ名	必須	N/A	VNetが存在するリソースグループ名。
リソース	オプション	該当なし	NetScaler Cloud ControllerコンテナのCPUとメモリのリソース。
サブネット名	必須	該当なし	AROクラスターへのルートを確立するAzure VNetサブ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/任意	デフォルト値	説明
			ネット名。このサブネットとAROクラスターは同じVNET内にある必要があります。
サブスクリプションID	必須	該当なし	Azureアカウントのサブスクリプション ID。
テナントID	必須	N/A	Azureサービスプリンシパルのテナント ID。
VNet名	必須	N/A	必要なサブネットとOpenShiftクラスターが存在するAzure VNet名。

3. 必要なパラメーターの値を更新した後、**作成**をクリックします。
4. **ワークロード > Pods**セクションに移動し、NetScaler Cloud Controllerポッドが稼働していることを確認します。

コントローラーポッドが稼働すると、必要なサブネットを持つルートテーブルが作成されます。

注記:

- ルートテーブルが存在しない場合、コントローラーはルートテーブルを作成し、指定されたサブネットに関連付けます。新しいテーブルの形式は `<SUBNET_NAME>_<LOCATION>_rt` になります。
- 他の自動化プロセスやユーザーによってNetScaler Cloud Controllerから作成/更新されたルートとの競合を避けるため、NetScaler Cloud Controllerによって管理されるルートは、追跡用のハッシュ値が続くプレフィックス**NSCC_**で始まります。競合を避けるために、同じ名前のプレフィックスを持つルートを作成しないでください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Cloud Controllerがデプロイされると、以下のロールがインストールされます。NetScaler Cloud Controllerは、そのCRDを管理し、ノード固有の情報を取得するためにこれらのロールを必要とします。

APIグループ	リソース	動詞
rbac.authorization.k8s.io	クラスターロールバインディング	作成、削除、取得、一覧表示、パッチ、更新、監視
rbac.authorization.k8s.io	クラスターロール	作成、削除、取得、一覧表示、パッチ、更新、監視
コア	ノード	すべて
コア	サービスアカウント	作成、削除、取得、一覧表示、パッチ、更新、監視
アプリ	デプロイメント	作成、削除、取得、一覧表示、パッチ、更新、監視
コア	ポッド	作成、削除、取得、一覧表示、パッチ、更新、監視、サービスアカウント

NetScaler Operator を使用して OpenShift に NetScaler GSLB Controller をデプロイする

NetScaler Operator を使用すると、OpenShift クラスターに NetScaler GSLB Controller をデプロイできます。GSLB の詳細については、GSLB の概要とデプロイのトポロジー を参照してください。

NetScaler® GSLB Controller をデプロイする

前提条件

- Red Hat OpenShift Cluster (バージョン 4.11 以降)。
- NetScaler Operator をデプロイします。NetScaler Operator のデプロイ方法については、NetScaler Operator のデプロイ を参照してください。

NetScaler Operator を使用して NetScaler GSLB Controller をデプロイする手順

次の手順を実行します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

1. **Operators > Installed Operators** に移動し、**NetScaler Operator** を選択します。

2. **NetScaler GSLB Controller** タブをクリックし、**Create NetscalerGslbController** をクリックします。

   NetScaler GSLB Controller インスタンスの YAML 定義が表示されます。オプションで、`Form view` ボタンを選択して、フォームでパラメーターを更新できます。

3. 表示された NetScaler GSLB Controller インスタンスの YAML で、必要なパラメーターの値を更新し、**Create** をクリックします。設定する必要があるパラメーターの詳細については、次の表を参照してください。

パラメータ	説明	必須/任意
:**gslbController**:		
イメージ	NetScaler GSLB Controller イメージ。	必須
pullPolicy	NetScaler GSLB Controller イメージのプルポリシー。	任意
imagePullSecrets	プライベート Docker レジストリまたはリポジトリからイメージをプルするために使用される OpenShift シークレットのリスト。このシークレットの作成方法の詳細については、「プライベートレジストリからイメージをプルする」を参照してください。	任意
entityPrefix	NetScaler VPX または NetScaler MPX 上のリソースのプレフィックス。	必須
acceptLicense	GSLB Controller のエンドユーザーライセンス契約に	必須

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	説明	必須/任意
	同意するには、yes に設定します。	
logLevel	NetScaler GSLB Controller によって生成されるログを制御するログレベルです。サポートされているログレベルは、CRITICAL、ERROR、WARNING、INFO、DEBUG、TRACE、および NONE です。詳細については、「ロギング」を参照してください。	任意
オープンシフト	OpenShift 環境を使用している場合は、この引数を設定します。	必須
disableAPIServerCertVerify	APIサーバー証明書の検証を無効にするには、このパラメーターを true に設定します。	任意
kubernetesURL	NetScaler GSLB Controller がイベントを登録するために使用する kube-apiserver の URL です。値が指定されていない場合、NetScaler GSLB Controller は内部の kube-apiserver IP アドレスを使用します。	オプション
cleanupGSLBSiteConfig	GSLBサイト設定をクリーンアップするには、このパラメーターをtrueに設定します。	オプション
LocalRegion	GSLBコントローラーがデプロイされているローカルリージョン。	必須

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	説明	必須/任意
LocalCluster	GSLBコントローラーがデプロイされているクラスターの名前。この値は、OpenShiftクラスターごとに一意です。	必須
siteData\[0].siteName	GSLBデバイスで構成されている最初のGSLBサイトの名前。	必須
siteData\[0].secretName	最初のGSLBサイトのログイン資格情報を含むシークレットの名前。	必須
siteData\[0].siteIp	最初のGSLBサイトのIPアドレス。site1のNetScalerのIPアドレスを*siteData\[0].siteIp*として追加します。	必須
sitedata\[0].siteRegion	最初のサイトのリージョン。	必須
sitedata\[0].siteMask	最初のGSLBサイトIPアドレスのネットマスク。	必須
sitedata\[0].sitePublicIp	最初のGSLBサイトのパブリックIPアドレス。	必須
sitedata\[1].siteName	GSLBデバイスで構成された2番目のGSLBサイトの名前。	必須
sitedata\[1].secretName	2番目のサイトのログイン資格情報を含むシークレット。	必須
sitedata\[1].siteIp	2番目のGSLBサイトのIPアドレス。site2のNetScalerのIPアドレスを	必須

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	説明	必須/任意
	*siteData\[0].siteIp*として追加します。	
sitedata\[1].siteRegion	2番目のサイトのリージョン。	必須
sitedata\[1].siteMask	2番目のGSLBサイトIPアドレスのネットマスク。	必須
sitedata\[1].sitePublicIp	2番目のGSLBサイトのパブリックIPアドレス。	必須

パラメータ	説明	必須/オプション
: **ネットスケーラー**:		
nsIP	NetScaler IPアドレス。	必須
adcCredentialSecret	GSLBコントローラーがGSLBデバイスに接続し、GSLBコントローラーから構成をプッシュするために必要なシークレット。	オプション
nsProtocol	NetScaler GSLB ControllerがNetScalerと通信するために使用するプロトコル。ポート80でHTTPを使用することもできます。	オプション
nsPort	NetScaler GSLB Controller が NetScaler と通信するために使用するポート。HTTP にはポート 80 を使用できます。	オプション
nitroReadTimeout	NetScaler GSLB Controller が NetScaler から応答を受信するまで接続を終了する前に待機する時間（秒単位）。	オプション

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	説明	必須/オプション
localSiteSelection	GSLB サービスの順序を設定する際に、ローカルサイトを優先するようにこのパラメーターを設定します。有効にすると、NetScaler に構成を適用する ConfigMap が作成されます。詳細については、「ローカルサイトの優先設定を構成するための ConfigMap」を参照してください。	オプション

パラメーター	必須/オプション	説明
名前	必須	NetScaler GSLB Controller インスタンスの名前。
acceptLicense	必須	NetScaler GSLB Controller EULAに同意するには、この値を `Yes` に設定します。
アフィニティ	任意	Pod割り当て用のアフィニティラベル。
nodeSelector	任意	NetScaler GSLB Controller の展開でnodeSelectorオプションに使用されるノードラベルのキーと値のペア。
podAnnotations	任意	Podに追加するアノテーションのマップ。
トレランス	任意	NetScaler GSLB Controller の展開に対するトレランス。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注記:

- 2つ以上のGSLBサイトを設定したい場合は、必要に応じて`siteData`リストにすべてのGSLBサイトの詳細を含めます。
- GSLB ControllerインスタンスのYAMLでNetScaler VPXまたはNetScaler MPXの資格情報を提供するには、**adcCredentialSecret** または **secretStore.username** と **secretStore.password** の組み合わせのいずれかを指定する必要があります。

4. **Workloads > Pods** セクションに移動し、NetScaler GSLB Controllerポッドが稼働していることを確認します。

   アプリケーションポッドが稼働中(/en-us/netscaler-k8s-ingress-controller/media/gslb-controller-a.png)

   アプリケーションポッドが稼働中(/en-us/netscaler-k8s-ingress-controller/media/gslb-controller-b.png)

   参照にスキップします。

参照

- グローバル・トラフィック・ポリシー (GTP) およびグローバル・サービス・エントリ (GSE) の展開に関する情報については、GTPとGSEの展開を参照してください。

NetScaler Operator を使用して、NetScaler® Ingress Controller をスタンドアロンポッドとしてデプロイする

NetScaler Operator を使用して、IPアドレス管理のためにNetScaler IPAM Controller をOpenShift クラスターにスタンドアロンポッドとしてデプロイできます。LoadBalancer または Ingress タイプのサービスを作成する際、NetScaler IPAM controller を使用してIPアドレスを自動的に割り当てることができます。IPAMコントローラーがデプロイされると、定義済みのIPアドレス範囲から LoadBalancer または Ingress タイプのサービスにIPアドレスを割り当てます。NetScaler ingress controller は、OpenShiftクラスターで実行されているアプリケーションのイングレスデバイスまたはルーターとしてデプロイされているNetScaler MPX または VPX で、サービスに割り当てられたIPアドレスを仮想IPアドレス (VIP) として構成します。以下の図はトポロジを説明しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

前提条件

NetScaler Operator を使用してNetScaler Ingress Controller をスタンドアロンポッドとしてデプロイするための前提条件は以下のとおりです。

- Red Hat OpenShift Cluster (バージョン 4.11 以降) を用意する。
- NetScaler Operator をデプロイします。NetScaler Operator のデプロイ方法については、NetScaler Operator のデプロイ を参照してください。
- NetScaler IPAM がIPアドレスを割り当てるNetScaler VPX 上でサービスまたはイングレスを構成するために、NetScaler Ingress Controller をデプロイします。NetScaler Ingress Controller のデプロイ方法については、NetScaler Ingress Controller のデプロイ を参照してください。

デプロイ手順

1. **OpenShift 4.x Cluster** コンソールにログインします。
2. **NetScaler IPAM Controller** を使用してイングレスにIPを割り当てます。
3. Apacheアプリケーションのイングレスに IPを割り当てます。

    a. **Networking > Ingress > Create Ingress** に移動します。

    b. 以下のYAMLのようなものを使用してイングレスを作成します。イングレスにIPアドレスを割り当てるIPAMに必要なアノテーションがあることを確認してください。このIPアドレスは、NetScaler Ingress Controller によってNetScaler VPX™ 上でVIPアドレスとして構成されます。

```
---

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

annotations:

    # IPAM uses the below annotation to select the IP Range, from
which it will allocate IPs

    ingress.citrix.com/ipam-range: ingress-range
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
name: apache-ingress

spec:

ingressClassName: nsic-vpx

rules:

- host: citrix-ingress-operator.com

    http:

    paths:

    - backend:

        service:

            name: apache

            port:

            number: 80

        path: /

        pathType: ImplementationSpecific

---
```

4. LoadBalancer タイプのサービスにIPを割り当てます。

    a. **Networking > Services > Create Service** に移動します。

    b. 以下のYAMLを使用してサービスを作成します。IPAMに必要な*アノテーション*があることを確認してください。これにより、サービスにIPアドレスが割り当てられます。このIPアドレスは、NetScaler Ingress ControllerによってNetScaler VPXのVIPアドレスとして構成されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
---

apiVersion: v1

kind: Service

metadata:

name: apache-lb

labels:

    app: apache

annotations:

    # CIC uses the below annotation to select services to be
configured on NetScaler VPX/MPX

    service.citrix.com/class: 'nsic-vpx'

    # IPAM uses the below annotation to select the IP Range, from
which it will allocate IPs

    service.citrix.com/ipam-range: 'service-range'

spec:

type: LoadBalancer

ports:

- name: http

    port: 80

    targetPort: http

selector:

    app: apache

---
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

5. **Operators > Installed Operators** に移動し、**NetScaler Operator** をクリックします。

6. **NetScaler IPAM Controller** タブをクリックし、**Create NetScalerIPAMController** を選択します。NetScaler IPAM ControllerのYAML定義が表示されます。

   オプションで、**Form view** ボタンを選択して、フォームでパラメータを更新できます。インストール中に構成できる必須およびオプションのパラメータを一覧表示するMandatory and Optional Parametersセクションを参照してください。ユースケースに応じて、他の利用可能なパラメータを構成できます。

   注記:

   - `acceptLicense` パラメータが `Yes` に設定されていることを確認してください。
   - `localIPAM` または `Infoblox` のパラメータに値を指定することは必須です。
   - Infobloxを使用する場合は、 `clusterName` を指定してください。

7. 必要なパラメータの値を更新した後、**Create** をクリックします。
8. NetScaler IPAM Controllerが正常に展開され、初期化されていることを確認します。**Workloads > Pods section** に移動し、NetScaler IPAM Controllerポッドが稼働中であることを確認します。
9. ログを確認してデプロイを検証します。また、**イングレス**の場合、イングレスステータスを確認して、NetScaler IPAM ControllerによってIPアドレスがイングレスに割り当てられていることを確認できます。

   イングレスステータス

```
root@master:~/Citrix/openshift# oc get ing

NAME              CLASS      HOSTS                          ADDRESS
PORTS   AGE

apache-ingress    nsic-vpx   citrix-ingress-operator.com    1.1.1.166
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
80      5s

root@master:~/Citrix/openshift# oc get vip

NAME                    VIP         AGE

ingress-apache-ingress   1.1.1.166   7s
```

サービスタイプLBの場合、サービスExternal-IPフィールドを確認して、NetScaler IPAM ControllerによってIPがサービスに割り当てられていることを確認できます。

サービスステータス

```
root@master:~/Citrix/openshift# oc get svc

NAME        TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)
 AGE

apache-lb   LoadBalancer   172.30.243.206   2.2.2.2       80:32498/TCP
  7s

root@master:~/Citrix/openshift# oc get vip

NAME                    VIP         AGE

service-apache-lb       2.2.2.2     16s
```

NetScaler IPAM Controllerのパラメータ

必須およびオプションのパラメータ

パラメータ	必須/オプション	デフォルト値	説明
イメージ	必須	quay.io/netscaler/netscaler-ipam-controller@sha256:7c25b59a2e1df0e4f5615a779a9bea939c60a3eca065cb6c2295daf0d03831d8	NetScaler IPAM Controllerイメージの詳細。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/オプション	デフォルト値	説明
pullPolicy	必須	IfNotPresent	NetScaler IPAM Controller のイメージプルポリシー。
reuseIngressVip	任意	真	同じ `vipRange` を使用するすべてのイングレスに同じ IP を使用できるようにします。
clusterName	infoblox.enabled が true の場合、必須	該当なし	IPAM コントローラーがデプロイされているクラスターを識別します。
tokenExpirationSeconds	必須	31536000	serviceAccount のトークンの有効期限が切れるまでの時間 (秒単位)。

ローカルIPAMパラメータ

パラメータ	必須/オプション	デフォルト値	説明
vipRange	必須	該当なし	IPAM VIP範囲を提供します。

Infobloxパラメータ

パラメータ	必須/オプション	デフォルト値	説明
gridHost	infoblox.enabled が true の場合、必須	N/A	Infoblox グリッドホストの IP または FQDN。
credentialSecret	infoblox.enabled が true の場合、必須	N/A	Infoblox 資格情報。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須/オプション	デフォルト値	説明
httpTimeout	任意	10	Infoblox クライアントの HTTP タイムアウト (秒単位)。
maxRetries	任意	3	Infoblox クライアントの失敗時の最大再試行回数。
netView	任意	デフォルト	インフォブロックスネットビュー。
vipRange	必須	N/A	インフォブロックス IPAM VIP レンジ。

NetScaler® Observability Exporter (NSOE) を NetScaler Operator を使用してデプロイする

NetScaler Observability Exporter は、NetScaler からメトリクスとトランザクションを収集し、JSON や AVRO などの適切な形式に変換するコンテナです。NSOE によって収集されたデータは、分析のために任意の必要なエンドポイントにエクスポートでき、NetScaler デバイスによってプロキシされたアプリケーションのマイクロサービスレベルで貴重な洞察を得ることができます。

前提条件

- Red Hat OpenShift Cluster (バージョン 4.1 以降)。
- NetScaler Operator をデプロイします。NetScaler Operator のデプロイ を参照してください。
- NSOE は任意のユーザー ID (uid) を介して動作するため、NSOE がデプロイされる名前空間に次のセキュリティコンテキスト制約 (SCC) をデプロイします。

```
oc adm policy add-scc-to-user anyuid
system:serviceaccount:<namespace>:default
```

NetScaler Operator を使用して NetScaler Observability Exporter をデプロイする

次の手順を実行します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

1. OpenShift 4.x クラスターコンソールにログインします。
2. **Operators > Installed Operators** に移動し、**NetScaler Operator** を選択します。

3. **NetScaler Observability Exporter** をクリックし、**Create NetScalerObservabilityExporter** を選択します。

   NetScaler Observability Exporter の YAML 定義が表示されます。オプションで、`Form view` ボタンを選択し、フォームでパラメータを更新できます。

4. インストール中に設定できる必須およびオプションのパラメータとそのデフォルト値を一覧表示する以下の表を参照してください。

パラメータ	必須またはオプション	デフォルト値	説明
acceptLicense	必須	no	NSOE EULAに同意するには、この値を `Yes` に設定します。
アフィニティ	オプション	N/A	Podの割り当て用アフィニティラベル。
nodePortRequired	オプション	偽	NSOEノードポートサービスを作成するには、このパラメータをtrueに設定します。
nodeSelector	任意	該当なし	NSOEデプロイメントのnodeSelectorオプションに使用されるノードラベルのキーと値のペア。
podAnnotations	任意	該当なし	ポッドに追加するアノテーションのマップ。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須またはオプション	デフォルト値	説明
pullPolicy	任意	IfNotPresent	NSOEイメージのプルポリシー。
レプリカ	任意	1	任意の時点で実行されるNSOEポッドの数。
リソース	任意	N/A	NSOEコンテナのCPU/メモリリソース。
トレランス	任意	N/A	NSOEデプロイメントのtolerations。
ヘッドレス	任意	フォールス	ヘッドレスサービスを作成するには、このパラメーターを `true` に設定します。
イメージ	必須	quay.io/netscaler/netscaler-observability-exporter@最新のNSOEイメージのshavalue	Quay.ioでホストされているNSOEイメージ。
nsoeLogLevel	任意	INFO	NSOEによって生成されるログを制御するためのログレベルです。サポートされているログレベルは次のとおりです: `CRITICAL`, `ERROR`, `WARNING`, `INFO`, `DEBUG`, `TRACE`, および `NONE`。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須または任意	デフォルト値	説明
: **エラスティックサーチ**:			
有効	任意	偽	トランザクション用のElasticsearchエンドポイントを有効にするには、このパラメータを`true`に設定します。
サーバー	オプション	`<http://elasticsearch:9200>`	Elasticsearchエンドポイント。

パラメータ	必須またはオプション	デフォルト値	説明
: **jsonTransRateLimiting**:			
有効	オプション	偽	`Splunk`、`ElasticSearch`、および`Zipkin`などのJSONベースのエンドポイントのトランザクションのレート制限を有効にするには、このパラメータを`true`に設定します。
制限	任意	100	JSONトランザクションのレート制限。100は、約800 TPSに相当します。
キュー制限	任意	1000	NSOEがトランザクションデータを破棄し始める前に保存で

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須またはオプション	デフォルト値	説明
			きるトランザクションデータの量。Zipkinの場合、1000は約64 MBのデータに相当します。SplunkおよびElasticSearchの場合、1000は約32 MBのデータに相当します。
ウィンドウ	任意	5	再計算ウィンドウ（秒単位）。ウィンドウサイズが小さいほど（0より大きい任意の値）、レート制限はより効果的になります。ただし、ウィンドウサイズを小さく設定すると、CPUオーバーヘッドが発生します。

パラメータ	必須または任意	デフォルト値	説明
: **カフカ**:			
監査ログ	オプション	いいえ	監査ログをKafkaにエクスポートするには、このパラメーターを`yes`に設定します。
ブローカー	オプション	N/A	KafkaブローカーのIPアドレス。
データ形式	オプション	AVRO	Kafkaにエクスポートされるデータの形式。指定可能な値は

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	必須または任意	デフォルト値	説明
			`JSON`と`AVRO`です。
有効	任意	偽	トランザクションデータをKafkaサーバーに送信できるようにするには、このパラメーターを`true`に設定します。
イベント	任意	no	イベントをKafkaにエクスポートするには、このパラメーターを`yes`に設定します。
トピック	任意	HTTP	データをアップロードするためのKafkaトピックの詳細。

パラメーター	必須または任意	デフォルト値	説明
: **エヌエストレーシング**:			
有効	オプション	偽	トレースデータをトレースサーバーに送信できるようにするには、このパラメーターを`true`に設定します。
サーバー	オプション	zipkin:9411/api/v1/spans	トレースサーバーのAPIエンドポイント。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須またはオプション	デフォルト値	説明
: **スプランク**:			
authToken	任意	該当なし	Splunk の認証トークン。
有効	任意	いいえ	トランザクションデータを Splunk サーバーに送信できるようにするには、このパラメーターを `true` に設定します。
indexPrefix	任意	該当なし	トランザクションをアップロードするための Splunk インデックスプレフィックス。
サーバー	任意	該当なし	Splunk サーバーの API エンドポイント。

パラメータ	必須またはオプション	デフォルト値	説明
: **時系列**:			
有効	オプション	いいえ	このパラメータを `true` に設定すると、Prometheus形式でのメトリクスデータアップロードが有効になります。現在、Prometheus のみがサポートされているメトリクスエンドポイントです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注記:

NSOEは複数の名前空間にデプロイできます。また、各インスタンスのデプロイ名が異なる限り、同じ名前空間にNSOEの複数のインスタンスをデプロイすることも可能です。デプロイする前に、ターゲットの名前空間に前提条件の`any-uid SCC`がデプロイされていることを確認してください。

5. 必要なパラメータの値を更新したら、**作成**をクリックします。

   NetScaler Observability Exporterが正常にデプロイされていることを確認します。

   ☐

6. **ワークロード > Pods**セクションに移動し、NetScaler Observability Exporterポッドが稼働中であることを確認します。

   ☐

© 1997–2026 Citrix Systems, Inc. All rights reserved.

プラットフォーム統合

プラットフォーム統合には、次のトピックがあります。

- OpenShift
- クラウド
- VMware Cloud Foundation 上の vSphere Kubernetes サービス での ネットスケーラー

OpenShift

OpenShift には以下のトピックがあります。

- NetScaler イングレスコントローラーをOpenShift ルーターとして展開する(/ja-jp/netscaler-k8s-ingress-controller/deploy/cic-openshift.html)
- OpenShift ルーターシャーディングサポート付きでNetScaler イングレスコントローラーを展開する(/ja-jp/netscaler-k8s-ingress-controller/deploy/openshift-sharding.html)

NetScaler Kubernetes Ingress Controller を OpenShift ルータープラグインとして展開する

OpenShift クラスターでは、外部クライアントがサービスにアクセスする方法が必要です。OpenShift は、クラスター内で実行されているサービスと通信するための次の2つのリソースを提供します。

- ルート
- イングレス

ルートは、特定のドメイン名でサービスを公開するか、ドメイン名をサービスに関連付けます。OpenShift ルーターは、ルートで指定されたルールに従って、外部リクエストをクラスター内のサービスにルーティングします。OpenShift ルーターを使用する場合、トラフィックが OpenShift ルーターに到達するように外部 DNS を構成する必要があります。

NetScaler Ingress Controller は、環境に展開された NetScaler と統合するためのルータープラグインとして展開できます。この展開により、OpenShift で NetScaler の高度な負荷分散およびトラフィック管理機能を使用できるようになります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サポートされているルートタイプ

OpenShift ルートは、セキュリティで保護されている場合と保護されていない場合があります。セキュリティで保護されたルートは、ルートの TLS 終端を指定します。

NetScaler Ingress Controller は、次の OpenShift ルートをサポートしています。

- **保護されていないルート**: 保護されていないルートの場合、HTTP トラフィックは暗号化されません。
- **エッジ終端**: エッジ終端の場合、TLS はルーターで終端されます。ルーターから内部ネットワーク経由でエンドポイントへのトラフィックは暗号化されません。
- **パススルー終端**: パススルー終端では、ルーターは TLS オフロードに関与せず、暗号化されたトラフィックは直接宛先に送信されます。
- **再暗号化終端**: 再暗号化終端では、ルーターは TLS 接続を終端しますが、その後、エンドポイントへの別の TLS 接続を確立します。

ルートの詳細については、OpenShift ドキュメントを参照してください。

展開オプション

OpenShiftクラスターの外部にアプライアンス（NetScaler MPXまたはVPX）をデプロイするか、クラスター内にNetScaler CPXポッドをデプロイできます。NetScaler Ingress Controllerは、NetScalerをOpenShiftと統合し、ルートルールから自動的に構成します。

ルータープラグインの2つのデプロイモード：

- **サイドカー**: NetScaler Ingress Controllerを、NetScaler CPXと同じポッド内でサイドカーコンテナとして実行します。イングレスがCPXを構成します。
- **スタンドアロンポッド**: NetScaler Ingress Controllerをスタンドアロンで実行し、クラスター外のNetScaler MPXまたはVPXアプライアンスを制御します。

NetScaler Ingress ControllerでOpenShiftイングレスを制御する方法については、Kubernetesデプロイ手順を参照してください。

NetScalerを使用して、OpenShiftコントロールプレーン（マスターノード）のロードバランシングを行うことができます。NetScalerは、NetScalerを構成するためのTerraformベースの自動化を提供します。OpenShiftコントロールプレーンのロードバランサーとしてのNetScalerを参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

代替バックエンドのサポート

OpenShift代替バックエンドはNetScaler Ingress Controllerでサポートされています。NetScalerはルート定義の重みに従って構成され、トラフィックはそれらの重みに基づいてサービスポッド間で分散されます。

代替バックエンドを持つルートマニフェストの例：

```
kind: Route

apiVersion: route.openshift.io/v1

metadata:

  name: r1

  labels:

    name: apache

  annotations:

    route.citrix.com/frontend-ip: "<Frontend-ip>"

spec:

  host: some.alternate-backends.com

  to:

    kind: Service

    name: apache-1

    weight: 30

  alternateBackends:

    - kind: Service

      name: apache-2

      weight: 20
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    - kind: Service
      name: apache-3
      weight: 50
  port:
    targetPort: 80
  wildcardPolicy: None
```

このルートの場合：トラフィックの30%が`apache-1`に、20%が`apache-2`に、50%が`apache-3`に送られます。

OpenShiftでサポートされているCitrixコンポーネント

CITRIXコンポーネント	バージョン
NetScaler イングレスコントローラー	最新
NetScaler VPX	12.1 50.x 以降
NetScaler CPX	13.0–36.28 以降

注:

NetScaler Ingress Controller 用に提供される CRDs は、OpenShift ルートではサポートされていません。CRD を使用するには、OpenShift イングレスを使用してください。

例

次の例では、アプリケーションの負荷分散を行うために、NetScaler Ingress Controller をルータープラグインとして展開します。

1. サンプルアプリケーション (`apache.yaml`) を展開し、それを公開します。

```
oc create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/openshift/manifest/apache.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

OpenShift では Apache Pod が失敗する可能性があります。これは、それらが root として実行されるか、ポート 80 にバインドされることが多いためです。回避策として、Pod のデフォルトサービスアカウントを特権 SCC に追加し、`anyuid` を追加します。

`oc adm policy add-scc-to-user privileged system:serviceaccount:default:default` `oc adm policy add-scc-to-group anyuid system:authenticated`

apache.yaml ファイルの内容は次のとおりです。

```
---

apiVersion: apps/v1

kind: Deployment

metadata:

  labels:

    name: apache-only-http

  name: apache-only-http

spec:

  replicas: 4

  selector:

    matchLabels:

      app: apache-only-http

  template:

    metadata:

      labels:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        app: apache-only-http
    spec:
      containers:
      - image: quay.io/sample-apps/apache:1.0.0
        imagePullPolicy: IfNotPresent
        name: apache-only-http
        ports:
        - containerPort: 80
          protocol: TCP
        - containerPort: 5080
          protocol: TCP
        - containerPort: 5081
          protocol: TCP
        - containerPort: 5082
          protocol: TCP
---
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    name: apache-only-ssl
  name: apache-only-ssl
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:
  replicas: 4
  selector:
    matchLabels:
      app: apache-only-ssl
  template:
    metadata:
      labels:
        app: apache-only-ssl
    spec:
      containers:
      - image: quay.io/sample-apps/apache:1.0.0
        imagePullPolicy: IfNotPresent
        name: apache-only-ssl
        ports:
        - containerPort: 443
          protocol: TCP
        - containerPort: 5443
          protocol: TCP
        - containerPort: 5444
          protocol: TCP
        - containerPort: 5445
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          protocol: TCP
---
apiVersion: v1
kind: Service
metadata:
  name: svc-apache-multi-http
spec:
  ports:
  - name: apache-http-6080
    port: 6080
    targetPort: 5080
  - name: apache-http-6081
    port: 6081
    targetPort: 5081
  - name: apache-http-6082
    port: 6082
    targetPort: 5082
  selector:
    app: apache-only-http
---
apiVersion: v1
kind: Service
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
metadata:
  name: svc-apache-multi-ssl
spec:
  ports:
  - name: apache-ssl-6443
    port: 6443
    targetPort: 5443
  - name: apache-ssl-6444
    port: 6444
    targetPort: 5444
  - name: apache-ssl-6445
    port: 6445
    targetPort: 5445
  selector:
    app: apache-only-ssl
---
```

2. 「NetScaler Ingress Controller を Pod として展開する」の手順を使用して、NetScaler VPX 用の NetScaler Ingress Controller を OpenShift クラスターにスタンドアロン Pod として展開します。

注:

OpenShiftクラスターにNetScaler CPXと共にNetScaler Ingress Controllerを展開するには、「NetScaler CPXをサイドカーとしてNetScaler Ingress Controllerを展開する」の手順を実行します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

3. アプリケーションを公開するためのOpenShiftルートを作成します。オプションは次のとおりです。

   a. 保護されていないOpenShiftルート `unsecured-route.yaml` を作成するには、次のコマンドを使用します。

```
apiVersion: v1

kind: Route

metadata:

  name: unsecured-route

spec:

  host: unsecured-route.openshift.citrix-cic.com

  path: "/"

  to:

    kind: Service

    name: svc-apache-multi-http
```

```
oc create -f unsecured-route.yaml
```

   b. エッジ終端で保護されたOpenShiftルート `secured-edge-route.yaml` を作成するには、次のコマンドを使用します。

```
apiVersion: v1

kind: Route

metadata:

  name: secured-edge-route

spec:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  host: secured-edge-route.openshift.citrix-cic.com

  path: "/"

  to:

    kind: Service

    name: svc-apache-multi-http

  tls:

    termination: edge



    key: |-

      ----BEGIN RSA PRIVATE KEY-----



          [...]



      -----END RSA PRIVATE KEY-----



    certificate: |-

      -----BEGIN CERTIFICATE-----

          [...]

      -----END CERTIFICATE-----
```

```
oc create -f secured-route-edge.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

c. パススルー終端で保護されたOpenShiftルート `secured-passthrough-route.yaml` を作成するには、次のコマンドを使用します。

```
apiVersion: v1

kind: Route

metadata:

  name: secured-passthrough-route

spec:

  host: secured-passthrough-route.openshift.citrix-cic.com

  to:

    kind: Service

    name: svc-apache-multi-ssl

  tls:

    termination: passthrough
```

```
oc create -f secured-passthrough-route.yaml
```

d. 再暗号化終端で保護されたOpenShiftルート `secured-reencrypt-route.yaml` を作成するには、次のコマンドを使用します。

```
apiVersion: v1

kind: Route

metadata:

  name: secured-reencrypt-route

spec:

  host: secured-reencrypt-route.openshift.citrix-cic.com
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  path: "/"
  to:
    kind: Service
    name: svc-apache-multi-ssl
  tls:
    termination: reencrypt


    key: |-
      -----BEGIN RSA PRIVATE KEY-----
            [...]
      -----END RSA PRIVATE KEY-----


    certificate: |-
      -----BEGIN CERTIFICATE-----
            [...]
      -----END CERTIFICATE-----


    destinationCACertificate: |-
      -----BEGIN CERTIFICATE-----
            [...]
      -----END CERTIFICATE-----
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
oc create -f secured-reencrypt-route.yaml
```

注:

パススルー終端で保護されたOpenShiftルートの場合、デフォルトの証明書を含める必要があります。

4. 次のコマンドを使用してアプリケーションにアクセスします。

```
curl http://<VIP of NetScaler VPX>/ -H 'Host: < host-name-as-per-the-host-configuration-in-route >'
```

OpenShiftルーターシャーディングサポート付きNetScaler Ingress Controllerのデプロイ

OpenShiftルーターシャーディングを使用すると、複数のOpenShiftルーター間で一連のルートを分散できます。デフォルトでは、OpenShiftルーターはすべてのネームスペースからすべてのルートを選択します。ルーターシャーディングでは、ルートまたはネームスペースにラベルが追加され、ルートをフィルタリングするためにルーターにラベルセレクターが追加されます。各ルーターシャードは、そのラベル選択パラメーターに一致する特定のラベルを持つルートのみを選択します。

NetScalerは2つの方法でOpenShiftと統合でき、どちらのデプロイメントもOpenShiftルーターシャーディングをサポートしています。

- クラスター内にNetScaler Ingress ControllerとともにOpenShiftルーターとしてデプロイされたNetScaler CPX
- クラスター外にデプロイされたNetScaler MPXまたはVPX用のルータープラグインとしてのNetScaler Ingress Controller

OpenShift上のNetScalerデプロイメントでルーターシャーディングを構成するには、シャードごとにNetScaler Ingress Controllerインスタンスが必要です。NetScaler Ingress Controllerインスタンスは、シャーディングに必要な基準に応じて、ルートまたはネームスペースのラベル、あるいはその両方を環境変数としてデプロイされます。 NetScaler Ingress Controllerがルートを処理する際、そのルートのラベルまたはルートのネームスペースのラベルを、自身に構成された選択基準と比較します。ルートが基準を満たしていれば、適切な構成がNetScalerに適用され、そうでなければ構成は適用されません。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ルーターシャーディングでは、ルート全体のプールからルートのサブセットを選択することは、選択式に基づいています。選択式は、複数の値と操作の組み合わせです。

例えば、以下の表に示すように、サービスレベルアグリーメント (sla)、地理的位置 (geo)、ハードウェア要件 (hw)、部門 (dept)、タイプ、頻度など、さまざまなラベルを持つルートがいくつかあるとします。

ラベル	値
サービスレベル	高、中、低
地理	東、西
hw	控えめ、強力
部署	財務、開発、運用
タイプ	静的、動的
頻度	高、毎週

以下の表は、ルートラベルまたはネームスペースラベルのセレクターと、例のラベルに基づいたいくつかのサンプル選択式を示しています。ルート選択基準は、環境変数 ROUTE_LABELS と NAMESPACE_LABLES を使用して NetScaler Ingress Controller で構成されます。

セレクターの種類	例
OR演算	ROUTE_LABELS='部門 in (開発, 運用)'
AND演算	ROUTE_LABELS='ハードウェア=強力,タイプ=動的,地域=西'
NOT演算	ルートラベル='dept!= finance'
完全一致	ネームスペースラベル='frequency=weekly'
ルートラベルとネームスペースラベルの両方で完全一致	ネームスペースラベル='frequency=weekly' ルートラベル='sla=low'
値に依存しないキーベースの一致	ネームスペースラベル='name'
値に依存しないキーベースの一致でのNOT操作	ネームスペースラベル='!name'

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

ラベルセレクターは、Kubernetesラベルでサポートされている言語を使用します。

必要に応じて、後で編集することでルートまたはネームスペースのラベルを変更できます。ラベルを変更すると、ルーターシャードが再検証され、その変更に基づいてNetScaler Ingress ControllerがNetScaler上の構成を更新します。

OpenShiftルーターシャーディングでNetScaler CPXを展開する

OpenShiftルーターシャーディングでCPXを展開するには、NetScaler Operatorを使用してNetScaler CPXのサイドカーとしてNetScaler Ingress Controllerを展開するのステップ6で指定されているように、イングレスコントローラーを持つCPXを作成するためにYAML定義を編集し、以下の引数を指定する必要があります。

```
routeLabels: "name in (abc,xyz)"

namespaceLabels: "frequency=high"
```

OpenShiftルーターシャーディングサポート付きNetScaler Ingress Controllerルータープラグインを展開する

NetScaler Ingress Controllerルータープラグインをルーターシャーディングでデプロイするには、Deploy NetScaler Ingress Controller in OpenShift using NetScaler Operator のステップ6で指定されているように、NSICを作成するためにYAML定義を編集し、以下の引数を指定する必要があります。

```
routeLabels: "name in (abc,xyz)"

namespaceLabels: "frequency=high"
```

例: OpenShiftルートを作成し、NetScaler VPXでルート構成を確認する

この例では、ラベル付きのOpenShiftルートを作成し、ルーターシャード構成を確認する方法を示します。 この例では、NetScaler VPX展開でルート構成が検証されます。

ラベル付きのサンプルルートを作成するには、以下の手順を実行します。

1. YAMLファイルでルートを定義します。以下は、`route.yaml`という名前のサンプルルートの例です。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: v1

kind: Route

metadata:

    name: web-backend-route

    namespace: default

    labels:

        sla: low

        name: abc

spec:

    host: web-frontend.cpx-lab.org

    path: "/web-backend"

    port:

        targetPort: 80

    to:

        kind: Service

        name: web-backend
```

2. ルートをデプロイするには、以下のコマンドを使用します。

```
oc create -f route.yaml
```

3. ルートを作成する名前空間にラベルを追加します。

```
oc label namespace default 'frequency=high'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ルート構成の確認

以下の手順を実行することで、NetScaler VPXでOpenShiftルート構成を確認できます。

1. 以下の手順を実行して、NetScaler VPXにログオンします。
   - PuTTyなどのSSHクライアントを使用して、NetScaler VPXへのSSH接続を開きます。
   - 管理者資格情報を使用してNetScaler VPXにログオンします。
2. 以下のコマンドを使用して、サービスグループが作成されているか確認します。

   ```
   show serviceGroup
   ```

3. `show serviceGroup`コマンド出力でNetScaler VPX上のルート構成を確認します。

   以下は、`show serviceGroup`コマンド出力からのサンプルルート構成です。

   ```
   > show serviceGroup

   k8s-web-backend-route_default_80_k8s-web-backend_default_80_svc - HTTP

   State: ENABLED  Effective State: DOWN Monitor Threshold : 0

   Max Conn: 0     Max Req: 0 Max Bandwidth: 0 kbits

   Use Source IP: NO

   Client Keepalive(CKA): NO

   TCP Buffering(TCPB): NO

   HTTP Compression(CMP): NO

   Idle timeout: Client: 180 sec    Server: 360 sec

   Client IP: DISABLED

   Cacheable: NO

   SC: OFF

   SP: OFF
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
Down state flush: ENABLED

Monitor Connection Close : NONE

Appflow® logging: ENABLED

ContentInspection profile name: ???

Process Local: DISABLED

Traffic Domain: 0
```

クラウド

クラウドには以下のトピックがあります。

- Azure Kubernetes EngineにNetScaler CPX™をIngressとして展開する
- NetScaler VPX™を使用してAzure Kubernetes ServiceクラスターにNetScaler ingress controllerを展開する
- Google Cloud PlatformにNetScaler CPXをIngressとして展開する
- Anthos Platform に NetScaler ingress controller をデプロイする(/ja-jp/netscaler-k8s-ingress-controller/deploy/anthos.html)
- Amazon ELBおよびNetScaler ingress controllerを使用して、EKS環境でアクティブ-アクティブ高可用性構成のNetScaler VPXを展開する
- AWSにおけるLoadBalancerタイプのサービス向けCitrix®ソリューション
- Amazon EKSおよびMicrosoft AKSクラスターを使用したマルチクラウドおよびGSLBソリューション

Azure Kubernetes ServiceクラスターにNetScaler CPXをIngressデバイスとして展開する

このトピックでは、Azure Kubernetes Service (AKS) クラスターにNetScaler CPXをIngressデバイスとして展開する方法について説明します。NetScaler CPXは、AKSのAdvanced Networking (Azure CNI)とBasic Networking (Kubenet)の両方のモードをサポートしています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

デフォルトのquay.ioイメージの代わりに、NetScaler CPXまたはNetScaler Ingress ControllerにAzureリポジトリイメージを使用する場合は、(/ja-jp/netscaler-k8s-ingress-controller/deploy/azure-image.html)を参照してください。

AKSクラスターにNetScaler CPXをIngressデバイスとして展開する

AKSクラスターにNetScaler CPXをIngressデバイスとして展開するには、次の手順を実行します。

注:

この手順では、Apache Webサーバーがサンプルアプリケーションとして使用されます。

1. 必要なアプリケーションをKubernetesクラスターに展開し、次のコマンドを使用してクラスター内でサービスとして公開します。

   ```
   kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/azure/manifest/apache.yaml
   ```

   注:

   この例では、`apache.yaml`が使用されます。ご使用のアプリケーションには、特定のYAMLファイルを使用する必要があります。

2. 次のコマンドを使用して、NetScaler CPXをクラスターにIngressデバイスとして展開します。

   ```
   kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/azure/manifest/standalone_cpx.yaml
   ```

3. 次のコマンドを使用して、Ingressリソースを作成します。

   ```
   kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/azure/manifest/cpx_ingress.yaml
   ```

4. 次のコマンドを使用して、NetScaler CPXにアクセスするためのLoadBalancerタイプのサービスを作成します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/azure/manifest/cpx_service.yaml
```

このコマンドは、トラフィックを受信するための外部IPを持つAzureロードバランサーを作成します。

5. サービスを確認し、ロードバランサーが外部IPを作成したかどうかを確認します。外部IPが作成されていない場合は、しばらくお待ちください。

```
kubectl  get svc



|NAME|TYPE|CLUSTER-IP|EXTERNAL-IP|PORT(S)| AGE|

|----|----|-----|-----|----|----|

|apache |ClusterIP|10.0.103.3|none|   80/TCP | 2m|

|cpx-ingress |LoadBalancer |10.0.37.255 | pending |80:32258/
TCP,443:32084/TCP |2m|

|Kubernetes |ClusterIP | 10.0.0.1 |none |  443/TCP | 22h |
```

6. ロードバランサーの外部IPが次のように利用可能になったら、ロードバランサーの外部IPを使用してリソースにアクセスできます。

```
kubectl  get svc



|NAME|TYPE|CLUSTER-IP|EXTERNAL-IP|PORT(S)|  AGE|

|---|---|----|----|----|----|

|apache|ClusterIP|10.0.103.3 |none|80/TCP|  3m|

|cpx-ingress |LoadBalancer|10.0.37.255|  EXTERNAL-IP CREATED| 80:32258/
TCP,443:32084/TCP |  3m|
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
|Kubernetes|    ClusterIP|10.0.0.1 |none| 443/TCP| 22h|
```

注:

クラウドロードバランサーのヘルスチェックは、NetScaler CPXデプロイyamlファイルで設定されたreadinessProbeから取得されます。ヘルスチェックが失敗した場合は、NetScaler CPX用に設定されたreadinessProbeを確認する必要があります。詳細については、readinessProbeおよび外部ロードバランサーを参照してください。

7. 次のコマンドを使用してアプリケーションにアクセスします。

```
curl http://&lt;External-ip-of-loadbalancer>/ -H 'Host: citrix-ingress.com
```

クイックデプロイ

デプロイを容易にするために、前のトピックで説明した手順を組み合わせた単一のオールインワンマニフェストをデプロイするだけで済みます。

1. all-in-one.yamlを使用して、組み込みのNetScaler Ingress Controllerを備えたNetScaler CPXイングレスをKubernetesクラスターにデプロイします。

```
kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/azure/manifest/all-in-one.yaml
```

2. 次のコマンドを使用してアプリケーションにアクセスします。

```
curl http://&lt;External-ip-of-loadbalancer>/ -H 'Host: citrix-ingress.com'
```

注:

デプロイを削除するには、`kubectl delete -f all-in-one.yaml`コマンドを使用します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Ingress ControllerをNetScaler VPXと共にAzure Kubernetes Serviceクラスターに展開する

このトピックでは、NetScaler Ingress ControllerをNetScaler VPXと共にAzure Kubernetes Service (AKS)クラスターに展開する方法について説明します。また、Azure VMs上にKubernetesクラスターを構成し、NetScaler Ingress ControllerをNetScaler VPXと共に展開することもできます。

AKSとAzure VMの両方への展開手順は同じです。ただし、Azure VM上にKubernetesを構成する場合は、Kubernetesクラスター用のCNIプラグインを展開する必要があります。

前提条件

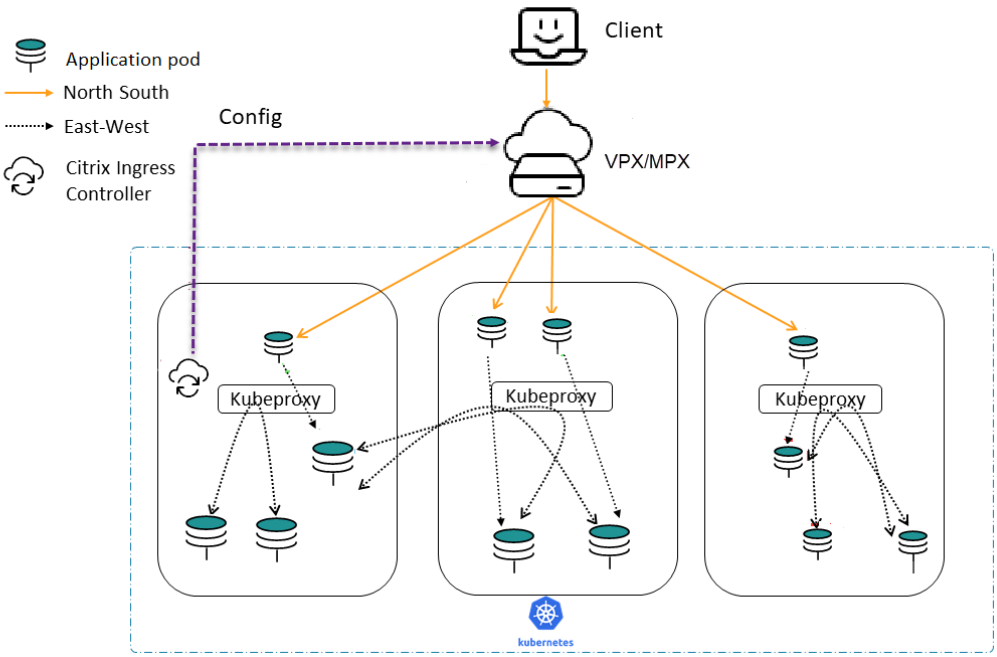
手順のステップを実行する前に、以下のタスクを完了する必要があります。

- Kubernetesクラスターが稼働していることを確認します。

> **注:**
>
> AKSでKubernetesクラスターを作成する方法の詳細については、AKSクラスターの作成ガイドを参照してください。

トポロジ

以下は、この展開で使用されるサンプル トポロジです。



© 1997–2026 Citrix Systems, Inc. All rights reserved.

Azure MarketplaceからNetScaler VPXインスタンスを取得する

Azure MarketplaceからNetScaler VPXを作成できます。 Azure MarketplaceからNetScaler VPXインスタンスを作成する方法の詳細については、Azure MarketplaceからNetScaler VPXを取得するを参照してください。

Azure マーケットプレイスからNetScaler Ingress Controllerを取得する

NetScaler Ingress Controllerを展開するには、Azure上にイメージレジストリを作成し、対応するイメージURLを使用してNetScaler Ingress Controllerイメージを取得する必要があります。

レジストリの作成方法とイメージURLの取得方法の詳細については、Azure MarketplaceからNetScaler Ingress Controllerを入手するを参照してください。

レジストリが作成されたら、NetScaler Ingress Controllerのレジストリ名をデプロイに使用するAKSクラスターにアタッチする必要があります。

```
az aks update -n &lt;cluster-name> -g &lt;resource-group-where-aks-deployed> --attach-acr &lt;cic-registry>
```

NetScaler Ingress Controllerのデプロイ

NetScaler Ingress Controllerをデプロイするには、次の手順を実行します。

1. Kubernetesシークレットを使用してNetScaler VPXのログイン資格情報を作成します。

   ```
   kubectl create secret  generic nslogin --from-literal=username='&lt;azure-vpx-instance-username>' --from-literal=password='&lt;azure-vpx-instance-password>'
   ```

   注:

   NetScaler VPXのユーザー名とパスワードは、AzureでNetScaler VPXを作成する際に設定した資格情報と同じである必要があります。

2. SSHを使用して、NetScaler VPXのセカンダリIPアドレスであるSNIPをNetScaler VPXに構成します。この手順は、NetScalerがKubernetesクラスター内のポッドと対話するために必要です。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
add ns ip &lt;snip-vpx-instance-private-ip> &lt;vpx-instance-primary-ip-subnet>
```

- `snip-vpx-instance-private-ip`は、NetScaler VPXインスタンスの作成中にSNIPを追加する際に割り当てられる動的プライベートIPアドレスです。
- `vpx-instance-primary-ip-subnet`は、NetScaler VPXインスタンスのプライマリプライベートIPアドレスのサブネットです。

プライベートIPアドレスのサブネットを確認するには、NetScaler VPXインスタンスにSSHで接続し、次のコマンドを使用します。

```
show ip &lt;primary-private-ip-addess>
```

3. NetScaler Ingress ControllerのYAMLファイルで、NetScaler VPXのイメージURL、管理IP、およびVIPを更新します。

   a. NetScaler Ingress ControllerのYAMLファイルをダウンロードします。

   ```
   wget https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/azure/manifest/azurecic/cic.yaml
   ```

   注:

   `wget`がインストールされていない場合は、`fetch`または`curl`コマンドを使用できます。

   b. `cic.yaml`ファイルで、NetScaler Ingress ControllerイメージをAzureイメージURLで更新します。

   ```
   - name: cic-k8s-ingress-controller

     # CIC Image from Azure

     image: "&lt;azure-cic-image-url>"
   ```

   c. `cic.yaml`の以下のフィールドで、NetScaler VPXのプライマリIPアドレスをAzure VPXインスタンスのプライベートプライマリIPアドレスで更新します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
# Set NetScaler NSIP/SNIP, SNIP in case of HA (mgmt has to be
enabled)

- name: "NS_IP"

value: "X.X.X.X"
```

d. `cic.yaml`の以下のフィールドで、NetScaler VPX VIPを、VPX Azureインスタンス作成時に割り当てられたVIPのプライベートIPアドレスで更新します。

```
# Set NetScaler® VIP for the data traffic

- name: "NS_VIP"

  value: "X.X.X.X"
```

4. 必要な値でNetScaler Ingress Controllerを設定したら、次のコマンドを使用してNetScaler Ingress Controllerを展開します。

```
kubectl create -f cic.yaml
```

サンプルアプリケーションを使用してデプロイを検証する

1. 次のコマンドを使用して、必要なアプリケーションをKubernetesクラスターに展開し、クラスターでサービスとして公開します。

```
kubectl create -f  https://raw.githubusercontent.com/citrix/citrix-k8s-
ingress-controller/master/deployment/azure/manifest/azurecic/
apache.yaml
```

2. 次のコマンドを使用してIngressリソースを作成します。

```
kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-
ingress-controller/master/deployment/azure/manifest/azurecic/
ingress.yaml
```

3. デプロイを検証するには、次のコマンドを使用します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
$ curl --resolve citrix-ingress.com:80:&lt;Public-ip-address-of-VIP>
http://citrix-ingress.com

&lt;html>&lt;body><h1>It works!</h1>&lt;/body>&lt;/html>
```

レスポンスは、Kubernetesクラスター内にあるサンプルマイクロサービス (Apache) から受信されます。NetScaler VPXはリクエストをロードバランシングしました。

Google Cloud Platform に Ingress デバイスとして NetScaler CPX を展開する

このトピックでは、Google Kubernetes Engine (GKE) に Ingress デバイスとして NetScaler CPX を展開する方法について説明します。

前提条件

この手順のステップを実行する前に、以下のタスクを完了する必要があります。

- Kubernetes クラスターが稼働していることを確認します。
- GKE でクラスターを実行している場合は、cluster-admin ロールバインディングが構成されていることを確認してください。

cluster-admin ロールバインディングを構成するには、次のコマンドを使用します。

```
kubectl create clusterrolebinding citrix-cluster-admin --
clusterrole=cluster-admin --user=&lt;email-id of your google account>
```

次のコマンドを使用して、Google アカウントの詳細を取得できます。

```
gcloud info | grep Account
```

Google Cloud Platform に Ingress デバイスとして NetScaler CPX を展開する

1. 次のコマンドを使用して、必要なアプリケーションを Kubernetes クラスターに展開し、クラスター内でサービスとして公開します。

```
kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-
ingress-controller/master/deployment/gcp/manifest/apache.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

この例では、`apache.yaml` が使用されています。ご使用のアプリケーションには、特定の YAML ファイルを使用する必要があります。

2. 次のコマンドを使用して、NetScaler CPX をクラスターに Ingress デバイスとして展開します。

```
kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/gcp/manifest/standalone_cpx.yaml
```

3. 次のコマンドを使用して、Ingress リソースを作成します。

```
kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/gcp/manifest/cpx_ingress.yaml
```

4. 次のコマンドを使用して、NetScaler CPX にアクセスするための LoadBalancer タイプのサービスを作成します。

```
kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/gcp/manifest/cpx_service.yaml
```

注記:

このコマンドは、トラフィックを受信するための外部IPアドレスを持つロードバランサーを作成します。

1. サービスを確認し、ロードバランサーが外部IPアドレスを作成したかどうかを確認します。外部IPアドレスが作成されていない場合は、しばらくお待ちください。

```
kubectl  get svc



|NAME | TYPE | CLUSTER-IP | EXTERNAL-IP | PORT(S) | AGE |

| --- | ---| ----| ----| ----| ----|
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
|apache | ClusterIP |10.7.248.216 |none |  80/TCP | 2m |

|cpx-ingress |LoadBalancer | 10.7.241.6 |  pending | 80:32258/
TCP,443:32084/TCP | 2m|

|kubernetes |ClusterIP |10.7.240.1 |none | 443/TCP | 22h|
```

2. ロードバランサーの外部IPアドレスが次のように利用可能になったら、ロードバランサーの外部IPアドレスを使用してリソースにアクセスできます。

```
kubectl  get svc



|Name | Type | Cluster-IP | External IP| Port(s) | Age |

|-----| -----| -------| -----| -----| ----|

|apache| ClusterIP|10.7.248.216|none|80/TCP |3m|

|cpx-ingress|LoadBalancer|10.7.241.6|EXTERNAL-IP CREATED|80:32258/
TCP,443:32084/TCP|3m|

|kubernetes| ClusterIP| 10.7.240.1|none|443/TCP|22h|`
```

注記: クラウドロードバランサーのヘルスチェックは、NetScaler CPXサービスYAMLファイルで構成されたreadinessProbeから取得されます。ヘルスチェックが失敗した場合は、NetScaler CPX用に構成されたreadinessProbeを確認する必要があります。 詳細については、readinessProbeおよび外部ロードバランサーを参照してください。

3. 次のコマンドを使用してアプリケーションにアクセスします。

```
curl http://&lt;External-ip-of-loadbalancer>/ -H 'Host: citrix-
ingress.com'
```

クイックデプロイ

デプロイを容易にするために、前のトピックで説明した手順を組み合わせた単一のオールインワンマニフェストをデプロイするだけで済みます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

1. 組み込みのNetScaler Ingress Controllerを備えたNetScaler CPXインプレスを、all-in-one.yaml を使用してKubernetesクラスターにデプロイします。

```
kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/gcp/manifest/all-in-one.yaml
```

2. 次のコマンドを使用してアプリケーションにアクセスします。

```
curl http://&lt;External-ip-of-loadbalancer>/ -H 'Host: citrix-ingress.com'
```

注記:

デプロイを削除するには、`kubectl delete -f all-in-one.yaml`コマンドを使用します。

Anthos に NetScaler Ingress Controller を展開する

Anthos は、既存のオンプレミスハードウェアまたはパブリッククラウドでアプリケーションを実行できるハイブリッドおよびマルチクラウドプラットフォームです。クラウドおよびオンプレミス環境向けに一貫した開発および運用エクスペリエンスを提供します。

NetScaler Ingress Controller は、以下の展開モードを使用して Anthos GKE オンプレミスに展開できます。

- サイドカーイングレスコントローラーを備えた NetScaler CPX を `LoadBalancer` タイプのサービスとして公開する。
- デュアルティアイングレス展開

Anthos GKE オンプレミスで NetScaler CPX を `LoadBalancer` タイプのサービスとして公開する

この展開では、NetScaler VPX または MPX はクラスター外の Tier-1 に展開され、NetScaler CPX は Anthos クラスター内の Tier-2 に展開されます。これはデュアルティア展開に似ています。ただし、Ingress を使用する代わりに、NetScaler CPX は `LoadBalancer` タイプの Kubernetes サービスを使用して公開されます。 NetScaler Ingress Controller は、サービス仕様の `LoadBalancerIP` フィールドで提供される IP アドレスを構成するプロセスを自動化します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

前提条件

- Anthos GKE オンプレミスユーザークラスターと同じサブネットに Tier-1 NetScaler VPX または MPX を展開する必要があります。
- Tier-1 NetScaler でサブネット IP アドレス (SNIP) を構成する必要があり、Anthos GKE オンプレミス クラスターノードはその IP アドレスを使用して到達可能である必要があります。
- 異なるネットワークから NetScaler VPX または MPX を使用するには、ノードコントローラーを使用して、NetScaler と Anthos GKE オンプレミス クラスター間の通信を有効にします。
- ロードバランサー IP アドレスとして使用する仮想 IP アドレス (VIP) を確保する必要があります。

Anthos GKE オンプレミスで NetScaler CPX を `LoadBalancer` タイプのサービスとして展開する

Anthos GKE オンプレミスで NetScaler CPX を `LoadBalancer` タイプのサービスとして展開するには、次の手順を実行します。

1. 必要なアプリケーションを Kubernetes クラスターに展開し、次のコマンドを使用してクラスター内でサービスとして公開します。

   ```
   kubectl --kubeconfig user-cluster-1-kubeconfig create -f https://
   raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/
   deployment/anthos/manifest/service-type-lb/apache.yaml
   ```

   注:

   この例では、`apache.yaml` が使用されています。アプリケーションには、特定のYAMLファイルを使用する必要があります。

2. cpx-cic.yaml ファイルを使用して、サイドカーNetScaler Ingress ControllerをTier-2 IngressデバイスとしてNetScaler CPXを展開します。

   ```
   kubectl --kubeconfig user-cluster-1-kubeconfig create -f https://
   raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/
   deployment/anthos/manifest/service-type-lb/cpx-cic.yaml
   ```

3. (オプション) TLS構成のためにIngressで使用する自己署名SSL証明書とキーを作成します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
openssl req -subj '/CN=anthos-citrix-ingress.com/O=Citrix Systems Inc/
C=IN' -new -newkey rsa:2048 -days 5794 -nodes -x509 -keyout $PWD/
anthos-citrix-certificate.key -out $PWD/anthos-citrix-
certificate.crt;openssl rsa -in $PWD/anthos-citrix-certificate.key -out
$PWD/anthos-citrix-certificate.key
```

注:

すでにSSL証明書をお持ちの場合は、それを使用してKubernetesシークレットを作成できます。これは自己署名証明書を作成するための単なるコマンド例であり、このコマンドはアプリケーションのホスト名が `anthos-citrix-ingress.com` であると想定しています。

4. 作成したSSL証明書とキーのペアを使用してKubernetesシークレットを作成します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig create secret tls
anthos-citrix --cert=$PWD/anthos-citrix-certificate.crt --key=$PWD/
anthos-citrix-certificate.key
```

5. tier-2-ingress.yaml ファイルを使用して、Tier-2用のIngressリソースを作成します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig create -f https://
raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/
deployment/anthos/manifest/service-type-lb/tier-2-ingress.yaml
```

6. Tier-1 NetScaler用のKubernetesシークレットを作成します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig create secret  generic
nslogin --from-literal=username='citrix-adc-username' --from-
literal=password='citrix-adc-password'
```

7. NetScaler Ingress ControllerをTier-1 Ingressコントローラーとして展開します。

   a. cic.yaml ファイルをダウンロードします。

   b. NetScalerの管理IPアドレスを入力します。`cic.yaml` ファイルで指定されたプレースホルダー `Tier-1-Citrix-ADC-IP` に、Tier-1 NetScalerの管理IPアドレスを更新します。

   c. 次のコマンドを使用して `cic.yaml` を保存して展開します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl --kubeconfig user-cluster-1-kubeconfig create -f cic.yaml
```

8. NetScaler CPXを`LoadBalancer`タイプのKubernetesサービスとして公開します。

   a. cpx-service-type-lb.yaml ファイルをダウンロードします。

   b. YAMLファイルを編集し、`VIP-for-accessing-microservices` の値を、クラスター内のアプリケーションにアクセスするために使用されるVIPアドレスとして指定します。このVIPアドレスは、ロードバランサーIPアドレスとして使用するために確保されているものです。

   c. 次のコマンドを使用して、`cpx-service-type-lb.yaml` ファイルを保存してデプロイします。

```
kubectl --kubeconfig user-cluster-1-kubeconfig create -f cpx-
service-type-lb.yaml
```

9. マイクロサービスにアクセスするために、`VIP-for-accessing-microservices` のIPアドレスでDNSレコードを更新します。この例では、Apacheマイクロサービスにアクセスするには、次のDNSエントリが必要です。

```
`<VIP-for-accessing-microservices> anthos-citrix-ingress.com`
```

10. 次のコマンドを使用してアプリケーションにアクセスします。

```
curl -k --resolve anthos-citrix-ingress.com:443:&lt;VIP-for-accessing-
microservices> https://anthos-citrix-ingress.com/
&lt;html>&lt;body><h1>It works!</h1>&lt;/body>&lt;/html>
```

注:

このコマンドでは、デモンストレーション目的で、ステップ9のDNS構成部分を上書きするために `--resolve anthos-citrix-ingress.com:443:<VIP-for-accessing-microservices>` が使用されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

インストールのクリーンアップ: NetScaler CPXを `LoadBalancer` タイプのサービスとして公開する

インストールをクリーンアップするには、`kubectl --kubeconfig delete` コマンドを使用して各デプロイメントを削除します。

NetScaler CPXサービスデプロイメント（CPX+CICサービス）を削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete -f cpx-service-type-lb.yaml
```

Tier-2 Ingressオブジェクトを削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete -f tier-2-ingress.yaml
```

サイドカーNetScaler Ingress ControllerとともにNetScaler CPXデプロイメントを削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete -f cpx-cic.yaml
```

スタンドアロンのNetScaler Ingress Controllerを削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete -f cic.yaml
```

Apacheマイクロサービスを削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete -f apache.yaml
```

Kubernetesシークレットを削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete secret anthos-citrix
```

`nslogin` シークレットを削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete secret nslogin
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

デュアルティアイングレス展開

デュアルティアイングレス展開では、NetScaler VPXまたはMPXがKubernetesクラスターの外部（Tier-1）に展開され、NetScaler CPXがKubernetesクラスターの内部（Tier-2）に展開されます。

Tier-1のNetScaler MPXまたはVPXデバイスは、クライアントからTier-2のNetScaler CPXへのトラフィック（南北）をプロキシします。Tier-2のNetScaler CPXは、Kubernetesクラスター内のマイクロサービスにトラフィックをルーティングします。スタンドアロンポッドとして展開されたNetScaler Ingress Controllerは、Tier-1 NetScalerを構成します。1つ以上のNetScaler CPXポッド内のサイドカーNetScaler Ingress Controllerは、同じポッド内の関連するNetScaler CPXを構成します。

前提条件

- Anthos GKE on-premユーザークラスターと同じサブネットにTier-1 NetScaler VPXまたはMPXを展開する必要があります。
- Tier-1 NetScalerでサブネットIPアドレス（SNIP）を構成する必要があり、そのIPアドレスを使用してAnthos GKE on-premクラスターノードに到達できる必要があります。
- 異なるネットワークからNetScaler VPXまたはMPXを使用するには、NetScalerとAnthos GKE on-premクラスター間の通信を有効にするためにノードコントローラーを使用します。
- Tier-1 Ingressマニフェストでフロントエンドipアドレスとして使用される仮想IPアドレスを確保する必要があります。

Anthos GKE on-premでのデュアルティアイングレス展開

Anthos GKE on-premでNetScalerのデュアルティアイングレス展開をデプロイするには、次の手順を実行します。

1. 必要なアプリケーションをKubernetesクラスターにデプロイし、次のコマンドを使用してクラスター内でサービスとして公開します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig create -f https://
raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/
deployment/anthos/manifest/dual-tiered-ingress/apache.yaml
```

注:

この例では、`apache.yaml`が使用されています。アプリケーションに固有のYAMLファイルを使用する必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

2. cpx-cic.yamlファイルを使用して、NetScaler Ingress Controllerを搭載したNetScaler CPXをTier-2 Ingressとしてデプロイします。

```
kubectl --kubeconfig user-cluster-1-kubeconfig create -f https://
raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/
deployment/anthos/manifest/dual-tiered-ingress/cpx-cic.yaml
```

3. `cpx-service.yaml`ファイルを使用して、NetScaler CPXをKubernetesサービスとして公開します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig create -f https://
raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/
deployment/anthos/manifest/dual-tiered-ingress/cpx-service.yaml
```

4. (オプション) TLS設定のためにIngressで使用する自己署名SSL証明書とキーを作成します。

注:

すでにSSL証明書をお持ちの場合は、それを使用してKubernetesシークレットを作成できます。

```
openssl req -subj '/CN=anthos-citrix-ingress.com/O=Citrix Systems Inc/
C=IN' -new -newkey rsa:2048 -days 5794 -nodes -x509 -keyout $PWD/
anthos-citrix-certificate.key -out $PWD/anthos-citrix-
certificate.crt;openssl rsa -in $PWD/anthos-citrix-certificate.key -out
$PWD/anthos-citrix-certificate.key
```

注:

これは自己署名証明書を作成するための単なるコマンド例であり、このコマンドはアプリケーションのホスト名が`anthos-citrix-ingress.com`であることを前提としています。

5. 作成したSSL証明書とキーのペアを使用してKubernetesシークレットを作成します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl --kubeconfig user-cluster-1-kubeconfig create secret tls
anthos-citrix --cert=$PWD/anthos-citrix-certificate.crt --key=$PWD/
anthos-citrix-certificate.key
```

6. tier-2-ingress.yamlファイルを使用して、Tier-2用のIngressリソースを作成します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig create -f https://
raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/
deployment/anthos/manifest/dual-tiered-ingress/tier-2-ingress.yaml
```

7. Tier-1 NetScaler用のKubernetesシークレットを作成します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig create secret  generic
nslogin --from-literal=username='citrix-adc-username' --from-
literal=password='citrix-adc-password'
```

8. NetScaler Ingress ControllerをTier-1イングレスコントローラーとして展開します。

   a. cic.yamlファイルをダウンロードします。

   b. NetScalerの管理IPアドレスを入力します。`cic.yaml`ファイルで指定されているプレースホルダー`Tier-1-Citrix-ADC-IP`に、Tier-1 NetScalerの管理IPアドレスを更新します。

   c. 次のコマンドを使用して`cic.yaml`を保存し、展開します。

   ```
   kubectl --kubeconfig user-cluster-1-kubeconfig create -f cic.yaml
   ```

9. tier-1-ingress.yamlファイルを使用して、Tier-1用のIngressリソースを作成します。

   a. tier-1-ingress.yamlファイルをダウンロードします。

   b. YAMLファイルを編集し、`VIP-Citrix-ADC`を確保しておいたVIPアドレスに置き換えます。

   c. 次のコマンドを使用して、`tier-1-ingress.yaml`ファイルを保存してデプロイします。

   ```
   kubectl --kubeconfig user-cluster-1-kubeconfig create -f tier-1-
   ingress.yaml
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

10. マイクロサービスにアクセスするために、`VIP-Citrix-ADC`のIPアドレスでDNSレコードを更新します。この例では、Apacheマイクロサービスにアクセスするには、次のDNSエントリが必要です。

```
&lt;VIP-Citrix-ADC> anthos-citrix-ingress.com
```

11. 次のコマンドを使用してアプリケーションにアクセスします。

```
curl -k --resolve anthos-citrix-ingress.com:443:&lt;VIP-Citrix-ADC>
https://anthos-citrix-ingress.com/

&lt;html>&lt;body><h1>It works!</h1>&lt;/body>&lt;/html>
```

注:

このコマンドでは、`--resolve anthos-citrix-ingress.com:443:<VIP-for-accessing-microservices>`はDNS構成部分を上書きするために使用されます。

インストールのクリーンアップ: デュアルティアIngress

インストールをクリーンアップするには、`kubectl --kubeconfig delete`コマンドを使用して各デプロイを削除します。

Tier-1 Ingressオブジェクトを削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete -f tier-1-
ingress.yaml
```

Tier-2 Ingressオブジェクトを削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete -f tier-2-
ingress.yaml`
```

NetScaler CPXデプロイメントとサイドカーNetScaler Ingress Controllerを削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete -f cpx-cic.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler CPXサービスデプロイメントを削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete -f cpx-service.yaml
```

スタンドアロンのNetScaler Ingress Controllerを削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete -f cic.yaml
```

Apacheマイクロサービスを削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete -f apache.yaml
```

Kubernetesシークレットを削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete secret anthos-citrix
```

`nslogin`シークレットを削除するには、次のコマンドを使用します。

```
kubectl --kubeconfig user-cluster-1-kubeconfig delete secret nslogin`
```

Amazon ELBとNetScaler Ingress Controllerを使用して、EKS環境でNetScaler VPXをアクティブ-アクティブ高可用性で展開する

このトピックでは、AWS Elastic Container Service (EKS) プラットフォームの複数のアベイラビリティゾーンにNetScaler VPXをアクティブ-アクティブ高可用性モードで展開するソリューションについて説明します。このソリューションは、AWS Elastic Load Balancing (ELB) とNetScaler VPXを組み合わせて、EKSクラスターに展開されたマイクロサービスへのイングレストラフィックをロードバランシングします。AWS ELBはレイヤー4トラフィックを処理し、NetScaler VPXは高度なロードバランシング、キャッシング、コンテンツベースルーティングなどの高度なレイヤー7機能を提供します。

ソリューションの概要

EKSクラスターの基本的なアーキテクチャには、次の図に示すように、3つのアベイラビリティゾーンに展開された3つのパブリックサブネットと3つのプライベートサブネットが含まれます。

EKSアーキテクチャ(/en-us/netscaler-k8s-ingress-controller/media/eks-diagram.png)

このソリューションでは、EKSクラスターのアーキテクチャは次の図に示すようになります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ソリューションアーキテクチャ(/en-us/netscaler-k8s-ingress-controller/media/eks-vpx-architecture.png)

AWSクラウドでは、AWS Elastic Load Balancing がレイヤー4 TCP接続を処理し、フローハッシュルーティングアルゴリズムを使用してトラフィックをロードバランシングします。ELBは、ネットワークロードバランサーまたはクラシックロードバランサーのいずれかです。

AWS ELBは、リスナーによって定義された受信接続をリッスンします。各リスナーは、新しい接続を利用可能なNetScaler VPXインスタンスのいずれかに転送します。NetScaler VPXインスタンスは、EKSポッドへのトラフィックをロードバランシングします。また、NetScaler VPXによって提供されるリライトポリシー、レスポンダーポリシー、SSLオフロードなどの他のレイヤー7機能も実行します。

各NetScaler VPXインスタンスに対して、NetScaler Ingress ControllerがEKSクラスターに展開されます。NetScaler Ingress Controllerは同じイングレスクラスで構成されます。そして、EKSクラスター内のイングレスオブジェクトをそれぞれのNetScaler VPXインスタンスに構成します。

AWS Elastic Load Balancing (ELB) には、IPアドレスが動的に割り当てられるDNS名があります。このDNS名は、EKSクラスターでホストされているアプリケーションにアクセスするために、Route53 でドメインのエイリアスAレコードとして追加できます。

展開プロセス

ソリューションを展開するには、以下を実行します。

1. NetScaler VPXインスタンスを展開する。
2. NetScaler Ingress Controllerを展開する。
3. Amazon Elastic Load Balancing をセットアップします。 Network Load Balancer または Classic Load Balancer のいずれかをセットアップできます。
4. ソリューションを確認します。

NetScaler VPX インスタンスを展開する

NetScaler VPX は CloudFormation Template として利用できます。CloudFormation テンプレートは、指定されたサブネットに単一の ENI を持つ NetScaler VPX のインスタンスを展開します。また、NetScaler VPX インスタンスの NSIP、VIP、および SNIP も構成します。

このソリューションでは、NetScaler VPX のインスタンスを 2 つ展開する必要があります。同じ NetScaler VPX と異なるパブリックサブネットを指定して、2 つのアベイラビリティゾーンに NetScaler VPX インスタンスを展開します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler VPX インスタンスを展開した後、次のスクリーンショットに示すように CloudFormation テンプレートの出力を確認することで、展開を検証できます。出力には、NetScaler VPX インスタンス用に構成されたさまざまな IP アドレス (VIP、SNIP、NSIP) が表示されている必要があります。

注記:

CloudFormation テンプレートは、NetScaler VPX EC2 インスタンスのプライマリ IP アドレスを VIP として、セカンダリ IP アドレスを管理 IP アドレスとして NetScaler VPX インスタンスを展開します。

NetScaler VPX インスタンスが正常に展開されたら、セキュリティグループを編集して、EKS ノードグループのセキュリティグループからのトラフィックを許可する必要があります。また、VPX インスタンスからのトラフィックを許可するように EKS ノードグループのセキュリティグループを変更する必要があります。

NetScaler Ingress Controller を展開する

各 NetScaler VPX インスタンスに NetScaler Ingress Controller の個別のインスタンスを展開します。NetScaler Ingress Controller を展開するには、展開手順 に従ってください。

NetScaler VPX インスタンスが起動したら、NetScaler VPX インスタンスにシステムユーザーアカウントを設定する必要があります。システムユーザーアカウントは、NetScaler Ingress Controller が NetScaler VPX インスタンスにログインするために使用されます。システムユーザーアカウントの設定手順については、「NetScaler で CIC のシステムユーザーアカウントを作成する」を参照してください。

1. NetScaler Ingress Controller の展開 YAML (citrix-ingress-controller.yaml) を編集します。

   `NS_IP` を、対応する NetScaler VPX インスタンスの `Private NSIP` アドレスに置き換えます。また、NetScaler VPX インスタンスで作成したシステムユーザーアカウントのユーザー名とパスワードを指定します。`citrix-ingress-controller.yaml` ファイルを編集したら、次のコマンドを使用して更新された YAML ファイルを展開します。

   ```
   kubectl apply -f citrix-ingress-controller .yaml
   ```

2. 2番目のNetScaler Ingress Controllerインスタンスでステップ1を実行します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

3. 両方のポッドが起動して実行されていることを確認します。また、ログを使用してNetScaler Ingress ControllerがそれぞれのNetScaler VPXインスタンスに接続できることを確認します。

```
kubectl logs &lt;cic_pod_name>
```

NetScaler Ingress ControllerポッドがEKSクラスターにデプロイされ、実行された後、`citrix` イングレスクラスで構成されたKubernetes Ingressリソースは、両方のNetScaler VPXインスタンスに自動的に構成されます。

エラスティックロードバランシングのセットアップ

要件に応じて、次のいずれかのロードバランサーを構成できます。

- ネットワークロードバランサー
- クラシックロードバランサー

ネットワークロードバランサーのセットアップ

ネットワークロードバランサー (NLB) は、TCP接続のロードバランシングを処理するのに適したオプションです。このソリューションでは、NLBは受信トラフィックを受け入れ、いずれかのNetScaler VPXインスタンスにルーティングするために使用されます。NLBは、プロトコル、送信元IPアドレス、送信元ポート、宛先IPアドレス、宛先ポート、およびTCPシーケンス番号に基づいてフローハッシュアルゴリズムを使用してロードバランシングを行います。

NLBをセットアップするには：

1. AWS Management Console for EC2 にログオンします。
2. 左側のナビゲーションバーで、**ターゲットグループ**をクリックします。2つの異なるターゲットグループを作成します。1つはポート80でトラフィックをルーティングするためのターゲットグループ (***Target-Group-80***)、もう1つはポート443でトラフィックをルーティングするためのターゲットグループ (***Target-Group-443***) です。

3. *__Target-Group-80__ という名前のターゲットグループを作成します。以下を実行します。

   a. **ターゲットグループ名**フィールドに、ターゲットグループ名として ***Target-Group-80*** を入力します。

   b. 「**ターゲットタイプ**」フィールドで、「インスタンス」を選択します。

   c. 「**プロトコル**」リストから「**TCP**」を選択します。

   d. 「**ポート**」フィールドに「**80**」と入力します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

e. 「**VCP**」リストから、EKSクラスターをデプロイしたVPCを選択します。

f. 「**ヘルスチェック設定**」セクションで、ヘルスチェックにTCPを使用します。

g. オプション。「**詳細ヘルスチェック設定**」を変更して、ヘルスチェックを設定できます。

4. ***Target-Group-443** という名前のターゲットグループを作成します。以下を実行します。

a. 「**ターゲットグループ名**」フィールドに、ターゲットグループ名として***Target-Group-443***と入力します。

b. 「**ターゲットタイプ**」フィールドで、「インスタンス」を選択します。

c. 「**プロトコル**」リストから「**TCP**」を選択します。

d. 「**ポート**」フィールドに「**443**」と入力します。

e. 「**VCP**」リストから、EKSクラスターをデプロイしたVPCを選択します。

f. 「**ヘルスチェック設定**」セクションで、ヘルスチェックにTCPを使用します。

g. オプション。「**詳細ヘルスチェック設定**」を変更して、ヘルスチェックを設定できます。

5. ターゲットグループを作成したら、ターゲットインスタンスを登録する必要があります。

a. リストページで作成したターゲットグループを選択し、**ターゲット**タブをクリックして、**編集**を選択します。

b. **インスタンス**タブで、2つのNetScaler VPXインスタンスを選択し、**登録済みに追加**をクリックします。

6. 作成した他のターゲットグループについても、**手順5**を繰り返します。

7. ネットワークロードバランサーを作成します。

a. 左側のナビゲーションバーで、**ロードバランサー**を選択し、**ロードバランサーの作成**をクリックします。

b. ロードバランサータイプの選択ウィンドウで、ネットワークロードバランサーパネルの**作成**をクリックします。

8. **ロードバランサーの設定**ページで、次の操作を行います。

a. **名前**フィールドに、ロードバランサーの名前を入力します。

b. スキームフィールドで、**インターネット向け**を選択します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

c. リスナーセクションで、**リスナーの追加**をクリックし、ロードバランサープロトコルとして`TCP`、ロードバランサーポートとしてそれぞれ`80`と`443`を持つ2つのエントリを、次の図に示すように追加します。

d. アベイラビリティーゾーンセクションで、NetScaler VPXインスタンスが展開されているVPC、アベイラビリティーゾーン、およびサブネットを選択します。

「アベイラビリティゾーン」(/en-us/netscaler-k8s-ingress-controller/media/availability-zones-section.png)

9. 「**ルーティングの設定**」ページで、以下を実行します。

a. 「**ターゲットグループ**」リストで、「**既存のターゲットグループ**」をクリックします。

b. 「**名前**」フィールドに「**Target-Group-80**」と入力します。

c. 「**ターゲットタイプ**」フィールドで、「**インスタンス**」を選択します。

d. 「**プロトコル**」リストで、「**TCP**」を選択します。

e. 「**ポート**」フィールドに「**80**」と入力します。

f. 次の図に示すように、「**ヘルスチェック**」セクションの「**プロトコル**」リストから「**TCP**」を選択します。

「ルーティングの設定」(/en-us/netscaler-k8s-ingress-controller/media/configure-routing.png)

10. 「**確認**」ページで、設定を確認し、「**作成**」をクリックします。

「確認」(/en-us/netscaler-k8s-ingress-controller/media/review-nlb-settings.png)

11. ネットワークロードバランサーが作成されたら、リストページ用に作成したロードバランサーを選択します。「**リスナー**」タブを選択し、「**TCP : 444**」を選択してから「**編集**」をクリックします。

「リスナー」タブ(/en-us/netscaler-k8s-ingress-controller/media/listerners-tab.png)

12. 「**リスナー**」ページで、デフォルトのアクションを削除し、「**転送先**」リストで「**Target-Group-443**」を選択します。

「転送先」(/en-us/netscaler-k8s-ingress-controller/media/select-forward-to.png)

13. 「**更新**」をクリックします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

クラシックロードバランサーのセットアップ

Amazon Network Load Balancer の代替として、Tier 1 TCP ロードバランサーとして Classic Load Balancer (CLB) をセットアップできます。

1. AWS Management Console for EC2にログオンします。
2. 左側のナビゲーションバーで、**ロードバランサー**を選択し、次に「**ロードバランサーの作成**」をクリックします。

3. 「**ロードバランサータイプの選択**」ウィンドウで、Classic Load Balancer パネルの「**作成**」をクリックします。

4. 「**ロードバランサーの定義**」ページで、次の操作を行います。
   a. 「**ロードバランサー名**」フィールドに、ロードバランサーの名前を入力します。
   b. 「**LB の作成場所**」リストで、NetScaler VPX を選択します。
   c. 「**リスナー設定**」セクションで、「**追加**」をクリックし、ロードバランサープロトコルとして`TCP`、ロードバランサーポートとしてそれぞれ`80`と`443`の2つのエントリを追加します。また、インスタンスプロトコルとして`TCP`、インスタンスポートとしてそれぞれ`80`と`443`を選択します。次の図に示すとおりです。

   d. 「**サブネットの選択**」セクションで、Classic Load Balancer がトラフィックをルーティングするために、2つの異なるアベイラビリティーゾーンにある2つのパブリックサブネットを選択します。これらのサブネットは、NetScaler VPX インスタンスをデプロイした場所と同じです。

   e. **セキュリティグループの割り当て**ページで、ELBインスタンスのセキュリティグループを選択します。セキュリティグループは、NetScaler VPX ENIにアタッチされているセキュリティグループと同じでも、新しいセキュリティグループでも構いません。 新しいセキュリティグループを使用する場合は、ELBセキュリティグループからNetScaler VPXセキュリティグループへのトラフィック、およびその逆のトラフィックを許可していることを確認してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

f. **ヘルスチェックの設定**ページで、ヘルスチェックの設定を選択します。デフォルトでは、ヘルスチェックはポート `80` で `TCP` として設定されていますが、オプションでポート `443` でもヘルスチェックを実行できます。

g. **EC2インスタンスの追加**ページで、以前にデプロイした2つのNetScaler VPXインスタンスを選択します。

```
![Classic ADD EC2 Instances](https://docs.netscaler.com/content/dam/
netscaler/us/en/netscaler-k8s-ingress-controller/media/classic-add-
ec2.png)
```

h. タグの追加ページで、要件に応じてタグを追加します。

i. 確認ページで、設定を確認します。

j. **作成**をクリックします。

ソリューションの検証

NetScaler VPX、AWS ELB、およびNetScaler Ingress Controllerを正常にデプロイした後、サンプルサービスを使用してソリューションを検証できます。

次の操作を実行します。

1. app.yamlを使用してサンプルサービスとイングレスをデプロイします。

```
kubectl apply -f app.yaml
```

2. NetScaler VPXインスタンスにログオンし、両方のNetScaler VPXインスタンスでコンテンツスイッチングvserverが正常に構成されていることを確認します。次の操作を実行します。

   a. NetScaler VPXインスタンスにログオンします。次の操作を実行します。

      i. PuTTyなどのSSHクライアントを使用して、NetScaler VPXインスタンスへのSSH接続を開きます。

      ii. 管理者資格情報を使用してインスタンスにログオンします。

   b. 次のコマンドを使用して、コンテンツスイッチング (cs) vserver がインスタンスに構成されていることを確認します。

```
sh cs vserver
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

出力:

```
1) k8s-10.0.139.87:80:http (10.0.139.87:80) - HTTP Type: CONTENT

          State: UP

          Last state change was at Fri Apr 12 14:24:13 2019

          Time since last state change: 3 days, 03:09:18.920

          Client Idle Timeout: 180 sec

          Down state flush: ENABLED

          Disable Primary Vserver On Down : DISABLED

          Comment:
uid=NNJRYQ54VM2KWCXOERK6HRJHR4VEQYRI7U3W4BNFQLTIAENMTHWA====

          Appflow® logging: ENABLED

          Port Rewrite : DISABLED

          State Update: DISABLED

          Default:  Content Precedence: RULE

          Vserver IP and Port insertion: OFF

          L2Conn: OFF Case Sensitivity: ON

          Authentication: OFF

          401 Based Authentication: OFF

          Push: DISABLED Push VServer:

          Push Label Rule: none

          Listen Policy: NONE

          IcmpResponse: PASSIVE
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        RHIstate:  PASSIVE

        Traffic Domain: 0
```

c. ELB インスタンスの DNS 名を使用して、アプリケーション `test.example.com` にアクセスします。

```
# curl -H 'Host: test.example.com' &lt;DNS name of the ELB>
```

例:

```
% curl -H 'Host: test.example.com' http://VPX-HA-829787521.us-
west-2.elb.amazonaws.com
```

d. デプロイを削除するには、次のコマンドを使用します。

```
kubectl delete -f app.yaml
```

トラブルシューティング

問題	解決策
CloudFormation スタックの失敗	IAM ユーザーまたはロールに、EC2 インスタンスと Lambda 設定を作成するための十分な権限があることを確認します。
	リソースクォータを超過していないことを確認します。
NetScaler Ingress Controller が NetScaler VPX インスタンスと通信できません。	`citrix-ingress-controller.yaml` ファイルのユーザー名とパスワードが正しいことを確認します。
	NetScaler VPX セキュリティグループが、EKS ノードグループセキュリティグループからのポート `80` および `443` でのトラフィックを許可していることを確認します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

問題	解決策
NetScaler VPXインスタンスのサービスがDOWNしています。	NetScaler VPXのトラフィックがEKSクラスターに到達できることを確認してください。EKSノードグループのセキュリティグループを変更し、NetScaler VPXセキュリティグループからのトラフィックを許可してください。
ELBからNetScaler VPXインスタンスへのトラフィックがルーティングされていません。	NetScaler VPXのセキュリティグループがELBセキュリティグループからのトラフィックを許可していることを確認してください。

AWSでLoadBalancerタイプのサービス向けCitrix®ソリューションを展開する

LoadBalancerタイプのサービスは、Kubernetesクラスターで実行されているマイクロサービスを外部に公開するための、よりシンプルで高速な方法です。クラウドデプロイメントでは、LoadBalancerタイプのサービスを作成すると、クラウドマネージドのロードバランサーがそのサービスに割り当てられます。その後、そのロードバランサーを使用してサービスが公開されます。LoadBalancerタイプのサービスに関する詳細については、LoadBalancerタイプのサービスを参照してください。

LoadBalancerタイプのサービス向けCitrixソリューションを使用すると、クラウドマネージドのロードバランサーの代わりにNetScalerを使用して、サービスを直接ロードバランスし、公開できます。NetScalerは、オンプレミスおよびクラウド向けのLoadBalancerタイプのサービスにこのソリューションを提供します。LoadBalancerタイプのサービスは、AWS、GCP、Azureなどのパブリッククラウド上のKubernetesデプロイメントでネイティブにサポートされています。

AWSでサービスをデプロイすると、ロードバランサーが自動的に作成され、IPアドレスがサービスの外部フィールドに割り当てられます。このCitrixソリューションでは、NetScaler IPAM ControllerがIPアドレスを割り当て、そのIPアドレスがNetScaler VPXのVIPとなります。Kubernetesクラスターにデプロイされ NetScaler Ingress Controllerは、クラスター外にデプロイされたNetScalerを設定して、受信トラフィックをロードバランスします。したがって、サービスはクラウドロードバランサーではなく、NetScaler VPXを介してアクセスされます。

サービス定義で、サービス`type`を`LoadBalancer`として指定する必要があります。`type`フィールドを`LoadBalancer`に設定すると、AWSでサービス用のロードバランサーがプロビジョニングされます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler IPAM Controllerは、指定されたIPアドレス範囲からLoadBalancerタイプのサービスにIPアドレスを自動的に割り当てるために使用されます。LoadBalancerタイプのサービス向けCitrixソリューションの詳細については、LoadBalancerタイプのサービスを公開するを参照してください。

HelmチャートまたはYAMLファイルを使用して、AWSでLoadBalancerタイプのサービス向けCitrixソリューションを展開できます。

前提条件

- Elastic Kubernetes Service (EKS) クラスターのバージョン1.18以降が稼働していることを確認してください。
- NetScaler VPXとEKSが同じVPCにデプロイされ、稼働していることを確認してください。AWSでNetScaler VPXを作成する方法については、AWS MarketplaceからNetScaler VPXインスタンスを作成するを参照してください。

HelmチャートをAWSでLoadBalancerタイプのサービス向けCitrixソリューションを展開する

Helmチャートを使用してLoadBalancerタイプのサービス向けCitrixソリューションを設定するには、以下の手順を実行します。

1. unified-lb-values.yamlファイルをダウンロードし、以下の詳細を指定するためにYAMLファイルを編集します。
    - NetScaler VPX NSIP。詳細については、NetScaler Ingress Controller Helmチャートを参照してください。
    - NetScaler VPXの資格情報を使用して作成されたシークレット。詳細については、NetScaler Ingress Controller Helmチャートを参照してください。
    - IPAMコントローラーで使用するVIPのリスト。詳細については、IPAM Helmチャートを参照してください。
2. 編集したYAMLファイルを使用して、NetScaler IPAM ControllerとNetScaler Ingress ControllerをAmazon EKSクラスターにデプロイします。以下のコマンドを使用します。

```
helm repo add citrix https://citrix.github.io/citrix-helm-charts/



helm install serviceLB citrix/citrix-cloud-native -f values.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

3. Amazon EKSにアプリケーションとサービスをデプロイします。

   a. サービスマニフェストに以下の注釈を追加します。

   ```
   beta.kubernetes.io/aws-load-balancer-type: "external"
   ```

   b. 変更された注釈を使用して、以下のコマンドでアプリケーションとサービスをデプロイします。

   ```
   kubectl create -f https://github.com/citrix/citrix-k8s-ingress-controller/blob/master/docs/how-to/typeLB/aws/guestbook-all-in-one-lb.yaml
   ```

   注: `guestbook` マイクロサービスは、この手順で使用されるサンプルです。任意のアプリケーションをデプロイできます。サービスがLoadBalancerタイプであり、サービスマニフェストに注釈が含まれていることを確認してください。

   c. NetScaler VPXのVIPにElastic IPアドレスを関連付けます。

   d. ブラウザを使用してアプリケーションにアクセスします。例: `http://EIP-associated-with-vip`。

YAMLを使用してAWSでLoadBalancerタイプのサービス向けCitrixソリューションをデプロイする

YAMLを使用してLoadBalancerタイプのサービス向けCitrixソリューションをデプロイするには、以下の手順を実行します。

1. citrix-k8s-ingress-controller.yamlファイルをダウンロードし、以下の詳細を指定します。

   - NetScaler VPX NSIP
   - NetScaler VPXの資格情報を使用して作成されたシークレット。シークレットの作成については、シークレットの作成を参照してください。
   - NetScaler IPAM Controllerの引数を指定します。

   ```
   args:

     - --ipam

       citrix-ipam-controller
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

2. 変更されたYAMLを使用してNetScaler Ingress Controllerをデプロイします。

```
kubectl create -f citrix-k8s-ingress-controller.yaml
```

3. NetScaler VIP CRD をデプロイします。これにより、NetScaler Ingress Controller と IPAM コントローラー間の通信が以下のコマンドを使用して有効になります。

```
kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/crd/vip/vip.yaml
```

NetScaler VIP CRD のデプロイの詳細については、VIP CRD のデプロイを参照してください。

4. IPAM コントローラーをデプロイします。IPAM コントローラーのデプロイの詳細については、IPAM コントローラーのデプロイを参照してください。

注:

IPAM デプロイ YAML ファイルの `VIP_RANGE` フィールドに NetScaler VPX VIP のリストを指定します。

5. 以下の手順で、Amazon EKS にサービスタイプ LoadBalancer のアプリケーションをデプロイします。

   a. サービスマニフェストに以下の注釈を追加します。

```
beta.kubernetes.io/aws-load-balancer-type: "external"
```

   b. 以下のコマンドを使用して、変更された注釈でアプリケーションとサービスをデプロイします。

```
kubectl create -f https://github.com/citrix/citrix-k8s-ingress-controller/blob/master/docs/how-to/typeLB/aws/guestbook-all-in-one-lb.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

> **注:**
>
> `guestbook` マイクロサービスは、この手順で使用されるサンプルです。任意のアプリケーションをデプロイできます。サービスが LoadBalancer タイプであり、サービスマニフェストに注釈が含まれていることを確認してください。

c. 弾力性のある IP アドレスを NetScaler VPX の VIP に関連付けます。

d. ブラウザを使用してアプリケーションにアクセスします。例: `http://EIP-associated-with-vip`。

Amazon EKSおよびMicrosoft AKSクラスターを使用したマルチクラウドおよびGSLBソリューション

異なるクラウドプロバイダーが提供する複数のクラウドに、同じアプリケーションの複数のインスタンスを展開できます。このマルチクラウド戦略は、回復性、高可用性、および近接性を確保するのに役立ちます。マルチクラウドのアプローチにより、ベンダーロックインやクラウド障害などのリスクを軽減することで、各クラウドプロバイダーの最高の利点を活用することもできます。

NetScalerは、NetScaler Ingress Controllerの助けを借りて、マルチクラウドロードバランシングを実行できます。NetScalerは、異なるクラウドプロバイダーサイトでホストされているクラスターにトラフィックを誘導できます。このソリューションは、Amazon EKS (Elastic Kubernetes Service) およびMicrosoft AKS (Azure Kubernetes Service) クラスターで実行されているワークロード間でトラフィックをインテリジェントに分散することで、ロードバランシングを実行します。

Amazon EKSとMicrosoft AKSを使用して、マルチクラウドおよびGSLBソリューションを展開できます。

展開トポロジ

次の図は、Amazon EKSおよびMicrosoft AKSが提供するKubernetesサービス向けのマルチクラウドイングレスおよびロードバランシングソリューションの展開トポロジを示しています。

展開トポロジ(/en-us/netscaler-k8s-ingress-controller/media/multi-cloud-ingress-architecture.png)

前提条件

- AWSとAzureに精通している必要があります。
- NetScalerとNetScalerネットワーキングに精通している必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- 同じアプリケーションのインスタンスは、Amazon EKSおよびMicrosoft AKS上のKubernetesクラスターに展開されている必要があります。

マルチクラウドおよびGSLBソリューションを展開するには、次のタスクを実行する必要があります。

1. AWSにNetScaler VPXを展開します。
2. AzureにNetScaler VPXを展開します。
3. AWSおよびAKSに展開されたNetScaler VPXでADNSサービスを構成します。
4. AWSおよびAKSにデプロイされたNetScaler VPXでGSLBサービスを設定します。
5. AWSおよびAzure KubernetesクラスターにGTPおよびGSE CRDを適用します。
6. GSLBコントローラーをデプロイします。

AWSへのNetScaler VPXのデプロイ

NetScaler VPXインスタンスがEKSクラスターと同じ仮想プライベートクラウド (VPC) にインストールされていることを確認する必要があります。これにより、NetScaler VPXはEKSワークロードと通信できます。既存のEKSサブネットを使用するか、サブネットを作成してNetScaler VPXインスタンスをインストールできます。

また、NetScaler VPXインスタンスを別のVPCにインストールすることもできます。その場合、EKS用のVPCがVPCピアリングを使用して通信できることを確認する必要があります。VPCピアリングの詳細については、VPCピアリングのドキュメントを参照してください。

高可用性 (HA) のために、NetScaler VPXの2つのインスタンスをHAモードでインストールできます。

1. AWSにNetScaler VPXをインストールします。AWSへのNetScaler VPXのインストールについては、AWSへのNetScaler VPXインスタンスのデプロイを参照してください。

   NetScaler VPXは、GSLBサービス同期およびADNSサービスを実行するために、NSIP以外のセカンダリパブリックIPアドレスを必要とします。

2. AWSコンソールを開き、**EC2** > **ネットワークインターフェース** > **VPXプライマリENI ID** > **IPアドレスの管理**を選択します。**新しいIPアドレスの割り当て**をクリックします。

   セカンダリパブリックIPアドレスがVPX ENIに割り当てられたら、それにElastic IPアドレスを関連付けます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

3. **EC2** > **ネットワークインターフェース** > **VPX ENI ID** - **アクション**を選択し、**IPアドレスの関連付け**をクリックします。セカンダリIPアドレスのElastic IPアドレスを選択し、**関連付け**をクリックします。

4. NetScaler VPXインスタンスにログインし、セカンダリIPアドレスを`SNIP`として追加し、次のコマンドを使用して管理アクセスを有効にします。

   ```
   add ip 192.168.211.73 255.255.224.0 -mgmtAccess ENABLED -type SNIP
   ```

   注記:

   - NetScaler VPXにSSHを使用してログインするには、セキュリティグループでSSHポートを有効にする必要があります。ルートテーブルには、デフォルトトラフィック用にインターネットゲートウェイが構成されている必要があり、NACLはSSHポートを許可する必要があります。
   - NetScaler VPXを高可用性 (HA) モードで実行している場合は、両方のNetScaler VPXインスタンスでこの構成を実行する必要があります。

5. 次のコマンドを使用して、NetScaler VPXでコンテンツスイッチング (CS)、ロードバランシング (LB)、グローバルサーバーロードバランシング (GSLB)、およびSSL機能を有効にします。

   ```
   enable feature *feature*
   ```

   注記:

   GSLBを有効にするには、追加のライセンスが必要です。

6. NetScaler VPXがDNSトラフィックを受信できるように、VPXセキュリティグループでUDPおよびTCPのポート53を有効にします。また、SSH用にTCPポート22を、GSLBメトリック交換用にTCPポート範囲3008～3011を有効にします。

   セキュリティグループにルールを追加する方法については、「セキュリティグループへのルールの追加」を参照してください。

7. 次のコマンドを使用して、NetScaler VPXにネームサーバーを追加します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
add nameserver *nameserver IP*
```

AzureでのNetScaler VPXの展開

AKSクラスターでスタンドアロンのNetScaler VPXインスタンスを実行することも、AKSクラスターで2つのNetScaler VPXインスタンスを高可用性モードで実行することもできます。

インストール中に、AKSクラスターがVPXインスタンスとの接続性を持っていることを確認してください。接続性を確保するために、AKSクラスターと同じ仮想ネットワーク (VNet) に、別のリソースグループでNetScaler VPXをインストールできます。

NetScaler VPXのインストール中に、AKSクラスターがインストールされているVNetを選択します。あるいは、VPXがAKSクラスターとは異なるVNetに展開されている場合は、VNetピアリングを使用してAKSとNetScaler VPX間の接続性を確保できます。

1. AWSにNetScaler VPXをインストールします。AKSにNetScaler VPXをインストールする方法については、「Microsoft AzureにNetScaler VPXインスタンスを展開する」を参照してください。

   GSLB同期およびADNSサービス用に、パブリックIPを持つSNIPが必要です。SNIPがすでに存在する場合は、それにパブリックIPアドレスを関連付けます。

2. 関連付けるには、**ホーム** > **リソース グループ** > **VPX インスタンス** > **VPX NIC インスタンス**を選択します。次の図に示すように、パブリック IP アドレスを関連付けます。**保存**をクリックして変更を保存します。

3. Azure NetScaler VPX インスタンスにログインし、次のコマンドを使用して、管理アクセスを有効にしたセカンダリ IP を SNIP として追加します。

   ```
   add ip 10.240.0.11 255.255.0.0 -type SNIP -mgmtAccess ENABLED
   ```

   リソースが存在する場合は、次のコマンドを使用して、既存のリソースで管理アクセスを有効に設定できます。

   ```
   set ip 10.240.0.11 -mgmtAccess ENABLED
   ```

4. 次のコマンドを使用して、NetScaler VPX で CS、LB、SSL、および GSLB 機能を有効にします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
enable feature *feature*
```

SSH を介して NetScaler VPX インスタンスにアクセスするには、NetScaler VPX プライマリインターフェイスにアタッチされている Azure ネットワーク セキュリティ グループで、SSH ポートのインバウンド ポート ルールを有効にする必要があります。

5. Azure ポータルのネットワーク セキュリティ グループで、次のポートのインバウンド ルールを有効にします。
   - TCP: GSLB メトリック交換用の 3008～3011
   - TCP: SSH 用の 22
   - TCP および UDP: DNS 用の 53
6. 次のコマンドを使用して、NetScaler VPX にネームサーバーを追加します。

```
add nameserver *nameserver IP*
```

AWS および Azure に展開された NetScaler VPX で ADNS サービスを構成する

NetScaler VPX の ADNS サービスは、ドメインの権威 DNS として機能します。ADNS サービスの詳細については、「権威 DNS サービス」を参照してください。

1. AWS NetScaler VPX にログインし、次のコマンドを使用して、セカンダリ IP アドレスとポート 53 で ADNS サービスを構成します。

```
add service Service-ADNS-1 192.168.211.73 ADNS 53
```

次のコマンドを使用して構成を確認します。

```
show service Service-ADNS-1
```

2. Azure NetScaler VPXにログインし、以下のコマンドを使用してセカンダリIPアドレスとポート53でADNSサービスを設定します。

```
add service Service-ADNS-1 10.240.0.8 ADNS 53
```

以下のコマンドを使用して設定を確認します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
show service Service-ADNS-1
```

3. ドメイン用に2つのADNSサービスを作成した後、ドメインのNSレコードを更新して、ドメインレジストラ内のADNSサービスを指すようにします。

   例えば、ADNSサービスのパブリックIPアドレスを指す「A」レコード `ns1.domain.com` を作成します。ドメインのNSレコードはns1.domain.comを指す必要があります。

AWSおよびAzureにデプロイされたNetScaler VPXでGSLBサービスを設定する

AWSおよびAzureにデプロイされたNetScaler VPX上にGSLBサイトを作成する必要があります。

1. AWS NetScaler VPXにログインし、以下のコマンドを使用してセカンダリIPアドレスにGSLBサイトを設定します。また、*–publicIP* 引数を使用してパブリックIPアドレスを指定します。例：

```
add gslb site aws_site 192.168.197.18 -publicIP 3.139.156.175



add gslb site azure_site 10.240.0.11 -publicIP 23.100.28.121
```

2. Azure NetScaler VPXにログインし、GSLBサイトを設定します。例：

```
add gslb site aws_site 192.168.197.18 -publicIP 3.139.156.175



add gslb site azure_site 10.240.0.11 -publicIP 23.100.28.121
```

3. 以下のコマンドを使用して、いずれかのサイトから同期を開始し、GSLB同期が成功したことを確認します。

```
sync gslb config –debug
```

注:

初期同期が失敗した場合は、AWSとAzureの両方でセキュリティグループを確認し、必要なポートを許可してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

AWSおよびAzure KubernetesクラスターにGTPおよびGSE CRDを適用する

グローバル・トラフィック・ポリシー (GTP) およびグローバル・サービス・エントリ (GSE) CRDは、KubernetesアプリケーションでGSLBを実行するためのNetScalerの設定に役立ちます。これらのCRDは、分散Kubernetesクラスターにデプロイされたアプリケーション用のNetScaler GSLBコントローラーを設定するために設計されています。

GTP CRD

GTP CRDは、デプロイタイプ (カナリア、フェイルオーバー、ローカルファースト)、GSLBドメイン、イングレスのヘルスモニター、サービスタイプなど、NetScalerでGSLBを設定するためのパラメータを受け入れます。

GTP CRDの定義については、GTP CRDを参照してください。次のコマンドを使用して、AWSおよびAzure KubernetesクラスターにGTP CRD定義を適用します。

```
kubectl apply -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/gslb/Manifest/gtp-crd.yaml
```

GSE CRD

GSE CRDは、各クラスターのエンドポイント情報（クラスターにトラフィックをルーティングするKubernetesオブジェクトに関する情報）を指定します。グローバルサービスエントリは、クラスターにトラフィックをルーティングするアプリケーションの外部IPアドレスを自動的に選択します。ルートの外部IPアドレスが変更された場合、グローバルサービスエントリは新しく割り当てられたIPアドレスを選択し、それに応じてNetScalerのGSLBエンドポイントを構成します。

GSE CRDの定義については、GSE CRDを参照してください。次のコマンドを使用して、AWSおよびAzure KubernetesクラスターにGSE CRD定義を適用します。

```
kubectl apply -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/gslb/Manifest/gse-crd.yaml
```

GSLBコントローラーの展開

GSLBコントローラーは、マルチクラウド環境におけるクラスター間のアプリケーションの高可用性を確保するのに役立ちます。

GSLBコントローラーは、AWSおよびAzureクラスターにインストールできます。GSLBコントローラーは、GTPおよびGSE CRDをリッスンし、マルチクラウド環境の複数のリージョンで高可用性を提供するGSLB用にNetScalerを構成します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

GSLBコントローラーを展開するには、次の手順を実行します。

1. AWSおよびAzure KubernetesクラスターにGSLBコントローラー用のRBACを作成します。

```
kubectl apply -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/gslb/Manifest/gslb-rbac.yaml
```

2. 次のコマンドを使用して、AWSおよびAzureクラスターにシークレットを作成します。

注:

シークレットにより、GSLBコントローラーはGSLBデバイスに接続し、構成をプッシュできます。

```
kubectl create secret generic secret-1 --from-literal=username=&lt;username> --from-literal=password=&lt;password>
```

注:

`add system user`コマンドを使用して、NetScalerにユーザーを追加できます。

3. gslb-controller.yamlからGSLBコントローラーのYAMLファイルをダウンロードします。
4. 次のコマンドを使用して、AWS クラスターに `gslb-controller.yaml` を適用します。

```
kubectl apply -f  gslb-controller.yaml
```

AWS 環境では、LOCAL_REGION、LOCAL_CLUSTER、および SITENAMES 環境変数を定義するために `gslb-controller.yaml` を編集します。

次の例では、環境変数 `LOCAL_REGION` を *us-east-2*、`LOCAL_CLUSTER` を *eks-cluster*、環境変数 `SITENAMES` を *aws_site,azure_site* として定義しています。

```
name: "LOCAL_REGION"

value: "us-east-2"

name: "LOCAL_CLUSTER"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
value: "eks-cluster"

name: "SITENAMES"

value: "aws_site,azure_site"

name: "aws_site_ip"

value: "NSIP of aws VPX(internal IP)"

name: "aws_site_region"

value: "us-east-2"

name: "azure_site_ip"

value: "NSIP of azure_VPX(public IP)"

name: "azure_site_region"

value: "central-india"

name: "azure_site_username"

valueFrom:

  secretKeyRef:

   name: secret-1

   key: username

name: "azure_site_password"

valueFrom:

  secretKeyRef:

   name: secret-1

   key: password

name: "aws_site_username"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
valueFrom:

  secretKeyRef:

   name: secret-1

   key: username

name: "aws_site_password"

valueFrom:

  secretKeyRef:

   name: secret-1

   key: password
```

次のコマンドを使用して、Azure クラスターに gslb-controller.yaml を適用します。

```
kubectl apply -f  gslb-controller.yaml
```

5. Azure サイトでは、`LOCAL_REGION`、`LOCAL_CLUSTER`、および `SITENAMES` 環境変数を定義するために `gslb-controller.yaml` を編集します。

   次の例では、環境変数 `LOCAL_REGION` を *central-india*、`LOCAL_CLUSTER` を *azure-cluster*、`SITENAMES` を *aws_site, azure_site* として定義しています。

```
name: "LOCAL_REGION"

value: "central-india"

name: "LOCAL_CLUSTER"

value: "aks-cluster"

name: "SITENAMES"

value: "aws_site,azure_site"

name: "aws_site_ip"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
value: "NSIP of AWS VPX(public IP)"

name: "aws_site_region"

value: "us-east-2"

name: "azure_site_ip"

value: "NSIP of azure VPX(internal IP)"

name: "azure_site_region"

value: "central-india"

name: "azure_site_username"

valueFrom:

  secretKeyRef:

   name: secret-1

   key: username

name: "azure_site_password"

valueFrom:

  secretKeyRef:

   name: secret-1

   key: password

name: "aws_site_username"

valueFrom:

  secretKeyRef:

   name: secret-1

   key: username
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
name: "aws_site_password"

valueFrom:

  secretKeyRef:

   name: secret-1

   key: password
```

注記：

GSLB サイト情報の順序は、すべてのクラスターで同じである必要があります。順序の最初のサイトは、構成をプッシュするためのマスターサイトと見なされます。マスターサイトがダウンすると、リストの次のサイトが新しいマスターになります。したがって、サイトの順序はすべての Kubernetes クラスターで同じである必要があります。

サンプルアプリケーションの展開

このサンプルアプリケーションの展開シナリオでは、apache の `https` イメージが使用されます。ただし、任意のサンプルアプリケーションを選択できます。

アプリケーションは、AWS および Azure クラスターの両方で LoadBalancer タイプとして公開されます。AWS および Azure Kubernetes クラスターの両方でコマンドを実行する必要があります。

1. 次のコマンドを使用して、サンプル apache アプリケーションのデプロイを作成します。

   ```
   kubectl create deploy apache --image=httpd:latest port=80
   ```

2. 次のコマンドを使用して、apache アプリケーションを LoadBalancer タイプのサービスとして公開します。

   ```
   kubectl expose deploy apache --type=LoadBalancer --port=80
   ```

3. 次のコマンドを使用して、LoadBalancer タイプのサービスに外部 IP アドレスが割り当てられていることを確認します。

   ```
   kubectl get svc apache

   NAME     TYPE           CLUSTER-IP    EXTERNAL-IP     PORT(S)
    AGE
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apache   LoadBalancer   10.0.16.231   20.62.235.193   80:32666/TCP
3m2s
```

AWS および Azure クラスターにアプリケーションをデプロイした後、マルチクラウドクラスターで高可用性を構成するために GTE カスタムリソースを構成する必要があります。

次の例に示すように、GTP YAML リソース `gtp_isntance.yaml` を作成します。

```
apiVersion: "citrix.com/v1beta1"

 kind: globaltrafficpolicy

 metadata:

   name: gtp-sample-app

   namespace: default

 spec:

   serviceType: 'HTTP'

   hosts:

   - host: <domain name>

     policy:

       trafficPolicy: 'FAILOVER'

       secLbMethod: 'ROUNDROBIN'

       targets:

       - destination: 'apache.default.us-east-2.eks-cluster'

         weight: 1

       - destination: 'apache.default.central-india.aks-cluster'

         primary: false
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        weight: 1
      monitor:
      - monType: http
        uri: ''
        respCode: 200
  status:
    {}
```

この例では、トラフィックポリシーは `FAILOVER` として設定されています。ただし、マルチクラスターコントローラーは複数のトラフィックポリシーをサポートしています。詳細については、トラフィックポリシーのドキュメントを参照してください。

次のコマンドを使用して、両方のクラスターにGTPリソースを適用します。

```
kubectl apply -f gtp_instance.yaml
```

サービスステータスから導出された必要なエンドポイント情報とともに、GSEリソースが両方のクラスターに自動的に作成されていることを確認できます。次のコマンドを使用して確認してください。

```
kubectl get gse

kubectl get gse *name* -o yaml
```

また、NetScaler VPXにログインし、次のコマンドを使用してGSLB構成が正常に作成されていることを確認してください。

```
show gslb runningconfig
```

GTP CRDがトラフィックポリシーとして `FAILOVER` に設定されているため、NetScaler VPXインスタンスはプライマリクラスター（この例ではEKSクラスター）からトラフィックを処理します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
curl -v http://*domain_name*
```

ただし、EKSクラスターでエンドポイントが利用できない場合、アプリケーションは自動的にAzureクラスターから提供されます。プライマリクラスターでレプリカ数を `0` に設定することで、これを確実にすることができます。

Amazon EKSおよびMicrosoft AKSクラスターのイングレスおよびGSLBデバイスとしてのNetScaler VPX

Amazon EKSおよびMicrosoft AKSを使用し、NetScaler VPXをGSLBとして、また同じNetScaler VPXをイングレスデバイスとしても使用して、マルチクラウドおよびマルチクラスターのイングレスおよびロードバランシングソリューションを展開できます。

NetScaler VPXをイングレスデバイスとして使用してマルチクラウドマルチクラスターのイングレスおよびロードバランシングを展開するには、前のセクションで説明されている以下のタスクを完了する必要があります。

1. AWSにNetScaler VPXを展開する
2. AzureにNetScaler VPXを展開する
3. AWSおよびAKSに展開されたNetScaler VPXでADNSサービスを構成する
4. AWSおよびAKSに展開されたNetScaler VPXでGSLBサービスを構成する
5. AWSおよびAzure KubernetesクラスターにGTPおよびGSE CRDを適用する
6. GSLBコントローラーを展開する

先行するタスクを完了した後、以下のタスクを実行します。

1. NetScaler VPXをAWSのイングレスデバイスとして構成する
2. NetScaler VPXをAzureのイングレスデバイスとして構成する

NetScaler VPXをAWSのイングレスデバイスとして構成する

以下の手順を実行します。

1. Kubernetesシークレットを使用してNetScaler VPXのログイン資格情報を作成します

   ```
   kubectl create secret  generic nslogin --from-literal=username='nsroot'
   --from-literal=password='&lt;instance-id-of-vpx>'
   ```

   NetScaler VPXのパスワードは、変更していない限り、通常はVPXのインスタンスIDです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

2. SSHを使用してNetScaler VPXに接続し、NetScaler VPXでSNIPを構成します。SNIPは、Citrix VPXのセカンダリIPアドレスであり、エラスティックIPアドレスが割り当てられていません。

```
add ns ip 192.168.84.93 255.255.224.0
```

この手順は、NetScalerがKubernetesクラスター内のポッドと対話するために必要です。

3. NetScaler Ingress Controllerマニフェストで、NetScaler VPXの管理IPアドレスとVIPを更新します。

```
wget https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-
controller/master/deployment/aws/quick-deploy-cic/manifest/cic.yaml
```

注:

`wget`がインストールされていない場合は、`fetch`または`curl`を使用できます。

4. 以下のフィールドで、`cic.yaml`内のNetScaler VPXのプライマリIPアドレスを更新します。

```
# Set NetScaler NSIP/SNIP, SNIP in case of HA (mgmt has to be enabled)

 - name: "NS_IP"

   value: "X.X.X.X"
```

5. 以下のフィールドで、`cic.yaml`内のNetScaler VPX VIPを更新します。これは、エラスティックIPアドレスを割り当てたプライベートIPアドレスです。

```
# Set NetScaler® VIP for the data traffic

- name: "NS_VIP"

  value: "X.X.X.X"
```

6. 必要な値でYAMLファイルを編集したら、NetScaler Ingress Controllerを展開します。

```
kubectl create -f cic.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler VPX を Azure 用のイングレスデバイスとして構成する

次の手順を実行します。

1. Kubernetes シークレットを使用して NetScaler VPX のログイン資格情報を作成します。

```
kubectl create secret  generic nslogin --from-
literal=username='&lt;azure-vpx-instance-username>' --from-
literal=password='&lt;azure-vpx-instance-password>'
```

注記：

NetScaler VPX のユーザー名とパスワードは、Azure で NetScaler VPX を作成する際に設定した資格情報と同じである必要があります。

2. SSH を使用して、NetScaler VPX に SNIP (NetScaler VPX のセカンダリ IP アドレス) を構成します。この手順は、NetScaler が Kubernetes クラスター内のポッドと通信するために必要です。

```
add ns ip &lt;snip-vpx-instance-private-ip> &lt;vpx-instance-primary-
ip-subnet>
```

- `snip-vpx-instance-private-ip` は、NetScaler VPX インスタンスの作成中に SNIP を追加する際に割り当てられる動的プライベート IP アドレスです。
- `vpx-instance-primary-ip-subnet` は、NetScaler VPX インスタンスのプライマリプライベート IP アドレスのサブネットです。

プライベート IP アドレスのサブネットを確認するには、NetScaler VPX インスタンスに SSH で接続し、次のコマンドを使用します。

```
show ip &lt;primary-private-ip-addess>
```

3. NetScaler Ingress Controller YAML ファイルで、NetScaler VPX のイメージ URL、管理 IP アドレス、および VIP を更新します。

   a. NetScaler Ingress Controller YAML ファイルをダウンロードします。

```
wget https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-
controller/master/deployment/azure/manifest/azurecic/cic.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注記：

`wget` がインストールされていない場合は、`fetch` または `curl` コマンドを使用できます。

b. `cic.yaml` ファイルで、NetScaler Ingress Controller イメージを Azure イメージ URL で更新します。

```
- name: cic-k8s-ingress-controller

  # CIC Image from Azure

  image: "&lt;azure-cic-image-url>"
```

c. `cic.yaml` で、NetScaler VPX のプライマリ IP アドレスを Azure VPX インスタンスのプライマリプライベート IP アドレスで更新します。

```
# Set NetScaler NSIP/SNIP, SNIP in case of HA (mgmt has to be
enabled)

- name: "NS_IP"

  value: "X.X.X.X"
```

d. `cic.yaml`で、NetScaler VPX VIPを、VPX Azureインスタンス作成時に割り当てられたVIPのプライベートIPアドレスで更新します。

```
# Set NetScaler VIP for the data traffic

- name: "NS_VIP"

  value: "X.X.X.X"
```

4. 必要な値でNetScaler Ingress Controllerを設定したら、次のコマンドを使用してNetScaler Ingress Controllerを展開します。

```
kubectl create -f cic.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

VMware クラウドファンデーション上の vSphere Kubernetes サービスにおけるネットスケーラー

NetScaler は、NetScaler Ingress Controller および NetScaler Kubernetes Gateway Controller を介して、VMware Cloud Foundation (VCF) 上の VMware vSphere Kubernetes Service (VKS) と統合します。両方のコントローラーは Kubernetes リソースを監視し、NetScaler 上での手動設定なしに、南北アプリケーション トラフィックを処理するように NetScaler VPX を自動的にプログラムします。

プラットフォームチームは、ロードバランシング、TLSオフロード、WAF、レート制限、および書き換えポリシーをすべてKubernetesマニフェストを通じて管理できます。このドキュメントでは、共同ソリューションの検証済み展開アーキテクチャについて説明します。

vSphere Kubernetes Service の概要

vSphere Kubernetes Service (VKS) は、VMware Cloud Foundation (VCF) に組み込まれている Kubernetes ランタイムです。これは、プラットフォームエンジニアがVCFクラウドサービスの全セットと並行して展開および管理できる、CNCF認定のKubernetesディストリビューションを提供します。クラウド管理者は、N-2 Kubernetes バージョンサポート、自動アップグレード、およびコンピューティング、ストレージ、ネットワークにわたる単一のライフサイクル管理プレーンを利用できます。

TCO の削減。 VKS は、既存の vSphere ツール、スキル、および運用プロセスを再利用することで、インフラストラクチャのサイロを削減します。統合されたライフサイクル管理により、最小限の手動作業でクラスターを最新の状態に保ちます。

運用の簡素化。 自動化されたクラスタープロビジョニング、アップグレード、およびライフサイクル管理により、運用後のオーバーヘッドが削減され、チームはインフラストラクチャの維持ではなくアプリケーションに集中できます。

大規模な Kubernetes。 VKS は、自己管理型 Kubernetes の運用上の負担なしに、初期プロビジョニングから継続的なアップグレードまで、本番環境で複数のクラスターを実行する完全なライフサイクルを処理します。

Kubernetes 向け NetScaler

NetScaler は、VKS ワークロードのイングレスプロキシとして機能し、SSLオフロード、ロードバランシング、およびL7ポリシー適用により南北アプリケーション トラフィックを処理します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler は Kubernetes Ingress API と Kubernetes Gateway API の両方をサポートしているため、チームは環境に合ったモデルを選択できます。

コントローラー	API	使用条件
NetScaler Kubernetes Gatewayコントローラー	Kubernetes ゲートウェイ API	新規デプロイ。プラットフォームチームがGatewayリソースを管理し、アプリケーションチームがHTTPRouteルールを管理します
NetScaler Ingressコントローラー	Kubernetes イングレス API	既存のIngressベースのワークロード。HTTPおよびTCP/SSLトラフィック

NetScalerは両方のコントローラーを提供しており、必要なのはどちらか一方だけです。既にKubernetes Ingressリソースを実行している場合、NetScaler Ingress Controller は既存のマニフェストを変更せずに動作します。新規に開始する場合は、NetScaler Kubernetes Gateway Controller が推奨される選択肢です。これはKubernetesプロジェクトによって定義された標準であるKubernetes Gateway APIを実装しており、標準のGatewayおよびHTTPRouteリソースを通じて、プラットフォームチームとアプリケーションチームに明確な責任分担を提供します。

両方のコントローラーは、外部ロードバランサーとしてNetScaler VPX を使用します。アプリケーションポッドには、VKSワーカーノード上のNodePortを使用して到達します。NetScalerとKubernetesクラスターノードがL2ネットワークを共有している場合、NetScaler Ingress Controllerで `nodeWatch=true` を設定すると、各ノードのポッドCIDRへの静的ルートが自動的にプログラムされます。L3で分離されている場合は、NetScaler Node Controller を使用してください。これはVXLANオーバーレイを構築し、ポッドIPに到達可能にします。

ロードバランシング、TLSオフロード、WAF、レスポンダーおよび書き換えポリシー、レート制限、可観測性など、すべてがKubernetesマニフェストを通じて構成されます。

ソリューションアーキテクチャ

NetScaler VPXは、VCF上の仮想マシンとしてVKSクラスターの外部に配置されます。これは、すべての南北トラフィック（SSL終端、ルーティング、アプリケーションポッドへのロードバランシング）を処理します。これは、検証済みの両方のVKSネットワークトポロジーで機能します。

NetScaler Ingress ControllerとNetScaler Kubernetes Gateway ControllerはVKSクラスター内で実行され、トラフィックパスにはありません。これらはKubernetesリソースを監視し、目的の

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler構成を計算し、NITRO REST APIを通じて変更をプッシュします。開発者が新しいIngressまたはGateway APIリソースを適用すると、手動介入なしでNetScalerが数秒以内に更新されます。

検証済みのVKSネットワークトポロジー:

- VCFネットワーク（VPC / NSX）を備えたSupervisor。参照: Supervisor Architecture with VPC Networking
- Foundation ロードバランサー (vSphere Distributed Switch) を備えたスーパーバイザー。参照: Foundation ロードバランサー を備えたスーパーバイザー

ソリューションの検証

ソフトウェアバージョン：

コンポーネント	バージョン
ヴイエムウェア クラウド ファンデーション	9.0
ヴイスフィア クバネティス サービス (VKS)	3.6
ヴイスフィア クバネティス リリース (VKr)	1.35
NetScaler VPX	14.1
ネットスケーラー クバネティス ゲートウェイ コントローラー	2.1.0
ネットスケーラー イングレス コントローラー	4.1.17

検証済み構成:

構成	結果
ゲートウェイコントローラー / ゲートウェイ API	VPC / NSX — NodePort 検証済み
ゲートウェイコントローラー / ゲートウェイ API	Foundation LB / vDS — NodePort 検証済み
イングレスコントローラー / HTTP	VPC / NSX — NodePort 検証済み

© 1997–2026 Citrix Systems, Inc. All rights reserved.

構成	結果
イングレスコントローラー / HTTP	Foundation LB / vDS — NodePort 検証済み
イングレスコントローラー / TCP/SSL	VPC / NSX — NodePort 検証済み
イングレスコントローラー / TCP/SSL	Foundation LB / vDS — NodePort 検証済み

Ingress リソースまたは Gateway/HTTPRoute の定義は、どちらのネットワークトポロジでも同じです。

VKS に NetScaler をデプロイする

前提条件

VKS クラスター

- VCF 9.0 以降の VKS ワークロードクラスター (どちらのネットワークトポロジでも可)
- cluster-admin 権限を持つ `kubectl` アクセス
- ワークステーション上の HELM 3。

NetScaler VPX

- VPX は、ノードと同じサブネットに SNIP が構成されている VKS ワーカーノードから到達可能である必要があります
- イングレスフロントエンド用に少なくとも1つの空き VIP
- NetScaler アカウントには書き込みアクセス権が必要です。コントローラーは NITRO API を介して仮想サーバーとサービスグループを作成および更新するため、読み取り専用アカウントでは機能しません。

ネットワーク

- VKS ノードはポート 443 で NetScaler NSIP に到達できる必要があります。NetScaler SNIP は NodePort 範囲 (30000〜32767) で VKS ノードに到達できる必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

このガイドでは、両方のデプロイオプションについて説明します。パートIではGateway Controller (Gateway API)について、パートIIではIngress Controller (Ingress API)について説明します。本番環境では、どちらか一方のみが必要です。決定する前に両方を評価したい場合は、同じクラスターで両方のパートを実行できます。

パートI: ゲートウェイAPI — ネットスケーラー Kubernetes ゲートウェイコントローラー

Gateway ControllerはKubernetes Gateway APIを実装します。プラットフォームチームはGatewayリソースとセキュリティポリシーを管理します。アプリケーションチームはHTTPRouteルールを管理します。すべての設定はKubernetesマニフェストにあります。誰もNetScalerにログインしません。

ステップ1：GatewayClassを作成する

```
apiVersion: gateway.networking.k8s.io/v1

kind: GatewayClass

metadata:

  name: gateway-class

spec:

  controllerName: citrix.com/nsgw-controller
```

```
kubectl apply -f gatewayclass.yaml
```

ステップ2：サンプルアプリケーションをデプロイする

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: webapp

  labels:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    name: webapp
    app: webapp
spec:
  selector:
    matchLabels:
      app: webapp
  replicas: 2
  template:
    metadata:
      labels:
        name: webapp
        app: webapp
    spec:
      containers:
      - name: webapp
        image: quay.io/sample-apps/cnn-website:v1.0.0
        ports:
        - name: http-80
          containerPort: 80
        - name: https-443
          containerPort: 443
---
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: v1
kind: Service
metadata:
  name: webapp
  labels:
    app: webapp
spec:
  type: NodePort
  ports:
  - name: http-80
    port: 80
    targetPort: 80
  - name: https-443
    port: 443
    targetPort: 443
    appProtocol: ssl
  selector:
    name: webapp
```

```
kubectl apply -f webapp.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ステップ3：TLS Secretを作成する

```
openssl genrsa -out webapp_key.pem 2048

openssl req -new -key webapp_key.pem -out webapp_csr.pem \

  -subj "/CN=webapp.demonetscaler.com"

openssl x509 -req -in webapp_csr.pem -sha256 -days 365 \

  -extensions v3_ca -signkey webapp_key.pem \

  -CAcreateserial -out webapp_cert.pem

kubectl create secret tls webapp-secret \

  --key webapp_key.pem --cert webapp_cert.pem
```

本番環境では、自己署名証明書をCAからのものに置き換えてください。残りの手順は同じです。

ステップ4：NetScaler Credential Secretを作成する

```
kubectl create secret generic nslogin \

  --from-literal=username=<username> \

  --from-literal=password=<password>
```

専用のシステムユーザーアカウントの作成方法については、NetScaler Ingress Controllerのシステムユーザーアカウントを作成するを参照してください。

ステップ5：Gateway Controllerをデプロイする

`values.yaml`:

```
netscaler:

 adcCredentialSecret: "nslogin"

 nsIP: "<NSIP/SNIP/CLIP>" #NSIP for standalone VPX, SNIP with management
access enabled for HA VPX or CLIP for VPX cluster mode
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
gatewayController:

 entityPrefix: gwy

 gatewayControllerName: "citrix.com/nsgw-controller"



license:

 accept: yes



nodeWatch: true
```

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/

helm install gateway-controller \

  netscaler/netscaler-kubernetes-gateway-controller \

  -f values.yaml

kubectl get pods -l app=gateway-controller
```

ステップ6：GatewayとHTTPRouteをデプロイする

NetScalerの空いている仮想IPを`<VIP_ADDRESS>`に置き換えてください。

```
apiVersion: gateway.networking.k8s.io/v1

kind: Gateway

metadata:

  name: vpx-gateway

spec:

  gatewayClassName: gateway-class
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  listeners:
  - name: https
    protocol: HTTPS
    port: 443
    tls:
      mode: Terminate
      certificateRefs:
      - kind: Secret
        name: webapp-secret
  addresses:
  - type: IPAddress
    value: <VIP_ADDRESS>
---
apiVersion: gateway.networking.k8s.io/v1
kind: HTTPRoute
metadata:
  name: http-route
spec:
  parentRefs:
  - name: vpx-gateway
  hostnames:
  - "webapp.demonetscaler.com"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  rules:

  - matches:

    - path:

        type: PathPrefix

        value: /

    backendRefs:

    - name: webapp

      namespace: default

      port: 443
```

```
kubectl apply -f gateway.yaml

kubectl get gateway vpx-gateway

kubectl get httproute http-route
```

ステップ7：テスト

```
curl -vik --resolve webapp.demonetscaler.com:443:<VIP_ADDRESS> \

  https://webapp.demonetscaler.com
```

期待される結果：NetScaler VIPからHTTPS経由で提供されるHTTP/2 200。

セキュリティユースケース: 不正なURLパスをブロックする

`rewritepolicy` CRDを使用してNetScaler Responderポリシーを適用し、ExtensionRefフィルターを使用してHTTPRouteにアタッチします。アプリケーションの変更は不要です。

```
apiVersion: citrix.com/v1

kind: rewritepolicy
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
metadata:

  name: blacklisturls

spec:

  responder-policies:

  - responder-policy:

      respondwith:

        http-payload-string: >

          "HTTP/1.1 403 Forbidden\r\n\r\n" +

          "Client: " + CLIENT.IP.SRC +

          " is not authorized to access URL:" +

          HTTP.REQ.URL.HTTP_URL_SAFE + "\n"

      respond-criteria: 'http.req.url.equals_any("blacklistUrls")'

      comment: 'Blacklist certain URLs'

    patset:

    - name: blacklistUrls

      values:

      - '/.hidden'

      - '/.password'
```

ポリシーを参照するようにHTTPRouteを更新します:

```
filters:

- type: ExtensionRef
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  extensionRef:
    group: citrix.com
    kind: rewritepolicy
    name: blacklisturls
```

ブロックされたパスが403 Forbiddenを返すことをテストします:

```
curl -vik --resolve webapp.demonetscaler.com:443:<VIP_ADDRESS> \
  https://webapp.demonetscaler.com/.hidden
```

第II部: イングレスAPI — NetScalerイングレスコントローラー

このセクションでは、同じクラスター上の2つのサービスタイプについて説明します。HTTPアプリケーション (Apache) とTCP/SSLアプリケーション (tcp-echo) の両方が、標準のKubernetes Ingressリソースを使用してNetScaler VPXを介して公開されます。

ステップ1: アプリケーションのデプロイ

アパッチ HTTP:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: apache
  labels:
    name: apache
spec:
  selector:
    matchLabels:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      app: apache
  replicas: 2
  template:
    metadata:
      labels:
        app: apache
    spec:
      containers:
      - name: apache
        image: httpd:latest
        ports:
        - containerPort: 80
---
apiVersion: v1
kind: Service
metadata:
  name: apache
spec:
  ports:
  - port: 80
    targetPort: 80
  selector:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    app: apache
```

TCPエコー (TCP/SSL):

```
apiVersion: v1
kind: Service
metadata:
  name: tcp-echo
  labels:
    app: tcp-echo
spec:
  type: NodePort
  ports:
  - name: tcp
    port: 9000
  - name: tcp-other
    port: 9001
  selector:
    app: tcp-echo
---
apiVersion: apps/v1
kind: Deployment
metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: tcp-echo
spec:
  replicas: 1
  selector:
    matchLabels:
      app: tcp-echo
      version: v1
  template:
    metadata:
      labels:
        app: tcp-echo
        version: v1
    spec:
      containers:
      - name: tcp-echo
        image: docker.io/istio/tcp-echo-server:1.3
        args: ["9000,9001,9002", "hello"]
        ports:
        - containerPort: 9000
        - containerPort: 9001
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl apply -f apache.yaml

kubectl apply -f tcp-echo.yaml
```

ステップ2: TLSシークレットの作成

```
# TCP-echo cert

openssl genrsa -out tcp_key.pem 2048

openssl req -new -key tcp_key.pem -out tcp_csr.pem -subj "/CN=*.tcp-
echo.com"

openssl x509 -req -in tcp_csr.pem -sha256 -days 365 \

  -extensions v3_ca -signkey tcp_key.pem -CAcreateserial -out tcp_cert.pem

kubectl create secret tls tcp-ssl-cert --key tcp_key.pem --cert
tcp_cert.pem



# Apache cert

openssl genrsa -out apache_key.pem 2048

openssl req -new -key apache_key.pem -out apache_csr.pem -subj "/
CN=*.demonetscaler.com"

openssl x509 -req -in apache_csr.pem -sha256 -days 365 \

  -extensions v3_ca -signkey apache_key.pem -CAcreateserial -out
apache_cert.pem

kubectl create secret tls apache-secret --key apache_key.pem --cert
apache_cert.pem
```

ステップ3: NetScalerクレデンシャルシークレットの作成

パートIで`nslogin`をすでに作成している場合は、このステップをスキップしてください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl create secret generic nslogin \

 --from-literal=username=<username> \

 --from-literal=password=<password>
```

ステップ4: Ingress Controllerのデプロイ

`values.yaml`:

```
adcCredentialSecret: nslogin

entityPrefix: unified

ingressClass: ['vpx']

serviceClass: ['vpx']

license:

 accept: yes

nsIP: <NSIP/SNIP/CLIP> #NSIP for standalone VPX, SNIP with management
access enabled for HA VPX or CLIP for VPX cluster mode

nodeWatch: true
```

```
helm install nsic netscaler/netscaler-ingress-controller \

 -f values.yaml

kubectl get pods
```

参照先は次のとおりです: netscaler-helm-charts / netscaler-ingress-controller

ステップ5: Ingressリソースの作成

TCP/SSL Ingress — `<VIP_ADDRESS>`を置き換える:

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: vpx-tcp-ssl

  annotations:

    ingress.citrix.com/secure-service-type: ssl_tcp

    ingress.citrix.com/frontend-ip: "<VIP_ADDRESS>"

    ingress.citrix.com/secure-port: "6379"

spec:

  tls:

  - secretName: tcp-ssl-cert

  ingressClassName: vpx

  defaultBackend:

    service:

      name: tcp-echo

      port:

        number: 9000
```

Apache用HTTPS Ingress — `<VIP_ADDRESS>`を置き換える:

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: vpx-ingress-apache
  annotations:
    ingress.citrix.com/frontend-ip: <VIP_ADDRESS>
    ingress.citrix.com/secure-port: '8443'
spec:
  tls:
  - secretName: "apache-secret"
  ingressClassName: vpx
  rules:
  - host: apache.demonetscaler.com
    http:
      paths:
      - backend:
          service:
            name: apache
            port:
              number: 80
        path: /
        pathType: Prefix
```

```
kubectl apply -f vpx-tcp-ssl-ingress.yaml
kubectl apply -f vpx-apache-ingress.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl get ingress
```

ステップ6: テスト

```
# TCP/SSL

openssl s_client -connect <VIP_ADDRESS>:6379 \

  -cert tcp_cert.pem -key tcp_key.pem



# Apache HTTPS

curl -vik --resolve apache.demonetscaler.com:8443:<VIP_ADDRESS> \

  https://apache.demonetscaler.com:8443
```

期待される結果: ApacheからのHTTP/2 200、TLSハンドシェイクの成功、および`tcp-echo`からのエコー応答。

トラブルシューティング

コントローラーログから始めます。ほとんどの障害はそこで説明されています。

```
# Gateway Controller

kubectl logs -l app=gateway-controller --tail=100



# Ingress Controller

kubectl logs -l app=nsic --tail=100
```

症状	最も可能性の高い原因	確認事項
コントローラーポッドのCrashLoopBackOff	不正な認証情報またはNSIPに到達不能	ログ; クラスター内から`telnet <NSIP> 443`をテスト

© 1997–2026 Citrix Systems, Inc. All rights reserved.

症状	最も可能性の高い原因	確認事項
NetScaler上に何も作成されていない	IngressClassまたはGatewayClassの名前の不一致	`kubectl get ingress -o yaml` と Helm ingressClass の値
コンテンツスイッチング仮想サーバーがダウンしているか、503エラー	NetScaler が VKS ノード上の NodePort に到達できない	`curl http://<node-ip>:<nodeport>`; ポート範囲 30000〜32767 のファイアウォールを確認してください
TLSハンドシェイクが失敗する	誤った名前空間のシークレット	シークレットは、コントローラーの名前空間ではなく、Ingress または Gateway と同じ名前空間にある必要があります
VIPからの404	ホストヘッダーが一致しない	cURL で `--resolve` が使用されていることを確認してください。HTTPRoute のホスト名または Ingress ルールのホストを確認してください
特定のパスでの403	レスポンダーポリシーが期待どおりに機能している	ブロックされていないパスをテストして、基本ルーティングが正常であることを確認してください

VMware VKS 上の NetScaler は、運用上の複雑さを増すことなく、Kubernetes 上でのエンタープライズアプリケーション配信への実用的な道筋をチームに提供します。プラットフォームチームは、すでに知っている NetScaler ポリシーを使い続けます。開発者は Kubernetes マニフェストで作業します。VKS はクラスターのライフサイクルを処理し、NetScaler はトラフィックを処理し、コントローラーは両者を自動的に同期させます。

結論

VMware VKS上のNetScalerは、運用上の複雑さを増すことなく、Kubernetes上でのエンタープライズアプリケーション配信への実用的な道筋をチームに提供します。プラットフォームチームは、すでに知っているNetScalerポリシーを使い続けます。開発者はKubernetesマニフェストで作業します。VKSがクラスターのライフサイクルを処理し、NetScalerがトラフィックを処理し、コントローラーが両者を自動的に同期させます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

リソース

- ネットスケーラー クバネティス ゲートウェイコントローラー ヘルムチャート のリードミー
- NetScaler Ingress Controller Helm チャート リードミー
- NetScaler Ingress Controller ドキュメント
- NetScaler Kubernetes Gateway Controller ドキュメント
- VPCネットワーキングを使用したVKS
- Foundationロードバランサーを使用したVKS
- VKS製品ページ(https://www.vmware.com/docs/the-unified-platform-for-all-workloads)

© 1997–2026 Citrix Systems, Inc. All rights reserved.

設定

設定には、以下のトピックがあります。

- アノテーション
- ConfigMap
- イングレス
- Ingressクラス
- サービス クラス
- HTTP、TCP、またはSSLプロファイル
- ログレベル
- Kubernetesシークレットリソースを介したLoadBalancerタイプのサービス用SSL証明書
- NetScaler CPXを使用したLoadBalancerタイプのサービスおよびIngressのBGPアドバタイズメント
- オンプレミスKubernetesクラスター向けレイヤー2モードでのMetalLBとNetScaler CPXの統合
- HTTPRoute CRDを使用したKubernetes Ingressの高度なコンテンツルーティング
- Listener CRDのプロファイルサポート
- IngressリソースのNetScaler IPAMコントローラーを使用したIPアドレス管理
- アノテーションを使用してCRDを適用する(/ja-jp/netscaler-k8s-ingress-controller/configure/crds-annotations.html)
- アノテーションを介したIngressのリスナーCRDサポート(/ja-jp/netscaler-k8s-ingress-controller/configure/listener-crd-ingress.html)
- リスナーでのSSLパススルーサポート(/ja-jp/netscaler-k8s-ingress-controller/configure/ssl-passthrough-multicluster.html)
- 一貫性のあるハッシュアルゴリズム(/ja-jp/netscaler-k8s-ingress-controller/configure/hashing-algorithm.html)
- NetScaler Ingress Controllerを使用したDNSレコードの追加(/ja-jp/netscaler-k8s-ingress-controller/configure/add-dns-records.html)
- OPAを構成する(/ja-jp/netscaler-k8s-ingress-controller/configure/opa.html)

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- メトリクスをPrometheusに直接エクスポートする(/ja-jp/netscaler-k8s-ingress-controller/configure/direct-promethues-export-support.html)
- 管理者パーティションを持つNetScaler用のNetScaler Ingress Controllerを展開する(/ja-jp/netscaler-k8s-ingress-controller/deploy/cic-adc-admin-partition.html)
- ネットワーク(/ja-jp/netscaler-k8s-ingress-controller/network/network.html)
- 証明書管理(/ja-jp/netscaler-k8s-ingress-controller/certificate-management/certificate-management.html)
- CRD(/ja-jp/netscaler-k8s-ingress-controller/crds/crds.html)
- メトリクス(/ja-jp/netscaler-k8s-ingress-controller/metrics/metrics.html)

アノテーション

Ingressアノテーション

NetScalerでサポートされているIngressアノテーションは次のとおりです。

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
`ingress.citrix.com/multicluster-policy-priority-order`	文字列	オプション	マルチクラスターIngressセットアップで異なるアプリケーションの負荷分散を行う場合、アプリケーションごとに個別のコンテンツスイッチングポリシーが作成されます。この	該当なし	ingress.citrix.com/マルチクラスター-ポリシー-優先順位: '{"フロントエンド": {"80": "3", "9443": "1"}, "バックエンド": "2"}'

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			ような場合、ポリシーバインディングに特定の順序が必要な場合は、`ingress.citrix.com/multicluster-policy-priority-order` アノテーションを使用してコンテンツスイッチングポリシーに優先度番号を割り当てる必要があります。詳細については、ポリシーバインディングを参照してください。		
`ingress.citrix.com/`	文字列	オプション	コンテンツスイッチング仮想サー	NA	数値IPアドレス。例: `1.2.3.4`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値	
`frontend-ip`			バーのIPアドレスとして使用する必要があるIPアドレスを指定します。**注:** コンテンツスイッチング仮想サーバーのIPアドレスを設定する方法は、IPAM構成、デフォルトのnsVIPなど、複数あります。			
`ingress.citrix.com/frontend-ipset-name`	文字列	オプション	コンテンツスイッチング仮想サーバーにバインドする必要があるIPSET名を指定します。このアノテーションを `ingress`	該当なし	NetScaler IPSETエンティティ名	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値	
			`.citrix.com/frontend-ip` と組み合わせて使用します。**注:** アノテーションで指定するIPSET名は、NetScalerで既に構成されている必要があります。			
`ingress.citrix.com/insecure-service-type`	文字列	オプション	コンテンツスイッチング仮想サーバー向けに、HTTP/TCP/UDP/sip_UDP/anyの中からプロトコルを指定します。	`http`	`http`、`tcp`、`udp`、`sip_udp`、または `any`	
`ingress.citrix.com/insecure-port`	文字列	オプション	http/tcp/udp/sip_udp/any プロトコル用	`80`	有効なポート番号	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			のコンテンツスイッチング仮想サーバーのポートを設定します。		
`ingress.citrix.com/secure-service-type`	文字列	オプション	コンテンツスイッチング仮想サーバーのプロトコルとして、SSL/SSL_TCPの中からプロトコルを指定します。	`ssl`	`ssl`, `ssl_tcp`
`ingress.citrix.com/secure-port`	文字列	オプション	HTTPSトラフィック用のコンテンツスイッチング仮想サーバーのポートを設定します。	`443`	有効なポート番号
`ingress.citrix.com/insecure-`	文字列	オプション	HTTPトラフィックの動作を設定します。HTTPト	`disallow`	`allow`、`redirect`、または

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
`termination`			ラフィックを許可するには `allow` を使用し、HTTPリクエストをHTTPSにリダイレクトするには `redirect` を使用します。HTTPトラフィックを破棄したい場合は `disallow` を使用します。		`disallow`
`ingress.citrix.com/default-response-code`	文字列	オプション	リクエストがデフォルトのコンテンツスイッチングまたはロードバランシング仮想サーバーに到達したときにNetScalerが送信するデフォ	NA	-

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			ルトの応答を設定します。たとえば、リクエストが設定されたホストまたはパスのいずれにも一致しない場合、またはバックエンドサービスに準備完了のエンドポイントがない場合などです。アクションタイプ（respondwith、redirect、またはrespondwithhtmlpage）、HTTP応答コード、およびオプションでターゲットURLと理由フ		

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			レーズをJSONとして指定します。詳細については、アノテーションキー `ingress.citrix.com/default-response-code` を参照してください。		
			`ingress.citrix.com/default-response-code: '{"type": "respondwithhtmlpage", "response-code": "521", "target": "https://status.example.com/`		

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			`service-unavailable.html", "reason-phrase": "Web server down"}'`		
`ingress.citrix.com/secure-backend`	文字列/JSON	オプション	NetScalerとバックエンドのKubernetesアプリケーション間で安全なHTTPS接続が必要な場合に指定します。指定された値がtrueまたはfalseの場合、イングレスYAML内のすべてのサービスに適用されます。各サービスに異なる設定が必要な場合は、以下	`False`	文字列として: `True` / `False` 。 JSONとして: `'{"<Service_Name>": "True/False", ...}'` 。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値	
			のようにJSON形式で値を指定してください。**例：** `ingress.citrix.com/secure-backend: '{"app1":"True", "app2":"False", "app3":"True"}'` 。ここでは、`app1` と `app3` はHTTPS経由でバックエンドと通信します。`app2` はプレーンな HTTP になります。			
`ingress.citrix.com/backend-secret`	文字列/JSON	オプション	NetScalerとKubernetes ポッド間のバックエンド通信に使	NA	文字列として：`"Kubernetes secret"` 。JSON	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			用する証明書を指定します。		として：`'{"<Service_Name>": "<kubernetes secret>", ...}'`
`ingress.citrix.com/backend-ca-secret`	文字列/JSON	オプション	NetScalerとKubernetes ポッド間のバックエンド通信に使用するCA 証明書を指定します。	NA	文字列として：`"kubernetes secret"`、JSONとして：`'{"<Service_Name>": "<kubernetes secret>", ...}'`
`ingress.citrix.com/preconfigured-certkey`	JSON	任意	コンテンツスイッチング仮想サーバー用に構成する必要がある、NetScaler上に既に存在するSSL証明書キーを	NA	証明書タイプdefault/sni/ca を持つ、1つ以上のNetScaler sslcertkeyエンティティ名。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値	
			指定します。**例:** `ingress.citrix.com/preconfigured-certkey: '{"certs": [{"name": "certkey1", "type": "default"}, {"name": "certkey2", "type": "sni"}]}'`。ここでは、certkey1が非SNIデフォルト証明書として使用され、certkey2がSNI証明書として使用されます。このセクションを参照して			

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値	
			ください。			
`ingress.citrix.com/preconfigured-backend-certkey`	文字列	任意	NetScalerで事前に設定された証明書キーの名前を指定し、バックエンドSSLサービスグループにバインドします。この証明書は、サーバー認証のためにSSLハンドシェイク中にサーバーに送信されます。**例:** `ingress.citrix.com/preconfigured-backend-certkey: '{"frontend-hotdrinks": "abc4cert",`	NA	NetScaler sslcertkeyエンティティ名	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値	
			`"coffee-beverage": "abc4cert"}'`			
`ingress.citrix.com/preconfigured-backend-ca-certkey`	文字列	オプション	NetScalerで事前に設定されたCA証明書キーの名前を指定し、サーバー認証のためにバックエンドSSLサービスグループにバインドします。 **例:** `ingress.citrix.com/preconfigured-backend-ca-certkey: '{"frontend-hotdrinks": "ca3bun", "coffee-beverage`	NA	NetScaler sslcertkey エンティティ名	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値	
			`": "ca3bun" }'`			
`ingress.citrix.com/lbvserver`	JSON	オプション	NetScaler LBVserverエンティティの設定/パラメータを構成します。**例:** `ingress.citrix.com/lbvserver: '{"app1": {"lbmethod":"SRCIPDESTIPHASH"}}'`。これらのパラメータを設定することで、さまざまなユースケースを実現できます。このセクションを参照してください。	該当なし	`key:value` 形式の有効なNetScalerエンティティパラメータ。	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
`ingress.citrix.com/servicegroup`	JSON	オプション	NetScaler Servicegroupエンティティの設定/パラメータを構成します。**例:** `ingress.citrix.com/servicegroup: '{"appname":{"cip":"Enabled","cipHeader":"X-Forwarded-For"}}'`。これらのパラメータを設定することで、さまざまなユースケースを実現できます。このセクションを参照してください。	NA	`key:value`形式の有効なNetScalerエンティティパラメータ。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
`ingress.citrix.com/monitor`	JSON	任意	• NetScaler モニターエンティティの設定/パラメータを構成します。 **例:** `ingress.citrix.com/monitor: '{"apname":{"type":"http"}}'`。このセクションを	N/A	`key:value` 形式の有効な NetScaler エンティティパラメータ。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			参照してください。 • 複数のバックエンドサービスに同じ事前設定済みモニターをバインドします。 **例:** `ingress.citrix.com/monitor: '{"apname":`		

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			`"<pre-configured-monitor-on-netscaler>", "appname2": "<pre-configured-monitor-on-netscaler>"}'`. • 各バックエンドサービスに異なる事前		

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			設定済みモニターをバインドします。 **例:** `ingress.citrix.com/monitor: '{"appname": "<pre-configured-monitor1-on-netscaler>", "appname2": "<pre`		

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値	
			`-configured-monitor2-on-netscaler>"}'`.			
`ingress.citrix.com/deployment`	文字列	オプション	NetScaler上でDirect Server Return (DSR) 設定を作成します。**例:** `ingress.citrix.com/deployment: "dsr"`	該当なし	`dsr`	
`ingress.citrix.com/path-match-method`	文字列	オプション	イングレス内のアプリケーションのパス照合を指定します。**例:** `ingress.citrix.com/path-match-`	`prefix`	`prefix` または `exact`。	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値	
			`method: "prefix"`。このクラスは下位互換性のために維持されています。代わりにspecで `pathType:` を使用してください。			
`ingress.citrix.com/ipam-range`	文字列	オプション	NetScaler IPAMコントローラーに指定された範囲のセットから特定のIPアドレス範囲を選択します。**例:** `ingress.citrix.com/ipam-range: 'Dev'`	N/A	IPAMコントローラーで構成された範囲名のいずれかに一致する値。	
`ingress.citrix.com/`	JSON	オプション	NetScalerでDNSサーバーを設定するために	NA	NA	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値	
`external-service`			ドメイン名を提供します。このセクションを参照してください。**例:** `ingress.citrix.com/external-service: '{"external-svc": {"domain": "www.externalsvc.com"}}'`			
`ingress.citrix.com/canary-weight`	文字列	任意	カナリアバージョンに誘導されるトラフィックの割合を指定します。このセクションを参照してください。**例:** `ingress.citrix.com/canary-`	NA	NA	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値	
			`weight: "10"`			
`ingress.citrix.com/canary-by-header`	文字列	任意	カナリアバージョンにトラフィックを誘導するためのHTTPヘッダーキーを指定します。このセクションを参照してください。**例:** `ingress.citrix.com/canary-by-header: 'some-header-info'`	該当なし		
`ingress.citrix.com/canary-by-header-value`	リスト	オプション	カナリアバージョンにトラフィックを誘導するためのHTTPヘッダー値を提供します。このセクションを	該当なし	文字列としてのヘッダー値のリスト。	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値	
			参照してください。**例:** `ingress.citrix.com/canary-by-header-value: '["value1","value2","value3"]'`			
`ingress.citrix.com/bot_crd`	文字列/JSON	オプション	BOT CRDによって作成されたポリシーをアプリケーションの負荷分散仮想サーバーにバインドします。このセクションを参照してください。**例:** `ingress.citrix.com/bot_crd: "botdefense"` はイングレス内のす	N/A	文字列として: `"CRD_Instance_Name"`。JSONとして: `'{"<Service_Name>":"CRD_Instance_Name"}'`	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値	
			べてのサービスにポリシーをバインドし、または `ingress.citrix.com/bot_crd: '{"appname": "botdefense"}'` はフロントエンドサービスのみにポリシーをバインドします。			
`ingress.citrix.com/ratelimit_crd`	文字列/JSON	オプション	Ratelimit CRDによって作成されたポリシーをアプリケーションの負荷分散仮想サーバーにバインドします。このセクションを参照してください。	N/A	文字列として: `"CRD_Instance_Name"`。JSONとして: `'{"<Service_Name>":"CRD_Instance_Name"}'`。	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			例: `ingress.citrix.com/ratelimit_crd: "ratelimitexample"` はポリシーをイングレス内のすべてのサービスにバインドするか、`ingress.citrix.com/ratelimit_crd: '{"appname": "ratelimitexample"}'` はポリシーをフロントエンドサービスのみにバインドします。		
`ingress.citrix.com/auth_crd`	文字列/JSON	オプション	Auth CRD によって作成されたポリ	N/A	文字列として: `"CRD_Instance_N`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			シーを、アプリケーションのロードバランシング仮想サーバーにバインドします。このセクションを参照してください。 **例:** `ingress.citrix.com/auth_crd: "authexample"` はポリシーをイングレス内のすべてのサービスにバインドするか、`ingress.citrix.com/auth_crd: '{"appname": "authexa`		`ame"`、JSONとして: `{"<Service_Name>":"CRD_Instance_Name"}`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			`mple"}'` はポリシーをフロントエンドサービスのみにバインドします。		
`ingress.citrix.com/waf_crd`	文字列/JSON	オプション	WAF CRDによって作成されたポリシーを、アプリケーションの負荷分散仮想サーバーにバインドします。このセクションを参照してください。 **例:** `ingress.citrix.com/waf_crd: "wafbasic"` はポリシーをイングレス内のすべての	NA	文字列として: "CRD_Instance_Name", JSONとして: `'{"<Service_Name>":"CRD_Instance_Name"}'`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			サービスにバインドするか、`ingress.citrix.com/waf_crd: '{"appname": "wafbasic"}'` はポリシーをフロントエンドサービスのみにバインドします		
`ingress.citrix.com/rewrite_responder_crd`	String/JSON	オプション	Rewrite-Responder CRDによって作成されたポリシーを、アプリケーションのロードバランシング仮想サーバーにバインドします。このセクションを参照してください。	N/A	文字列として: `"CRD_Instance_Name"`、JSONとして: `'{"<Service_Name>":"CRD_Instance_Name"}'`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			例: `ingress.citrix.com/rewrite_responder_crd: "blockurlpolicy"` は、イングレス内のすべてのサービスにポリシーをバインドします。または `ingress.citrix.com/rewrite_responder_crd: '{"appname": "blockurlpolicy"}'` は、フロントエンドサービスのみにポリシーをバインドします。		
`ingress.citrix.`	文字列/JSON	オプション	rewrite-responder	NA	文字列として:

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
`com/rewrite_responder_crd`			CRDによって作成されたポリシーを、アプリケーションの負荷分散仮想サーバーにバインドします。このセクションを参照してください。 **例:** `ingress.citrix.com/rewrite_responder_crd: "blockurlpolicy"` はポリシーをイングレス内のすべてのサービスにバインドし、`ingress.citrix.com/rewrite_responde`		`"CRD_Instance_Name"`。JSONとして: `'{"<Service_Name>":"CRD_Instance_Name"}'`。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			`r_crd: '{"appname": "blockurlpolicy"}'` はポリシーをフロントエンドサービスのみにバインドします。		

サービスアノテーション

以下は、NetScalerでサポートされているサービスアノテーションです。

サービスアノテーションでは、`index` はサービス仕様ファイル内のポートの順序付きインデックスです。例えば、サービス仕様に2つのポートがある場合、最初のポートのインデックスはゼロで、2番目のポートのインデックスは1です。

アノテーション	タイプ	必須	説明	デフォルト	可能な値
`service.citrix.com/frontend-ip`	文字列	オプション	コンテンツスイッチング仮想サーバーのIPアドレスとして使用する必要があるIPアドレスを指定します。コンテンツスイッチング仮	該当なし	数値IPアドレス（例: '1.2.3.4'）

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	可能な値	
			想サーバーのIPアドレスを指定する方法は複数あります。			
`service.citrix.com/ipam-range`	文字列	オプション	NetScaler IPAMコントローラーに指定された範囲のセットから、特定のIPアドレス範囲を選択します。このアノテーションは、LoadBalancerタイプのサービスに使用されます。**例:** `service.citrix.com/ipam-range: 'Dev'`	該当なし	IPAMコントローラーで構成された範囲名のいずれかに一致する値	
`service.citrix.com/insecure`	JSON	任意	HTTPトラフィックをセキュアな	該当なし		

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	可能な値	
`-redirect`			ポートにリダイレクトします。**例:** `service.citrix.com/insecure-redirect: '{"port-443": 80 }'` または `service.citrix.com/insecure-redirect: '{"443-tcp": 80 }'`			
`service.citrix.com/ssl-termination-<index>`	文字列	任意	SSLターミネーションを指定します。**例:** `service.citrix.com/ssl-termination-0: 'EDGE'`	該当なし	`EDGE` と `REENCRYPT`	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	可能な値	
`service.citrix.com/service-type-<index>`	文字列	オプション	作成されるNetScalerエンティティのプロトコルを指定します。**例:** `service.citrix.com/service-type-0: 'SSL'`。	`HTTP`	`TCP`、`HTTP`、`SSL`、`UDP`、`ANY`、`SSL_TCP`、および`SIP_UDP`。	
`service.citrix.com/ssl-certificate-data-<index>`	文字列	オプション	PEM形式でサーバー証明書値を指定します。**例:** `service.citrix.com/ssl-certificate-data-0: | <certificate>`	N/A	PEM形式の証明書データ	
`service.citrix.com/ssl-key-data-<index>`	文字列	任意	PEM形式でサーバーキーの値を指定します。**例:** `service.citrix.`	NA	PEM形式のキーデータ	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	可能な値	
			`com/ssl-key-data-0: | <key>`			
`service.citrix.com/ssl-ca-certificate-data-<index>`	文字列	任意	PEM形式でクライアント証明書を検証するためのサーバーCA証明書値を指定します。**例:** `service.citrix.com/ssl-ca-certificate-data-0: | <certificate>`	該当なし	PEM形式のCA証明書データ	
`service.citrix.com/ssl-backend-ca-certificate-data-<index>`	文字列	任意	PEM形式でバックエンドのサーバー証明書を検証するためのCA証明書値を指定します。**例:** `service.citrix.com/ssl-backend-`	該当なし	PEM形式のCA証明書データ	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	可能な値	
			`ca-certificate-data-0: | <certificate>`			
`service.citrix.com/secret`	文字列	オプション	フロントエンドサーバー証明書のシークレットリソース名を指定します。複数のフロントエンドサーバー証明書をバインドするには、証明書用に構成されたカンマ区切りのシークレットのリストを指定します。詳細と例については、ロードバランサータイプのサービス用SSL証	NA	Kubernetesシークレット名	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	可能な値	
			明書を参照してください。 **例:** `service.citrix.com/secret: hotdrink-secret`, `service.citrix.com/secret: apache-secret1, apache-secret2`			
`service.citrix.com/ca-secret`	文字列	オプション	クライアント証明書認証用のCA証明書を提供します。この証明書は、NetScalerのフロントエンドSSL仮想サーバーにバインドされます。詳細と例については、ロードバランサータイプの	該当なし	Kubernetesシークレット名	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	可能な値	
			サービス用SSL証明書を参照してください。 **例:** `service.citrix.com/ca-secret: 'hotdrink-ca-secret'`			
`service.citrix.com/backend-secret`	文字列^	任意	NetScalerとワークロード間のバックエンド通信が暗号化されたチャネル上で行われ、ワークロードでクライアント認証が必要な場合は、このアノテーションを使用します。この証明書は、SSLハンドシェイク中にサーバーに送	NA	Kubernetesシークレット名	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	可能な値	
			信され、バックエンドSSLサービスグループにバインドされます。詳細と例については、ロードバランサータイプのサービス用SSL証明書を参照してください。 **例:** `service.citrix.com/backend-secret: 'hotdrink-secret'`			
`service.citrix.com/backend-ca-secret`	文字列	任意	バックエンドサーバー証明書を認証するサーバー認証を有効にします。詳細と例については、ロードバラン	該当なし	Kubernetesシークレット名	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	可能な値	
			サータイプのサービス用SSL証明書を参照してください。**例:** `service.citrix.com/backend-ca-secret: 'hotdrink-ca-secret'`			
`service.citrix.com/preconfigured-certkey`	文字列	任意	フロントエンドサーバー証明書として使用するNetScalerの既存のSSL証明書キーの名前を指定します。複数のフロントエンドサーバー証明書をバインドするには、証明書用にすでに構成されてい	該当なし	NetScaler sslcertkey エンティティ名	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	可能な値	
			る、コンマ区切りの証明書キーのリストを指定します。**例:** - `service.citrix.com/preconfigured-certkey: coffee-cert`, `service.citrix.com/preconfigured-certkey: preconfcert1,preconfcert2`			
`service.citrix.com/preconfigured-ca-certkey`	文字列	オプション	クライアント証明書認証用のCA証明書として使用する、NetScalerで事前設定された証明書キーの名前を指定します。	NA	NetScaler sslcertkey エンティティ名	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	可能な値	
			この証明書は、NetScalerのフロントエンドSSL仮想サーバーにバインドされます。**例:** `service.citrix.com/preconfigured-backend-certkey: 'coffee-cert'`			
`service.citrix.com/preconfigured-backend-certkey`	文字列	オプション	NetScalerで事前設定された証明書キーの名前を指定し、バックエンドSSLサービスグループにバインドします。この証明書は、サーバー認証のためにSSLハンドシェイク	NA	NetScaler sslcertkeyエンティティ名	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	可能な値	
			中にサーバーに送信されます。**例:** `service.citrix.com/preconfigured-ca-certkey: 'coffee-ca-cert'`			
`service.citrix.com/preconfigured-backend-ca-certkey`	文字列	オプション	サーバー認証のために、NetScalerで事前設定されたCA証明書キーの名前を指定し、バックエンドSSLサービスグループにバインドします。**例:** `service.citrix.com/preconfigured-backend-ca-certkey:`	NA	NetScaler sslcertkeyエンティティ名	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	可能な値	
			`'coffee-ca-cert'`			

HTTP、TCP、またはSSLプロファイル用のスマートアノテーション

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
`ingress.citrix.com/frontend-httpprofile`	文字列/JSON	オプション	フロントエンドHTTPプロファイル（クライアントプレーン）を作成します	該当なし	**例:** `ingress.citrix.com/frontend-httpprofile: '{"dropinvalreqs":"enabled", "websocket" : "enabled"}'`
`ingress.citrix.com/backend-httpprofile`	文字列/JSON	オプション	バックエンドHTTPプロファイル（サーバープレーン）を作成します。	NA	**例:** `ingress.citrix.com/backend-httpprofile: '{"app-1": {"dropinvalreqs":"enabled", "websocket" : "enabled"}}'`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
`ingress.citrix.com/frontend-tcpprofile`	文字列/JSON	任意	フロントエンドTCPプロファイルを作成します (クライアントプレーン)	NA	**例:** `ingress.citrix.com/frontend-tcpprofile: '{"ws":"enabled", "sack" : "enabled"}'`
`ingress.citrix.com/backend-tcpprofile`	文字列/JSON	任意	バックエンドTCPプロファイルを作成します (サーバープレーン)	NA	**例:** `ingress.citrix.com/backend-tcpprofile: '{"citrix-svc": {"ws":"enabled", "sack" : "enabled"}}'`
`ingress.citrix.com/frontend-sslprofile`	文字列/JSON	オプション	フロントエンドSSLプロファイル（クライアントプレーン）を作成します。フロントエンドSSLプロファイルは、クライアントプレーンで	該当なし	**例:** `ingress.citrix.com/frontend-sslprofile: '{"hsts":"enabled", "tls12" : "enabled"}'`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーション	タイプ	必須	説明	デフォルト	指定可能な値
			TLSを有効にしている場合にのみ必要です。		
`ingress.citrix.com/backend-sslprofile`	文字列/JSON	オプション	バックエンドSSLプロファイル（サーバープレーン）を作成します。SSLバックエンドプロファイルは、`ingress.citrix.com/secure-backend`を使用する場合にのみ必要です。	該当なし	**例:** `ingress.citrix.com/backend-sslprofile: '{"citrix-svc":{"hsts":"enabled", "tls1" : "enabled"}}'`

Ingressのスマートアノテーション

スマートアノテーションは、NetScaler Ingress Controllerが提供するオプションで、NetScalerエンティティ名を使用してNetScaler機能を効率的に有効にします。NetScaler Ingress Controllerは、KubernetesのIngressをNetScalerオブジェクトのセットに変換します。スマートアノテーションを使用すると、これらのオブジェクトを効率的に制御できます。

注

スマートアノテーションを使用するには、NetScalerの機能とそれぞれのエンティティ名をよく理解している必要があります。NetScalerの機能とエンティティ名の詳細については、NetScalerドキュメントを参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

スマートアノテーションはJSON形式を入力として受け取ります。JSON形式で渡すキーと値は、NetScaler NITRO形式と一致する必要があります。NetScaler NITRO APIの詳細については、NetScaler REST APIs - NITRO documentationを参照してください。

例えば、`SRCIPDESTIPHASH`ベースのLBメソッドを有効にする場合、対応するNITROキーと値の形式`lbmethod`、`SRCIPDESTIPHASH`をそれぞれ使用する必要があります。

次の表に、NetScaler Ingress Controllerが提供するスマートアノテーションの詳細を示します。

NETSCALERエンティティ名	スマートアノテーション	例
エルビーブイサーバー	`ingress.citrix.com/lbvserver`	`ingress.citrix.com/lbvserver: '{"appname": {"lbmethod":"SRCIPDESTIPHASH"}}'`
サービスグループ	`ingress.citrix.com/servicegroup`	`ingress.citrix.com/servicegroup: '{"appname":{"cip": "Enabled","cipHeader":"X-Forwarded-For"}}'`
モニター	`ingress.citrix.com/monitor`	`ingress.citrix.com/monitor: '{"appname": {"type":"http"}}'`
シーエスブイサーバー	`ingress.citrix.com/csvserver`	`ingress.citrix.com/csvserver: '{"stateupdate": "ENABLED"}'`

HTTP、TCP、およびSSLプロファイル用のスマートアノテーションについては、「NetScalerでHTTP、TCP、またはSSLプロファイルを構成する」を参照してください。

スマートアノテーションを使用したIngress YAMLの例

次のIngress YAMLの例には、lbvserver、servicegroup、monitorなどのエンティティを使用してNetScaler機能を有効にするスマートアノテーションが含まれています。

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kind: Ingress

metadata:

  annotations:

    ingress.citrix.com/frontend-ip: 192.168.1.1

    ingress.citrix.com/insecure-port: "80"

    ingress.citrix.com/lbvserver: '{"appname":
{"lbmethod":"LEASTCONNECTION", "persistenceType":"SOURCEIP"}}'

    ingress.citrix.com/monitor: '{"appname":{"type":"http"}}'

    ingress.citrix.com/servicegroup: '{"appname":{"usip":"yes"}}'

  name: citrix

spec:

  rules:

  - host: citrix.org

    http:

      paths:

      - backend:

          service:

            name: appname

            port:

              number: 80

        path: /

        pathType: Prefix
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
EOF
```

サンプルのIngress YAMLには、サービスに関連するユースケース`citrix-svc`が含まれており、次の表はサンプルで使用されているスマートアノテーションについて説明しています。

スマートアノテーション	説明
`ingress.citrix.com/lbvserver: '{"appname": {"lbmethod":"LEASTCONNECTION", "persistenceType":"SOURCEIP"}}'`	ロードバランシング方式を最小接続に設定し、送信元IPアドレスの永続性も構成します。
`ingress.citrix.com/servicegroup: '{"appname":{"usip":"yes"}}'`	NetScalerデバイスで送信元IPモードの使用(USIP)を有効にします。NetScalerでUSIPを有効にすると、クライアントのIPアドレスを使用してバックエンドポッドと通信します。
`ingress.citrix.com/monitor: '{"appname":{"type":"http"}}'`	サービスグループのカスタムHTTPモニターを作成します。

注記:

複数のイングレスが同じフロントエンドIPアドレスとポートを共有している場合、複数のイングレス設定を通じて提供される競合する設定を持つことはできません。

デフォルトでは、コンテンツスイッチング仮想サーバーは、それにバインドされているターゲットの負荷分散仮想サーバーの状態に依存しません。アノテーション`ingress.citrix.com/csvserver: '{"stateupdate": "ENABLED"}'`は、コンテンツスイッチングポリシーを使用してそれにバインドされている負荷分散仮想サーバーの状態に基づいて、コンテンツスイッチング仮想サーバーが自身の状態を考慮するように設定します。

ルートのスマートアノテーション

Ingress と同様に、OpenShift ルートでもスマートアノテーションを使用できます。NetScaler Ingress Controller は、OpenShift のルートを NetScaler オブジェクトのセットに変換します。

次の表は、NetScaler Ingress Controller が提供するスマートアノテーションの詳細を示します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NETSCALER エンティティ名	スマートアノテーション	例
`lbvserver`	`route.citrix.com/lbvserver`	`route.citrix.com/lbvserver: '{"appname":{"lbmethod":"SRCIPDESTIPHASH"}}'`
`servicegroup`	`route.citrix.com/servicegroup`	`route.citrix.com/servicegroup: '{"appname":{"cip":"Enabled","cipHeader":"X-Forwarded-For"}}'`
`monitor`	`route.citrix.com/monitor`	`route.citrix.com/monitor: '{"appname":{"type":"http"}}'`
`certificates`	`route.citrix.com/preconfigured-certkey`	`route.citrix.com/preconfigured-certkey : '{"certs": [{"name": "certkey1", "type": "default"}, {"name": "certkey2", "type": "sni"}]}'`

スマートアノテーションを使用したルートマニフェストの例

次の例は、ルートYAMLファイルです。

```
kubectl apply -f - <<EOF

apiVersion: route.openshift.io/v1

kind: Route

metadata:

  name: citrix

  annotations:

    route.citrix.com/lbvserver: '{"appname":{"lbmethod":"LEASTCONNECTION",
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
"persistenceType":"SOURCEIP"}}'

    route.citrix.com/servicegroup: '{"appname":{"usip":"yes"}}'

    route.citrix.com/monitor: '{"appname":{"type":"http"}}'

    route.citrix.com/preconfigured-certkey : '{"certs": [{"name":
"certkey1", "type": "default"}, {"name": "certkey2", "type": "sni"}]}'

spec:

  host:  citrix.org

  port:

    targetPort: 80

  to:

    kind: Service

    name: appname

    weight: 100

  wildcardPolicy: None

EOF
```

サンプルルートマニフェストには、サービス`citrix-svc`に関連するユースケースが含まれており、次の表はサンプルルートで使用されているスマートアノテーションについて説明しています。

スマートアノテーション	説明
`route.citrix.com/lbvserver: '{"appname": {"lbmethod":"LEASTCONNECTION", "persistenceType":"SOURCEIP"}}'`	負荷分散方法を最小接続として設定し、送信元IPアドレスの永続性も構成します。
`route.citrix.com/servicegroup: '{"appname":{"usip":"yes"}}'`	NetScalerで送信元IPモードの使用 (USIP)を有効にします。NetScalerでUSIPを有効にする

© 1997–2026 Citrix Systems, Inc. All rights reserved.

スマートアノテーション	説明
	と、バックエンドポッドとの通信にクライアントのIPアドレスが使用されます。
`route.citrix.com/monitor: '{"appname":{"type":"http"}}'`	サービスグループ用のカスタムHTTPモニターを作成します。
`route.citrix.com/preconfigured-certkey : '{"certs": [{"name": "certkey1", "type": "default"}, {"name": "certkey2", "type": "sni"}]}'`	証明書タイプ `default/sni/c` を持つ1つ以上のNetScaler `sslcertkey` エンティティ名を作成します。

安全でないトラフィックをリダイレクトするためのサービスアノテーションを含むYAMLの例

この例では、安全でないポート80でリクエストを行うクライアントからのトラフィックを、安全なポート443にリダイレクトする方法を示します。

トラフィックをリダイレクトするために、サービスYAMLファイルに次のアノテーションが指定されています。

```
service.citrix.com/insecure-redirect: '{"port-443": 80}'
```

サービス定義の例:

```
kubectl apply -f - <<EOF

apiVersion: v1

kind: Service

metadata:

  name: frontend-service

  annotations:

    service.citrix.com/service-type-0: SSL

    service.citrix.com/frontend-ip: '192.2.170.26'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    service.citrix.com/secret: '{"port-443": "web-ingress-secret"}'
    service.citrix.com/ssl-termination-0: 'EDGE'
    service.citrix.com/insecure-redirect: '{"port-443": 80}'
spec:
  type: LoadBalancer
  selector:
    app: frontend
  ports:
  - port: 443
    targetPort: 80
    name: port-443
EOF
```

サービス用スマートアノテーション

サービス用スマートアノテーションは、NetScaler構成パラメーターのカスタム値を使用してNetScalerを構成するために使用されます。これらのアノテーションは、タイプ`LoadBalancer`のサービス、およびEast-Westトラフィックに使用されるNetScaler CPXのサービスに使用されます。

注:

North-Southトラフィック用にNodePortまたはClusterIPでサービスを構成している場合、NetScalerはサービスアノテーションではなく、該当するイングレススマートアノテーションを使用して構成されます。

サービス用スマートアノテーションは、JSON形式を入力として受け取ります。JSON形式で渡すキーと値は、NetScaler NITRO形式と一致する必要があります。NetScaler NITRO APIの詳細については、NetScaler REST APIs - NITRO Documentationを参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サービス用スマートアノテーションの例:

```
service.citrix.com/lbvserver: '{"80-tcp":{"lbmethod":"SRCIPDESTIPHASH"}}'
```

このアノテーションは、指定されたサービスの`80-tcp`ポートに対するロードバランシング仮想サーバーで、ロードバランシング方式を`SRCIPDESTIPHASH`として設定します。

以下の表に、サービス用のスマートアノテーションについて説明します。

NETSCALERエンティティ名	サービス用スマートアノテーション	例
エルビーブイサーバー	`service.citrix.com/lbvserver`	`service.citrix.com/lbvserver: '{"80-tcp":{"lbmethod":"SRCIPDESTIPHASH"}}'`
シーエスブイサーバー	`service.citrix.com/csvserver`	`service.citrix.com/csvserver: '{"l2conn":"on"}'`
サービスグループ	`service.citrix.com/servicegroup`	`service.citrix.com/servicegroup: '{"80-tcp":{"usip":"yes"}}'`
モニター	`service.citrix.com/monitor`	`service.citrix.com/monitor: '{"80-tcp":{"type":"http"}}'`, `service.citrix.com/monitor: '{"80-tcp": "<pre-configured-monitor1-on-NetScaler>", "443-tcp": "<pre-configured-monitor2-on-NetScaler>"}'`
アナリティクスプロファイル	`service.citrix.com/analyticsprofile`	`service.citrix.com/analyticsprofile: '{"80-tcp":{"webinsight": {"httpurl":"ENABLED",`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NETSCALERエンティティ名	サービス用スマートアノテーション	例
		`"httpuseragent":"ENABLED"}}}'`

サービスにスマートアノテーションを次のように使用できます。

- アノテーションに`port-protocol`の値を指定する場合: サービス定義で、アノテーションに`port-protocol`の値を指定すると、そのアノテーションはそのサービスの特定のポートに制限されます。
- アノテーションに`port-protocol`の値を指定しない場合: アノテーションに`port-protocol`の値を指定しない場合、そのアノテーションはサービスで使用されるすべてのポートに適用されます。

サービス用のスマートアノテーションを含むIngress YAMLの例

次のYAMLは、基本的なApache Webサーバーベースのアプリケーションのデプロイメントとサービス定義の例です。これには、lbvserver、csvserver、servicegroup、monitor、analyticsprofileなどのエンティティを使用してNetScaler機能を有効にするためのサービス用のスマートアノテーションが含まれています。

```
kubectl apply -f - <<EOF

apiVersion: apps/v1

kind: Deployment

metadata:

  name: apache

  labels:

      name: apache

spec:

  selector:

    matchLabels:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      app: apache
  replicas: 8
  template:
    metadata:
      labels:
        app: apache
    spec:
      containers:
      - name: apache
        image: httpd:latest
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent


---
#Expose the apache web server as a service
apiVersion: apps/v1
kind: Service
metadata:
  name: apache
  annotations:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    service.citrix.com/csvserver: '{"l2conn":"on"}'

    service.citrix.com/lbvserver: '{"80-tcp":
{"lbmethod":"SRCIPDESTIPHASH"}}'

    service.citrix.com/servicegroup: '{"80-tcp":{"usip":"yes"}}'

    service.citrix.com/monitor: '{"80-tcp":{"type":"http"}}'

    service.citrix.com/frontend-ip: '10.217.212.16'

    service.citrix.com/analyticsprofile: '{"80-tcp":{"webinsight":
{"httpurl":"ENABLED", "httpuseragent":"ENABLED"}}}'

    NETSCALER_VPORT: '80'

  labels:

    name: apache

spec:

  externalTrafficPolicy: Local

  type: LoadBalancer

  selector:

    name: apache

  ports:

  - name: http

    port: 80

    targetPort: http

  selector:

    app: apache

---
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
EOF
```

ServicetypeLB: スマートアノテーションのイベント変更

NetScaler Ingress Controllerリリース4.0.16以降、`ServiceTypeLB`で次のいずれかのアノテーションを変更すると、NetScaler Ingress ControllerはNetScalerで構成を削除して再作成するのではなく、変更します。

```
"service.citrix.com/lbvserver",

"service.citrix.com/csvserver",

"service.citrix.com/servicegroup",

"service.citrix.com/monitor",

"service.citrix.com/analyticsprofile",

"service.citrix.com/insecure-redirect",

"service.citrix.com/secret",

"service.citrix.com/preconfigured-certkey",

"service.citrix.com/ca-secret",

"service.citrix.com/preconfigured-ca-certkey",

"service.citrix.com/backend-secret",

"service.citrix.com/preconfigured-backend-certkey",

"service.citrix.com/backend-ca-secret",

"service.citrix.com/preconfigured-backend-ca-certkey",

'service.citrix.com/ssl-termination-<index>',

'service.citrix.com/frontend-tcpprofile-<index>',

'service.citrix.com/backend-tcpprofile-<index>',
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
'service.citrix.com/frontend-httpprofile-<index>',

'service.citrix.com/backend-httpprofile-<index>',

'service.citrix.com/frontend-sslprofile-<index>',

'service.citrix.com/backend-sslprofile-<index>'
```

例

不安全なサービスタイプアノテーションにおけるSIP_UDPサポートのIngress YAMLの例

次のIngress YAMLの例には、`ingress.citrix.com/insecure-service-type`アノテーションを使用してUDP経由のSIPサポートを有効にするための構成が含まれています。

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  annotations:

    ingress.citrix.com/frontend-ip: 1.1.1.1

    ingress.citrix.com/insecure-port: "5060"

    ingress.citrix.com/insecure-service-type: sip_udp

    ingress.citrix.com/lbvserver: '{"asterisk17":
{"lbmethod":"CALLIDHASH","persistenceType":"CALLID"}}'

  name: sip-ingress

spec:

  defaultBackend:

    service:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      name: asterisk17

      port:

        number: 5060

EOF
```

アノテーションキー `ingress.citrix.com/default-response-code`

NetScaler Ingress Controller は、アノテーションに対して有効な JSON を処理します。

```
ingress.citrix.com/default-response-code: '{"type": "<action-type>",
"response-code": "<code>", "target": "<target>", "reason-phrase":
"<text>"}'
```

キー	必須	説明
`type`	いいえ — デフォルトの `respondwith` は、キーが存在しない場合にのみ適用されます	レスポンダーアクションタイプ。小文字の文字列 `respondwith`、`respondwithhtmlpage`、`redirect` のいずれかである必要があります。マッチングは大文字と小文字を区別し、トリミングされません。`Redirect`、`RESPONDWITH`、`"redirect "`、`""`、`null` はすべて無効であり、アノテーション全体が拒否されます。`respondwith` へのフォールバックはありません。キーが空の文字列としてレンダリングされる可能性があるテンプレートに注意してください。
`response-code`	はい、3種類すべてに適用されます	最初の桁が1〜5の正確に3桁のASCII数字である場合のみ受け入れられます（実際には100〜599）。JSON数値は引

© 1997–2026 Citrix Systems, Inc. All rights reserved.

キー	必須	説明
		用符で囲まれた文字列と同様に受け入れられます（`{"response-code": 404}` は機能します）。`99`、`0404`、`600`、`999`、浮動小数点数、ブール値は拒否されます。チェックは構文のみです。
`target`	`respondwithhtmlpage` と `redirect` の場合は必須。`respondwith` の場合はオプション	`respondwithhtmlpage`: NetScaler がページをインポートするソース URL。`redirect`: `Location` ヘッダーで送信される URL。どちらも有効な絶対 http/https URL である必要があります。`respondwith` の場合、これはオプションの生の NetScaler 高度ポリシー式であり、応答ペイロード全体になります。その内容は意図的に検証されませんが、JSON 文字列である必要があります。数値、オブジェクト、または配列はアノテーション全体を拒否します。
`reason-phrase`	いいえ	NITRO `-reasonPhrase` フィールドとして `respondwithhtmlpage` に送信されます。`target` が省略されている場合にのみ `respondwith` で尊重され、生成されたペイロードに補間されます。`redirect` ではサイレントに無視されます。二重引用符、バックスラッシュ、CR、LFは削除され、結果はトリミングされます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

キー	必須	説明
		文字列以外の値は空として扱われます。

respondwith (デフォルト)

- `response-code`はステータスフィールドとしてNetScalerに送信されません。これは手動で構築されたペイロード式に補間され、ステータス行はそのペイロード内に存在します。
- `target`がない場合: `reason-phrase`も存在しない場合、`404`と`503`は正確な組み込みメッセージを保持します。その他のコード、または`reason-phrase`を伴う404/503は、生成されたボディ`HTTP/1.1 <code> <reason>\r\n\r\nRequest to <host><url> could not be served`を生成します。— ここで、`<reason>`はサニタイズされた`reason-phrase`、または何も指定されていない場合はリテラルワード`Error`です。
- カスタム`target`を使用する場合: あなたの式はそのまま使用され、NetScaler Ingress Controllerはそれを引用符で囲んだりエスケープしたりしません。その場合、`response-code`と`reason-phrase`は応答に全く影響を与えません。実際のステータス行を式に入れてください。NetScaler Ingress Controllerは不一致に関する警告をログに記録しません。`"Service down"`のような裸の文字列は有効な高度なポリシー式ではありません。NetScalerは`add responder action`を拒否し、デフォルトのLB全体が破棄されます。
- このタイプでは、`-responseStatusCode`も`-reasonPhrase`も送信されません。

リダイレクト

- `target`は式ではなく、リテラルの引用符付き文字列として出力されます。元のリクエストのホスト、パス、クエリは追加または保持されません。デフォルトのLB仮想サーバーに到達するすべてのリクエストは、正確にその1つのURLに送信されます。このアノテーションは元のURIを引き継ぐことはできません。
- `response-code`は`-responseStatusCode`として送信されます。NetScaler Ingress Controllerはそれが3xxであるかどうかをチェックしないため、`{"type":"redirect","response-code":"404"}`は受け入れられ、構成されますが、誰もリダイレクトされません。
- `reason-phrase`は無視されます。

レスポンドウィズHTMLページ

- NetScaler Ingress Controllerは`import responder htmlpage <page-name> <target> -overwrite`を発行し、レスポンダーアクションをインポートされたページオブジェクトを名

© 1997–2026 Citrix Systems, Inc. All rights reserved.

前(`-htmlpage <page-name>`)で指定し、URLでは指定しません。あなたのURLは、アプライアンス上ではインポートされたオブジェクトの `src` としてのみ表示されます。

- `-responseStatusCode` と `-reasonPhrase` の両方が送信されます。
- ページホストはアプライアンスから到達可能である必要があります。

例とターゲットURLルール

a. `respondwith` — ステータスコードとNetScaler Ingress Controllerによって生成された本文。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: defaultlb-respondwith

  annotations:

    kubernetes.io/ingress.class: citrix

    ingress.citrix.com/frontend-ip: 5.5.5.5

    ingress.citrix.com/default-response-code: '{"response-code": "404"}'

spec:

  rules:

  - host: www.example.com

    http:

      paths:

      - pathType: Prefix

        path: /

        backend:

          service:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            name: frontend

            port:

              number: 80
```

b. `respondwithhtmlpage` — 独自のHTMLページを提供します（同じIngress、異なるアノテーション値）。

```
ingress.citrix.com/default-response-code: '{"type": "respondwithhtmlpage",
"response-code": "521", "target": "https://status.example.com/service-
unavailable.html", "reason-phrase": "Web server down"}'
```

c. `redirect` — クライアントを別の場所に送信します。

```
ingress.citrix.com/default-response-code: '{"type": "redirect", "response-
code": "302", "target": "https://status.example.com/service-
unavailable.html"}'
```

注

- `path: /` と `pathType: Prefix` を使用すると、`www.example.com` へのすべてのリクエストがルールに一致するため、同じフロントエンドIP上の異なるホストへのリクエスト、またはフロントエンドサービスに準備完了のエンドポイントがない場合の `www.example.com` へのリクエストによって、デフォルトの応答が実行されます。
- `target` URLの受け入れ/拒否ルール（`redirect` と `respondwithhtmlpage` のみ）。アノテーションが解析される際に適用され、NetScalerに何も送信される前に行われます。

ルール	受け入れられる例	拒否される例
httpまたはhttpsスキームを持つ絶対パスである必要があります	`https://example.com/p.html`, `HTTP://EXAMPLE.COM/P` (スキームは大文字と小文字を区別しません)	`/service-unavailable.html`, `p.html`, `example.com/x`, `//evil.com/x`, `ftp://a.com/p`, `javascript:alert(1)`,

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ルール	受け入れられる例	拒否される例
		`data:text/html,x`, `file:///etc/passwd`
ホストが必要です。ポートが存在する場合、有効な数値である必要があります	`https://h.example.com:8443/a/b?q=1&r=2`, `http://192.0.2.10/maintenance`, `https://[2001:db8::1]:8443/p.html`	`http://`, `http:///p.html`, `http://:8080/p.html`, `http://host:notaport/`, `http://host:99999/`, `http://host:-1/`
どのような形式であっても、資格情報 (ユーザー情報) を埋め込むことはできません。	パスまたはクエリ内の `@` は問題ありません: `http://example.com/path@foo`, `https://example.com/a?b=1@2`	`http://user:pass@host/x`, `https://user@host/x`, `http://@host/`, `https://:pass@host/`, `https://user:@host/`
二重引用符、バックスラッシュ、C0制御文字、DEL、または空白文字を含めてはなりません。代わりにパーセントエンコードを使用してください。	`https://example.com/p%20q`	`https://example.com/p q`, 二重引用符、バックスラッシュ、TAB、CR、またはLFを含むURL
空でない文字列である必要があります。	—	`""`, JSON `null`, JSONの数値/オブジェクト/配列

NetScaler Ingress Controller の ConfigMap サポート

ConfigMap API リソースは、ポッドで利用できる、またはコントローラーなどのシステムコンポーネントの構成データを保存できる、構成データのキーと値のペアを保持します。

ConfigMap を使用すると、構成をポッドから分離し、ワークロードをポータブルにすることができます。ConfigMap を使用すると、ワークロードの構成を簡単に変更および管理でき、構成データをポッド仕様にハードコーディングする必要性を減らすことができます。

NetScaler Ingress Controller は、NetScaler Ingress Controller の展開 で言及されている構成コマンドライン引数と環境変数をサポートしています。ただし、NetScaler Ingress Controller ポッドを再起動せずに、これらの構成をランタイムで更新することはできません。ConfigMap のサポート

© 1997–2026 Citrix Systems, Inc. All rights reserved.

により、NetScaler Ingress Controller ポッドを実行したまま、構成を自動的に更新できます。更新後にポッドを再起動する必要はありません。

NetScaler Ingress Controller でサポートされている環境変数

NetScaler Ingress Controller の以下の環境変数の値は、ConfigMap で指定できます。

- LOGLEVEL: NetScaler Ingress Controller によって生成されるログを制御するためのログレベル (debug、info、critical など) を指定します。デフォルト値は `debug` です。
- NS_HTTP2_SERVER_SIDE: NetScaler サービスグループ構成で HTTP2 を有効にします。指定可能な値は ON または OFF です。
- NS_PROTOCOL: ADC セッションを確立するためのプロトコル (HTTP/HTTPS) を指定します。デフォルト値は `http` です。
- NS_PORT: セッションを確立するためのポートを指定します。デフォルト値は `80` です。
- NS_COOKIE_VERSION: 永続性クッキーのバージョン (0 または 1) を指定します。デフォルト値は `0` です。
- NS_DNS_NAMESERVER: NetScaler VPX に DNS ネームサーバーを追加できるようにします。
- POD_IPS_FOR_SERVICEGROUP_MEMBERS: 外部のティア 1 NetScaler で `LoadBalancer` または `NodePort` タイプのサービスを構成する際に、`NodeIP` および `NodePort` の代わりにポッドの IP アドレスとポートをサービスグループメンバーとして追加するように指定します。
- IGNORE_NODE_EXTERNAL_IP: 外部のティア 1 NetScaler で `LoadBalancer` または `NodePort` タイプのサービスの NodeIP を構成する際に、外部 IP アドレスを無視し、NodeIP に内部 IP アドレスを追加するように指定します。
- FRONTEND_HTTP_PROFILE: イングレス定義の `ingress.citrix.com/frontend-httpprofile` スマートアノテーションによって上書きされない限り、フロントエンド仮想サーバー (クライアントプレーン) の HTTP オプションを設定します。
- FRONTEND_TCP_PROFILE: フロントエンド仮想サーバー（クライアント側）のTCPオプションを設定します。ただし、イングレス定義の`ingress.citrix.com/frontend-tcpprofile`スマートアノテーションによって上書きされない限りです。
- FRONTEND_SSL_PROFILE: フロントエンド仮想サーバー（クライアント側）のSSLオプションを設定します。ただし、イングレス定義の`ingress.citrix.com/frontend-sslprofile`スマートアノテーションによって上書きされない限りです。
- JSONLOG: ログメッセージがJSON形式で必要な場合は、この引数をtrueに設定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- NS_ADNS_IPS: NetScalerをADNSサーバーとして構成できるようにします。

プロファイル環境変数（FRONTEND_HTTP_PROFILE、FRONTEND_TCP_PROFILE、およびFRONTEND_SSL_PROFILE）に関する詳細については、NetScalerでHTTP、TCP、またはSSLプロファイルを構成するを参照してください。

注:

これはConfigMapサポートの初期バージョンであり、現在、いくつかのパラメーターのみをサポートしています。以前は、これらのパラメーターは`NS_HTTP2_SERVER_SIDE`パラメーターを除いて環境変数を通じて構成可能でした。

NetScaler Ingress ControllerのConfigMapサポートの構成

この例では、ConfigMapを作成し、そのConfigMapをNetScaler Ingress Controllerに適用する方法を示します。また、変更を加えた後にConfigMapを再適用する方法も示します。変更をオプションで削除することもできます。

NetScaler Ingress ControllerのConfigMapサポートを構成するには、以下を実行します。

1. ConfigMapに必要なキーと値のペアを含むYAMLファイル`cic-configmap.yaml`を作成します。

```
apiVersion: v1

kind: ConfigMap

metadata:

  name: cic-configmap

  labels:

    app: citrix-ingress-controller

data:

  LOGLEVEL: 'info'

  NS_PROTOCOL: 'http'

  NS_PORT: '80'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  NS_COOKIE_VERSION: '0'

  NS_HTTP2_SERVER_SIDE: 'ON'
```

2. 次のコマンドを使用して`cic-configmap.yaml`をデプロイします。

```
kubectl create -f cic-configmap.yaml
```

3. NetScaler Ingress Controllerをスタンドアロンポッドとしてデプロイするために`cic.yaml`ファイルを編集し、以下を指定します。

```
Args:

    - --configmap

        default/cic-configmap
```

注:

名前空間の指定は必須です。名前空間が指定されていない場合、ConfigMapは考慮されません。

ConfigMap構成でNetScaler Ingress Controllerをデプロイするための YAMLファイルのサンプルを以下に示します。完全なYAMLファイルについては、citrix-k8s-ingress-controller.yamlを参照してください。

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: cic-k8s-ingress-controller

spec:

  selector:

    matchLabels:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      app: cic-k8s-ingress-controller
    replicas: 1
    template:
      metadata:
        name: cic-k8s-ingress-controller
        labels:
          app: cic-k8s-ingress-controller
      annotations:
      spec:
        serviceAccountName: cic-k8s-role
        containers:
        - name: cic-k8s-ingress-controller
          image: "quay.io/citrix/citrix-k8s-ingress-controller:1.36.5"
          env:
          # Set NetScaler NSIP/SNIP, SNIP in case of HA (mgmt has to be
enabled)
          - name: "NS_IP"
            value: "x.x.x.x"
          - name: "EULA"
            value: "yes"
          args:
            - --ingress-classes
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            citrix
          - --feature-node-watch
            false
          - --configmap
            default/cic-configmap
        imagePullPolicy: Always
```

4. YAMLを適用して、NetScaler Ingress Controllerをスタンドアロンポッドとしてデプロイします。

```
kubectl apply -f cic.yaml
```

5. 環境変数の値を変更したい場合は、ConfigMapの値を編集します。この例では、NS_HTTP2_SERVER_SIDEの値が「OFF」に変更されています。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cic-configmap
  labels:
    app: citrix-ingress-controller
data:
  LOGLEVEL: 'info'
  NS_PROTOCOL: 'http'
  NS_PORT: '80'
  NS_COOKIE_VERSION: '0'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  NS_HTTP2_SERVER_SIDE: 'OFF'
```

6. 次のコマンドを使用してConfigMapを再適用します。

```
kubectl apply -f cic-configmap.yaml
```

7. (オプション) ConfigMapを削除する必要がある場合は、次のコマンドを使用します。

```
kubectl delete -f cic-configmap.yaml
```

   ConfigMapを削除すると、環境変数構成は次の優先順位に従ってフォールバックします。
   ConfigMap構成 > 環境変数構成 > デフォルト

(オプション) ConfigMap内のすべてのキーをNetScaler Ingress Controllerの環境変数として定義したい場合は、NetScaler Ingress ControllerのデプロイYAMLファイルで以下を使用します。

```
envFrom:

        - configMapRef:

          name: cic-configmap
```

Ingress の構成

Kubernetes の Ingress は、リクエストのホストまたはパスに基づいてサービスにリクエストをルーティングする方法を提供し、多数のサービスを単一のエントリポイントに一元化します。

NetScaler Ingress Controller は Kubernetes Ingress を中心に構築されており、Ingress リソース構成に基づいて 1 つ以上の NetScaler を自動的に構成します。

ホスト名ベースのルーティング

次の Ingress 定義の例は、ホスト名に基づいてトラフィックをルーティングするように Ingress を設定する方法を示しています。

```
apiVersion: networking.k8s.io/v1

kind: Ingress
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
metadata:
  name: virtual-host-ingress
  namespace: default
spec:
  rules:
  - host: foo.bar.com
    http:
      paths:
      - backend:
          service:
            name: service1
            port:
              number: 80
        pathType: Prefix
        path: /
  - host: bar.foo.com
    http:
      paths:
      - backend:
          service:
            name: service2
            port:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
              number: 80
        pathType: Prefix
        path: /
```

Ingress 定義の例がデプロイされると、ホストヘッダーを持つすべての HTTP リクエストは NetScaler によって `service1` にロードバランスされます。また、ホストヘッダーを持つ HTTP リクエストは NetScaler によって `service2` にロードバランサーされます。

パスベースのルーティング

次の Ingress 定義の例は、URL パスに基づいてトラフィックをルーティングするように Ingress を設定する方法を示しています。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: path-ingress
  namespace: default
spec:
  rules:
  - host: test.example.com
    http:
      paths:
      - backend:
          service:
            name: service1
            port:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
              number: 80
        path: /foo
        pathType: Prefix
      - backend:
          service:
            name: service2
            port:
              number: 80
        path: /
        pathType: Prefix
```

Ingress 定義の例がデプロイされると、ホスト `test.example.com` と URL パスにプレフィックス `/foo` を持つすべての HTTP リクエストは NetScaler によって `service1` にルーティングされ、その他のすべてのリクエストは `service2` にルーティングされます。

NetScaler Ingress Controller は、パスを評価するために最初の一致ポリシーに従います。効果的なマッチングのために、NetScaler Ingress Controller はパスの長さの降順に基づいてパスを順序付けします。また、複数の Ingress リソースに属する同じホストのパスも順序付けします。

ワイルドカードホストルーティング

次の Ingress 定義の例は、ワイルドカードホストを持つ Ingress を設定する方法を示しています。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: wildcard-ingress
  namespace: default
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:

  rules:

  - host: '*.example.com'

    http:

      paths:

      - backend:

          service:

            name: service1

            port:

              number: 80

        path: /

        pathType: Prefix
```

Ingress 定義の例がデプロイされると、`example.com` のすべてのサブドメインへの HTTP リクエストは NetScaler によって `service1` にルーティングされます。

注記：

ワイルドカードホストではないルールは、ワイルドカードホストよりも高い優先度が与えられます。異なるワイルドカードホスト間では、ルールはホストの長さの降順で順序付けされます。

正確なパスの一致

`networking.k8s.io/v1` APIバージョンに属するIngressは、`PathType: Exact` を使用してパスを正確な一致として考慮できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: Path-exact-Ingress

  namespace: default

spec:

  rules:

  - host: test.example.com

    http:

      paths:

      - backend:

          service:

            name: service1

            port:

              name: 80

        path: /exact

        pathType: Exact
```

(Kubernetes 1.22以降では非推奨) デフォルトでは、`extension/v1beta1` に属するIngressの場合、パスは `Prefix` 式として扱われます。Ingress定義でアノテーション `ingress.citrix.com/path-match-method: "exact"` を使用すると、NetScaler Ingress Controllerはパスを正確な一致として考慮するように定義されます。

以下のIngress定義のサンプルは、正確なパスの一致のためにIngressを設定する方法を示しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: extension/v1beta1

kind: Ingress

metadata:

  name: path-exact-ingress

  namespace: default

  annotations:

    ingress.citrix.com/path-match-method: "exact"

spec:

  rules:

  - host:test.example.com

    http:

      paths:

      - path: /exact

        backend:

          serviceName: service1

          servicePort: 80
```

サンプルIngress定義がデプロイされると、パス `/exact` を持つHTTPリクエストはNetScalerによって `service1` にルーティングされますが、`/exact/somepath` にはルーティングされません。

非ホスト名ルーティング

以下の例は、ホストベースのルートと一致しないデフォルトトラフィックのパスベースルーティングを示しています。このIngressルールは、指定されたIPアドレスを介したすべてのインバウンドHTTPトラフィックに適用されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: default-path-ingress
  namespace: default
spec:
  rules:
  - http:
      paths:
      - backend:
          service:
            name: service1
            port:
              number: 80
        path: /foo
        pathType: Prefix
      - backend:
          service:
            name: service2
            port:
              number: 80
        path: /
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        pathType: Prefix
```

ホスト名を持つIngressルールと一致しないすべての受信トラフィックは、ルーティングのためのパスでここで一致します。

デフォルトのバックエンド

デフォルトのバックエンドは、どのIngressルールとも一致しないすべてのトラフィックを処理するサービスです。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: default-ingress
  namespace: default
spec:
  defaultBackend:
    service:
      name: testsvc
      port:
        number: 80
```

注：

NetScaler CPXがトラフィックをロードバランシングしている場合、グローバルなデフォルトバックエンドを指定できます。NetScaler VPXまたはMPXがIngressデバイスである場合、`frontend-ip:port` の組み合わせごとにデフォルトバックエンドを作成できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Ingressクラスのサポート

Ingressクラスとは?

Kubernetesクラスターには複数のIngressコントローラーが存在する可能性があり、特定のIngressリソースをIngressコントローラーに関連付ける方法が必要です。

Ingressリソース定義で`ingressClassName`を使用することで、Ingressリソースを処理するIngressコントローラーを指定できます。

ネットスケーラー イングレス コントローラー と イングレス クラス

NetScaler Ingress Controllerは、`spec.ingressClassName`を持つ複数のIngressリソースを受け入れることをサポートしています。各Ingressリソースは1つの`ingress.class`のみに関連付けることができます。ただし、Ingress Controllerは異なるクラスからの様々なIngressリソースを処理する必要がある場合があります。

YAMLファイルの`spec`セクションにある`--ingress-classes`引数を使用して、Ingress Controllerを複数のIngressクラスに関連付けることができます。

Ingress Controllerに対して`ingress-classes`が指定されていない場合、Ingressオブジェクトに`ingressClassName`が存在するかどうかにかかわらず、すべてのIngressリソースを受け入れます。

`ingress-classes`が指定されている場合、Ingress Controllerは`ingressClassName`に一致するIngressリソースのみを受け入れます。この場合、Ingressコントローラーは`ingress.class`のないIngressリソースを処理しません。

注: Ingressクラス名は大文字と小文字を区別しません。

Ingressクラスを含むYAML設定の例

以下は、`ingress-classes`をIngress Controllerに関連付けるためのサンプルYAMLファイルからの抜粋です。この設定は、Ingress Controllerがスタンドアロンポッドとして実行される場合と、NetScaler CPXのサイドカーとして実行される場合の両方で機能します。指定されたYAMLスニペットでは、以下のIngressクラスがIngress Controllerに関連付けられています。

- `my-custom-class`
- `Citrix`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:

    serviceAccountName: cic-k8s-role

    containers:

    - name: cic-k8s-ingress-controller

      image:"quay.io/citrix/citrix-k8s-ingress-controller:latest"

      # specify the ingress classes names to be supportedbyIngress
Controller in args section.

      # First line should be --ingress-classes, andeverysubsequent line
should be

      # the name of allowed ingress class. In the givenexampletwo classes
named

      # "citrix" and "my-custom-class" are accepted. Thiswill be case-
insensitive.

      args:

        - --ingress-classes

          Citrix

          my-custom-class
```

以下は、Ingressクラスの関連付けが示されているIngress YAMLファイルからの抜粋です。指定された例では、`web-ingress`という名前のIngressリソースがIngressクラス`my-custom-class`に関連付けられています。NetScaler Ingress Controllerが`my-custom-class`を受け入れるように構成されている場合、このIngressリソースを処理します。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  annotations:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: web-ingress

spec:

  ingressClassName: my-custom-class
```

Ingress V1とIngressClassのサポート

Kubernetesバージョン1.19では、Ingressリソースが一般提供されるようになりました。 この変更の一環として、`IngressClass`という名前の新しいリソースがIngress APIに追加されました。このリソースを使用すると、特定のIngressコントローラーをIngressに関連付けることができます。`IngressClass`リソースの詳細については、Kubernetesドキュメントを参照してください。

以下は、`IngressClass`リソースのサンプルです。

```


apiVersion: networking.k8s.io/v1

kind: IngressClass

metadata:

  name: citrix

spec:

  controller: citrix.com/ingress-controller
```

`IngressClass`リソースは、以下に示すようにIngressルールを実装するコントローラーに関連付けられたIngressクラスを参照する必要があります。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: minimal-ingress
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:

  ingressClassName: citrix

  rules:

  - host: abc.com

    http:

      paths:

      - path: /

        pathType: Prefix

        backend:

          service:

            name: test

            port:

              number: 80
```

NetScaler Ingress Controllerは、Ingressを照合するために以下のルールを使用します。

- NetScaler Ingress Controllerが`--ingress-classes`引数を指定せずに起動された場合：
  - Kubernetesのバージョンが1.19未満の場合（IngressClass V1リソースがサポートされています）
    - 任意のIngressオブジェクトに一致します
  - Kubernetesのバージョンが1.19以上の場合（IngressClass V1リソースがサポートされています）
    - `spec.ingressClassName`フィールドが設定されていない任意のIngressオブジェクトに一致します。
    - Ingressオブジェクトの`spec.ingressClassName`フィールドが設定されており、同じ名前の`v1.IngressClass`リソースが存在し、そのリソースの`spec.controller`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールドが`citrix.com/ingress-controller`である場合、任意のIngressに一致します。

- NetScaler Ingress Controllerが`--ingress-classes`引数を使用して1つ以上のIngressクラスを設定して起動された場合。
  - Kubernetesのバージョンが1.19未満の場合（IngressClass V1リソースがサポートされています）
    - 設定されたIngressクラスと一致するIngressクラスアノテーション`kubernetes.io/ingress.class`を持つ任意のIngressに一致します。
  - Kubernetesのバージョンが1.19以上の場合（IngressClass V1リソースがサポートされています）。
    - 設定されたイングレスクラスとイングレスクラスアノテーション`kubernetes.io/ingress.class`が一致するすべてのイングレスに一致します。このアノテーションは非推奨ですが、下位互換性をサポートするために`spec.IngressClassName`フィールドよりも優先度が高くなります。
    - 次の属性を持つ`v1.IngressClass`リソースが存在する場合、任意のイングレスオブジェクトに一致します。
      - リソースの名前が`--ingress-classes`引数の値と一致します。
      - リソースの`spec.controller`フィールドが`citrix.com/ingress-controller`として設定されています。
      - リソースの名前がIngressオブジェクトの`spec.ingressClassName`フィールドと一致します。
    - `spec.ingressClassName`フィールドが設定されておらず、かつ次の属性を持つ`v1.IngressClass`リソースが存在する場合、任意のイングレスオブジェクトに一致します。
      - リソースの名前が`--ingress-classes`引数の値と一致します。
      - リソースの`spec.controller`フィールドが`citrix.com/ingress-controller`として設定されています。
      - リソースは`ingressclass.kubernetes.io/is-default-class`アノテーションを使用してデフォルトクラスとして構成されます。詳細については、Kubernetes ドキュメント を参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

- アノテーションと `spec.ingressClassName` の両方が定義されている場合、アノテーションは `spec.ingressClassName` よりも前に照合されます。アノテーションが一致しない場合、`spec.ingressClassName` フィールドの照合操作は実行されません。
- Helm チャートを使用して NetScaler Ingress Controller をインストールする場合、`IngressClass` リソースがサポートされており、NetScaler Ingress Controller が `--ingress-classes` 引数で展開されている場合、`v1.IngressClass` リソースはデフォルトで作成されます。

指定されたIPアドレスを持つIngressリソースのIngressステータスを更新する

NetScaler Ingress Controller によって管理されるIngressリソースの `Status.LoadBalancer.Ingress` フィールドを割り当てられたIPアドレスで更新するには、NetScaler Ingress Controller の起動時にコマンドライン引数 `--update-ingress-status yes` を指定します。

注:

Helmデプロイの場合、`update-ingress-status` に対応するHelmチャートのパラメータは `updateIngressStatus` であり、`true` に設定する必要があります。

この機能は、スタンドアロンおよびサイドカーNSICデプロイメントの両方でサポートされています。サイドカーとしてデプロイされたNSICの場合、この機能はNSICバージョン2.2.10以降でサポートされます。

サイドカーデプロイメントのIngressステータス更新

Kubernetesでは、Ingressは複数のアプリケーションを外部に公開するための単一のエントリポイントとして使用できます。Ingressには、Ingressの作成が成功した後に更新される `Address` (`Status.LoadBalancer.IP`) フィールドがあります。このフィールドは、Kubernetesアプリケーションに到達できるパブリックIPアドレスまたはホスト名で更新されます。クラウドデプロイメントでは、このフィールドはクラウドロードバランサーのIPアドレスまたはホスト名であることもあります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

クラウドデプロイメントでは、NetScaler CPXはIngressコントローラーとともに、`type LoadBalancer` タイプのサービスを使用して公開され、これによりクラウドロードバランサーが作成されます。クラウドロードバランサーは、NetScaler CPXをIngressコントローラーとともに公開します。したがって、NetScaler CPXで公開されるIngressリソースは、クラウドロードバランサーのパブリックIPアドレスまたはホスト名を使用して更新する必要があります。

この更新はオンプレミスデプロイメントにも適用されます。

- デュアルティアIngressデプロイメントにおいて、NetScaler CPXがタイプ `LoadBalancer` のサービスを使用してティア1 NetScaler VPX Ingressに公開されている場合、NetScaler CPXによって操作されるIngressリソースの `Address` (`Status.LoadBalancer.IP`) フィールドはVIPアドレスで更新されます。
- デュアルティアIngressデプロイメントにおいて、NetScaler CPXがタイプ `ClusterIP` または `NodePort` のサービスを使用して公開されている場合、NetScaler CPXによって操作されるIngressリソースの `Address` (`Status.LoadBalancer.IP`) フィールドは、CPXサービスのクラスターIPアドレスで更新されます。

Ingressステータス更新後のIngress出力例

以下は、Ingressステータス更新後のIngress出力例です。

```
    $ kubectl get ingress



    NAME             HOSTS              ADDRESS
PORTS    AGE

    sample-ingress   sample.citrix.com   sample.abc.somexampledomain.com
80       1d
```

LoadBalancer タイプのサービス用サービス クラス

LoadBalancer タイプのサービスがデプロイされると、そのようなすべてのサービスは NetScaler Ingress Controller によって処理され、NetScaler 上で構成されます。ただし、複数の Ingress Controller がデプロイされている場合に、特定のサービスのみを NetScaler Ingress Controller に関連付けたい状況があるかもしれません。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Ingress リソースの場合、この機能は Ingress クラス 機能を使用してすでに利用可能です。Ingress リソースの Ingress クラス機能と同様に、`LoadBalancer` タイプのサービスに対してサービス クラス機能が追加されました。

YAML ファイルの `spec` セクションにある `--service-classes` 引数を使用して、NetScaler Ingress Controller を複数のサービス クラスに関連付けることができます。Ingress Controller にサービス クラスが指定されていない場合、サービス内の `service.citrix.com/class` アノテーションの有無にかかわらず、`LoadBalancer` タイプのすべてのサービスを受け入れます。

NetScaler Ingress Controller にサービス クラスが指定されている場合、`service.citrix.com/class` アノテーションに一致する `LoadBalancer` タイプのサービスのみを受け入れます。この場合、NetScaler Ingress Controller は、`service.citrix.com/class` アノテーションに関連付けられていない `LoadBalancer` タイプのサービスを処理しません。

サービス クラスを含む YAML 設定の例

以下は、`service-classes` を Ingress Controller に関連付けるためのサンプル YAML ファイルからの抜粋です。この抜粋では、以下のサービス クラスが Ingress Controller に関連付けられています。

- `svc-class1`
- `svc-class2`

```
spec:

serviceAccountName: cic-k8s-role

containers:

- name: cic-k8s-ingress-controller

  # specify the service classes to be supported by NetScaler Ingress
Controller in args section.

  # First line should be --service-classes, and every subsequent line
should be

  # the name of allowed service class. In the given example two classes
named

  # "svc-class1" and "svc-class2" are accepted. This will be case-
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
insensitive.

  args:

    - --service-classes

      svc-class1

      svc-class2
```

以下は、サービス クラスの関連付けが示されている LoadBalancer タイプのサービス定義 YAML ファイルからの抜粋です。この例では、Apache サービスがサービス クラス `svc-class1` に関連付けられています。NetScaler Ingress Controller が `svc-class1` を受け入れるように構成されている場合、NetScaler 上でサービスを構成します。

```
apiVersion: v1

kind: Service

metadata:

     name: apache

     annotations:

         service.citrix.com/class: 'svc-class1'

     labels:

        name: apache

spec:

   type: LoadBalancer

   selector:

       name: apache

   ports:

   - name: http
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    port: 80
    targetPort: http
  selector:
  app: apache
```

NetScalerでHTTP、TCP、またはSSLプロファイルを構成する

NetScalerアプライアンスのHTTP、TCP、またはSSLなどの構成は、それぞれHTTPプロファイル、TCPプロファイル、またはSSLプロファイルなどの個別のエンティティを使用して指定できます。プロファイルは、個々のプロトコルに関連する設定の集合体です。たとえば、HTTPプロファイルはHTTP設定の集合体です。これにより、構成が容易になり、柔軟性が向上します。各エンティティで設定を構成する代わりに、プロファイルで設定を構成し、その設定が適用されるすべてのエンティティにプロファイルをバインドできます。

NetScaler Ingress Controllerを使用すると、プロファイルを使用してIngress NetScalerでHTTP、TCP、またはSSL関連の構成を行うことができます。

Kubernetes環境におけるNetScaler構成を理解する

Kubernetes環境では、Ingress NetScalerは外部トラフィックのフロントエンドとしてコンテンツスイッチング (CS)仮想サーバーを使用します。つまり、クライアントからの要求を受信するエンティティです。要求を処理した後、CS仮想サーバーは要求データを負荷分散 (LB)エンティティに渡します。LB仮想サーバーと関連するサービスグループは要求データを処理し、適切なアプリ（マイクロサービス）に転送します。

クライアントからトラフィックを受信するエンティティ（図では`Client Plane`として強調表示）には個別の`front end configuration`を、NetScalerからKubernetesのマイクロサービスにトラフィックを転送するエンティティ（図では`Server Plane`として強調表示）には`back end configuration`が必要です。

NetScaler Ingress Controllerは、要件に基づいて使用できるフロントエンドおよびバックエンド構成用の個別のスマートアノテーションを提供します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

HTTPプロファイル

HTTPプロファイルはHTTP設定の集合体です。デフォルトのHTTPプロファイル（`nshttp_default_profile`）は、すべてのサービスと仮想サーバーにグローバルにデフォルトで適用されるHTTP構成を設定するように構成されています。

NetScaler Ingress Controllerは、HTTPプロファイル用に次の2つのスマートアノテーションを提供します。これらのアノテーションを使用して、NetScalerのHTTP設定を定義できます。これらのアノテーションを含むIngressをデプロイすると、NetScaler Ingress ControllerはNetScalerに構成されているデフォルトのHTTPプロファイル（`nshttp_default_profile`）から派生したHTTPプロファイルを作成します。その後、アノテーションで指定したパラメーターを新しいHTTPプロファイルに適用し、そのプロファイルをNetScalerに適用します。

スマートアノテーション	説明	サンプル
`ingress.citrix.com/frontend-httpprofile`	このアノテーションを使用して、フロントエンドHTTPプロファイル（クライアントプレーン）を作成します。	`ingress.citrix.com/frontend-httpprofile: '{"dropinvalreqs":"enabled", "websocket" : "enabled"}'`
`ingress.citrix.com/backend-httpprofile`	このアノテーションを使用して、バックエンドHTTPプロファイル（サーバープレーン）を作成します。	`ingress.citrix.com/backend-httpprofile: '{"app-1": {"dropinvalreqs":"enabled", "websocket" : "enabled"}}'`
	注: NetScalerでHTTP関連のグローバルパラメータを手動で有効にしてください。たとえば、バックエンド（サーバープレーン）でHTTP2を使用するには、NetScalerで`HTTP2Serverside`グローバルパラメータを有効にできることを確認してください。詳細については、HTTP2の構成を参照してください。	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

TCPプロファイル

TCPプロファイルは、TCP設定の集合です。デフォルトのTCPプロファイル（`nstcp_default_profile`）は、すべてのサービスと仮想サーバーにグローバルにデフォルトで適用されるTCP構成を設定するように構成されています。

NetScaler Ingress Controllerは、TCPプロファイル用に以下の2つのスマートアノテーションを提供します。これらのアノテーションを使用して、NetScalerのTCP設定を定義できます。これらのアノテーションを含むイングレスを展開すると、NetScaler Ingress Controllerは、NetScalerに構成されているデフォルトのTCPプロファイル（`nstcp_default_profile`）から派生したTCPプロファイルを作成します。その後、アノテーションで指定したパラメータを新しいTCPプロファイルに適用し、そのプロファイルをNetScalerに適用します。

スマートアノテーション	説明	サンプル
`ingress.citrix.com/frontend-tcpprofile`	このアノテーションを使用して、フロントエンドTCPプロファイル（クライアントプレーン）を作成します。	`ingress.citrix.com/frontend-tcpprofile: '{"ws":"enabled", "sack" : "enabled"}'`
`ingress.citrix.com/backend-tcpprofile`	このアノテーションを使用して、バックエンドTCPプロファイル (サーバープレーン) を作成します。	`ingress.citrix.com/backend-tcpprofile: '{"citrix-svc": {"ws":"enabled", "sack" : "enabled"}}'`

SSLプロファイル

SSLプロファイル は、SSLエンティティの設定の集合です。これにより、設定が容易になり、柔軟性が向上します。各エンティティで設定を構成する代わりに、プロファイルで設定を構成し、その設定が適用されるすべてのエンティティにプロファイルをバインドできます。

前提条件

NetScalerでは、デフォルトで、Ingress NetScalerでSSLプロファイルが有効になっていません。NetScalerでSSLプロファイルを手動で有効にしてください。SSLプロファイルを有効にすると、NetScaler上の既存のすべてのSSL関連設定が上書きされます。SSLプロファイルの詳細については、SSLプロファイル を参照してください。

SSLプロファイルは、次の2つのカテゴリに分類されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- フロントエンドプロファイル: フロントエンドエンティティに適用されるパラメータを含みます。つまり、クライアントから要求を受信するエンティティに適用されます。
- バックエンドプロファイル: バックエンドエンティティに適用されるパラメータを含みます。つまり、クライアント要求をサーバーに送信するエンティティに適用されます。

NetScalerでSSLプロファイルを有効にすると、デフォルトのフロントエンドプロファイル (`ns_default_ssl_profile_frontend`) がSSL仮想サーバーに適用され、デフォルトのバックエンドプロファイル (`ns_default_ssl_profile_backend`) がNetScaler上のサービスまたはサービスグループに適用されます。

NetScaler Ingress Controllerは、SSLプロファイル用に次の2つのスマートアノテーションを提供します。これらのアノテーションを使用して、要件に基づいてデフォルトのフロントエンドプロファイル (`ns_default_ssl_profile_frontend`) とバックエンドプロファイル (`ns_default_ssl_profile_backend`) をカスタマイズできます。

スマートアノテーション	説明	例
`ingress.citrix.com/frontend-sslprofile`	このアノテーションを使用して、フロントエンドSSLプロファイル（クライアントプレーン）を作成します。フロントエンドSSLプロファイルは、クライアントプレーンでTLSを有効にしている場合にのみ必要です。	`ingress.citrix.com/frontend-sslprofile: '{"hsts":"enabled", "tls12" : "enabled"}'`
`ingress.citrix.com/backend-sslprofile`	このアノテーションを使用して、バックエンドSSLプロファイル（サーバープレーン）を作成します。SSLバックエンドプロファイルは、バックエンドにingress.citrix.com/secure-backendアノテーションを使用する場合にのみ必要です。	`ingress.citrix.com/backend-sslprofile: '{"citrix-svc": {"hsts":"enabled", "tls1" : "enabled"}}'`

重要: SSLプロファイルでは、SSL証明書を構成することはできません。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーションを使用したフロントエンドプロファイルの構成

HTTP、TCP、およびSSLフロントエンドプロファイルは、クライアント側のコンテンツスイッチング仮想サーバーまたはSSL仮想サーバーにアタッチされます。同じ`frontend-ip`を使用し、フロントエンドで同じコンテンツスイッチング仮想サーバーを使用する複数のイングレスが存在する可能性があるため、フロントエンドIPアドレスを共有する複数のイングレスで指定されたフロントエンドプロファイルアノテーションから競合が発生する可能性があります。

以下は、HTTP、TCP、およびSSLのフロントエンドプロファイルアノテーションに関するガイドラインです。

- 同じフロントエンドIPアドレスを持つすべてのイングレスでは、すべてのイングレスでフロントエンドプロファイルに同じ値を指定することをお勧めします。
- フロントエンドIPアドレスを共有する複数のイングレスがある場合、空のルールを持つ各フロントエンドIPアドレスに対して個別のイングレス（フロントエンドイングレスと呼ばれる）を作成することもできます。そのイングレスでは、以下の例に示すようにフロントエンドIPアノテーションを指定できます。各イングレス定義でフロントエンドプロファイルアノテーションを指定する必要はありません。
    - HTTPタイプの仮想サーバーのフロントエンドイングレスを作成するには、以下の例を参照してください。

```
 #Sample ingress manifest for the front-end configuration for an
HTTP virtual server

 #The values for the parameters are for demonstration purpose only.




 apiVersion: networking.k8s.io/v1

 kind: Ingress

 metadata:

   name: frontend-ingress

   annotations:

   # /* The CS virtual server is derived from the combination of
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
 insecure-port/secure-port, frontend-ip, and

 secure-service-type/insecure-service-type annotations. */

     ingress.citrix.com/insecure-port: "80"

     ingress.citrix.com/frontend-ip: "x.x.x.x"

     ingress.citrix.com/frontend-
httpprofile:'{"dropinvalreqs":"enabled", "markconnreqInval" :
"enabled"}'

     ingress.citrix.com/frontend-tcpprofile: '{"ws":"enabled",
"sack" :

 "enabled"}'

 spec:

   rules:

    # Empty rule

   - host:
```

- SSLタイプのサービスのフロントエンドイングレスを作成するには、以下の例を参照してください。

```
 #Sample ingress manifest for the front-end configuration for an SSL
virtual server

 #The values for the parameters are for demonstration purpose only.




 apiVersion: networking.k8s.io/v1

 kind: Ingress

 metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    name: frontend-ingress

    annotations:

    #The CS virtual server is derived from the combination of

    #insecure-port/secure-port, frontend-ip, and

    #secure-service-type/insecure-service-type annotations.

      ingress.citrix.com/insecure-port: "80"

      ingress.citrix.com/secure-port: "443"

      ingress.citrix.com/frontend-ip: "x.x.x.x"

      ingress.citrix.com/frontend-sslprofile:

 '{"tls13":"enabled", "hsts" : "enabled"}'

      ingress.citrix.com/frontend-tcpprofile: '{"ws":"enabled",
"sack" :

 "enabled"}'

 spec:

   rules:

   - host:

     #Presense of tls is considered as a secure service

   tls:

   - hosts:
```

- 複数のイングレスで同じフロントエンドプロファイルアノテーションに異なる値がある場合、プロファイルを仮想サーバーにバインドするために次の順序が使用されます。
  - いずれかのイングレス定義に、事前設定されたプロファイルを持つフロントエンドアノテーションがある場合、それは仮想サーバーにバインドされます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- 同じフロントエンドIPアドレスの異なるイングレスからすべての（キー、値）をマージし、結果の（キー、値）をフロントエンドプロファイルのスマートアノテーションに使用します。
- 異なるイングレスからの異なる値が原因で同じキーに競合がある場合、値はランダムに選択され、他の値は無視されます。競合する値を持たないようにする必要があります。

- フロントエンドIPアドレスを共有するいずれのイングレスにもフロントエンドプロファイルアノテーションが指定されていない場合、`FRONTEND_HTTP_PROFILE`、`FRONTEND_TCP_PROFILE`、または`FRONTEND_SSL_PROFILE`であるConfigMapからのグローバル値が、それぞれHTTP、TCP、およびSSLフロントエンドプロファイルに使用されます。

ConfigMap変数を使用したグローバルフロントエンドプロファイル構成

ConfigMap変数は、フロントエンドIPアドレスを共有する1つ以上のイングレスでフロントエンドプロファイルのスマートアノテーションによって上書きされない限り、フロントエンドプロファイルに使用されます。すべてのイングレスに対して任意のフロントエンドプロファイルを使用して機能を有効または無効にする必要がある場合は、HTTP、TCP、およびSSLプロファイルに対してそれぞれ変数`FRONTEND_HTTP_PROFILE`、`FRONTEND_TCP_PROFILE`、または`FRONTEND_SSL_PROFILE`を使用できます。たとえば、すべてのSSLイングレスでTLS 1.3を有効にしたい場合、各イングレス定義でスマートアノテーションを使用する代わりに、`FRONTEND_SSL_PROFILE`を使用してこの値を設定できます。NetScaler Ingress ControllerでConfigMapを使用する方法については、ConfigMapドキュメントを参照してください。

FRONTEND_HTTP_PROFILEを使用した構成

`FRONTEND_HTTP_PROFILE`変数は、イングレス定義の`ingress.citrix.com/frontend-httpprofile`スマートアノテーションによって上書きされない限り、フロントエンド仮想サーバー（クライアントプレーン）のHTTPオプションを設定するために使用されます。

NetScaler上の既存のプロファイルを使用するか、組み込みのHTTPプロファイルを使用する場合。

```
apiVersion: v1

kind: ConfigMap

metadata:

  name: cic-configmap

  labels:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    app: citrix-ingress-controller
data:
  FRONTEND_HTTP_PROFILE: |
    preconfigured: my_http_profile
```

この例では、`my_http_profile`はNetScalerに事前に存在するHTTPプロファイルです。

あるいは、次のように指定されたプロファイルパラメータを設定できます。すべての可能なキーと値については、HTTPプロファイルNITROドキュメントを参照してください。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cic-configmap
  labels:
    app: citrix-ingress-controller
data:
  FRONTEND_HTTP_PROFILE: |
    config:
      dropinvalreqs: 'ENABLED'
      websocket: 'ENABLED'
```

FRONTEND_TCP_PROFILEを使用した構成

`FRONTEND_TCP_PROFILE`変数は、イングレス定義の`ingress.citrix.com/frontend-tcpprofile`スマートアノテーションによって上書きされない限り、フロントエンド仮想サーバー（クライアント側）のTCPオプションを設定するために使用されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler上の既存のプロファイルを使用するか、組み込みのTCPプロファイルを使用する場合：

```
apiVersion: v1

kind: ConfigMap

metadata:

  name: cic-configmap

  labels:

    app: citrix-ingress-controller

data:

  FRONTEND_TCP_PROFILE: |

    preconfigured: my_tcp_profile
```

この例では、`my_tcp_profile`はNetScalerに事前に存在するTCPプロファイルです。

または、次のようにプロファイルパラメータを設定できます。すべての可能なキー値については、NetScaler TCPプロファイルNITROドキュメントを参照してください。

```
apiVersion: v1

kind: ConfigMap

metadata:

  name: cic-configmap

  labels:

    app: citrix-ingress-controller

data:

  FRONTEND_TCP_PROFILE: |

    config:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      sack: 'ENABLED'

      nagle: 'ENABLED'
```

FRONTEND_SSL_PROFILEを使用した構成

`FRONTEND_SSL_PROFILE`変数は、イングレス定義の`ingress.citrix.com/frontend-sslprofile`スマートアノテーションによって上書きされない限り、フロントエンド仮想サーバー（クライアント側）のSSLオプションを設定するために使用されます。

注記: SSLプロファイルを正しく機能させるには、`set ssl parameter -defaultProfile ENABLED`コマンドを使用してNetScalerでデフォルトプロファイルを有効にする必要があります。デフォルトプロファイルを有効にした後、NetScaler Ingress Controllerを再起動してください。NetScaler CPXがイングレスデバイスとして使用されている場合、デフォルトプロファイルは自動的に有効になります。SSLデフォルトプロファイルの詳細については、SSLプロファイルドキュメントを参照してください。

NetScaler上の既存のプロファイルを使用するか、組み込みのSSLプロファイルを使用するには、,

```
apiVersion: v1

kind: ConfigMap

metadata:

  name: cic-configmap

  labels:

    app: citrix-ingress-controller

data:

  FRONTEND_SSL_PROFILE: |

    preconfigured: my_ssl_profile
```

この例では、`my_ssl_profile`はNetScalerに既存のSSLプロファイルです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注記:

デフォルトのフロントエンドプロファイル（`ns_default_ssl_profile_frontend`）は、`FRONTEND_SSL_PROFILE.preconfigured`変数を使用してサポートされていません。

または、次の例に示すようにプロファイルパラメータを設定できます。すべての可能なキー値については、SSLプロファイルNITROドキュメントを参照してください。

```
apiVersion: v1

kind: ConfigMap

metadata:

  name: cic-configmap

  labels:

    app: citrix-ingress-controller

data:

  FRONTEND_SSL_PROFILE: |

    config:

      tls13: 'ENABLED'

      hsts: 'ENABLED'
```

次の例は、SSL暗号グループをSSLプロファイルにバインドする方法を示しています。順序はリストで指定されたとおりで、リストの最初のものに高い優先度が与えられ、以下同様です。このフィールドでは、NetScalerで利用可能な任意のSSL暗号、またはユーザーが作成した暗号グループを使用できます。NetScalerで利用可能な暗号のリストについては、NetScalerの暗号を参照してください。

```
apiVersion: v1

kind: ConfigMap
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
metadata:
  name: cic-configmap
  labels:
    app: citrix-ingress-controller
data:
  FRONTEND_SSL_PROFILE: |
    config:
      tls13: 'ENABLED'
      ciphers:
      - TLS1.3-AES256-GCM-SHA384
      - TLS1.3-CHACHA20-POLY1305-SHA256
```

バックエンド構成

サービスの詳細、`spec:rules:host`、`spec:backend`エントリなどを含むイングレス定義は、バックエンド構成と見なされます。

TLS構成なしのバックエンドイングレスマニフェストの例

```
#The values for the parameters are for demonstration purpose only.
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
  # /* The CS virtual server is derived from the combination of insecure-
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
port/secure-port, frontend-ip, and secure-service-type/insecure-service-
type annotations. */

    ingress.citrix.com/backend-httpprofile: '{"apache":{"markhttp09inval":
"disabled"}}'

    ingress.citrix.com/backend-tcpprofile: '{"apache":{"sack":"enabled"}}'

    ingress.citrix.com/frontend-ip: 'VIP_IP'

    ingress.citrix.com/insecure-port: "80"

  name: apache-ingress

spec:

  rules:

  - host: www.apachetest.com

    http:

      paths:

      - backend:

          service:

            name: apache

            port:

              number: 80

        path: /

        pathType: Prefix
```

TLS構成ありのバックエンドイングレスマニフェストの例

```


#The values for the parameters are for demonstration purpose only.
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  annotations:

  # /* The CS virtual server is derived from the combination of insecure-
port/secure-port, frontend-ip, and secure-service-type/insecure-service-
type annotations. */

    ingress.citrix.com/backend-httpprofile: '{"hotdrink":
{"markhttp09inval": "disabled"}}'

    ingress.citrix.com/backend-sslprofile: '{"hotdrink":{"snienable":
"enabled"}}'

    ingress.citrix.com/backend-tcpprofile: '{"hotdrink":
{"sack":"enabled"}}'

    ingress.citrix.com/frontend-ip: 'VIP_IP'

    ingress.citrix.com/secure-backend: '{"hotdrink":"true"}'

    ingress.citrix.com/secure-port: "443"

  name: hotdrink-ingress

spec:

  rules:

  - host: hotdrinks.beverages.com

    http:

      paths:

      - backend:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          service:
            name: hotdrink
            port:
              number: 443
        path: /
        pathType: Prefix
  tls:
  - secretName: hotdrink.secret
```

Ingress NetScalerで組み込みまたは既存のユーザー定義プロファイルを使用する

個々のスマートアノテーションを使用して、要件に基づいてIngress NetScaler上の組み込みプロファイルまたは既存のユーザー定義プロファイルをフロントエンドおよびバックエンド構成に設定できます。組み込みプロファイルの詳細は、組み込みTCPプロファイル および 組み込みHTTPプロファイル を参照してください。

フロントエンド構成の場合、Ingress NetScaler上の組み込みまたは既存のユーザー定義プロファイルの名前を指定できます。以下は、Ingressアノテーションのサンプルです。

```
ingress.citrix.com/frontend-httpprofile: "http_preconf_profile1"
```

ここで、「http_preconf_profile1」はIngress NetScaler上に存在するプロファイルです。

バックエンド構成の場合、Ingress NetScaler上の組み込みまたは既存のプロファイルの名前とバックエンドサービス名を指定する必要があります。以下は、Ingressアノテーションのサンプルです。

```
ingress.citrix.com/backend-httpprofile: '{"citrix-svc":
"http_preconf_profile1"}'
```

ここで、「http_preconf_profile1」はIngress NetScaler上に存在するプロファイルであり、`citrix-svc` はバックエンドサービス名です。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

HTTPプロファイルの例

```
ingress.citrix.com/frontend-httpprofile: "http_preconf_profile"

ingress.citrix.com/backend-httpprofile: '{"citrix-svc":
"http_preconf_profile"}'
```

TCPプロファイルの例

```
ingress.citrix.com/frontend-tcpprofile: "tcp_preconf_profile"

ingress.citrix.com/backend-tcpprofile: '{"citrix-
svc":"tcp_preconf_profile"}'
```

SSLプロファイルの例

```
ingress.citrix.com/frontend-sslprofile: "ssl_preconf_profile"

ingress.citrix.com/backend-sslprofile: '{"citrix-
svc":"ssl_preconf_profile"}'
```

HTTP、SSL、およびTCPプロファイルを適用する例

この例では、HTTP、SSL、またはTCPプロファイルを適用する方法を示します。

SSL、TCP、およびHTTPプロファイルを作成し、定義されたIngressリソースにバインドするには、次の手順を実行します。

1. 必要なプロファイルを使用してフロントエンドIngressリソースを定義します。このIngressリソースでは、バックエンドとTLSは定義されていません。

   サンプルYAML (`ingress1.yaml`) は次のとおりです。

   ```
   apiVersion: networking.k8s.io/v1

   kind: Ingress

   metadata:

     name: ingress-vpx1
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  annotations:

    ingress.citrix.com/insecure-termination: "allow"

    ingress.citrix.com/frontend-ip: "10.221.36.190"

    ingress.citrix.com/frontend-tcpprofile: '{"ws":"disabled", "sack" :
"disabled"}'

    ingress.citrix.com/frontend-httpprofile:
'{"dropinvalreqs":"enabled", "markconnreqInval" : "enabled"}'

    ingress.citrix.com/frontend-sslprofile: '{"hsts":"enabled",
"tls13" : "enabled"}'

spec:

  ingressClassName: "vpx"

  tls:

  - hosts:

  rules:

  - host:
```

2. フロントエンドIngressリソースを展開します。

   kubectl create -f ingress1.yaml を実行します。

3. 同じフロントエンドIPアドレスとTLS、およびロードバランシングリソース定義を作成するバックエンドが定義されたセカンダリイングレスリソースを定義します。

   サンプルのYAML (ingress2.yaml) は以下のとおりです。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: ingress-vpx2
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  annotations:
  ingress.citrix.com/insecure-termination: "allow"
  ingress.citrix.com/frontend-ip: "10.221.36.190"
spec:
  ingressClassName: "vpx"
  tls:
  - secretName: <hotdrink-secret>
  rules:
  - host:  hotdrink.beverages.com
    http:
      paths:
      - path:
        backend:
          serviceName: frontend-hotdrinks
          servicePort: 80
```

4. バックエンドイングレスリソースを展開します。

```
kubectl create -f ingress2.yaml
```

5. YAMLが適用されると、対応するエンティティ、プロファイル、およびイングレスリソースが作成され、イングレスリソースにバインドされます。

```
# show cs vserver &lt;k8s150-10.221.36.190_443_ssl>
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
k8s150-10.221.36.190_443_ssl (10.221.36.190:443) - SSL Type: CONTENT

State: UP

Last state change was at Thu Apr 22 20:14:44 2021

Time since last state change: 0 days, 00:10:56.850

Client Idle Timeout: 180 sec

Down state flush: ENABLED

Disable Primary Vserver On Down : DISABLED

Comment: uid=QEYQI2LDW5WR4A6P3NSZ37XICKOJKV4HPEM2H4PSK4HWA3JQWCLQ====

TCP profile name: k8s150-10.221.36.190_443_ssl

HTTP profile name: k8s150-10.221.36.190_443_ssl

Appflow® logging: ENABLED

State Update: DISABLED

Default:   Content Precedence: RULE

Vserver IP and Port insertion: OFF

L2Conn: OFF Case Sensitivity: ON

Authentication: OFF

401 Based Authentication: OFF

Push: DISABLED Push VServer:

Push Label Rule: none

Persistence: NONE

Listen Policy: NONE

IcmpResponse: PASSIVE
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  RHIstate:  PASSIVE

  Traffic Domain: 0



  1)  Content-Switching Policy: k8s150-ingress-vpx1_tier-2-
adc_443_k8s150-frontend-hotdrinks_tier-2-adc_80_svc    Priority:
200000004   Hits: 0

  Done
```

例: SNI証明書をSSL仮想サーバーに追加する

この例では、単一のSNI証明書を追加する方法を示します。

注:

SSLプロファイルが正しく機能するためには、`set ssl parameter -defaultProfile ENABLED` コマンドを使用してNetScalerでデフォルトプロファイルを有効にする必要があります。デフォルトプロファイルを有効にした後、NetScaler Ingress Controllerが再起動されていることを確認してください。SSLデフォルトプロファイルの詳細は、ドキュメントを参照してください。

1. 必要なプロファイルを持つフロントエンドイングレスリソースを定義します。このイングレスリソースでは、バックエンドとTLSは定義されていません。

   サンプルのYAML (ingress1.yaml) は以下のとおりです。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: ingress-vpx1

  annotations:

   ingress.citrix.com/insecure-termination: "allow"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
   ingress.citrix.com/frontend-ip: "10.221.36.190"

   ingress.citrix.com/frontend-tcpprofile: '{"ws":"disabled", "sack" :
"disabled"}'

   ingress.citrix.com/frontend-httpprofile:
'{"dropinvalreqs":"enabled", "markconnreqInval" : "enabled"}'

   ingress.citrix.com/frontend-sslprofile: '{"snienable": "enabled",
"hsts":"enabled", "tls13" : "enabled"}'

spec:

  ingressClassName: "vpx"

  tls:

  - hosts:

  rules:

  - host:
```

2. フロントエンドイングレスリソースを展開します。

```
kubectl create -f ingress1.yaml
```

3. 同じフロントエンドIPアドレスを持ち、バックエンドとSNI証明書を定義するセカンダリイングレスリソースを定義します。ホストが指定されている場合、シークレット名として指定されたcertkeyがSNI証明書として追加されます。

   サンプルのYAML (ingress2.yaml) は以下のとおりです。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: ingress-vpx2
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  annotations:
   ingress.citrix.com/insecure-termination: "allow"
   ingress.citrix.com/frontend-ip: "10.221.36.190"
spec:
  ingressClassName: "vpx"
  tls:
  - hosts:
      - hotdrink.beverages.com
    secretName: hotdrink-secret
  rules:
  - host: hotdrink.beverages.com
    http:
      paths:
      - path: /
        backend:
          serviceName: web
          servicePort: 80
```

4. セカンダリイングレスリソースを展開します。

```
kubectl create -f ingress2.yaml
```

複数のSNI証明書をフロントエンドVIPにバインドする必要がある場合、以下はサンプルYAMLファイルです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: ingress-vpx-frontend

  ingress.citrix.com/insecure-termination: "allow"

  ingress.citrix.com/frontend-ip: "10.221.36.190"

spec:

  ingressClassName: "vpx"

  tls:

  - hosts:

      - hotdrink.beverages.com

    secretName: hotdrink-secret

  - hosts:

      - frontend.agiledevelopers.com

    secretName: &lt;frontend-secret>

  rules:

  - host: hotdrink.beverages.com

    http:

      paths:

      - path: /

        backend:

          serviceName: web
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          servicePort: 80
  - host: frontend.agiledevelopers.com
    http:
      paths:
      - path: /
        backend:
          serviceName: frontend-developers
          servicePort: 80
```

例: SSL暗号グループのバインド

この例では、SSL暗号グループをバインドする方法を示します。

注:

SSLプロファイルが正しく機能するためには、`set ssl parameter -defaultProfile ENABLED` コマンドを使用してNetScalerでデフォルトプロファイルを有効にする必要があります。デフォルトプロファイルを有効にした後、NetScaler Ingress Controllerが再起動されていることを確認してください。

NetScaler Ingress Controllerを展開する前に、`set ssl parameter -defaultProfile ENABLED` コマンドを使用してNetScalerでデフォルトのSSLプロファイルを設定します。NetScaler Ingress Controllerをすでに展開している場合は、再展開してください。SSLデフォルトプロファイルの詳細は、ドキュメントを参照してください。

NetScalerアプライアンスでサポートされている暗号に関する情報は、NetScalerアプライアンスで利用可能な暗号を参照してください。

暗号のセキュリティ保護に関する情報は、暗号のセキュリティ保護を参照してください。

サンプルのYAML (cat frontend_ingress.yaml) は次のとおりです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: ingress-vpx

  annotations:

   ingress.citrix.com/insecure-termination: "allow"

   ingress.citrix.com/frontend-ip: "10.221.36.190"

   ingress.citrix.com/frontend-tcpprofile: '{"ws":"disabled", "sack" :
"disabled"}'

   ingress.citrix.com/frontend-httpprofile: '{"dropinvalreqs":"enabled",
"markconnreqInval" : "enabled"}'

   ingress.citrix.com/frontend-sslprofile: '{"snienable": "enabled",
"hsts":"enabled", "tls13" : "enabled", "ciphers" : [{"ciphername": "test",
"cipherpriority" :"1"}]}'

spec:

  ingressClassName: "citrix"

  tls:

  - hosts:

  rules:

   - host:
```

ログレベル

NetScaler Ingress Controllerによって生成されたログは、kubernetes logsの一部として利用できます。NetScaler Ingress Controllerに以下のログレベルでログを記録するように指定できます。

- CRITICAL

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- ERROR
- WARNING
- INFO
- DEBUG

デフォルトでは、NetScaler Ingress Controllerは`INFO`ログレベルでログを記録するように設定されています。特定のログレベルでNetScaler Ingress Controllerにログを記録させたい場合は、NetScaler Ingress Controllerを展開する前に、NetScaler Ingress Controllerの展開YAMLファイルでログレベルを指定する必要があります。YAMLファイルの`spec`セクションで、次のようにログレベルを指定できます。

```
apiVersion: v1

kind: Pod

metadata:

  name: citrixingresscontroller

  labels:

    app: citrixingresscontroller

spec:

      serviceAccountName: cpx

      containers:

      - name: citrixingresscontroller

        image: "quay.io/citrix/citrix-k8s-ingress-controller:1.36.5"

        env:

        # Set kube api-server URL

        - name: "kubernetes_url"

          value: "https://10.x.x.x:6443"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        # Set NetScaler Management IP
        - name: "NS_IP"
          value: "10.x.x.x"
        # Set log level
        - name: "LOGLEVEL"
          value: "DEBUG"
        - name: "EULA"
          value: "yes"
        args:
        - --feature-node-watch
          true
        imagePullPolicy: Always
```

ログレベルの変更

NetScaler Ingress Controllerインスタンスに設定されているログレベルを変更するには、インスタンスを削除し、以下のセクションでログレベルの値を更新して、NetScaler Ingress Controllerインスタンスを再展開する必要があります。

```
# Set log level
- name: "LOGLEVEL"
  value: "XXXX"
```

ログレベルを更新したら、YAMLファイルを保存し、次のコマンドを使用して展開します。

```
kubectl create -f citrix-k8s-ingress-controller.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Kubernetesシークレットリソースを介したLoadBalancerタイプのサービス用SSL証明書

このセクションでは、SSL証明書を KubernetesシークレットとしてLoadBalancerタイプのサービスで使用する方法について説明します。アノテーション`service.citrix.com/service-type`が`SSL`または`SSL_TCP`の場合、証明書が適用されます。

NetScaler Ingress Controllerのデフォルト証明書を使用する

SSL証明書が提供されていない場合、デフォルトのNetScaler Ingress Controller証明書を使用して、SSLおよびSSL SNI証明書を構成できます。default-ssl-certificateおよびdefault-ssl-sni-certificate引数を使用して、それぞれ非SNI証明書とSNI証明書を構成するためのシークレットを提供できます。

```
`--default-ssl-certificate <NAMESPACE>/<SECRET_NAME>`

`--default-ssl-sni-certificate <NAMESPACE>/<SECRET_NAME>`
```

Kubernetesシークレットとしての SSL証明書のためのサービスアノテーション

NetScaler Ingress Controllerは、`LoadBalancer`タイプのサービスに対してKubernetesシークレットとして保存されているSSL証明書を使用するための以下のサービスアノテーションを提供します。

サービスアノテーション	説明
`service.citrix.com/secret`	このアノテーションを使用して、フロントエンドサーバー証明書のシークレットリソース名を指定します。これには証明書とキーが含まれている必要があります。証明書セクションに、サーバー証明書に続いて中間CA証明書のリストを提供することもできます。これらの中間CAは、SSLハンドシェイク中に自動的にリンクされ、クライアントに送信されます。複数のフロントエンドサーバー証明書をバインドするには、証明書用に構成されたカンマ区切りのシークレットのリストを提供します。例: `service.citrix.com/secret: apache-secret1,apache-secret2`。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サービスアノテーション	説明
`service.citrix.com/ca-secret`	このアノテーションを使用して、クライアント証明書認証用のCA証明書を提供します。この証明書は、NetScalerのフロントエンドSSL仮想サーバーにバインドされます。
`service.citrix.com/backend-secret`	NetScalerとワークロード間のバックエンド通信が暗号化されたチャネル上で行われ、ワークロードでクライアント認証が必要な場合は、このアノテーションを使用します。この証明書は、SSLハンドシェイク中にサーバーに送信され、バックエンドSSLサービスグループにバインドされます。
`service.citrix.com/backend-ca-secret`	このアノテーションを使用して、バックエンドサーバー証明書を認証するサーバー認証を有効にします。この設定により、サーバーのCA証明書がNetScaler上のSSLサービスにバインドされます。
`service.citrix.com/preconfigured-certkey`	このアノテーションを使用して、フロントエンドサーバー証明書として使用するNetScaler内の既存の証明書キーの名前を指定します。複数のフロントエンドサーバー証明書をバインドするには、証明書用にすでに設定されているコンマ区切りの証明書キーのリストを指定します。例: `service.citrix.com/preconfigured-certkey: preconfcert1,preconfcert2`。
`service.citrix.com/preconfigured-ca-certkey`	このアノテーションを使用して、クライアント証明書認証用のCA証明書として使用するNetScaler内の事前設定された証明書キーの名前を指定します。この証明書は、NetScalerのフロントエンドSSL仮想サーバーにバインドされます。
`service.citrix.com/preconfigured-backend-certkey`	このアノテーションを使用して、バックエンドSSLサービスグループにバインドするNetScaler内の事前設定された証明書キーの名前を指定します。この証明書は、サーバー認証

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サービスアノテーション	説明
	のためにSSLハンドシェイク中にサーバーに送信されます。
`service.citrix.com/preconfigured-backend-ca-certkey`	このアノテーションを使用して、サーバー認証のためにバックエンドSSLサービスグループにバインドするNetScaler内の事前設定されたCA証明書キーの名前を指定します。

例: フロントエンドシークレットとフロントエンドCAシークレット

`service.citrix.com/secret`アノテーションの例をいくつか示します。

次のアノテーションは、サービス内のすべてのポートに適用されます。

```
service.citrix.com/secret: hotdrink-secret
```

キーとして`portname`または`port`-`protocol`のいずれかを指定することで、特定のポートに適用される証明書を指定するために次の表記を使用できます。

```
        # port-protocol : secret



        service.citrix.com/secret: '{"443-tcp": "hotdrink-secret", "8443-
tcp": "hotdrink-secret"}'



         # portname: secret



        service.citrix.com/secret: '{"https": "hotdrink-secret"}'
```

`service.citrix.com/ca-secret`アノテーションの例をいくつか示します。

Kubernetesにデプロイされたサービスのクライアント証明書認証に使用される生成されたCAシークレットをアタッチするには、次のアノテーションを指定する必要があります。

以下の注釈は、サービス内のすべてのポートに適用されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
service.citrix.com/ca-secret: hotdrink-ca-secret
```

キーとして`portname`または`port`-`protocol`のいずれかを与えることで、特定のポートに適用される証明書を指定するために、以下の表記を使用できます。

```
  # port-protocol: secret

  service.citrix.com/ca-secret: '{"443-tcp": "hotdrink-ca-secret", "8443-
tcp": "hotdrink-ca-secret"}'



  # portname: secret



  service.citrix.com/ca-secret: '{"https": "hotdrink-ca-secret"}'
```

例: バックエンドシークレットとバックエンドCAシークレット

`service.citrix.com/backend-secret`注釈の例をいくつか示します。

```
  # port-protocol: secret

  service.citrix.com/backend-secret: '{"443-tcp": "hotdrink-secret",
"8443-tcp": "hotdrink-secret"}'



  # portname: secret



  service.citrix.com/backend-secret: '{"tea-443": "hotdrink-secret",
"tea-8443": "hotdrink-secret"}'



  # applicable to all ports
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  service.citrix.com/backend-secret: "hotdrink-secret"
```

`service.citrix.com/backend-ca-secret`注釈の例をいくつか示します。

```
  # port-proto: secret

  service.citrix.com/backend-ca-secret: '{"443-tcp": "coffee-ca", "8443-
tcp": "tea-ca"}'



  # portname: secret



  service.citrix.com/backend-ca-secret: '{"coffee-443": "coffee-ca",
"tea-8443": "tea-ca"}'



  # applicable to all ports



  service.citrix.com/backend-ca-secret: "hotdrink-ca-secret"
```

NetScaler CPXを使用したLoadBalancerタイプのサービスおよびIngressの外部IPアドレスのBGPアドバタイズメント

`LoadBalancer`タイプのKubernetesサービスサポートは、クラウド環境のクラウドロードバランサーによって提供されます。 クラウドサービスプロバイダーは、ロードバランサーを自動的に作成し、サービスステータスの一部として表示されるIPアドレスを割り当てることで、このサポートを有効にします。外部IPアドレス宛のトラフィックは、クラウドロードバランサーによってNodeIPとNodePortでロードバランシングされます。トラフィックがKubernetesクラスターに到達すると、kube-proxyはiptablesまたはIP仮想サーバーのルールを使用して、実際のアプリケーションポッドへのルーティングを実行します。ただし、オンプレミス環境では、クラウドロードバランサーの自動構成は利用できません。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Ingress Controllerと、NetScaler VPXまたはMPXなどのTier-1 NetScalerデバイスを使用して、`LoadBalancer`タイプのサービスを公開できます。Kubernetesクラスターの外部にあるNetScaler VPXまたはMPXは、Kubernetesサービスへの受信トラフィックをロードバランシングします。このようなデプロイメントの詳細については、「`LoadBalancer`タイプのサービスを公開する」を参照してください。

しかし、オンプレミス環境でLoadBalancerタイプのサービスを公開するために外部ADCデバイスを使用することが常に可能であるとは限りません。外部コンポーネントなしで、Kubernetesクラスター自体から関連するすべてのリソースを管理することが望ましい場合があります。NetScaler Ingress Controllerは、Kubernetesクラスター内で実行されるNetScaler CPXを使用して、LoadBalancerタイプのサービスを公開する方法を提供します。Kubernetesノードへのトラフィックをルーティングするための既存のBGPファブリックが、このソリューションの実装に活用されます。

このデプロイメントでは、NetScaler CPXはホストモードでKubernetesノード上にデーモンセットとしてデプロイされます。NetScaler CPXは、ネットワークルーターとのBGPピアリングセッションを確立し、そのピアリングセッションを使用して外部クラスターサービスのIPアドレスをアドバタイズします。ルーターがECMP機能を備えている場合、トラフィックはアップストリームルーターによって複数のCPXインスタンスにロードバランシングされ、それが実際のアプリケーションポッドにロードバランシングされます。このモードでNetScaler CPXをデプロイすると、NetScaler CPXはKubernetesノード上のLoadBalancerタイプの各サービスに対してiptablesルールを追加します。外部IPアドレス宛のトラフィックは、NetScaler CPXポッドにルーティングされます。

次の図は、NetScaler CPXがLoadBalancerタイプのサービスを公開しているデプロイメントを説明しています。

図に示すように、NetScaler CPXはデーモンセットとして実行され、Kubernetesノードリソースによって示されるノードIPアドレス上のポート179でBGPセッションを実行します。Kubernetes APIサーバーに追加されるLoadBalancerタイプのすべてのサービスに対して、NetScaler Ingress Controllerは、設定されたBGPルーターに外部IPアドレスをアドバタイズするようにNetScaler CPXを構成します。/32プレフィックスは外部ルーターへのルートをアドバタイズするために使用され、ノードIPアドレスは外部IPアドレスに到達するためのゲートウェイとして使用されます。トラフィックがKubernetesノードに到達すると、iptablesルールがトラフィックをNetScaler CPXに誘導し、NetScaler CPXが実際のサービスポッドにロードバランシングします。

このデプロイメントでは、Kubernetes Ingressリソースを使用し、Ingress仮想IP (VIP) アドレスをルーターにアドバタイズすることもできます。NetScaler Ingress Controllerをデプロイする際に、すべてのIngressリソースのVIPとして機能する`NS_VIP`環境変数を指定できます。Ingressリソー

© 1997–2026 Citrix Systems, Inc. All rights reserved.

スが追加されると、NetScaler CPXはBGPを介して`NS_VIP`を外部ルーターにアドバタイズし、トラフィックを引き付けます。トラフィックが`NS_VIP`に到達すると、NetScaler CPXはIngressリソースで指定されたコンテンツスイッチングとロードバランシングを実行します。

注:

このソリューションが機能するには、NetScaler Ingress Controllerがrootユーザーとして実行され、`NET_ADMIN`機能を持っている必要があります。

LoadBalancerタイプのサービス向けNetScaler CPXソリューションをデプロイする

この手順では、ホストネットワークにデーモンセットとしてNetScaler CPXをデプロイし、LoadBalancerタイプのサービスを公開する方法を説明します。

この構成には、次のタスクが含まれます。

- NetScaler Ingress Controllerをサイドカーとして使用してNetScaler CPXをデプロイする
- BGP設定
- サービス設定

前提条件

- ECMPをサポートするBGPルーティング用にアップストリームルーターを設定し、Kubernetesノードをネイバーとして追加する必要があります。
- ルーターがロードバランシングをサポートしている場合、均等なロードバランシングのために、より高いエントロピーを持つ安定したECMPハッシュアルゴリズムを使用することをお勧めします。

次の操作を実行します。

1. rbac.yamlファイルをダウンロードし、NetScaler CPXおよびNetScaler Ingress ControllerのRBACルールを展開します。

   ```
   kubectl apply -f rbac.yaml
   ```

2. 次のコマンドを使用して、citrix-k8s-cpx-ingress.ymlをダウンロードします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
wget  https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/docs/configure/cpx-bgp-router/citrix-k8s-cpx-ingress.yml
```

3. `citrix-k8s-cpx-ingress.yaml`ファイルを編集し、必要な値を指定します。
   - 引数`–configmap`は、`namespace/name`の形式でNetScaler Ingress ControllerのConfigMapの場所を指定します。
   - 自動IPアドレス割り当てのためにNetScaler IPAM Controllerを実行している場合は、引数`--ipam citrix-ipam-controller`を指定できます。
   - (オプション) NetScaler CPXデーモンセットを実行する必要があるノードを選択するには、`nodeSelector`を使用します。デフォルトでは、すべてのワーカーノードで実行されます。
4. `citrix-k8s-cpx-ingress.yaml`ファイルを適用して、NetScaler CPXとNetScaler Ingress Controllerを起動するデーモンセットを作成します。

```
kubectl apply -f citrix-k8s-cpx-ingress.yml
```

5. NetScaler Ingress Controllerへの引数として渡されるBGP設定を含むConfigMap (configmap.yaml) を作成します。BGP設定の詳細については、BGP設定を参照してください。

   BGPルーティングを設定するには、次の情報が必要です。

   - NetScaler CPXが接続するルーターのIPアドレス
   - ルーターの自律システム（AS番号）
   - NetScaler CPXのAS番号

   以下は、BGP構成を含むConfigMapのサンプルです。

```
apiVersion: v1

kind: ConfigMap

metadata:

  name: config

  labels:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    app: cic
data:
  NS_BGP_CONFIG: |
    bgpConfig:
    - bgpRouter:
        localAS: 100
        neighbor:
        - address: 10.102.33.33
          remoteAS: 100
          advertisementInterval: 10
          ASOriginationInterval: 10
```

6. ステップ5で作成したConfigMapを適用して、BGP構成を適用します。

   kubectl apply -f configmap.yaml を実行します。

7. LoadBalancerタイプのサービスに必要な構成を含むYAMLファイルを作成します。

   注:

   詳細については、サービス構成を参照してください。サービス構成セクションでは、サービスに外部IPアドレスを取得するさまざまな方法と、NetScalerが提供するサービスアノテーションを使用してさまざまなNetScaler機能を構成する方法について説明しています。

   以下は、LoadBalancerタイプのサービスの構成例です。

```
apiVersion: v1
kind: Service
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
metadata:

  name: kuard-service

  annotations:

    # This uses IPAM to allocate an IP from range 'Dev'

    # service.citrix.com/ipam-range: 'Dev'

    service.citrix.com/frontend-ip: 172.217.163.17

    service.citrix.com/service-type-0: 'HTTP'

    service.citrix.com/service-type-1: 'SSL'

    service.citrix.com/lbvserver: '{"80-tcp":
{"lbmethod":"ROUNDROBIN"}}'

    service.citrix.com/servicegroup: '{"80-tcp":{"usip":"yes"}}'

    service.citrix.com/ssl-termination: edge

    service.citrix.com/monitor: '{"80-tcp":{"type":"http"}}'

    service.citrix.com/frontend-httpprofile:
'{"dropinvalreqs":"enabled", "websocket" : "enabled"}'

    service.citrix.com/backend-httpprofile:
'{"dropinvalreqs":"enabled", "websocket" : "enabled"}'

    service.citrix.com/frontend-tcpprofile: '{"ws":"enabled", "sack" :
"enabled"}'

    service.citrix.com/backend-tcpprofile: '{"ws":"enabled", "sack" :
"enabled"}'

    service.citrix.com/frontend-sslprofile: '{"hsts":"enabled",
"tls12" : "enabled"}'

    service.citrix.com/backend-sslprofile: '{"tls12" : "enabled"}

    service.citrix.com/ssl-certificate-data-1: |
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      -----BEGIN-----

           [...]

      -----END-----

    service.citrix.com/ssl-key-data-1: |

spec:

  type: LoadBalancer

  selector:

    app: kuard

  ports:

  - port: 80

    targetPort: 8080

    name: http

  - port: 443

    targetPort: 8443

    name: https
```

8. LoadBalancerタイプのサービスを適用します。

```
kubectl apply -f service-example.yaml
```

サービスが適用されると、NetScaler Ingress ControllerはBGPルートヘルスインジェクションが有効なロードバランシング仮想サーバーを作成します。ロードバランシング仮想サーバーの状態が `UP` の場合、外部IPアドレスのルートは、ノードIPアドレスをゲートウェイとする/32プレフィックスでネイバールーターにアドバタイズされます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

BGP構成

BGP構成は、NetScaler Ingress Controllerに引数として渡されるConfigMapを使用して実行されます。

BGPルーティングを構成するには、以下の情報が必要です。

- NetScaler CPXが接続できるようにするルーターのIPアドレス
- ルーターの自律システム（AS番号）
- NetScaler CPXのAS番号

以下のBGP構成のConfigMapでは、`bgpConfig`フィールドがBGP構成を表します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config
  labels:
    app: cic
data:
  NS_BGP_CONFIG: |
    bgpConfig:
    - bgpRouter:
        localAS: 100
        neighbor:
        - address: x.x.x.x
          remoteAS: 100
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        advertisementInterval: 10

        ASOriginationInterval: 10
```

次の表は、`bgpConfig`フィールドのさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	デフォルト値	必須
`nodeSelector`	`nodeSeclector`フィールドが存在する場合、BGPルーター構成はnodeSelectorフィールドに一致するノードに適用されます。`nodeSelector`は、各キーがラベル名を表し、値がラベル値であるカンマ区切りの`key=value`ペアを受け入れます。例：nodeSelector: datacenter=ds1,rack-rack1	文字列		いいえ
`bgpRouter`	BGP構成を指定します。`bgpRouter`のさまざまなフィールドについては、次の表を参照してください。	bgpRouter		はい

次の表は、`bgpRouter`フィールドの各フィールドについて説明しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	デフォルト値	必須
`localAS`	NetScaler CPXのAS番号	整数		はい
`neighbor`	ネイバールーターのBGP構成。	ネイバー		はい

次の表は、`neighbor` フィールドについて説明しています。

フィールド	説明	タイプ	デフォルト値	必須	
`address`	ネイバールーターのIP アドレス。	文字列		はい	
`remoteAS`	ネイバールーターのAS 番号。	整数			はい
`advertisementInterval`	このフィールドは、BGPルーティングアップデートの送信間の最小間隔（秒単位）を設定します。	整数	10秒	はい	
`ASOriginationInterval`	このフィールドは、AS発信ルーティングアップデートの送信間隔（秒単位）を設定します。	整数	10秒	はい	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ノードごとに異なるネイバー

デフォルトでは、クラスター内のすべてのノードは、設定にリストされているすべてのネイバーに接続します。しかし、Kubernetesクラスターが異なるデータセンターや異なるネットワークにまたがっている場合、ノードごとに異なるネイバー設定が必要になることがあります。BGPルーティング設定に必要なノードを選択するには、`nodeSelector`フィールドを使用できます。

`nodeSelector`設定を含むConfigMapの例を以下に示します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: config
  labels:
    app: cic
data:
  NS_BGP_CONFIG: |
    bgpConfig:
    - nodeSelector: datacenter=ds1
      bgpRouter:
        localAS: 100
        neighbor:
        - address: 10.102.33.44
          remoteAS: 100
          advertisementInterval: 10
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        ASOriginationInterval: 10

  - nodeSelector: datacenter=ds2

    bgpRouter:

      localAS: 100

      neighbor:

      - address: 10.102.28.12

        remoteAS: 100

        advertisementInterval: 10

        ASOriginationInterval: 10
```

この例では、IPアドレス10.102.33.44を持つルーターは、`datacenter=ds1`ラベルを持つノードによってネイバーとして使用されます。IPアドレス10.102.28.12を持つルーターは、`datacenter=ds2`ラベルを持つノードによって使用されます。

サービス構成

外部IPアドレス構成

LoadBalancerタイプのサービス用の外部IPアドレスは、以下のいずれかの方法で取得できます。

- サービス仕様で`service.citrix.com/frontend-ip`アノテーションを次のように指定します。

  ```
  metadata:

      annotations:

          service.citrix.com/frontend-ip: 172.217.163.17
  ```

- サービス仕様の`spec.loadBalancerIP`フィールドにIPアドレスを次のように指定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:

    loadBalancerIP: 172.217.163.17
```

- NetScalerが提供するIPAMコントローラーを使用して、サービスに仮想IPアドレスを自動的に割り当てます。他の2つの方法のいずれかが指定されている場合、その方法がIPAMコントローラーよりも優先されます。IPAMソリューションは、InfobloxなどのExternalDNSプロバイダーと簡単に統合できるように設計されています。詳細については、「ExternalDNSとの相互運用性」を参照してください。NetScaler IPAM Controllerの展開と使用については、ドキュメントを参照してください。

サービスアノテーション構成

NetScaler Ingress Controllerは、NetScalerのさまざまな機能を活用するための多くのサービスアノテーションを提供します。たとえば、ロードバランシング仮想サーバーのデフォルトのサービスタイプは`TCP`ですが、`service.citrix.com/service-type`アノテーションによってこの構成を上書きできます。

```
metadata:

    annotations:

        service.citrix.com/service-type-0: 'HTTP'

        service.citrix.com/service-type-1: 'SSL'
```

NetScaler Ingress Controllerが提供するさまざまなアノテーションを利用することで、SSLオフロード、HTTPリライト、レスポンダーポリシー、その他のカスタムリソース定義 (CRD) など、さまざまなADC機能を活用できます。

LoadBalancerタイプのサービスのすべてのアノテーションの詳細については、以下を参照してください。 サービスアノテーション。

LoadBalancerタイプのサービスのSSL証明書にシークレットリソースを使用する方法については、LoadBalancerタイプのサービスのSSL証明書を参照してください。

外部トラフィックポリシー構成

デフォルトでは、NetScaler Ingress Controllerは、NetScaler CPX内のロードバランシング仮想サービスに対して、すべてのサービスポッドをバックエンドとして追加します。この手順により、高可用性が向上し、サービスポッドインスタンスへの均等な分散が保証されます。NetScaler CPX

© 1997–2026 Citrix Systems, Inc. All rights reserved.

を実行しているすべてのノードは、アップストリームサーバーへのルートをアドバタイズし、ルーターからのトラフィックを引き付けます。この動作は、サービスの `spec.externalTrafficPolicy` を `Local` に設定することで変更できます。外部トラフィックポリシーが `Local` に設定されている場合、次の図に示すように、同じノードで実行されているポッドのみがロードバランシング仮想サーバーのバックエンドとして追加されます。このモードでは、サービスポッドを持つノードのみがルーターに外部IPアドレスをアドバタイズし、CPXはローカルポッドにのみトラフィックを送信します。 パフォーマンス上の理由からノード間でトラフィックがホッピングするのを避けたい場合は、この機能を使用できます。

Ingressリソースの使用

NetScaler Ingress Controllerは、すべてのIngressリソースの外部IPアドレスであるnt変数 `NS_VIP` を提供します。Ingressリソースが追加されるたびに、NetScaler CPXはIngress IPアドレスを外部ルーターにアドバタイズします。 NetScaler Ingress Controllerは、Ingress用のさまざまなアノテーションを提供します。詳細については、Ingressアノテーションのドキュメントを参照してください。

Ingress構成には、次の手順を実行します。

1. rbac.yamlファイルをダウンロードし、NetScaler CPXおよびNetScaler Ingress ControllerのRBACルールを展開します。

   ```
   kubectl apply -f rbac.yaml
   ```

2. 次のコマンドを使用して、citrix-k8s-cpx-ingress.ymlをダウンロードします。

   ```
   wget  https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/docs/configure/cpx-bgp-router/citrix-k8s-cpx-ingress.yml
   ```

3. `citrix-k8s-cpx-ingress.yml` ファイルを編集し、必要な値を指定します。

   - 引数 `-configmap` は、NetScaler Ingress ControllerのConfigMapの場所を名前空間または名前の形式で指定します。
   - すべてのIngressリソースに使用する外部IPを指定する環境変数 `NS_VIP` 。(これは必須パラメーターです)。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

4. `citrix-k8s-cpx-ingress.yml`ファイルを適用して、NetScaler CPXとNetScaler Ingress Controllerを起動するデーモンセットを作成します。

```
kubectl apply -f citrix-k8s-cpx-ingress.yml
```

5. 前のセクションで示したように、ConfigMapを使用してBGPを構成します。
6. 次のようにサンプルIngressリソースを展開します。この手順では、`NS_VIP`環境変数で指定されたIPアドレスを、ConfigMapで構成された外部ルーターにアドバタイズします。

```
kubectl apply -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/docs/configure/cpx-bgp-router/ingress-example.yaml
```

7. `NS_VIP:<port>`を使用してアプリケーションにアクセスします。デフォルトでは、Ingressは安全でない通信にはポート80を、安全な通信にはポート443を使用します（TLSセクションが提供されている場合）。

注： 現在、`ingress.citrix.com/frontend-ip`アノテーションはBGPアドバタイズメントではサポートされていません。

Helmのインストール

Helmチャートを使用して、NetScaler CPXをBGPルーターとしてインストールできます。詳細については、Citrix Helmチャートのドキュメントを参照してください。

トラブルシューティング

- デフォルトでは、NetScaler CPX は内部通信に IP アドレス範囲 192.168.1.0/24 を使用し、ホストへの内部ゲートウェイとして IP アドレス 192.168.1.1 を使用し、NSIP として IP アドレス 192.168.1.2 を使用します。ポート 9080 および 9443 は、HTTP および HTTPS 用の NetScaler Ingress Controller と NetScaler CPX 間の管理ポートとして使用されます。192.168.1.0/24 ネットワークが PodCIDR の範囲内にある場合、内部通信用に異なる IP アドレスのセットを割り当てることができます。`NS_IP` および `NS_GATEWAY` 環境変数は、NetScaler CPX が NSIP およびゲートウェイにそれぞれ使用する IP アドレスを制御します。NetScaler Ingress Controller と NetScaler CPX 間の通信を確立するには、同じ IP アドレスを NetScaler Ingress Controller 環境変数 `NS_IP` の一部として指定する必要もあります。
- デフォルトでは、NetScaler CPX の BGP はポート 179 で実行され、TCP ポート 179 に到達するすべての BGP トラフィックは NetScaler CPX によって処理されます。競合がある場合、たとえば Calico の外部 BGP ピアリング機能を使用して BGP 経由でクラスタプレフィックス

© 1997–2026 Citrix Systems, Inc. All rights reserved.

をアドバタイズしている場合、NetScaler Ingress Controller の環境変数 `BGP_PORT` を使用して BGP ポートを変更できます。

- Kubernetes の制約により、NetScaler のソース IP (USIP) モードは機能しません。サービスでソース IP アドレスが必要な場合は、次のアノテーションを使用して、HTTP/SSL サービスタイプのサービスで CIP (クライアント IP ヘッダー) 機能を有効にできます。

  service.citrix.com/servicegroup: '{"cip":"ENABLED", “cipheader”:”x-forwarded-for”}' を設定します。

オンプレミスKubernetesクラスター向けMetalLBレイヤー2モードでのNetScaler CPX統合

クラウド環境では、`LoadBalancer`タイプのKubernetesサービスサポートはクラウドロードバランサーによって提供されます。クラウドサービスプロバイダーは、ロードバランサーを自動的に作成し、サービスステータスの一部として表示されるIPアドレスを割り当てることで、このサポートを有効にします。外部IPアドレス宛てのトラフィックはすべて、クラウドロードバランサーによってNodeIPとNodePortでロードバランシングされます。

NetScalerは、オンプレミス環境で`LoadBalancer`タイプのサービスをサポートするためのさまざまなオプションを提供します。

- 外部のNetScaler VPXまたはNetScaler MPXをティア1ロードバランサーとして使用し、Kubernetesサービスへの受信トラフィックをロードバランシングする。

このようなデプロイメントの詳細については、LoadBalancerタイプのサービスを公開するを参照してください。

- Kubernetesクラスター内で実行されているNetScaler CPXデーモンセットと、BGP経由のECMPをサポートするルーターを使用して、Kubernetesクラスター内で実行されているアプリケーションを公開する。ECMPルーターは、複数のNetScaler CPXインスタンスにトラフィックをロードバランシングします。NetScaler CPXインスタンスは、実際のアプリケーションポッドをロードバランシングします。このようなデプロイメントの詳細については、NetScaler CPXを使用したLoadBalancerタイプのサービスおよびIngressの外部IPアドレスのBGPアドバタイズを参照してください。
- NetScaler CPXサービスを、ノードの外部IPアドレスを持つ外部IPサービスとして公開する。ティア1として外部ADCが実現不可能で、BGPルーターが存在しない場合に、このオプションを使用できます。このデプロイメントでは、Kubernetesはサービスポート上のNetScaler CPXサービスの`spec.externalIP`に来るトラフィックをNetScaler CPXポッドにルーティングします。Ingressリソースは、NetScaler Ingress Controllerを使用して、SSL（Secure Sockets

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Layer）オフロードとアプリケーションのロードバランシングを実行するように構成できます。ただし、このデプロイメントには、ノード障害が発生した場合に信頼性が低いという大きな欠点があります。

- NetScaler CPXと連携してレイヤー2モードでベアメタルKubernetesクラスター用のロードバランサー実装であるMetalLBを使用し、Ingress機能を実現する。

このドキュメントでは、他のソリューションが実現不可能な場合に、MetalLBとNetScaler CPXを連携させてベアメタルクラスターでIngress機能を実現する方法を示します。レイヤー2モードのMetalLBは、すべてのトラフィックをNetScaler CPXサービスに送信するように1つのノードを構成します。ノード障害が発生した場合、MetalLBはIPアドレスを別のノードに自動的に移動します。これにより、ExternalIPサービスよりも優れた信頼性が提供されます。

注: MetalLBはまだベータ版です。プロジェクトの成熟度と制限事項については、公式ドキュメントを参照してください。

オンプレミスKubernetesクラスター向けに、レイヤー2モードでMetalLBとのNetScaler CPX統合をデプロイするには、次の手順を実行します。

1. MetalLBをインストールして構成する
2. レイヤー2用のMetalLB構成を構成する
3. NetScaler CPXサービスをインストールする

MetalLBをインストールして構成する

まず、MetalLBをレイヤー2モードでインストールする必要があります。MetalLBのさまざまなインストールタイプに関する詳細については、MetalLB documentationを参照してください。

MetalLBをインストールするには、以下の手順を実行します。

1. MetalLBをデプロイするための名前空間を作成します。

```
kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/
v0.9.5/manifests/namespace.yaml
```

2. 以下のコマンドを使用してMetalLBをデプロイします。

```
kubectl apply -f https://raw.githubusercontent.com/metallb/metallb/
v0.9.5/manifests/metallb.yaml
```

3. 初めてインストールを実行する場合は、以下の手順を実行します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl create secret generic -n metallb-system memberlist --from-
literal=secretkey="$(openssl rand -base64 128)"
```

4. 以下のコマンドを使用して、MetalLBのインストールを確認し、スピーカーとコントローラーが実行状態であることを確認します。

```
kubectl get pods -n metallb-system
```

これらの手順により、MetalLBは`metallb-system`名前空間の下にクラスターにデプロイされます。

MetalLBデプロイメントYAMLファイルには、以下のコンポーネントが含まれています。

- metallb-system/controllerデプロイメント: このコンポーネントは、IPアドレスの割り当てを処理するクラスター全体のコントローラーです。
- metallb-system/speakerデーモンセット: このコンポーネントは、選択したプロトコルを使用して通信し、サービスに到達可能にします。
- コントローラーとスピーカーのサービスアカウント、およびコンポーネントが機能するために必要なRBAC権限。

レイヤー2のMetalLB構成

MetalLBがインストールされたら、レイヤー2モード用にMetalLBを構成する必要があります。MetalLBは、タイプLoadBalancerサービスに外部IPとして割り当てられるIPアドレスの範囲を受け取ります。このデプロイメントでは、NetScaler CPXサービスが他のすべてのアプリケーションのフロントエンドとして機能します。したがって、単一のIPアドレスで十分です。

以下のコマンドを使用してMetalLBのConfigMapを作成します。ここで、metallb-config.yamlはMetalLB構成を含むYAMLファイルです。

```
kubectl create –f metallb-config.yaml
```

以下は、レイヤー2モードのMetalLB構成のサンプルです。この例では、192.168.1.240-192.168.1.240がIPアドレス範囲として指定されています。

```
apiVersion: v1
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kind: ConfigMap

metadata:

  namespace: metallb-system

  name: config

data:

  config: |

    address-pools:

    - name: default

      protocol: layer2

      addresses:

      - 192.168.1.240-192.168.1.240
```

NetScaler CPX サービスインストール

MetalLB が正常にインストールされたら、NetScaler CPX デプロイメントと `LoadBalancer` タイプのサービスをインストールできます。

NetScaler CPX をインストールするには、YAML ファイルまたは Helm チャートを使用できます。

YAML ファイルを使用して NetScaler CPX をインストールするには、次の手順を実行します。

1. NetScaler CPX デプロイメントマニフェストをダウンロードします。

   ```
   wget https://github.com/citrix/citrix-k8s-ingress-controller/blob/
   master/deployment/baremetal/citrix-k8s-cpx-ingress.yml
   ```

2. NetScaler CPX デプロイメント YAML を編集します。

   - 必要に応じてレプリカ数を設定します。高可用性を実現するには、複数のレプリカを持つ方が適切です。
   - サービスタイプを `LoadBalancer` に変更します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

3. 編集した YAML ファイルを Kubectl コマンドを使用して適用します。

```
kubectl apply –f citrix-k8s-cpx-ingress.yaml
```

4. 次のコマンドを使用してサービスを表示します。

```
kubectl get svc cpx-service -output yaml
```

MetalLB が NetScaler CPX サービスに外部 IP アドレスを次のように割り当てていることがわかります。

```
apiVersion: v1
kind: Service
metadata:
  name: cpx-service
  namespace: default
spec:
  clusterIP: 10.107.136.241
  externalTrafficPolicy: Cluster
  healthCheckNodePort: 31916
  ports:
  - name: http
    nodePort: 31528
    port: 80
    protocol: TCP
    targetPort: 80
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    - name: https
      nodePort: 31137
      port: 443
      protocol: TCP
      targetPort: 443
    selector:
      app: cpx-ingress
    sessionAffinity: None
    type: LoadBalancer
  status:
    loadBalancer:
      ingress:
      - ip: 192.168.1.240
```

サンプルアプリケーションをデプロイする

サンプルアプリケーションをデプロイし、デプロイメントを検証するには、次の手順を実行します。

1. sample-deployment.yaml ファイルを使用してサンプルデプロイメントを作成します。

   ```
   kubectl create –f sample-deployment.yaml
   ```

2. sample-service.yaml ファイルを使用して、サービスでアプリケーションを公開します。

   ```
   kubectl create –f sample-service.yaml
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

3. サービスが作成されたら、sample-ingress.yamlを使用してイングレスリソースを追加できます。

```
kubectl create –f sample-ingress.yaml
```

次のように`cpx-service`外部IPアドレスを使用してアプリケーションにアクセスすることで、イングレスをテストできます。

```
curl -v http://192.168.1.240 -H ‘host: testdomain.com’
```

その他のリファレンス

MetalLBの構成とトラブルシューティングの詳細については、次のリンクを参照してください。

- Metal LBのトラブルシューティング
- レイヤー2モードでのMetal LBのルーティングの構成

Kubernetes IngressのHTTPRoute CRDを使用した高度なコンテンツルーティング

KubernetesネイティブのIngressは、NetScaler Ingress Controllerでサポートされている基本的なホストおよびパスベースのルーティングを提供します。 NetScalerは、高度なルーティング機能をサポートするために、コンテンツルーティングCRDを使用する代替アプローチも提供します。コンテンツルーティングCRDには、Listener CRDとHTTPRoute CRDが含まれます。これらのCRDは、正規表現ベースの式や、クエリパラメータ、Cookie、HTTPヘッダー、その他のNetScalerカスタム式に基づくコンテンツスイッチングなど、高度なコンテンツルーティング機能を提供します。

Ingressバージョン`networking.k8s.io/v1`では、Kubernetesはリソースバックエンドのサポートを導入しています。リソースバックエンドは、Ingressオブジェクトと同じ名前空間内の別のKubernetesリソースへの`ObjectRef`です。

現在、NetScalerはIngressのリソースバックエンドとしてHTTPルートCRDリソースの設定をサポートしています。デフォルトでは、Ingressはパスおよびホストベースのルーティングのような限られたコンテンツルーティング機能のみをサポートしています。この機能により、Ingressに高度なコンテンツルーティング機能を拡張し、さまざまなコンテンツスイッチングオプションを設定できます。特定のドメインの場合、Kubernetes Ingress APIのサードパーティ互換性サポートを

© 1997–2026 Citrix Systems, Inc. All rights reserved.

失うことなく、`HTTPRoute`カスタムリソースを使用してコンテンツスイッチングを設定できます。

注:

- この機能は、Kubernetes 1.19以降のバージョンで利用可能なKubernetes Ingressバージョン`networking.k8s.io/v1`をサポートしています。
- 同じドメインでIngressパスルーティングと`HTTPRoute`が使用されている場合、`HTTPRoute`リソースからのすべてのコンテンツルーティングポリシーは、Ingressベースのコンテンツルーティングポリシーよりも優先度が低くなります。したがって、高度なコンテンツルーティングが必要な場合は、特定のドメインに対して`HTTPRoute`リソースのすべてのコンテンツスイッチングポリシーを設定することをお勧めします。

HTTPRoute CRDを使用してKubernetes Ingressの高度なコンテンツルーティングを設定する

この手順では、高度なコンテンツルーティングをサポートするために、HTTPRouteリソースをリソースバックエンドとしてデプロイする方法を示します。

前提条件

- KubernetesクラスターでIngress APIバージョン`networking.k8s.io/v1`が利用可能であることを確認してください。
- HTTPRoute CRDがデプロイされていることを確認してください。

Ingressリソースをデプロイする

YAMLファイルで、リソースバックエンドが`HTTPRoute`カスタムリソースを指すようにIngressリソースを定義します。証明書、フロントエンドプロファイル、フロントエンドIPアドレス、Ingressクラスなど、すべてのフロントエンド設定をIngressリソースの一部として指定します。

以下は、`sample-ingress.yaml`という名前のIngressリソースのサンプルです。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: kuard-ingress
  annotations:
    ingress.citrix.com/frontend-ip: "x.x.x.x"
    ingress.citrix.com/insecure-termination: "redirect"
spec:
  ingressClassName: citrix
  tls:
  - secretName: web-ingress-secret
  rules:
  - host: kuard.example.com
    http:
      paths:
      - pathType: ImplementationSpecific
        backend:
          resource:
            apiGroup: citrix.com
            kind: HTTPRoute
            name: kuard-example-route
```

YAMLファイルでIngressリソースを定義した後、次のコマンドを使用してYAMLファイルをデプロイします。ここで、`sample-ingress.yaml`はYAMLファイルの定義です。

```
kubectl apply -f sample-ingress.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

この例では、ドメイン`kuard.example.com`のコンテンツスイッチングポリシーは、`kuard-example-route`という名前の`HTTPRoute`カスタムリソースの一部として定義されています。`Certificates`、`frontend-ip`、および`ingress class`はIngressリソースの一部として指定されています。ロードバランシング方法やサービスグループ構成などのバックエンドアノテーションは、`HTTPRoute`カスタムリソースの一部として指定されています。

HTTPRouteリソースをデプロイする

YAMLファイルでHTTPルート構成を定義します。YAMLファイルでは、`kind`フィールドでHTTPRouteを使用し、`spec`セクションでHTTPルート構成の要件に基づいて`HTTPRoute` CRD属性を追加します。

APIの説明と例の詳細については、HTTPRouteドキュメントを参照してください。

以下は、`HTTPRoute`リソース構成のサンプルです。この例は、さまざまなKubernetesバックエンドマイクロサービスに対してクエリパラメータベースのコンテンツスイッチングを使用する方法を示しています。

```
apiVersion: citrix.com/v1

kind: HTTPRoute

metadata:

   name: kuard-example-route

spec:

  hostname:

  - kuard.example.com

  rules:

  - name: kuard-blue

    match:

    - queryParams:

      - name: version
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        contains: v2
    action:
      backend:
        kube:
          service: kuard-blue
          port: 80
  - name: kuard-green
    match:
    - queryParams:
      - name: version
        contains: v3
    action:
      backend:
        kube:
          service: kuard-green
          port: 80
  - name: kuard-default
    match:
    - path:
       prefix: /
    action:
      backend:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        kube:

          service: kuard-purple

          port: 80
```

YAMLファイルでHTTPルートを定義したら、YAMLファイルをデプロイします。この例では、`httproute`がYAML定義です。

```
kubectl apply -f httproute.yaml
```

Listener CRDのプロファイルサポート

Listener CRDのHTTP、TCP、SSLをそれぞれ設定するために、HTTPプロファイル、TCPプロファイル、SSLプロファイルなどの個々のエンティティを使用できます。Listener CRDのプロファイルサポートは、デフォルトのプロトコル動作をカスタマイズするのに役立ちます。SSL仮想サーバーのSSL暗号を選択することもできます。

HTTPプロファイル

HTTPプロファイルは、HTTP設定の集合です。`nshttp_default_profile`という名前のデフォルトHTTPプロファイルが、HTTP設定を行うために構成されています。これらの設定は、デフォルトで、すべてのサービスと仮想サーバーにグローバルに適用されます。
`spec.policies.httpprofile`を指定することで、Listenerリソースの HTTP設定をカスタマイズできます。指定された場合、NetScaler Ingress Controllerは、デフォルトHTTPプロファイルから派生したデフォルト値を持つ新しいHTTPプロファイルを作成し、指定された値を構成します。

これは、デフォルトHTTPプロファイルからデフォルト値を派生させ、指定された値を構成するのに役立ちます。

次のYAML例は、特定のフロントエンド仮想サーバーでWebSocketを有効にする方法を示しています。

```
apiVersion: citrix.com/v1

kind: Listener

metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: test-listener
  namespace: default
spec:
  vip: x.x.x.x
  port: 80
  protocol: http
  policies:
    httpprofile:
      config:
        websocket: "ENABLED"
```

HTTPプロファイルのすべての可能なキーと値のペアについては、HTTPプロファイルを参照してください。

注:

「name」は自動生成されます。

次の例に示すように、組み込みのHTTPプロファイルまたは事前設定されたHTTPプロファイルを指定し、それをフロントエンド仮想サーバーにバインドすることもできます。

```
apiVersion: citrix.com/v1
kind: Listener
metadata:
  name: test-listener
  namespace: default
spec:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  vip: x.x.x.x

  port: 80

  protocol: http

  policies:

    httpprofile:

      preconfigured: 'nshttp_default_strict_validation'
```

TCPプロファイル

TCPプロファイルは、TCP設定の集合です。`nstcp_default_profile`という名前のデフォルトTCPプロファイルが、TCP設定を行うために構成されています。これらの設定は、デフォルトで、すべてのサービスと仮想サーバーにグローバルに適用されます。`spec.policies.tcpprofile`を指定することで、TCP設定をカスタマイズできます。`spec.policies.tcpprofile`を指定すると、NetScaler Ingress Controllerは、デフォルトTCPプロファイルから派生したTCPプロファイルを作成し、仕様で提供された値を適用し、それをフロントエンド仮想サーバーにバインドします。

TCPプロファイルのすべての可能なキーと値のペアについては、TCPプロファイルを参照してください。

注:

名前は自動生成されます。

次の例は、フロントエンド仮想サーバーに対して`tcpfastopen`と`HyStart`を有効にする方法を示しています。

```
apiVersion: citrix.com/v1

kind: Listener

metadata:

  name: test-listener
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  namespace: default
spec:
  vip: x.x.x.x
  port: 80
  protocol: http
  policies:
    tcpprofile:
      config:
        tcpfastopen: "ENABLED"
        hystart: "ENABLED"
```

次の例に示すように、組み込みのTCPプロファイルまたは事前設定されたTCPプロファイル名を指定することもできます。

```
apiVersion: citrix.com/v1
kind: Listener
metadata:
  name: test-listener
  namespace: default
spec:
  vip: x.x.x.x
  port: 80
  protocol: http
  policies:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    tcpprofile:

      preconfigured: 'nstcp_default_Mobile_profile'
```

SSLプロファイル

SSLプロファイルは、SSLエンティティの設定の集合です。SSLプロファイルにより、設定がより簡単かつ柔軟になります。各エンティティで設定を行う代わりに、プロファイルで設定を構成し、そのプロファイルを仮想サーバーにバインドできます。SSLプロファイルを使用すると、TLSプロトコルや暗号などの多くのSSLパラメータをカスタマイズできます。SSLプロファイルの詳細については、SSLプロファイルインフラストラクチャを参照してください。

注:

デフォルトでは、NetScalerはレガシーSSLプロファイルを作成します。レガシーSSLプロファイルには、SSLv3などの高度なプロトコルをサポートしないなど、多くの欠点があります。したがって、NetScaler Ingress Controllerを起動する前に、NetScalerでデフォルトのSSLプロファイルを有効にすることをお勧めします。

高度なSSLプロファイルを有効にするには、NetScalerコマンドラインで次のコマンドを使用します。

SSLパラメータを設定 -defaultProfile ENABLED

このコマンドは、既存のすべてのSSL仮想サーバーとSSLサービスグループに対してデフォルトのSSLプロファイルを有効にします。

SSLプロファイルをカスタマイズするために`spec.policies.sslprofile`を指定できます。指定すると、NetScaler Ingress ControllerはデフォルトのSSLフロントエンドプロファイル`ns_default_ssl_profile_frontend`から派生したSSLプロファイルを作成します。

SSLプロファイルでサポートされているキーと値のペアについては、SSLプロファイルを参照してください。

注:

`name`は自動生成されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

次の例は、フロントエンド仮想サーバーに対してTLS1.3とHSTSを有効にする方法を示しています。

```
apiVersion: citrix.com/v1

kind: Listener

metadata:

  name: test-listener

  namespace: default

spec:

  vip: x.x.x.x

  port: 443

  certificates:

  - secret:

      name: my-cert

  protocol: https

  policies:

    sslprofile:

      config:

        tls13: "ENABLED"

        hsts: "ENABLED"
```

次の例に示すように、組み込みまたは事前設定されたSSLプロファイル名を指定できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: citrix.com/v1

kind: Listener

metadata:

  name: test-listener

  namespace: default

spec:

  vip: x.x.x.x

  port: 443

  certificates:

  - secret:

      name: my-cert

  protocol: https

  policies:

    sslprofile:

      preconfigured: 'ns_default_ssl_profile_secure_frontend'
```

SSL暗号

Ingress NetScalerには、組み込みの暗号グループがあります。デフォルトでは、仮想サーバーはSSLトランザクションにDEFAULT暗号グループを使用します。DEFAULT暗号グループの一部ではない暗号を使用するには、それらをSSLプロファイルに明示的にバインドする必要があります。`spec.policies.sslciphers`を使用して、暗号のリスト、組み込みの暗号グループのリスト、またはユーザー定義の暗号グループのリストを提供できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

暗号の優先順位は、リストで定義されている順序と同じです。リストの最初のものが最初の優先順位を得て、同様に続きます。

次の例は、組み込みの暗号スイートのリストを提供する方法を示しています。

```
apiVersion: citrix.com/v1

kind: Listener

metadata:

  name: test-listener

  namespace: default

spec:

  vip: x.x.x.x

  port: 443

  certificates:

  - secret:

      name: my-cert

  protocol: https

  policies:

    sslciphers:

    - 'TLS1.2-ECDHE-RSA-AES128-GCM-SHA256'

    - 'TLS1.2-ECDHE-RSA-AES256-GCM-SHA384'

    - 'TLS1.2-ECDHE-RSA-AES-128-SHA256'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    - 'TLS1.2-ECDHE-RSA-AES-256-SHA384'
```

NetScalerで利用可能な暗号スイートのリストについては、SSLプロファイルインフラストラクチャを参照してください。

ユーザー定義の暗号グループを使用するには、NetScalerにユーザー定義の暗号グループがあることを確認してください。ユーザー定義の暗号グループを設定するには、次の手順を実行します。

1. ユーザー定義の暗号グループを作成します。例: `MY-CUSTOM-GROUP`。
2. 必要なすべての暗号をユーザー定義の暗号グループにバインドします。
3. ユーザー定義の暗号グループ名を控えておきます。

詳細な手順については、ユーザー定義の暗号グループを構成するを参照してください。

注: 暗号の優先順位は、リストで定義されている順序と同じです。リストの最初のものが最初の優先順位を得て、同様に続きます。

次の例は、組み込みの暗号グループおよび/またはユーザー定義の暗号グループのリストを提供する方法を示しています。ユーザー定義の暗号グループは、Listenerに適用する前にNetScalerに存在している必要があります。

```
apiVersion: citrix.com/v1

kind: Listener

metadata:

  name: test-listener

  namespace: default

spec:

  vip: x.x.x.x

  port: 443

  certificates:

  - secret:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      name: my-cert
  protocol: https
  policies:
    sslciphers:
    - 'SECURE'
    - 'HIGH'
    - 'MY-CUSTOM-CIPHERS'
```

前の例では、`SECURE`と`HIGH`はNetScalerの組み込みの暗号グループです。`MY-CUSTOM-CIPHERS`は事前設定されたユーザー定義の暗号グループです。

注: 事前設定されたSSLプロファイルを指定した場合、NetScalerを介して暗号を手動でバインドする必要があり、`spec.policies.sslciphers`は事前設定されたSSLプロファイルには適用されません。

注記: 組み込みの暗号グループは、Tier-1およびTier-2 NetScalerで使用できます。ユーザー定義の暗号グループは、Tier-1 NetScalerでのみ使用できます。

アナリティクスプロファイル

アナリティクスプロファイルを使用すると、NetScalerはトランザクションまたはデータの種類を外部プラットフォームにエクスポートできます。NetScaler Observability Exporter を使用してメトリックとトランザクションデータを収集し、ElasticsearchやPrometheusなどのエンドポイントにエクスポートする場合、アナリティクスプロファイルを構成して、エクスポートする必要があるデータの種類を選択できます。

注:

アナリティクスプロファイルが機能するには、NetScaler Observability Exporterを構成する必要があります。ConfigMapを使用したアナリティクス構成のサポート。

次の例は、アナリティクスプロファイルで`webinsight`と`tcpinsight`を有効にする方法を示しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: citrix.com/v1

  kind: Listener

  metadata:

    name: test-listener

    namespace: default

  spec:

    vip: x.x.x.x

    port: 443

    certificates:

    - secret:

        name: my-cert

    protocol: https

    policies:

     analyticsprofile:

       config:

       - type: webinsight

       - type: tcpinsight
```

次の例は、NetScaler Observability Exporterにエクスポートする `webinsight` の種類の追加パラメーターを選択する方法を示しています。有効なキーと値のペアについては、アナリティクスプロファイル を参照してください。

```
apiVersion: citrix.com/v1

  kind: Listener
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  metadata:
    name: test-listener
    namespace: default
  spec:
    vip: x.x.x.x
    port: 443
    certificates:
    - secret:
        name: my-cert
    protocol: https
    policies:
     analyticsprofile:
       config:
       - type: webinsight
         parameters:
           httpdomainname: "ENABLED"
           httplocation: "ENABLED"
```

次の例は、事前構成されたアナリティクスプロファイルを使用する方法を示しています。

```
apiVersion: citrix.com/v1
kind: Listener
metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: test-listener
  namespace: default
spec:
  vip: x.x.x.x
  port: 443
  certificates:
  - secret:
      name: my-cert
  protocol: https
  policies:
   analyticsprofile:
     preconfigured:
     - 'custom-websingiht-analytics-profile'
     - 'custom-tcpinsight-analytics-profile'
```

IPAMコントローラーを使用したIPアドレス管理

IPAMコントローラーは、NetScalerが提供するIPアドレス管理用のアプリケーションであり、Kubernetesクラスター内でNetScaler Ingress Controllerと並行して実行されます。IPAMコントローラーを使用すると、指定されたIPアドレス範囲から、イングレスリソース、LoadBalancerタイプのサービス、およびリスナーリソースにIPアドレスを割り当てることができます。NetScaler Ingress Controllerは、サービス、イングレス、またはリスナーリソースに割り当てられたIPアドレスを、NetScaler MPXまたはNetScaler VPXの仮想IPアドレス（VIP）として構成します。

IPAMコントローラーには、NetScalerが提供するVIP CustomResourceDefinition（CRD）が必要です。VIP CRDは、NetScaler Ingress ControllerとIPAMコントローラー間の内部通信に使用されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

IPAMコントローラーとInfobloxの統合

Infoblox統合により、IPAMコントローラーはInfobloxからLoadBalancerタイプのサービス、イングレス、またはリスナーリソースにIPアドレスを割り当てます。

イングレスリソース、リスナーリソース、およびLoadBalancerタイプのサービスでそれぞれサポートされているアノテーションは次のとおりです。

- `ingress.citrix.com/ipam-range`
- `listeners.citrix.com/ipam-range`
- `service.citrix.com/ipam-range`

これらのアノテーションを使用して、IPAMコントローラーで定義されたIPアドレス範囲をそれぞれのリソースに提供する必要があります。IPアドレス範囲は、IPAM VIP範囲またはInfoblox VIP範囲のいずれかである必要があります。

注:

- Infoblox統合は、IPAM Controllerバージョン2.2.10以降、バージョン2.12.3までサポートされています。
- Infobloxを使用してIPAMによって割り当てられたIPアドレスを更新または削除してはなりません。

IPAMコントローラーは、マルチクラスター設定において次の課題を提起します。

- IPアドレスは手動で提供する必要があります。
- 複数のクラスターにデプロイされた各アプリケーションには、共通のIPアドレスを割り当てる必要があります。
- 他のクラスターのIPAMコントローラー間でIPアドレス範囲を共有することに制限があり、タスクが複雑になります。

InfobloxとIPAMの統合は、以下のタスクに役立ちます。

- Infobloxネットワーク管理を使用して、指定されたIPアドレス範囲から利用可能なIPアドレスを割り当てます。
- ドメイン名に関連付けられたIPアドレスを要求し、既存のIPアドレスの取得を確実にします。
- マルチクラスターイングレス設定では、さまざまなクラスターにデプロイされたアプリケーションに、単一の一貫したIPアドレスを使用してアクセスできます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

初期化中に、IPAMコントローラーはInfobloxに接続し、まだ利用可能でない場合はネットワークビューを作成します。このネットワークビューの作成は、提供されたDNSビュー内のネットワーク範囲に基づいています。DNSビュー内のネットワーク範囲が提供されていない場合、IPAMコントローラーはネットワークビューを作成します。

IPAMコントローラーを使用してIPアドレスを割り当てる

IPAMコントローラーからのIPアドレスを使用して、イングレスリソース、リスナーリソース、またはLoadBalancerタイプのサービスを構成するには、次の手順を実行します。

1. VIP CRDを展開する
2. NetScaler Ingress Controllerを展開する
3. IPAMコントローラーを展開する
4. アプリケーションとイングレスリソースを展開する

ステップ1: VIP CRDを展開する

NetScaler Ingress ControllerとIPAMコントローラー間の通信を可能にするNetScaler VIP CRDを展開するには、次の手順を実行します。

```
kubectl create -f https://raw.githubusercontent.com/netscaler/netscaler-k8s-ingress-controller/master/crd/vip/vip.yaml
```

VIP CRD の詳細については、VIP CustomResourceDefinition を参照してください。

ステップ 2: NetScaler Ingress Controller を展開する

前提条件

- Kubernetes クラスターと、クラスターと通信するための kubectl コマンドラインツール。
- 次のコマンドを使用して、NetScaler VPX または NetScaler MPX の資格情報でシークレットを作成します。

  ```
  kubectl create secret  generic nslogin --from-literal=username=<username> --from-literal=password=<password>
  ```

- 次のコマンドを使用して、NetScaler Helm チャートリポジトリを追加します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/
```

NSIC を展開する

1. 次のコマンドを使用して、NetScaler Helm チャートリポジトリを更新します。

```
helm repo update netscaler
```

2. 以下に示すように、values.yaml を更新して NetScaler Ingress Controller を構成します。

   サンプル `values.yaml`:

```
license:

  accept: yes

adcCredentialSecret: nslogin # K8s Secret Name

nsIP: <x.x.x> # CLIP (for appliances in Cluster mode), SNIP (for appliances in High Availability mode) , NSIP (for standalone appliances)

openshift: false # set to true for OpenShift deployments

entityPrefix: cluster1 # unique for each NSIC instance.

ipam: true

ingressClass: ['cic-vpx'] # ingress class used in the ingress resources

serviceClass: ['netscaler'] # To use service type LB, specify the service class
```

3. 次の Helm コマンドを使用して、NetScaler VPX または NetScaler 用の NetScaler Ingress Controller をインストールします。

```
helm install nsic netscaler/netscaler-ingress-controller -f values.yaml
```

Helm チャートを使用した NetScaler Ingress Controller の展開と構成の詳細については、Helm チャートリポジトリ を参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ステップ 3: IPAM コントローラーを展開する

前提条件

- Infoblox 統合の場合、次の権限を持つユーザーロールを Infoblox で作成していることを確認してください。
  - **IPアドレス管理**:
    - IPアドレスの割り当てと管理。
    - ホストレコードの作成、更新、削除。
  - **拡張可能属性管理**:
    - 拡張可能属性の作成、更新、削除。
- Infobloxとの統合のために、次のコマンドを実行してInfobloxユーザー認証情報を含むKubernetesシークレットを作成します。

```
kubectl create secret  generic infobloxlogin --from-
literal=username=<Infoblox_username> --from-
literal=password=<Infoblox_password>
```

この手順では、IPAMコントローラーを展開し、InfobloxをIPAMコントローラーと統合します。

1. 次のコマンドを使用して、NetScaler Helmチャートリポジトリをローカルレジストリに追加します。

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-
charts/
```

2. 次のコマンドを使用して、NetScaler IPAMコントローラーをインストールします。

```
helm install netscaler-ipam-controller netscaler/netscaler-ipam-
controller -f values.yaml
```

サンプル `values.yaml`:

```
cluster: 'cluster1' # enables you to know in which cluster IPAM is
installed in multi-cluster solution

vipRange: "[{\"range1\": [\"1.2.1.51-1.2.1.59\"]\,\"range2\":
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
[\"1.2.1.61-1.2.1.69\"]}]" # range1 has IPs from 1.2.1.51 to 1.2.1.59 and
range2 has IPs from 1.2.1.61 to 1.2.1.69

reuseIngressVip: "True"  # set to False to assigns  different IP address
to each ingress resource

infoblox:

  enabled: True #set to false to disable Infoblox integration with IPAM
controller

  credentialSecret: <secret> # Kubernetes secret created using Infoblox
user credentials

  gridHost: <x.x.x.x>  # Infoblox grid host IP or FQDN

  vipRange: "[{"infoblox-range": ["1.1.1.0/24"]}]"  # Infoblox IPAM VIP
Range, which is different from the vipRange mentioned above. This has to
be in CIDR format like 1.1.1.0/24
```

次の表は、IPAMコントローラーのインストール時に設定可能なすべてのパラメーターに関する情報を示しています。

パラメーター	必須またはオプション	デフォルト値	説明
imageRegistry	必須	quay.io	NetScaler IPAM Controller のイメージレジストリ。
imageRepository	必須	netscaler/netscaler-ipam-controller	NetScaler IPAM Controller のイメージリポジトリ。
imageTag	必須	2.1.2	NetScaler IPAM Controller のイメージタグ。
pullPolicy	必須	`IfNotPresent`	NetScaler IPAM Controller のイメージプルポリシー。
vipRange	必須	N/A	この変数を使用すると、IPアドレス範囲

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須またはオプション	デフォルト値	説明
			を定義できます。IPアドレス範囲を定義するか、一意の名前に関連付けられたIPアドレス範囲を定義できます。NetScaler IPAMコントローラーは、このIPアドレス範囲から `LoadBalancer` タイプのサービスにIPアドレスを割り当てます。
reuseIngressVip	オプション	真	この変数を使用すると、同じ `vipRange` を使用するすべてのイングレスで同じIPを使用できます。
クラスター	`infoblox.enabled` がTrueの場合、必須	N/A	この変数を使用すると、IPAMコントローラーがデプロイされているクラスターを識別するために使用されるクラスター名を提供できます。
infoblox.enabled	オプション	偽	Infoblox IPAM を有効または無効にできるブール値。
infoblox.gridHost	`infoblox.enabled` が true の場合、必須	N/A	クラスターから到達可能である必要がある Infoblox グリッドホストの IP または FQDN。
infoblox.credentialSecret	`infoblox.enabled` が true の場合、必須	N/A	前提条件に記載されている必要なアクセス権限を持つ Infoblox のユーザー

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須またはオプション	デフォルト値	説明
			名とパスワードを持つ、同じ名前空間にある Kubernetes Secret。
infoblox.httpTimeout	オプション	10	この変数を使用すると、Infoblox クライアントの HTTP タイムアウトを秒単位で指定できます。
infoblox.maxRetries	オプション	3	この変数を使用すると、障害発生時の Infobloxクライアントの最大再試行回数を指定できます。
infoblox.netView	オプション	デフォルト	この変数を使用すると、Infobloxネットワークビューを指定できます。ネットワークビューが存在しない場合は、作成されます。
infoblox.vipRange	infoblox.enabled が true の場合、必須	N/A	コントローラーによって管理される Infoblox IPAM仮想IPアドレス範囲。この範囲内のすべてのIPアドレスは、コントローラーによって Infobloxで管理され、他のプロセスと競合してはなりません。IP範囲はCIDR形式 a.b.c.d/n である必要があります。例: "[{"infoblox-range": ["1.1.1.0/24"]}]"。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメーター	必須またはオプション	デフォルト値	説明
serviceAccount.create	必須	はい	この変数は、NetScaler IPAMコントローラーのサービスアカウントを作成します。
serviceAccount.tokenExpirationSeconds	必須	31536000	サービスアカウントのトークンの有効期限が切れるまでの時間（秒単位）。
serviceAccount.name	オプション	""	IPAM Controller のサービスアカウント名。すでに作成済みで自分で管理しているサービスアカウントを使用する場合は、ここにその名前を指定し、`serviceAccount.create` を false に設定します。

ステップ4：Ingressリソースを展開する

サンプルアプリケーションとIngressリソースを展開するには、次の手順を実行します。

1. 次のコマンドを使用してCNNアプリケーションを展開します。

```
kubectl apply -f - <<EOF

apiVersion: apps/v1

kind: Deployment

metadata:

  name: cnn-website

  labels:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    name: cnn-website
    app: cnn-website
spec:
  selector:
    matchLabels:
      app: cnn-website
  replicas: 2
  template:
    metadata:
      labels:
        name: cnn-website
        app: cnn-website
    spec:
      containers:
      - name: cnn-website
        image: quay.io/sample-apps/cnn-website:v1.0.0
        ports:
        - name: http-80
          containerPort: 80
        - name: https-443
          containerPort: 443
---
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: v1
kind: Service
metadata:
  name: cnn-website
  labels:
    app: cnn-website
spec:
  type: NodePort
  ports:
  - name: http-80
    port: 80
    targetPort: 80
  - name: https-443
    port: 443
    targetPort: 443
  selector:
    name: cnn-website
EOF
```

2. 次のコマンドを使用して、CNNアプリケーションにトラフィックを送信するIngressリソースを展開します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: cnn-ingress

  annotations:

    ingress.citrix.com/ipam-range: "<IPAM-vip-range or Infoblox-vip-
range>"

spec:

  ingressClassName: cic-vpx

  rules:

  - host: www.cnn.com

    http:

      paths:

      - path: /

        pathType: Prefix

        backend:

          service:

            name: cnn-website

            port:

              number: 80

EOF
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

上記のIngressの例では、IP割り当ての順序は次のとおりです。

- デフォルトの `NS_VIP` 環境変数が指定されている場合、NetScaler Ingress Controller は、Ingressで範囲名 (`ingress.citrix.com/ipam-range:`) が指定されている場合にのみIPAMコントローラーに要求を行います。アノテーションが指定されていない場合は、そのIngressに `NS_VIP` が使用されます。
- デフォルトの `NS_VIP` 環境変数が指定されていない場合、NetScaler Ingress Controller は常にIPAMコントローラーにIP割り当てを要求します。

IPアドレスの割り当て

- LoadBalancerタイプのサービスの場合、VIP範囲またはInfoblox VIP範囲から各サービスに一意のIPアドレスが割り当てられます。
- イングレスリソースの場合、指定されたIP範囲内のIPアドレスが割り当てられます。複数のイングレスリソースが同じVIP範囲を参照する場合、最初のイングレスリソースに割り当てられたIPアドレスが、他のすべてのイングレスリソースに割り当てられます。
- LoadBalancerタイプのサービス、イングレスリソース、およびリスナーリソースは、NetScaler IPAMコントローラーを同時に使用してIPアドレスを割り当てることができます。いずれかのリソースタイプにIPアドレスが割り当てられている場合、他のリソースタイプでは使用できません。ただし、同じIPアドレスを複数のイングレスリソースに割り当てることができます。

一意の名前に関連付けられたIPアドレス範囲

IPアドレス範囲に一意の名前を割り当て、 `VIP_RANGE` 環境変数でその範囲を定義できます。このIPアドレス範囲の割り当て方法により、IPアドレス範囲を区別できます。 `LoadBalancer` タイプのサービスを作成する際、サービス定義で `service.citrix.com/ipam-range` アノテーションを使用して、IPアドレス割り当てに使用するIPアドレス範囲を指定できます。

複数のIPアドレス割り当て

イングレスリソースの場合、Helmチャートパラメーター `reuseIngressVip` を `true` として指定することで、IPアドレスを複数回割り当てることができます。これは、複数のイングレスリソースが単一のコンテンツスイッチング仮想サーバーによって処理される可能性があるためです。指定されたIP範囲に単一のIPアドレスしかない場合、それは複数回割り当てられます。また、IP範囲が複数のIPアドレスで構成されている場合、そのうちの1つだけが繰り返し割り当てられます。

複数の割り当てを容易にするために、IPAMコントローラーは割り当てられたIPアドレスを追跡します。IPAMコントローラーは、イングレスリソースによるそのIPアドレスのすべての割り当てが解放された場合にのみ、IPアドレスをフリープールに入れます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アノテーションを介してCRDを適用する

Rewrite and Responder、Ratelimit、Auth、WAF、BotなどのCRDを、アノテーションを使用して参照することで、イングレスリソースおよびロードバランサータイプのサービスに適用できるようになりました。この機能を使用すると、Ingressリソースに複数のサービスがある場合、要件に基づいて特定のサービスまたはすべてのサービスにリライトおよびレスポンダーポリシーを適用できます。

この機能には次の2つの利点があります。

- CRDをイングレスごと、サービスごとに適用できます。たとえば、内部VIPを介して参照される同じサービスでも、外部に公開されるサービスとは異なるリライト・レスポンダーポリシーのセットを持つ場合があります。
- 運用チームはサービス名を指定せずにCRDインスタンスを作成できます。アプリケーション開発者は、要件に基づいて適切なポリシーを選択できます。

注:

CRDインスタンスはサービス名を指定せずに作成する必要があります。

CRDを参照するためのIngressアノテーション

Ingressリソースは、`ingress.citrix.com/rewrite-responder`アノテーションを使用してRewrite and Responder CRDを直接参照できます。

アノテーションを使用してリライト・レスポンダーCRDを参照するさまざまな方法を次に示します。

- 指定されたイングレスで参照されているすべてのサービスにRewrite and Responder CRDを適用するには、次の形式を使用します。

```
ingress.citrix.com/rewrite_responder_crd: &lt;Rewritepolicy Custom-resoure-instance-name>
```

例:

```
ingress.citrix.com/rewrite_responder_crd: "blockurlpolicy"
```

この例では、指定されたイングレスで参照されているすべてのサービスにRewrite and Responderポリシーが適用されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- Ingressリソース内の指定されたKubernetesサービスにRewrite and Responder CRDを適用するには、次の形式を使用します。

  ```
  ingress.citrix.com/rewrite_responder_crd: '{&lt;Kubernetes-service-name>: &lt;Rewritepolicy Custom-resoure-instance-name>}'
  ```

  例:

  ```
  ingress.citrix.com/rewrite_responder_crd: '{"frontendsvc": "blockurlpolicy", "backendsvc": "addresponseheaders"}'
  ```

  この例では、リライトポリシー `blockurlpolicy` が `frontendsvc` サービスへのトラフィックに適用され、`addresponseheaders` ポリシーが現在のイングレスを介して来る `backendsvc` サービスに適用されます。

イングレスアノテーションを使用して、Auth、Bot、WAF、および Ratelimit CRD を適用することもできます。

次の表は、Auth、Bot、WAF、および Ratelimit CRD のアノテーションと例を説明しています。

アノテーション	例	説明
`ingress.citrix.com/bot_crd`	`ingress.citrix.com/bot_crd: '{"frontend": "botdefense"}'`	フロントエンドサービスへの受信トラフィックに `botdefense` ポリシーを適用します。
`ingress.citrix.com/auth_crd`	`ingress.citrix.com/auth_crd: '{"frontend": "authexample"}'`	フロントエンドサービスに `authexample` ポリシーを適用します。
`ingress.citrix.com/waf_crd`	`ingress.citrix.com/waf_crd: "wafbasic"`	Ingress 内のすべてのサービスに WAF ポリシー `wafbasic` を適用します。
`ingress.citrix.com/ratelimit_crd`	`ingress.citrix.com/ratelimit_crd: "throttlecoffeeperclientip"`	Ingress内のすべてのサービスにレート制限ポリシー `throttlecoffeeperclientip` を適用します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

RewriteおよびResponder CRDを参照するためのLoadBalancerタイプサービスのアノテーション

LoadBalancerタイプのサービスは、アノテーションを使用してRewriteおよびResponder CRDを参照できます。

以下はアノテーションの形式です。

```
service.citrix.com/rewrite-responder: &lt;Rewritepolicy Custom-resoure-instance-name>
```

アノテーションによるIngressのリスナーCRDサポート

Ingressは、バックエンドのKubernetesサービスへのHTTPルーティング機能を提供する標準的なKubernetesリソースです。NetScaler Ingress Controllerは、フロントエンドとバックエンドの両方の構成についてIngressパラメータを微調整するためのさまざまなアノテーションを提供します。たとえば、`ingress.citrix.com/frontend-ip`アノテーションを使用すると、NetScaler Ingress ControllerによってNetScalerに構成されたフロントエンドリスナーのIPアドレスを指定できます。同様に、HTTPおよびSSLパラメータを微調整するための他のフロントエンドアノテーションもあります。複数のIngressリソースがあり、それらがフロントエンドIPとポートを共有している場合、これらのアノテーションを各Ingressリソースに指定するのは困難です。

ネットワーク運用プロフェッショナル（NetOps）と開発者の間で責任が分かれている場合があります。NetOpsは、フロントエンドIP、証明書、SSLパラメータなどのフロントエンド構成を担当します。開発者は、HTTPルーティングとバックエンド構成を担当します。NetScaler Ingress Controllerは、すでにフロントエンド構成用のリスナーCRDやバックエンドルーティングロジック用の`HTTProute`などのコンテンツルーティングCRDを提供しています。 現在、リスナーCRDは、NetScalerが提供するアノテーションを使用してIngressリソースに適用できます。

この機能により、Ingressリソースにリスナー CRDを使用し、フロントエンド構成の作成をIngress定義から分離できます。したがって、NetOpsはリスナーリソースを個別に定義して、フロントエンドIP、証明書、およびその他のフロントエンドパラメータ（TCP、HTTP、SSL）を構成できます。各Ingressリソースを変更することなく、リスナーリソースに構成変更を適用できます。NetScalerでは、リスナーリソースはコンテンツスイッチング仮想サーバー、SSL仮想サーバー、証明書キー、およびフロントエンドのHTTP、SSL、TCPプロファイルに対応します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注記:

- Analyticsプロファイルを機能させるには、NetScaler Observability Exporterを構成する必要があります。詳細については、「ConfigMapを使用したAnalytics構成のサポート」を参照してください。
- この機能を使用する場合、同じフロントエンドIPとポートを持つすべてのIngressが同じリスナーリソースを参照していることを確認する必要があります。同じフロントエンドIPとポートの組み合わせを使用するIngressの場合、一方のIngressがリスナーリソースを参照し、もう一方のIngressが `ingress.citrix.com/frontend-ip` アノテーションを参照することはサポートされていません。

制限事項

リスナーがフロントエンド構成に使用される場合、以下のアノテーションは無視され、効果がない場合があります。

- `ingress.citrix.com/frontend-ip`
- `Ingress.citrix.com/frontend-ipset-name`
- `ingress.citrix.com/secure-port`
- `ingress.citrix.com/insecure-port`
- `ingress.citrix.com/insecure-termination`
- `ingress.citrix.com/secure-service-type`
- `ingress.citrix.com/insecure-service-type`
- `ingress.citrix.com/csvserver`
- `ingress.citrix.com/frontend-tcpprofile`
- `ingress.citrix.com/frontend-sslprofile`
- `ingress.citrix.com/frontend-httpprofile`

Ingress 用の Listener CRD リソースのデプロイ

`ingress.citrix.com/listener` アノテーションを使用して、Ingress 用の Listener リソースの名前と名前空間を `namespace/name` の形式で指定できます。Listener リソースが Ingress と同じ名前空間にある場合、名前空間は必須ではありません。

アノテーションの例を次に示します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
ingress.citrix.com/listener: default/listener1
```

ここで、`default` は Listener リソースの名前空間であり、`listener1` はフロントエンドパラメータを指定する Listener リソースの名前です。

Ingress 用の Listener リソースをデプロイするには、次の手順を実行します。

1. 次のように Listener リソース (`listener.yaml`) を作成します。

```
apiVersion: citrix.com/v1

kind: Listener

metadata:

  name: my-listener

  namespace: default

spec:

  ingressclass: citrix®

  vip: '192.168.0.1' # Virtual IP address to be used, not required when
CPX is used as ingress device

  port: 443

  protocol: https

  redirectPort: 80

  secondaryVips:

  - "10.0.0.1"

  - "1.1.1.1"

  policies:

    httpprofile:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      config:
        websocket: "ENABLED"
    tcpprofile:
      config:
        sack: "ENABLED"
    sslprofile:
      config:
        ssl3: "ENABLED"
    sslciphers:
    - SECURE
    - MEDIUM
    analyticsprofile:
      config:
      - type: webinsight
        parameters:
          allhttpheaders: "ENABLED"
    csvserverConfig:
      rhistate: 'ACTIVE'
```

ここで、デフォルトの名前空間にある Listener リソース `my-listener` は、VIP、セカンダリ VIP、HTTP プロファイル、TCP プロファイル、SSL プロファイル、SSL 暗号などのフロントエンド構成を指定します。NetScaler にポート 443 で HTTPS トラフィック用のコンテンツスイッチング仮想サーバーを作成し、ポート 80 のすべての HTTP トラフィックは HTTPS にリダイレクトされます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

NetScaler CPX が Ingress デバイスとして使用される場合、Listener リソースの `vip` フィールドは必須ではありません。NetScaler VPX の場合、VIP は NetScaler Ingress Controller によって自動的に構成されるポッド IP アドレスと同じです。

2. Listener リソースを適用します。

```
kubectl apply -f listener.yaml
```

3. Listenerリソースを参照して、Ingressリソース (`ingress.yaml`) を作成します。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  namespace: default
  annotations:
    ingress.citrix.com/listener: my-listener
spec:
  ingressClassName: citrix
  tls:
  - secretName: my-secret
    hosts:
    - example.com
  rules:
  - host: example.com
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: kuard
            port:
              number: 80
```

ここでは、Ingressリソース `my-ingress` が、フロントエンド構成のためにデフォルトの名前空間にあるListenerリソース `my-listener` を参照しています。

4. Ingressリソースを適用します。

```
kubectl apply -f ingress.yaml
```

証明書の管理

Ingressリソースの証明書を指定する方法は2つあります。Ingressリソースの一部として証明書を指定するか、Listenerリソースの一部として証明書を提供できます。

Ingressリソースによる証明書の管理

このアプローチでは、すべての証明書は通常のIngressリソースの一部として次のように指定されます。Listenerリソースは証明書を指定しません。このモードでは、Ingressリソースの一部として証明書を指定する必要があります。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: my-ingress
  namespace: default
  annotations:
    ingress.citrix.com/listener: my-listener
spec:
  ingressClassName: citrix
  tls:
  - secretName: my-secret
    hosts:
    - example.com
  rules:
  - host: example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: kuard
            port:
              number: 80
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Listenerリソースによる証明書の管理

このアプローチでは、証明書はListenerリソースの一部として提供されます。Ingressリソースの一部として証明書を指定する必要はありません。

次のListenerリソースの例は証明書を示しています。

```
apiVersion: citrix.com/v1

kind: Listener

metadata:

  name: my-listener

  namespace: default

spec:

  ingressclass: citrix

  certificates:

  - secret:

      name: my-secret

    # Secret named 'my-secret' in current namespace bound as default
certificate

    default: true

  - secret:

      # Secret 'other-secret' in demo namespace bound as SNI certificate

      name: other-secret

      namespace: demo

  vip: '192.168.0.1' # Virtual IP address to be used, not required when
CPX is used as ingress device

  port: 443
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  protocol: https

  redirectPort: 80
```

Ingressリソースでは、次の例に示すようにシークレットは指定されません。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: my-ingress

  namespace: default

  annotations:

    ingress.citrix.com/listener: my-listener

spec:

  ingressClassName: citrix

  tls:

  # TLS field is empty as the certs are specified in Listener

  rules:

  - host: example.com

    http:

      paths:

      - path: /

        pathType: Prefix

        backend:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        service:
          name: kuard
          port:
            number: 80
```

リスナーにおけるSSLパススルーのサポート

SSLパススルーは、着信SSL/TLSリクエストがロードバランサー（NetScaler）によって復号されないネットワーク構成です。代わりに、暗号化されたトラフィックは、復号のために指定されたバックエンドサーバー（アプリケーションポッド）に直接転送されます。

- NetScalerはトラフィックを復号しませんが、リスナーに証明書をバインドすることは依然として必須です。
- NetScalerは、リクエストを転送する前に、クライアントの最初のClient Helloメッセージで Server Name Indication (SNI) マッチングを実行します。これにより、ルーティングとセキュリティ検証の追加レイヤーが提供されます。
- SSLパススルーは、リスナー内でアノテーションとして参照される場合、任意のリスナーでサポートされるようになりました。
- このサポートには、マルチクラスターデプロイメントが含まれます。
    - あるクラスターのマルチクラスターリスナーでパススルー機能が有効になっている場合、マルチクラスターソリューションに参加している他のすべてのクラスターでも有効にする必要があります。
- パススルーはリスナーレベルで設定されます。リスナーを参照するすべてのIngressでパススルーが有効になります。

リスナーCRDは、NetScaler上のフロントエンド仮想IP (VIP) の動作を制御します。以下のパラメーターをリスナースペックに含める必要があります。

パラメーター	値	デフォルト	説明
パススルー	真	いいえ	このリスナーのVIPとポートに対してSSLパススルーを有効にします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

リスナーCRDの例

```
apiVersion: citrix.com/v1

kind: Listener

metadata:

  name: listener-https-passthrough

  namespace: default

spec:

  multicluster: true # Required for this feature to be active

  vip: '1.2.3.4'

  port: 443

  protocol: https

  passthrough: true # Enables SSL Passthrough

  certificates:

  - secret:

    name: app-secret # Mandatory certificate binding
```

イングレス YAML

SSLパススルーを有効にするためにIngress YAML定義に変更は必要ありません。パススルーの動作はリスナーCRDによって完全に制御されます。

以下は、設定されたリスナーを参照するIngress定義のサンプルです。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: https-ingress
  annotations:
    ingress.citrix.com/listener: listener-https-passthrough
spec:
  ingressClassName: "cic-vpx"
  rules:
  - host: httpsapp-netscaler.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: https-app
            port:
              number: 80
```

NetScaler Ingress Controller を使用した一貫性ハッシュアルゴリズムの構成

ロードバランシングアルゴリズムは、NetScalerアプライアンスが各クライアント要求をリダイレクトするサービスを選択するために使用する基準を定義します。異なるロードバランシングアルゴリズムは異なる基準を使用し、一貫性ハッシュはNetScalerがサポートするロードバランシングアルゴリズムの1つです。 一貫性ハッシュアルゴリズムは、バックエンドがキャッシュサーバーである場合に、ステートレスな永続性を実現するためにロードバランシングによく使用されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

一貫性ハッシュは、キャッシュサーバーが削除されたときに、その特定のサーバーにキャッシュされた要求のみが再ハッシュされ、残りの要求は影響を受けないことを保証できます。一貫性ハッシュアルゴリズムの詳細については、NetScalerドキュメントを参照してください。

NetScaler Ingress Controller を使用して、NetScaler で一貫性ハッシュアルゴリズムを構成できるようになりました。この構成は、ConfigMap を使用して NetScaler Ingress Controller 内で有効になります。

ハッシュアルゴリズムの構成

ハッシュアルゴリズムをサポートするために、NetScaler Ingress Controller の ConfigMap に新しいパラメータ `NS_LB_HASH_ALGO` が導入されました。 `NS_LB_HASH_ALGO` パラメータの下で ConfigMap を使用した一貫性ハッシュアルゴリズムのサポートされる環境変数:

- `hashFingers`: ハッシュアルゴリズムに使用するフィンガーの数を指定します。指定可能な値は1から1024です。フィンガーの数を増やすと、追加メモリを消費しますが、トラフィックの分散が向上します。
- `hashAlgorithm`: サポートされているアルゴリズムを指定します。サポートされているアルゴリズムは `default`、 `jarh`、 `prac` です。

次の例は、NetScaler Ingress Controller を使用して一貫性ハッシュアルゴリズムを構成するための ConfigMap のサンプルを示しています。この例では、ハッシュアルゴリズムとして Prime Re-Shuffled Assisted CARP (PRAC) が使用され、PRAC で使用されるフィンガーの数は50に設定されています。

```
apiVersion: v1

kind: ConfigMap

metadata:

name: cic-configmap

labels:

    app: citrix-ingress-controller

data:

NS_LB_HASH_ALGO: |
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    hashFingers: 50

    hashAlgorithm: 'prac'
```

NetScaler Ingress Controller を使用して DNS レコードを追加する

DNSアドレスレコードは、ドメイン名をIPアドレスにマッピングしたものです。 NetScalerをDNSリゾルバーとして使用したい場合、NetScaler Ingress Controllerを使用してNetScalerにDNSレコードを追加できます。

NetScalerでのDNSレコードの作成に関する詳細については、NetScalerドキュメントを参照してください。

IngressリソースのDNSレコードを追加する

IngressリソースのDNSレコードを追加するには、NetScaler Ingress Controllerのデプロイ中に以下の環境変数を有効にする必要があります。

`NS_CONFIG_DNS_REC`: この変数は起動時に設定され、実行時に変更することはできません。可能な値は`true`または`false`です。デフォルト値はfalseであり、DNSサーバー設定を有効にするにはtrueに設定する必要があります。値を`true`に設定すると、NetScaler上にアドレスレコードが作成されます。

LoadBalancerタイプのサービスのDNSレコードを追加する

LoadBalancerタイプのサービスのDNSレコードを追加するには、以下のタスクを実行する必要があります。

- LoadBalancerタイプのサービスのDNSレコードを追加するには、`NS_SVC_LB_DNS_REC`環境変数の値を`True`に設定して有効にします。
- `service.citrix.com/dns-hostname`アノテーションを使用してDNSホスト名を指定します。

`service.citrix.com/dns-hostname`アノテーションを持つLoadBalancerタイプのサービスを作成すると、NetScaler Ingress ControllerはNetScalerにDNSレコードを追加します。DNSレコードは、アノテーションで指定されたドメイン名と、サービスに割り当てられた外部IPアドレスを使用して構成されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

`service.citrix.com/dns-hostname` アノテーションを持つLoadBalancerタイプのサービスを削除すると、NetScaler Ingress ControllerはNetScalerからDNSレコードを削除します。 NetScaler Ingress Controllerは、サービスが利用できない場合、起動時にDNSレコードの古いエントリも削除します。

以下の例は、NetScalerにDNSレコードを追加するためのアノテーション設定を持つLoadBalancerタイプのサービスサンプルを示しています。

```
apiVersion: v1

kind: Service

metadata:

  name: guestbook

  annotations:

      service.citrix.com/dns-hostname: "guestbook.com"

spec:

  loadBalancerIP: "192.2.212.16"

  type: LoadBalancer

  ports:

  - port: 9006

    targetPort: 80

    protocol: TCP

  selector:

    app: guestbook
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler を使用した Kubernetes の Open Policy Agent サポート

Open Policy Agent (OPA) は、さまざまなテクノロジーやシステムにわたるポリシー適用を統合する、オープンソースの汎用ポリシーエンジンです。OPA は、ポリシーをコードとして指定できる高レベルの宣言型言語と、ソフトウェアからポリシー決定をオフロードするためのシンプルな API を提供します。OPA を使用すると、ポリシー決定とポリシー適用を分離できます。Kubernetes 環境で NetScaler を介してポリシーを適用するために OPA を使用できます。

OPA を使用すると、複数の NetScaler または分散された複数のデバイスを含む環境向けに、一元化されたポリシー決定システムを作成できます。このアプローチの利点は、複数のデバイスに適用される決定固有の変更については、OPA サーバー上でのみ変更を行えばよいことです。

OPA の詳細については、OPA ドキュメント を参照してください。

NetScaler での OPA 統合は HTTP コールアウトを介してサポートでき、OPA は認証の有無にかかわらず使用できます。HTTP コールアウトは、ポリシー評価の一部として NetScaler アプライアンスが生成し、外部アプリケーションに送信する HTTP または HTTPS リクエストです。

HTTP コールアウトのサポートの詳細については、HTTP コールアウトのドキュメント を参照してください。

認証サポートの詳細については、NetScaler を使用した Kubernetes の認証および承認ポリシー を参照してください。

次の図は、OPA を NetScaler クラウドネイティブソリューションと統合する方法の概要を示しています。

OPA 統合図では、各番号は次のリストの対応するタスクを表しています。

1. Kubernetes コマンドを使用して必要な Kubernetes オブジェクトを作成します。この手順には、HTTP コールアウトを OPA サーバーに送信するための CRD の作成が含まれる必要があります。
2. NetScaler の構成。NetScaler は、作成された Kubernetes オブジェクトに基づいて NetScaler Ingress Controller によって自動的に構成されます。
3. クライアントからリソースのユーザーリクエストを送信します。認証 CRD が作成されている場合、ユーザーは認証される可能性があります。
4. 認証パラメーターを保持する JSON 形式の HTTP コールアウトを NetScaler から OPA サーバーに送信します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

5. OPA のポリシー言語である REGO で定義されたルールに基づいて、OPA サーバーから承認決定を送信します。
6. 承認決定に基づいてクライアントに応答を送信する。

使用例

例1：クライアントの送信元IPアドレスに基づいてリソースへのアクセスを許可または拒否する

以下は、クライアントの送信元IPアドレスと対応するOPAルールに基づいてリソースへのアクセスを許可または拒否するために、リライトポリシーCRDを使用してOPAサーバーにHTTPコールアウトポリシーを適用する例です。

この例では、クライアントの送信元IPアドレスが192.2.162.0/24の場合、OPAサーバーは`"result": true`で応答し、それ以外の場合は`"result":false`で応答します。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: calloutexample

spec:

  responder-policies:

    - servicenames:

        - frontend

      responder-policy:

        respondwith:

          http-payload-string: '"HTTP/1.1 401 Access denied\r\n\r\n"'
#Access is denied if the respose from OPA server contains false.

        respond-criteria:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
'sys.http_callout("callout_name").CONTAINS("false")'

        comment: 'Invalid access'

  httpcallout_policy:

    - name: callout_name

      server_ip: "192.2.156.160" #OPA Server IP

      server_port: 8181 #OPA Server Port

      http_method: 'POST'

      host_expr: "\"192.2.156.160\""

      url_stem_expr: "\"/v1/data/example/allow\"" #URL stem expression to
be used

      body_expr: '"{\"input\": {\"clientinfo\": [{\"id\": \"ci\", \"ip\":
[\""+ CLIENT.IP.SRC +"\"]}]}}"' #JSON to OPA server carrying client IP

      headers:

      - name: Content-Type

        expr: '"application/json"'

      return_type: TEXT

      result_expr: "HTTP.RES.BODY(100)"
```

以下は、この例のHTTPコールアウトポリシーのためにOPAサーバー上でRegoポリシー言語を通じて定義されたルールです。

```
    package example



    default allow = false                                # unless otherwise
defined, allow is false
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    allow = true {                                          # allow is true
if...

        count(violation) != 0                               # the ip matches
regex.

    }



    violation[client.id] {                                  # a client is in
the violation set if...

        client := input.clientinfo[_]

        regex.match("192.2.162.", client.ip[_])             # the client is
not part of 192.2.162.0/24 network.

    }
```

例2：認証後のユーザーグループに基づいてアクセスを許可または拒否する

以下は、認証後のユーザーグループと対応するOPAルールに基づいてリソースへのアクセスを許可または拒否するために、リライトポリシーCRDを使用してOPAサーバーにHTTPコールアウトポリシーを適用する例です。

この例では、ユーザーが`beverages`グループの一部である場合、OPAサーバーは`"result":true`で応答し、それ以外の場合は`"result":false`で応答します。

以下は、リライトポリシーCRDを介したOPAサーバーへのHTTPコールアウトポリシーです。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: calloutexample

spec:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  responder-policies:

    - servicenames:

        - frontend

      responder-policy:

        respondwith:

          http-payload-string: '"HTTP/1.1 401 Access denied\r\n\r\n"'
#Access is denied if the respose from OPA server contains false.

        respond-criteria:
'sys.http_callout("callout_name").CONTAINS("false")'

        comment: 'Invalid access'



  httpcallout_policy:

    - name: callout_name

      server_ip: "192.2.156.160" #OPA Server IP

      server_port: 8181 #OPA Server Port

      http_method: 'POST'

      host_expr: "\"192.2.156.160\""

      url_stem_expr: "\"/v1/data/example/allow\"" #URL stem expression to
be used

      body_expr: '"{\"input\": {\"users\": [{\"name\": \""+ AAA.USER.NAME
+"\", \"group\": [\""+ AAA.USER.GROUPS +"\"]}]}}"' #JSON to OPA server
carrying username and group information

      headers:

      - name: Content-Type
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        expr: '"application/json"'

      return_type: TEXT

      result_expr: "HTTP.RES.BODY(100)"
```

以下は、この例のためにOPAサーバー上でRego言語を通じて定義されたルールです。

```
    package example



    default allow = false                                  # unless otherwise
defined, allow is false



    allow = true {                                         # allow is true
if...

        count(isbeveragesuser) != 0                        # the user is part
of beverages group.

    }



    isbeveragesuser[user.name] {                           # a user is
beverages user...

        user := input.users[_]

        user.group[_] == "beverages"                       # if it is part of
beverages group.

    }
```

認証は、リクエストヘッダー（401ベース）またはフォームベースを使用して実行できます。

以下は、リクエストヘッダーベースの認証を使用する認証ポリシーの例です。このポリシーでは、ローカル認証が使用されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: citrix.com/v1beta1

kind: authpolicy

metadata:

  name: localauth

spec:

    servicenames:

    - frontend



    authentication_mechanism:

      using_request_header: 'ON'



    authentication_providers:



        - name: "local-auth-provider"

          basic_local_db:

              use_local_auth: 'YES'



    authentication_policies:



        - resource:

            path: []

            method: []
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          provider: ["local-auth-provider"]


    authorization_policies:


        - resource:
            path: []
            method: []
            claims: []
```

以下は、フォームベースの認証を使用する認証ポリシーの例です。このポリシーでは、ローカルベースの認証が使用されます。

```
apiVersion: citrix.com/v1beta1
kind: authpolicy
metadata:
  name: localauth
spec:
    servicenames:
    - frontend


    authentication_mechanism:
      using_forms:
        authentication_host: "fqdn_authenticaton_host"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        authentication_host_cert:
          tls_secret: authhost-tls-cert-secret
        vip: "192.2.156.156"


    authentication_providers:


        - name: "local-auth-provider"
          basic_local_db:
              use_local_auth: 'YES'


    authentication_policies:


        - resource:
            path: []
            method: []
          provider: ["local-auth-provider"]



    authorization_policies:


        - resource:
            path: []
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            method: []

            claims: []
```

例3：認証中に取得された認証属性に基づいてアクセスを許可または拒否する

以下は、認証中に取得された認証属性と対応するOPAルールに基づいてアクセスを許可または拒否するための、リライトポリシーCRDを使用したOPAサーバーへのHTTPコールアウトポリシーの例です。

この例では、ユーザーの`memberof`属性に`grp1`が含まれている場合、OPAサーバーは`"result":true'`で応答し、そうでない場合は`"result":false`で応答します。

以下は、リライトポリシーCRDを介したOPAサーバーへのサンプルHTTPコールアウトポリシーです。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: calloutexample

spec:

  responder-policies:

    - servicenames:

        - frontend

      responder-policy:

        respondwith:

          http-payload-string: '"HTTP/1.1 401 Access denied\r\n\r\n"'
#Access is denied if the respose from OPA server contains false.

        respond-criteria:
'sys.http_callout("callout_name").CONTAINS("false")'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        comment: 'Invalid access'



  httpcallout_policy:

    - name: callout_name

      server_ip: "192.2.156.160" #OPA Server IP

      server_port: 8181 #OPA Server Port

      http_method: 'POST'

      host_expr: "\"192.2.156.160\""

      url_stem_expr: "\"/v1/data/example/allow\"" #URL stem expression to
be used

      body_expr: '"{\"input\": {\"users\": [{\"name\": \""+ AAA.USER.NAME
+"\", \"attr\": [\""+ aaa.user.attribute("memberof") +"\"]}]}}"' #JSON to
OPA server carrying username and "memberof" attribute information

      headers:

      - name: Content-Type

        expr: '"application/json"'

      return_type: TEXT

      result_expr: "HTTP.RES.BODY(100)"
```

以下は、この例のためにOPAサーバー上でRego言語を介して定義されたルールです。

```
    package example



    default allow = false                                # unless otherwise
defined, allow is false
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    allow = true {                                          # allow is true
if...

        count(isbeveragesuser) != 0                         # the user is part
of grp1.

    }



    isbeveragesuser[user.name] {                            # a user is part
of allow group...

        user := input.users[_]

        regex.match("CN=grp1", user.attr[_])                # if it is part of
grp1 group.
```

}

リクエストヘッダー（401ベース）またはフォームベースを使用して認証を実行できます。この例ではLDAP認証が使用され、ユーザーの`memberof`属性は認証中にLDAPサーバーから取得されます。

以下は、リクエストヘッダーベースの認証を使用した認証ポリシーの例です。

```
apiVersion: citrix.com/v1beta1

kind: authpolicy

metadata:

  name: ldapauth

spec:

    servicenames:

    - frontend
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    authentication_mechanism:

      using_request_header: 'ON'



    authentication_providers:

        - name: "ldap-auth-provider"

          ldap:

              server_ip: "192.2.156.160"

              base: 'dc=aaa,dc=local'

              login_name: accountname

              sub_attribute_name: CN

              server_login_credentials: ldapcredential

              attributes_to_save: memberof #memberof attribute to be
obtained from LDAP server for user



    authentication_policies:

        - resource:

            path: []

            method: []

          provider: ["ldap-auth-provider"]



    authorization_policies:

        - resource:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            path: []

            method: []

            claims: []
```

以下は、フォームベースの認証を使用した認証ポリシーの例です。

```


apiVersion: citrix.com/v1beta1

kind: authpolicy

metadata:

  name: authhotdrinks

spec:

    servicenames:

    - frontend



    authentication_mechanism:

      using_forms:

        authentication_host: "fqdn_authenticaton_host"

        authentication_host_cert:

          tls_secret: authhost-tls-cert-secret

        vip: "192.2.156.156"



    authentication_providers:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        - name: "ldap-auth-provider"

          ldap:

              server_ip: "192.2.156.160"

              base: 'dc=aaa,dc=local'

              login_name: accountname

              sub_attribute_name: CN

              server_login_credentials: ldapcredential

              attributes_to_save: memberof #memberof attribute to be
obtained from LDAP server for user



    authentication_policies:



        - resource:

            path: []

            method: []

          provider: ["ldap-auth-provider"]
```

メトリクスをPrometheusに直接エクスポートする

NetScaler Ingress Controllerは、NetScalerからPrometheusへメトリクスを直接エクスポートすることをサポートするようになりました。 NetScaler Ingress Controllerを使用すると、メトリクスを直接エクスポートするためにNetScalerで必要となる構成を自動化できます。

メトリクスをエクスポートすると、Grafanaなどのツールを使用して、エクスポートされたNetScalerメトリクスを視覚化し、より簡単に解釈および理解できるようになります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler CPXからPrometheusへのメトリクスの直接エクスポートを構成する

NetScaler Ingress ControllerがNetScaler CPXを構成して、Prometheusへのメトリクスの直接エクスポートをサポートできるようにするには、次の手順を実行する必要があります。

1. ユーザーの読み取り専用アクセスを有効にするKubernetesシークレットを作成します。この手順は、NetScaler CPXがメトリクスをPrometheusにエクスポートするために必要です。

```
kubectl create secret generic prom-user --from-
literal=username=<prometheus-username> --from-
literal=password=<prometheus-password>
```

2. 次のHelmコマンドを使用して、NetScaler Ingress ControllerでNetScaler CPXを展開します。

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-
charts/



helm install my-release netscaler/netscaler-cpx-with-ingress-controller
--set license.accept=yes,nsic.prometheusCredentialSecret=<Secret-for-
read-only-user-
creation>,analyticsConfig.required=true,analyticsConfig.timeseries.metr
ics.enable=true,analyticsConfig.timeseries.port=5563,analyticsConfig.ti
meseries.metrics.mode=prometheus,analyticsConfig.timeseries.metrics.ena
bleNativeScrape=true
```

   コマンドで指定された新しいパラメーターは、次のとおりです。

   - `nsic.prometheusCredentialSecret`: ネイティブPrometheusサポート用の読み取り専用ユーザーを作成するためのKubernetesシークレット名を指定します。
   - `analyticsConfig.timeseries.metrics.enableNativeScrape`: メトリクスをPrometheusに直接エクスポートするには、この値を`true`に設定します。

3. Prometheusの展開に応じて、Prometheus構成の`scrape_configs`の下に適切なPrometheusスクレイプジョブを追加します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- PrometheusサーバーがKubernetesクラスターの外部にある場合は、Prometheus構成の `scrape_configs` の下にスクレイプジョブを追加します。Prometheusスクレイプジョブの例については、Prometheus統合ドキュメントを参照してください。
- Prometheusサーバーが同じKubernetesクラスター内にある場合は、NetScaler CPXポッドから直接エクスポートするようにPrometheusを構成するための新しいPrometheusジョブを追加します。詳細については、kubernetes_sd_configを参照してください。Prometheusジョブの例を次に示します。

```
- job_name: 'kubernetes-cpx'

  scheme: http

  metrics_path: /nitro/v1/config/systemfile

  params:

    args:
['filename:metrics_prom_ns_analytics_time_series_profile.log,filelocati
on:/var/nslog']

    format: ['prometheus']

  basic_auth:

    username:  # Prometheus username set in
nsic.prometheusCredentialSecret

    password:  # Prometheus password set in
nsic.prometheusCredentialSecret

  scrape_interval: 30s

  kubernetes_sd_configs:

  - role: pod

  relabel_configs:

  - source_labels:
[__meta_kubernetes_pod_annotation_netscaler_prometheus_scrape]
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    action: keep

    regex: true

  - source_labels: [__address__,
__meta_kubernetes_pod_annotation_netscaler_prometheus_port]

    action: replace

    regex: ([^:]+)(?::\d+)?;(\d+)

    replacement: $1:$2

    target_label: __address__

  - source_labels: [__meta_kubernetes_namespace]

    action: replace

    target_label: kubernetes_namespace

  - source_labels: [__meta_kubernetes_pod_name]

    action: replace

    target_label: kubernetes_pod_name
```

注：

Prometheus統合の詳細については、NetScaler Prometheus統合ドキュメントを参照してください。

NetScaler VPX または NetScaler MPX から Prometheus へのメトリクスの直接エクスポートの構成

NetScaler Ingress Controller が NetScaler VPX または NetScaler MPX を構成して、Prometheus へのメトリクスの直接エクスポートをサポートできるようにするには、次の手順を実行する必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

1. Helm コマンドを使用して、NetScaler Ingress Controller をスタンドアロンポッドとして展開します。

```
    helm repo add netscaler https://netscaler.github.io/netscaler-helm-
charts/



   helm install my-release netscaler/netscaler-ingress-controller --set
nsIP=<NSIP>,license.accept=yes,adcCredentialSecret=<Secret-for-
NetScaler-
credentials>,analyticsConfig.required=true,analyticsConfig.timeseries.m
etrics.enable=true,analyticsConfig.timeseries.port=5563,analyticsConfig
.timeseries.metrics.mode=prometheus,analyticsConfig.timeseries.metrics.
enableNativeScrape=true
```

2. NetScaler VPX 用の読み取り専用アクセスを持つシステムユーザーを作成します。ユーザー作成の詳細については、NetScaler Prometheus 統合ドキュメント を参照してください。
3. Prometheus が NetScaler VPX からスクレイピングできるように、Prometheus 構成 の `scrape_configs` にスクレイプジョブを追加します。Prometheus スクレイプジョブの例については、Prometheus 構成 を参照してください。

注:

スクレイプ構成セクションでは、ターゲットのセットと、それらをスクレイプする方法を記述する構成パラメーターを指定します。構成で使用される NetScaler 固有のパラメーターの詳細については、NetScaler ドキュメント を参照してください。

管理パーティションを持つNetScaler向けNetScaler Ingress Controllerを展開する

NetScaler Ingress Controllerは、Ingressリソース構成に基づいて1つ以上のNetScalerを自動的に構成するために使用されます。Ingress NetScalerアプライアンス（MPXまたはVPX）は、管理パーティションと呼ばれる論理エンティティに分割でき、各パーティションは個別のNetScalerアプライアンスとして構成および使用できます。詳細については、管理パーティションを参照してください。NetScaler Ingress Controllerは、管理パーティションを持つNetScalerを構成するためにも展開できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

管理パーティションを持つNetScalerの場合、各パーティションに対してNetScaler Ingress Controllerの単一インスタンスを展開する必要があります。また、そのパーティションはNetScaler Ingress Controllerインスタンスに固有のパーティションユーザーに関連付けられている必要があります。

注：

NetScaler Metrics Exporterは、NetScalerの管理パーティションからメトリクスをエクスポートすることをサポートしています。

前提条件

以下を確認してください。

- NetScalerアプライアンスで管理パーティションが構成されていること。手順については、管理パーティションの構成を参照してください。
- NetScaler Ingress Controller専用のパーティションユーザーを作成します。NetScaler Ingress Controllerは、このパーティションユーザーアカウントを使用してNetScalerを構成します。このパーティションユーザーをNetScalerアプライアンス内の他のパーティションに関連付けないようにしてください。

  注：

  管理パーティションでのSSL関連のユースケースについては、NetScalerバージョン12.0–56.8以降を使用していることを確認してください。

管理パーティションを持つNetScaler向けNetScaler Ingress Controllerを展開するには

1. 次のコマンドを使用してcitrix-k8s-ingress-controller.yamlをダウンロードします。

   ```
   wget  https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/baremetal/citrix-k8s-ingress-controller.yaml
   ```

2. citrix-k8s-ingress-controller.yamlファイルを編集し、以下の環境変数の値を入力します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

環境変数	必須またはオプション	説明
NS_IP	必須	NetScalerアプライアンスのIPアドレスです。詳細については、前提条件を参照してください。
NS_USER および NS_PASSWORD	必須	NetScaler Ingress Controller用に作成したパーティションユーザーのユーザー名とパスワードです。詳細については、前提条件を参照してください。
NS_VIP	必須	NetScaler Ingress Controllerは、この環境変数で提供されるIPアドレスを使用して、Ingressトラフィックを受信するNetScalerに仮想IPアドレスを構成します。**注:** NS_VIPは、Ingress YAMLでfrontend-ipアノテーションが指定されていない場合のフォールバックとして機能します。Ingressのみがサポートされます。
NS_SNIPS	オプション	NetScalerアプライアンス上のSNIPアドレス、またはNetScalerアプライアンス上の特定の管理者パーティション上のSNIPアドレスを指定します。
NS_ENABLE_MONITORING	必須	NetScalerを監視するには、`Yes`の値を設定します。**注:** 管理パーティションを持つNetScalerの場合、NetScaler監視を無効にしてください。値を`No`に設定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

環境変数	必須またはオプション	説明
EULA	必須	エンドユーザーライセンス契約。値を`Yes`として指定します。
Kubernetes_url	オプション	NetScaler Ingress Controllerがイベントを登録するために使用するkube-apiserverのURL。値が指定されていない場合、NetScaler Ingress Controllerは内部kube-apiserver IPアドレスを使用します。
LOGLEVEL	オプション	NetScaler Ingress Controllerによって生成されるログを制御するためのログレベル。デフォルトでは、値はDEBUGに設定されています。サポートされている値は、CRITICAL、ERROR、WARNING、INFO、DEBUGです。詳細については、ログレベルを参照してください。
NS_PROTOCOL と NS_PORT	オプション	NetScaler Ingress ControllerがNetScalerと通信するために使用する必要があるプロトコルとポートを定義します。デフォルトでは、NetScaler Ingress Controllerはポート443でHTTPSを使用します。ポート80でHTTPを使用することもできます。
イングレスクラス	オプション	複数のイングレスロードバランサーが異なるイングレスリソースの負荷分散に使用される場合。この環境変数を使用して、特定のイン

© 1997–2026 Citrix Systems, Inc. All rights reserved.

環境変数	必須またはオプション	説明
		グレスクラスに関連付けられたNetScalerを構成するようにNetScaler Ingress Controllerを指定できます。イングレスクラスの詳細については、「イングレスクラスのサポート」を参照してください。

3. 環境変数を更新したら、YAMLファイルを保存し、次のコマンドを使用してデプロイします。

```
kubectl create -f citrix-k8s-ingress-controller.yaml
```

4. 次のコマンドを使用して、NetScaler Ingress Controllerが正常にデプロイされたことを確認します。

```
kubectl get pods --all-namespaces
```

ユースケース: NetScaler Consoleのパーティションを使用してマルチテナントのマイクロサービスベースアプリケーションを安全に提供する方法

NetScaler Ingress Controllerを使用すると、NetScaler Consoleのパーティションで、異なるマイクロサービスベースアプリケーション間のイングレストラフィックを分離できます。NetScaler Consoleのパーティションは、単一のNetScalerインスタンスでソフトウェアレベルでのマルチテナンシーを可能にします。各パーティションには、独自のコントロールプレーンとネットワークプレーンがあります。

クラスター内の各名前空間にNetScaler Ingress Controllerのインスタンスを1つデプロイできます。

たとえば、Kubernetesクラスターに2つの名前空間があり、これらを2つの異なる管理者で互いに分離したいとします。管理者パーティション機能を使用して、これら2つの名前空間を分離できます。名前空間1と名前空間2を作成し、これら両方の名前空間にNetScaler Ingress Controllerを個別にデプロイします。

NetScaler Ingress Controllerインスタンスは、YAMLマニフェストで指定されたシステムユーザーアカウントを使用して、それぞれのNetScalerパーティションに構成指示を提供します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

この例では、apacheとguestbookのサンプルアプリケーションが、Kubernetesクラスター内の2つの異なる名前空間（それぞれ名前空間1と名前空間2）にデプロイされています。apacheとguestbookの両方のアプリケーションチームは、ワークロードを独立して管理し、リソースを共有したくないと考えています。NetScaler Consoleのパーティションはマルチテナンシーの実現に役立ち、この例では、両方のアプリケーションワークロードを個別に管理するために2つのパーティション（default、partition1）が使用されます。

以下の前提条件が適用されます。

- NetScalerアプライアンスで管理者パーティションを構成していることを確認してください。手順については、「管理者パーティションの構成」を参照してください。
- NetScaler Ingress Controller専用のパーティションユーザーアカウントを作成していることを確認してください。NetScaler Ingress Controllerは、このパーティションユーザーアカウントを使用してNetScalerを構成します。このパーティションユーザーをNetScalerアプライアンスの他のパーティションに関連付けないでください。

例

Kubernetesクラスター内の異なる名前空間に異なるアプリケーションをデプロイする方法と、管理パーティションを使用してADCからリクエストを分離する方法を、以下のシナリオ例で示します。

この例では、2つのサンプルアプリケーションがKubernetesクラスター内の2つの異なる名前空間にデプロイされています。この例では、`apache`アプリケーションにはNetScalerのデフォルトパーティションが使用され、`guestbook`アプリケーションには管理パーティション`p1`が使用されます。

名前空間を作成する

次のコマンドを使用して、2つの名前空間`ns1`と`ns2`を作成します。

```
kubectl create namespace ns1

kubectl create namespace ns2
```

名前空間`ns1`での構成

1. `ns1`に`apache`アプリケーションをデプロイします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: v1

kind: Namespace

metadata:

  name: ns1



---

apiVersion: apps/v1

kind: Deployment

metadata:

  labels:

    app: apache-ns1

  name: apache-ns1

  namespace: ns1

spec:

  replicas: 2

  selector:

    matchLabels:

      app: apache-ns1

  template:

    metadata:

      labels:

        app: apache-ns1
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    spec:
      containers:
      - image: httpd
        name: httpd
---


apiVersion: v1
kind: Service
metadata:
  creationTimestamp: null
  labels:
    app: apache-ns1
  name: apache-ns1
  namespace: ns1
spec:
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: apache-ns1
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

2. `ns1`にNetScaler Ingress Controllerをデプロイします。

   NetScaler Ingress Controllerをデプロイするには、YAMLファイルを使用するか、Helmチャートを使用できます。

   デフォルトパーティションにバインドされているユーザー資格情報を使用していることを確認してください。

```
helm install cic-def-part-ns1 citrix/citrix-ingress-controller --set
nsIP=&lt;nsIP of
ADC>,license.accept=yes,adcCredentialSecret=nslogin,ingressClass[0]=cit
rix-def-part-ns1 --namespace ns1
```

3. Ingressリソースをデプロイします。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: ingress-apache-ns1

  namespace: ns1

  annotations:

    ingress.citrix.com/frontend-ip: "< ADC VIP IP >"

spec:

  ingressClassName: "citrix-def-part-ns1"

  rules:

  - host: apache-ns1.com

    http:

      paths:

      - backend:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          service:
            name: apache-ns1
            port:
              number: 80
        pathType: Prefix
        path: /index.html
```

4. `ns1`のNetScaler Ingress Controllerは、デフォルトパーティション内のADCエンティティを構成します。

名前空間`ns2`での構成

1. `ns2`に`guestbook`アプリケーションをデプロイします。

```
apiVersion: v1
kind: Namespace
metadata:
  name: ns2
---
apiVersion: v1
kind: Service
metadata:
  name: redis-master
  namespace: ns2
  labels:
    app: redis
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    tier: backend

    role: master

spec:

  ports:

  - port: 6379

    targetPort: 6379

  selector:

    app: redis

    tier: backend

    role: master

---

apiVersion: apps/v1 #  for k8s versions before 1.9.0 use apps/v1beta2
 and before 1.8.0 use extensions/v1beta1

kind: Deployment

metadata:

  name: redis-master

  namespace: ns2

spec:

  selector:

    matchLabels:

      app: redis

      role: master
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      tier: backend
  replicas: 1
  template:
    metadata:
      labels:
        app: redis
        role: master
        tier: backend
    spec:
      containers:
      - name: master
        image: k8s.gcr.io/redis:e2e  # or just image: redis
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        ports:
        - containerPort: 6379
---
apiVersion: v1
kind: Service
metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: redis-slave

  namespace: ns2

  labels:

    app: redis

    tier: backend

    role: slave

spec:

  ports:

  - port: 6379

  selector:

    app: redis

    tier: backend

    role: slave

---

apiVersion: apps/v1 #  for k8s versions before 1.9.0 use apps/v1beta2
 and before 1.8.0 use extensions/v1beta1

kind: Deployment

metadata:

  name: redis-slave

  namespace: ns2

spec:

  selector:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    matchLabels:
      app: redis
      role: slave
      tier: backend
  replicas: 2
  template:
    metadata:
      labels:
        app: redis
        role: slave
        tier: backend
    spec:
      containers:
      - name: slave
        image: gcr.io/google_samples/gb-redisslave:v1
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          # If your cluster config does not include a dns service, then
to

          # instead access an environment variable to find the master

          # service's host, comment out the 'value: dns' line above,
and

          # uncomment the line below:

          # value: env

        ports:

        - containerPort: 6379

---

apiVersion: v1

kind: Service

metadata:

  name: frontend

  namespace: ns2

  labels:

    app: guestbook

    tier: frontend

spec:

  # if your cluster supports it, uncomment the following to
automatically create

  # an external load-balanced IP for the frontend service.

  # type: LoadBalancer
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  ports:

  - port: 80

  selector:

    app: guestbook

    tier: frontend

---

apiVersion: apps/v1 #  for k8s versions before 1.9.0 use apps/v1beta2
 and before 1.8.0 use extensions/v1beta1

kind: Deployment

metadata:

  name: frontend

  namespace: ns2

spec:

  selector:

    matchLabels:

      app: guestbook

      tier: frontend

  replicas: 3

  template:

    metadata:

      labels:

        app: guestbook
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        tier: frontend
    spec:
      containers:
      - name: php-redis
        image: gcr.io/google-samples/gb-frontend:v4
        resources:
          requests:
            cpu: 100m
            memory: 100Mi
        env:
        - name: GET_HOSTS_FROM
          value: dns
          # If your cluster config does not include a dns service, then
to
          # instead access environment variables to find service host
          # info, comment out the 'value: dns' line above, and
uncomment the
          # line below:
          # value: env
        ports:
        - containerPort: 80
```

2. 名前空間`ns2`にNetScaler Ingress Controllerをデプロイします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パーティション `p1` にバインドされているユーザー資格情報を使用していることを確認してください。

```
helm install cic-adm-part-p1 citrix/citrix-ingress-controller --set
nsIP=&lt;nsIP of ADC>,nsSNIPS='[&lt;SNIPs in partition
p1>]',license.accept=yes,adcCredentialSecret=admin-part-user-
p1,ingressClass[0]=citrix-adm-part-ns2 --namespace ns2
```

3. `guestbook` アプリケーション向けにIngressを展開します。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  annotations:

  ingress.citrix.com/frontend-ip: "&lt;VIP in partition 1>"

  name: guestbook-ingress

  namespace: ns2

spec:

  ingressClassName: citrix-adm-part-ns2

  rules:

  - host: www.guestbook.com

    http:

      paths:

      - backend:

          service:

            name: frontend
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            port:
              number: 80
        path: /
        pathType: Prefix
```

4. `ns2`のNetScaler Ingress Controllerは、`p1`パーティション内のADCエンティティを設定します。

ネットワーク

ネットワークには以下のトピックがあります。

- 静的ルーティング
- ノードコントローラーを使用してK8sノードとIngress NetScaler間のネットワークを確立する
- NetScalerノードコントローラーを使用したOpenShiftのOVN-CNIサポート
- NodePortタイプのサービスを公開する
- Calicoを使用したPod間通信
- LoadBalancerタイプのサービスの機能強化

NetScaler VPX または MPX で静的ルートを構成する

Kubernetes クラスターでは、Pod は Flannel、Calico、Weave などのオーバーレイネットワーク上で実行されます。クラスター内の Pod には、ホストネットワークとは異なるオーバーレイネットワークから IP アドレスが割り当てられます。

Kubernetes クラスター外の NetScaler MPX または NetScaler VPX は、Kubernetes クラスターにデプロイされたマイクロサービスへのすべてのイングレストラフィックを受信します。イングレストラフィックがマイクロサービスに到達するためには、NetScaler インスタンスと Pod の間にネットワーク接続を確立する必要があります。

Pod と Kubernetes クラスター外の NetScaler MPX または NetScaler VPX インスタンス間のネットワーク接続を実現する方法の 1 つは、NetScaler インスタンスにオーバーレイネットワークへのルートを構成することです。これは手動で行うことも、NetScaler Ingress Controller がネットワークを自動的に構成するオプションを提供することもできます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注

NetScaler MPX または NetScaler VPX に、ホストネットワーク上で SNIP が構成されていることを確認してください。ホストネットワークとは、Kubernetes ノードが相互に通信するネットワークです。

NetScaler でルートを手動で構成する

1. Kubernetes クラスターのマスターノードで、次のコマンドを使用して podCIDR を取得します。

   ```
   kubectl get nodes -o jsonpath="{range .items[*]}{'podNetwork: '}
   {.spec.podCIDR}{'\t'}{'gateway: '}{.status.addresses[0].address}{'\n'}{end}"
   ```

   ```
   podNetwork: 10.244.0.0/24    gateway: 10.106.162.108

   podNetwork: 10.244.2.0/24    gateway: 10.106.162.109

   podNetwork: 10.244.1.0/24    gateway: 10.106.162.106
   ```

2. NetScaler インスタンスにログオンします。
3. podCIDR 情報を使用して NetScaler にルートを追加します。次のコマンドを使用します。

   ```
   add route <pod_network> <podCIDR_netmask> <gateway>
   ```

   例:

   ```
   add route 192.244.0.0 255.255.255.0 192.106.162.108
   ```

   ```
   add route 192.244.2.0 255.255.255.0 192.106.162.109
   ```

   ```
   add route 192.244.1.0 255.255.255.0 192.106.162.106
   ```

NetScalerでルートを自動的に構成する

NSICデプロイメントでは、関連するNetScalerインスタンスでルートを自動的に構成するために、パラメーター `nodeWatch` を使用できます。

自動ルート構成を有効にするには、`nodeWatch` の値を `true` として指定します。例:

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
helm install my-release netscaler/netscaler-ingress-controller --set
nsIP=<NSIP>,license.accept=yes,adcCredentialSecret=<Secret-for-NetScaler-
credentials>,nodeWatch=true
```

注:

デフォルトでは、`nodeWatch` 引数は `false` に設定されています。自動ルート構成を有効にするには、引数を `true` に設定します。

ノードコントローラーを使用してKubernetesノードとIngress NetScaler間のネットワークを確立する

Kubernetes環境では、Ingressデバイスを介して外部アクセス用にサービスを公開する場合、KubernetesノードとIngressデバイス間のネットワークを適切に構成する必要があります。

ポッドはCNIフレームワークに基づいてプライベートIPアドレスを使用するため、ネットワークの構成は困難です。適切なネットワーク構成がないと、Ingressデバイスはこれらのプライベート IPアドレスにアクセスできません。また、このような到達可能性を確保するためにネットワークを手動で構成することは、Kubernetes環境では面倒です。

また、KubernetesクラスターとIngress NetScaler®が異なるサブネットにある場合、静的ルーティングを使用してそれらの間にルートを確立することはできません。このシナリオでは、KubernetesクラスターとIngress NetScaler間のルートを確立するためにオーバーレイメカニズムが必要です。

NetScalerは、KubernetesノードとIngress NetScaler間に仮想拡張LAN (VXLAN) ベースのオーバーレイネットワークを作成するために使用できるノードコントローラーを提供します。これは次の図に示されています。

注:

NetScaler Node Controllerは、NetScalerクラスターがIngressデバイスとして構成されているセットアップでは機能しません。ノードコントローラーは、ルートを構成するためにNetScalerとKubernetesノード間にVXLANトンネルを確立する必要がありますが、NetScalerクラスターでのVXLANの作成はサポートされていません。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

自己修復ルーターポッドとネイティブヘルスチェック

NetScaler Node Controllerリリース3.1.0以降、適格なノードごとに1つの`kube-cnc-router`ポッドをデプロイして、そのノードとNetScaler間のVXLANオーバーレイを構築できます。この機能強化により、そのフリートは自己修復可能になり、Kubernetesネイティブのヘルスチェックが追加されるため、一時的な問題から回復する単一のノードは手動での操作を必要としません。

この機能により、以下が得られます。

- ノードごとの自己修復 — ルーターポッドが削除されたり、あるノードで失敗した場合、コントローラーはそのノードのポッドのみを再作成します。コントローラーの再起動は不要で、他のノードへの影響もありません。
- ノードコントローラーとルーターポッドの両方におけるネイティブのliveness/readinessプローブ。
- 安定したVTEP IP — ノードはポッドの再作成やコントローラーの再起動後も同じオーバーレイIPを保持するため、ADC PBRのネクストホップは変更されません。

以前は、コントローラーはノードイベントのみを監視し、ルーターConfigMap内の`Host-<nodeId>`キーの存在をポッドが存在することの証拠として使用していました。ルーターポッドが削除された場合（たとえば、ノードネットワークの一時的な障害の後）、キーは残っていたため、コントローラーは`Update not required as pod entry found in cmap`をログに記録し、ポッドを再作成することはありませんでした。唯一の回避策は、Node Controllerを再起動することでしたが、これはクラスター内のすべてのルーターポッドを再作成するため、正常なノードに影響を与えました。

この機能の仕組み

- **自己修復:** ポッドのライフサイクルウォッチャーは、ノードの削除、障害、またはスタック状態 (ノードが存在し、準備ができており、ノードのテイントを許容することを確認した後) で、そのノードのポッドを再作成します。
- **ヘルスチェック:** ノードコントローラー — readiness = インフォーマーキャッシュが同期済み、liveness = 内部ハートビートが最新。ルーターポッド — readiness = オーバーレイ設定完了、liveness = VXLANインターフェースが存在し、そのアドレスを保持している (またはポッドが意図的に停止している)。
- **安定したVTEP IP:** 各ノードのVTEP CIDRは、専用の`<router-configmap-name>-vtep` ConfigMapに記録され、再作成/再起動のたびに再利用されます。ノードが削除された場合にのみ解放されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

前提条件

- Kubernetes環境を使用している場合は、Kubernetes バージョン1.6以降。
- OpenShiftプラットフォームを使用している場合は、OpenShift バージョン4.8以降。
- Helm バージョン2.x以降。Helmのインストール に記載されている手順に従ってインストールできます。
- コントローラーがNetScalerと通信するために必要なイングリッドNetScaler IPアドレスを決定します。IPアドレスは、NetScalerの展開タイプに応じて次のいずれかになります。
  - (スタンドアロンアプライアンス) NSIP - スタンドアロンNetScalerの管理IPアドレス。詳細については、「NetScalerでのIPアドレス指定」を参照してください。
  - (高可用性モードのアプライアンス) SNIP - サブネットIPアドレス。詳細については、「NetScalerでのIPアドレス指定」を参照してください。
- イングレスNetScaler SNIPを決定します。このIPアドレスは、コントローラーがNetScalerと通信するために必要なKubernetesクラスター間にオーバーレイネットワークを確立するために使用されます。
- イングレスデバイスとして使用されるNetScaler VPXまたはMPXアプライアンスのユーザー名とパスワード。ノードコントローラーがNetScaler VPXまたはMPXアプライアンスを構成できるように、NetScalerには特定の権限を持つシステムユーザーアカウント (非デフォルト) が必要です。NetScalerでシステムユーザーアカウントを作成する手順については、「NetScalerでNSNCのシステムユーザーアカウントを作成する」を参照してください。

  Kubernetesシークレットを使用してユーザー名とパスワードを渡す必要があります。次のコマンドを使用して、ユーザー名とパスワードのKubernetesシークレットを作成します。

  ```
  kubectl create secret generic nsloginns --from-literal=username='nsnc'
  --from-literal=password='mypassword'
  ```

NetScalerでNetScalerノードコントローラーのシステムユーザーアカウントを作成する

ノードコントローラーは、NetScalerのシステムユーザーアカウントを使用してNetScalerを設定します。NSNCがNetScalerで以下を設定する権限を持つように、システムユーザーアカウントは特定の権限を持っている必要があります。

- ルートの追加、削除、または表示
- ARPの追加、削除、または表示
- VXLANの追加、削除、または表示

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- IPの追加、削除、または表示

注記:

システムユーザーアカウントは、定義するコマンドポリシーに基づいて権限を持っている必要があります。

システムユーザーアカウントを作成するには、次の手順を実行します。

1. NetScalerアプライアンスにログオンします。次の手順を実行します。
   a. PuTTyなどのSSHクライアントを使用して、NetScalerアプライアンスへのSSH接続を開きます。
   b. 管理者資格情報を使用してアプライアンスにログオンします。
2. 次のコマンドを使用してシステムユーザーアカウントを作成します。

```
add system user <username> <password>
```

例:

```
add system user nsnc mypassword
```

3. システムユーザーアカウントに必要な権限を付与するポリシーを作成します。次のコマンドを使用します。

```
add cmdpolicy nsnc-policy ALLOW  (^\S+\s+arp)|(^\S+\s+arp\s+.*)|(^\S+
\s+route)|(^\S+\s+route\s+.*)|(^\S+\s+vxlan)|(^\S+\s+vxlan\s+.*)|(^\S+
\s+ns\s+ip)|(^\S+\s+ns\s+ip\s+.*)|(^\S+\s+bridgetable)|(^\S+
\s+bridgetable\s+.*)
```

4. 次のコマンドを使用して、ポリシーをシステムユーザーアカウントにバインドします。

```
bind system user nsnc nsnc-policy 0
```

NetScalerノードコントローラーを展開する

ノードコントローラーを使用してネットワーク接続を確立するには:

1. NetScaler Ingress Controller をデプロイします。次の手順を実行します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

a. 次のコマンドを使用して、ノードコントローラーのHelmチャートリポジトリを追加します。

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/
```

b. リリース名 `nsnc` を使用してチャートをインストールするには、次のコマンドを使用します。

```
helm install nsnc netscaler/netscaler-node-controller --set license.accept=yes,nsIP=<NSIP>,vtepIP=<NetScaler SNIP>,vxlan.id=<VXLAN ID>,vxlan.port=<VXLAN PORT>,network=<IP-address-range-for-VTEP-overlay>,adcCredentialSecret=<Secret-for-NetScaler-credentials>
```

または、この後に示すように `values.yaml` ファイルを作成し、次のコマンドを使用することもできます。

```
helm install nsnc netscaler/netscaler-node-controller -f values.yaml
```

Values.yaml

```
adcCredentialSecret: "nsloginns" # Kubernetes Secret created for login into Netscaler as mentioned in pre-requisite section.

clusterName: "" # For e.g. "cluster1", provide the name of the cluster, provide same cluster name in NetScaler Ingress Controller deployment as well.

cniType: "" # provide the CNI supported (canal, flannel, Calico, cilium, weave)

license:

  accept: 'yes'

network: "<network>" #for e.g. 172.16.3.0/24 # Give a network IP not conflicting with pod IP range and cluster IP range

nsIP: <IP-ADDRESS>  # Management IP of the NetScaler (can be NSIP or
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
SNIP with management access enabled for HA and CLIP for cluster)

vtepIP: <IP-ADDRESS>  # SNIP IP of the NetScaler

healthProbes:

  enabled: true

vxlan:

  id: <ID> # VXLAN ID that you want to use

  port: <PORT> # VXLAN PORT that you want to use
```

注:

`healthProbes.enabled` パラメーターは、Kubernetes の活性プローブと準備プローブがノードコントローラーに追加されるかどうかを制御します。これらのプローブにより、Kubernetes は異常なコントローラーを自動的に検出し、再起動できます。このパラメーターはデフォルトで有効 (`true`) になっており、ノードコントローラーイメージバージョン 3.1.0 以降が必要です。以前のイメージバージョンを使用している場合は、このパラメーターを `false` に設定してください。

異なる引数の詳細については、必須およびオプションのパラメーター を参照してください。

2. デプロイメントを確認します。

ノードコントローラーをデプロイした後、ノードコントローラーがNetScaler上にルートを設定したかどうかを確認できます。

確認するには、NetScalerにログオンし、次のコマンドを使用して、ノードコントローラーによってNetScalerに設定されたVXLAN VNID、VXLAN PORT、SNIP、ルート、およびブリッジテーブルを確認します。

スクリーンショットのハイライトは、NetScaler 上でノードコントローラーによって設定された VXLAN VNID、VXLAN PORT、SNIP、ルート、およびブリッジテーブルを示しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

クラスターデプロイメントの検証

`netscaler-node-controller` デプロイメントとは別に、他のリソースも作成されます。

- NSNC がデプロイされている名前空間で:
  - 各ワーカーノードに、`kube-cnc-router` ポッド。
  - ConfigMap `kube-cnc-router`。

各ワーカーノードに、インターフェース `cncvxlan<hash-of-namespace>` と `iptables` ルールが作成されます。

```
# Delete one node's router pod; a replacement appears within ~30s, other
nodes are

# untouched, and the new pod keeps the same VTEP IP.

kubectl delete pod kube-cnc-router-<nodeId> -n <namespace>

kubectl get pods -n <namespace> -w



# The per-node IP allocation survives recreates/restarts

kubectl get configmap <router-configmap-name>-vtep -n <namespace> -o yaml



# Probes on the pod spec

kubectl describe pod <pod> -n <namespace> | grep -A2 -E 'Liveness|
Readiness'
```

NetScaler Kubernetes ノードコントローラーの削除

```
helm delete nsnc
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

インターネットアクセスなしで NetScaler ノードコントローラーを実行する

ノードコントローラーは、各 Kubernetes クラスターノードにヘルパーポッド (`kube-cnc-router` ポッド) を内部的に作成します。デフォルトで使用されるイメージは、インターネットアクセスを必要とする `quay.io/citrix/cnc-router:1.1.0` です。Kubernetes ノードがインターネットアクセスを持たない場合、`kube-cnc-router` ポッドの作成は失敗します。

しかし、NetScaler は、内部リポジトリからイメージにアクセスする方法を提供しており、インターネットアクセスなしでノードコントローラーを実行できます。`CNC_ROUTER_IMAGE` 環境変数を使用すると、`quay.io/citrix/cnc-router:1.1.0` の内部リポジトリイメージを指すことができます。

内部リポジトリのイメージを使用するように NetScaler ノードコントローラーを設定する

ノードコントローラーをデプロイする際に、`CNC_ROUTER_IMAGE` 環境変数を指定し、その変数の値をイメージ `quay.io/citrix/cnc-router:1.1.0` の内部リポジトリパスとして設定します。

この環境変数を指定すると、ノードコントローラーは、`CNC_ROUTER_IMAGE` 環境変数によって提供される内部リポジトリイメージを使用して、`kube-cnc-router` ヘルパーポッドを作成します。この環境変数が指定されていない場合、インターネットアクセスを必要とするデフォルトイメージ `quay.io/citrix/cnc-router:1.1.0` を使用します。ノードコントローラーをデプロイする際に、`nsncRouterImage` を設定してください。

```
nsncRouterImage: "quay.io/citrix/cnc-router"
```

同じクラスターで複数のNetScalerノードコントローラーを実行する

同じクラスターに複数のノードコントローラーをデプロイする場合は、ノードコントローラーをデプロイする際に以下の引数を設定して、`CNC_ROUTER_NAME` に別の名前を使用してください。

```
nsncRouterName: "abc-nsnc-router"
```

ノードコントローラーのデプロイのトラブルシューティングについては、ノードコントローラーのトラブルシューティング を参照してください。

NetScalerノードコントローラーを使用したOpenShift向けOVN-CNIサポート

ノードコントローラーとルーターポッドは、OpenShift 4.xクラスターのデフォルトCNIであるOVN-Kubernetesをサポートするようになりました。OpenShiftノードとNetScaler®の間で、`ovn-k8s-mp0` インターフェースを使用してVXLANオーバーレイネットワークが確立されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Node Controllerは、NetScalerへのVXLANオーバーレイを構築するために、各ノードで1つのkube-cnc-routerポッドを実行します。この機能強化により、そのポッドセットは自己修復可能になります。ルーターポッドが削除されたり、単一ノードで障害が発生したりした場合（例えば、ノードネットワークの一時的な問題の後など）、Node Controllerはそのノードのポッドのみを自動的に再作成します。これは、Node Controller全体を再起動することなく、健全なノードを中断することなく行われます。各ノードは、ポッドの再作成やコントローラーの再起動後も安定したVTEP IPを維持します。さらに、この機能強化により、Kubernetesネイティブの活性プローブと準備プローブがNode Controllerとルーターポッドの両方に追加されます。詳細については、「自己修復ルーターポッドとネイティブヘルスチェック」を参照してください。

OVN-Kubernetesを使用してOpenShiftをデプロイする

前提条件

- OVN-Kubernetes CNIを備えたOpenShift 4.xクラスター
- すべてのクラスターノードから到達可能なNetScaler (NetScaler)
- VTEPオーバーレイ用の専用サブネット（ポッドまたはノードCIDRと重複しないこと）
- `kubectl` / `oc` CLI（クラスター管理者アクセス権付き）

ステップ1: 名前空間とシークレットを作成する

```
oc new-project netscaler



kubectl create secret generic nslogin \

  --from-literal=username='<your-adc-username>' \

  --from-literal=password='<your-adc-password>' \

  -n netscaler
```

NetScalerでNSNCのシステムユーザーアカウントを作成するを参照してください

ステップ2: セキュリティコンテキスト制約を作成する（OpenShiftのみ）

ルーターポッドはOpenShiftで特権アクセスを必要とします。サービスアカウントのセキュリティコンテキスト制約（SCC）バインディングを作成します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
oc adm policy add-scc-to-user privileged
system:serviceaccount:<namespace>:<service account of node controller>
```

`<namespace>`と`<service account of node controller>`を更新します。

ステップ3: ノードコントローラーをデプロイする

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/



helm install nsnc netscaler/netscaler-node-controller --set
license.accept=yes,nsIP=<NSIP>,vtepIP=<NetScaler SNIP>,vxlan.id=<VXLAN
ID>,vxlan.port=<VXLAN PORT>,network=<IP-address-range-for-VTEP-
overlay>,adcCredentialSecret=<Secret-for-NetScaler-
credentials>,cniType=<CNI-overlay-name>,nsncRouterImage=<Image of
nsncRouter>
```

変数	説明	例
`nsIP`	NetScaler管理IP (NSIP/SNIP/CLIP)	-
`adcCredentialSecret`	NetScaler資格情報 (シークレット `nslogin` 経由)	—
`network`	VTEPオーバーレイサブネット — ポッド/ノードCIDRと重複してはなりません	`172.16.3.0/24`
`vtepIP`	VTEPエンドポイントとして使用されるNetScaler SNIP	`10.10.10.2`
`vxlan.id`	VXLAN VNI — NetScaler上の既存のVXLANと競合してはなりません	`175`
`vxlan.port`	VXLAN UDPポート — NetScaler上の既存のVXLANと競合してはなりません	`8472`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

変数	説明	例
`cniType`	**OpenShift OVN-Kubernetes の場合は `ovn` に設定します**	`ovn`
`nsncRouterImage`	ノードコントローラールーターイメージ	`quay.io/netscaler/nsnc-router:2.0.0`
`image`	ノードコントローラーイメージ	`quay.io/netscaler/netscaler-k8s-node-controller/3.0.0`

ステップ4: デプロイの検証

ノードコントローラーポッドが実行中であることを確認します:

```
kubectl get pods -n netscaler
```

ノードごとにルーターポッドが作成されていることを確認します:

```
kubectl get pods -n netscaler | grep cnc-router
```

CNCルーターConfigMapが設定されていることを確認します:

```
kubectl get configmap -n netscaler -o yaml
```

各ノードには次のエントリが必要です: `Host-<node>`, `Node-<ip>`, `Mac-<ip>`, `Interface-<ip>`, `CNI-<ip>`。

NetScalerで検証します:

```
show vxlan

show bridgetable

show route

show ip
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

期待される結果: VXLANトンネル、ノードごとのブリッジテーブルエントリ、SNIP、およびポッドネットワークルートがすべて存在します。

クリーンアップ / アンインストール

```
helm delete nsnc -n netscaler
```

制限事項

- ノードコントローラーの削除中に、NetScalerからブリッジテーブルエントリはクリーンアップされません。(ルート、VXLAN、SNIPのみが削除されます)。

Ingress を使用して NodePort タイプのサービスを公開する

単一階層展開では、Kubernetes クラスターの外部にある Ingress NetScaler (VPX または MPX) が、Kubernetes クラスターに展開されたマイクロサービスへのすべての Ingress トラフィックを受信します。Ingress トラフィックがマイクロサービスに到達するためには、Ingress NetScaler インスタンスとポッド間のネットワーク接続を確立する必要があります。

ポッドはオーバーレイネットワーク上で実行されるため、ポッドの IP アドレスはプライベート IP アドレスであり、Ingress NetScaler インスタンスはポッド内で実行されているマイクロサービスに到達できません。クラスターの外部からサービスにアクセスできるようにするには、NodePort タイプのサービスを作成できます。NetScaler インスタンスは、Ingress トラフィックをポッドを含むノードに負荷分散します。

NodePort タイプのサービスを作成するには、サービス定義ファイルで `spec.type:NodePort` を指定し、オプションで 30000〜32767 の範囲のポートを指定します。

展開例

NodePort ベースのサービス (たとえば、`apache` アプリ) を使用しており、Ingress を使用して南北トラフィックにアプリを公開したいシナリオを考えてみましょう。この場合、`apache` アプリの展開を作成し、`NodePort` タイプのサービスを定義し、Ingress NetScaler が南北トラフィックを `apache` アプリのノードポートに送信するように構成する Ingress 定義を作成する必要があります。

この例では、`apache` という名前の展開を作成し、Kubernetes クラスターに展開します。

1. `apache-deployment.yaml` という名前の展開のマニフェストを作成します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
# If using this on GKE

# Make sure you have cluster-admin role for your account

# kubectl create clusterrolebinding citrix-cluster-admin --
clusterrole=cluster-admin --user=<username of your google account>

#



#For illustration a basic apache web server is used as a application

apiVersion: apps/v1

kind: Deployment

metadata:

  name: apache

  labels:

      name: apache

spec:

  selector:

    matchLabels:

      app: apache

  replicas: 4

  template:

    metadata:

      labels:

        app: apache
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    spec:
      containers:
      - name: apache
        image: httpd:latest
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
```

この展開のコンテナはポート 80 でリッスンします。

2. 次のコマンドを使用して展開を作成します。

```
kubectl create -f apache-deployment.yaml
```

3. 次のコマンドを使用して、4 つのポッドが実行されていることを確認します。

```
kubectl get pods
```

4. ポッドが起動して実行されていることを確認したら、`NodePort` タイプのサービスを作成します。サービスの次のマニフェストです。

```
#Expose the apache web server as a Service
apiVersion: v1
kind: Service
metadata:
  name: apache
  labels:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    name: apache
spec:
  type: NodePort
  ports:
  - name: http
    port: 80
    targetPort: http
  selector:
    app: apache
```

5. マニフェストを `apache-service.yaml` という名前のファイルにコピーし、次のコマンドを使用してサービスを作成します。

```
kubectl create -f apache-service.yaml
```

このサンプルは、Apache Web サーバーをサービスとして展開および公開します。`<NodeIP>:<NodePort>` アドレスを使用してサービスにアクセスできます。

6. サービスを展開したら、Ingress リソースを作成して、Ingress NetScaler が南北トラフィックを `apache` アプリのノードポートに送信するように構成します。`vpx-ingress.yaml` という名前の Ingress 定義のマニフェストは次のとおりです。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.citrix.com/frontend-ip: xx.xxx.xxx.xx
  name: vpx-ingress
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:

  defaultBackend:

    service:

      name: apache

      port:

        number: 80
```

7. Ingressオブジェクトをデプロイします。

```
kubectl create -f vpx-ingress.yaml
```

Calico を使用した Pod 間通信の構成

Kubernetes でネットワークを構成することは困難です。クラスターシステム内の多数のノードと Pod を処理する必要があります。ネットワークを構成する際に、対処すべき4つの問題があります。

- コンテナ間 (集合的にサービスを提供する) 通信
- Pod 間通信
- Pod からサービスへの通信
- 外部からサービスへの通信

Pod 間通信

デフォルトでは、Docker はホストマシン上に `docker0` と呼ばれる仮想ブリッジを作成し、それにプライベートネットワーク範囲を割り当てます。作成される各コンテナには、このブリッジに仮想イーサネットデバイスが接続されます。仮想イーサネットデバイスは、ネットワーク範囲からの IP を使用して、コンテナ内の `eth0` にマッピングされます。このプロセスは、Docker を実行している各ホストで発生します。これらのホスト間には連携がないため、ネットワーク範囲が衝突する可能性があります。

このため、コンテナは接続されているコンテナとのみ通信できます。 同じ仮想ブリッジに。他のホスト上の他のコンテナと通信するには、ポートマッピングに依存する必要があります。つま

© 1997–2026 Citrix Systems, Inc. All rights reserved.

り、ホストマシン上のポートを各コンテナに割り当て、そのポート上のすべてのトラフィックをそのコンテナに転送する必要があります。

アプリケーションのローカル IP アドレスはホストマシン上のホスト IP アドレスとポートに変換されるため、Kubernetes はすべてのノードが NAT なしで相互に通信できると想定しています。また、コンテナが自身に対して認識する IP アドレスは、他のコンテナがそのコンテナに対して認識する IP アドレスと同じであると想定しています。このアプローチにより、仮想マシンからコンテナへのアプリケーションの移植も容易になります。

Calico は、Kubernetes にこれらの機能を提供する多くの異なるネットワークオプションの1つです。

カリコ

Calico は、クラウドネットワークを簡素化、スケーリング、および保護するように設計されています。このオープンソースフレームワークは クラウド全体のクラスター向けに Kubernetes ネットワークとネットワークポリシーを可能にします。Kubernetes エコシステム内で、Calico は最も人気のあるネットワークフレームワークまたはプラグインの1つとして台頭し始めており、多くの企業が大規模に利用しています。

Calico は純粋な IP ネットワークファブリックを使用して高性能な Kubernetes ネットワークを提供し、そのポリシーエンジンは高レベルのネットワークポリシー管理のための開発者の意図を強制します。Calico はレイヤー3ネットワーク機能を提供し、各ノードに仮想ルーターを関連付けます。これにより、ホスト間および Pod 間ネットワークが可能になります。Calico は、BGP を介したゾーン境界の確立、または IP over IP または VXLAN メソッドを介したカプセル化を可能にします。

Kubernetes と Calico の統合

Calicoは、完全に分散されたレイヤー3アーキテクチャ上に構築されたCNIプラグインを介してKubernetesと統合されます。そのため、単一のラップトップから大規模なエンタープライズまでスムーズに拡張できます。IPレイヤーに依存しており、既存のツールでデバッグするのが比較的簡単です。

Calicoでネットワークを構成する

まず、以下のコマンドを使用してCalicoでKubernetesクラスターを起動します。

```
> kubeadm init --pod-network-cidr=192.168.0.0/16

> export KUBECONFIG=/etc/kubernetes/admin.conf
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
> kubectl apply -f calico.yaml
```

CalicoをCNIとして使用してマスターノードが作成されます。マスターノードが起動して実行されたら、`join`コマンドを使用して他のノードをマスターに参加させることができます。

Kubernetesマスターノードの一部であるCalicoプロセスは次のとおりです。

- カリコ etcd

  kube-system calico-etcd-j4rwc 1/1 現在、正常に稼働しています
- Calicoコントローラー

  kube-system calico-kube-controllers-679568f47c-vz69g 1/1 現在、正常に稼働しています
- Calicoノード

  kube-system calico-node-ct6c9 2/2 実行中

注:

ノードをKubernetesクラスターに参加させると、新しい*Calicoノード*がKubernetesノードで初期化されます。

Ingress NetScalerでBGPピアを構成する

クラスターにCalicoネットワークを確立した後でアプリケーションをデプロイするたびに、KubernetesはCalicoのIPアドレスプールからアプリケーションに関連付けられたサービスにIPアドレスを割り当てます。

Border Gateway Protocol (BGP) は、リモートノードを識別するために autonomous system number (AS number) を使用します。AS番号は、IANAによって割り当てられる特別な番号で、主にBGPで、独自のルーティングポリシーを使用する単一のネットワーク管理下にあるネットワークを識別するために使用されます。

INGRESS NETSCALER を使用して KUBERNETES で BGP を構成する

YAMLファイルを使用して、`kubectl create`コマンドでリモートノードのBGP構成を適用できます。YAMLファイルには、ピアIPアドレスとAS番号を追加する必要があります。ピアIPアドレスはIngress NetScalerのIPアドレスであり、AS番号はIngress NetScalerで使用されているAS番号です。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

クラスターのAS番号を取得する

次の図に示すように、`calicoctl` コマンドを使用して、KubernetesクラスターでCalico BGPが使用するAS番号を取得できます。

グローバルBGPピアを構成する

`calicoctl` ユーティリティを使用して、Calicoノードをグローバルbgpスピーカーとピアリングできます。この種のピアはグローバルピアと呼ばれます。

次の定義で、`bgp.yml` というYAML定義ファイルを作成します。

```
apiVersion: projectcalico.org/v3  # This is the version of Calico
kind: BGPPeer  # BGPPeer specifies that its Global peering.
metadata:
    name: bgppeer-global-3040  # The name of the configuration
spec:
    peerIP: 10.102.33.208  # IP address of the Ingress NetScaler
    asNumber: 500  # AS number configured on the Ingress NetScaler
```

次のコマンドを使用して定義ファイルをデプロイします。

```
> kubectl create -f bgp.yml
```

INGRESS NETSCALER にBGP構成を追加する

以下を実行します。

1. NetScaler コマンドラインインターフェイスにログオンします。
2. 次のコマンドを使用してBGP機能を有効にします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
> en feature bgp

  Done
```

3. `vtysh` と入力し、**Enter** を押します。

```
> vtysh

ns#
```

4. `conf t` コマンドを使用して、config terminal に変更します。

```
ns#conf t

Enter configuration commands, one per line. End with CNTL/Z.

ns(config)#
```

5. デモンストレーション目的で、AS番号を500としてBGPルートを追加します。AS番号には任意の数字を使用できます。

```
ns(config)# router bgp 500

ns(config-router)#
```

6. 次のコマンドを使用してネイバーを追加します。

```
ns(config-router)# Neighbor 10.102.33.198 remote-as 64512

ns(config-router)# Neighbor 10.102.22.202 remote-as 64512
```

7. 次のコマンドを使用して実行中の設定を確認します。

```
ns(config-router)#show running-config

!

log syslog
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
log record-priority

!

ns route-install bgp

!

interface lo0

 ip adress 127.0.0.1/8

 ipv6 address fe80: :1/64

 ipv6 address : :1/128

!

interface vlan0

 ip address 10.102.33.208/24

 ipv6 address fe80::2cf6:beff:fe94:9f63/64

!

router bgp 500

 max-paths ebgp 8

 max-paths ibgp 8

 neighbor 10.102.33.198 remote-as 64512

 neighbor 10.102.33.202 remote-as 64512

!

end

ns(config-router)#
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

このサンプルでは、CalicoのAS番号は64512ですが、この番号は要件に応じて変更できます。

8. 次のコマンドを使用して、NetScalerルーティングテーブルにBGPルートをインストールします。

```
ns(config)# ns route-install bgp

ns(config)#

exit

ns#exit

 Done
```

9. 次のコマンドを使用して、ルートを確認し、ルーティングテーブルに追加します。

ルートがインストールされると、NetScalerはKubernetesクラスター内に存在するサービスと通信できるようになります。

トラブルシューティング

Kubernetesクラスターのマスターノードで、`calicoctl` スクリプトを使用してBGP設定を確認できます。

ピアIPアドレスとAS番号の設定を表示する

次のコマンドを使用して、ピアIPアドレスとAS番号の設定を表示できます。

```
>./calicoctl.1 get bgpPeer

NAME                    PEERIP          NODE        ASN

bgppeer-global-3040  10.102.33.208    (global)      500
```

BGPノードのステータスを表示する

以下のコマンドを使用して、BGPノードのステータスを確認できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
>calicoctl node status

IPV4 BGP status

+--------------+-----------+-------+----------+-------------+

| PEER ADDRESS | PEER TYPE | STATE | SINCE | INFO |

+--------------+-----------+-------+----------+-------------+

| 10.102.33.208 | global | up | 16:38:14 | Established |

+--------------+-----------+-------+----------+-------------+
```

NetScaler Ingress ControllerにおけるLoadBalancerタイプのKubernetesサービスサポートの機能強化

NetScaler Ingress ControllerにおけるLoadBalancerタイプのKubernetesサービスサポートは、以下の機能で強化されています。

- BGPルートヘルスインジェクション (RHI) のサポート
- ノードのラベルで定義された一連のノード (ゾーン) におけるサービスのポッドの可用性に基づいて、ロードバランサーIPアドレス (VIP) をアドバタイズまたはリコールする

NetScalerでのBGP RHIの自動構成のサポート

ルートヘルスインジェクション (RHI) により、NetScalerはBGPを使用してネットワーク全体にホストルートとしてVIPの可用性をアドバタイズできます。ただし、RHIをサポートするには、NetScalerで手動で構成を実行する必要がありました。Kubernetes環境に展開されたNetScaler Ingress Controllerを使用すると、NetScalerでのVIPアドバタイズの構成を自動化できます。

`LoadBalancer` タイプのサービスが作成されると、NetScaler Ingress ControllerはサービスのVIPをNetScalerに構成します。NetScaler Ingress ControllerでBGP RHIサポートが有効になっている場合、NetScalerはVIPをBGPネットワークにアドバタイズするように自動的に構成されます。`service.citrix.com/vipparams` アノテーションを使用すると、VIPのIPパラメーターを有効にできます。たとえば、NetScaler Ingress Controllerを使用したNetScalerでのBGP RHIの構成 のステップ5にある `service.YAML` ファイルを参照してください。サポートされているIPパラメーターの詳細については、nsip configuration を参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ポッドの可用性に基づいてVIPをアドバタイズおよびリコールする

次の図に示すトポロジでは、Kubernetesクラスター内のノードは3つの異なるラックに物理的に分散されています。それらは論理的に3つのゾーンにグループ化されています。各ゾーンには、Tier-1 ADCとしてNetScaler MPXがあり、Kubernetesクラスター内の同じ場所にNetScaler Ingress Controllerがあります。すべてのゾーンのNetScaler Ingress Controllerは、同じKubernetes APIサーバーをリッスンします。したがって、`LoadBalancer` タイプのサービスが作成されるたびに、クラスター内のすべてのNetScalerは同じIPアドレスをBGPファブリックにアドバタイズします。ゾーンにワークロードがない場合でも、そのゾーンのNetScalerはIPアドレスをアドバタイズします。

NetScalerは、ゾーン内のポッドの可用性に基づいてVIPをアドバタイズまたはリコールするソリューションを提供します。NetScaler Ingress Controllerが同じゾーンに属するノードを識別できるように、各ゾーンのノードにラベルを付ける必要があります。各ゾーンのNetScaler Ingress Controllerは、ゾーン内のノードにポッドがあるかどうかを確認します。ゾーン内のノードにポッドがある場合、VIPをアドバタイズします。それ以外の場合は、そのゾーンのNetScalerからVIPのアドバタイズを取り消します。

NetScaler Ingress Controllerを使用したNetScalerでのBGP RHIの構成

このトピックでは、サンプル トポロジに基づいて、NetScaler Ingress Controllerを使用してNetScalerでBGP RHIを構成する方法について説明します。このトポロジでは、Kubernetesクラスター内のノードは2つのゾーンに展開されています。各ゾーンには、Tier-1 ADCとしてNetScaler VPXまたはMPXがあり、KubernetesクラスターでADCを構成するためのNetScaler Ingress Controllerがあります。ADCは、BGPを使用してアップストリームルーターとピアリングされています。

前提条件

- アップストリームルーターとBGPピアとしてNetScaler MPXまたはVPXを設定します。

サンプルトポロジに基づいてBGP RHIサポートを設定するには、次の手順を実行します。

1. 次のコマンドを使用して、各ゾーンのノードにラベルを付けます。

   ゾーン1の場合：

   ```
   kubectl label nodes node1 rack=rack-1

   kubectl label nodes node2 rack=rack-1
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ゾーン2の場合：

```
kubectl label nodes node3 rack=rack-2

kubectl label nodes node4 rack=rack-2
```

2. NetScaler Ingress Controller構成YAMLファイルで、次の環境変数を次のように設定します。

   ゾーン1の場合：

```
- name: "NODE_LABELS"

  value: "rack-1"

- name: "BGP_ADVERTISEMENT"

  value: "True"
```

   ゾーン2の場合：

```
- name: "NODE_LABELS"

  value: "rack-2"

- name: "BGP_ADVERTISEMENT"

  value: "True"
```

   ゾーン1にNetScaler Ingress Controllerをデプロイするためのサンプル `cic.yaml` ファイルは次のとおりです。

```
apiVersion: v1

kind: Pod

metadata:

  name: cic-k8s-ingress-controller-1

  labels:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    app: cic-k8s-ingress-controller-1
spec:
  serviceAccountName: cic-k8s-role
  containers:
  - name: cic-k8s-ingress-controller
    image: "quay.io/citrix/citrix-k8s-ingress-controller:1.36.5"



    env:
    # Set NetScaler NSIP/SNIP, SNIP in case of HA (mgmt has to be
enabled)
    - name: "NS_IP"
      value: "10.217.212.24"
   # Set username for Nitro
    - name: "NS_USER"
      valueFrom:
        secretKeyRef:
          name: nslogin
          key: username
   # Set user password for Nitro
    - name: "NS_PASSWORD"
      valueFrom:
        secretKeyRef:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        name: nslogin

        key: password

    - name: "EULA"

      value: "yes"

    - name: "NODE_LABELS"

      value: "rack=rack-1"

    - name: "BGP_ADVERTISEMENT"

      value: "True"

args:

  - --ipam

    citrix-ipam-controller

imagePullPolicy: Always
```

3. 次のコマンドを使用してNetScaler Ingress Controllerをデプロイします。

   注：

   NetScaler Ingress Controllerを両方のラック（ゾーンごと）にデプロイする必要があります。

   ```
   Kubectl create -f cic.yaml
   ```

4. `web-frontend-lb.yaml`ファイルを使用してサンプルアプリケーションをデプロイします。

   Kubectl create -f web-frontend-lb.yaml を実行します。

   `web-frontend-lb.yaml`の内容は次のとおりです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: v1

kind: Deployment

metadata:

  name: web-frontend

spec:

  selector:

    matchLabels:

      app: web-frontend

  replicas: 4

  template:

    metadata:

      labels:

        app: web-frontend

    spec:

      containers:

      - name: web-frontend

        image: 10.217.6.101:5000/web-test:latest

        ports:

          - containerPort: 80

        imagePullPolicy: Always
```

5. アプリケーションを公開するために、`LoadBalancer`タイプのサービスを作成します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
Kubectl create -f web-frontend-lb-service.yaml
```

`web-frontend-lb-service.yaml`の内容は以下のとおりです。

```
apiVersion: v1

kind: Service

metadata:

  name: web-frontend

  annotations:

    service.citrix.com/class: 'cic-vpx'

    service.citrix.com/frontend-ip: 1.1.1.1

    service.citrix.com/vipparams: '{"vserverrhilevel": "ONE_VSERVER",
"hostroute": "ENABLED", "metric": 10}'

  labels:

    app: web-frontend

spec:

  type: LoadBalancer

  ports:

  - port: 80

    protocol: TCP

    name: http

  selector:

    app: web-frontend
```

6. 以下のコマンドを使用して、NetScaler上でサービスグループの作成を確認します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
show servicegroup &lt;service-group-name>
```

以下は、そのコマンドのサンプル出力です。

```
#  show servicegroup k8s-web-frontend_default_80_svc_k8s-web-
frontend_default_80_svc



k8s-web-frontend_default_80_svc_k8s-web-frontend_default_80_svc - TCP

State: ENABLED Effective State: UP Monitor Threshold : 0

Max Conn: 0 Max Req: 0 Max Bandwidth: 0 kbits

Use Source IP: NO

Client Keepalive(CKA): NO

TCP Buffering(TCPB): NO

HTTP Compression(CMP): NO

Idle timeout: Client: 9000 sec Server: 9000 sec

Client IP: DISABLED

Cacheable: NO

SC: OFF

SP: OFF

Down state flush: ENABLED

Monitor Connection Close : NONE

Appflow® logging: ENABLED

ContentInspection profile name: ???

Process Local: DISABLED
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
Traffic Domain: 0




1)   10.217.212.23:30126 State: UP Server Name: 10.217.212.23 Server
ID: None Weight: 1

  Last state change was at Wed Jan 22 23:35:11 2020

  Time since last state change: 5 days, 00:45:09.760



  Monitor Name: tcp-default  State: UP Passive: 0

  Probes: 86941 Failed [Total: 0 Current: 0]

  Last response: Success - TCP syn+ack received.

  Response Time: 0 millisec



2)   10.217.212.22:30126 State: UP Server Name: 10.217.212.22 Server
ID: None Weight: 1

  Last state change was at Wed Jan 22 23:35:11 2020

  Time since last state change: 5 days, 00:45:09.790



  Monitor Name: tcp-default State: UP Passive: 0

  Probes: 86941 Failed [Total: 0 Current: 0]

  Last response: Success - TCP syn+ack received.
```

7. 以下のコマンドを使用して、BGPルーター上でVIPアドバタイズメントを確認します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
 >VTYSH

# show ip route bgp

  B       172.29.46.78/32   [200/0] via 2.2.2.100, vlan20, 1d00h35m

                            [200/0] via 2.2.2.101, vlan20, 1d00h35m

  Gateway of last resort is not set
```

証明書管理

証明書管理には、次のトピックがあります。

- NetScaler Ingress ControllerでのTLS証明書の処理
- NetScalerでのTLSクライアント認証
- NetScalerでのTLSサーバー認証
- NetScaler Ingress Controllerを使用してNetScalerに証明書をインストール、リンク、および更新する
- Kubernetes Ingressを使用したSSLパススルーの構成
- cert-managerによる自動証明書管理の概要
- NetScaler Ingress Controllerとcert-managerを使用して、K8sにHTTPS Webアプリケーションをデプロイし、Let's Encryptを使用する
- NetScaler Ingress Controllerとcert-managerを使用して、K8sにHTTPS Webアプリケーションをデプロイし、HashiCorp Vaultを使用する
- NetScaler証明書検証を有効にする
- APIサーバー証明書の検証を無効にする
- 自己署名証明書を作成し、Kubernetesシークレットにリンクする

NetScaler Ingress ControllerにおけるTLS証明書の処理

NetScaler Ingress Controllerは、NetScalerのSSLベースの仮想サーバー向けにTLS証明書を設定するオプションを提供します。SSL仮想サーバーはSSLトラフィックを傍受し、復号化して処理した後、仮想サーバーにバインドされているサービスに送信します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

デフォルトでは、SSL仮想サーバーは1つのデフォルト証明書にバインドでき、アプリケーションは証明書にバインドされたポリシーに基づいてトラフィックを受信します。ただし、Server Name Indication (SNI) オプションを使用すると、複数の証明書を単一の仮想サーバーにバインドできます。NetScalerは、TLSハンドシェイクのドメイン名に基づいて、クライアントに提示する証明書を決定します。

NetScaler Ingress Controllerは、以下の3つの方法で証明書を処理します。

- NetScaler Ingress Controllerのデフォルト証明書
- 事前設定された証明書
- Ingress YAMLのTLSセクション

前提条件

NetScaler Ingress Controllerを使用してTLS証明書を処理するには、アプリケーションに対してNetScalerでTLSサポートを有効にする必要があります。また、Kubernetesデプロイメントで証明書を使用している場合は、証明書を使用してKubernetesシークレットを生成する必要があります。

アプリケーション向けにNetScalerでTLSサポートを有効にする

NetScaler Ingress Controllerは、Ingress定義の**TLS**セクションをNetScalerでのTLSサポートを有効にするものとして使用します。

注:

デフォルト証明書がある場合、または事前設定された証明書がある場合は、TLSを有効にするために、Ingress定義の***spec.tls.secretname***フィールドに空のシークレットを追加する必要があります。

Ingress定義の以下のサンプルスニペット:

```
spec:

  tls:

  - secretName:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Kubernetesシークレットを生成する

既存の証明書用のKubernetesシークレットを生成するには、次の`kubectl`コマンドを使用します。

```
     kubectl create secret tls k8s-secret --cert=path/to/tls.cert --
key=path/to/tls.key  --namespace=default



    secret “k8s-secret” created
```

このコマンドは、`tls.crt`キーの下にPEM形式の証明書を、`tls.key`キーの下にPEM形式の秘密鍵を持つKubernetesシークレットを作成します。

または、次のYAML定義を使用してKubernetesシークレットを生成することもできます。

```
apiVersion: v1

kind: Secret

metadata:

  name: k8s-secret

data:

  tls.crt: base64 encoded cert

  tls.key: base64 encoded key
```

`kubectl -create <file-name>`コマンドを使用してYAMLをデプロイします。これにより、`tls.crt`キーの下にPEM形式の証明書、`tls.key`キーの下にPEM形式の秘密鍵を持つKubernetesシークレットが作成されます。

NetScaler Ingress Controllerのデフォルト証明書

NetScaler Ingress Controllerで提供されるデフォルトのシークレットは、NetScalerでSSLおよびSSL SNI証明書を設定するために使用できます。 非SNI証明書とSNI証明書をそれぞれ設定するためのシークレットを提供するには、`default-sni-certificate`と`default-ssl-sni-certificate`引数を使用できます。NetScaler Ingress Controllerのデプロイ YAMLファイルで引数

© 1997–2026 Citrix Systems, Inc. All rights reserved.

を指定する際は、シークレット名と、クラスター内でシークレットがデプロイされている名前空間を次のように指定します。

- デフォルト証明書を非SNI証明書として使用するためにYAMLファイルに追加する引数: `--default-ssl-certificate <NAMESPACE>/<SECRET_NAME>`。
- デフォルト証明書をSNI証明書として使用するためにYAMLファイルに追加する引数: `--default-ssl-sni-certificate <NAMESPACE>/<SECRET_NAME>`

注:

Helmチャートまたはオペレーターを使用してNetScaler Ingress Controllerをデプロイする場合、両方のデフォルトシークレットは、NetScaler Ingress Controllerがデプロイされているのと同じ名前空間にデプロイする必要があります。NetScaler Ingress Controllerのインストール中に、非SNIシークレット名を`defaultSSLCertSecret`パラメーターに、SNIシークレット名を`defaultSSLSNICertSecret`パラメーターに指定します。例: `defaultSSLCertSecret: <non-SNI secret name>`; `defaultSSLSNICertSecret: <SNI secret name>`。

以下は、`ssl`名前空間から取得され、NetScaler Ingress Controllerのデフォルト証明書として提供されるTLSシークレット (`hotdrink.secret`) を含むNetScaler Ingress Controller YAML定義ファイルのサンプルです。

注:

名前空間は、有効なSECRET_NAMEとともに必須です。

```
apiVersion: v1

kind: Pod

metadata:

  name: cic

  labels:

    app: cic

spec:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  serviceAccountName: cpx
  containers:
  - name: cic
    image: "xxxx"
    imagePullPolicy: Always
    args:
    - --default-ssl-certificate
      ssl/hotdrink.secret
    env:
    # Set NetScaler Console Management IP
    - name: "NS_IP"
      value: "xx.xx.xx.xx"
    # Set port for Nitro
    - name: "NS_PORT"
      value: "xx"
    # Set Protocol for Nitro
    - name: "NS_PROTOCOL"
      value: "HTTP"
    # Set username for Nitro
    - name: "NS_USER"
      value: "nsroot"
    # Set user password for Nitro
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    - name: "NS_PASSWORD"

      value: "nsroot"
```

KubernetesクラスターにおけるKubernetesイングレスに関連するさまざまなシナリオでのNetScaler Ingress Controllerの動作については、次の表を参照してください。

NETSCALER INGRESS CONTROLLERのデフォルトSSLシークレット	NETSCALER INGRESS CONTROLLERのデフォルトSSL SNIシークレット	INGRESSのホスト	INGRESSのシークレット	アクション
はい	いいえ	提供されていません	提供されていません	非SNIシークレットをSSL仮想サーバーの非SNI証明書としてバインドします。
いいえ	はい	提供されていません	提供されていません	デフォルトのSNIシークレットをSSL仮想サーバーのSNI証明書としてバインドします。
はい	はい	提供されていません	提供されていません	• 非SNIシークレットを非SNI証明書としてバイン

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NETSCALER INGRESS CONTROLLERのデフォルトSSLシークレット	NETSCALER INGRESS CONTROLLERのデフォルトSSL SNIシークレット	INGRESSのホスト	INGRESSのシークレット	アクション	
				ドします。* SNIシークレットをSNI証明書としてバインドします。* また、複数のシークレットがバインドされるため、非SNIシークレットをSNIとしてバインドします。	
はい	いいえ	提供済み	提供されていません	非SNIシークレットをSSL仮想サーバーの非SNI証明書としてバ	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NETSCALER INGRESS CONTROLLERのデフォルトSSLシークレット	NETSCALER INGRESS CONTROLLERのデフォルトSSL SNIシークレット	INGRESSのホスト	INGRESSのシークレット	アクション
				インドします。
いいえ	はい	提供済み	提供されていません	SNIシークレットをSSL仮想サーバーのSNI証明書としてバインドします。
はい	はい	提供済み	提供されていません	• 非SNIシークレットをSSL仮想サーバーの非SNI証明書としてバインドします。* SNIシークレットをSSL仮想サーバーのSNI証明書とし

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NETSCALER INGRESS CONTROLLERのデフォルトSSLシークレット	NETSCALER INGRESS CONTROLLERのデフォルトSSL SNIシークレット	INGRESSのホスト	INGRESSのシークレット	アクション
				てバインドします。* また、複数のシークレットがバインドされるため、非SNIシークレットもSNI証明書としてバインドします。
任意	任意	提供されていません	提供されています	提供されたシークレットをSSL仮想サーバーの非SNI証明書としてバインドします。
任意	何でも	提供済み	提供済み	SSL仮想サーバーでSNI証明書として提供

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NETSCALER INGRESS CONTROLLERのデフォルトSSLシークレット	NETSCALER INGRESS CONTROLLERのデフォルトSSL SNIシークレット	INGRESSのホスト	INGRESSのシークレット	アクション
				されたシークレットをバインドします。
何でも	何でも	提供済み/未提供	複数	SSL仮想サーバーでSNI証明書として提供されたすべてのシークレットをバインドします。

OpenShiftクラスター内のOpenShiftルートに関連するさまざまなシナリオにおけるNetScaler Ingress Controllerの動作については、次の表を参照してください。

NETSCALER INGRESS CONTROLLERのデフォルトSSLシークレット	NETSCALER INGRESS CONTROLLERのデフォルトSSL SNIシークレット	ルートタイプ	ルート内のキーと証明書	アクション
はい	いいえ	エッジ	提供されていません	非SNIシークレットをSSL仮想サーバーの非SNI証明書としてバインドします。
いいえ	はい	エッジ	提供されていません	デフォルトのSNIシークレットを

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NETSCALER INGRESS CONTROLLERのデフォルトSSLシークレット	NETSCALER INGRESS CONTROLLERのデフォルトSSL SNIシークレット	ルートタイプ	ルート内のキーと証明書	アクション
				SSL仮想サーバーのSNI証明書としてバインドします。
はい	はい	エッジ	提供されていません	• 非SNIシークレットを非SNI証明書としてバインドします。* SNIシークレットをSNI証明書としてバインドします。* また、複数のシークレットがSSL仮想サーバーに

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NETSCALER INGRESS CONTROLLERのデフォルトSSLシークレット	NETSCALER INGRESS CONTROLLERのデフォルトSSL SNIシークレット	ルートタイプ	ルート内のキーと証明書	アクション	
				バインドされるため、非SNIシークレットをSNIとしてバインドします。	
はい	いいえ	再暗号化	提供されていません	非SNIシークレットをSSL仮想サーバーの非SNI証明書としてバインドします。	
いいえ	はい	再暗号化	提供されていません	SNIシークレットをSSL仮想サーバーのSNI証明書としてバインドします。	
はい	はい	再暗号化	提供されていません	• 非SNIシークレット	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NETSCALER INGRESS CONTROLLERのデフォルトSSLシークレット	NETSCALER INGRESS CONTROLLERのデフォルトSSL SNIシークレット	ルートタイプ	ルート内のキーと証明書	アクション	
				をSSL仮想サーバーの非SNI証明書としてバインドします。* SNIシークレットをSNI証明書としてバインドします。* また、複数のシークレットがSSL仮想サーバーにバインドされているため、非SNI	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NETSCALER INGRESS CONTROLLERのデフォルトSSLシークレット	NETSCALER INGRESS CONTROLLERのデフォルトSSL SNIシークレット	ルートタイプ	ルート内のキーと証明書	アクション
				シークレットをSNIとしてバインドします。
はい	いいえ	パススルー	提供されていません	非SNIシークレットをSNI証明書としてバインドします。
いいえ	はい	パススルー	提供されていません	SNIシークレットをSSL仮想サーバーのSNI証明書としてバインドします。
はい	はい	パススルー	提供されていません	• 非SNIシークレットをSSL仮想サーバーのSNI証明書とし

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NETSCALER INGRESS CONTROLLERのデフォルトSSLシークレット	NETSCALER INGRESS CONTROLLERのデフォルトSSL SNIシークレット	ルートタイプ	ルート内のキーと証明書	アクション	
				てバインドします。* SNIシークレットをSSL仮想サーバーのSNI証明書としてバインドします。	
任意	任意	エッジ	提供済み	提供されたシークレットをSSL仮想サーバーのSNI証明書としてバインドします。	
何でも	何でも	再暗号化	提供済み	SSL仮想サーバーでSNI証明書として提供されたシークレットをバインドします。	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NETSCALER INGRESS CONTROLLERのデフォルトSSLシークレット	NETSCALER INGRESS CONTROLLERのデフォルトSSL SNIシークレット	ルートタイプ	ルート内のキーと証明書	アクション
何でも	何でも	パススルー	OpenShiftは許可しません	NA

事前設定された証明書

NetScaler Ingress Controllerを使用すると、NetScalerにすでに設定されている証明書キーを使用できます。イングレス定義で次のアノテーションを使用して、証明書の詳細を指定する必要があります。

```
ingress.citrix.com/preconfigured-certkey : '{"certs": [ {"name":
"&lt;name>", "type": "default|sni|ca"} ] }'
```

アノテーション内で複数の証明書の詳細をリストとして提供できます。また、証明書の処理方法を定義することもできます。次のサンプルアノテーションでは、certkey1は非SNI証明書として使用され、certkey2はSNI証明書として使用されています。

```
ingress.citrix.com/preconfigured-certkey : '{"certs": [ {"name":
"certkey1", "type": "default"}, {"name": "certkey2", "type": "sni"} ] }’
```

証明書の名前とともにtypeパラメーターが指定されていない場合、それはデフォルト（非SNI）タイプと見なされます。

NetScaler Ingress Controllerを使用すると、OpenShiftの外部で実行されているNetScaler VPX/MPX/BLXインスタンス上のルートで、すでに設定されている証明書をアノテーション（イングレスと同様）を使用して使用できます。コンテンツスイッチング仮想サーバー用に設定する必要があるNetScaler上の既存のSSL証明書キーを指定する必要があります。

```
route.citrix.com/preconfigured-certkey
```

例:

© 1997–2026 Citrix Systems, Inc. All rights reserved.

次の例では、`certkey1` は非SNIデフォルト証明書として使用され、`certkey2` はSNI証明書として使用されます。

```
route.citrix.com/preconfigured-certkey : '{"certs": [{"name": "certkey1",
"type": "default"}, {"name": "certkey2", "type": "sni"}]}'
```

注:

NetScaler上に存在する証明書を再利用し、NetScaler Ingress Controllerによって管理されるアプリケーションにバインドしたい場合に、この機能を使用するようにしてください。NetScaler Ingress Controllerは証明書のライフサイクルを管理しません。つまり、証明書を作成または削除するのではなく、必要なアプリケーションにバインドするだけです。

ingress YAML の TLS セクション

Kubernetes では、ingress 定義の `spec:` セクションで TLS シークレットを提供できます。このセクションでは、NetScaler Ingress Controller がこれらのシークレットをどのように使用するかを説明します。

ホストセクションを使用する場合

シークレット名がホストセクションとともに提供された場合、NetScaler Ingress Controller はそのシークレットをSNI証明書としてバインドします。

```
spec:

  tls:

  - secretName: fruitjuice.secret

    hosts:

    - items.fruit.juice
```

ホストセクションを使用しない場合

シークレット名がホストセクションなしで提供された場合、NetScaler Ingress Controller はそのシークレットをデフォルト証明書としてバインドします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:

  tls:

  - secretName: colddrink.secret
```

注:

複数のシークレットが指定された場合、NetScaler Ingress Controller はすべての証明書をSNI対応証明書としてバインドします。

注意点

1. 複数のシークレットがNetScaler Ingress Controllerに提供される場合、以下の優先順位が適用されます。
   a. preconfigured-default-certkey または非ホスト TLS シークレット
   b. デフォルトSSL証明書
2. 同一グレードの証明書間で優先順位の競合がある場合（例えば、2つのIngressファイルがそれぞれ非ホストTLSシークレットをデフォルト/非SNIタイプとして構成している場合）、NetScaler Ingress Controllerは、NetScaler Ingress Controllerのデフォルト証明書を非SNI証明書としてバインドし、他のすべての証明書をSNIで使用します。
3. TLSセクションで指定されたシークレットに使用される証明書にはCN名が必要です。そうでない場合、NetScalerにバインドされません。
4. SSL仮想サーバーでSNIが有効になっている場合：
   - 非SNI（デフォルト）証明書は、以下のHTTPSリクエストに使用されます。

     ```
     curl -1 -v -k https://1.1.1.1/



     curl -1 -v -k -H 'HOST:*.colddrink.beverages' https://1.1.1.1/
     ```

   - SNIが有効な証明書は、完全なドメイン名を持つリクエストに使用されます。

     ```
     curl -1 -v -k https://items.colddrink.beverages/
     ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

証明書と一致しないリクエストが受信された場合、CN名が一致しないため、失敗します。

NetScalerにおけるTLSクライアント認証のサポート

TLSクライアント認証では、サーバーは認証のためにクライアントから有効な証明書を要求し、承認されたマシンとユーザーのみがアクセスできるようにします。

NetScalerのSSLベースの仮想サーバーを使用して、TLSクライアント認証を有効にできます。NetScaler SSL仮想サーバーでクライアント認証が有効になっている場合、NetScalerはSSLハンドシェイク中にクライアント証明書を要求します。アプライアンスは、発行者の署名や有効期限などの通常の制約について、クライアントから提示された証明書をチェックします。

次の図は、NetScalerにおけるTLSクライアント認証機能について説明しています。

TLSクライアント認証(/en-us/netscaler-k8s-ingress-controller/media/ssl-client-authentication.png)

TLSクライアント認証は、必須またはオプションに設定できます。 SSLクライアント認証が必須に設定されており、SSLクライアントが有効なクライアント証明書を提供しない場合、接続は切断されます。有効なクライアント証明書とは、特定の認証局によって署名または発行されており、期限切れまたは失効していないことを意味します。 オプションとしてマークされている場合、NetScalerはクライアント証明書を要求しますが、接続は切断されません。クライアントが証明書を提示しない場合や証明書が誤っている場合でも、NetScalerはSSLトランザクションを続行します。オプション設定は、二要素認証のような認証シナリオで役立ちます。

TLSクライアント認証の構成

TLSクライアント認証を構成するには、次の手順を実行します。

1. NetScalerでTLSサポートを有効にします。

   NetScaler Ingress Controllerは、Ingress定義の**TLS**セクションをNetScalerでのTLSサポートの有効化に使用します。 以下は、Ingress定義のサンプルスニペットです。

   ```
   spec:

     tls:

      - secretName:
   ```

2. CA証明書をKubernetes環境に適用します。

   既存の証明書に対してKubernetesシークレットを生成するには、次のkubectlコマンドを使用します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl create secret generic tls-ca --from-file=tls.crt=cacerts.pem
```

注:

シークレットを作成する際に「tls.crt=」を指定する必要があります。このファイルは、NetScaler Ingress ControllerがCAシークレットを解析する際に使用されます。

3. クライアント認証を有効にするようにIngressを構成します。

   Kubernetesにデプロイされたサービスに対するクライアント証明書認証に使用される、生成されたCAシークレットをアタッチするには、以下の注釈を指定する必要があります。

```
ingress.citrix.com/ca-secret: '{"frontend-hotdrinks": "hotdrink-ca-secret"}'
```

   デフォルトでは、クライアント証明書認証は`mandatory`に設定されていますが、フロントエンド構成で`frontend_sslprofile`注釈を使用して`optional`に設定できます。

```
ingress.citrix.com/frontend_sslprofile: '{"clientauth":"ENABLED", “clientcert”: “optional”}'
```

注:

`frontend_sslprofile`はフロントエンドのIngress構成のみをサポートします。詳細については、フロントエンド構成を参照してください。

NetScaler Ingress Controller を使用した NetScaler での TLS サーバー認証サポート

サーバー認証を使用すると、クライアントはアクセスしているWebサーバーの信頼性を検証できます。 通常、NetScalerデバイスはWebサーバーの代わりにSSLオフロードとアクセラレーションを実行し、Webサーバーの証明書を認証しません。ただし、エンドツーエンドのSSL暗号化を必要とする展開では、サーバーを認証できます。

このような状況では、NetScaler デバイスは SSL クライアントとなり、次の処理を実行します。

- SSL サーバーとの安全なトランザクションを実行します

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- 証明書が SSL サービスにバインドされている CA がサーバー証明書に署名していることを確認します
- サーバー証明書の有効性を確認します。

サーバーを認証するには、まずサーバー認証を有効にし、次にサーバーの証明書に署名した CA の証明書を NetScaler アプライアンスの SSL サービスにバインドする必要があります。証明書をバインドするときは、バインドを CA オプションとして指定する必要があります。

TLS サーバー認証の構成

TLS サーバー認証を構成するには、次の手順を実行します。

1. NetScaler で TLS サポートを有効にします。

   NetScaler Ingress Controllerは、NetScalerでのTLSサポートを有効にするために、Ingress定義の**TLS**セクションを使用します。 以下は、Ingress定義のサンプルスニペットです。

   ```
   spec:

     tls:

      - secretName:
   ```

2. 既存の証明書に対して Kubernetes シークレットを生成するには、次の手順を実行します。

   a. サービスで使用するクライアント証明書を生成します。

   ```
   kubectl create secret tls tea-beverage --cert=path/to/tls.cert --
   key=path/to/tls.key --namespace=default
   ```

   b. 既存の CA 証明書のシークレットを生成します。この証明書は、バックエンドサーバー証明書に署名するために必要です。

   ```
   kubectl create secret generic tea-ca --from-file=tls.crt=cacerts.pem
   ```

   注:

   シークレットを作成する際に`tls.crt=`を指定する必要があります。このファイルは、NetScaler Ingress Controller が CA シークレットを解析する際に使用されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

3. Ingress 設定で次のアノテーションを使用して、サービスへの安全なバックエンド通信を有効にします。

```
ingress.citrix.com/secure-backend: "True"
```

4. 次のアノテーションを使用して、証明書を SSL サービスにバインドします。この証明書は、NetScaler がクライアントとしてバックエンドサーバーにリクエストを送信する際に使用されます。

```
ingress.citrix.com/backend-secret: '{"tea-beverage": "tea-beverage",
"coffee-beverage": "coffee-beverage"}'
```

5. バックエンドサーバー証明書を認証するサーバー認証を有効にするには、次のアノテーションを使用できます。この設定は、サーバーの CA 証明書を NetScaler 上の SSL サービスにバインドします。

```
ingress.citrix.com/backend-ca-secret: '{"coffee-beverage":"coffee-ca",
"tea-beverage":"tea-ca"}
```

NetScaler Ingress Controller を使用して NetScaler 上で証明書をインストール、リンク、更新する

Ingress NetScaler では、証明書をインストール、リンク、更新できます。多くのサーバー証明書は、複数の階層型認証局 (CA) によって署名されています。これは、証明書がチェーンを形成することを意味します。

証明書チェーンは、SSL証明書と認証局 (CA) 証明書を含む、順序付けられた証明書のリストです。これにより、受信者は送信者とすべてのCAが信頼できることを検証できます。チェーンまたはパスはSSL証明書から始まり、チェーン内の各証明書は、チェーン内の次の証明書によって識別されるエンティティによって署名されます。

SSL証明書とルート証明書の間に位置する証明書は、チェーン証明書または中間証明書と呼ばれます。中間証明書は、SSL証明書の署名者または発行者です。ルートCA証明書は、中間証明書の署名者または発行者です。

中間証明書がサーバー (SSL証明書がインストールされている場所) にインストールされていない場合、一部のブラウザ、モバイルデバイス、およびアプリケーションがSSL証明書を信頼できなく

© 1997–2026 Citrix Systems, Inc. All rights reserved.

なる可能性があります。SSL証明書をすべてのクライアントと互換性のあるものにするには、中間証明書がインストールされている必要があります。

証明書チェーン(/en-us/netscaler-k8s-ingress-controller/media/cert-chain.jpg)

Kubernetesでの証明書のリンク

NetScaler Ingress Controller は、Kubernetes の cert-manager を使用した TLS 証明書の自動プロビジョニングと更新をサポートしています。`cert-manager` は、Let's Encrypt や HashiCorp Vault などのさまざまなソースから証明書を発行し、それらを Kubernetes シークレットに変換します。

次の図は、`cert-manager` が証明書管理をどのように実行するかを説明しています。 証明書管理(/en-us/netscaler-k8s-ingress-controller/media/cert-management.png)

複数のCA証明書が埋め込まれたPEM証明書からKubernetesシークレットを作成する場合、サーバー証明書を関連するCAにリンクする必要があります。 Kubernetesシークレットを適用する際に、Ingress NetScaler を使用して、サーバー証明書を関連するすべてのCAにリンクできます。 サーバー証明書とCAをリンクすることで、受信者は送信者とCAが信頼できるかどうかを検証できます。

以下は、Ingress 定義の例です。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: frontendssl

spec:

  rules:

  - host: frontend.com

    http:

      paths:

      - backend:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          service:
            name: frontend
            port:
              number: 443
        path: /web-frontend/frontend.php
        pathType: Prefix
  tls:
  - secretName: certchain1
```

NetScaler で、証明書が NetScaler に追加されているかどうかを確認できます。次の手順を実行します。

1. NetScaler コマンドラインインターフェイスにログオンします。
2. 次のコマンドを使用して、証明書が NetScaler に追加されているかどうかを確認します。

   ```
   >show certkey
   ```

   サンプル出力については、NetScalerのドキュメント を参照してください。
3. サーバー証明書とCAが以下のコマンドを使用してリンクされていることを確認します。

   ```
   >show certlink
   ```

   出力:

   ```
   1)  Cert Name: k8s-3KC24EQYHG6ZKEDAY5Y3SG26MT2   CA Cert Name:
   k8s-3KC24EQYHG6ZKEDAY5Y3SG2_ic1
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
2)  Cert Name: k8s-3KC24EQYHG6ZKEDAY5Y3SG2_ic1   CA Cert Name:
k8s-3KC24EQYHG6ZKEDAY5Y3SG2_ic2
```

NetScaler Ingress Controller を使用した NetScaler® の証明書キーバンドル

NetScaler Ingress Controller は、証明書バンドル (`certkeybundle`) 機能をサポートするようになりました。これは、リリース 14.1 ビルド 21.x 以降の NetScaler でサポートされています。この機能により、証明書チェーンの問題と、2つの証明書が中間CAを共有する場合に必要な追加の処理が解決されます。NetScaler での証明書キーバンドルのサポートの詳細については、「SSL 証明書キーバンドルのサポート」を参照してください。

NetScaler Ingress Controller は、以下の場所で提供される証明書に対して証明書バンドルを作成します。

- Ingress の TLS セクション
- `serviceTypeLB` の `service.citrix.com/secret` アノテーション
- Listener のシークレット

この機能を有効にするには、NetScaler Ingress Controller に新しい環境変数 `CERT_BUNDLE` が追加されます。これは、Helm Charts の `certBundle` 引数を使用して設定できます。`certBundle` の設定の詳細については、「NetScaler Ingress Controller」の Helm Charts を参照してください。NetScaler Ingress Controller は、証明書バンドルを SSL タイプのコンテンツスイッチング仮想サーバーに追加します。

NetScaler での証明書バンドルの作成は、以下のシナリオでは機能しません。

- セキュアなバックエンド (SSL タイプのサービスグループ) が構成されている。
- 自己署名証明書が存在する。
- ingress、servicetypeLB、または Listener で `preconfigured-certkey` アノテーションが設定されている。

注意事項

1. NetScaler での証明書バンドルの作成は、以下の順序が満たされていない場合、失敗します。
    - サーバー証明書 (SC) はバンドルファイルの先頭に配置する必要があります。
    - IC は中間証明書です。IC は IC によって発行されます。証明書は順序通りに配置する必要があり、すべての中間証明書がバンドル内に存在する必要があります。
    - 証明書はPEM形式である必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- サーバー証明書キー (SCK) はバンドル内のどこにでも配置できます。
- 最大9つの中間証明書がサポートされています

2. `CERT_BUNDLE` 機能でコントローラーをアップグレードすると、以前の証明書キーバインディングは削除され、新しい `certkeybundle` が作成されます。新しい `certkeybundle` は、SSLタイプのコンテキストスイッチング仮想サーバーにバインドされます。

NetScaler Ingress ControllerでのSSL CA証明書バンドルのサポート

リリース14.1ビルド29.x以降、NetScalerは、CA証明書バンドルをSSL仮想サーバー、サービス、およびサービスグループにバインドすることをサポートしています。このサポートにより、各CA証明書に対して個別のファイルを作成することなく、CA証明書バンドルをバインドできます。NetScaler Ingress Controllerは、Ingressまたは `serviceTypeLB` に基づく仮想サーバーおよびサービスグループへのSSL CA証明書バンドルのバインディングをサポートしています。

顧客はCA証明書バンドル用のKubernetesシークレットを作成し、アノテーションを使用して、IngressおよびserviceTypeLBでKubernetesシークレットを提供できます。

```
Kubectl create secret generic tls-ca --from-file=tls.crt=rootCA.crt
```

上記の例では、`rootCA.crt` にCA証明書チェーンが含まれています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

- 次の順序が満たされない場合、バンドルの作成は失敗します。
  - `IC[1-9]` は中間証明書です。`IC[i]` は `IC[i+1]` によって発行されます。証明書は順序通りに配置する必要があり、すべての中間証明書がバンドル内に存在し、ルート証明書が最後にある必要があります。 次の例では、証明書セット (bundle.pem) に以下のファイルが含まれています。
    - 1番目の中間証明書 (bundle_ic1)
    - 2番目の中間証明書 (bundle_ic2)
    - ルート証明書
  - 最大9つの中間証明書がサポートされています。
  - 証明書はPEM形式である必要があります。
- CA証明書キーペアが仮想サーバー、サービス、またはサービスグループにバインドされている場合、CA証明書バンドルをバインドすることはできません。その逆も同様です。
- 暗号化された秘密鍵はサポートされていません。
- DTLSバインドとクラスターはサポートされていません。
- OCSPとCRLはサポートされていません。

提供されたKubernetesシークレットをNetScalerでCA証明書バンドルとしてバインドするには、アノテーションで`cacertbundle`引数を`true`として設定します。

イングレスの場合

```
ingress.citrix.com/ca-secret: '{"frontend-hotdrinks": {"ca-name": "abcca",
"cacertbundle": "true"}, "coffee-beverage": {"ca-name": "abc1ca",
"cacertbundle": "true"}}'
```

```
ingress.citrix.com/backend-ca-secret: '{"frontend-hotdrinks": {"ca-name":
"abcca", "cacertbundle": "true"}, "coffee-beverage": {"ca-name": "abc1ca",
"cacertbundle": "true"}}'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

上記の例では、`frontend-hotdrinks`と`coffee-beverage`はイングレス内のバックエンドサービスの名前であり、`abcca`と`abc1ca`はCA証明書チェーン用に作成されたKubernetesシークレットです。

ServiceTypeLBの場合

```
service.citrix.com/ca-secret: '{"https-443":{"ca-name" : "abc1ca",
"cacertbundle": "true"}}'
```

```
service.citrix.com/backend-ca-secret: '{"https-443":{"ca-name" : "abc1ca",
"cacertbundle": "true"}}'
```

上記の例では、`https-443`はサービスの`port-protocol`の組み合わせであり、`abc1ca`はCA証明書チェーン用に作成された2つのKubernetesシークレットです。

NetScalerの事前設定済みCA証明書バンドル

イングレスの場合

```
ingress.citrix.com/preconfigured-certkey: '{"certs": [ {"name": "ca3bun",
"type": "ca"}]}'
```

```
ingress.citrix.com/preconfigured-backend-ca-certkey: '{"frontend-
hotdrinks": "ca3bun", "coffee-beverage": "ca3bun"}'
```

ServiceTypeLBの場合

```
service.citrix.com/preconfigured-backend-ca-certkey: '{"443-tcp":
"ca3bun", "https-6443" :"ca3bun"}'
```

```
service.citrix.com/preconfigured-ca-certkey: '{"443-tcp": ["ca3bun",
"ca4bun"], "https-6443": "ca3bun"}'
```

上記の例では、`ca3bun` と `ca4bun` は、NetScaler で作成された CA 証明書バンドルです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Ingress Controller を介して NetScaler で作成された事前設定済み証明書バンドルのサポート

NetScaler Ingress Controller リリース 3.1.34 以降、証明書チェーン用に作成された Kubernetes Secret の証明書バンドル構成がサポートされています。お客様は、NetScaler で証明書チェーン用の certkey バンドルを作成できるようになり、それを Ingress または `serviceTypeLB` で提供できます。

Ingress の場合

```
ingress.citrix.com/preconfigured-certkey: '{"certs": [{"name": "abc4cert",
"type": "default"}]}'
```

```
ingress.citrix.com/preconfigured-backend-certkey: '{"frontend-hotdrinks":
"abc4cert", "coffee-beverage": "abc4cert"}'
```

上記の例では、`abc4cert` は、証明書チェーン用に NetScaler で作成された証明書バンドルであり、`frontend-hotdrinks` と `coffee-beverage` は、Ingress で指定されたバックエンドサービス名です。

serviceTypeLB の場合

```
service.citrix.com/preconfigured-backend-certkey: '{"443-
tcp":"abc3cert","https-6443": "abc4cert"}'
```

```
service.citrix.com/preconfigured-certkey: '{"443-tcp": " abc3cert ",
"https-6443": "abc4cert "}'
```

上記の例では、`443-tcp` と `https-6443` は、`serviceType LB` で提供されるポートとプロトコルの組み合わせであり、`abc3cert` と `abc4cert` は、certkey バンドルを使用して NetScaler で作成された2つの証明書チェーンです。

参照

- CA 証明書バンドルのサポート
- NetScaler Ingress Controller を使用した NetScaler® の証明書キーバンドル

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Kubernetes Ingress を使用した SSL パススルーの構成

SSLパススルー機能を使用すると、ロードバランサーを使用して要求を復号化するのではなく、受信するセキュリティソケットレイヤー（SSL）要求を復号化のためにサーバーに直接渡すことができます。SSLパススルーはWebアプリケーションのセキュリティに広く使用されており、TCPモードを使用して暗号化されたデータをサーバーに渡します。

プロキシSSLパススルー構成では、ロードバランサーにSSL証明書をインストールする必要はありません。SSL接続はロードバランサーではなくバックエンドサーバーが処理するため、SSL証明書はバックエンドサーバーにインストールされます。

次の図は、SSLパススルー機能について説明しています。

SSLパススルー(/en-us/netscaler-k8s-ingress-controller/media/ssl-passthrough.png)

この図に示すように、SSLトラフィックはNetScalerで終端されず、SSLトラフィックはNetScalerを介してバックエンドサーバーに渡されます。バックエンドサーバーのSSL証明書は、SSLハンドシェイクに使用されます。

NetScaler Ingress Controllerは、Ingress NetScalerでSSLパススルーを有効にするために使用できる次のIngressアノテーションを提供します。

```
ingress.citrix.com/ssl-passthrough: 'True|False'
```

アノテーションのデフォルト値は`False`です。

SSLパススルーは、Ingress定義で提供されるすべてのサービスまたはホスト名に対して有効になります。SSLパススルーはホスト名（ワイルドカードホスト名もサポートされています）を使用し、Ingressで指定されたパスは無視します。

注:

NetScaler Ingress Controllerは、ホスト名ベースではないIngressのSSLパススルーをサポートしていません。また、SSLパススルーはデフォルトのバックエンドIngressには無効です。

Ingress NetScalerでSSLパススルーを構成するには、次のサンプルIngress定義に示すように`ingress.citrix.com/ssl-passthrough:`を定義する必要があります。また、例に示すように、ホストのTLSを有効にする必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
    ingress.citrix.com/frontend-ip: x.x.x.x
    ingress.citrix.com/insecure-termination: redirect
    ingress.citrix.com/secure-backend: "True"
    ingress.citrix.com/ssl-passthrough: "True"
  name: hotdrinks-ingress
spec:
  ingressClassName: citrix
  rules:
  - host: hotdrinks.beverages.com
    http:
      paths:
      - backend:
          service:
            name: frontend-hotdrinks
            port:
              number: 443
        path: /
        pathType: Prefix
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
tls:

- secretName: beverages
```

cert-manager を使用した自動証明書管理

NetScaler Ingress Controller は、cert-manager を使用したTLS証明書の自動プロビジョニングと更新をサポートしています。`cert-manager` は、Kubernetesネイティブの証明書管理コントローラーです。Let's Encrypt や HashiCorp Vault など、さまざまなソースから証明書を発行します。

次の図に示すように、`cert-manager` は外部の認証局 (CA) と連携して証明書に署名し、それをKubernetesシークレットに変換します。これらのシークレットは、NetScaler Ingress ControllerによってNetScaler上のSSL仮想サーバーを構成するために使用されます。

詳細な構成については、以下を参照してください。

- NetScaler Ingress Controller と Let's Encrypt を使用して cert-manager で Kubernetes 上に HTTPS Web アプリケーションを展開する
- NetScaler Ingress Controller と HashiCorp Vault を使用して cert-manager で Kubernetes 上に HTTPS Web アプリケーションを展開する

NetScaler Ingress Controllerとcert-managerを使用して、Kubernetes上にHTTPSウェブアプリケーションをLet`s Encryptでデプロイする

Let's Encrypt とACME (Automatic Certificate Management Environment) プロトコルを使用すると、HTTPSサーバーをセットアップし、ブラウザが信頼する証明書を自動的に取得できます。Let's Encryptからウェブサイトのドメインの証明書を取得するには、特定のチャレンジを達成してドメインの制御を証明する必要があります。チャレンジとは、ドメインを制御する者だけが達成できる、指定されたタスクのリストの1つです。

現在、チャレンジには2つのタイプがあります。

- HTTP-01チャレンジ：HTTP-01チャレンジは、ウェブサイト上の指定された場所に指定されたファイルを投稿することで完了します。Let's Encrypt CAは、HTTP URIに対してHTTPリクエストを行うことでファイルを検証し、チャレンジを達成します。
- DNS-01チャレンジ：DNS-01チャレンジは、DNS TXTレコードに存在する計算されたキーを提供することで完了します。このTXTレコードがインターネット全体に伝播されると、ACMEサーバーはDNSルックアップを介してこのキーを正常に取得できます。ACMEサーバーは、クライアントが要求された証明書のドメインを所有していることを検証できます。適切な権限が

© 1997–2026 Citrix Systems, Inc. All rights reserved.

あれば、cert-managerは指定されたDNSプロバイダーに対してこのTXTレコードを自動的に提示します。

チャレンジの検証が成功すると、そのドメインの証明書が付与されます。

このトピックでは、以下を使用してKubernetesクラスターにHTTPSウェブアプリケーションを安全にデプロイする方法に関する情報を提供します。

- ネットスケーラー イングレス コントローラー
- JetStackのcert-managerを使用して、Let's Encrypt projectからTLS証明書をプロビジョニングします。

前提条件

以下を確認してください。

- 証明書が要求されるドメインが公開アクセス可能であること。
- KubernetesクラスターでRBACが有効になっていること。
- Tier 1またはTier 2デプロイメントモデルでNetScaler MPX、VPX、またはCPXがデプロイされていること。

  Tier 1デプロイメントモデルでは、NetScaler MPXまたはVPXがアプリケーションデリバリーコントローラー (ADC) として使用されます。Kubernetesクラスターで実行されているNetScaler Ingress Controllerは、Kubernetesクラスターで実行されているサービス用の仮想サービスを構成します。NetScalerは、公開ルーティング可能なIPアドレスで仮想サービスを実行し、Let's Encryptによって生成された証明書を使用してクライアントトラフィックのSSLをオフロードします。

  Tier 2デプロイモデルでは、Kubernetesクラスターの外部で実行されているNetScaler（VPX/MPX）上にTCPサービスが構成されます。このサービスは、Kubernetesクラスター内で実行されているNetScaler CPXインスタンスにトラフィックを転送するために作成されます。NetScaler CPXはSSLセッションを終了し、実際のサービスポッドにトラフィックをロードバランスします。
- NetScaler Ingress Controllerをデプロイしました。さまざまなデプロイシナリオについては、こちらをクリックしてください。
- Let's Encrypt CAがHTTP01チャレンジのドメインを検証するために、ファイアウォールで仮想IPアドレスのポート80を開放しました。
- ACME DNS01チャレンジのためにWebアプリケーションをホストする、お客様が管理するDNSドメイン。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- すべてのデプロイ手順に対する管理者権限。権限が原因で障害が発生した場合は、管理者権限があることを確認してください。

cert-managerをインストールする

cert-managerをインストールするには、cert-managerのインストールに関するドキュメントを参照してください。

cert-managerは、マニフェストファイルまたはHelmチャートのいずれかを使用してインストールできます。

cert-managerをインストールしたら、インストールの検証で説明されているように、cert-managerが稼働していることを確認します。

サンプルWebアプリケーションをデプロイする

サンプルWebアプリケーションをデプロイするには、次の手順を実行します。

注記：

このトピックでは、KubernetesデモアプリケーションであるKuardを参考にしています。

1. 次の構成でKuardのデプロイYAMLファイル（`kuard-deployment.yaml`）を作成します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: kuard
  name: kuard
spec:
  replicas: 1
  selector:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            matchLabels:
              app: kuard
          template:
            metadata:
              labels:
                app: kuard
            spec:
              containers:
              - image: gcr.io/kuar-demo/kuard-amd64:1
                imagePullPolicy: Always
                name: kuard
                ports:
                - containerPort: 8080
                  protocol: TCP
```

2. 次のコマンドを使用して、Kuardデプロイファイル（`kuard-deployment.yaml`）をクラスターにデプロイします。

   % kubectl create -f kuard-deployment.yaml

   ```
   deployment.extensions/kuard created
   ```

   % kubectl get pod -l app=kuard

   ```
   NAME                     READY   STATUS    RESTARTS   AGE
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kuard-6fc4d89bfb-djljt   1/1     Running   0          24s
```

3. デプロイメントのサービスを作成します。次の構成で`service.yaml`というファイルを作成します。

```
apiVersion: v1

kind: Service

metadata:

  name: kuard

spec:

  ports:

  - port: 80

    targetPort: 8080

    protocol: TCP

  selector:

    app: kuard
```

4. 次のコマンドを使用してサービスをデプロイし、検証します。

```
% kubectl create -f service.yaml



service/kuard created

% kubectl get svc kuard

NAME    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE

kuard   ClusterIP   10.103.49.171   &lt;none>        80/TCP    13s
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

5. NetScaler CPXまたはVPXにコンテンツスイッチング仮想サーバーとしてデプロイされるIngressを作成して、このサービスを外部に公開します。

注記：

NetScaler Ingress Controllerが起動しているIngressクラスに、`ingressClassName: citrix`の値を変更してください。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
  name: kuard
spec:
  ingressClassName: citrix®
  rules:
  - host: kuard.example.com
    http:
      paths:
      - backend:
          service:
            name: kuard
            port:
              number: 80
        path: /
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        pathType: Prefix
```

注記：

`spec.rules.host`の値を、お客様が管理するドメインに変更する必要があります。NetScaler CPXまたはVPXにトラフィックをルーティングするためのDNSエントリが存在することを確認してください。

1. 次のコマンドを使用してIngressをデプロイします。

```
% kubectl apply -f ingress.yml

ingress.extensions/kuard created



root@ubuntu-:~/cert-manager# kubectl get ingress

NAME    HOSTS               ADDRESS   PORTS   AGE

kuard   kuard.example.com             80      7s
```

2. 次のコマンドを使用して、IngressがNetScaler CPXまたはVPXで構成されていることを確認します。

```
$ kubectl exec -it cpx-ingress-5b85d7c69d-ngd72 /bin/bash



root@cpx-ingress-55c88788fd-qd4rg:/# cli_script.sh 'show cs vserver'

exec: show cs vserver

1)  k8s-192.168.8.178_80_http (192.168.8.178:80) - HTTP Type: CONTENT

  State: UP

  Last state change was at Sat Jan  4 13:36:14 2020
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  Time since last state change: 0 days, 00:18:01.950

  Client Idle Timeout: 180 sec

  Down state flush: ENABLED

  Disable Primary Vserver On Down : DISABLED

  Comment: uid=MPPL57E3AFY6NMNDGDKN2VT57HEZVOV53Z7DWKH44X2SGLIH4ZWQ====

  Appflow® logging: ENABLED

  Port Rewrite : DISABLED

  State Update: DISABLED

  Default:  Content Precedence: RULE

  Vserver IP and Port insertion: OFF

  L2Conn: OFF Case Sensitivity: ON

  Authentication: OFF

  401 Based Authentication: OFF

  Push: DISABLED  Push VServer:

  Push Label Rule: none

  Persistence: NONE

  Listen Policy: NONE

  IcmpResponse: PASSIVE

  RHIstate:  PASSIVE

  Traffic Domain: 0

Done
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
root@cpx-ingress-55c88788fd-qd4rg/# exit

exit
```

3. `curl`コマンドを使用して要求されたときに、ウェブページが正しく提供されていることを確認します。

```
% curl -sS -D - kuard.example.com -o /dev/null

HTTP/1.1 200 OK

Content-Length: 1458

Content-Type: text/html

Date: Thu, 21 Feb 2019 09:09:05 GMT
```

HTTPチャレンジを使用したACME証明書の発行を構成する

このセクションでは、HTTP検証を使用してACME証明書を発行する方法について説明します。DNS検証を使用する場合は、このセクションをスキップして次のセクションに進んでください。

cert-managerを使用したHTTP検証は、ドメインのLet's Encryptから証明書を取得する簡単な方法です。この方法では、特定のファイルがドメインに存在することを確認することで、ドメインの所有権を証明します。指定されたパスの下に指定されたファイルを公開できる場合、そのドメインを制御していると見なされます。

HTTP01 チャレンジプロバイダーを使用して Let's Encrypt ClusterIssuer をデプロイする

cert-manager は、構成のために2つの異なるCRDをサポートしています。単一の名前空間にスコープされた `Issuer` と、クラスター全体にスコープされた `ClusterIssuer` です。

NetScaler Ingress Controller が任意の名前空間から Ingress を使用するには、`ClusterIssuer` を使用します。あるいは、Ingress リソースを作成する各名前空間に対して `Issuer` を作成することもできます。

詳細については、cert-manager の HTTP validation ドキュメントを参照してください。

1. `issuer-letsencrypt-staging.yaml` という名前のファイルを作成し、次の構成を記述します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: cert-manager.io/v1alpha2

kind: ClusterIssuer

metadata:

  name: letsencrypt-staging

spec:

  acme:

    # You must replace this email address with your own.

    # Let's Encrypt will use this to contact you about expiring

    # certificates, and issues related to your account.

    email: user@example.com

    server: https://acme-staging-v02.api.letsencrypt.org/directory

    privateKeySecretRef:

      # Secret resource used to store the account's private key.

      name: example-issuer-account-key

    # Add a single challenge solver, HTTP01 using citrix

    solvers:

    - http01:

        ingress:

          class: citrix
```

`spec.acme.solvers[].http01.ingress.class` は NetScaler Ingress Controller の Ingress クラスを指します。NetScaler Ingress Controller に Ingress クラスがない場合、このフィールドを指定する必要はありません。 **注:** これは cert-manager.io/v1alpha2 リソースのサンプル

© 1997–2026 Citrix Systems, Inc. All rights reserved.

`Clusterissuer` です。詳細については、cert-manager http01 ドキュメント を参照してください。

ステージングの Let's Encrypt サーバーは偽の証明書を発行しますが、本番サーバーの API レート制限 には拘束されません。このアプローチにより、レート制限を気にすることなく環境をセットアップおよびテストできます。Let's Encrypt の本番サーバーに対しても同じ手順を繰り返すことができます。

2. ファイルを編集して保存したら、次のコマンドを使用してファイルをデプロイします。

```
% kubectl apply -f issuer-letsencrypt-staging.yaml

clusterissuer "letsencrypt-staging" created
```

3. 発行者が作成され、ACME サーバーに登録されていることを確認します。

```
% kubectl get issuer

NAME                  AGE

letsencrypt-staging   8d
```

4. コマンド `kubectl describe issuer letsencrypt-staging` を使用して、`ClusterIssuer` が適切に登録されていることを確認します。

```
% kubectl describe issuer letsencrypt-staging



Status:

  Acme:

    Uri:  https://acme-staging-v02.api.letsencrypt.org/acme/acct/
8200869

  Conditions:

    Last Transition Time:  2019-02-11T12:06:31Z

    Message:               The ACME account was registered with the
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
ACME server

    Reason:                ACMEAccountRegistered

    Status:                True

    Type:                  Ready
```

Ingress オブジェクトの証明書を発行する

`ClusterIssuer` が正常に登録されたら、Ingress ドメイン 'kuard.example.com' の証明書を取得できます。

次の方法を使用して、指定された Ingress リソースの証明書を要求できます。

- Ingress オブジェクトに `Ingress-shim` アノテーションを追加する。
- `certificate` CRD オブジェクトを作成する。

最初の方法は迅速かつ簡単ですが、証明書更新に関してより多くのカスタマイズと粒度が必要な場合は、2番目の方法を選択できます。ニーズに応じて方法を選択できます。

INGRESSオブジェクトに`INGRESS-SHIM`アノテーションを追加する

このアプローチでは、ACMEサーバーから証明書を要求するIngressオブジェクトに、以下の2つのアノテーションを追加します。

```
certmanager.io/cluster-issuer: "letsencrypt-staging"
```

注:

`Ingress-shim`用のcert-managerがサポートするすべてのアノテーションは、supported-annotationsで確認できます。

また、シークレットを指定して、`ingress.yaml`がTLSを使用するように変更します。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  annotations:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    certmanager.io/cluster-issuer: letsencrypt-staging
  name: kuard
spec:
  ingressClassName: citrix
  rules:
  - host: kuard.example.com
    http:
      paths:
      - backend:
          service:
            name: kuard
            port:
              number: 80
        pathType: Prefix
        path: /
  tls:
  - hosts:
    - kuard.example.com
    secretName: kuard-example-tls
```

`cert-manager.io/cluster-issuer: "letsencrypt-staging"`アノテーションは、cert-managerに`letsencrypt-staging`クラスター全体のイシューアーを使用して、Let's Encryptのステージングサーバーから証明書を要求するように指示します。Cert-managerは、`kuard.example.com`の証明書のライフサイクルを管理するために使用される`certificate`オ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ブジェクトを作成します。証明書オブジェクトのドメイン名とチャレンジメソッドの値は、Ingressオブジェクトから派生します。Ingressがクラスターに存在する限り、Cert-managerはシークレットの内容を管理します。

次のコマンドを使用して、`ingress.yaml`ファイルをデプロイします。

```
% kubectl apply -f ingress.yml



ingress.extensions/kuard configured

% kubectl get ingress kuard

NAME    HOSTS               ADDRESS   PORTS     AGE

kuard   kuard.example.com             80, 443   4h39m
```

証明書CRDリソースを作成する

あるいは、Ingressオブジェクトとは独立して証明書CRDオブジェクトをデプロイすることもできます。「certificate」CRDのドキュメントは、HTTP validationで確認できます。

1. 次の構成で`certificate.yaml`ファイルを作成します。

```
apiVersion: cert-manager.io/v1alpha2

kind: Certificate

metadata:

  name: example-com

  namespace: default

spec:

  secretName: kuard-example-tls

  issuerRef:

    name: letsencrypt-staging
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
commonName: kuard.example.com

dnsNames:

- www.kuard.example.com
```

`spec.secretName`キーは、証明書が正常に発行されたときに証明書が保存されるシークレットの名前です。

1. Kubernetesクラスターに`certificate.yaml`ファイルをデプロイします。

   ```
   kubectl create -f certificate.yaml

   certificate.cert-manager.io/example-com created
   ```

2. Ingressで指定された証明書を表す、cert-managerによって証明書のカスタムリソースが作成されていることを確認します。数分後、ACME検証が正常に完了すると、証明書の「READY」ステータスがtrueに設定されます。

   ```
   % kubectl get certificates.cert-manager.io kuard-example-tls

   NAME                READY   SECRET              AGE

   kuard-example-tls   True    kuard-example-tls   3m44s




   % kubectl get certificates.cert-manager.io kuard-example-tls

   Name:         kuard-example-tls

   Namespace:    default

   Labels:       &lt;none>

   Annotations:  &lt;none>

   API Version:  cert-manager.io/v1alpha2
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
Kind:         Certificate
Metadata:
  Creation Timestamp:  2020-01-04T17:36:26Z
  Generation:          1
  Owner References:
    API Version:           extensions/v1beta1
    Block Owner Deletion:  true
    Controller:            true
    Kind:                  Ingress
    Name:                  kuard
    UID:                   2cafa1b4-2ef7-11ea-8ba9-06bea3f4b04a
  Resource Version:        81263
  Self Link:               /apis/cert-manager.io/v1alpha2/namespaces/
default/certificates/kuard-example-tls
  UID:                     bbfa5e51-2f18-11ea-8ba9-06bea3f4b04a
Spec:
  Dns Names:
    acme.cloudpst.net
  Issuer Ref:
    Group:      cert-manager.io
    Kind:       ClusterIssuer
    Name:       letsencrypt-staging
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  Secret Name:  kuard-example-tls

Status:

  Conditions:

    Last Transition Time:  2020-01-04T17:36:28Z

    Message:               Certificate is up to date and has not
expired

    Reason:                Ready

    Status:                True

    Type:                  Ready

  Not After:               2020-04-03T16:36:27Z

Events:

  Type    Reason        Age   From          Message

  ----    ------        ----  ----          -------

  Normal  GeneratedKey  24m   cert-manager  Generated a new private key

  Normal  Requested     24m   cert-manager  Created new
CertificateRequest resource "kuard-example-tls-3030465986"

  Normal  Issued        24m   cert-manager  Certificate issued
successfully
```

3. シークレットリソースが作成されていることを確認します。

```
% kubectl get secret  kuard-example-tls

  NAME               TYPE               DATA   AGE

  kuard-example-tls   kubernetes.io/tls   3      3m13s
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

DNSチャレンジを使用してACME証明書を発行する

このセクションでは、DNS検証を使用してLet'sEncrypt CAからACME証明書を取得する方法について説明します。DNS-01チャレンジでは、ドメインのDNSレコードを制御していることを証明することで、ドメインの所有権を証明します。これは、ドメインのDNSレコードを制御していることを証明する特定のコンテンツを含むTXTレコードを作成することによって行われます。DNSチャレンジの詳細な説明と、DNSチャレンジをデプロイする際のベストセキュリティプラクティスについては、A Technical Deep Dive: Securing the Automation of ACME DNS Challenge Validationを参照してください。

注記:

この手順では、`route53`がDNSプロバイダーとして使用されます。他のプロバイダーについては、cert-managerのDNS検証のドキュメントを参照してください。

DNS01チャレンジプロバイダーを使用してLet's Encrypt ClusterIssuerをデプロイする

DNS01チャレンジプロバイダーを使用してLet's Encrypt `ClusterIssuer`をデプロイするには、以下を実行します。

1. AWS IAMユーザーアカウントを作成し、シークレットアクセスキーIDとシークレットアクセスキーをダウンロードします。
2. 以下のIAMポリシーをユーザーに付与します。

   Route53アクセスポリシー

3. `kube-system`名前空間にKubernetesシークレット`acme-route53`を作成します。

   ```
   % kubectl create secret generic acme-route53 --from-literal secret-access-key=&lt;secret_access_key>
   ```

4. DNS01チャレンジプロバイダーを使用して`Issuer`または`ClusterIssuer`を作成します。

   DNS01の下に複数のプロバイダーを指定し、証明書作成時に使用するプロバイダーを指定できます。 cert-managerがTXTレコードを作成するには、DNSプロバイダーへのアクセス権が必要です。資格情報は、`spec.dns01.secretAccessKeySecretRef`で指定されたKubernetesシークレットに保存されます。資格情報の取得方法の詳細については、DNSプロバイダーのドキュメントを参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: cert-manager.io/v1alpha2

kind: ClusterIssuer

metadata:

  name: letsencrypt-staging

  spec:

    acme:

    # You must replace this email address with your own.

    # Let's Encrypt will use this to contact you about expiring

    # certificates, and issues related to your account.

      email: user@example.com

      server: https://acme-staging-v02.api.letsencrypt.org/directory

      privateKeySecretRef:

        name: example-issuer-account-key

      solvers:

      - dns01:

          route53:

            region: us-west-2

            accessKeyID: <IAMKEY>

            secretAccessKeySecretRef:

              name: acme-route53

              key: secret-access-key
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注記:

`user@example.com` をあなたのメールアドレスに置き換えてください。DNS01スタンザで言及されている各ドメインについて、cert-managerは参照されたIssuerからのプロバイダーの資格情報を使用して、`_acme-challenge` というTXTレコードを作成します。このレコードはACMEサーバーによって検証され、証明書が発行されます。DNSプロバイダーの設定とサポートされているプロバイダーのリストの詳細については、DNS01リファレンスドキュメントを参照してください。

5. ファイルを編集して保存したら、次のコマンドを使用してファイルをデプロイします。

```
% kubectl apply -f acme_clusterissuer_dns.yaml

clusterissuer "letsencrypt-staging" created
```

6. 以下のコマンドを使用して、発行者が作成され、ACMEサーバーに登録されているかを確認します。

```
% kubectl get issuer

NAME                 AGE

letsencrypt-staging   8d
```

7. コマンド `kubectl describe issuer letsencrypt-staging` を使用して、`ClusterIssuer` が適切に登録されているかを確認します。

```
Status:

  Acme:

    Uri:  https://acme-staging-v02.api.letsencrypt.org/acme/acct/
8200869

  Conditions:

    Last Transition Time:  2019-02-11T12:06:31Z

    Message:               The ACME account was registered with the
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
 ACME server

     Reason:                ACMEAccountRegistered

     Status:                True

     Type:                  Ready
```

Ingressリソースの証明書を発行する

発行者が正常に登録されると、Ingressドメイン `kuard.example.com` の証明書を取得できます。HTTP01チャレンジと同様に、指定されたIngressリソースの証明書を要求する方法は2つあります。

- Ingressオブジェクトに `Ingress-shim` アノテーションを追加する。
- `certificate` CRDオブジェクトを作成する。詳細な手順については、証明書CRDリソースを作成する を参照してください。

INGRESSオブジェクトに `INGRESS-SHIM` アノテーションを追加する

以下の `spec.tls` セクションとともに、Ingressオブジェクトにアノテーションを追加します。

```
certmanager.io/cluster-issuer: "letsencrypt-staging"
```

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  annotations:

    cert-manager.io/cluster-issuer: letsencrypt-staging

  name: kuard

spec:

  ingressClassName: citrix

  rules:

  - host: kuard.example.com
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    http:
      paths:
      - backend:
          service:
            name: kuard
            port:
              number: 80
        pathType: Prefix
        path: /
  tls:
  - hosts:
    - kuard.example.com
    secretName: kuard-example-tls
```

cert-managerは、DNS01チャレンジで `Certificate` CRDリソースを作成します。これは、`ClusterIssuer` で指定された資格情報を使用して、所有するドメインのDNSサーバーにTXTレコードを作成します。その後、Let's Encrypt CAはTXTレコードの内容を検証してチャレンジを完了します。

`CERTIFICATE` CRDリソースを追加する**

あるいは、証明書の自動生成をトリガーするために、証明書カスタムリソース定義リソースを明示的に作成することもできます。

1. 以下の構成で `certificate.yaml` ファイルを作成します。

```
apiVersion: cert-manager.io/v1alpha2
kind: Certificate
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
metadata:
  name: example-com
  namespace: default
spec:
  secretName: kuard-example-tls
  issuerRef:
    name: letsencrypt-staging
  commonName: kuard.example.com
  dnsNames:
  - www.kuard.example.com
```

ドメイン名の検証が成功すると、証明書のREADYステータスがTrueに設定されます。

2. 証明書が発行されていることを確認します。

```
% kubectl get certificate kuard-example-tls


   NAME           READY   SECRET              AGE
   -example-tls   True    kuard-example-tls   10m
```

以下のコマンドを使用して、証明書の発行状況を監視できます。

```
% kubectl describe certificates kuard-example-tls | tail -n 6
```

```
Not After:              2020-04-04T13:34:23Z
Events:
Type    Reason     Age    From          Message
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
----    ------     ----   ----          -------

Normal  Requested  11m    cert-manager  Created new CertificateRequest
resource "kuard-example-tls-3030465986"

Normal  Issued     7m21s  cert-manager  Certificate issued successfully
```

NetScaler で証明書を検証する

Letsencrypt CA はドメインを正常に検証し、ドメインの新しい証明書を発行しました。Ingress の `tls:` フィールドで指定された `secretName` を持つ `kubernetes.io/tls` シークレットが作成されます。また、cert-manager は有効期限の 30 日前に自動的に更新を開始します。

HTTP チャレンジの場合、cert-manager は一時的な Ingress リソースを作成し、Let's Encrypt CA が生成したトラフィックを cert-manager ポッドにルーティングします。ドメインの検証が成功すると、この一時的な Ingress は削除されます。

1. 次のコマンドを使用してシークレットが作成されたことを確認します。

   ```
   % kubectl get secret kuard-example-tls



   NAME                TYPE                DATA   AGE

   kuard-example-tls   kubernetes.io/tls   3      30m
   ```

   NetScaler Ingress Controller はシークレットを取得し、証明書を NetScaler CPX のコンテンツスイッチング仮想サーバーにバインドします。中間 CA 証明書がある場合、それは自動的にサーバー証明書にリンクされ、SSL ネゴシエーション中にクライアントに提示されます。

2. NetScaler CPX にログオンし、証明書が SSL 仮想サーバーにバインドされていることを確認します。

   ```
   % kubectl exec -it cpx-ingress-55c88788fd-n2x9r bash -c cpx-ingress

   Defaulting container name to cpx-ingress.

   Use 'kubectl describe pod/cpx-ingress-55c88788fd-n2x9r -n default' to
   see all of the containers in this pod.
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
% cli_script.sh 'sh ssl vs k8s-192.168.8.178_443_ssl'

exec: sh ssl vs k8s-192.168.8.178_443_ssl



  Advanced SSL configuration for VServer k8s-192.168.8.178_443_ssl:

  DH: DISABLED

  DH Private-Key Exponent Size Limit: DISABLED Ephemeral RSA: ENABLED
 Refresh Count: 0

  Session Reuse: ENABLED  Timeout: 120 seconds

  Cipher Redirect: DISABLED

  ClearText Port: 0

  Client Auth: DISABLED

  SSL Redirect: DISABLED

  Non FIPS Ciphers: DISABLED

  SNI: ENABLED

  OCSP Stapling: DISABLED

  HSTS: DISABLED

  HSTS IncludeSubDomains: NO

  HSTS Max-Age: 0

  HSTS Preload: NO

  SSLv3: ENABLED  TLSv1.0: ENABLED  TLSv1.1: ENABLED  TLSv1.2: ENABLED
 TLSv1.3: DISABLED

  Push Encryption Trigger: Always
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  Send Close-Notify: YES

  Strict Sig-Digest Check: DISABLED

  Zero RTT Early Data: DISABLED

  DHE Key Exchange With PSK: NO

  Tickets Per Authentication Context: 1

, P_256, P_384, P_224, P_5216) CertKey Name: k8s-
GVWNYGVZKKRHKF7MZVTLOAEZYBS Server Certificate for SNI



7) Cipher Name: DEFAULT

  Description: Default cipher list with encryption strength >= 128bit

Done



% cli_script.sh 'sh certkey'

1) Name: k8s-GVWNYGVZKKRHKF7MZVTLOAEZYBS

  Cert Path: k8s-GVWNYGVZKKRHKF7MZVTLOAEZYBS.crt

  Key Path: k8s-GVWNYGVZKKRHKF7MZVTLOAEZYBS.key

  Format: PEM

  Status: Valid,   Days to expiration:89

  Certificate Expiry Monitor: ENABLED

  Expiry Notification period: 30 days

  Certificate Type: "Client Certificate" "Server Certificate"

  Version: 3
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  Serial Number: 03B2B57EA9E61A93F1D05EA3272FA95203C2

  Signature Algorithm: sha256WithRSAEncryption

  Issuer:  C=US,O=Let's Encrypt,CN=Let's Encrypt Authority X3

  Validity

    Not Before: Jan  5 13:34:23 2020 GMT

    Not After : Apr  4 13:34:23 2020 GMT

  Subject:  CN=acme.cloudpst.net

  Public Key Algorithm: rsaEncryption

  Public Key size: 2048

  Ocsp Response Status: NONE

2) Name: k8s-GVWNYGVZKKRHKF7MZVTLOAEZYBS_ic1

  Cert Path: k8s-GVWNYGVZKKRHKF7MZVTLOAEZYBS.crt_ic1

  Format: PEM

  Status: Valid,   Days to expiration:437

  Certificate Expiry Monitor: ENABLED

  Expiry Notification period: 30 days

  Certificate Type: "Intermediate CA"

  Version: 3

  Serial Number: 0A0141420000015385736A0B85ECA708

  Signature Algorithm: sha256WithRSAEncryption

  Issuer:  O=Digital Signature Trust Co.,CN=DST Root CA X3

  Validity
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    Not Before: Mar 17 16:40:46 2016 GMT

    Not After : Mar 17 16:40:46 2021 GMT

  Subject:  C=US,O=Let's Encrypt,CN=Let's Encrypt Authority X3

  Public Key Algorithm: rsaEncryption

  Public Key size: 2048

  Ocsp Response Status: NONE

Done
```

HTTPS ウェブサーバーは、偽の LE 署名付き証明書で稼働しています。次のステップは、実際の Let's Encrypt 証明書を使用して本番環境に移行することです。

本番環境に移行する

Let's Encrypt-staging でのテストが成功した後、実際の Let's Encrypt 証明書を取得できます。

Let's Encrypt エンドポイントを `https:acme-staging-v02.api.letsencrypt.org/directory` から `https:acme-v02.api.letsencrypt.org/directory` に変更する必要があります。

次に、ClusterIssuer の名前を `letsencrypt-staging` から `letsencrypt-production` に変更します。

```


apiVersion: cert-manager.io/v1alpha2

kind: ClusterIssuer

metadata:

  name: letsencrypt-prod

spec:

  acme:

    # You must replace this email address with your own.
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    # Let's Encrypt will use this to contact you about expiring

    # certificates, and issues related to your account.

    email: user@example.com

    server: https://acme-v02.api.letsencrypt.org/directory

    privateKeySecretRef:

      # Secret resource used to store the account's private key.

      name: example-issuer-account-key

    # Add a single challenge solver, HTTP01 using citrix

    solvers:

    - http01:

        ingress:

          class: citrix
```

注記:

`user@example.com` をメールアドレスに置き換えてください。

次のコマンドを使用してファイルをデプロイします。

```
% kubectl apply -f letsencrypt-prod.yaml



 clusterissuer "letsencrypt-prod" created
```

次に、Ingress のアノテーションを変更するか、新しい証明書の生成をトリガーする CRD 証明書を作成する手順を繰り返します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注

cert-manager が本番 CA で新しいチャレンジを開始できるように、古いシークレットを削除してください。

```
% kubectl delete secret kuard-example-tls



 secret "kuard-example-tls" deleted
```

HTTP ウェブサイトが稼働したら、Ingress オブジェクトの `ingress.citrix.com/insecure-termination: redirect` アノテーションを使用して、HTTP から HTTPS へのトラフィックをリダイレクトできます。

トラブルシューティング

証明書の生成には複数のコンポーネントが関与するため、このセクションでは、障害が発生した場合に使用できるトラブルシューティング手法をまとめます。

証明書生成のステータスを確認する

証明書 CRD オブジェクトは、証明書の生成と更新のライフサイクル管理を定義します。`kubectl describe` コマンドを使用して、証明書のステータスを次のように表示できます。

```
% kubectl get certificate



NAME                READY   SECRET              AGE

kuard-example-tls   False   kuard-example-tls   9s



%  kubectl describe certificate kuard-example-tls



Status:

  Conditions:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    Last Transition Time:  2019-03-05T09:50:29Z
    Message:               Certificate does not exist
    Reason:                NotFound
    Status:                False
    Type:                  Ready
Events:
  Type    Reason        Age   From          Message
  ----    ------        ----  ----          -------
  Normal  OrderCreated  22s   cert-manager  Created Order resource "kuard-example-tls-1754626579"
```

また、`kubectl events` コマンドを使用して、主要な証明書イベントを表示することもできます。

```
kubectl get events



LAST SEEN   TYPE     REASON              KIND          MESSAGE
36s         Normal   Started             Challenge     Challenge scheduled for processing
36s         Normal   Created             Order         Created Challenge resource "kuard-example-tls-1754626579-0" for domain "acme.cloudpst.net"
38s         Normal   OrderCreated        Certificate   Created Order resource "kuard-example-tls-1754626579"
38s         Normal   CreateCertificate   Ingress       Successfully created Certificate "kuard-example-tls"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

cert-manager からのログを分析する

障害が発生した場合、最初のステップは cert-manager コンポーネントからのログを分析することです。次のコマンドを使用して cert-manager ポッドを特定します。

```
% kubectl get po -n cert-manager



NAME                                       READY   STATUS      RESTARTS   AGE

cert-manager-76d48d47bf-5w4vx              1/1     Running     0          23h

cert-manager-webhook-67cfb86d56-6qtxr      1/1     Running     0          23h

cert-manager-webhook-ca-sync-x4q6f         0/1     Completed   4          23h
```

ここで、`cert-manager-76d48d47bf-5w4vx` はメインの cert-manager ポッドであり、他の2つのポッドは cert-manager ウェブフックポッドです。

次のコマンドを使用して cert-manager のログを取得します。

```
% kubectl logs -f cert-manager-76d48d47bf-5w4vx -n cert-manager
```

証明書の取得に失敗した場合、ERROR ログに失敗の詳細が示されます。

Kubernetes シークレットを確認する

`kubectl describe` コマンドを使用して、証明書とキーの両方がKubernetesシークレットに格納されているかを確認します。

```
% kubectl describe secret kuard-example-tls



Name:         kuard-example-tls

Namespace:    default

Labels:       certmanager.k8s.io/certificate-name=kuard-example-tls

Annotations:  certmanager.k8s.io/alt-names: acme.cloudpst.net
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
              certmanager.k8s.io/common-name: acme.cloudpst.net

              certmanager.k8s.io/issuer-kind: ClusterIssuer

              certmanager.k8s.io/issuer-name: letsencrypt-staging



Type:  kubernetes.io/tls



Data

====

tls.crt:  3553 bytes

tls.key:  1679 bytes

ca.crt:   0 bytes
```

`tls.crt` と `tls.key` の両方がKubernetesシークレットに格納されている場合、証明書の生成は完了しています。`tls.key` のみが存在する場合、証明書の生成は不完全です。問題の詳細については、cert-managerのログを分析してください。

NetScaler Ingress Controllerからのログを分析する

Kubernetesシークレットが生成され、完全であるにもかかわらずNetScalerにアップロードされていない場合、次のコマンドを使用してNetScaler Ingress Controllerからのログを分析できます。

```
% kubectl logs -f cpx-ingress-685c8bc976-zgz8q
```

cert-managerを使用して、NetScaler Ingress ControllerとHashiCorp VaultでKubernetes上にHTTPSウェブアプリケーションをデプロイする

NetScaler Ingress Controllerでデプロイされたイングレスリソースの場合、cert-managerとHashiCorp Vaultを使用してTLS証明書のプロビジョニング、失効、更新を自動化できます。このトピックでは、cert-managerからの証明書署名要求に対して、HashiCorp Vaultを自己署名証明書認証局として使用するワークフローの例を示します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

具体的には、このワークフローはVault PKI Secrets Engineを使用して証明書認証局（CA）を作成します。このチュートリアルは、Vaultサーバーがインストールされており、Kubernetesクラスターから到達可能であることを前提としています。VaultのPKIシークレットエンジンは、内部アプリケーションに適しています。公開信頼が必要な外部向けアプリケーションについては、Let's Encrypt CAを使用したTLS証明書の自動化を参照してください。

このワークフローでは、Vaultシークレットエンジンと認証方法を使用します。Vaultの全機能については、以下のVaultドキュメントを参照してください。

- ヴォールト シークレットエンジン
- ヴォールト 認証方法

このトピックでは、以下を使用してKubernetesクラスター上にHTTPSウェブアプリケーションをデプロイする方法に関する情報を提供します。

- ネットスケーラー イングレスコントローラー
- JetStackのcert-managerを使用してHashiCorp VaultからTLS証明書をプロビジョニングする
- ハシコープ ヴォールト

前提条件

以下を確認してください。

- Vaultサーバーがインストールされ、アンシールされており、Kubernetesクラスターから到達可能であること。Vaultサーバーのインストールについては、Vaultのインストールに関するドキュメントを参照してください。
- KubernetesクラスターでRBACが有効になっていること。
- NetScaler MPX、VPX、またはCPXがTier 1またはTier 2デプロイモデルでデプロイされていること。

  Tier 1展開モデルでは、NetScaler MPXまたはVPXがアプリケーションデリバリーコントローラー (ADC) として使用されます。Kubernetesクラスターで実行されているNetScaler Ingress Controllerは、Kubernetesクラスターで実行されているサービス用の仮想サービスを構成します。NetScalerは、公開ルーティング可能なIPアドレスで仮想サービスを実行し、Let's Encryptによって生成された証明書を使用してクライアントトラフィックのSSLをオフロードします。

  Tier 2展開では、Kubernetesクラスターの外部で実行されているNetScaler (VPX/MPX) 上でTCPサービスが構成され、Kubernetesクラスターで実行されているNetScaler CPXインスタンスにトラフィックを転送します。NetScaler CPXはSSLセッションを終了し、実際のサービスポッドにトラフィックをロードバランスします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- NetScaler Ingress Controllerが展開されました。さまざまな展開シナリオについては、展開トポロジを参照してください。
- すべての展開手順に対する管理者権限。権限が原因で障害が発生した場合は、管理者権限があることを確認してください。

注：

次の手順では、NetScaler CPXをイングレスデバイスとして使用して、Vaultを証明機関として構成する手順を示します。NetScaler VPXまたはMPXがイングレスデバイスとして使用される場合、NetScalerでのイングレス構成を確認する手順を除いて、手順は同じです。

マニフェストファイルを使用してcert-managerを展開する

提供されているYAMLマニフェストファイルを使用してcert-managerを展開するには、次の手順を実行します。

1. cert-managerをインストールします。cert-managerのインストールについては、cert-managerドキュメントを参照してください。

   ```
   kubectl apply -f https://github.com/jetstack/cert-manager/releases/
   download/vx.x.x/cert-manager.yaml
   ```

   Helmを使用してcert-managerをインストールすることもできます。詳細については、cert-managerドキュメントを参照してください。

2. 次のコマンドを使用して、cert-managerが稼働していることを確認します。

   ```
   % kubectl -n cert-manager get all

   NAME                                       READY   STATUS    RESTARTS
   AGE

   pod/cert-manager-77fd74fb64-d68v7          1/1     Running   0
    4m41s

   pod/cert-manager-webhook-67bf86d45-k77jj   1/1     Running   0
    4m41s



   NAME                           TYPE        CLUSTER-IP       EXTERNAL-IP
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  PORT(S)   AGE

service/cert-manager-webhook   ClusterIP   10.108.161.154   &lt;none>
     443/TCP   13d



NAME                                READY   UP-TO-DATE   AVAILABLE
AGE

deployment.apps/cert-manager        1/1     1            1
13d

deployment.apps/cert-manager-webhook   1/1     1            1
13d



NAME                                           DESIRED   CURRENT
READY   AGE

replicaset.apps/cert-manager-77fd74fb64          1         1         1
      13d

replicaset.apps/cert-manager-webhook-67bf86d45   1         1         1
      13d



NAME                                              COMPLETIONS
DURATION   AGE

job.batch/cert-manager-webhook-ca-sync              1/1           22s
     13d

job.batch/cert-manager-webhook-ca-sync-1549756800   1/1           21s
     10d

job.batch/cert-manager-webhook-ca-sync-1550361600   1/1           19s
     3d8h



NAME                                        SCHEDULE   SUSPEND
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
ACTIVE   LAST SCHEDULE   AGE

cronjob.batch/cert-manager-webhook-ca-sync   @weekly    False     0
   3d8h            13d
```

サンプルWebアプリケーションを展開する

サンプルWebアプリケーションを展開するには、次の手順を実行します。

注：

このトピックでは、Kubernetesデモアプリケーションである[Kuard]が参照用に使用されています。

1. Kuard用のデプロイメントYAMLファイル (`kuard-deployment.yaml`) を以下の構成で作成します。

```
        apiVersion: apps/v1

        kind: Deployment

        metadata:

          name: kuard

        spec:

          replicas: 1

          selector:

            matchLabels:

              app: kuard

          template:

            metadata:

              labels:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
                app: kuard
            spec:
              containers:
              - image: gcr.io/kuar-demo/kuard-amd64:1
                imagePullPolicy: Always
                name: kuard
                ports:
                - containerPort: 8080
```

2. 以下のコマンドを使用して、Kuardデプロイメントファイル (`kuard-deployment.yaml`) をクラスターにデプロイします。

```
% kubectl create -f kuard-deployment.yaml

deployment.extensions/kuard created

% kubectl get pod -l app=kuard

NAME                     READY   STATUS    RESTARTS   AGE

kuard-6fc4d89bfb-djljt   1/1     Running   0          24s
```

3. デプロイメント用のサービスを作成します。`service.yaml` という名前のファイルを以下の構成で作成します。

```
apiVersion: v1

kind: Service

metadata:

  name: kuard

spec:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  ports:
  - port: 80
    targetPort: 8080
    protocol: TCP
  selector:
    app: kuard
```

4. 以下のコマンドを使用してサービスをデプロイし、検証します。

```
% kubectl create -f service.yaml
service/kuard created
% kubectl get svc kuard
NAME    TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)   AGE
kuard   ClusterIP   10.103.49.171   &lt;none>        80/TCP    13s
```

5. NetScaler CPXまたはVPXにコンテンツスイッチング仮想サーバーとしてデプロイされるIngressを作成して、このサービスを外部に公開します。

注:

NetScaler Ingress Controllerが起動しているIngressクラスに`ingressClassName: citrix`を変更してください。

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: kuard
spec:
  ingressClassName: citrix
  rules:
  - host: kuard.example.com
    http:
      paths:
      - backend:
          service:
            name: kuard
            port:
              number: 80
        path: /
        pathType: Prefix
```

注:

`spec.rules.host`の値を、お客様が管理するドメインに変更してください。NetScaler CPXまたはVPXにトラフィックをルーティングするためのDNSエントリが存在することを確認してください。

6. 以下のコマンドを使用してIngressをデプロイします。

```
% kubectl apply -f ingress.yml
ingress.extensions/kuard created
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
root@ubuntu-vivek-225:~/cert-manager# kubectl get ingress

NAME    HOSTS               ADDRESS   PORTS   AGE

kuard   kuard.example.com             80      7s
```

7. 以下のコマンドを使用して、IngressがNetScaler CPXまたはVPXで構成されていることを確認します。

```
kubectl exec -it cpx-ingress-5b85d7c69d-ngd72 /bin/bash

root@cpx-ingress-5b85d7c69d-ngd72:/# cli_script.sh 'sh cs vs'

exec: sh cs vs

1) k8s-10.244.1.50:80:http (10.244.1.50:80) - HTTP Type: CONTENT

  State: UP

  Last state change was at Thu Feb 21 09:02:14 2019

  Time since last state change: 0 days, 00:00:41.140

  Client Idle Timeout: 180 sec

  Down state flush: ENABLED

  Disable Primary Vserver On Down : DISABLED

  Comment: uid=75VBGFO7NZXV7SCI4LSDJML2Q5X6FSNK6NXQPWGMDOYGBW2IMOGQ====

  Appflow® logging: ENABLED

  Port Rewrite : DISABLED

  State Update: DISABLED

  Default: Content Precedence: RULE

  Vserver IP and Port insertion: OFF

  L2Conn: OFF Case Sensitivity: ON
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  Authentication: OFF

  401 Based Authentication: OFF

  Push: DISABLED Push VServer:

  Push Label Rule: none

  Listen Policy: NONE

  IcmpResponse: PASSIVE

  RHIstate:  PASSIVE

  Traffic Domain: 0

Done

root@cpx-ingress-5b85d7c69d-ngd72:/# exit

exit
```

8. `curl`コマンドを使用して要求されたときに、ページが正しく提供されていることを確認します。

```
% curl -sS -D - kuard.example.com -o /dev/null

HTTP/1.1 200 OK

Content-Length: 1458

Content-Type: text/html

Date: Thu, 21 Feb 2019 09:09:05 GMT
```

サンプルHTTPアプリケーションをデプロイしたら、HTTPS経由でアプリケーションを利用できるように進めることができます。ここでは、Vaultサーバーがcert-managerによって生成されたCSRに署名し、アプリケーション用のサーバー証明書が自動的に生成されます。

以下の手順では、設定済みのVaultを認証局として使用し、cert-managerがCSRの署名機関としてVaultを使用するように構成します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

HashiCorp Vaultを認証局として構成する

この手順では、HashiCorp Vault を使用して中間 CA 証明書署名要求を設定します。この Vault エンドポイントは、cert-manager によってイングレスリソースの証明書に署名するために使用されます。

注記：

これらの手順を実行する前に、`jq` ユーティリティがインストールされていることを確認してください。

ルート CA を作成する

サンプルワークフローでは、Vault 内で独自のルート認証局を生成できます。本番環境では、Vault が証明書を生成するために使用する中間 CA に署名するために、外部ルート CA を使用する必要があります。ルート CA が他の場所で生成されている場合は、この手順をスキップしてください。

注記：

`PKI_ROOT` はルート CA をマウントするパスで、通常は `pki` です。この手順における ${DOMAIN} は `example.com` です。

```
% export DOMAIN=example.com
% export PKI_ROOT=pki

% vault secrets enable -path="${PKI_ROOT}" pki

# Set the max TTL for the root CA to 10 years
% vault secrets tune -max-lease-ttl=87600h "${PKI_ROOT}"

% vault write -format=json "${PKI_ROOT}"/root/generate/internal \
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
 common_name="${DOMAIN} CA root" ttl=87600h | tee \

>(jq -r .data.certificate > ca.pem) \

>(jq -r .data.issuing_ca > issuing_ca.pem) \

>(jq -r .data.private_key > ca-key.pem)



#Configure the CA and CRL URLs:



% vault write "${PKI_ROOT}"/config/urls \

       issuing_certificates="${VAULT_ADDR}/v1/${PKI_ROOT}/ca" \

       crl_distribution_points="${VAULT_ADDR}/v1/${PKI_ROOT}/crl"
```

中間 CA を生成する

ルート CA を作成した後、ルート CA を使用して中間 CSR を作成するには、次の手順を実行します。

1. ルート CA とは異なるパス `PKI_INT` (通常は `pki\_int`) から pki を有効にします。次のコマンドを使用します。

   ```
   % export PKI_INT=pki_int

   % vault secrets enable -path=${PKI_INT} pki



   # Set the max TTL to 3 year



   % vault secrets tune -max-lease-ttl=26280h ${PKI_INT}
   ```

2. ルート CA によって署名される必要がある `${DOMAIN}` の CSR を生成します。キーは Vault の内部に保存されます。次のコマンドを使用します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
   % vault write -format=json "${PKI_INT}"/intermediate/generate/
internal \

   common_name="${DOMAIN} CA intermediate" ttl=26280h | tee \

   >(jq -r .data.csr > pki_int.csr) \

   >(jq -r .data.private_key > pki_int.pem)
```

3. `${DOMAIN}` 証明書を中間 CA としてルート CA を使用して生成および署名し、`intermediate.cert.pem` として保存します。次のコマンドを使用します。

```
% vault write -format=json "${PKI_ROOT}"/root/sign-intermediate
csr=@pki_int.csr

        format=pem_bundle ttl=26280h \

        | jq -r '.data.certificate' > intermediate.cert.pem
```

   外部ルート CA を使用している場合は、前の手順をスキップし、ルート CA を使用して CSR に手動で署名します。

4. CSR が署名され、ルート CA が証明書を返したら、次のコマンドを使用して Vault に戻す必要があります。

```
% vault write "${PKI_INT}"/intermediate/set-signed
certificate=@intermediate.cert.pem
```

5. 次のコマンドを使用して、CA および CRL の場所を設定します。

```
vault write "${PKI_INT}"/config/urls issuing_certificates="$
{VAULT_ADDR}/v1/${PKI_INT}/ca" crl_distribution_points="${VAULT_ADDR}/
v1/${PKI_INT}/crl"
```

中間CAが設定されており、イングレスリソースの証明書に署名するために使用できます。

ロールを構成する

ロールはポリシーにマッピングされる論理名です。管理者はロールを通じて証明書の生成を制御できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

このCAを使用して証明書を発行または署名するためのポリシーセットを提供する、中間CAのロールを作成します。

ロールを作成する際に設定できる構成は多数あります。詳細については、Vaultロールのドキュメントを参照してください。

ワークフローのために、`${DOMAIN}`とそのサブドメインの証明書に90日間のTTLで署名できるロール`kube-ingress`を作成します。

```
# with a Max TTL of 90 days

vault write ${PKI_INT}/roles/kube-ingress \

          allowed_domains=${DOMAIN} \

          allow_subdomains=true \

          max_ttl="2160h" \

          require_cn=false
```

AppRoleベースの認証を作成する

証明書に署名するための中間CAを構成した後、cert-managerがVaultを使用して証明書に署名するための認証メカニズムを提供する必要があります。Cert-managerは、アプリケーションがVaultで定義されたロールにアクセスする方法を提供するAppRole認証メソッドをサポートしています。

`AppRole`は、それらのポリシーを持つトークンを受け取るために満たされなければならないVaultポリシーとログイン制約のセットを表します。この認証方法の詳細については、AppRoleのドキュメントを参照してください。

AppRoleを作成する

`Kube-role`という名前のAppRoleを作成します。cert-managerが認証にこのAppRoleを使用するためには、`secret_id`が期限切れになっていない必要があります。したがって、TTLを設定しないか、0に設定してください。

```
% vault auth enable approle
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
% vault write auth/approle/role/kube-role token_ttl=0
```

ポリシーをAppRoleに関連付ける

AppRoleにポリシーを関連付けるには、次の手順を実行します。

1. 中間CAの署名エンドポイントを許可するために、次の構成でファイル`pki_int.hcl`を作成します。

   ```
   path "${PKI_INT}/sign/*" {

         capabilities = ["create","update"]

       }
   ```

2. 次のコマンドを使用して、ファイルを`kube_allow_sign`という新しいポリシーに追加します。

   ```
   vault policy write kube-allow-sign pki_int.hcl
   ```

3. 以下のコマンドを使用して、このポリシーをApproleに更新します。

   ```
   vault write auth/approle/role/kube-role policies=kube-allow-sign
   ```

`kube-role` approleは、中間CAでCSRに署名することを可能にします。

ロールIDとシークレットIDを生成する

ロールIDとシークレットIDは、cert-managerがVaultで認証するために使用されます。

ロールIDとシークレットIDを生成し、シークレットIDをBase64でエンコードします。以下を実行します。

```
% vault read auth/approle/role/kube-role/role-id

role_id     db02de05-fa39-4855-059b-67221c5c2f63



% vault write -f auth/approle/role/kube-role/secret-id
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
secret_id               6a174c20-f6de-a53c-74d2-6018fcceff64

secret_id_accessor      c454f7e5-996e-7230-6074-6ef26b7bcf86



# encode secret_id with base64

% echo 6a174c20-f6de-a53c-74d2-6018fcceff64 | base64

NmExNzRjMjAtZjZkZS1hNTNjLTc0ZDItNjAxOGZjY2VmZjY0Cg==
```

Kubernetesでの証明書発行を構成する

Vaultを中間CAとして、またcert-managerがVaultにアクセスするためのApprole認証方法を設定した後、Ingress用の証明書を設定する必要があります。

ApproleシークレットIDでシークレットを作成する

ApproleシークレットIDでシークレットを作成するには、以下を実行します。

1. 以下の構成で`secretid.yaml`という名前のシークレットファイルを作成します。

```
apiVersion: v1

kind: Secret

type: Opaque

metadata:

  name: cert-manager-vault-approle

  namespace: cert-manager

data:

  secretId: "NmExNzRjMjAtZjZkZS1hNTNjLTc0ZDItNjAxOGZjY2VmZjY0Cg=="
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注記：

シークレットID `data.secretId` は、ロールIDとシークレットIDを生成する で生成されたBase64エンコードされたシークレットIDです。次のステップでIssuerリソースを使用する場合、シークレットは `Issuer` と同じ名前空間にある必要があります。`ClusterIssuer` の場合、シークレットは `cert-manager` 名前空間にある必要があります。

2. 以下のコマンドを使用して、シークレットファイル (`secretid.yaml`) をデプロイします。

```
% kubectl create -f secretid.yaml
```

Vaultクラスターイシューアーをデプロイする

cert-managerは、構成用に2つの異なるCRDをサポートしています。1つは単一の名前空間にスコープされた `Issuer`、もう1つはクラスター全体にわたる `ClusterIssuer` です。このワークフローでは、`ClusterIssuer` を使用する必要があります。

Vaultクラスター発行者をデプロイするには、次の手順を実行します。

1. 次の構成で `issuer-vault.yaml` というファイルを作成します。

```
apiVersion: cert-manager.io/v1

kind: ClusterIssuer

metadata:

  name: vault-issuer

spec:

  vault:

    path: pki_int/sign/kube-ingress

    server: &lt;vault-server-url>

    #caBundle: &lt;base64 encoded caBundle PEM file>

    auth:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      appRole:
        path: approle
        roleId: "db02de05-fa39-4855-059b-67221c5c2f63"
        secretRef:
          name: cert-manager-vault-approle
          key: secretId
```

`SecretRef`は、前の手順で作成したKubernetesシークレット名です。`roleId`をVaultから取得した`role_id`に置き換えます。 VaultサーバーへのTLS接続を検証するために、PEM形式のオプションのbase64エンコードされたcaBundleを提供できます。caBundleが設定されている場合、cert-managerを実行しているコンテナ内のCAバンドルを置き換えます。使用される接続がプレーンHTTPの場合、このパラメータは効果がありません。

2. 次のコマンドを使用して、ファイル (`issuer-vault.yaml`) をデプロイします。

```
% kubectl create -f issuer-vault.yaml
```

3. 次のコマンドを使用して、Vaultクラスター発行者がVaultで正常に認証されていることを確認します。

```
% kubectl describe clusterIssuer vault-issuer  | tail -n 7
  Conditions:
    Last Transition Time:  2019-02-26T06:18:40Z
    Message:               Vault verified
    Reason:                VaultVerified
    Status:                True
    Type:                  Ready
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
Events:                  &lt;none>
```

これで、VaultをCAとしてcert-managerのセットアップが正常に完了しました。次のステップは、サーバー証明書を生成してイングレスを保護することです。イングレスを保護するには2つの異なるオプションがあります。いずれかの方法でイングレスを保護できます。

- Ingress Shimアプローチ
- 証明書用の`certificate` CRDオブジェクトを手動で作成する。

Ingress-shimアプローチ

このアプローチでは、cert-managerのイングレスアノテーションを変更して、指定されたホスト名の証明書を自動的に生成し、指定されたシークレットに保存します。

1. ホスト名とシークレットを指定する`tls`セクションでイングレスを変更します。また、cert-managerアノテーション`cert-manager.io/cluster-issuer`を次のように指定します。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  annotations:

    cert-manager.io/cluster-issuer: vault-issuer

  name: kuard

spec:

  ingressClassName: citrix

  rules:

  - host: kuard.example.com

    http:

      paths:

      - backend:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          service:
            name: kuard-service
            port:
              number: 80
        path: /
        pathType: Prefix
  tls:
  - hosts:
    - kuard.example.com
    secretName: kuard-example-tls
```

2. 変更されたイングレスを次のようにデプロイします。

```
% kubectl apply -f ingress.yml
ingress.extensions/kuard created


% kubectl get ingress kuard
NAME    HOSTS               ADDRESS   PORTS     AGE
kuard   kuard.example.com             80, 443   12s
```

このステップは、cert-managerによって`certificate`オブジェクトをトリガーし、ドメイン`kuard.example.com`の証明書署名要求 (CSR) を作成します。CSRの署名が成功すると、証明書はイングレスで指定されたシークレット名`kuard-example-tls`に保存されます。

1. 次のコマンドを使用して、証明書が正常に発行されたことを確認します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
% kubectl describe certificates kuard-example-tls  | grep -A5 Events

Events:

Type    Reason      Age   From          Message

----    ------      ----  ----          -------

Normal  CertIssued  48s   cert-manager  Certificate issued successfully
```

証明書用の `certificate` CRDオブジェクトを作成する

発行者が正常に登録されたら、イングレスドメイン `kuard.example.com` の証明書を取得する必要があります。

`commonName` と `dnsNames` を使用して `certificate` リソースを作成する必要があります。詳細については、cert-manager documentation を参照してください。証明書の SAN フィールドに使用される複数の dnsNames を指定できます。

証明書用の「certificate」CRD オブジェクトを作成するには、以下を実行します。

1. 次の構成で `certificate.yaml` という名前のファイルを作成します。

```
apiVersion: cert-manager.io/v1

kind: Certificate

metadata:

  name: kuard-example-tls

  namespace: default

spec:

  secretName: kuard-example-tls

  issuerRef:

    kind: ClusterIssuer

    name: vault-issuer
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  commonName: kuard.example.com

  duration: 720h

  #Renew before 7 days of expiry

  renewBefore: 168h

  commonName: kuard.example.com

  dnsNames:

  - www.kuard.example.com
```

証明書には CN=`kuard.example.com` と SAN=`Kuard.example.com,www.kuard.example.com` があります。 `spec.secretName` は、証明書が正常に発行された後に証明書が保存されるシークレットの名前です。

2. 次のコマンドを使用して、ファイル (`certificate.yaml`) を Kubernetes クラスターにデプロイします。

   % kubectl create -f certificate.yaml certificate.certmanager.k8s.io/kuard-example-tls 作成されました

証明書が発行されているか確認する

次のコマンドを使用して、証明書の発行状況を監視できます。

```
% kubectl describe certificates kuard-example-tls  | grep -A5 Events

Events:

  Type    Reason      Age   From          Message

  ----    ------      ----  ----          -------

  Normal  CertIssued  48s   cert-manager  Certificate issued successfully
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注

Vault ポリシーが原因でエラーが発生する場合があります。そのようなエラーが発生した場合は、Vault に戻って修正してください。

署名が成功すると、`Certificate` リソースで指定された `secretName` を持つ `kubernetes.io/tls` シークレットが作成されます。

```
% kubectl get secret kuard-example-tls

NAME                TYPE                DATA   AGE

kuard-exmaple-tls   kubernetes.io/tls   3      4m20s
```

生成されたシークレットを使用するようにイングレスを変更する

生成されたシークレットを使用するようにイングレスを変更するには、次の手順を実行します。

1. 元のイングレスを編集し、シークレット `kuard-example-tls` を指定する `spec.tls` セクションを次のように追加します。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  annotations:

  name: kuard

spec:

  ingressClassName: citrix®

  rules:

  - host: kuard.example.com

    http:

      paths:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      - backend:
          service:
            name: kuard
            port:
              number: 80
        pathType: Prefix
        path: /
  tls:
  - hosts:
    - kuard.example.com
    secretName: kuard-example-tls
```

2. 以下のコマンドを使用してイングレスをデプロイします。

```
% kubectl apply -f ingress.yml
ingress.extensions/kuard created



% kubectl get ingress kuard
NAME    HOSTS               ADDRESS   PORTS     AGE
kuard   kuard.example.com             80, 443   12s
```

NetScaler でイングレス構成を確認する

証明書が正常に生成されると、NetScaler Ingress Controller はこの証明書を使用してフロントエンド SSL 仮想サーバーを構成します。以下の手順で確認できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

1. NetScaler CPX にログオンし、証明書が SSL 仮想サーバーにバインドされていることを確認します。

```
% kubectl exec -it cpx-ingress-668bf6695f-4fwh8 bash

cli_script.sh 'shsslvs'

exec: shsslvs

1) Vserver Name: k8s-10.244.3.148:443:ssl

  DH: DISABLED

  DH Private-Key Exponent Size Limit: DISABLED Ephemeral RSA: ENABLED
Refresh Count: 0

  Session Reuse: ENABLED Timeout: 120 seconds

  Cipher Redirect: DISABLED

  SSLv2 Redirect: DISABLED

  ClearText Port: 0

  Client Auth: DISABLED

  SSL Redirect: DISABLED

  Non FIPS Ciphers: DISABLED

  SNI: ENABLED

  OCSP Stapling: DISABLED

  HSTS: DISABLED

  HSTS IncludeSubDomains: NO

  HSTS Max-Age: 0

  SSLv2: DISABLED  SSLv3: ENABLED  TLSv1.0: ENABLED  TLSv1.1: ENABLED
 TLSv1.2: ENABLED  TLSv1.3: DISABLED
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  Push Encryption Trigger: Always

  Send Close-Notify: YES

  Strict Sig-Digest Check: DISABLED

  Zero RTT Early Data: DISABLED

  DHE Key Exchange With PSK: NO

  Tickets Per Authentication Context: 1

Done



root@cpx-ingress-668bf6695f-4fwh8:/# cli_script.sh 'shsslvs
k8s-10.244.3.148:443:ssl'

exec: shsslvs k8s-10.244.3.148:443:ssl



  Advanced SSL configuration for VServer k8s-10.244.3.148:443:ssl:

  DH: DISABLED

  DH Private-Key Exponent Size Limit: DISABLED Ephemeral RSA: ENABLED
Refresh Count: 0

  Session Reuse: ENABLED Timeout: 120 seconds

  Cipher Redirect: DISABLED

  SSLv2 Redirect: DISABLED

  ClearText Port: 0

  Client Auth: DISABLED

  SSL Redirect: DISABLED

  Non FIPS Ciphers: DISABLED
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  SNI: ENABLED

  OCSP Stapling: DISABLED

  HSTS: DISABLED

  HSTS IncludeSubDomains: NO

  HSTS Max-Age: 0

  SSLv2: DISABLED  SSLv3: ENABLED  TLSv1.0: ENABLED  TLSv1.1: ENABLED
 TLSv1.2: ENABLED  TLSv1.3: DISABLED

  Push Encryption Trigger: Always

  Send Close-Notify: YES

  Strict Sig-Digest Check: DISABLED

  Zero RTT Early Data: DISABLED

  DHE Key Exchange With PSK: NO

  Tickets Per Authentication Context: 1

, P_256, P_384, P_224, P_5216) CertKey Name: k8s-
LMO3O3U6KC6WXKCBJAQY6K6X6JO Server Certificate for SNI



7) Cipher Name: DEFAULT

  Description: Default cipher list with encryption strength >= 128bit

Done



root@cpx-ingress-668bf6695f-4fwh8:/# cli_script.sh 'sh certkey k8s-
LMO3O3U6KC6WXKCBJAQY6K6X6JO'

exec: sh certkey k8s-LMO3O3U6KC6WXKCBJAQY6K6X6JO
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  Name: k8s-LMO3O3U6KC6WXKCBJAQY6K6X6JO Status: Valid,   Days to
expiration:0

  Version: 3

  Serial Number: 524C1D9306F784A2F5277C05C2A120D5258D9A2F

  Signature Algorithm: sha256WithRSAEncryption

  Issuer:  CN=example.com CA intermediate

  Validity

    Not Before: Feb 26 06:48:39 2019 GMT

    Not After : Feb 27 06:49:09 2019 GMT

  Certificate Type: "Client Certificate" "Server Certificate"

  Subject:  CN=kuard.example.com

  Public Key Algorithm: rsaEncryption

  Public Key size: 2048

  Ocsp Response Status: NONE

  2) URI:http://127.0.0.1:8200/v1/pki_int/crl

  3) VServer name: k8s-10.244.3.148:443:ssl Server Certificate for SNI

Done
```

HTTPS ウェブサーバーは、Vault で署名された証明書で稼働しています。Cert-manager は、証明書に指定された `RenewBefore` パラメーターに従って、証明書の有効期限が切れる前に自動的に証明書を更新します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

証明書の有効期限がルート CA または中間 CA の有効期限を超えている場合、Vault による証明書の署名は失敗します。CA 証明書は、有効期限が切れる前に事前に更新されていることを確認する必要があります。

2. HTTPS プロトコルを使用してアプリケーションにアクセスできることを確認します。

```
% curl -sS -D - https://kuard.example.com -k -o /dev/null

HTTP/1.1 200 OK

Content-Length: 1472

Content-Type: text/html

Date: Tue, 11 May 2021 20:39:23 GMT
```

NetScaler Ingress Controller で NetScaler 証明書検証を有効にする

NetScaler Ingress Controller は、HTTPS プロトコルを使用して NetScaler Ingress Controller と NetScaler 間の安全な通信を確保するオプションを提供します。これは、NetScaler に事前にロードされた証明書を使用することで実現できます。考えられる中間者 (MITM) 攻撃を回避するための追加措置として、NetScaler Ingress Controller は NetScaler によって提供される SSL サーバー証明書を検証することもできます。

NetScaler Ingress Controller による ADC サーバー証明書の証明書署名と共通名検証を有効にするには、セキュリティ管理者はオプションで署名済み (または自己署名) 証明書を NetScaler にインストールし、対応する CA 証明書バンドルで NetScaler Ingress Controller を構成できます。検証が有効になり、CA 証明書バンドルが構成されると、NetScaler Ingress Controller は証明書の検証 (証明書名検証を含む) を開始します。検証が失敗した場合、NetScaler Ingress Controller はその旨をログに記録し、構成は安全でないチャネルでは使用されません。

この検証はデフォルトで無効になっており、管理者は次のように NetScaler Ingress Controller で検証を有効にすることを選択できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

前提条件

- 証明書検証を有効にするには、適切な SSL サーバー証明書 (証明書のサブジェクトに適切なサーバー名または IP アドレスを含む) を持つ NetScaler を構成する必要があります。詳細については、NetScaler ドキュメント を参照してください。
- インストールされているサーバー証明書とキーのペアの CA 証明書は、これらの証明書の検証を有効にするために NetScaler Ingress Controller を構成するために使用されます。

証明書検証のために NetScaler Ingress Controller を構成する

CA 証明書を構成で利用できるようにするには、NetScaler Ingress Controller がマウントされたストレージボリュームでアクセスできるように、CA 証明書を Kubernetes シークレットとして構成する必要があります。

既存の証明書に対して Kubernetes シークレットを生成するには、次の `kubectl` コマンドを使用します。

```
  $ kubectl create secret generic ciccacert --from-file=path/myCA.pem –
namespace default



  secret “ciccacert” created
```

または、次の YAML 定義を使用して Kubernetes シークレットを生成することもできます。

```
apiVersion: v1

kind: Secret

metadata:

  name: ciccacert

  data:

    myCA.pem: &lt;base64 encoded cert>
```

以下は、証明書検証を有効にするための NetScaler Ingress Controller 構成を含むサンプル YAML ファイルです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kind: Pod

metadata:

  name: cic

  labels:

    app: cic

spec:

  serviceAccountName: cpx

  # Make secret available as a volume

  volumes:

  - name: certs

    secret:

      secretName: ciccacert

  containers:

  - name: cic

    image: "xxxx"

    imagePullPolicy: Always

    args: []

    # Mounting certs in a volume path

    volumeMounts:

    - name: certs

      mountPath: <Path to mount the certificate>

      readOnly: true
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    env:
    # Set NetScaler Console Management IP
    - name: "NS_IP"
      value: "xx.xx.xx.xx"
    # Set port for Nitro
    - name: "NS_PORT"
      value: "xx"
    # Set Protocol for Nitro
    - name: "NS_PROTOCOL"
      # Enable HTTPS protocol for secure communication
      value: "HTTPS"
    # Set username for Nitro
    - name: "NS_USER"
      value: "nsroot"
    # Set user password for Nitro
    - name: "NS_PASSWORD"
      value: "nsroot"
    # Certificate validation configurations
    - name: "NS_VALIDATE_CERT"
      value: "yes"
    - name: "NS_CACERT_PATH"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      value: " <Mounted volume path>/myCA.pem"
```

サンプル YAML ファイルで指定されているように、NetScaler Ingress Controller で証明書検証を有効にするために必要な特定の変更は次のとおりです。

Kubernetes シークレットをボリュームとして構成する

- `secret` をソースとして宣言されたボリュームセクションを構成します。ここで、`secretName` は CA 証明書用に作成された Kubernetes シークレット名と一致する必要があります。

CA証明書のボリュームマウント場所を構成する

- ボリュームセクションの`secretName`と同じ名前の`volumeMounts`セクションを構成します。
- CA証明書をマウントするための`mountPath`ディレクトリを宣言します。
- ボリュームを`ReadOnly`として設定します。

セキュアな通信を構成する

- 環境変数`NS_PROTOCOL`をHTTPSとして設定します。
- 環境変数`NS_PORT`をADC HTTPSポートとして設定します。

CA検証と証明書パスを有効にして構成する

- 環境変数`NS_VALIDATE_CERT`を`yes`に設定します（無効にする場合は`no`）。
- 環境変数`NS_CACERT_PATH`をマウントパス（volumeMounts->mountPath）/ PEMファイル名（シークレット作成時に使用）として設定します。

APIサーバー証明書の検証を無効にする

NetScaler Ingress ControllerまたはGSLBイングレスからAPIサーバーと通信する際、NetScaler Ingress ControllerでAPIサーバー証明書の検証を無効にするオプションがあります。

NetScaler Ingress ControllerまたはGSLBイングレスでAPIサーバー証明書の検証を無効にする

YAMLを使用してNetScaler Ingress Controllerを展開する場合、NetScaler Ingress Controller展開YAMLファイルに次の引数を指定することで、APIサーバー証明書の検証を無効にできます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
args:

   - --disable-apiserver-cert-verify

     true
```

Helmチャートを使用してNetScaler Ingress Controllerを展開する場合、パラメータ `disableAPIServerCertVerify` は、Helm valuesファイルで `True` として次のように指定できます。

```
disableAPIServerCertVerify: True
```

自己署名証明書を作成し、Kubernetesシークレットにリンクする

手順のステップを使用して、OpenSSLで自己署名証明書を作成し、Kubernetesシークレットにリンクします。このシークレットを使用して、Ingressを保護できます。

自己署名証明書を作成する

以下の手順でTLSシークレットを作成できます。この手順では、自己署名証明書とキーが作成されます。 それをKubernetesシークレットにリンクし、そのシークレットをIngressでIngressを保護するために使用できます。

```
openssl genrsa -out cert_key.pem 2048

openssl req -new -key cert_key.pem -out cert_csr.pem -subj "/
CN=example.com"

openssl x509 -req -in cert_csr.pem -sha256 -days 365 -extensions v3_ca
-signkey cert_key.pem -CAcreateserial -out cert_cert.pem
```

注:

ここでは、 `example.com` が参照用に使用されています。 `example.com` を必要なドメイン名に置き換える必要があります。

この例では、日数が365日と記載されているため、生成された証明書の有効期間は1年間です。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

証明書をKubernetesシークレットにリンクする

証明書をKubernetesシークレットにリンクするには、以下の手順を実行します。

1. 作成したTLS証明書に基づいてKubernetesシークレットを作成するには、次のコマンドを実行します。

```
kubectl create secret tls tls-secret --cert=cert_cert.pem --
key=cert_key.pem
```

2. TLS証明書情報を含むシークレットを表示するには、次のコマンドを実行します。

```
kubectl get secret tls-secret
```

Ingressをデプロイする

Ingress構成を作成して適用します。以下のYAMLは参照用として使用できます。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: ingress-demo

  namespace: netscaler®

  annotations:

spec:

  ingressClassName: citrix®

  tls:

  - secretName: tls-secret

    hosts:

      - "example.com"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
rules:

- host:  "example.com"

  http:

    paths:

    - path: /

      pathType: Prefix

      backend:

        service:

          name: service-test

          port:

            number: 80
```

CRD

CRDには以下のトピックがあります:

- OWASP Top 10 保護ポリシー
- 認証
- レート制限
- リライトおよびレスポンダーポリシー
- ICAP を使用したリモートコンテンツ検査またはコンテンツ変換サービス
- IPAM または VIP
- HTTPルート
- コンテンツルーティング
- リスナー
- Web アプリケーションファイアウォールポリシー
- ボット

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- CORS
- AppQoE を使用したリクエスト再試行の有効化
- ワイルドカードDNS(/ja-jp/netscaler-k8s-ingress-controller/crds/wildcarddns.html)

OWASP CRD を使用して OWASP Top 10 保護ポリシーを構成する

OWASP Top 10:2025 は、Webアプリケーションにとって最も重大なセキュリティリスクを表します。OWASP CRD (`owasppolicy`) を使用すると、単一のKubernetesカスタムリソースで、OWASP Top 10カテゴリに沿った包括的なWAFおよびボット管理保護を構成できます。

OWASP Top 10:2025 のカテゴリは次のとおりです。

1. A01:2025 - 破損したアクセス制御
2. A02:2025 - セキュリティの誤設定
3. A03:2025 - ソフトウェアサプライチェーンの障害
4. A04:2025 - 暗号化の障害
5. A05:2025 - インジェクション
6. A06:2025 - 安全でない設計
7. A07:2025 - 認証の障害
8. A08:2025 - ソフトウェアまたはデータの整合性の障害
9. A09:2025 - セキュリティログとアラートの障害
10. A10:2025 - 例外条件の不適切な処理

OWASP Top 10 に従って、次のルールが `owasppolicy` CRD によってサポートされています。CRD は主に2つのセクションで構成されています。

- ボット保護
- WAF (ウェブアプリケーションファイアウォール) 保護。

注記:

既存の **Bot** および **Waf** CRD を使用したい場合は、 `Owasppolicy` CRD で同じフィールドを有効にしないようにしてください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

OWASP CRD 定義

OWASP CRD の定義は owasp-crd.yaml で入手できます。次のコマンドを実行してデプロイします。

```
kubectl create -f owasp-crd.yaml
```

この CRD は `apiVersion: citrix.com/v1` および `kind: owasppolicy` を使用します。

OWASP CRD 構造

トップレベルの仕様には以下が含まれます。

CRD 属性	説明
`ingressclass`	どの Ingress コントローラーがこのリソースを処理するかをスコープする Ingress クラス。
`servicenames`	OWASP ポリシーが適用されるサービスの一覧。
`bot`	ボット管理保護設定。
`waf`	Webアプリケーションファイアウォール保護設定。

NetScalerのトップレベルエンティティマッピング:

CRDセクション	NETSCALERエンティティ	NETSCALER属性
`bot`	`botprofile`	すべてのボット保護設定を含むプロファイル
`bot`	`botpolicy`	`rule`、`profilename` — LB仮想サーバーにバインドされたポリシー
`bot`	`lbvserver_botpolicy_binding`	`policyname`, `bindpoint=REQUEST`
`waf`	`appfwprofile`	すべてのWAF保護設定を含むプロファイル

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRDセクション	NETSCALERエンティティ	NETSCALER属性
`waf`	`appfwpolicy`	`rule`, `profilename` — LB仮想サーバーにバインドされたポリシー
`waf`	`lbvserver_appfwpolicy_binding`	`policyname`, `bindpoint=REQUEST`

ボット保護

許可リストとブロックリスト

トラフィックを許可またはブロックするためのルール式を追加します。

属性	説明
`action`	`allow` または `block`
`expression`	トラフィックを照合するためのADCポリシー式。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`action: allow`	`botprofile`	`bot_enable_white_list`	`ON`
`action: allow`, `expression`	`botprofile_whitelist_binding`	`bot_whitelist_type`, `bot_whitelist_value`, `bot_whitelist_enabled`	`EXPRESSION`, `HTTP.REQ.URL.PATH.EQ("apache.com")`, `ON`
`action: block`	`botprofile`	`bot_enable_black_list`	`ON`
`action: block`, `expression`	`botprofile_blacklist_binding`	`bot_blacklist_type`, `bot_blacklist_va`	`EXPRESSION`, `HTTP.REQ.URL.PATH.EQ("nginx.com")`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
		`lue`, `bot_blacklist_action`, `bot_blacklist_enabled`	, `[LOG, DROP]`, `ON`

地域ブロック

ISO 3166-1 alpha-2 国コードを使用して、ブロックする国のリストを追加します。

属性	説明
`blocked-countries`	国コードのリスト（例：`CN`、`RU`）。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`blocked-countries`	`policypatset`	`name`	`<profile_name>`
`blocked-countries[*]`	`policypatset_pattern_binding`	`String`, `index`	`CN`, `1`
`blocked-countries`	`botprofile_blacklist_binding`	`bot_blacklist_type`, `bot_blacklist_value`, `bot_blacklist_action`	`EXPRESSION`, `CLIENT.IP.SRC.MATCHES_LOCATION("<patset_name>")`, `[LOG, DROP]`

IPレピュテーション

以下のカテゴリから選択します（デフォルト：すべて有効）：

- スパムソース
- Windowsエクスプロイト
- Webエクスプロイト

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- ボットネットランサムウェアC&C
- スキャナー
- フィッシング
- 匿名または不審なプロキシ
- モバイルの脅威
- Torプロキシ
- DoS
- クラウド
- クラウド (AWS)
- クラウド (Azure)
- クラウド (IBM)
- クラウド (Oracle)
- クラウド (Salesforce)
- IP
- ネットワーク
- レピュテーション

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`ip-reputation`	`botprofile`	`bot_enable_ip_reputation`	`ON`
`spam-sources`	`botprofile_ireputation_binding`	`category`, `bot_iprep_enabled`, `bot_iprep_action`	`SPAM_SOURCES`, `ON`, `[LOG, DROP]`
`windows-exploits`	`botprofile_ireputation_binding`	`category`	`WINDOWS_EXPLOITS`
`web-exploits`	`botprofile_ireputation_binding`	`category`	`WEB_ATTACKS`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
botnets-ransomware-ccs	botprofile_ipreputation_binding	category	BOTNETS
scanners	botprofile_ipreputation_binding	category	SCANNERS
phishing	botprofile_ipreputation_binding	category	PHISHING
anonymous-or-suspicious-proxies	botprofile_ipreputation_binding	category	PROXY
tor-proxies	botprofile_ipreputation_binding	category	TOR_PROXY
dos	botprofile_ipreputation_binding	category	DOS
cloud	botprofile_ipreputation_binding	category	CLOUD
cloud-aws	botprofile_ipreputation_binding	category	CLOUD_AWS
cloud-azure	botprofile_ipreputation_binding	category	CLOUD_AZURE
cloud-ibm	botprofile_ipreputation_binding	category	CLOUD_IBM
cloud-oracle	botprofile_ipreputation_binding	category	CLOUD_ORACLE
cloud-salesforce	botprofile_ipreputation_binding	category	CLOUD_SALESFORCE
ip	botprofile_ipreputation_binding	category	IP

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`network`	`botprofile_ipreputation_binding`	`category`	`NETWORK`
`reputation`	`botprofile_ipreputation_binding`	`category`	`REPUTATION`

レート制限

URL、セッションクッキー、または送信元IPアドレスごとにレート制限ポリシーを追加します。

属性	説明
`url-pattern`	レート制限するURLパターン。
`session-cookie`	レート制限するセッションクッキー名。
`sourceip`	送信元IPアドレスでレート制限するには、`true`に設定します。
`requests-per-second`	1秒あたりの最大リクエスト数。
`action`	実行するアクション: `drop`、`log`、`reset`。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`rate-limit`	`botprofile`	`bot_enable_rate_limit`	`ON`
`url-pattern`	`botprofile_ratelimit_binding`	`bot_rate_limit_type`, `bot_rate_limit_url`	`URL`, `/api/v1/*`
`session-cookie`	`botprofile_ratelimit_binding`	`bot_rate_limit_type`,	`SESSION`, `JSESSIONID`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
		`bot_rate_limit_url`	
`sourceip`	`botprofile_ratelimit_binding`	`bot_rate_limit_type`	`SOURCE_IP`
`requests-per-second`	`botprofile_ratelimit_binding`	`rate`, `bot_rate_limit_time_slice`	`100`, `1000`
`action`	`botprofile_ratelimit_binding`	`bot_rate_limit_action`, `bot_rate_limit_enabled`	`[LOG, DROP]`, `ON`

ボットシグネチャ

シグネチャファイルへのURLを提供することで、ボットシグネチャ検出を有効または無効にします。

属性	説明
`bot-signature`	外部ボット署名JSONファイルへのURL。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`bot-signature`	`botprofile`	`signature`	`<signature_object_name>`

ボットトラップ

自動ボットを検出するための隠しトラップURLとして機能する適用URLを追加します。

属性	説明
`enforcement-url`	トラップとして挿入するURL。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
`action`	アクション: `block`、`log`、`redirect`。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`enforcement-url`	`botprofile_trapinsertionurl_binding`	`bot_trap_url`、`bot_trap_url_insertion_enabled`	`apache.com`、`ON`

ボットTPS

Transactions Per Second (TPS) しきい値を使用して、レートベースのボット検出を適用します。設定されたTPS制限を超えるリクエストは、ボットトラフィックとして扱われます。

属性	説明
`action`	実行するアクション: `drop`、`reset`、`none`。デフォルト: `drop`。
`enforcement-type`	TPSベースの適用タイプ: `ClientIP`、`GeoLocation`、`Host`、`SourceUrl`。デフォルト: `ClientIP`。
`minimum-tps`	適用するための最小ボットTPSしきい値。
`maximum-tps-percentage`	最大TPS増加率 (30分間隔あたり)。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`bot-tps`	`botprofile`	`bot_enable_tps`	`ON`
`action`	`botprofile_tps_binding`	`bot_tps_action`	`[LOG, DROP]`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`enforcement-type`	`botprofile_tps_binding`	`bot_tps_type`	`SOURCE_IP`
`minimum-tps`	`botprofile_tps_binding`	`threshold`	`4`
`maximum-tps-percentage`	`botprofile_tps_binding`	`percentage`	`10000`

デバイスフィンガープリント

ブラウザとデバイスの属性を収集することで、ボットと人間を区別するために、デバイスフィンガープリントを有効または無効にします。

属性	説明
`action`	アクション: `log`、`drop`、`redirect`、`reset`。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`device-fingerprint`	`botprofile`	`devicefingerprint`	`ON`
`action`	`botprofile`	`devicefingerprintaction`	`[LOG]`

WAF保護

SQLインジェクション

SQLインジェクション攻撃 (OWASP A05: インジェクション) から保護し、例外を追加するオプションを提供します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
`action`	アクション: `block`。
`exceptions`	例外のリスト。
`type`	例外コンテンツタイプ: `JSON` または `HTML`。
`url-pattern`	除外するURLパターン。
`form-field-name`	除外するフォームフィールド名。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`sql-injection`	`appfwprofile`	`SQLInjectionAction`	`[block, log, stats]`
`exceptions`	`appfwprofile_sqlinjection_binding`	`sqlinjection`, `formactionurl_sql`, `isregex_sql`	`.*`, `.*`, `REGEX`

バッファオーバーフロー

サイズ制限を適用することで、バッファオーバーフロー攻撃から保護します。

属性	説明
`action`	アクション: `block`。
`max-url-length`	URLの最大長。
`max-cookie-length`	Cookieの最大長。
`max-header-length`	ヘッダーの最大長。

JSON保護:

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
`max-container-depth`	JSONの最大コンテナ深度。
`max-document-length`	JSONの最大ドキュメント長。
`max-object-key-count`	JSONの最大オブジェクトキー数。
`max-object-key-length`	JSONの最大オブジェクトキー長。
`max-array-length`	JSONの最大配列長。
`max-string-length`	JSONの最大文字列長。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`buffer-overflow`	`appfwprofile`	`bufferOverflowAction`	`[block, log, stats]`
`max-url-length`	`appfwprofile`	`bufferOverflowMaxURLLength`	`1024`
`max-cookie-length`	`appfwprofile`	`bufferOverflowMaxCookieLength`	`4096`
`max-header-length`	`appfwprofile`	`bufferOverflowMaxHeaderLength`	`4096`
`max-container-depth`	`appfwprofile`	`JSONMaxContainerDepth`	`1`
`max-document-length`	`appfwprofile`	`JSONMaxDocumentLength`	`200000000`
`max-object-key-count`	`appfwprofile`	`JSONMaxObjectKeyCount`	`10000`
`max-object-key-length`	`appfwprofile`	`JSONMaxObjectKeyLength`	`10000`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`max-array-length`	`appfwprofile`	`JSONMaxArrayLength`	`10000`
`max-string-length`	`appfwprofile`	`JSONMaxStringLength`	`10000000`

Cookieの一貫性

Cookieの一貫性適用を有効または無効にします。

属性	説明
`action`	アクション: `block`。
`exceptions`	除外するCookie名のリスト。
`cookie-name`	例外として追加する新しいCookie名。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`cookie-consistency`	`appfwprofile`	`cookieConsistencyAction`	`[block, log, stats]`
`exceptions[*].cookie-name`	`appfwprofile_cookieconsistency_binding`	`cookieconsistency`	`session_tracker`

クロスサイトスクリプティング

クロスサイトスクリプティング攻撃 (OWASP A05: インジェクション) から保護します。

属性	説明
`action`	アクション: `block`。
`content-type`	`HTML`、`JSON`、または`all`。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
`exceptions`	例外のリスト。
`type`	例外コンテンツタイプ: `JSON` または `HTML` 。
`url-pattern`	除外するURLパターン。
`form-field-name`	除外するフォームフィールド名。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`cross-site-scripting`	`appfwprofile`	`crossSiteScriptingAction`	`[block, log, stats]`
`exceptions`	`appfwprofile_crosssitescripting_binding`	`crosssitescripting` , `formactionurl_xss` , `isregex_xss`	`.*` , `/admin/editor` , `REGEX`

コマンドインジェクション

OSコマンドインジェクション攻撃 (OWASP A05: Injection) から保護します。

属性	説明
`action`	アクション: `block` 。
`content-type`	`HTML` 、 `JSON` 、または `all` 。
`exceptions`	例外のリスト。
`type`	例外コンテンツタイプ: `JSON` または `HTML` 。
`url-pattern`	除外するURLパターン。
`form-field-name`	除外するフォームフィールド名。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`command-injection`	`appfwprofile`	`cmdInjectionAction`	`[block, log, stats]`
`exceptions`	`appfwprofile_cmdinjection_binding`	`cmdInjection`, `formactionurl_cmd`, `isregex_cmd`	`.*`, `/admin/cmd-tool`, `REGEX`

CSRF

クロスサイトリクエストフォージェリ (OWASP A01: 破損したアクセス制御) から保護します。

属性	説明
`action`	アクション: `block`。
`content-type`	`HTML`。
`exceptions`	除外するURLパターンのリスト。
`url-pattern`	除外するURLパターン。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`csrf`	`appfwprofile`	`CSRFtagAction`	`[block, log, stats]`
`exceptions[*].url-pattern`	`appfwprofile_csrftag_binding`	`csrftag`, `csrfformactionurl`	`.*`, `apache.com`

フィールド形式

入力制約を適用するために、フィールド形式の検証を有効にします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
`action`	アクション: `block`。
`field-name`	フィールド名。
`format`	フォーマットタイプ: `Integer`, `Alphabets`, `Alphanumeric`, `No HTML`, `Any`, `SSN`, `Credit Card`, `Custom`。
`customregex`	フォーマットが `Custom` の場合のカスタム正規表現。
`min-length`	最小フィールド長。
`max-length`	最大フィールド長。
`url-pattern`	適用するURLパターン範囲。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`field-format`	`appfwprofile`	`fieldFormatAction`	`[block, log, stats]`
`enforcements[*]`	`appfwprofile_fieldformat_binding`	`fieldformat`, `formactionurl_ff`, `fieldtype`	`age`, `.*`, `integer`
`min-length`, `max-length`	`appfwprofile_fieldformat_binding`	`fieldformatminlength`, `fieldformatmaxlength`	`1`, `3`
`customregex`	`appfwfieldtype`	`name`, `regex`, `priority`	`<fieldtype_name>`, `<regex>`, `2`

フィールドの整合性

フォームフィールドの改ざんから保護します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
`action`	アクション: `block`.
`content-type`	`HTML`.
`exceptions`	例外のリスト。
`url-pattern`	除外するURLパターン。
`field-name`	除外するフィールド名。

NetScaler エンティティマッピング:

CRD フィールド	NETSCALER エンティティ	NETSCALER 属性	例の値
`field-consistency`	`appfwprofile`	`fieldConsistencyAction`	`[block, log, stats]`
`exceptions[*]`	`appfwprofile_fieldconsistency_binding`	`fieldconsistency`, `formactionurl_ffc`	`.*`, `/form/dynamic`

WAF シグネチャ

シグネチャベースの検出を有効にするには、WAFシグネチャファイルへのURLを提供します。

属性	説明
`waf-signature`	WAFシグネチャXMLファイルのURL。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER属性	例の値
`waf-signature`	`appfwprofile`	`signatures`	`<signature_object_name>`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

データ漏洩防止

レスポンスでの機密データ漏洩を防ぐために、SAFEオブジェクトを設定します。

既存のSAFEオブジェクトタイプ: `SSN`、`CreditCard`

カスタムSAFEオブジェクト:

属性	説明
`type`	`SSN`、`CreditCard`、または`Custom`。
`action`	アクション: `mask`、`block`、`log`。
`max-match-length`	パターンに対する最大一致長。
`regex`	カスタム正規表現パターン (タイプが`Custom`の場合)。

NetScalerエンティティマッピング:

CRDフィールド	NETSCALERエンティティ	NETSCALER 属性	例の値
`type: SSN`	`appfwprofile_safeobject_binding`	`safeobject`, `as_expression`, `maxmatchlength`, `action`, `state`	`ssn`, `SSN`, `15`, `[log]`, `ENABLED`
`type: CreditCard`	`appfwprofile_safeobject_binding`	`safeobject`, `as_expression`, `maxmatchlength`, `action`	`creditcard`, `CreditCard`, `255`, `[block]`
`type: Custom`	`appfwprofile_safeobject_binding`	`safeobject`, `as_expression`, `maxmatchlength`, `action`	`custom`, `\b[A-Z]{2}\d{6}\b`, `8`, `[log]`

OWASP CRD をデプロイする

1. CRD定義をデプロイします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl create -f owasp-crd.yaml
```

2. OWASPポリシーをデプロイします。

```
kubectl apply -f owasp_waf_bot.yaml
```

完全な例: 組み合わせたWAFおよびボットポリシー

以下は、サポートされているすべての保護を示す完全な例です (`owasp_waf_bot.yaml`):

```
apiVersion: citrix.com/v1

kind: owasppolicy

metadata:

  name: test-bm-owasp-policy

  namespace: default

spec:

  bot:

    bot-signature: "http://1.1.1.1/crd/bot_sig.json"

    allow-and-block-list:

      rules:

      - action: allow

        expression: HTTP.REQ.URL.PATH.EQ("apache.com")

      - action: block

        expression: HTTP.REQ.URL.PATH.EQ("nginx.com")

    bot-trap:

      enforcements:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      - action: block
        enforcement-url: apache.com
      - action: log
        enforcement-url: example.com
      - action: redirect
        enforcement-url: redirect.example.com
    bot-tps:
      enforcement-type: ClientIP
      minimum-tps: 4
      maximum-tps-percentage: 10000
      action: drop
    device-fingerprint:
      action: log
    geo-blocking:
      blocked-countries:
      - CN
      - RU
    ip-reputation:
      categories:
        anonymous-or-suspicious-proxies: true
        botnets-ransomware-ccs: true
        mobile-threats: true
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        phishing: true
        scanners: true
        spam-sources: true
        tor-proxies: true
        web-exploits: true
        windows-exploits: true
    rate-limit:
      policies:
      - action: drop
        requests-per-second: 100
        url-pattern: /api/v1/*
      - action: drop
        requests-per-second: 5
        url-pattern: /login
      - action: drop
        requests-per-second: 50
        url-pattern: /search
      - action: log
        requests-per-second: 20
        session-cookie: JSESSIONID
      - action: reset
        requests-per-second: 200
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      sourceip: true
ingressclass: cic-vpx
servicenames:
- frontend
waf:
  buffer-overflow:
    action: block
    json-protection:
      max-array-length: 10000
      max-container-depth: 1
      max-document-length: 200000000
      max-object-key-count: 10000
      max-object-key-length: 10000
      max-string-length: 10000
    max-cookie-length: 4096
    max-header-length: 4096
    max-url-length: 1024
  command-injection:
    action: block
    content-type: all
    exceptions:
    - type: HTML
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      url-pattern: /admin/cmd-tool
    - type: JSON
      url-pattern: /api/script-runner
  cookie-consistency:
    action: block
    exceptions:
    - cookie-name: session_tracker
  cross-site-scripting:
    action: block
    content-type: all
    exceptions:
    - type: HTML
      url-pattern: /admin/editor
    - type: JSON
      url-pattern: /api/render
  csrf:
    action: block
    content-type: HTML
    exceptions:
    - url-pattern: apache.com
  data-leak-prevention:
    safe-objects:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      - action: mask
        type: SSN
        max-match-length: 15
      - action: block
        type: CreditCard
      - action: log
        type: Custom
        regex: "\\b[A-Z]{2}\\d{6}\\b"
        max-match-length: 8
    field-consistency:
      action: block
      content-type: HTML
      exceptions:
      - url-pattern: /form/dynamic
    field-format:
      enforcements:
      - action: block
        format: Any
      - action: block
        field-name: "email"
        format: Custom
        customregex: "^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\\.[a-zA-Z]{2,}$"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        max-length: 254
        min-length: 5
        url-pattern: /register
      - action: log
        field-name: "age"
        format: Integer
        max-length: 3
        min-length: 1
    sql-injection:
      action: block
      exceptions:
      - type: HTML
        url-pattern: .*
    waf-signature: "http://1.1.1.1/crd/sig.xml"
```

NetScaler を使用した Kubernetes の認証および認可ポリシー

認証および認可ポリシーは、アプリケーションまたはAPIサーバーによってホストされるリソースへのアクセス制限を適用するために使用されます。認証ポリシーを使用してIDを検証できますが、認可ポリシーは、指定されたリクエストがリソースにアクセスするために必要な権限を持っているかどうかを検証するために使用されます。

NetScaler は、NetScaler Ingress Controller とともに使用して NetScaler 上で認証ポリシーを定義できる、**Auth CRD** と呼ばれる Kubernetes の CustomResourceDefinition (CRD) を提供します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Auth CRD の定義

Auth CRD は、NetScaler Ingress Controller の GitHub リポジトリの auth-crd.yaml で入手できます。Auth CRD は、NetScaler 上で認証ポリシーを定義するために必要なさまざまなオプションの属性 を提供します。

Auth CRD の属性

Auth CRD は、認証ポリシーを定義するために使用する次の属性を提供します。

- `servicenames`
- `authentication_mechanism`
- `authentication_providers`
- `authentication_policies`
- `authorization_policies`

サービス名

認証および認可ポリシーを適用する必要があるサービスの名前。

認証メカニズム

以下の認証メカニズムがサポートされています。

- リクエストヘッダーの使用:リクエストヘッダーを使用したユーザー認証を有効にします。このメカニズムは、資格情報またはAPIキーがヘッダー（通常はAuthorizationヘッダー）で渡される場合に使用できます。たとえば、基本認証、ダイジェスト認証、ベアラー認証、またはAPIキーにリクエストヘッダーを使用した認証を使用できます。
- フォームの使用: このメカニズムは、OpenID Connectの依拠当事者構成やSAMLのサービスプロバイダー構成を含む、ユーザー認証またはWeb認証で使用できます。

認証メカニズムが指定されていない場合、デフォルトはリクエストヘッダーを使用した認証です。

以下は、フォームベース認証の属性です。

属性	説明
`authentication_host`	ADC認証サービスのためにユーザーがリダイレクトされる完全修飾ドメイン名 (FQDN) を指定します。このFQDNは一意であり、Ingress/サービスタイプLoadBalancerを持つNetScalerのフロントエンドIPアドレス、また

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
	はListener CRDのVIPアドレスに解決される必要があります。
`authentication_host_cert`	`authentication_host`で使用するSSL証明書の名前を指定します。この証明書は、フォームを使用して認証を実行する際に必須です。
`ingress_name`	フォームを使用した認証が適用されるIngress名を指定します。
`ingressclass`	指定されたIngressクラスに関連付けられたIngressコントローラーのみがリソースを処理するように、Ingressクラスを指定します。そうしないと、クラスター内のすべてのコントローラーがこのリソースを処理します。
`lb_service_name`	フォームを使用した認証が適用されるLoadBalancerタイプのサービス名を指定します。
`listener_name`	フォームを使用した認証が適用されるListener CRDの名前。
`vip`	フォームを使用した認証が適用されるIngressのフロントエンドIPアドレスを指定します。この属性は、Ingressで提供される`frontend-ip`アドレスを参照します。同じフロントエンドIPを使用するIngressリソースが複数ある場合は、VIPを使用することをお勧めします。

注:

- フォームを使用する場合、すべての種類のトラフィックに対して認証を有効にできます。現在、きめ細かな認証はサポートされていません。
- フォームベース認証を適用する必要があるリソースに応じて、`ingress_name`、`lb_service_name`、`listener_name`、または`vip`のいずれかの属性を使用してリソースを指定できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

認証プロバイダー

プロバイダーは、認証メカニズムに必要な認証メカニズムとパラメーターを定義します。

基本認証

HTTP基本認証スキームでローカル認証が使用されることを指定します。基本認証を使用するには、イングレスNetScalerでユーザーアカウントを作成する必要があります。

注:

ローカル認証のユーザーは、NetScaler VPXまたはNetScaler MPXで作成する必要があります。CLIを使用してNetScalerでローカル認証のユーザーを作成するには、次のコマンドを実行します: `add aaa user <username> -password <password>`。GUIを使用してNetScalerでローカル認証のユーザーを作成する方法については、ローカルユーザーの構成を参照してください。

OAUTH認証

OAuth認証メカニズムでは、oAuth2を使用してクライアントを認証し、アクセストークンを発行するために外部のIDプロバイダーを必要とします。クライアントがアクセストークンをアクセス資格情報としてNetScalerに提示すると、NetScalerは設定された値を使用してトークンを検証します。トークンの検証が成功した場合、NetScalerはクライアントにアクセスを許可します。
OAuth認証属性
OAuth認証の属性は次のとおりです。

属性	説明
`Issuer`	認証のためにトークンを受け入れる必要があるサーバーのID（通常はURL）。
`jwks_uri`	JWT（JSON Web Token）検証のためのJWK（JSON Web Key）を含むエンドポイントのURL。
`audience`	トークンが適用されるサービスまたはアプリケーションのID。
`token_in_hdr`	トークンが存在するカスタムヘッダー名。デフォルト値は`Authorization`ヘッダーです。
	注: 複数のヘッダーを指定できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
`token_in_param`	トークンが存在するクエリパラメータ。
`signature_algorithms`	許可される署名アルゴリズムのリストを指定します。デフォルトでは、HS256、RS256、およびRS512アルゴリズムが許可されています。
`introspect_url`	認証サーバー (IdP) のイントロスペクションエンドポイントのURL。提示されたアクセストークンが不透明なトークンである場合、トークン検証のためにイントロスペクションが使用されます。
`client_credentials`	認証サーバーで認証するために必要なクライアントIDとクライアントシークレットを含むKubernetesシークレットオブジェクトの名前。
`claims_to_save`	保存するクレームのリスト。クレームは認可ポリシーの作成に使用されます。

OpenID Connect (OIDC) は、OAuth 2.0 プロトコル上に構築されたシンプルなIDレイヤーです。OIDC を使用すると、クライアントは認可サーバーによって実行された認証に基づいてエンドユーザーのIDを検証したり、エンドユーザーに関する基本的なプロファイル情報を取得したりできます。OAuth 属性に加えて、OIDC を構成するために次の属性を使用できます。

属性	説明
`metadata_url`	OAUTHまたはOIDCプロバイダーのメタデータを取得するために使用されるURLを指定します。
`user_field`	トークンからユーザー名を抽出する属性を指定します。デフォルトでは、NetScaler はユーザー ID のためにメール属性を調べます。
`default_group`	認証が成功した場合にリクエストに割り当てられるグループを指定します。このグループは、トークンから抽出されたグループに追加されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
`grant_type`	トークンエンドポイントへのフローのタイプを指定します。デフォルト値は `CODE` です。
`pkce`	Proof Key for Code Exchange (PKCE) を有効にするかどうかを指定します。デフォルト値は `ENABLED` です。
`token_ep_auth_method`	トークンエンドポイントで使用する認証方法を指定します。デフォルト値は `client_secret_post` です。

SAML認証

Security Assertion Markup Language (SAML) は、製品や組織間でユーザー認証を可能にするXMLベースのオープン標準です。SAML認証メカニズムでは、クライアントを認証するために外部のIDプロバイダーが必要です。SAMLは、クライアントIDをIDプロバイダーからNetScalerに転送することで機能します。クライアントIDの検証が成功すると、NetScalerはクライアントにアクセスを許可します。

SAML認証の属性は以下のとおりです。

属性	説明
`metadata_url`	SAMLメタデータを取得するために使用されるURLを指定します。**注:** これは必須です。
`metadata_refresh_interval`	指定されたメタデータURLからメタデータをフェッチする間隔を分単位で指定します。デフォルト値は36000です。
`signing_cert`	サービスプロバイダー (SP) からIDプロバイダー (IdP) へのリクエストに署名するためのSSL証明書を指定します。このパラメーターは、事前設定されている場合は文字列、またはKubernetes TLSタイプ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
	のシークレットである場合があります。
`audience`	トークンが適用されるサービスまたはアプリケーションのIDを指定します。
`issuer_name`	SPからIdPに送信されるリクエストで使用され、NetScalerを識別するための名前を指定します。
`binding`	SAMLメッセージの転送メカニズムを指定します。デフォルト値は `POST` です。
`artifact_resolution_service_url`	IdP上のアーティファクト解決サービスのURLを指定します。
`logout_binding`	SAMLログアウトのトランスポートメカニズムを指定します。デフォルト値は `POST` です。
`reject_unsigned_assertion`	署名されていないSAMLアサーションを拒否します。この値が `ON` の場合、署名のないアサーションを拒否します。デフォルト値はONです。
`user_field`	SAMLアサーションで指定されたSAMLユーザーIDを指定します
`default_authentication_group`	抽出されたグループに加えて、認証が成功したときに選択されるデフォルトグループを指定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
`skew_time`	受信SAMLアサーションで許可されるクロックスキュー時間（分単位）を指定します。デフォルト値は5です。
`attributes_to_save`	NetScaler上のセッションのためにキーと値のペアとして抽出および保存する必要がある、コンマで区切られた属性名のリストを指定します。

LDAP認証

LDAP (Lightweight Directory Access Protocol) は、インターネットプロトコル (IP) ネットワーク上で分散ディレクトリ情報サービスにアクセスし、維持するための、オープンでベンダーに依存しない業界標準のアプリケーションプロトコルです。LDAPの一般的な用途は、ユーザー名とパスワードを保存するための一元的な場所を提供することです。LDAPを使用すると、多くの異なるアプリケーションやサービスがLDAPサーバーに接続してユーザーを検証できます。

注記:

LDAP認証は、リクエストヘッダーを使用する認証メカニズムとフォームを使用する認証メカニズムの両方でサポートされています。

以下は、LDAP認証の属性です。

属性	説明
`server_ip`	LDAPサーバーに割り当てられたIPアドレスを指定します。
`server_name`	FQDNとしてのLDAPサーバー名を指定します。
`server_port`	LDAPサーバーが接続を受け入れるポートを指定します。デフォルト値は389です。
`base`	LDAP検索を開始するベースノードを指定します。LDAPサーバーがローカルで実行されてい

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
	る場合、ベースのデフォルト値は `dc=netscaler`、`dc=com`です。
`server_login_credentials`	LDAPサーバーにログインするための資格情報を提供するKubernetesシークレットオブジェクトを指定します。シークレットデータにはユーザー名とパスワードが含まれている必要があります。
`login_name`	「**LDAPログイン名**」属性を指定します。NetScalerは、外部LDAPサーバーまたはActive Directoryを照会するためにLDAPログイン名を使用します。
`security_type`	NetScalerとLDAPサーバー間の通信に使用されるセキュリティの種類を指定します。デフォルトはTLSです。
`validate_server_cert`	LDAPサーバー証明書を検証します。デフォルト値は`NO`です。
`hostname`	LDAPサーバーのホスト名を指定します。`validate_server_cert`が`ON`の場合、この値はLDAPからの証明書上のホスト名である必要があります。ホスト名が一致しないと、接続エラーが発生します。
`sub_attribute_name`	LDAPグループのサブ属性名を指定します。この属性は、LDAPサーバーからのグループ抽出に使用されます。
`group_attribute_name`	LDAPグループの属性名を指定します。この属性は、LDAPサーバーでのグループ抽出に使用されます。
`search_filter`	デフォルトのLDAPユーザー検索文字列と組み合わせて検索値を形成する文字列を指定します。たとえば、検索フィルター「"vpnallowed=true"」がLDAPログイン名「"samaccount"」とユーザーが指定したユーザー名「"bob"」と組み合わされた場合、結果はLDAP検索文字列「"(&(vpnallowed=true)

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
	(samaccount=bob))"」になります。検索文字列は二重引用符で二重に囲んでください。
`auth_timeout`	NetScalerがサーバーからの応答を待つ秒数を指定します。デフォルト値は3です。
`password_change`	パスワード変更要求を許可します。デフォルト値は`DISABLED`です。
`attributes_to_save`	LDAPサーバーから取得し、NetScaler上のセッションのキーと値のペアとして保存する必要がある、コンマで区切られた属性名のリスト。

認証ポリシー

authentication_policiesを使用すると、認証メカニズムを適用するためのトラフィック選択基準を定義し、選択したトラフィックに使用するプロバイダーを指定できます。

認証ポリシーは、認証ルールを指定できる2つの形式をサポートしています。

- リソース形式
- 式形式

リソース形式のポリシーの属性は次のとおりです。

属性	説明
`path`	特定のAPIエンドポイントを参照するURLパスプレフィックスの配列。例: `/api/v1/products/`。
`method`	HTTPメソッドの配列。許可される値はGET、PUT、POST、またはDELETEです。受信リクエストURIが、いずれかのパスおよびリストされたいずれかのメソッドと一致する場合にトラフィックが選択されます。メソッドが指定されていない場合、パスのみがトラフィック選択基準として使用されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
`provider`	使用する必要がある認証メカニズムを指定します。認証メカニズムが提供されていない場合、認証は実行されません。

以下の属性は、式形式の認証ポリシー用です。

属性	説明
`expression`	認証に基づいて評価されるNetScaler式を指定します。
`provider`	使用する必要がある認証メカニズムを指定します。認証メカニズムが提供されていない場合、認証は実行されません。

注:

特定のエンドポイントの認証をスキップしたい場合は、`provider` 属性を空のリストとして設定したポリシーを作成します。そうしないと、リクエストは拒否されます。

承認ポリシー

承認ポリシーを使用すると、選択したトラフィックに承認要件を適用するためのトラフィック選択基準を定義できます。

承認ポリシーは、承認ルールを指定できる2つの形式をサポートしています。

- リソース形式
- 式形式

以下は、リソース形式の承認ポリシーの属性です。

属性	説明
`path`	特定のAPIエンドポイントを参照するURLパスプレフィックスの配列です。例えば、`/api/v1/products/`。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
`method`	HTTPメソッドの配列です。許可される値はGET、PUT、POST、またはDELETEです。
`claims`	特定のAPIエンドポイントにアクセスするために必要なクレームを指定します。`name`はクレーム名を示し、`values`は必要な権限を示します。複数のクレームを持つことができます。空のリストが指定された場合、承認は不要であることを意味します。
	注: 承認に使用する必要があるクレームは、認証の一部として保存する必要があります。

以下は、式形式の承認ポリシーの属性です。

属性	説明
`expression`	認可のために評価される式を指定します。

注意:

NetScalerは、APIトラフィックに対して認証ポリシーと認可ポリシーの両方を必要とします。したがって、認証ポリシーとともに認可ポリシーを設定する必要があります。認可チェックがない場合でも、空のクレームを持つ認可ポリシーを作成する必要があります。そうしないと、リクエストは403エラーで拒否されます。

注意:
受信リクエストがポリシー（パス、メソッド、クレーム）に一致する場合、認可は成功します。一致するまで、すべてのポリシーが試行されます。特定のエンドポイントの認可を選択的にバイパスする必要がある場合は、明示的なポリシーを作成する必要があります。

Auth CRD のデプロイ

Auth CRD をデプロイするには、以下を実行します。

1. CRD (auth-crd.yaml) をダウンロードします。
2. 次のコマンドを使用して Auth CRD をデプロイします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl create -f auth-crd.yaml
```

例:

```
root@master:~# kubectl create -f auth-crd.yaml



customresourcedefinition.apiextensions.k8s.io/authpolicies.citrix.com
created
```

認証ポリシーと認可ポリシーの記述方法

KubernetesクラスターにNetScalerが提供するCRDをデプロイした後、`.yaml`ファイルで認証ポリシー構成を定義できます。`.yaml`ファイルでは、`kind`フィールドで`authpolicy`を使用し、`spec`セクションでポリシー構成の要件に基づいて**Auth CRD**属性を追加します。

`.yaml`ファイルをデプロイすると、NetScaler Ingress ControllerはNetScalerに認証ポリシー構成を適用します。

ローカル認証プロバイダー

以下は、local-auth-provider タイプの認証および認可ポリシー定義のサンプルです(local_auth.yaml)。

```
apiVersion: citrix.com/v1beta1

kind: authpolicy

metadata:

  name: authexample

spec:

    servicenames:

    - frontend



    authentication_providers:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        - name: "local-auth-provider"
          basic_local_db:
              use_local_auth: 'YES'


    authentication_policies:
        - resource:
            path:
              - '/orders/'
              - '/shipping/'
            method: [GET, POST]
          provider: ["local-auth-provider"]


        # skip authentication for this
        - resource:
            path:
              - '/products/'
            method: [GET]
          provider: []


    authorization_policies:
        # skip authorization
        - resource:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            path: []

            method: []

            claims: []
```

サンプルのポリシー定義は以下を実行します。

- NetScaler は、以下のリクエストに対してローカル認証を実行します。
  - orders および shipping エンドポイントに対する **GET** または **POST** 操作。
- NetScaler は、**products** エンドポイントに対する **GET** 操作の認証を実行しません。
- NetScaler は、いかなる認可権限も適用しません。

注:

ローカル認証のユーザーは、NetScaler VPX または NetScaler MPX 上で作成する必要があります。CLI を使用して NetScaler 上でローカル認証のユーザーを作成するには、次のコマンドを実行します: `add aaa user <username> -password <password>`。GUI を使用して NetScaler 上でローカル認証のユーザーを作成する方法については、「ローカルユーザーの構成」を参照してください。

oAuth JWT 検証

以下は、oAuth JWT 検証の認証および認可ポリシー定義のサンプルです (oauth_jwt_auth.yaml)。

```
apiVersion: citrix.com/v1beta1

kind: authpolicy

metadata:

  name: authexample

spec:

    servicenames:

    - frontend
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    authentication_providers:
      - name: "jwt-auth-provider"
        oauth:
          issuer: "https://sts.windows.net/tenant1/"
          jwks_uri: "https://login.microsoftonline.com/tenant1/discovery/
v2.0/keys"
          audience : ["https://api.service.net"]
          claims_to_save : ["scope"]


    authentication_policies:
        - resource:
            path:
              - '/orders/'
              - '/shipping/'
            method: [GET, POST]
          provider: ["jwt-auth-provider"]


        # skip authentication for this
        - resource:
            path:
              - '/products/'
            method: [GET]
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      provider: []


authorization_policies:
    - resource:
        path:
          - '/orders/'
          - '/shipping/'
        method: [POST]
        claims:
          - name: "scope"
            values: ["read", "write"]
    - resource:
        path:
          - '/orders/'
        method: [GET]
        claims:
          - name: "scope"
            values: ["read"]
    # skip authorization, no claims required
    - resource:
        path:
          - '/shipping/'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            method: [GET]

            claims: []
```

サンプルのポリシー定義は以下を実行します。

- NetScaler は、以下のリクエストに対して JWT 検証を実行します。
  - **orders** および **shipping** エンドポイントに対する **GET** または **POST** 操作。
- NetScaler は、**products** エンドポイントに対する **GET** 操作の認証をスキップします。
- NetScaler は、**orders** および **shipping** エンドポイントでの **POST** 操作に対して、`read` および `write` 権限を持つスコープクレームを必要とします。
- NetScaler は、**orders** エンドポイントでの **GET** 操作に対して、読み取り権限を持つスコープクレームを必要とします。
- NetScaler は、**shipping** エンドポイントでの **GET** 操作に対して、いかなる権限も必要としません。

OAuth の場合、トークンがカスタムヘッダーに存在する場合、`token_in_hdr` 属性を使用して次のように指定できます。

```
oauth:

  issuer: "https://sts.windows.net/tenant1/"

  jwks_uri: "https://login.microsoftonline.com/tenant1/discovery/v2.0/
keys"

  audience : ["https://vault.azure.net"]

  token_in_hdr : [“custom-hdr1”]
```

同様に、トークンがクエリパラメータに存在する場合、`token_in_param` 属性を使用して次のように指定できます。

```
oauth:

  issuer: "https://sts.windows.net/tenant1/"

  jwks_uri: "https://login.microsoftonline.com/tenant1/discovery/v2.0keys"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
audience : ["https://vault.azure.net"]

token_in_param : [“query-param1”]
```

OAuth イントロスペクション

以下は、OAuth JWT 検証の認証および認可ポリシー定義のサンプルです。
(oauth_intro_auth.yaml)

```
apiVersion: citrix.com/v1beta1

kind: authpolicy

metadata:

  name: authexample

spec:

    servicenames:

    - frontend



    authentication_providers:

        - name: "introspect-provider"

          oauth:

            issuer: "ns-idp"

            jwks_uri: "https://idp.aaa/oauth/idp/certs"

            audience : ["https://api.service.net"]

            client_credentials: "oauthsecret"

            introspect_url: https://idp.aaa/oauth/idp/introspect

            claims_to_save : ["scope"]
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
authentication_policies:
    - resource:
        path: []
        method: []
      provider: ["introspect-provider"]

authorization_policies:
    - resource:
        path: []
        method: [POST]
        claims:
        - name: "scope"
          values: ["read", "write"]
    - resource:
        path: []
        method: [GET]
        claims:
        - name: "scope"
          values: ["read"]
```

サンプルポリシー定義は以下を実行します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- NetScaler は、すべてのリクエストに対して、プロバイダー `introspect-provider` で指定された OAuth イントロスペクションを実行します。
- NetScaler は、すべての **POST** リクエストに対して、`read` および `write` 権限を持つスコープクレームを必要とします。
- NetScaler は、すべての **GET** リクエストに対して、読み取り権限を持つスコープクレームを必要とします。

イントロスペクション用のクライアント資格情報を含むシークレットオブジェクトの作成

OAuth イントロスペクションを設定するには、Kubernetes シークレットオブジェクトが必要です。 以下の例に示すように、シークレットオブジェクトを作成できます。

```
apiVersion: v1

kind: Secret

metadata:

  name: oauthsecret

type: Opaque

stringData:

 client_id: "nsintro"

 client_secret: "nssintro"
```

注:

不透明なシークレットオブジェクトのキーは `client_id` と `client_secret` である必要があります。ユーザーは必要に応じてそれらの値を設定できます。

フォームを使用したSAML認証

以下は、フォームを使用したSAML認証の例です。この例では、`authhost-tls-cert-secret`と`saml-tls-cert-secret`は、証明書とキーを参照するKubernetes TLSシークレットです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

`certkey.cert` と `certkey.key` がそれぞれ認証ホストの証明書とキーである場合、`authhost-tls-cert-secret` は次のコマンドを使用して作成できます。

```
kubectl create secret tls authhost-tls-cert-secret --key="certkey.key" --
cert="certkey.cert
```

同様に、このコマンドを使用して、必要な証明書とキーで `saml-tls-cert-secret` を形成できます。

```
apiVersion: citrix.com/v1beta1

kind: authpolicy

metadata:

  name: samlexample

spec:

    servicenames:

    - frontend


    authentication_mechanism:

      using_forms:

        authentication_host: "fqdn_authenticaton_host"

        authentication_host_cert:

          tls_secret: authhost-tls-cert-secret

        listener_name: “example-listener”
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
authentication_providers:
    - name: "saml-auth-provider"
      saml:
          metadata_url: "https://idp.aaa/metadata/samlidp/aaa"
          signing_cert:
              tls_secret: saml-tls-cert-secret

authentication_policies:

    - resource:
        path: []
        method: []
      provider: ["saml-auth-provider"]

authorization_policies:

    - resource:
        path: []
        method: []
        claims: []
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サンプルポリシー定義は以下を実行します。

- NetScalerは、すべてのリクエストに対して、プロバイダー`saml-auth-provider`で指定されたSAML認証を実行します。 **注:** フォームメカニズムでは、きめ細かな認証はサポートされていません。
- NetScalerは、すべての**POST**リクエストに対して、`admin`権限を持つグループクレームを要求します。
- NetScalerは、**GET**リクエストに対して特定の権限を要求しません。

フォームを使用したOpenID Connect認証

以下は、外部IDプロバイダーのユーザーを認証するために、NetScalerをリレーイングパーティ（RP）ロールで構成するOpenID Connect認証を作成する例です。OpenID Connectの手順をトリガーするには、`authentication_mechanism`を`using_forms`に設定する必要があります。

```
apiVersion: citrix.com/v1beta1

kind: authpolicy

metadata:

  name: authoidc

spec:

    servicenames:

    - frontend

    authentication_mechanism:

        using_forms:

            authentication_host: "10.221.35.213"

            authentication_host_cert:

                 tls_secret: "oidc-tls-secret"

            ingress_name:  “example-ingress”
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    authentication_providers:



        - name: "oidc-provider"

          oauth:

            audience : ["https://app1.citrix.com"]

            client_credentials: "oidcsecret"

            metadata_url: "https://10.221.35.214/oauth/idp/.well-known/
openid-configuration"

            default_group: "groupA"

            user_field: "sub"

            pkce: "ENABLED"

            token_ep_auth_method: "client_secret_post"



    authentication_policies:



        - resource:

            path: []

            method: []

          provider: ["oidc-provider"]



    authorization_policies:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        #default - no authorization requirements
        - resource:
            path: []
            method: []
            claims: []
```

サンプルポリシー定義は以下を実行します。

- NetScalerは、すべてのリクエストに対して、プロバイダー`oidc-provider`で指定されたOIDC認証（リレーイングパーティ）を実行します。

  注: フォームメカニズムでは、きめ細かな認証はサポートされていません。
- NetScalerは、いかなる承認権限も要求しません。

リクエストヘッダーを使用したLDAP認証

以下は、リクエストヘッダーを使用したLDAP認証の例です。

この例では、`ldapcredential`はLDAPサーバーの資格情報を参照するKubernetesシークレットです。LDAPサーバーの資格情報の作成方法については、`ldap_secret.yaml`ファイルを参照してください。

```
apiVersion: citrix.com/v1beta1
kind: authpolicy
metadata:
  name: ldapexample
spec:
    servicenames:
    - frontend
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
authentication_providers:
    - name: "ldap-auth-provider"
      ldap:
          server_ip: "192.2.156.160"
          base: 'dc=aaa,dc=local'
          login_name: accountname
          sub_attribute_name: CN
          server_login_credentials: ldapcredential

    - name: "local-auth-provider"
      basic_local_db:
          use_local_auth: 'YES'

authentication_policies:

    - resource:
        path: []
        method: []
      provider: ["ldap-auth-provider"]
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    authorization_policies:



        - resource:

            path: []

            method: []

            claims: []
```

注: リクエストヘッダーベースの認証メカニズムでは、トラフィックに基づいたきめ細かな認証がサポートされています。

フォームを使用したLDAP認証

この例では、`authhost-tls-cert-secret` は証明書とキーを参照するKubernetes TLSシークレットです。

`certkey.cert` と `certkey.key` がそれぞれ認証ホストの証明書とキーである場合、`authhost-tls-cert-secret` は次の コマンドを使用して形成できます。

```
kubectl create secret tls authhost-tls-cert-secret --key="certkey.key" --
cert="certkey.cert
```

この例では、`ldapcredential` はLDAPサーバーの資格情報を参照するKubernetesシークレットです。LDAPサーバーの資格情報の作成方法については、`ldap_secret.yaml` ファイルを参照してください。

```
apiVersion: citrix.com/v1beta1

kind: authpolicy

metadata:

  name: ldapexample

spec:

    servicenames:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    - frontend


    authentication_mechanism:
      using_forms:
        authentication_host: "fqdn_authenticaton_host"
        authentication_host_cert:
          tls_secret: authhost-tls-cert-secret
        vip: "192.2.156.156"


    authentication_providers:
        - name: "ldap-auth-provider"
          ldap:
              server_ip: "192.2.156.160"
              base: 'dc=aaa,dc=local'
              login_name: accountname
              sub_attribute_name: CN
              server_login_credentials: ldapcredential


    authentication_policies:


        - resource:
            path: []
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            method: []

          provider: ["ldap-auth-provider"]


    authorization_policies:


        - resource:

            path: []

            method: []

            claims: []
```

サンプルポリシー定義は以下を実行します。

- NetScalerは、すべてのトラフィック（すべてのリクエスト）に対してLDAP認証を実行します。
- NetScalerは、いかなる承認権限も適用しません。

以下は、`LDAP_secret.yaml` の例です。

```
apiVersion: v1

kind: Secret

metadata:

  name: ldapcredential

type: Opaque

stringData:

  username: 'ldap_server_username'

  password: 'ldap_server_password'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Auth CRDでのNetScaler式サポートの例

この例では、認証および承認ポリシーとともにNetScaler式を指定する方法を示します。

```
apiVersion: citrix.com/v1beta1

kind: authpolicy

metadata:

  name: authexample

spec:

    servicenames:

    - frontend


    authentication_mechanism:

      using_request_header: 'ON'


    authentication_providers:

        - name: "ldap-auth-provider"

          ldap:

              server_ip: "192.2.156.160"

              base: 'dc=aaa,dc=local'

              login_name: accountname

              sub_attribute_name: CN

              server_login_credentials: ldapcredential
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
                # "memberof" attribute details are extracted from LDAP
server.

                attributes_to_save: memberof



      authentication_policies:

          # Perform LDAP authentication for the host
hotdrink.beverages.com

          - expression:
'HTTP.REQ.HOSTNAME.SET_TEXT_MODE(IGNORECASE).EQ("hotdrink.beverages.com")'

            provider: ["ldap-auth-provider"]




      authorization_policies:

          # ALLOW the session only if the authenticated user is associated
with attribute "memberof" having value "grp4"

          - expression: 'aaa.user.attribute("memberof").contains("grp4")'
```

NetScalerを使用したKubernetesでのレート制限

Kubernetesデプロイメントでは、イングレスNetScalerが提供するレート制限機能を使用して、バックエンドサーバーまたはサービス上のリソースへのリクエストをレート制限できます。

NetScalerは、**レート制限CRD**と呼ばれるKubernetesのCustomResourceDefinitions (CRD) を提供します。これはNetScaler Ingress Controllerと組み合わせて使用して、イングレスデバイスとして使用されるNetScaler上でレート制限設定を構成できます。

Kubernetes環境内のサービスへのリクエストをレート制限するだけでなく、APIセキュリティのためにもレート制限CRDを使用できます。レート制限CRDを使用すると、APIサーバーまたはAPIサーバー上の特定のAPIエンドポイントへのREST APIリクエストを制限できます。これは、API

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サーバーまたはエンドポイントへのリクエストを、タイムスライスあたりの許可された制限に対して監視および追跡し、DDoS攻撃などの攻撃から保護します。

レート制限CRDで可観測性のためのロギングを有効にできます。ログはNetScalerに保存され、シェルコマンドを使用してログを確認することで表示できます。ファイルの場所はsyslog設定に基づいています。例: `/var/logs/ns.log`。

レート制限CRDの定義

レート制限CRDの仕様は、NetScaler Ingress ControllerのGitHubリポジトリのratelimit-crd.yamlで入手できます。**レート制限CRDは**、APIゲートウェイとして機能するイングレスNetScaler上でレート制限ポリシーを定義するために必要なさまざまなオプションの属性を提供します。

レート制限CRDの属性

注:

NetScaler Ingress Controller 4.2.26以降のリリースでは、`ratelimit` CRDスキーマの属性は、`ratelimits`仕様の下に更新されました。また、新しいレート制限属性が導入されました。以前のリリースでは、`ratelimit` CRDスキーマは`spec`の下に直接定義されていました。

次の表に、レート制限CRDで提供されるさまざまな属性を示します。

CRD属性	説明
`ingressclass`	イングレスクラスを指定します。これにより、指定されたイングレスクラスに関連付けられたイングレスコントローラーのみがリソースを処理します。そうでない場合、クラスター内のすべてのコントローラーがこのリソースを処理します。
`servicenames`	レート制限ポリシーを適用するKubernetesサービスのリスト。
`selector_keys`	スロットリングが適用され監視されるAPIリクエストを識別するためにトラフィックをフィルタリングするトラフィックセレクターキー。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD属性	説明
	注: `selector_keys` はオプションの属性です。セレクターキーを0個、1個、または複数設定できます。複数のセレクターキーが設定されている場合、それらは論理AND式として扱われます。
	このバージョンのレート制限CRDでは、`selector_keys` は `basic` 設定セクションを提供し、設定された制限が監視およびスロットリングされるキーとして、以下の一般的に使用されるトラフィック特性を設定できます。
	path: 特定のAPIエンドポイントを参照するURLパスプレフィックスの配列。例: `/api/v1/products/`。
	method: HTTPメソッドの配列。許可される値は、`GET`、`PUT`、`POST`、`DELETE`、`HEAD`、`OPTIONS`、`TRACE`、`CONNECT`、`PATCH`、または `UNKNOWN_METHOD` です。
	header_name: 一意のAPIクライアントまたはユーザー識別子を持つHTTPヘッダー。例: `X-apikey`。これは、リクエストを送信するAPIクライアントを識別する一意のAPIキーとともに提供されます。
	per_client_ip: 一意のクライアントIPアドレスごとに受信される各APIリクエストに対して、設定されたしきい値を監視および適用できます。
`ratelimits`	制限と、制限を超過した場合に実行するアクションを定義するレート制限設定のリスト。この属性は、制限が適用されるトラフィックのみを選択する `selector_keys` とは別です。各エントリでサポートされる属性については、ratelimits attributes を参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD属性	説明
`streams`	可観測性のためのトラフィック分析と追跡を収集するために使用されるストリーム識別子設定のリスト。各ストリーム識別子でサポートされる属性については、Stream identifier attributes を参照してください。

注:

`ratelimits` または `streams` の少なくともいずれか1つを指定する必要があります。

ratelimits 属性

次の表に、`ratelimits` リストの各エントリで構成できる属性を示します。`req_threshold` 属性は必須です。

CRD 属性	説明
`req_threshold`	指定されたタイムスライスで許可されるリクエストの最大数（リクエストレート）。この属性は必須です。
`timeslice`	設定された制限に対してリクエストが監視される、ミリ秒単位（10の倍数）で指定された時間間隔。指定しない場合、デフォルトは1000ミリ秒です。
`limittype`	制限を適用するために使用するスロットリングアルゴリズムのタイプを設定できます。サポートされているアルゴリズムは `BURSTY` と `SMOOTH` です。デフォルトは `SMOOTH` です。
`mode`	制限が適用されるメトリック。許可される値は `REQUEST_RATE`、`CONNECTION`、および `TOKEN_RATE` です。AI バックエンドのトークンベースのレート制限を適用するには、`TOKEN_RATE` を使用します。
`alertsintimeslice`	タイムスライス内で発生させるアラートの数。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD 属性	説明
`throttle_action`	設定されたしきい値を超過してスロットルされたトラフィックに対して実行する必要があるスロットルアクションを定義できます。
	定義できるスロットルアクションは次のとおりです。
	DROP: 設定されたトラフィック制限を超過したリクエストをドロップします。
	RESET: 設定された制限を超過したリクエストの接続をリセットします。
	REDIRECT: トラフィックを設定された `redirect_url` にリダイレクトします。
	RESPOND: 標準の「***429 Too many requests***」応答で応答します。
	NOOP: 設定された制限を超過したリクエストに対してアクションを実行しません。
`redirect_url`	この属性はオプションの属性であり、`throttle_action` が値 `REDIRECT` で設定されている場合にのみ必要です。
`logpackets`	監査ログを有効にします。これには `logexpression` と `loglevel` の属性が含まれており、`logpackets` が使用される場合、これら両方が必須です。
`logexpression`	ログメッセージの形式と内容を定義するデフォルト構文式を指定します。
`loglevel`	生成されるログメッセージの重大度レベルを指定します。

ストリーム識別子属性

次の表に、`streams` リストの各エントリで構成できる属性を示します。`sort` 属性は必須です。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD属性	説明
`interval`	リクエスト数、帯域幅、応答時間などのセッション統計を計算する際に使用するデータの分数。サポートされる範囲は1〜10080です。
`sampleCount`	評価のためにリクエストを選択するサンプルのサイズ。すべてのリクエストを評価するには、サンプル数を1に設定します。サポートされる範囲は1〜65535です。
`sort`	指定された統計列で、保存されたレコードを降順にソートします。許可される値は、`REQUESTS` (デフォルト)、`CONNECTIONS`、`RESPTIME`、`BANDWIDTH`、`RESPTIME_BREACHES`、`TOKENS`、および`NONE` です。この属性は必須です。
`snmpTrap`	ストリーム識別子のSNMPトラップを有効または無効にします。許可される値は `ENABLED` と `DISABLED` です。
`appflowLog`	ストリーム識別子のAppFlow®ロギングを有効または無効にします。許可される値は `ENABLED` と `DISABLED` です。
`trackAckOnlyPackets`	ACKのみのパケットを追跡します。パケットレート制限が使用されている場合にのみ適用されます。許可される値は`ENABLED`と`DISABLED`です。
`trackTransactions`	設定されたしきい値を超えるトランザクションを追跡します。許可される値は`RESPTIME`、`TOKENS`、および`NONE`です。`TOKENS`に設定されている場合、トランザクションしきい値属性は適用されません。
`maxTransactionThreshold`	追跡されるメトリックのトランザクションごとの最大値。最小値は0です。
`minTransactionThreshold`	追跡されるメトリックのトランザクションごとの最小値。最小値は0です。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD属性	説明
`acceptanceThreshold`	違反しないトランザクションの総トランザクションに対する比率をパーセンテージで表したもの。最大6桁の小数点以下がサポートされています。
`breachThreshold`	間隔で計算された違反トランザクションのしきい値。最小値は0です。
`log`	識別子で収集されたオブジェクトがログに記録される場所。許可される値は`SYSLOG`と`NONE`です。
`logInterval`	収集されたオブジェクトをログに記録する時間間隔（分）。ストリーム識別子の間隔以上である必要があります。サポートされる範囲は1〜10080です。
`logLimit`	ログ間隔でログに記録されるオブジェクトの最大数。サポートされる範囲は1〜1000です。

レート制限CRDを展開する

レート制限CRDを展開するには、以下を実行します。

1. CRD (ratelimit-crd.yaml) をダウンロードします。
2. 次のコマンドを使用してレート制限CRDを展開します。

```
kubectl create -f ratelimit-crd.yaml
```

例：

```
root@master:~# kubectl create -f ratelimit-crd.yaml



customresourcedefinition.apiextensions.k8s.io/ratelimits.citrix.com
created
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
root@master:~# kubectl get crd



NAME CREATED AT

ratelimits.citrix.com 2019-08-27T01:06:30Z
```

レートベースのポリシー構成の記述方法

KubernetesクラスターにNetScalerが提供するCRDを展開した後、レートベースのポリシー構成を`.yaml`ファイルで定義できます。`.yaml`ファイルでは、`kind`フィールドで`ratelimit`を使用し、`spec`セクションでポリシー構成の要件に基づいてレート制限CRD属性を追加します。

`.yaml`ファイルを展開すると、NetScaler Ingress ControllerはIngress NetScalerデバイスにレートベースのポリシー構成を適用します。

以下は、レート制限ポリシー構成のいくつかの例です。

構成されたAPIエンドポイントプレフィックスへのAPIリクエストを制限する

NetScalerで、各一意のクライアントIPアドレスから構成されたAPIエンドポイントプレフィックスへのAPIリクエストを1分あたり15リクエストに制限するレートベースのポリシーを定義したいシナリオを考えてみましょう。`ratelimit-example1.yaml`という名前の`.yaml`ファイルを作成し、適切なCRD属性を使用してレートベースのポリシーを次のように定義します。

```
apiVersion: citrix.com/v1beta1

kind: ratelimit

metadata:

  name: throttle-req-per-clientip

spec:

  servicenames:

    - frontend

  selector_keys:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
   basic:
    path:
     - "/api/v1/products"
     - "/api/v1/orders/"
    per_client_ip: true
  ratelimits:
    - req_threshold: 15
      timeslice: 60000
      throttle_action: "RESPOND"
      logpackets:
        logexpression: "http.req.url"
        loglevel: "INFORMATIONAL"
```

注：

異なるレート制限設定を必要とする異なるパスに対して、複数のKubernetesオブジェクトを起動できます。

ポリシー設定を定義したら、次のコマンドを使用して`.yaml`ファイルをデプロイします。

```
root@master:~#kubectl create -f ratelimit-example1.yaml

ratelimit.citrix.com/throttle-req-per-clientip created
```

NetScaler Ingress Controllerは、Ingress NetScalerデバイスにポリシー設定を適用します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

カレンダーAPIへのAPIリクエストを制限する

NetScalerでレートベースのポリシーを定義し、HTTPヘッダー`X-API-Key`を使用して識別される各APIクライアントからのAPIリクエスト（GETまたはPOST）をカレンダーAPIに対して5つに制限するシナリオを考えてみましょう。`ratelimit-example2.yaml`という名前の`.yaml`ファイルを作成し、適切なCRD属性を使用してレートベースのポリシーを次のように定義します。

```
apiVersion: citrix.com/v1beta1

kind: ratelimit

metadata:

  name: throttle-calendarapi-perapikey

spec:

  servicenames:

    - frontend

  selector_keys:

    basic:

      path:

        - "/api/v1/calender"

      method:

        - "GET"

        - "POST"

      header_name: "X-API-Key"

  ratelimits:

    - req_threshold: 5

      throttle_action: "RESPOND"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      logpackets:

        logexpression: "rate exceeded, you may want to configure higher
limit"

        loglevel: "INFORMATIONAL"
```

ポリシー設定を定義したら、次のコマンドを使用して`.yaml`ファイルをデプロイします。

```
root@master:~#kubectl create -f ratelimit-example2.yaml

ratelimit.citrix.com/throttle-req-per-clientip created
```

NetScaler Ingress Controllerは、Ingress NetScalerデバイスにポリシー設定を適用します。

Kubernetesでリライトおよびレスポンダーポリシーを使用する

Kubernetes環境で、次のようなシナリオを処理するために特定のレイヤー7ポリシーを展開するには：

- HTTPトラフィックを特定のURLにリダイレクトする
- DDoS攻撃を軽減するために一連のIPアドレスをブロックする
- HTTPからHTTPSへの強制

マイクロサービス内に適切なライブラリを追加し、ポリシーを手動で構成する必要があります。代わりに、Ingress NetScalerデバイスが提供するリライトおよびレスポンダー機能を使用して、これらのポリシーを展開できます。

NetScalerは、NetScaler Ingress Controllerと連携して、Ingressデバイスとして使用されるNetScaler上のこれらのポリシーの構成と展開を自動化するために使用できるKubernetesのカスタムリソース定義 (CRD) を提供します。

NetScalerが提供するリライトおよびレスポンダーCRDは、最前線のNetScalerで使用される一連のツールを公開するように設計されています。これらの機能を使用すると、イングレスおよびエグレスHTTPトラフィックのヘッダーとペイロードを書き換えたり、マイクロサービスに代わってHTTPトラフィックに応答したりできます。

Kubernetesクラスターにリライトおよびレスポンダーを展開すると、データセット、パターンセット、文字列マップを使用して広範なリライトおよびレスポンダーポリシーを定義し、イングレスデバイスの統計のために監査ログを有効にすることができます。NetScaler ADCが提供する

© 1997–2026 Citrix Systems, Inc. All rights reserved.

リライトおよびレスポンダーポリシー機能の詳細については、リライトポリシーおよびレスポンダーポリシーを参照してください。

注：

リライトおよびレスポンダーCRDはOpenShiftルートではサポートされていません。リライトおよびレスポンダーCRDを使用するには、OpenShiftイングレスを使用できます。

NetScaler®リライトおよびレスポンダーCRDを展開する

NetScalerリライトおよびレスポンダーCRD展開YAMLファイル：rewrite-responder-policies-deployment.yaml。

注：

展開YAMLファイルを変更しないようにしてください。

次のコマンドを使用して、CRD をデプロイします。

```
kubectl create -f rewrite-responder-policies-deployment.yaml
```

例:

```
root@master:~# kubectl create -f rewrite-responder-policies-
deployment.yaml

customresourcedefinition.apiextensions.k8s.io/rewritepolicies.citrix.com
created
```

リライトおよびレスポンダー CRD 属性

CRD は、リライトおよびレスポンダーポリシーを定義するために必要なさまざまなオプションの属性を提供します。また、リライトおよびレスポンダーポリシー内で使用するデータセット、パターンセット、文字列マップ、および監査ログの属性も提供します。これらの CRD 属性は、それぞれ **NetScaler** コマンドおよび属性に対応しています。

リライトポリシー

次の表に、リライトポリシーを定義するために使用できる **CRD** 属性を示します。また、この表には対応する NetScaler コマンドと属性も示されています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD 属性	NETSCALER コマンド	NETSCALER 属性
リライト条件	リライトポリシーの追加	ルール
デフォルトアクション	リライトポリシーの追加	undefAction
操作	リライトアクションの追加	タイプ
ターゲット	リライトアクションの追加	ターゲット
変更式	リライトアクションの追加	stringBuilderExpr
複数回変更	リライトアクションの追加	検索
追加の複数回変更	リライトアクションの追加	検索の絞り込み
方向	LB仮想サーバーをバインド	タイプ

レスポンダーポリシー

次の表に、レスポンダーポリシーを定義するために使用できる**CRD**属性を示します。また、この表には対応するNetScalerコマンドと属性も記載されています。

CRD属性	NETSCALERコマンド	NETSCALER属性
リダイレクト	レスポンダーアクションを追加	タイプ (typeの値)
URL	レスポンダーアクションを追加	ターゲット
リダイレクトステータスコード	レスポンダーアクションを追加	responseStatusCode
リダイレクト理由	レスポンダーアクションを追加	reasonPhrase
～で応答	レスポンダーアクションを追加	タイプ（type の値）
HTTPペイロード文字列	レスポンダーアクションを追加	ターゲット
何もしない	レスポンダーポリシーを追加	アクション（action の値）

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD属性	NETSCALERコマンド	NETSCALER属性
リセット	レスポンダーポリシーを追加	アクション (action の値)
ドロップ	レスポンダーポリシーを追加	アクション (action の値)
応答基準	レスポンダーポリシーを追加	ルール
デフォルトアクション	レスポンダーポリシーを追加	undefAction

監査ログ

次の表に、リライトポリシーまたはレスポンダーポリシー内で監査ログを有効にするために提供される **CRD** 属性を示します。また、この表には対応するNetScalerコマンドと属性も示されています。

CRD属性	NETSCALERコマンド	NETSCALER属性
ログ式	監査メッセージアクションの追加	stringBuilderExpr
ログレベル	監査メッセージアクションの追加	ログレベル

データセット

次の表に、リライトポリシーまたはレスポンダーポリシー内で使用できるデータセットの**CRD**属性を示します。また、この表には対応するNetScalerコマンドと属性も記載されています。

CRD属性	NETSCALERコマンド	NETSCALER属性
名前	ポリシーデータセットの追加	名前
タイプ	ポリシーデータセットの追加	タイプ
値	ポリシーデータセットのバインド	値

パットセット

CRD属性	NETSCALERコマンド	NETSCALER属性
名前	ポリシーパットセットの追加	名前

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD属性	NETSCALERコマンド	NETSCALER属性
値	ポリシーパットセットのバインド	文字列

文字列マップ

CRD属性	NETSCALERコマンド	NETSCALER属性
名前	ポリシー文字列マップの追加	名前
キー	ポリシー文字列マップのバインド	キー
値	ポリシー文字列マップのバインド	値

Goto-優先度-式

次の表に、複数の連続するポリシーのグループをサービスにバインドするためのCRD属性である `goto-priority-expression` 属性に関する情報を示します。

CRD属性	NETSCALERコマンド	NETSCALER属性	サポートされている値	デフォルト値
ゴートゥー・プリロティ・エクスプレッション	LB仮想サーバーをバインド	gotoPriorityExpression	ネクストとエンド	End

`goto-priority-expression` 属性の使用方法の詳細については、例「要求されたURLの文字列とホスト名を変更する」を参照してください。

ポリシー構成の記述方法

NetScalerによって提供されるCRDをKubernetesクラスターに展開した後、`.yaml` ファイルでポリシー構成を定義できます。`.yaml` ファイルでは、`kind` フィールドで `rewritepolicy` を使用し、要件に基づいて、ポリシー構成のために `spec` に次の個別のセクションのいずれかを追加します。

- `rewrite-policy` - リライトポリシー構成を定義します。
- `responder-policy` - レスポンダーポリシー設定を定義するため。
- `logpackets` - 監査ログを有効にするため。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- `dataset` - 広範なポリシー設定にデータセットを使用するため。
- `patset` - 広範なポリシー設定にパターンセットを使用するため。
- `stringmaps` - 広範なポリシー設定に文字列マップを使用するため。

これらのセクションでは、ポリシーを定義するために、それぞれのポリシー設定（リライトまたはレスポンダー）に提供されているCRD属性を使用する必要があります。

また、`spec`セクションでは、ポリシーを適用する必要があるサービスを指定するために、`rewrite-policies`セクションを含める必要があります。詳細については、ポリシー設定の例を参照してください。

`.yaml`ファイルをデプロイすると、NetScaler Ingress ControllerはIngress NetScalerデバイスにポリシー設定を適用します。

ポリシー設定のガイドライン

- CRDが名前空間に関連付けられている場合、デフォルトでは、ポリシーはその名前空間に関連付けられているサービスに適用されます。たとえば、複数の名前空間に同じサービス名が関連付けられている場合、ポリシーはCRDに関連付けられている名前空間に属するサービスに適用されます。
- 単一の`.yaml`ファイルで複数のポリシーを定義している場合、ファイル内で最初に定義されたポリシー設定が優先され、その後のポリシー設定は順序に従って適用されます。異なるファイルで複数のポリシーを定義している場合、最初にデプロイしたファイルで定義された最初のポリシー設定が優先されます。

Goto-priority-expressionの使用に関するガイドライン

- リライトポリシーとレスポンダーポリシーは、`goto-priority-expression`フィールド内の`NEXT`キーワードを使用して複数のグループとして組み合わせることができます。
- 現在のポリシー内で`goto-priority-expression`フィールドが`NEXT`であり、現在のポリシーが`True`と評価される場合、`goto-priority-expression`フィールドが`END`を指していない限り、グループ内の次のポリシーが実行され、フローは次の連続するポリシーに移動します。
- 現在のポリシーが`FALSE`と評価される場合、ポリシーの実行は現在のポリシーで停止するため、`goto-priority-expression`は影響を与えません。
- リライトまたはレスポンダーポリシー内のリライトまたはレスポンダーポリシーグループは、`goto-priority-expression`がNEXTとして割り当てられたポリシーから始まり、`goto-priority-expression`フィールドに`END`が割り当てられるまですべての連続するポリシーを含みます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- `goto-priority-expression`を使用してリライトまたはレスポンダーポリシーをグループ化する場合、グループ内のポリシーにバインドされているサービス名は同じである必要があります。
- リライトポリシーまたはレスポンダーポリシー内の最後のポリシーは、常に`goto-priority-expression`を`END`として持つ必要があります。
- ポリシーに対して`goto-priority-expression`フィールドが指定されていない場合、デフォルト値のENDが`goto-priority-expression`に割り当てられます。

注:

`goto-priority-expression`フィールドの使用方法の詳細については、例「要求されたURLの文字列とホスト名を変更する」を参照してください。

リライトおよびレスポンダーポリシーの作成と検証

NetScalerで、すべての受信URLを`new-url-for-the-application`に書き換え、マイクロサービスに送信するポリシーを定義するシナリオを考えてみましょう。`target-url-rewrite.yaml`という名前の`.yaml`ファイルを作成し、適切なCRD属性を使用して、リライトポリシーを次のように定義します。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: targeturlrewrite

spec:

  rewrite-policies:

    - servicenames:

        - citrix-svc

      logpackets:

        logexpression: "http.req.url"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        loglevel: INFORMATIONAL

      rewrite-policy:

        operation: replace

        target: 'http.req.url'

        modify-expression: '"new-url-for-the-application"'

        comment: 'Target URL Rewrite - rewrite the url of the HTTP
request'

        direction: REQUEST

        rewrite-criteria: 'http.req.is_valid'
```

ポリシー構成を定義したら、次のコマンドを使用して`.yaml`ファイルをデプロイします。

```
kubectl create -f target-url-rewrite.yaml
```

`.yaml`ファイルをデプロイすると、NetScaler Ingress Controllerは、Ingress NetScalerデバイスにポリシー構成を適用します。

Kubernetesクラスターのマスターノードで、次のコマンドを使用して、適用されたリライトポリシーCRDのステータスを確認できます。

```
Kubectl get rewritepolicies.citrix.com targeturlrewrite
```

ステータスは次のように表示されます。

```
kubectl get rewritepolicies.citrix.com targeturlrewrite

NAME               STATUS    MESSAGE

targeturlrewrite   Success   CRD Activated
```

CRDの作成または適用中に問題が発生した場合、citrix-k8s-ingress-controllerログを使用してデバッグできます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl logs citrixingresscontroller
```

また、次の手順を使用して、構成がNetScalerに適用されているかどうかを確認できます。

1. NetScalerコマンドラインにログオンします。
2. 以下のコマンドを使用して、設定がNetScalerに適用されているかを確認します。

   ```
   show run | grep `lb vserver`

   add lb vserver k8s-citrix_default_80_k8s-citrix-svc_default_80_svc HTTP
   0.0.0.0 0 -persistenceType NONE -cltTimeout 180

   bind lb vserver k8s-citrix_default_80_k8s-citrix-svc_default_80_svc
   k8s-citrix_default_80_k8s-citrix-svc_default_80_svc

   bind lb vserver k8s-citrix_default_80_k8s-citrix-svc_default_80_svc
   -policyName k8s_crd_rewritepolicy_rwpolicy_targeturlrewrite_0_default
   -priority 100300076 -gotoPriorityExpression END -type REQUEST
   ```

   ポリシー `k8s_crd_rewritepolicy_rwpolicy_targeturlrewrite_0_default` がロードバランシング仮想サーバーにバインドされていることを確認できます。

ポリシー設定の例

レスポンダーポリシー設定

以下はレスポンダーポリシー設定の例です (`block-list-urls.yaml`)

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: blocklisturls

spec:

  responder-policies:

    - servicenames:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        - citrix-svc
      responder-policy:
        respondwith:
          http-payload-string: '"HTTP/1.1 401 Access denied"'
        respond-criteria: 'http.req.url.equals_any("blocklistUrls")'
        comment: 'Blocklist certain Urls'



  patset:
    - name: blocklistUrls
      values:
        - '/app1'
        - '/app2'
        - '/app3'
```

この例では、NetScalerが `patset` で定義された `/app1`、`/app2`、または `/app3` の文字列に一致するURLを受信した場合、NetScalerはそのURLをブロックします。

監査ログが有効なポリシー

以下は、監査ログが有効なポリシーの例です (block-list-urls-audit-log.yaml)。

```
apiVersion: citrix.com/v1
kind: rewritepolicy
metadata:
  name: blocklisturls
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:
  responder-policies:
    - servicenames:
        - citrix-svc
      logpackets:
        logexpression: "http.req.url"
        loglevel: INFORMATIONAL
      responder-policy:
        respondwith:
          http-payload-string: '"HTTP/1.1 401 Access denied"'
        respond-criteria: 'http.req.url.equals_any("blocklistUrls")'
        comment: 'Blocklist certain Urls'



  patset:
    - name: blocklistUrls
      values:
        - '/app1'
        - '/app2'
        - '/app3'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

複数のポリシー設定

単一の `.yaml` ファイルに複数のポリシー設定を追加し、NetScalerデバイスにポリシーを適用できます。各ポリシー設定には個別のセクションを追加する必要があります (multi-policy-config.yaml)。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: multipolicy

spec:

  responder-policies:

    - servicenames:

        - citrix-svc

      responder-policy:

        redirect:

          url: '"www.citrix.com"'

        respond-criteria:
'client.ip.src.TYPECAST_text_t.equals_any("redirectIPs")'

        comment: 'Redirect IPs to citrix.com'

    - servicenames:

        - citrix-svc

      responder-policy:

        redirect:

          url:
'HTTP.REQ.HOSTNAME+http.req.url.MAP_STRING_DEFAULT_TO_KEY("modifyurls")'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      respond-criteria: 'http.req.is_valid'
      comment: 'modify specific URLs'


rewrite-policies:
  - servicenames:
      - citrix-svc
    rewrite-policy:
      operation: insert_http_header
      target: 'sessionID'
      modify-expression: '"48592th42gl24456284536tgt2"'
      comment: 'insert SessionID in header'
      direction: RESPONSE
      rewrite-criteria: 'http.res.is_valid'




dataset:
  - name: redirectIPs
    type: ipv4
    values:
      - 10.1.1.100
      - 1.1.1.1 - 1.1.1.100
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        - 2.2.2.2/10



  stringmap:

    - name: modifyurls

      comment: Urls to be modified string

      values:

        - key: '"/app1/"'

          value: '"/internal-app1/"'

        - key: '"/app2/"'

          value: '"/internal-app2/"'
```

この例には、2つのレスポンダーポリシーと1つの書き換えポリシーが含まれており、これらのポリシーに基づいてNetScalerは以下を実行します。

- `redirectIPs` データセットで指定されたクライアントIPアドレス、つまり `10.1.1.100`、範囲 `1.1.1.1 - 1.1.1.100` のIPアドレス、およびサブネット `2.2.2.2/10` のIPアドレスに一致するすべてのリクエストは、 `www.citrix.com` にリダイレクトされます。
- `modifyurls` ストリングマップで提供される文字列を含むすべての受信URLは、ストリングマップで提供される値に修正されます。たとえば、受信URLに `/app1/` の文字列がある場合、 `/internal-app1/` に修正されます。
- クライアントへの応答に、セッションIDを新しいヘッダーとして追加します。

使用例

レスポンスヘッダーを追加

クライアントからのリクエストURLに `/citrix-app/` が含まれている場合、リライトポリシーを使用して、マイクロサービスからクライアントへのHTTPレスポンスに以下のヘッダーを追加できます。

- クライアントのソースポートをヘッダーに

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- サーバーの宛先IPアドレス
- ランダムなHTTPヘッダー

次のサンプルリライトポリシー定義は、これらのヘッダーをマイクロサービスからクライアントへのHTTPレスポンスに追加します。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

name: addresponseheaders

spec:

rewrite-policies:

  - servicenames:

      - frontend

    rewrite-policy:

      operation: insert_before_all

      target: http.res.full_header

      modify-expression: '"\r\nx-port: "+client.tcp.srcport+"\r\nx-
ip:"+client.ip.dst+"\r\nx-new-dummy-header: Sending_a_gift"'

      multiple-occurence-modify: 'text("\r\n\r\n")'

      comment: 'Response header rewrite'

      direction: RESPONSE

      rewrite-criteria: 'http.req.url.contains("/citrix-app/")'
```

リライトポリシー定義を含むYAMLファイル (`add_response_headers.yaml`) を作成し、次のコマンドを使用してYAMLファイルをデプロイします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl create -f add_response_headers.yaml
```

レスポンスに追加されたHTTPヘッダーは、次のように確認できます。

```
$ curl -vvv http://app.cic-citrix.org/citrix-app/

*   Trying 10.102.33.176...

* TCP_NODELAY set

* Connected to app.cic-citrix.org (10.102.33.176) port 80 (#0)

> GET /citrix-app/ HTTP/1.1

> Host: app.cic-citrix.org

> User-Agent: curl/7.54.0

> Accept: */*

>

< HTTP/1.1 200 OK

< Server: nginx/1.8.1

< Date: Fri, 29 Mar 2019 11:14:04 GMT

< Content-Type: text/html

< Transfer-Encoding: chunked

< Connection: keep-alive

< X-Powered-By: PHP/5.5.9-1ubuntu4.14

< x-port: 22481 ==================> NEW RESPONSE HEADER

< x-ip:10.102.33.176 ==================> NEW RESPONSE HEADER

< x-new-dummy-header: Sending_a_gift ==================> NEW RESPONSE
HEADER
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
<

&lt;html>

&lt;head>

&lt;title> Front End App - v1 &lt;/title>




TRIMMED

.......
```

HTTPレスポンスパケットにカスタムヘッダーを追加

リライトポリシーを使用すると、マイクロサービスからクライアントへのHTTPレスポンスにカスタムヘッダーを追加できます。

次のサンプルリライトポリシー定義は、カスタムヘッダーをマイクロサービスからクライアントへのHTTPレスポンスに追加します。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: addcustomheaders

spec:

  rewrite-policies:

    - servicenames:

        - frontend

      rewrite-policy:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        operation: insert_before_all

        target: http.res.full_header

        modify-expression: '"\r\nx-request-time:"+sys.time+"\r\nx-using-
citrix-ingress-controller: true"'

        multiple-occurence-modify: 'text("\r\n\r\n")'

        comment: 'Adding custom headers'

        direction: RESPONSE

        rewrite-criteria: 'http.req.is_valid'
```

リライトポリシー定義を含むYAMLファイル (`add_custom_headers.yaml`) を作成し、次のコマンドを使用してYAMLファイルをデプロイします。

```
kubectl create -f add_custom_headers.yaml
```

レスポンスに追加されたカスタムHTTPヘッダーは、次のように確認できます。

```
$ curl -vvv http://app.cic-citrix.org/

* Trying 10.102.33.176...

* TCP_NODELAY set

* Connected to app.cic-citrix.org (10.102.33.176) port 80 (#0)

> GET / HTTP/1.1

> Host: app.cic-citrix.org

> User-Agent: curl/7.54.0

> Accept: */*

>
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
< HTTP/1.1 200 OK

< Server: nginx/1.8.1

< Date: Fri, 29 Mar 2019 12:15:09 GMT

< Content-Type: text/html

< Transfer-Encoding: chunked

< Connection: keep-alive

< X-Powered-By: PHP/5.5.9-1ubuntu4.14

< x-request-time:Fri, 29 Mar 2019 13:27:40 GMT ==============> NEW HEADER
ADDED

< x-using-citrix-ingress-controller: true  ================> NEW HEADER
ADDED

<

&lt;html>

&lt;head>

&lt;title> Front End App - v1 &lt;/title>

&lt;style>



TRIMMED

........
```

リクエスト内のホスト名を置き換える

次のYAML例 (`http_request_modify_prefixasprefix.yaml`) に示すようにリライトポリシーを定義して、要件に応じてHTTPリクエストのホスト名を置き換えることができます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: httpheadermodifyretainprefix

spec:

  rewrite-policies:

    - servicenames:

        - frontend

      rewrite-policy:

        operation: replace_all

        target: 'http.req.header("host")'

        modify-expression: '"citrix-service-app"'

        multiple-occurence-modify: 'text("app.cic-citrix.org")'

        comment: 'HTTP header rewrite of hostname'

        direction: REQUEST

        rewrite-criteria: 'http.req.is_valid'
```

リライトポリシー定義を含むYAMLファイル (`http_request_modify_prefixasprefix.yaml`) を作成し、以下のコマンドを使用してYAMLファイルをデプロイします。

```
kubectl create -f http_request_modify_prefixasprefix.yaml
```

`curl` コマンドを使用してポリシー定義を検証できます。リクエスト内のホスト名は、定義されたホスト名に置き換えられます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
curl http://app.cic-citrix.org/prefix/foo/bar
```

出力:

アプリケーションルートの変更

既存のアプリケーションルートが`/`の場合、アプリケーションルートを変更するリライトポリシーを定義できます。

次のサンプルリライトポリシーは、リクエストURLの`/`を`/citrix-approot/`に変更します。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

 name: httpapprootrequestmodify

spec:

 rewrite-policies:

   - servicenames:

       - frontend

     rewrite-policy:

       operation: replace

       target: http.req.url

       modify-expression: '"/citrix-approot/"'

       comment: 'HTTP app root request modify'

       direction: REQUEST

       rewrite-criteria: http.req.url.eq("/")
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

リライトポリシー定義を含むYAMLファイル (`http_approot_request_modify.yaml`) を作成し、以下のコマンドを使用してYAMLファイルをデプロイします。

kubectl create -f http_approot_request_modify.yaml

`curl` コマンドを使用して、要件に応じてアプリケーションルートが変更されたかどうかを検証できます。

```
curl -vvv http://app.cic-citrix.org/
```

出力:

リクエストされたURL内の文字列を変更する

要件に応じて、リクエストされたURL内の文字列を変更するリライトポリシーを定義できます。

次のサンプルリライトポリシーは、リクエストされたURL内の文字列`something`を`simple`に置き換えます。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

name: httpurlreplacestring

spec:

rewrite-policies:

 - servicenames:

     - frontend

   rewrite-policy:

     operation: replace_all

     target: http.req.url

     modify-expression: '"/"'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
     multiple-occurence-modify: 'regex(re~((^(\/something\/))|(^\/
something$))~)'

     comment: 'HTTP url replace string'

     direction: REQUEST

     rewrite-criteria: http.req.is_valid
```

リライトポリシー定義を含むYAMLファイル (`http_url_replace_string.yaml`) を作成し、以下のコマンドを使用してYAMLをデプロイします。

```
kubectl create -f http_url_replace_string.yaml
```

`curl` リクエストと文字列 `something` を使用してポリシー定義を検証できます。文字列 `something` は、次の例に示すように文字列 `simple` に置き換えられます。

例 1:

```
curl http://app.cic-citrix.org/something/simple/citrix
```

出力:

例 2:

```
curl http://app.cic-citrix.org/something
```

または、,

```
curl http://app.cic-citrix.org/something/
```

出力:

HTTP リクエスト内に `X-Forwarded-For` ヘッダーを追加する

HTTP リクエスト内に `X-Forwarded-For` ヘッダーを追加するには、次の YAML 例 (`http_x_forwarded_for_insert.yaml`) に示すようにリライトポリシーを定義できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: httpxforwardedforaddition

spec:

  rewrite-policies:

    - servicenames:

        - frontend

      rewrite-policy:

        operation: insert_http_header

        target: X-Forwarded-For

        modify-expression: client.ip.src

        comment: 'HTTP Initial X-Forwarded-For header add'

        direction: REQUEST

        rewrite-criteria: 'HTTP.REQ.HEADER("X-Forwarded-For").EXISTS.NOT'



    - servicenames:

        - frontend

      rewrite-policy:

        operation: replace

        target: HTTP.REQ.HEADER("X-Forwarded-For")

        modify-expression: 'HTTP.REQ.HEADER("X-Forwarded-
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
For").APPEND(",").APPEND(CLIENT.IP.SRC)'

        comment: 'HTTP Append X-Forwarded-For IPs'

        direction: REQUEST

        rewrite-criteria: 'HTTP.REQ.HEADER("X-Forwarded-For").EXISTS'
```

リライトポリシー定義を含む YAML ファイル (`http_x_forwarded_for_insert.yaml`) を作成し、次のコマンドを使用して YAML ファイルをデプロイします。

```
kubectl create -f http_x_forwarded_for_insert.yaml
```

`curl` コマンドを使用して、`X-Forwarded-For` ヘッダーの有無にかかわらず HTTP パケットを検証できます。

例: `X-Forwarded-For` ヘッダーなしの HTTP リクエストパケットの出力:

```
curl http://app.cic-citrix.org/
```

出力:

例：`X-Forwarded-For`ヘッダーを含むHTTPリクエストパケットの出力：

```
curl  curl --header "X-Forwarded-For: 1.1.1.1" http://app.cic-citrix.org/
```

出力：

レスポンダーポリシーを使用してHTTPリクエストをHTTPSリクエストにリダイレクトする

HTTPリクエストをHTTPSリクエストにリダイレクトするには、次のYAMLの例(`http_to_https_redirect.yaml`)に示すように、レスポンダーポリシー定義を定義できます。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: httptohttps

spec:

  responder-policies:

    - servicenames:

        - frontend

      responder-policy:

        redirect:

          url: '"https://"
+http.req.HOSTNAME.SERVER+":"+"443"+http.req.url'

        respond-criteria: 'http.req.is_valid'

        comment: 'http to https'
```

レスポンダーポリシー定義を含むYAMLファイル(`http_to_https_redirect.yaml`)を作成し、次のコマンドを使用してYAMLファイルをデプロイします。

```
kubectl create -f http_to_https_redirect.yaml
```

HTTPリクエストがHTTPSにリダイレクトされているかどうかは、次のように確認できます。

例1：

```
$ curl -vvv  http://app.cic-citrix.org

* Rebuilt URL to: http://app.cic-citrix.org/

*   Trying 10.102.33.176...

* TCP_NODELAY set

* Connected to app.cic-citrix.org (10.102.33.176) port 80 (#0)
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
> GET / HTTP/1.1

> Host: app.cic-citrix.org

> User-Agent: curl/7.54.0

> Accept: */*

>

< HTTP/1.1 302 Found : Moved Temporarily

< Location: https://app.cic-citrix.org:443/   =======> Redirected to HTTPS

< Connection: close

< Cache-Control: no-cache

< Pragma: no-cache

<

* Closing connection 0
```

例2：

```
$ curl -vvv  http://app.cic-citrix.org/simple

*   Trying 10.102.33.176...

* TCP_NODELAY set

* Connected to app.cic-citrix.org (10.102.33.176) port 80 (#0)

> GET /simple HTTP/1.1

> Host: app.cic-citrix.org

> User-Agent: curl/7.54.0

> Accept: */*
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
>

< HTTP/1.1 302 Found : Moved Temporarily

< Location: https://app.cic-citrix.org:443/simple     ========> Redirected
to HTTPS

< Connection: close

< Cache-Control: no-cache

< Pragma: no-cache

<

* Closing connection 0
```

リクエストされたURL内の文字列とホスト名を変更する

この例では、`goto-priority-expression`属性の使用方法を示します。`goto-priority-expression`フィールドの使用ガイドラインは、で確認できます。この例では、URL `http://www.citrite.org/something/simple/citrix`を`http://app.cic-citrix.org/simple/citrix`に変更します。

URLを変更するために、2つのリライトポリシーが記述されています。

- リライトポリシー1：このポリシーは、ホスト名`www.citrite.org`を`app.cic-citrix.org`に変更するために使用されます。
- リライトポリシー2：このポリシーは、URL `/something/simple/citrix`を`/simple/citrix`に変更するために使用されます。

次のYAMLに示すように、`goto-priority-expression`属性を使用して2つのポリシーをバインドできます。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: hostnameurlrewrite
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:

  rewrite-policies:

    - servicenames:

        - citrix-svc

      goto-priority-expression: NEXT

      rewrite-policy:

        operation: replace_all

        target: 'http.req.header("host")'

        modify-expression: '"app.cic-citrix.org"'

        multiple-occurence-modify: 'text("www.citrite.org")'

        comment: 'HTTP header rewrite of hostname'

        direction: REQUEST

        rewrite-criteria:
'http.req.is_valid.and(HTTP.REQ.HOSTNAME.EQ("www.citrite.org"))'

    - servicenames:

        - citrix-svc

      goto-priority-expression: END

      rewrite-policy:

        operation: replace_all

        target: http.req.url

        modify-expression: '"/"'

        multiple-occurence-modify: 'regex(re~((^(\/something\/))|(^\/
something$))~)'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        comment: 'HTTP url replace string'

        direction: REQUEST

        rewrite-criteria:
'http.req.is_valid.and(HTTP.REQ.HOSTNAME.EQ("www.citrite.org"))'`
```

検証

以下のcurlリクエスト `http://www.citrite.org/something/simple/citrix` が `http://app.cic-citrix.org/simple/citrix` に変更されたかどうかを確認できます。

例: 要求されたURLの変更

```
curl http://www.citrite.org/something/simple/citrix
```

要求されたURLの変更されたホスト名とURLは、以下に示す画像に表示されています。

HTTPコールアウト

HTTPコールアウトを使用すると、NetScalerはポリシー評価の一部として外部サーバーにHTTPまたはHTTPSリクエストを生成して送信し、外部サーバーから取得した応答に基づいて適切なアクションを実行できます。リライトおよびレスポンダーCRDを使用して、NetScalerからHTTPコールアウトリクエストを開始できます。詳細については、HTTPコールアウトのドキュメント を参照してください。

関連資料

- 機能ドキュメント
  - NetScaler Rewrite機能ドキュメント
  - NetScaler Responder機能ドキュメント
- 開発者向けドキュメント
  - NetScaler Rewriteポリシー
  - NetScaler Rewriteアクション
  - NetScaler Responderポリシー
  - NetScaler Responderアクション

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- NetScaler監査メッセージアクション
- NetScalerポリシーデータセット

ICAPを使用したリモートコンテンツ検査またはコンテンツ変換サービス

インターネットコンテンツ適応プロトコル (ICAP) は、HTTPメッセージで付加価値変換サービスを実行するためのシンプルで軽量なプロトコルです。一般的なシナリオでは、ICAPクライアントはHTTPリクエストとレスポンスを1つ以上のICAPサーバーに転送して処理します。ICAPサーバーはリクエストに対してコンテンツ変換を実行し、リクエストまたはレスポンスに対して適切なアクションを実行するレスポンスを返します。

NetScalerのセットアップでは、NetScalerはアンチマルウェアやデータ損失防止 (DLP) などのサードパーティICAPサーバーと相互運用するICAPクライアントとして機能します。NetScalerが受信ウェブトラフィックを受信すると、トラフィックを傍受し、コンテンツ検査ポリシーを使用してHTTPリクエストがICAP処理を必要とするかどうかを評価します。必要であれば、NetScalerはメッセージを復号化し、プレーンテキストとしてICAPサーバーに送信します。ICAPサーバーはリクエストメッセージに対してコンテンツ変換サービスを実行し、NetScalerに応答を返します。変更されたメッセージは、HTTPリクエストまたはHTTPレスポンスのいずれかです。NetScalerが複数のICAPサーバーと相互運用する場合、NetScalerはICAPサーバーの負荷分散を実行します。このセットアップは、1つのICAPサーバーではすべてのトラフィック負荷を処理するのに十分でない場合に役立ちます。ICAPサーバーが変更されたメッセージを返した後、NetScalerは変更されたメッセージをバックエンドのオリジンサーバーに転送します。

NetScalerは、受信トラフィックがHTTPSタイプの場合、セキュアなICAPサービスも提供します。NetScalerはSSLベースのTCPサービスを使用して、NetScalerとICAPサーバー間のセキュアな接続を確立します。

Kubernetes環境では、NetScaler Ingress Controllerを介してNetScalerでICAPを有効にするために、NetScalerはICAPカスタムリソース定義 (CRD) を提供します。ICAPを有効にすることで、次のアクションを実行できます。

- 指定された文字列を含むURLをブロックする
- DDoS攻撃を軽減するために一連のIPアドレスをブロックする
- HTTPからHTTPSへの強制

KubernetesクラスターにICAP CRDをデプロイした後、ICAP CRD属性を使用してICAPポリシーを定義し、NetScalerの統計情報のために監査ログを有効にすることができます。NetScalerが提供するICAP機能の詳細については、リモートコンテンツ検査のためのICAPを参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ICAPポリシーは、大きくリクエスト変更ポリシーとレスポンス変更ポリシーに分類できます。

リクエスト変更: リクエスト変更 (REQMOD) モードでは、NetScalerはクライアントから受信したHTTPリクエストをICAPサーバーに転送します。リクエスト変更CRDリソースの例については、サンプルポリシー構成を参照してください。

レスポンス変更: レスポンス変更 (RESPMOD) モードでは、NetScalerはHTTPレスポンスをICAPサーバーに送信します (NetScalerが送信するレスポンスは通常、オリジンサーバーが送信するレスポンスです)。レスポンス変更CRDリソースの例については、サンプルポリシー構成を参照してください。

リクエストおよびレスポンス変更ポリシーの詳細については、リモートコンテンツ検査のためのICAPを参照してください。

NetScaler® ICAP CRDのデプロイ

NetScaler ICAP CRDデプロイYAMLファイルは、GitHubの次の場所にあります: icap-crd-deployment.yaml。

> **注記:**
>
> デプロイメントYAMLファイルを変更しないようにしてください。

次のコマンドを使用してICAP CRDを展開します。

```
kubectl create -f icap-crd-deployment.yaml
```

例:

```
root@master:~# kubectl create -f icap-crd-deployment.yaml

customresourcedefinition.apiextensions.k8s.io/icap-crd.citrix.com created
```

ICAP CRD属性

ICAP CRD は、コンテンツ検査を有効にし、ICAPポリシーを定義するために必要なさまざまなオプションを設定するための属性を提供します。これらのCRD属性は、それぞれ **NetScaler** コマンドと属性に対応しています。

次の表に、ICAPポリシーを定義するために使用できる **CRD** 属性を示します。また、この表には対応するNetScalerコマンドと属性も記載されています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ICAPサーバーの追加

ICAP CRD仕様の `icap-servers` オブジェクトの下にICAPサーバーを指定できます。要件に基づいて、1つまたは複数のICAPサーバーを指定できます。

CRD属性	NETSCALERコマンド	NETSCALER属性	説明
ip (必須)	`add service <name> <IP> <serviceType> <port>`	IP	ICAPサーバーのIPアドレス
ポート (必須)	`add service <name> <IP> <serviceType> <port>`	ポート	ICAPサーバーが通信するポート
サーバータイプ (必須)	`add service <name> <IP> <serviceType> <port>`	serviceType	ICAPサーバーのタイプ。指定可能な値は `TCP` と `SSL_TCP` です。

バックエンドサービスとイングレスクラスの指定

次の設定を行うには、`Services` と `ingressclass` 属性を指定します。

- ICAPポリシーを有効にする必要があるバックエンドサービスを指定します。
- イングレスコントローラーによって処理される必要があるICAPリソースを指定します。

CRD属性	NetScalerコマンド	説明
Services (必須)	NA	ICAPポリシーを有効にする必要があるバックエンドサービスを一覧表示します。
ingressclass (必須)	NA	指定されたイングレスクラスに関連付けられたイングレスコントローラーのみがリソースを処理するように、イングレスクラスを指定します。そうでない場合、クラスター内のすべてのコントローラーが

© 1997–2026 Citrix Systems, Inc. All rights reserved.

		このCRDリソースを処理します。

ICAPプロファイルの追加

NetScaler の ICAP 設定は、ICAP プロファイルと呼ばれるエンティティで指定されます。このプロファイルには、ICAP 設定のコレクションが含まれています。設定には、ICAP リクエストを動的に生成し、ICAP レスポンスを受信し、コンテンツ検査データをログに記録するためのパラメータが含まれます。

CRD属性	NETSCALERコマンド	NETSCALER属性	説明
preconfigured-profile (オプション)	NA	NA	事前設定されたICAPプロファイルの名前。
direction (必須)	`add ns icapProfile <name> -uri <string> -Mode ( REQMOD | RESPMOD )`	`Mode`	ICAPの動作モード。可能な値は `REQUEST` と `RESPONSE` です。
uri (必須)	`add ns icapProfile <name> -uri <string> -Mode ( REQMOD | RESPMOD )`	URI	ICAPサービスを表すURI。
プレビュー	`add ns icapProfile <name> -uri <string> -Mode ( REQMOD | RESPMOD ) -preview ( ENABLED | DISABLED )]`	プレビュー	ICAPリクエストでプレビューヘッダーを有効または無効にします。この機能により、ICAPサーバーはトランザクションの開始部分を確認できます。その後、リクエストメッセージの残りの部分を受信する代わりに、早期にトランザクションか

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD属性	NETSCALERコマンド	NETSCALER属性	説明
			らオプトアウトするかどうかを決定できます。
プレビュー長	`add ns icapProfile <name> -uri <string> -Mode ( REQMOD | RESPMOD ) -previewLength <positive_integer>`	previewLength	`Preview Header` フィールドの値。NetScalerは、この設定値とOPTIONSで受信したプレビューサイズのうち、小さい方の値を使用します。
ホストヘッダー	`add ns icapProfile <name> -uri <string> -Mode ( REQMOD | RESPMOD ) -hostHeader <string>`	hostHeader	ICAPホストヘッダー。
ユーザーエージェントヘッダー	`add ns icapProfile <name> -uri <string> -Mode ( REQMOD | RESPMOD ) -userAgent <string>`	userAgent	ICAPユーザーエージェントヘッダー。
クエリパラメータ	`add ns icapProfile <name> -uri <string> -Mode ( REQMOD | RESPMOD )`	queryParams	ICAPリクエストURIに含めるクエリパラメータ。入力値は `arg=value` 形式である必要があります。複数のパラメータがある場合は、パラメータ間に `&` を

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD属性	NETSCALERコマンド	NETSCALER属性	説明
	`-queryParams <string>`		追加します。例: `arg1=val1&arg2=val2`。
コネクションキープアライブ	`add ns icapProfile <name> -uri <string> -Mode ( REQMOD | RESPMOD ) -connectionKeepAlive ( ENABLED | DISABLED )`	connectionKeepAlive	ICAPリクエストでAllow: 204ヘッダーの送信を有効または無効にします。
ICAPヘッダーの挿入	`add ns icapProfile <name> -uri <string> -Mode ( REQMOD | RESPMOD ) -insertICAPHeaders <string>`	insertICAPHeaders	ICAPサーバーに送信するICAPリクエストにカスタムICAPヘッダーを挿入します。ヘッダーは静的でも、PIポリシー式を使用して動的に構築することもできます。たとえば、静的なユーザーエージェントとクライアントのIPアドレスを送信するには、式を`"User-Agent: NS-ICAP-Client/V1.0r0-Client-IP: "+CLIENT.IP.SRC+" r0`として指定できます。NetScalerは、指定されたヘッダー名と値の有効性をチェックしません。指定されたヘッダー構文を手動で検証する必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD属性	NETSCALERコマンド	NETSCALER属性	説明
HTTPリクエストの挿入	`add ns icapProfile <name> -uri <string> -Mode ( REQMOD | RESPMOD ) -insertHTTPRequest <string>`	insertHTTPRequest	NetScalerがカプセル化してICAPサーバーに送信する、式形式の正確なHTTPリクエスト。このパラメーターを設定すると、ICAPリクエストはこのヘッダーのみを使用して送信されます。この属性は、HTTPヘッダーを送信できない場合や、ICAPサーバーが受信HTTPリクエストの一部のみを必要とする場合に使用できます。NetScalerはこのリクエストの有効性をチェックしません。リクエストを手動で検証する必要があります。
リクエストタイムアウト	`add ns icapProfile <name> -uri <string> -Mode ( REQMOD | RESPMOD ) -reqTimeout <positive_integer>`	reqTimeout	リモートサーバーがICAPリクエストに応答するまでの時間（秒単位）。NetScalerがこの時間内に完全な応答を受信しない場合、指定されたリクエストタイムアウトアクションが実行されます。値がゼロに設定されている場合、この機能は無効になります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD属性	NETSCALERコマンド	NETSCALER属性	説明
リクエストタイムアウトアクション	`add ns icapProfile <name> -uri <string> -Mode ( REQMOD | RESPMOD ) -reqTimeoutAction <reqTimeoutAction>`	reqTimeoutAction	設定されたタイムアウト値内にリモートサービスを表す仮想サーバーが応答しない場合に実行するアクション。サポートされているアクションは次のとおりです。**BYPASS** - リモートサーバーの応答を無視し、リクエスト/応答をクライアント/サーバーに送信します。カプセル化されたヘッダーを含むICAP応答が設定されたリクエストタイムアウト値内に受信されない場合、このオプションはリモートICAPサーバーの応答を無視し、完全なリクエスト/応答をサーバー/クライアントに送信します。**DROP** - ユーザーに応答を送信せずにリクエストを破棄します。**RESET** - クライアント接続を閉じてリセットします。クライアントは必要に応じてリクエストを再送信できます。

コンテンツ検査ポリシーとアクション

コンテンツ検査機能を有効にした後、ICAPリクエスト情報を処理するためのICAPアクションを追加する必要があります。作成されたICAPプロファイルとサービス、またはロードバランシング仮想サーバーは、ICAPアクションにバインドされます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD 属性	NETSCALER コマンド	NETSCALER 属性	説明
コンテンツ検査基準	`add contentInspection policy <name> -rule <expression> -action <string>`	ルール	ポリシーが指定されたアクションを実行するかどうかを決定するために使用する式。
デフォルトアクション	`add contentInspection action <name> -type ICAP -serverName <string> -icapProfileName <string> -ifserverdown <if-server-down>`	undefAction	ポリシー評価の結果が未定義 (UNDEF) の場合に実行するアクション。UNDEF イベントは内部エラー状態を示します。これより前の組み込みアクションのみを使用できます。
ログアクション	`add contentinspection policy <name> -rule <expression> -action <string> -logAction <string>`	logAction	このポリシーに一致するリクエストに使用するメッセージログアクションの名前。
操作	`add ContentInspection action <name> -type ICAP -serverip <ip> -serverport <port> -icapProfileName <string>`	タイプ	このICAPアクションが実行する操作のタイプ。設定可能なアクションは次のとおりです。 **ICAP** - 受信リクエストまたは応答をICAPサーバーに転送して変更します。 **INLINEINSPECTION** - 受信または送信パケットをIPSサー

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD 属性	NETSCALER コマンド	NETSCALER 属性	説明
			バーに転送して侵入防止を行います。 **MIRROR** - クローンされたパケットを転送して侵入検知を行います。 **NOINSPECTION** - 受信および送信パケットを検査デバイスに転送しません。 **NSTRACE** - このトランザクションで現在およびそれ以降の受信パケットをキャプチャします。
サーバー障害時のアクション	`add contentInspection action <name> -type ICAP -serverName <string> -icapProfileName <string> -ifserverdown <>`	サーバーダウン時	リモートサービスを表す仮想サーバーが稼働していない場合に実行するアクション。サポートされているアクションは次のとおりです。 **RESET** - クライアント接続を閉じてリセットします。ブラウザなどのクライアントプログラムがこのアクションを処理し、ユーザーに通知する場合があります。クライアントは必要に応じてリクエストを再送信できます。 **DROP** - ユーザーに応答を送信せずにリクエストを破棄します。 **CONTINUE** - コンテンツ検査をバイパス

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD 属性	NETSCALER コマンド	NETSCALER 属性	説明
			し、クライアントまたはサーバーへのトラフィックフローを再開します。

Goto優先度式

次の表は、ポリシーのグループをサービスにバインドするためのCRD属性である`goto-priority-expression`属性に関する情報を提供します。

CRD属性	NETSCALERコマンド	NETSCALER属性	サポートされている値	デフォルト値
ゴートゥー・プライオリティ・エクスプレッション	lb vserverをバインド	gotoPriorityExpression	NEXT と END	End

ポリシー構成の記述方法

NetScalerが提供するICAP CRD仕様をKubernetesクラスターにデプロイした後、`.yaml` ファイルでポリシー構成を定義できます。`.yaml` ファイルでは、`kind` フィールドで `icappolicy` を使用し、要件に基づいてポリシー構成の属性に値を指定します。必須およびオプションのパラメーターとその説明については、前のセクションを参照してください。

`.yaml` ファイルをデプロイすると、NetScaler Ingress ControllerはNetScalerにポリシー構成を適用します。

ポリシー構成のガイドライン

- CRDがnamespace に関連付けられている場合、デフォルトでは、ポリシーはその名前空間に関連付けられているサービスに適用されます。たとえば、複数の名前空間に同じサービス名が関連付けられている場合、ポリシーはCRDに関連付けられている名前空間に属するサービスに適用されます。
- 単一の `.yaml` ファイルで複数のポリシーを定義している場合、ファイル内で最初に定義されたポリシー構成が優先され、その後のポリシー構成はシーケンスに従って適用されます。異なるファイルで複数のポリシーを定義している場合、最初にデプロイしたファイルで定義された最初のポリシー構成が優先されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Goto-priority-expression の使用に関するガイドライン

- ICAPポリシーは、`goto-priority-expression` フィールド内で `NEXT` キーワードを使用することで、複数のグループとして組み合わせることができます。
- 現在のポリシー内で `goto-priority-expression` フィールドが `NEXT` であり、現在のポリシーが `True` と評価される場合、グループ内の次のポリシーが実行されます。このプロセスは、`goto-priority-expression` フィールドが `END` に設定されているポリシーが見つかるまで、後続のポリシーで継続されます。
- 現在のポリシーが`FALSE`と評価される場合、ポリシーの実行は現在のポリシーで停止するため、`goto-priority-expression`は影響しません。
- ICAPポリシー内のICAPポリシーグループは、NEXTとして`goto-priority-expression`が割り当てられたポリシーから始まり、`goto-priority-expression`フィールドに`END`が割り当てられるまですべての連続するポリシーを含みます。
- `goto-priority-expression`を使用してICAPポリシーをグループ化する場合、グループ内のポリシーにバインドされているサービス名は同じである必要があります。
- ICAP内の最後のポリシーは、常に`goto-priority-expression`を`END`として持つ必要があります。
- ポリシーに対して`goto-priority-expression`フィールドが指定されていない場合、ENDのデフォルト値が`goto-priority-expression`に割り当てられます。

ICAPポリシーの作成と検証

文字列`example`を含むマイクロサービスへのすべての受信URLをドロップするためにNetScalerでポリシーを定義したいシナリオを考えます。`exampleicappolicy.yaml`という名前の`.yaml`ファイルを作成し、適切なCRD属性を使用してICAPポリシーを次のように定義します。

```
apiVersion: citrix.com/v1beta1

kind: icappolicy

metadata:

  name: exampleicappolicy

spec:

  ingressclass: "cic-vpx"

  services:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    - "frontend"
  icap-servers:
    servers:
      - ip: "192.168.1.1"
        port: 1344
    server-type: "TCP"
  icap:
  - direction: "REQUEST"
    profile:
      preview: "ENABLED"
      preview-length: 1024
      uri: "http://icap.example.com/reqmod"
      host-header: "icap.example.com"
      user-agent-header: "testAgent"
      query-params: "arg1=val1&arg2=val2"
      connection-keep-alive: "ENABLED"
      req-timeout: 30
      req-timeout-action: "BYPASS"
    content-inspection-criteria: "HTTP.REQ.URL.CONTAINS(\"example\")"
    default-action: "DROP"
    goto-priority-expression: "END"
    operation: "ICAP"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    server-failure-action: "CONTINUE"
```

コマンド`kubectl create -f exampleicappolicy.yaml`を実行して`.yaml`ファイルをデプロイした後、NetScaler Ingress ControllerはNetScalerにポリシー構成を適用します。

Kubernetesクラスターのマスターノードで、次のコマンドを実行して適用されたICAPポリシーCRDのステータスを確認できます。

```
kubectl get icappolicies.citrix.com exampleicappolicy
```

次のコマンドを実行してステータスを表示できます。

```
kubectl get icappolicies.citrix.com exampleicappolicy

NAME               STATUS    MESSAGE

exampleicappolicy   Success   CRD Activated
```

CRDの作成または適用中に問題がある場合、イングレスコントローラーのログを使用してデバッグできます。

```
kubectl logs <nsic pod name>
```

また、次の手順を使用して、構成がNetScalerに適用されているかどうかを確認できます。

1. NetScaler CLIにログインします。
2. 構成がNetScalerに適用されているかどうかを確認するには、次のコマンドを使用します。

```
show ns icapProfile | grep exampleicappolicy

show icapProfile <icapProfile-name>

show run | grep -i icap
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ポリシー設定の例

応答モードでのICAPポリシー

この例では、ICAPポリシーが応答トラフィックに適用されます。つまり、ポリシーはICAPサーバーからNetScalerに送信される応答を検査し、場合によっては変更します。

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1beta1

kind: icappolicy

metadata:

  name: exampleicappolicy

spec:

  ingressclass: "cic-vpx"

  services:

    - "frontend"

  icap-servers:

    servers:

      - ip: "192.168.1.1"

        port: 1344

    server-type: "TCP"

  icap:

  - direction: "RESPONSE"

    profile:

      preview: "ENABLED"

      preview-length: 1024
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      uri: "http://icap.example.com"

      user-agent-header: "testAgent"

      query-params: "arg1=val1&arg2=val2"

      connection-keep-alive: "ENABLED"

      req-timeout: 30

      req-timeout-action: "BYPASS"

    content-inspection-criteria:
 "HTTP.RES.HEADER(\"Location\").CONTAINS(\"example\")"

    default-action: "DROP"

    goto-priority-expression: "END"

    operation: "ICAP"

    server-failure-action: "CONTINUE"

EOF
```

複数のICAPサーバーのパイプライン処理

この例では、リクエストの処理に複数のICAPサーバーを使用します。

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1beta1

kind: icappolicy

metadata:

  name: exampleicappolicy

spec:

  ingressclass: "cic-vpx"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
services:
  - "frontend"
icap-servers:
  servers:
    - ip: "192.168.1.1"
      port: 1344
    - ip: "192.168.1.2"
      port: 1344
    - ip: "192.168.1.3"
      port: 1344
  server-type: "TCP"
icap:
- direction: "RESPONSE"
  profile:
    preview: "ENABLED"
    preview-length: 1024
    uri: "http://icap.example.com"
    user-agent-header: "testAgent"
    query-params: "arg1=val1&arg2=val2"
    connection-keep-alive: "ENABLED"
    req-timeout: 30
    req-timeout-action: "BYPASS"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    content-inspection-criteria:
 "HTTP.RES.HEADER(\"Location\").CONTAINS(\"example\")"

    default-action: "DROP"

    goto-priority-expression: "END"

    operation: "ICAP"

    server-failure-action: "CONTINUE"

EOF
```

複数のポリシー設定

単一の`.yaml`ファイルに複数のポリシー設定を追加し、それらのポリシーをNetScalerに適用できます。ここに示されている例のように、各ポリシー設定に対して個別のセクションを追加する必要があります。

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1beta1

kind: icappolicy

metadata:

  name: exampleicappolicy1

spec:

  ingressclass: "cic-vpx"

  services:

    - "frontend"

  icap-servers:

    servers:

      - ip: "192.168.1.1"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        port: 1344
    server-type: "TCP"
  icap:
    - direction: "RESPONSE"
      profile:
        preview: "ENABLED"
        preview-length: 1024
        uri: "http://icap.example.com"
        user-agent-header: "testAgent"
        query-params: "arg1=val1&arg2=val2"
        connection-keep-alive: "ENABLED"
        req-timeout: 30
        req-timeout-action: "BYPASS"
      content-inspection-criteria:
"HTTP.RES.HEADER(\"Location\").CONTAINS(\"example\")"
      default-action: "DROP"
      goto-priority-expression: "NEXT"
      operation: "ICAP"
      server-failure-action: "CONTINUE"
---
apiVersion: citrix.com/v1beta1
kind: icappolicy
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
metadata:
  name: exampleicappolicy2
spec:
  ingressclass: "cic-vpx"
  services:
    - "frontend"
  icap-servers:
    servers:
      - ip: "192.168.1.1"
        port: 1344
    server-type: "TCP"
  icap:
    - direction: "RESPONSE"
      profile:
        preview: "ENABLED"
        preview-length: 1024
        uri: "http://icap.example.com"
        user-agent-header: "testAgent"
        query-params: "arg1=val1&arg2=val2"
        connection-keep-alive: "ENABLED"
        req-timeout: 30
        req-timeout-action: "BYPASS"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      content-inspection-criteria:
"HTTP.RES.HEADER(\"Location\").CONTAINS(\"sample\")"

      default-action: "DROP"

      goto-priority-expression: "END"

      operation: "ICAP"

      server-failure-action: "CONTINUE"

EOF
```

VIPカスタムリソース定義 (CRD)

NetScaler は、NetScaler **IPAMコントローラー** と NetScaler Ingress Controller の間の非同期通信のために、カスタムリソース定義 (CRD) と呼ばれる **VIP** を提供します。

NetScalerはIPアドレス管理のために**IPAM controller**を提供します。NetScaler **IPAM controller** は、定義されたIPアドレス範囲からサービスにIPアドレスを割り当てます。NetScaler Ingress Controllerは、サービスに割り当てられたIPアドレスをNetScaler VPXの仮想IP (VIP) アドレスとして構成し、サービスはこのVIPアドレスを使用して公開されます。

新しいサービスが作成されると、NetScaler Ingress ControllerはIPアドレスなしでそのサービス用のCRDオブジェクトを作成します。NetScaler **IPAM controller** は、CRDオブジェクトの追加、削除、または変更をチェックし、CRDオブジェクトをIPアドレスで更新します。CRDオブジェクトが更新されると、NetScaler Ingress Controllerはティア1 NetScaler VPXでNetScaler固有の構成を自動的に行います。

注:

VIP CRDはOpenShiftルートではサポートされていません。VIP CRDを使用するには、OpenShift Ingressを使用できます。

参照

- NetScaler IPAMコントローラーからのIPアドレスを使用してLoadBalancerタイプのサービスを公開する

© 1997–2026 Citrix Systems, Inc. All rights reserved.

HTTPルート

HTTPRoute は、コンテンツスイッチングのルーティング決定を定義するカスタムリソースです。現在、HTTPRoute は以下のルーティングをサポートしています。

- ホスト名ベースのルーティング
- パスベースのルーティング
- HTTPヘッダー名ベースのルーティング
- HTTPヘッダー値ベースのルーティング。
- Cookieベースのルーティング
- クエリパラメータベースのルーティング
- HTTPメソッドベースのルーティング
- NetScalerポリシー式を使用したルーティング

HTTPRouteオブジェクトの一部として、1つ以上のルールを定義でき、各ルールはルーティングのマッチング条件として機能します。受信HTTPリクエストに対してマッチング条件が満たされた場合、各ルールに対してアクションが定義されます。アクションは、トラフィックがバックエンドサービスにロードバランスされる「バックエンド」か、リダイレクト応答がクライアントに返送される「リダイレクト」のいずれかです。「バックエンド」アクションはADCでコンテンツスイッチングポリシーを作成し、「リダイレクト」アクションはADCでレスポンダーポリシーを作成します。

以下の表で説明されているように、マッチング条件には3つの異なる方法があります。

マッチング条件	説明
`exact`	受信リクエストと完全に一致します。この条件では大文字と小文字は区別されません。
`prefix`	受信リクエストのプレフィックスと一致します。この条件では大文字と小文字は区別されません。例: `/a` は `/a/b` および `/a/c` と一致しますが、`/c/a` とは一致しません。
`contains`	受信リクエストに指定されたキーワードが含まれている場合に一致します。この条件では大文字と小文字が区別されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

このトピックでは、HTTPRoute CRD オブジェクトのサンプルと、HTTPRoute CRD のさまざまな属性について説明します。完全な CRD 定義については、HTTPRoute.yaml を参照してください。

HTTP CRD オブジェクトの例

以下は、HTTP CRD オブジェクトのサンプルです。

```
apiVersion: citrix.com/v1

kind: HTTPRoute

metadata:

  name: test-route

  labels:

    domain: abc.com

spec:

  hostname:

  - abc.com

  rules:

  - name: exactpath

    match:

    - path:

        exact: /resources

    action:

      backend:

        kube:

          service: resource
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        port: 80
- name: prefixpath
  match:
  - path:
      prefix: /cart
  action:
    backend:
      kube:
        service: cart
        port: 80
- name: header
  match:
  - headers:
    - headerName:
        contains: Mobile
  action:
    backend:
      kube:
        service: mobile
        port: 443
        backendConfig:
          secureBackend: true
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            lbConfig:

              lbmethod: ROUNDROBIN
```

その他の例については、HTTP ルートの例 を参照してください。

HTTPRoute.spec

HTTPRoute カスタムリソースは、HTTP ルーティング仕様を表す spec フィールドを定義します。この仕様には、定義された各ルールのアクションを含むルールのリストがあります。

次の表は、`HTTPRoute.spec` 属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`hostname`	サーバーのホスト名リストを指定します。ホスト名は、RFC 1123 で定義されている有効なサブドメインである必要があります（例: test.example.com ）。 `*.example.com` の形式のワイルドカードホスト名も有効です。その場合、 `example.com` の任意のサブドメインがマッチングの対象となります。デフォルト値は `*` で、これはすべての受信 HTTP リクエストに一致することを意味します。	文字列	はい
`rules`	アクションに関連付けられた、マッチングルーティング基準	[] ルール	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
	を持つルールリストを指定します。		

HTTPRoute.rules

次の表は、`HTTPRoute.rules` 属性のさまざまなフィールドについて説明します。

フィールド	説明	型	必須
`name`	ルールを表す名前を指定します。このフィールドは、NetScaler のコンテンツルーティングポリシー名で識別子として使用されます。**注:** 各ルールについて、名前は一意である必要があります。	文字列	はい
	注: 各ルールについて、名前は一意である必要があります。		
`action`	一致するルールのアクションを指定します。	rules.action	はい
`match`	同じアクションを持つ一致するルートのリスト。複数のエントリが存在する場合、この一致ルールは `OR` 条件として扱われ、いずれかの一致に対して同じアクションが選択されます。	[ ] rules.match	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

HTTPRoute.rules.match

次の表は、`HTTPRoute.rules.match` 属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`path`	URLパスベースのルーティングルールを指定します。	HTTPRoute.rules.match.path	いいえ
`headers`	コンテンツルーティングのヘッダーベースの一致のリストを指定します。複数のルールがある場合、この一致条件は `AND` 条件として扱われ、すべてのルールが一致する必要があります。	[ ] HTTPRoute.rules.match.headers	いいえ
`cookies`	コンテンツルーティングのCookieベースの一致のリストを指定します。複数のルールがある場合、この一致条件は `AND` 条件として扱われ、すべてのルールが一致する必要があります。	[ ] HTTPRoute.rules.match.cookies	いいえ
`queryParams`	コンテンツルーティングのクエリパラメータのリストを指定します。複数のルールがある場合、このマッチング条件は `AND` 条件として扱われ、すべてのルールが一致する必要があります。	[ ] HTTPRoute.rules.match.queryParams	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
`method`	HTTPメソッドベースのルーティングルールを指定します。可能なオプションは、GET、POST、PUTなどです。一致するメソッドを持つHTTPリクエストに対してアクションが選択されます。	文字列	いいえ
`policyExpression`	NetScalerポリシー式ベースのルーティングルールを指定します。コンテンツルーティングルールには、任意のカスタムNetScalerポリシー式を指定できます。NetScaler Ingress Controllerは式の正確性をチェックしません。したがって、式の正確性を確認する必要があります。ポリシー式の詳細については、Expression Prefixを参照してください。例: `HTTP.REQ.URL.PATH.GET(1).EQ("foo")`	文字列	いいえ

HTTPRoute.rules.match.path

この属性は、コンテンツルーティングのパスベースのマッチングを指定します。

`HTTPRoute.rules.match.path`属性の例を次に示します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
     match:
     - path:
         prefix: /resources
     action:
       backend:
         kube:
           service: resource
           port: 80
---
     match:
     - path:
         regex: '/foo/[A-Z0-9]{3}'
     action:
       backend:
         kube:
           service: resource
           port: 80
```

次の表は、`HTTPRoute.rules.match.path`属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`prefix`	パスのプレフィックス式を照合基準として指定します。	文字列	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
	HTTPリクエストの開始パスが指定されたパスと一致する場合、照合を実行します。たとえば、`/a` はURL `/a` および `/a/b` と一致します。		
`exact`	正確なパスを照合基準として指定します。リクエストパスが指定されたパスと完全に一致する場合にのみ照合を実行します。	文字列	いいえ
`regex`	パスのマッチング条件として正規表現を指定します。指定された正規表現が受信リクエストと一致する場合にマッチングを実行します。Perl互換正規表現（PCRE）形式の正規表現のみがサポートされています。NetScalerでサポートされている正規表現の詳細については、正規表現を参照してください。	文字列	いいえ

HTTPRoute.rules.match.headers

この属性は、コンテンツルーティングのためのヘッダーベースのマッチングを表します。

次の表は、`HTTPRoute.rules.match.headers` 属性のさまざまなフィールドについて説明しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
`headerName`	コンテンツルーティングのマッチング条件としてヘッダー名を指定します。ヘッダーが存在する場合、マッチングに使用されます。この場合、ヘッダー値はマッチングの対象になりません。	HTTPRoute.rules.match.headers.headerName	いいえ
`headerValue`	コンテンツルーティングの一致条件としてヘッダー名と値を指定します。名前の場合は、正確な名前が一致条件となり、値の一致条件は、`exact`、`regex`、または`contains`式として指定できます。	HTTPRoute.rules.match.headers.headerValue	いいえ

HTTPRoute.rules.match.headers.headerName

この属性は、コンテンツルーティングのためのヘッダー名に基づいた一致を表します。

以下の例は、`HTTPRoute.rules.match.headers.headerName`属性の設定に関するサンプルスニペットを示しています。

```
match:

- headers:

  - headerName:

    exact: mobile

action:

  backend:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        kube:
          service: mobile-service
          port: 80
---
    match:
    - headers:
       - headerName:
         regex: "Header-[a-z]{1}"
    action:
      backend:
        kube:
          service: resource-service
          port: 80
```

以下の表は、`HTTPRoute.rules.match.headers.headerName`属性内のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`exact`	ルーティングの一致条件として、正確なヘッダー名を指定します。	文字列	いいえ
`contains`	`contains`文字列で指定された文字列を、ヘッダー名の照合基準として指定します。	文字列	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
`regex`	正規表現を照合基準として指定します。ヘッダー名が指定された正規表現と一致する場合に照合を実行します。PCRE形式の正規表現のみがサポートされています。	文字列	いいえ
`not`	この属性のデフォルト値はfalseです。この値がtrueの場合、ヘッダー名は受信リクエストに存在してはなりません。	ブール値	いいえ

HTTPRoute.rules.match.headers.headerValue

この属性は、コンテンツルーティングのためのヘッダー名と値の照合を表します。ヘッダー名は完全に一致し、値は指定されたフィールドに従って照合されます。

次の例は、`HTTPRoute.rules.match.headers.headerValue`属性設定のサンプルスニペットを示しています。

```
match:
- headers:
  - headerValue:
      name: Origin
      exact: mobile
      not: true
action:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      backend:
        kube:
          service: mobile
          port: 80
---
    match:
    - headers:
      - headerValue:
          name: Origin
          prefix: header1
    action:
      backend:
        kube:
          service: service1
          port: 80
---
    match:
    - headers:
      - headerValue:
          name: Origin
          regex: "[a-z]{1}"
    action:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      backend:
        kube:
          service: example
          port: 80
```

次の表は、`HTTPRoute.rules.match.headers.headerValue` 属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`name`	値と一致する必要があるヘッダーの名前を指定します。ヘッダーの値を照合するには、`exact`、`contains`、および `regex` フィールドが使用されます。`exact`、`contains`、および `regex` のいずれのフィールドも存在しない場合、その名前の任意の値が照合されます。	文字列	はい
`exact`	名前フィールドを持つHTTPヘッダーの値が正確に一致する場合に照合します。	文字列	いいえ
`contains`	名前フィールドを持つHTTPヘッダーの値が指定された文字列を含む場合に照合します。	文字列	いいえ
`regex`	nameフィールドを持つHTTPヘッダーの値が正規表現と一	文字列	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
	致する場合にマッチします。PCRE形式の正規表現のみがサポートされています。		
`not`	この属性のデフォルト値はfalseです。この値がtrueの場合、ヘッダー名は値と一致してはなりません。	ブール値	いいえ

HTTPRoute.rules.match.cookies

この属性は、コンテンツルーティングにおけるCookieベースのマッチングを表します。HTTPリクエストのCookieヘッダーがマッチングに使用されます。nameフィールドを持つCookieは、値が存在する場合にその値と照合されます。値が指定されていない場合は、任意の値と照合されます。

以下の例は、`HTTPRoute.rules.match.cookies`属性設定のサンプルスニペットを示しています。

```
    match:
    - cookies:
      - name: version
        contains: v1
    action:
      backend:
        kube:
          service: v1-app
          port: 80
---
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    match:
    - cookies:
      - name: version
        exact: v1
    action:
      backend:
        kube:
          service: v1-app
          port: 80
---
    match:
    - cookies:
      - name: version
        regex: '[a-z]{1}'
    action:
      backend:
        kube:
          service: v1-app
          port: 80
```

以下の表は、`HTTPRoute.rules.match.cookies`属性の様々なフィールドを説明しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
`name`	値がマッチングに使用されるクッキーの名前を指定します。`exact`、`regex`、`contains` のようなマッチング条件がクッキー名に存在しない場合、その名前が存在すれば、クッキー名の任意の値がマッチングされます。	文字列	はい
`exact`	名前フィールドを持つクッキーの値が完全に一致する場合にマッチします。	文字列	いいえ
`contains`	名前フィールドを持つクッキーの値が指定された文字列を含む場合にマッチします。	文字列	いいえ
`regex`	名前フィールドを持つCookieの値が正規表現と一致する場合にマッチします。PCRE形式の正規表現のみがサポートされています。	文字列	いいえ
`not`	この属性のデフォルト値はfalseです。この値がtrueの場合、名前を持つCookieは存在する必要がありますが、値と一致してはいけません。	ブール値	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

HTTPRoute.rules.match.queryParams

この属性は、コンテンツルーティングのためのURLマッチングにおけるHTTPクエリパラメータを表します。

```
    match:
     - queryParams:
       - name: version
         contains: v1
    action:
      backend:
        kube:
          service: v1-app
          port: 80
---
    match:
    - queryParams:
      - name: version
        regex: '[a-z]{1}'
    action:
      backend:
        kube:
          service: v1-app
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          port: 80

---

     match:

     - queryParams:

      - name: version

        exact: v1

        not: true

     action:

       backend:

        kube:

          service: mobile

          port: 80
```

次の例は、`HTTPRoute.rules.match.cookies`属性設定のサンプルスニペットを示しています。

フィールド	説明	型	必須
`name`	値が照合されるクエリパラメータの名前を指定します。`exact`、`regex`、`contains`のような条件がどれも存在しない場合、クエリパラメータの名前が存在すれば、その名前の任意の値が照合されます。	文字列	はい
`exact`	名前フィールドを持つクエリパラメータ	文字列	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	型	必須
	の値が完全に一致する場合に照合されます。		
`contains`	名前フィールドを持つクエリパラメータの値が指定された文字列を含む場合に照合されます。	文字列	いいえ
`regex`	名前フィールドを持つクエリパラメータの値が正規表現に一致する場合に照合されます。PCRE形式の正規表現のみがサポートされています。	文字列	いいえ
`not`	この属性のデフォルト値はfalseです。この値がtrueの場合、名前を持つクエリパラメータは存在する必要がありますが、値と一致してはいけません。	ブール値	いいえ

HTTPRoute.rules.action

この属性は、マッチングルールのアクションを表します。

次の表は、`HTTPRoute.rules.action`属性のさまざまなフィールドについて説明します。

フィールド	説明	タイプ	必須
`backend`	このフィールドのデフォルトアクションは、トラフィックをバックエンドサービスに送信することです。バックエンドま	rules.action.backend	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
	たはリダイレクトのいずれかが必要です。		
`redirect`	デフォルトのアクションはトラフィックをリダイレクトすることです。バックエンドまたはリダイレクトのいずれかが必要です。	rules.action.redirect	いいえ

HTTPRoute.rules.action.backend

この属性は、トラフィックをバックエンドサービスにルーティングすることを表します。 次の表は、`HTTPRoute.rules.action.backend` 属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`kube`	バックエンドサービス用のKubernetesサービス情報を指定します。	action.backend.kube	

HTTPRoute.rules.action.backend.kube

この属性は、デフォルトのバックエンドのKubernetesサービスを表します。サービスはHTTPRouteリソースと同じ名前空間に属している必要があります。サービスが `NodePort` または `Loadbalancer` タイプの場合、ポッドを持つそれらのノードのノードIPアドレスとNodePortのリストが、7でバックエンドサービスとして使用されます。

`HTTPRoute.rules.action.backend.kube` 属性の例を次に示します。

```
kube:

  service: service

  namespace: default
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  port: 80

  backendConfig:

    lbConfig:

      lbmethod: ROUNDROBIN

    servicegroupConfig:

      clttimeout: '20'
```

次の表は、`HTTPRoute.rules.action.backend.kube`属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`service`	デフォルトのバックエンドのKubernetesサービスの名前を指定します。	文字列	はい
`port`	デフォルトのバックエンドのKubernetesサービスのポート番号を指定します。	整数	はい
`backendConfig`	デフォルトのバックエンドのバックエンド構成を指定します。	BackendConfig	いいえ

BackendConfig

この属性は、NetScaler のバックエンド構成を表します。 以下は、`BackendConfig` 属性構成の例です。

```
backendConfig:

 sercureBackend: true
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
lbConfig:

  lbmethod: ROUNDROBIN

servicegroupConfig:

  clttimeout: '20'
```

次の表は、`BackendConfig` 属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`secureBackend`	通信がセキュアかどうかを指定します。`secureBackend` フィールドの値が `true` の場合、バックエンドとの通信にはセキュアな通信が使用されます。デフォルト値は `false` で、これはバックエンド通信に HTTP が使用されることを意味します。		
`lbConfig`	指定されたバックエンドの NetScaler ロードバランシング仮想サーバー構成を指定します。例に示すようにキーと値のペアを指定して、バックエンドの LBVserver 構成を設定できます。すべての有効な構成については、LB virtual server configurations を参照してください。	オブジェクト	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
`servicegroupConfig`	指定されたバックエンドのNetScalerサービスグループ構成を指定します。例に示すように、バックエンドのサービスグループ構成を設定するキーと値のペアを指定できます。すべての有効な構成については、サービスグループ構成を参照してください。	オブジェクト	いいえ

HTTPRoute.rules.action.redirect

この属性はリダイレクトアクションを表します。

```
action:

  redirect:

   httpsRedirect: true

   responseCode: 302
```

次の表は、`HTTPRoute.rules.action.redirect`属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`httpsRedirect`	このフィールドが`yes`に設定されている場合、HTTPトラフィックをHTTPSにリダイレクトします。他のURL部分を変更せずに、スキームのみがHTTPSに変更されます。	ブール値	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
	`httpsRedirect`、`hostRedirect`、または `targetExpression` のいずれかが必要です。		
`hostRedirect`	URLのホスト名部分をこの属性に設定された値に書き換え、トラフィックをリダイレクトします。URLの他の部分はリダイレクト中に変更されません。	文字列	いいえ
`targetExpression`	リダイレクト用のNetScaler式を指定します。例えば、HTTPからHTTPSへトラフィックをリダイレクトするには、以下の式を使用できます: ""https://"+HTTP.REQ.HOSTNAME + HTTP.REQ.URL.HTTP_URL_SAFE"。	文字列	いいえ
`responseCode`	レスポンスコードを指定します。デフォルトのレスポンスコードは302で、この属性を使用してカスタマイズできます。	整数	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScalerによるKubernetes向け高度なコンテンツルーティング

Kubernetesネイティブのingressは、基本的なホストベースおよびパスベースのルーティングを提供します。しかし、ヘッダー値やクエリ文字列に基づくルーティングのような他の高度なルーティング技術は、Ingress構造ではサポートされていません。これらの機能は、Ingressアノテーションを介してKubernetes Ingress上で公開できますが、アノテーションは管理と検証が複雑です。

NetScaler®が提供する高度なコンテンツルーティング機能を、Kubernetes向けのカスタムリソース定義 (CRD) APIとして公開できます。

コンテンツルーティングCRDを使用すると、次のパラメーターに基づいてトラフィックをルーティングできます。

- ホスト名
- URLパス
- HTTPヘッダー
- クッキー
- クエリパラメーター
- HTTPメソッド
- NetScalerポリシー式

注:

Ingressリソースとコンテンツルーティング CRDは、同じサービス (IPアドレスとポート) に対して共存できません。Ingressとコンテンツルーティング CRDの併用はサポートされていません。

高度なコンテンツルーティング機能は、次のCRDを使用してKubernetesで公開されます。

- リスナー
- HTTPルート

リスナー CRD

Listener CRDオブジェクトは、仮想IPアドレス、ポート、証明書、その他のフロントエンド構成などのエンドポイント情報を表します。また、デフォルトのトラフィックをバックエンドに送信したり、トラフィックをリダイレクトしたりするなどのデフォルトアクションも定義します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Listener CRDオブジェクトは、受信HTTPリクエストのHTTPルーティングロジックを表すHTTPRoute CRDオブジェクトを参照できます。

完全なCRD定義については、Listener CRDを参照してください。 Listener CRDのすべての属性に関する完全な情報については、Listener CRDドキュメントを参照してください。

Listener CRDは、HTTP、SSL、およびTCPプロファイルをサポートしています。これらのプロファイルを使用すると、デフォルトのプロトコル動作をカスタマイズできます。Listener CRDは、NetScalerがトランザクションまたはデータの種類を異なるエンドポイントにエクスポートできるようにするアナリティクスプロファイルもサポートしています。 Listener CRDのプロファイルサポートの詳細については、Listener CRDのプロファイルサポートを参照してください。

Listener CRDのデプロイ

1. Listener CRDをダウンロードします。
2. 次のコマンドでListener CRDをデプロイします。

```
Kubectl create -f  Listener.yaml
```

例:

```
root@k8smaster:# kubectl create -f Listener.yaml

customresourcedefinition.apiextensions.k8s.io/listeners.citrix.com
created
```

Listener CRDオブジェクトの記述方法

KubernetesクラスターにNetScalerが提供するCRDをデプロイした後、YAMLファイルでリスナー構成を定義できます。YAMLファイルでは、kindフィールドに`Listener`を使用し、specセクションにリスナー構成の要件に基づいてListener CRD属性を追加します。 YAMLファイルをデプロイすると、NetScaler Ingress ControllerはNetScalerにリスナー構成を適用します。

以下は、`Listener-crd.yaml`という名前のListener CRDオブジェクト定義のサンプルです。

```
apiVersion: citrix.com/v1

kind: Listener

metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: my-listener

  namespace: default

spec:

  certificates:

  - secret:

      name: my-secret

    # Secret named 'my-secret' in current namespace bound as default
certificate

    default: true

  - secret:

      # Secret 'other-secret' in demo namespace bound as SNI certificate

      name: other-secret

      namespace: demo

  - preconfigured: second-secret

    # preconfigured certkey name in ADC

  vip: '192.168.0.1' # Virtual IP address to be used, not required when
CPX is used as ingress device

  port: 443

  protocol: https

  redirectPort: 80

  secondaryVips:

  - "10.0.0.1"

  - "1.1.1.1"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  policies:
    httpprofile:
      config:
        websocket: "ENABLED"
    tcpprofile:
      config:
        sack: "ENABLED"
    sslprofile:
      config:
        ssl3: "ENABLED"
    sslciphers:
    - SECURE
    - MEDIUM
    analyticsprofile:
      config:
      - type: webinsight
        parameters:
           allhttpheaders: "ENABLED"
    csvserverConfig:
      rhistate: 'ACTIVE'
  routes:
    # Attach the policies from the below Routes
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  - name: domain1-route
    namespace: default
  - name: domain2-route
    namespace: default
  - labelSelector:
      # Attach all HTTPRoutes with label route=my-route
      route: my-route
  # Default action when traffic matches none of the policies in the
HTTPRoute
  defaultAction:
    backend:
      kube:
        namespace: default
        port: 80
        service: default-service
        backendConfig:
          lbConfig:
            # Use round robin LB method for default service
            lbmethod: ROUNDROBIN
          servicegroupConfig:
            # Client timeout of 20 seconds
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            clttimeout: "20"
```

この例では、リスナーがHTTPSエンドポイントを公開しています。証明書セクションでは、エンドポイントのSSL証明書は、`my-secret`と`other-secret`という名前のKubernetesシークレットと、`second-secret`という名前のcertkeyを持つデフォルトのADC事前構成済み証明書を使用して構成されます。リスナーのデフォルトアクションは、Kubernetesサービスとして構成されます。ルートは、ラベルセレクターと、名前および名前空間を使用した個別のルート参照の両方を使用して、リスナーにアタッチされます。

YAMLファイルでListener CRDオブジェクトを定義したら、次のコマンドを使用してYAMLファイルをデプロイします。この例では、`Listener-crd.yaml`がYAML定義です。

```
Kubectl create -f  Listener-crd.yaml
```

HTTPRoute カスタムリソース定義

HTTPRoute CRDオブジェクトは、受信HTTPリクエストのHTTPルーティングロジックを表します。ホスト名、パス、ヘッダー、クエリパラメータ、Cookieなど、さまざまなHTTPパラメータを組み合わせて、受信トラフィックをバックエンドサービスにルーティングできます。HTTPRouteオブジェクトは、エンドポイント情報を表す1つ以上のListenerオブジェクトにアタッチできます。HTTPRouteオブジェクトには1つ以上のルールを設定でき、各ルールには関連付けられたアクションが指定されます。HTTPRouteオブジェクト内のルールの評価順序は、オブジェクトに記述されている順序と同じです。例えば、rule1とrule2という順序の2つのルールがあり、rule1がrule2の前に記述されている場合、rule2の前にrule1が最初に評価されます。

HTTPRoute CRDの定義はHTTPRoute.yamlで入手できます。HTTP Route CRDの属性に関する完全な情報については、HTTPRoute CRD documentationを参照してください。

現在、NetScalerは、Kubernetes Ingressバージョン`networking.k8s.io/v1`のIngressで、HTTPルートCRDリソースをリソースバックエンドとして構成することをサポートしています。この機能により、高度なコンテンツルーティング機能をIngressに拡張できます。詳細については、Advanced content routing for Kubernetes Ingress using HTTPRoute CRDを参照してください。

HTTPRoute CRD を展開する

HTTPRoute CRD を展開するには、以下を実行します。

1. HTTPRoute.yamlをダウンロードします。
2. 次のコマンドを使用して、クラスターにHTTPRoute CRDを適用します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Kubectl apply -f HTTPRoute.yaml を実行します。

例:

```
root@k8smaster:# kubectl create -f HTTPRoute.yaml

customresourcedefinition.apiextensions.k8s.io/httproutes.citrix.com
configured
```

HTTPRoute CRD オブジェクトの記述方法

HTTPRoute CRDを展開したら、YAMLファイルでHTTPルート構成を定義できます。YAMLファイルでは、kindフィールドに`HTTPRoute`を使用し、specセクションにHTTPルート構成の要件に基づいてHTTPRoute CRD属性を追加します。

以下は、`Route-crd.yaml`という名前のHTTPRoute CRDオブジェクト定義のサンプルです。

```
apiVersion: citrix.com/v1

kind: HTTPRoute

metadata:

   name: test-route

spec:

  hostname:

  - host1.com

  rules:

  - name: header-routing

    match:

    - headers:

      - headerName:

          exact: my-header
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    action:
      backend:
        kube:
          service: mobile-app
          port: 80
          backendConfig:
            secureBackend: true
            lbConfig:
              lbmethod: ROUNDROBIN
  - name: path-routing
    match:
    - path:
        prefix: /
    action:
      backend:
        kube:
          service: default-app
          port: 80
```

この例では、`my-header`に一致するヘッダー名を持つリクエストはmobile-appサービスにルーティングされ、その他のすべてのトラフィックはdefault-appサービスにルーティングされます。HTTPRouteの詳細な説明とAPI仕様については、HTTPRoute CRDを参照してください。

YAMLファイルでHTTPルートを定義したら、次のコマンドを使用してHTTPRoute CRDオブジェクトのYAMLファイルを展開します。この例では、`Route-crd.yaml`がYAML定義です。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
Kubectl create -f  Route-crd.yaml
```

YAMLファイルをデプロイすると、NetScaler Ingress ControllerはIngress NetScalerデバイスにHTTPルート設定を適用します。

HTTPRoute CRDオブジェクトをListener CRDオブジェクトにアタッチする

HTTPRoute CRDオブジェクトをListener CRDオブジェクトにアタッチするには、次の2つの方法があります。

- 名前と名前空間を使用する
- ラベルとセレクターを使用する

名前と名前空間を使用してHTTPRoute CRDオブジェクトをアタッチする

このアプローチでは、Listener CRDオブジェクトは、`routes` セクションで名前と名前空間を指定することにより、1つ以上のHTTPRouteオブジェクトを明示的に参照します。 HTTPRouteオブジェクトの評価順序は、Listener CRDオブジェクトで指定された順序と同じであり、最初のHTTPRouteオブジェクトが最初に評価され、その後も同様です。

例えば、Listener CRDオブジェクトの抜粋を以下に示します。

```
routes:

 - name: route-1

   namespace: default

 - name: route-2

   namespace: default
```

この例では、route1という名前のHTTPRoute CRDオブジェクトは、route2という名前のHTTPRouteよりも先に評価されます。

ラベルとセレクターを使用してHTTPRoute CRDオブジェクトをアタッチする

ラベルとセレクターを使用して、HTTPRouteオブジェクトをListenerオブジェクトにアタッチすることもできます。Listener CRDオブジェクトに1つ以上のラベルを指定できます。ラベルに一致するHTTPRouteオブジェクトは、自動的にListenerオブジェクトにリンクされ、NetScalerでルールが作成されます。このアプローチを使用する場合、複数のHTTPRouteオブジェクト間に特定の評価順序はありません。唯一の例外は、デフォルトルート（ホスト名のみ、または「/」パスを持つ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ルート）を持つHTTPRouteオブジェクトであり、これは最後のオブジェクトとして評価されます。

例えば、リスナーリソースの抜粋は次のとおりです。

```
routes:

- labelSelector:

    team: team1
```

リスナー

リスナーCRDは、コンテンツスイッチング負荷分散仮想サーバーのエンドポイント情報を表します。このトピックには、リスナーCRDオブジェクトのサンプルが含まれており、リスナーCRDのさまざまな属性についても説明しています。完全なCRD定義については、Listener.yamlを参照してください。

注:

イングレスリソースとコンテンツルーティングCRD（リスナーCRDおよびHTTPRoute CRD）は、同じエンドポイント（IPアドレスとポート）で共存できません。イングレスでのコンテンツルーティングCRDの使用はサポートされていません。

リスナーCRDオブジェクトの例

以下は、リスナーCRDオブジェクトの例です。

```
apiVersion: citrix.com/v1

kind: Listener

metadata:

  name: my-listener

  namespace: default

spec:

  certificates:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  - secret:

      name: my-secret

    # Secret named 'my-secret' in current namespace bound as default
certificate

    default: true

  - secret:

      # Secret 'other-secret' in demo namespace bound as SNI certificate

      name: other-secret

      namespace: demo

  - preconfigured: second-secret

    # preconfigured certkey name in ADC

  vip: '192.168.0.1' # Virtual IP address to be used, not required when
CPX is used as ingress device

  port: 443

  protocol: https

  redirectPort: 80

  secondaryVips:

  - "10.0.0.1"

  - "1.1.1.1"

  policies:

    httpprofile:

      config:

        websocket: "ENABLED"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    tcpprofile:
      config:
        sack: "ENABLED"
    sslprofile:
      config:
        ssl3: "ENABLED"
    sslciphers:
    - SECURE
    - MEDIUM
    analyticsprofile:
      config:
      - type: webinsight
        parameters:
          allhttpheaders: "ENABLED"
    csvserverConfig:
      rhistate: 'ACTIVE'
  routes:
    # Attach the policies from the below Routes
  - name: domain1-route
    namespace: default
  - name: domain2-route
    namespace: default
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  - labelSelector:

      # Attach all HTTPRoutes with label route=my-route

      route: my-route

  # Default action when traffic matches none of the policies in the
HTTPRoute

  defaultAction:

    backend:

      kube:

        namespace: default

        port: 80

        service: default-service

        backendConfig:

          lbConfig:

            # Use round robin LB method for default service

            lbmethod: ROUNDROBIN

          servicegroupConfig:

            # Client timeout of 20 seconds

            clttimeout: "20"
```

その他の例については、リスナーの例を参照してください。

Listener.spec

`Listener.spec`属性は、リスナーカスタムリソースの仕様を定義します。次の表は、`Listener.spec`属性のさまざまなフィールドについて説明しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ ;	必須
`protocol`	負荷分散コンテンツスイッチング仮想サーバーのプロトコルを指定します。許可される値は、`tcp`、`udp`、`http`、および`https`です。	文字列	はい
`port`	ロードバランシングコンテンツスイッチング仮想サーバーのポート番号を指定します。HTTPプロトコルのデフォルトポート番号は80、HTTPSプロトコルは443です。	整数	いいえ
`routes`	Listenerリソースにアタッチされる HTTPRouteリソースのリストを指定します。評価順序はリストの順序に従います。つまり、複数のエントリが存在する場合、リストで最初に指定されたルートが最も高い優先順位を持ち、以降も同様です。	[ ] ルート	いいえ
`ingressclass`	指定されたイングレスクラスに関連付けられたイングレスコントローラーのみがリソースを処理するように、イングレスクラスを指定しま	文字列	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ ;	必須
	す。そうでない場合、クラスター内のすべてのコントローラーがこのリソースを処理します。		
`certificates`	プロトコルがHTTPSの場合、SSL仮想サーバーの証明書リストを指定します。プロトコルがHTTPSの場合、このフィールドは必須です。	[ ] 証明書	いいえ
`vip`	ロードバランシングコンテンツスイッチング仮想サーバーのエンドポイントIPアドレスを指定します。このアドレスはNetScaler VPXおよびMPXデバイスには必須ですが、Kubernetesクラスターに存在するNetScaler CPXには必須ではありません。NetScaler CPXの場合、`vip` はCNIによって割り当てられたNetScaler CPXのプライマリIPアドレスと同じです。	文字列	いいえ
`defaultAction`	`routes` で指定されたHTTPRouteリソースのいずれもトラフィックに一致しない場合に実行する	アクション	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ ;	必須
	デフォルトのアクションを指定します。		
`policies`	フロントエンド仮想サーバーに関連付けられたHTTP、TCP、およびSSLポリシーをカスタマイズできるオプションを指定します。	[ ] Listener.policies	いいえ
`redirectPort`	このポートのHTTPトラフィックがHTTPSポートにリダイレクトされることを指定します。	整数	いいえ
`secondaryVips`	プライマリVIPとともにVIPとして使用されるIPアドレスのセットを指定します。IPセットが作成され、これらのVIPがIPセットに追加されます。	[ ] 文字列	いいえ
`multicluster`	コンテンツスイッチング仮想サーバーのIPアドレスが、マルチクラスターとNetScaler Ingress Controllerによって共有されるかどうかを指定します。	ブール値	いいえ
`passthrough`	SSLトラフィックのSSLパススルーを有効にします。トラフィックは復号化されずにバックエンド	ブール値	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ ;	必須
	に転送されます。リスナーがイングレスアノテーションで参照されている場合にのみ利用可能で、ルートでは使用できません。		

Listener.certificates

`Listener.certificates` 属性は、SSL仮想サーバーのTLS証明書関連情報を定義します。

`Listener.certificates` 属性の例を次に示します。

```
certificates:

- secret:

    name: my-secret

    namespace: demo

  default: true

- preconfigured: configured-secret
```

次の表は、`Listener.certificates` 属性のさまざまなフィールドについて説明しています。

フィールド	説明	型	必須
`secret`	Kubernetesシークレットリソースを介して指定されたTLS証明書を指定します。シークレットには、キー `tls.crt` と `tls.key` が含まれている必要があります。これらのキーには、証明書と秘密	certificates.secret	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	型	必須
	鍵が含まれています。シークレットまたは事前設定されたフィールドのいずれかが必要です。すべての証明書は、SNI証明書としてSSL仮想サーバーにバインドされます。		
`preconfigured`	NetScalerで事前設定されたTLS `certkey` の名前を指定します。このフィールドは、Tier-1 VPXおよびMPXデバイスにのみ適用されます。 `certkey` は、Listenerリソースの実際のデプロイメントの前に存在している必要があります。そうでない場合、リソースのデプロイメントはエラーで失敗します。NetScaler Ingress Controllerは `certkey` のライフサイクルを管理しません。したがって、 `certkey` の追加または削除は手動で管理する必要があります。シークレットまたは事前設定されたフィールドのいずれかが必要です。	文字列	いいえ
`default`	デフォルト証明書を指定します。証明書	ブール値	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	型	必須
	のうち1つだけをデフォルトとしてマークできます。仮想サーバーがSNIフィールドなしでトラフィックを受信した場合、デフォルト証明書が提示されます。この証明書は、IPアドレスを使用してHTTPSアプリケーションにアクセスするために使用できます。適用可能な値は `true` と `false` です。		

Listener.certificates.Secret

この属性は、SSL仮想サーバーにバインドする必要があるTLS証明書用のKubernetesシークレットリソースを表します。

次の表は、`Listener.certificate.Secret` 属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`name`	Kubernetesシークレットリソースの名前を指定します。シークレットには、`tls.crt` と `tls.key` という名前のキーが含まれている必要があります。これらのキーには、証明書と秘密鍵が含まれています。シークレットオブジェクトに複数の `tls.crt` フィール	文字列	はい

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
	ドが存在する場合、最初の証明書はサーバー証明書と見なされ、残りの証明書は中間CA証明書と見なされます。また、証明書はサーバー証明書から開始して、互いに再帰的にリンクされます。		
`namespace`	Kubernetesシークレットリソースの名前空間を指定します。この値が指定されていない場合、名前空間はListenerリソースと同じと見なされます。	文字列	いいえ

Listener.routes

この属性は、Listenerリソースにアタッチされ ているHTTPRouteオブジェクトのリストを表します。 次の表は、`Listener.routes`属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`name`	バックエンドサーバーへのルーティング決定のために評価されるHTTPRouteリソースの名前を指定します。`name`または`labelSelector`のいずれかが必要です。	文字列	いいえ
`namespace`	HTTPRouteリソースの名前空間を指定します。デフォルト値	文字列	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
	はListenerリソースの名前空間です。		
`labelSelector`	HTTPRouteリソースのラベルセレクターを指定します。このフィールドは、HTTPRouteリソースをアタッチする別の方法を提供します。このセレクターに一致するラベルキーと値を持つHTTPRouteオブジェクトは、リスナーリソースに自動的にアタッチされます。ルートがlabelSelectorを介してアタッチされる場合、ルートは特定の順序なしでアタッチされます。このルールの例外は、常に最後にアタッチされるデフォルトパス ('/') を持つルートです。例に示すように、ラベルを持つHTTPRouteオブジェクトはリスナーオブジェクトにアタッチされます。ラベルとセレクターの詳細については、Kubernetes Documentationを参照してください。	オブジェクト	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Listener.action

この属性は、ロードバランシング仮想サーバーへのリクエストがListener.routesフィールドに示されているルートオブジェクトのいずれにも一致しない場合のデフォルトアクションを表します。

次の表は、`Listener.action`属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`backend`	このフィールドのデフォルトアクションは、トラフィックをバックエンドサービスに送信することです。バックエンドまたはリダイレクトのいずれかが必要です。	action.backend	いいえ
`redirect`	デフォルトアクションはトラフィックをリダイレクトすることです。バックエンドまたはリダイレクトのいずれかが必要です。	action.redirect	いいえ

Listener.action.backend

この属性は、デフォルトアクションのバックエンドサービスを指定します。 次の表は、`Listener.action.backend`属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`kube`	バックエンドサービス用のKubernetesサービス情報を指定します。	action.backend.kube	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Listener.action.backend.kube

この属性は、デフォルトのバックエンドに対するKubernetesバックエンドサービスを表します。サービスが`NodePort`または`Loadbalancer`タイプの場合、ノードIPアドレスとNodePortがバックエンドへのトラフィックの送信に使用されます。

`Listener.action.backend.kube`属性の例を次に示します。

```
kube:

  service: default-service

  namespace: default

  port: 80

  backendConfig:

    lbConfig:

      lbmethod: ROUNDROBIN

    servicegroupConfig:

      clttimeout: '20'
```

次の表は、`Listener.action.backend.kube`属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`service`	デフォルトのバックエンドに対するKubernetesサービスの名前を指定します。	文字列	はい
`namespace`	デフォルトバックエンドのKubernetesサービスのネームスペースを指定します。	文字列	はい

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
`port`	デフォルトバックエンドのKubernetesサービスのポート番号を指定します。	整数	はい
`backendConfig`	デフォルトバックエンドのバックエンド構成を指定します。	BackendConfig	いいえ

BackendConfig

この属性は、NetScalerのバックエンド構成を表します。 `BackendConfig`属性の例を次に示します。

```
backendConfig:

 sercureBackend: true

 lbConfig:

   lbmethod: ROUNDROBIN

 servicegroupConfig:

   clttimeout: '20'
```

次の表は、 `BackendConfig`属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`secureBackend`	通信が安全かどうかを指定します。 `secureBackend` フィールドの値が `true`の場合、バックエンドとの通信には安全な通信が使用されます。デフォルト値は`false`で、これはバックエンド		

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
	通信にHTTPが使用されることを意味します。		
`lbConfig`	指定されたバックエンドのNetScalerロードバランシング仮想サーバー構成を指定します。例に示すように、バックエンドのLBVserver構成を設定するキーと値のペアを指定できます。有効なすべての構成については、LB仮想サーバー構成を参照してください。	オブジェクト	いいえ
`servicegroupConfig`	指定されたバックエンドのNetScalerサービスグループ構成を指定します。例に示すように、バックエンドのサービスグループ構成を設定するキーと値のペアを指定できます。有効なすべての構成については、サービスグループ構成を参照してください。	オブジェクト	いいえ

Listener.action.redirect

```
defaultAction:

  redirect:

   httpsRedirect: true
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
responseCode: 302
```

次の表は、`Listener.action.redirect` 属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`httpsRedirect`	このフィールドが `yes` に設定されている場合、HTTPトラフィックをHTTPSにリダイレクトします。他のURL部分を変更せずに、スキームのみがHTTPSに変更されます。`httpsRedirect`、`hostRedirect`、または `targetExpression` のいずれかが必要です。	ブール値	いいえ
`hostRedirect`	URLのホスト名部分をここで設定された値に書き換え、トラフィックをリダイレクトします。リダイレクト中にURLの他の部分は変更されません。	文字列	いいえ
`targetExpression`	リダイレクト用のNetScaler式を指定します。たとえば、HTTPからHTTPSにトラフィックをリダイレクトするには、次の式を使用できます: *""https://"+HTTP.REQ.HOSTNAME +*	文字列	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
	HTTP.REQ.URL.HTTP_URL_SAFE"。		
`responseCode`	応答コードを指定します。デフォルトの応答コードは302ですが、この属性を使用してカスタマイズできます。	整数	いいえ

Listener.policies

この属性は、ポリシーが指定されていない場合にリスナーに使用されるデフォルトポリシーを表します。`Listener.policies`を使用することで、TCP、HTTP、およびSSLの動作をカスタマイズできます。

`Listener.policies`属性の例を次に示します。

```
policies:

 httpprofile:

  config:

   websocket: "ENABLED"

 tcpprofile:

  config:

   sack: "ENABLED"

 sslprofile:

  config:

   ssl3: "ENABLED"

 sslciphers:

  - HIGH
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
- MEDIUM

analyticsprofile:

 config:

 - type: webinsight

   parameters:

    allhttpheaders: "ENABLED"

csvserverConfig:

 rhistate: 'ACTIVE'

 stateupdate: ‘ENABLED’
```

次の表は、`Listener.policies`属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`httpprofile`	フロントエンド仮想サーバーのHTTP設定を指定します。	Listener.policies.httpprofile	いいえ
`tcpprofile`	フロントエンド仮想サーバーのTCP設定を指定します。	Listener.policies.tcpprofile	いいえ
`sslprofile`	フロントエンド仮想サーバーのSSL設定を指定します。	Listener.policies.sslprofile	いいえ
`sslciphers`	SSLプロファイルにバインドされる暗号のリストを指定します。優先順位はリストで指定された順序で、リストの最初にあるものに高い優先順位が与えられま	[ ] 文字列	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
	す。このフィールドでは、NetScalerで利用可能な任意のSSL暗号、またはユーザーが作成した暗号グループを使用できます。NetScalerで利用可能な暗号のリストについては、NetScalerの暗号を参照してください。		
`analyticsprofile`	フロントエンド仮想サーバーの分析プロファイル設定を指定します	Listener.policies.analyticsprofile	いいえ
`csvserverConfig`	リスナーのフロントエンドCS仮想サーバー設定を指定します。フロントエンドのCS仮想サーバー設定を行う例に示されているように、キーと値のペアを指定できます。	オブジェクト	いいえ

Listener.policies.tcpprofile

この属性は、フロントエンドCS仮想サーバーのTCPプロファイル設定を表します。

`Listener.policies.tcpprofile`属性の例を次に示します。

```
    policies:

     tcpprofile:

      config:

       sack: "ENABLED"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
       nagle: “ENABLED”

---

    policies:

     tcpprofile:

      preconfigured: test-tcp-profile
```

次の表は、`Listener.policies.tcpprofile`属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`preconfigured`	フロントエンドCS仮想サーバーに使用される、事前設定されたTCPプロファイルの名称を指定します。このプロファイルは、ポリシーを適用する前にNetScalerに存在している必要があります。そうでない場合、Listenerリソースの適用は失敗します。`preconfigured`または`config`のいずれかが必要です。	文字列	いいえ
`config`	フロントエンド仮想サーバーのTCPプロファイル設定を指定します。仮想サーバーのTCP特性を調整するために、例に示すようにキーと値のペアを指定できます。	オブジェクト	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Listener.policies.httpprofile

この属性は、フロントエンドCS仮想サーバーのHTTP構成を表します。

`Listener.policies.httpprofile`属性の例を次に示します。

```
    policies:

     httpprofile:

      config:

       websocket: "ENABLED"

---

    policies:

     httpprofile:

     preconfigured: test-http-profile
```

次の表は、`Listener.policies.httpprofile`属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`preconfigured`	フロントエンドCS仮想サーバーに使用される、事前設定されたHTTPプロファイルの名称を指定します。このプロファイルは、ポリシーを適用する前にNetScalerに存在している必要があります。そうでない場合、Listenerリソースの適用は失敗します。`preconfigured` ま	文字列	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
	たは `config` のいずれかが必要です。		
`config`	フロントエンド仮想サーバーのHTTPプロファイル設定を指定します。例に示すようにキーと値のペアを指定して、仮想サーバーのHTTPプロトコル特性を調整できます。	オブジェクト	いいえ

Listener.policies.sslprofile

この属性は、フロントエンドCS仮想サーバーのSSLプロファイルを表します。

`Listener.policies.sslprofile` 属性の例を次に示します。

```
    policies:

     sslprofile:

      config:

       ssl3: "ENABLED"

---

    policies:

     sslprofile:

      preconfigured: test-ssl-profile
```

次の表は、`Listener.policies.sslprofile` 属性のさまざまなフィールドについて説明しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
`preconfigured`	フロントエンドSSL仮想サーバーに使用される、事前設定されたSSLプロファイルの名称を指定します。このプロファイルは、ポリシーを適用する前にNetScalerに存在している必要があります。そうでない場合、リスナーリソースの適用は失敗します。`preconfigured`または`config`のいずれかが必要です。	文字列	いいえ
`config`	フロントエンド仮想サーバーのSSLプロファイル設定を指定します。例に示すようにキーと値のペアを指定することで、仮想サーバーのSSL特性を調整できます。	オブジェクト	いいえ
	注: 高度なSSL機能を使用するには、NetScalerで`set ssl parameter -defaultProfile ENABLED`コマンドを使用してデフォルトプロファイルを有効にする必要があります。		

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Listener.policies.analyticsprofile

この属性は、カウンターとメトリクスをNetScaler Observability Exporterにエクスポートするために使用されるアナリティクスプロファイルを表します。この属性を設定することで、アナリティクスプロファイルを作成してバインドすることにより、エクスポートする内容を選択できます。

`Listener.policies.analyticsprofile`属性の例を次に示します。

```
    policies:

     analyticsprofile:

      config:

      - type: webinsight

        parameters:

         allhttpheaders: "ENABLED"

---

    policies:

     analyticsprofile:

      preconfigured:

      - test-analytics-profile

      - test2-analytics-profile
```

次の表は、`Listener.policies.analyticsprofile` 属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`preconfigured`	フロントエンド仮想サーバーにバインドする必要がある、事前設定されたアナリティクスプロファイルのリストを指定し	[] 文字列	いいえ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
	ます。これらのプロファイルは、ポリシーを適用する前にNetScalerに存在している必要があります。そうでない場合、Listenerリソースの適用は失敗します。`preconfigured` または `config` のいずれかが必要です。		
`config`	フロントエンド仮想サーバーにバインドされるアナリティクスプロファイルのリストを指定します。これにより、NetScaler Observability Exporterにエクスポートされるフィールドが決定されます。	[ ] Listener.policies.analyticsprofile.config	いいえ

Listener.policies.analyticsprofile.config

この属性は、エクスポートする必要があるさまざまな種類のインサイトおよびHTTPヘッダーパラメーターのアナリティクスプロファイル構成を表します。

次の表は、`Listener.policies.analyticsprofile.config` 属性のさまざまなフィールドについて説明しています。

フィールド	説明	タイプ	必須
`type`	有効にする分析プロファイルのタイプを決定するタイプを指定します。次のタイプの1つ以上を有効	文字列	はい

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールド	説明	タイプ	必須
	にできます: `webinsight`, `tcpinsight`, `securityinsight`, `videoinsight`, `hdxinsight`, `gatewayinsight`, `timeseries`, `lsninsight`, および `botinsight`		
`parameters`	分析プロファイル の一部として有効にする追加パラメーターを指定します。例に示すように、キーと値のペアを指定できます。たとえば、このフィールドを使用して、`webinsight` の一部としてエクスポートするHTTPパラメーターを選択できます。	オブジェクト	いいえ

NetScaler Ingress Controller を使用して Web アプリケーションファイアウォールポリシーを構成する

NetScaler は、Kubernetes 用の WAF CRD と呼ばれるカスタムリソース定義 (CRD) を提供します。WAF CRD を使用して、NetScaler VPX、MPX、SDX、および CPX 上の NetScaler Ingress Controller で Web アプリケーションファイアウォールポリシーを構成できます。WAF CRD は、Web アプリケーションファイアウォールポリシーを適用するために、NetScaler Ingress Controller と NetScaler 間の通信を可能にします。

Kubernetes デプロイでは、WAF CRD を使用して、サーバーを保護するための Web アプリケーションファイアウォールポリシーを適用できます。Web アプリケーションファイアウォールの詳細については、「Web アプリケーションセキュリティ」を参照してください。

WAF CRD を使用すると、Kubernetes ネイティブアプリケーションに対して次の種類のセキュリティチェックを適用するようにファイアウォールセキュリティポリシーを構成できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

一般的な保護

- バッファオーバーフロー
- コンテンツタイプ
- URL の許可
- URL のブロック
- Cookie の一貫性
- クレジットカード

HTML 保護

- CSRF (クロスサイトリクエストフォージェリ) フォームタグ付け
- フィールド形式
- フォームフィールドの一貫性
- ファイルアップロードの種類
- HTMLクロスサイトスクリプティング
- HTML SQLインジェクション

JSON保護

- JSONサービス拒否
- JSON SQLインジェクション
- JSONクロスサイトスクリプティング

XML保護

- XML Webサービス相互運用性
- XML添付ファイル
- XMLクロスサイトスクリプティング
- XMLサービス拒否
- XML形式
- XMLメッセージ検証
- XML SOAP障害フィルタリング
- XML SQLインジェクション

© 1997–2026 Citrix Systems, Inc. All rights reserved.

セキュリティチェックの種類に基づいて、メタデータを指定し、WAF CRD .yamlファイル内のCRD属性を使用してWAFポリシーを定義できます。

WAF CRDの定義

WAF CRDは、NetScaler Ingress ControllerのGitHubリポジトリのwaf-crd.yamlで入手できます。WAF CRDは、NetScalerでWebアプリケーションファイアウォールポリシーを定義するために必要なさまざまなオプションの属性を提供します。

WAF CRD属性

次の表に、WAF CRDで提供されるさまざまな属性を示します。

CRD属性	説明
`commonchecks`	コンテンツタイプに関係なく適用される、一般的なセキュリティチェックのリストを指定します。
`block_urls`	URLを保護します。
`buffer_overflow`	バッファオーバーフローを保護します。
`content_type`	コンテンツタイプを保護します。
`htmlchecks`	HTMLコンテンツタイプに適用されるセキュリティチェックのリストを指定します。
`cross_site_scripting`	クロスサイトスクリプティング攻撃を防止します。
`sql_injection`	SQLインジェクション攻撃を防止します。
`form_field_consistency`	フォームの改ざんを防止します。
`csrf`	クロスサイトリクエストフォージェリ (CSRF) 攻撃を防止します。
`cookie_consistency`	クッキーの改ざんやセッションの乗っ取りを防止します。
`field_format`	フォームの送信を検証します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD属性	説明
`fileupload_type`	悪意のあるファイルのアップロードを防止します。
`jsonchecks`	JSONコンテンツタイプのセキュリティチェックを指定します。
`xmlchecks`	XMLコンテンツタイプのセキュリティチェックを指定します。
`wsi`	Webサービスの相互運用性を保護します。
`redirect_url`	保護でブロックが有効になっている場合にURLをリダイレクトします。
`servicenames`	WAFポリシーが適用されるサービスを指定します。
`application_type`	アプリケーションタイプを保護します。
`signatures`	外部署名ファイルの場所を指定します。
`html_error_object`	HTMLまたは一般的な違反が試行されたときに応答する、カスタマイズされたエラーページの場所を指定します。
`xml_error_object`	XML違反が試行されたときに応答する、カスタマイズされたエラーページの場所を指定します。
`json_error_object`	JSON違反が試行されたときに応答する、カスタマイズされたエラーページの場所を指定します。
`ip_reputation`	IPレピュテーション機能を有効にします。
`target`	WAFによって検査されるトラフィックを決定します。対象となるトラフィックを指定しない場合、すべてのトラフィックがデフォルトで検査されます。
`target.path`	検査するHTTP URLのリストを指定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD属性	説明
`target.method`	検査するHTTPメソッドのリストを指定します。
`target.header`	検査対象のHTTPヘッダーのリストを指定します。
`exclude`	WAFによって除外されるトラフィックを決定します。除外するトラフィックを指定しない場合、すべてのトラフィックがデフォルトで検査されます。
`exclude.path`	除外するHTTP URLのリストを指定します。
`exclude.method`	除外するHTTPメソッドのリストを指定します。
`exclude.header`	除外するHTTPヘッダーのリストを指定します。

WAF CRDを展開する

WAF CRDを展開するには、次の手順を実行します。

1. CRD (waf-crd.yaml) をダウンロードします。
2. 次のコマンドを使用してWAF CRDを展開します。

```
kubectl create -f waf-crd.yaml
```

例:

```
root@master:~# kubectl create -f waf-crd.yaml

customresourcedefinition.apiextensions.k8s.io/wafpolicies.citrix.com
created
```

WAF設定の記述方法

NetScaler®が提供するWAF CRDをKubernetesクラスターにデプロイした後、.yamlファイルでWebアプリケーションファイアウォールポリシー設定を定義できます。.yamlファイルでは、kind

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールドにwafを使用します。specセクションで、ポリシー設定の要件に基づいてWAF CRD属性を追加します。

.yamlファイルをデプロイすると、NetScaler Ingress ControllerはIngress NetScalerデバイスにWAF設定を適用します。

以下は、Webアプリケーションファイアウォールポリシーを記述するいくつかの例です。

クロスサイトスクリプティングおよびSQLインジェクション攻撃に対する保護を有効にする

NetScalerでWebアプリケーションファイアウォールポリシーを定義および指定して、クロスサイトスクリプティングおよびSQLインジェクション攻撃に対する保護を有効にするシナリオを考えてみましょう。`wafhtmlxsssql.yaml`という名前の.yamlファイルを作成し、適切なCRD属性を使用してWAFポリシーを次のように定義できます。

```
apiVersion: citrix.com/v1

kind: waf

metadata:

    name: wafhtmlxsssql

spec:

    servicenames:

        - frontend

    application_type:

        - "HTML"

    html_error_object: "http://x.x.x.x/crd/error_page.html"

    security_checks:

        html:

          cross_site_scripting: "on"

          sql_injection: "on"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

既知のコンテンツタイプのみを許可するルールを適用する

既知のコンテンツタイプのみを許可し、不明なコンテンツタイプをブロックするルールを指定するWebアプリケーションファイアウォールポリシーを定義するシナリオを考えてみましょう。`waf-contenttype.yaml`という名前の`.yaml`ファイルを作成し、適切なCRD属性を使用してWAFポリシーを次のように定義します。

```
apiVersion: citrix.com/v1

kind: waf

metadata:

    name: wafcontenttype

spec:

    servicenames:

        - frontend

    application_type:

        - "HTML"

    html_error_object: "http://x.x.x.x/crd/error_page.html"

    security_checks:

        common:

          content_type: "on"

    relaxations:

        common:

          content_type:

            types:

                - custom_cnt_type
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
                - image/crd
```

既知の攻撃から保護する

以下は、外部署名を適用するためのWAF CRD設定の例です。最新のWAF署名をSignature LocationからローカルWebサーバーにコピーし、コピーしたファイルの場所を*signature_url*として指定できます。

```
apiVersion: citrix.com/v1

kind: waf

metadata:

    name: wafhtmlsigxsssql

spec:

    servicenames:

        - frontend

    application_type:

        - "HTML"

    signatures: "http://x.x.x.x/crd/sig.xml"

    html_error_object: "http://x.x.x.x/crd/error_page.html"

    security_checks:

        html:

          cross_site_scripting: "on"

          sql_injection: "on"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ヘッダーバッファオーバーフロー攻撃から保護し、複数のヘッダーをブロックする

以下は、バッファオーバーフローを保護するためのWAF CRD設定の例です。

```
apiVersion: citrix.com/v1

kind: waf

metadata:

    name: wafhdrbufferoverflow

spec:

    servicenames:

        - frontend

    application_type:

        - "HTML"

    html_error_object: "http://x.x.x.x/crd/error_page.html"

    security_checks:

        common:

          buffer_overflow: "on"

          multiple_headers:

            action: ["block", "log"]

    settings:

        common:

          buffer_overflow:

            max_cookie_len: 409

            max_header_len: 4096
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            max_url_len: 1024
```

Webサイト上のランダムなURLへの繰り返しのアクセス試行を防ぐ

以下は、URLフィルター規則を提供するWAF CRD設定の例です。許可するURLを*allow_url*の下に、拒否するURLを*block_url*の下に追加できます。URLは正規表現にすることもできます。`target`と`exclude`オプションを使用すると、WAFポリシーが検査または無視する必要があるURL、ヘッダー、およびメソッドを指定できます。`target`または`exclude`を設定しない場合、WAFはデフォルトですべてのURLを検査します。

```
apiVersion: citrix.com/v1

kind: waf

metadata:

    name: wafurlchecks

spec:

    servicenames:

        - frontend

    application_type:

        - "HTML"

    html_error_object: "http://x.x.x.x/crd/error_page.html"

    target:

        path:

            - /

        method:

            - GET
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            - POST
        header:
            - Host
    exclude:
        path:
            - index.html
          method:
            - GET
    security_checks:
        common:
          allow_url: "on"
          block_url: "on"
    relaxations:
        common:
          allow_url:
            urls:
                - payment.php
                - cover.php
    enforcements:
        common:
          block_url:
            urls:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
                - "^[^?]*(passwd|passwords?)([.][^/?]*)?([?].*)?$"

                - "^[^?]*(htaccess|access_log)([.][^/?]*)?([~])?([?].*)?$"
```

機密データの漏洩を防ぐ

データ侵害には、クレジットカードや社会保障番号（SSN）などの機密データの漏洩が伴います。機密データ用のカスタム正規表現は、*Enforcements safe objects* セクションに追加できます。

以下は、機密データの漏洩を防ぐためのWAF CRD構成の例です。

```
apiVersion: citrix.com/v1

kind: waf

metadata:

    name: wafdataleak

spec:

    servicenames:

        - frontend

   application_type:

        - "HTML"

    html_error_object: "http://x.x.x.x/crd/error_page.html"

    security_checks:

        common:

          credit_card: "on"

    settings:

        common:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          credit_card:
            card_type: ["visa","amex"]
            max_allowed: 1
            card_xout: "on"
            secure_logging: "on"
    enforcements:
        common:
          safe_object:
            - rule:
                name: aadhar
                expression: "[1-9]{4,4}\s[1-9]{4,4}\s[1-9]{4,4}"
                max_match_len: 19
                action: ["log","block"]
```

CSRFおよびフォーム攻撃からHTMLフォームを保護する

以下は、CSRFおよびフォーム攻撃からHTMLフォームを保護するためのWAF CRD構成の例です。

```
apiVersion: citrix.com/v1
kind: waf
metadata:
    name: wafforms
spec:
    servicenames:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        - frontend
    application_type:
        - "HTML"
    html_error_object: "http://x.x.x.x/crd/error_page.html"
    security_checks:
        html:
          cross_site_scripting: "on"
          sql_injection: "on"
          form_field_consistency:
            action: ["log","block"]
          csrf: "on"
```

フォームとヘッダーを保護する

以下は、フォームとヘッダーの両方を保護するためのWAF CRD構成の例です。

```
apiVersion: citrix.com/v1
kind: waf
metadata:
    name: wafhdrforms
spec:
    servicenames:
        - frontend
   application_type:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      - "HTML"
  html_page_url: "http://x.x.x.x/crd/error_page.html"
  security_checks:
      common:
        buffer_overflow: "on"
        multiple_headers:
          action: ["block", "log"]
      html:
        cross_site_scripting: "on"
        sql_injection: "on"
        form_field_consistency:
          action: ["log","block"]
        csrf: "on"
  settings:
      common:
        buffer_overflow:
          max_cookie_len: 409
          max_header_len: 4096
          max_url_len: 1024
  ip_reputation: on
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

基本的なWAFセキュリティチェックを有効にする

基本的なセキュリティチェックは、パフォーマンスへの影響を最小限に抑えながらアプリケーションを保護するために必要です。セッション化は不要です。以下は、基本的なWAFセキュリティチェックを有効にするためのWAF CRD構成の例です。

```
apiVersion: citrix.com/v1

kind: waf

metadata:

    name: wafbasic

spec:

    servicenames:

        - frontend

    security_checks:

        common:

          allow_url: "on"

          block_url: "on

          buffer_overflow: "on"

          multiple_headers:

            action: ["block", "log"]

        html:

          cross_site_scripting: "on"

          field_format: "on"

          sql_injection: "on"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          fileupload_type: "on"
        json:
          dos: "on"
          sql_injection: "on"
          cross_site_scripting: "on"
        xml:
          dos: "on"
          wsi: "on"
          attachment: "on"
          format: "on"
    relaxations:
        common:
          allow_url:
            urls:
                - "^[^?]+.$"
                - "^[^?]+.([?].*)?$"
```

高度なWAFセキュリティチェックを有効にする

クッキーの一貫性、URLクローズの許可、フィールドの一貫性、CSRFなどの高度なセキュリティチェックは、WAFセッション化を必要とするため、リソースを大量に消費します（CPUとメモリ）。たとえば、フォームがWAFによって保護されている場合、応答内のフォームフィールド情報はシステムメモリに保持されます。クライアントが次のリクエストでフォームを送信すると、情報がWebサーバーに送信される前に不整合がないかチェックされます。このプロセスはセッション化として知られています。以下は、WAFの高度なセキュリティチェックを有効にするためのWAF CRD構成の例です。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: citrix.com/v1

kind: waf

metadata:

    name: wafadvanced

spec:

    servicenames:

        - frontend

    security_checks:

        common:

          allow_url: "on"

          block_url: "on"

          buffer_overflow: "on"

          content_type: "on"

          cookie_consistency: "on"

          multiple_headers:

            action: ["log"]

        html:

          cross_site_scripting: "on"

          field_format: "on"

          sql_injection: "on"

          form_field_consistency: "on"

          csrf: "on"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          fileupload_type: "on"
        json:
          dos: "on"
          sql_injection: "on"
          cross_site_scripting: "on"
        xml:
          dos: "on"
          wsi: "on"
          validation: "on"
          attachment: "on"
          format: "on"
    settings:
        common:
          allow_url:
            closure: "on"
```

IPレピュテーションを有効にする

以下は、評判の悪いIPアドレスからのリクエストを拒否するためにIPレピュテーションを有効にするWAF CRD構成の例です。

```
apiVersion: citrix.com/v1
kind: waf
metadata:
    name: wafiprep
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:

    application_type:

        - "HTML"

    servicenames:

        - frontend

    ip_reputation: "on"
```

特定のカテゴリのリクエストを拒否するためにIPレピュテーションを有効にする

以下は、特定の脅威カテゴリからのリクエストを拒否するためにIPレピュテーションを有効にするWAF CRD構成の例です。

```
apiVersion: citrix.com/v1

kind: waf

metadata:

    name: wafiprepcategory

spec:

    application_type:

        - "HTML"

    servicenames:

        - frontend

    ip_reputation:

        action: block

        threat-categories:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            - SPAM_SOURCES

            - WINDOWS_EXPLOITS

            - WEB_ATTACKS

            - BOTNETS

            - SCANNERS

            - DOS

            - REPUTATION

            - PHISHING

            - PROXY

            - NETWORK

            - CLOUD_PROVIDERS

            - MOBILE_THREATS
```

サービス拒否攻撃からJSONアプリケーションを保護する

JSONアプリケーションをサービス拒否攻撃から保護するためのWAF CRD構成の例を次に示します。

```
metadata:

    name: wafjsondos

spec:

    servicenames:

        - frontend

    application_type: JSON
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    json_error_object: "http://x.x.x.x/crd/error_page.json"
    security_checks:
        json:
          dos: "on"
    settings:
        json:
          dos:
            container:
              max_depth: 2
            document:
              max_len: 20000000
            array:
              max_len: 5
            key:
              max_count: 10000
              max_len: 12
            string:
              max_len: 1000000
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

RESTful APIを保護する

SQLインジェクション、クロスサイトスクリプティング、およびサービス拒否攻撃からRESTful APIを保護するためのWAF CRD構成の例を次に示します。 ここでは、バックエンドアプリケーションまたはサービスは純粋にRESTful APIに基づいています。

```
apiVersion: citrix.com/v1

kind: waf

metadata:

    name: wafjson

spec:

    servicenames:

        - frontend

    application_type: JSON

    json_error_object: "http://x.x.x.x/crd/error_page.json"

    security_checks:

        json:

          dos: "on"

          sql_injection:

            action: ["block"]

          cross_site_scripting: "on"

    settings:

        json:

          dos:

            container:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
              max_depth: 5
            document:
              max_len: 20000000
            array:
              max_len: 10000
            key:
              max_count: 10000
              max_len: 128
            string:
              max_len: 1000000
```

XMLアプリケーションをサービス拒否攻撃から保護する

XMLアプリケーションをサービス拒否攻撃から保護するためのWAF CRD構成の例を次に示します。

```
apiVersion: citrix.com/v1
kind: waf
metadata:
    name: wafxmldos
spec:
    servicenames:
        - frontend
    application_type: XML
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    xml_error_object: "http://x.x.x.x/crd/error_page.xml"
    security_checks:
        xml:
          dos: "on"
    settings:
        xml:
          dos:
            attribute:
                max_attributes: 1024
                max_name_len: 128
                max_value_len: 128
            element:
                max_elements: 1024
                max_children: 128
                max_depth: 128
            file:
                max_size: 2123
                min_size: 9
            entity:
                max_expansions: 512
                max_expansions_depth: 9
            namespace:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
                max_namespaces: 16

                max_uri_len: 256

            soaparray:

                max_size: 1111

            cdata:

                max_size: 65
```

XMLアプリケーションをセキュリティ攻撃から保護する

この例では、XMLアプリケーションを以下のセキュリティ攻撃から保護するためのWAF CRD構成を示します。

- SQLインジェクション
- クロスサイトスクリプティング
- 検証（スキーマまたはメッセージ）
- フォーマット
- サービス拒否
- Webサービス相互運用性（WSI）

```
apiVersion: citrix.com/v1

kind: waf

metadata:

    name: wafxml

spec:

    servicenames:

        - frontend

    application_type: XML
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    xml_error_object: "http://x.x.x.x/crd/error_page.json"
    security_checks:
        xml:
          dos: "on"
          sql_injection: "on"
          cross_site_scripting: "off"
          wsi:
            action: ["block"]
          validation: "on"
          attachment: "on"
          format:
            action: ["block"]
    settings:
        xml:
          dos:
            attribute:
                max_attributes: 1024
                max_name_len: 128
                max_value_len: 128
            element:
                max_elements: 1024
                max_children: 128
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            max_depth: 128
        file:
            max_size: 2123
            min_size: 9
        entity:
            max_expansions: 512
            max_expansions_depth: 9
        namespace:
            max_namespaces: 16
            max_uri_len: 256
        soaparray:
            max_size: 1111
        cdata:
            max_size: 65
    wsi:
        checks: ["R1000","R1003"]
    validation:
        soap_envelope: "on"
        validate_response: "on"
    attachment:
        url:
            max_size: 1111
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            content_type:

                value: "crd_test"
```

NetScaler Ingress Controller を使用してボット管理ポリシーを構成する

ボットは、手動タスクを自動化するソフトウェアアプリケーションです。ボット管理ポリシーを使用すると、有用なボットがクラウドネイティブ環境にアクセスすることを許可し、悪意のあるボットをブロックできます。

カスタムリソース定義 (CRD) は、クラウドネイティブデプロイメントでポリシーを構成する主要な方法です。NetScaler が提供するボット CRD を使用すると、NetScaler VPX 上の NetScaler Ingress Controller でボット管理ポリシーを構成できます。ボット CRD は、ボット管理ポリシーを適用するために NetScaler Ingress Controller と NetScaler 間の通信を可能にします。

Kubernetes デプロイメントでは、ボット CRD を使用して、サーバーとの間のリクエストおよびレスポンスにボット管理ポリシーを適用できます。セキュリティの脆弱性の詳細については、ボット検出 を参照してください。

ボット CRD を使用すると、Kubernetes ネイティブアプリケーションの以下の種類のセキュリティ脆弱性に対して、ボット管理セキュリティポリシーを構成できます。

- 許可リスト
- ブロックリスト
- デバイスフィンガープリント (DFP)
- ボット TPS
- トラップ挿入
- IP レピュテーション
- レート制限

必要な保護の種類に基づいて、メタデータを指定し、ボット CRD `.yaml` ファイル内の CRD 属性を使用してボットポリシーを定義できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ボット CRD の定義

ボット CRD は、NetScaler Ingress Controller の GitHub リポジトリの bot-crd.yaml で入手できます。ボット CRD は、NetScaler 上でボット管理ポリシーを定義するために必要なさまざまなオプションの属性を提供します。

Bot CRDの属性

以下の表に、Bot CRDで提供されるさまざまな属性を示します。

CRD属性	説明
`security_checks`	受信トラフィックに適用されるセキュリティチェックのリスト。
`allow_list`	許可されたIP、サブネット、およびポリシー式のリスト。
`block_list`	許可されていないIP、サブネット、およびポリシー式のリスト。
`device_fingerprint`	JavaScriptを挿入し、クライアントのブラウザとデバイスのパラメータを収集します。
`trap`	応答に隠しURLを挿入します。
`tps`	設定されたトランザクションのパーセンテージ変更に基づいて、リクエストの異常な急増を引き起こすボットを防止します。
`reputation`	設定されたレピュテーションカテゴリに基づいて、不正なIPへのアクセスを防止します。
`ratelimit`	レート制限に基づいてボットを防止します。
`redirect_url`	保護でブロックが有効になっている場合のURLをリダイレクトします。
`ingressclass`	指定されたイングレスクラスに関連付けられたイングレスコントローラーのみがリソースを処理するように、イングレスクラスを指定します。そうでない場合、クラスター内のすべてのコントローラーがこのリソースを処理します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CRD属性	説明
`servicenames`	ボットポリシーが適用されるサービスの名前。
`signatures`	外部ボットシグネチャファイルの場所。
`target`	ボットによって検査されるトラフィックを決定します。対象となるトラフィックを指定しない場合、すべてのトラフィックがデフォルトで検査されます。
`paths`	検査対象のHTTP URLのリスト。
`method`	検査対象のHTTPメソッドのリスト。
`header`	検査対象のHTTPヘッダーのリスト。

Bot CRDを展開する

Bot CRDを展開するには、次の手順を実行します。

1. bot-crd.yamlをダウンロードします。
2. 次のコマンドを使用してBot CRDを展開します。

`kubectl create -f bot-crd.yaml`

例：

```
root@master:~# kubectl create -f bot-crd.yaml

customresourcedefinition.apiextensions.k8s.io/bots.citrix.com created
```

Bot構成の記述方法

KubernetesクラスターにNetScaler®が提供するBot CRDを展開した後、YAMLファイルでボット管理ポリシー構成を定義できます。YAMLファイルでは、kindフィールドにbotを指定します。specセクションには、ポリシー構成の要件に基づいてBot CRD属性を追加します。

YAMLファイルを展開すると、NetScaler Ingress ControllerはIngress NetScalerデバイスにボット構成を適用します。

ボットポリシー設定の例を次に示します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

既知のIP、サブネット、またはADCポリシー式を使用して悪意のあるトラフィックをブロックする

NetScalerでWebボット管理ポリシーを定義および適用して、悪意のあるトラフィックをブロックするためにボットを有効にする場合は、`botblocklist.yaml`というYAMLファイルを作成し、適切なCRD属性を使用してボットポリシーを次のように定義できます。

```
apiVersion: citrix.com/v1

kind: bot

metadata:

    name: botblocklist

spec:

    servicenames:

        - frontend

    security_checks:

        block_list: "ON"

    bindings:

        block_list:

            - subnet:

                value:

                    - 172.16.1.0/12

                    - 172.16.2.0/12

                    - 172.16.3.0/12

                    - 172.16.4.0/12

                action:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
                - "drop"

        - ip:

            value: 10.102.30.40

        - expression:

            value:  http.req.url.contains("/robots.txt")

            action:

                - "reset"

                - "log"
```

ボットセキュリティチェックなしで既知のトラフィックを許可する

ステージングトラフィックや信頼できるトラフィックなど、特定のトラフィックのセキュリティチェックを回避したい場合は、そのようなトラフィックをセキュリティチェックから除外できます。`botallowlist.yaml`というYAMLファイルを作成し、適切なCRD属性を使用してボットポリシーを次のように定義できます。

```
apiVersion: citrix.com/v1

kind: bot

metadata:

    name: botallowlist

spec:

    servicenames:

        - frontend

    security_checks:

        allow_list: "ON"

    bindings:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        allow_list:

            - subnet:

                value:

                    - 172.16.1.0/12

                    - 172.16.2.0/12

                    - 172.16.3.0/12

                    - 172.16.4.0/12

                action:

                    - "log"

            - ip:

                value: 10.102.30.40

            - expression:

                value:  http.req.url.contains("index.html")

                action:

                    - "log"
```

ボットシグネチャを有効にしてボットを検出する

NetScalerは、ユーザーエージェントに基づいてボットを検出するための何千もの組み込みシグネチャを提供します。Citrix脅威インテリジェンスチームは、2週間ごとに新しいボットシグネチャを更新およびリリースしています。最新のボットシグネチャファイルは、ボットシグネチャで入手できます。`botsignatures.yaml`というYAMLファイルを作成し、適切なCRD属性を使用してボットポリシーを次のように定義できます。

```
apiVersion: citrix.com/v1

kind: bot
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
metadata:

    name: botsignatures

spec:

    servicenames:

        - frontend

    redirect_url: "/error_page.html"

    signatures: "http://10.106.102.242/ganeshka/bot_sig.json"
```

ボットデバイスフィンガープリントを有効にしてアクションをカスタマイズする

デバイスフィンガープリントは、クライアントへのHTML応答にJavaScriptスニペットを挿入することを含みます。このJavaScriptスニペットは、クライアント上のブラウザによって呼び出されると、ブラウザとクライアントの属性を収集します。そして、その情報とともにNetScalerにPOSTリクエストを送信します。これらの属性は、接続がボットから要求されたものか、人間から要求されたものかを判断するために検査されます。`botdfp.yaml`というYAMLファイルを作成し、適切なCRD属性を使用してボットポリシーを次のように定義できます。

```
apiVersion: citrix.com/v1

kind: bot

metadata:

    name: botdfp

spec:

    servicenames:

        - frontend

    redirect_url: "/error_page.html"

    security_checks:

       device_fingerprint:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          action:
              - "log"
              - "drop"
```

ボットTPSを有効にしてアクションをカスタマイズする

ボットTPSが設定されている場合、設定された時間間隔で最大リクエスト数またはリクエストの増加が超過すると、受信トラフィックをボットとして検出します。*bindings*セクションで、*地理位置情報*、*ホスト*、*送信元IP*、および*URL*に従ってTPS制限を設定できます。`bottps.yaml`というYAMLファイルを作成し、適切なCRD属性を使用してボットポリシーを次のように定義できます。

```
apiVersion: citrix.com/v1
kind: bot
metadata:
    name: bottps
spec:
    servicenames:
        - frontend
    redirect_url: "/error_page.html"
    security_checks:
        tps: "ON"
    bindings:
        tps:
            geolocation:
                threshold: 101
                percentage: 100
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            host:
                threshold: 10
                percentage: 100
                action:
                    - "log"
                    - "mitigation"
```

トラップ挿入保護を有効にしてアクションをカスタマイズする

クライアント応答にトラップURLをアドバタイズすることで、自動ボットを検出してブロックします。このURLは、人間であればクライアントには見えず、アクセスできません。この検出方法は、自動ボットからの攻撃をブロックするのに効果的です。URL応答へのトラップURLの挿入はランダムです。トラップバインディングを設定することで、特定のURL応答へのトラップURLの挿入を強制できます。`trapinsertion.yaml`というYAMLファイルを作成し、適切なCRD属性を使用してボットポリシーを次のように定義できます。

```
apiVersion: citrix.com/v1
kind: bot
metadata:
    name: trapinsertion
spec:
    servicenames:
        - frontend
    redirect_url: "/error_page.html"
    security_checks:
       trap:
         action:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            - "log"

            - "drop"

    bindings:

      trapinsertion:

        urls:

          - "/index.html"

          - "/submit.php"

          - "/login.html"
```

IPレピュテーションを有効にして特定のカテゴリのリクエストを拒否する

以下は、ユーザー環境に適したIPレピュテーションの特定の脅威カテゴリのみを有効にするBot CRD設定の例です。`botiprepcategory.yaml`というYAMLファイルを作成し、適切なCRD属性を使用してボットポリシーを次のように定義できます。

```
apiVersion: citrix.com/v1

kind: bot

metadata:

    name: botiprepcategory

spec:

    servicenames:

        - frontend

    redirect_url: "/error_page.html"

    security_checks:

       reputation: "ON"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    bindings:

      reputation:

        categories:

            - SPAM_SOURCES:

                action:

                    - "log"

                    - "redirect"

            - MOBILE_THREATS

            - SPAM_SOURCES
```

リクエストレートを制御するためにレート制限を有効にする

以下は、URL、Cookie、IP のパラメータを使用してリクエストレート制限を適用するための Bot CRD 設定の例です。`botratelimit.yaml` という YAML ファイルを作成し、適切な CRD 属性を使用してボットポリシーを次のように定義できます。

```
apiVersion: citrix.com/v1

kind: bot

metadata:

  name: botratelimit

spec:

  servicenames:

    - frontend

  redirect_url: "/error_page.html"

  security_checks:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    ratelimit: "ON"

  bindings:

    ratelimit:

      - url:

          value: index.html

          rate: 2000

          timeslice: 1000

      - cookie:

          value: citrix_bot_id

          rate: 2000

          timeslice: 1000

      - ip:

          rate: 2000

          timeslice: 1000

          action:

              - "log"

              - "reset"
```

NetScaler Ingress Controller を使用してクロスオリジンリソース共有ポリシーを構成する

NetScaler は、Kubernetes 用の CORS CRD と呼ばれるカスタムリソース定義 (CRD) を提供します。CORS CRD を使用して、NetScaler 上の NetScaler Ingress Controller でクロスオリジンリソース共有 (CORS) ポリシーを構成できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CORS とは

クロスオリジンリソース共有は、特定の Web アプリケーションが異なるオリジンからの別の Web アプリケーションとリソースを共有できるかどうかをブラウザが判断できるようにするメカニズムです。これにより、ユーザーは元のドメイン外のドメインからリソース (画像、フォント、ビデオなど) を要求できます。

CORS プリフライト

Web ブラウザが Javascript に URL への POST を発行させる前に、`pre-flight` リクエストを実行します。プリフライトリクエストは、POST ではなく OPTIONS メソッドを使用して、同じ URL でサーバーへの単純なリクエストです。Web ブラウザは、CORS 関連ヘッダーの HTTP ヘッダーをチェックして、ユーザーに代わって POST 操作が許可されているかどうかを判断します。

CORS CRD の定義

CORS CRD は、NetScaler Ingress Controller GitHub リポジトリの cors-crd.yaml で入手できます。CORS CRD は、API ゲートウェイとして機能する Ingress NetScaler 上で CORS ポリシーを定義するために必要なさまざまなオプションの属性を提供します。必要な属性には、`servicenames`、`allow_origin`、`allow_methods`、および `allow_headers` が含まれます。

CORS CRD で提供される属性は次のとおりです。

属性	説明
`ingressclass`	指定されたイングレスクラスに関連付けられたイングレスコントローラーのみがリソースを処理するように、イングレスクラスを指定します。そうでない場合、クラスター内のすべてのコントローラーがこのリソースを処理します。
`servicenames`	CORSポリシーを適用したいKubernetesサービスのリストを指定します。
`allow_origin`	許可されたオリジンのリストを指定します。受信オリジンはこのリストに対してスクリーニングされます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
`allow_methods`	CORSプロトコルの一部として許可されるメソッドのリストを指定します。
`allow_headers`	CORSプロトコルの一部として許可されるヘッダーのリストを指定します。
`max_age`	`Access-Control-Allow-Methods`および`Access-Control-Allow-Headers`ヘッダーによって提供される情報をキャッシュできる秒数を指定します。デフォルト値は86400です。
`allow_credentials`	リクエストの資格情報モードが「include」の場合にレスポンスを共有できるかどうかを指定します。デフォルト値は「true」です。

CORS CRDを展開する

CORS CRDを展開するには、以下を実行します。

1. CORS CRDをダウンロードします。
2. 次のコマンドを使用してCORS CRDを展開します。

```
kubectl create -f cors-crd.yaml
```

例:

```
$ kubectl create -f cors-crd.yaml

customresourcedefinition.apiextensions.k8s.io/corspolicies.citrix.com
created

$ kubectl get crd

NAME                          CREATED AT

corspolicies.citrix.com       2021-05-21T20:01:13Z
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

CORSポリシー設定の記述方法

NetScalerが提供するCORS CRDをKubernetesクラスターにデプロイした後、`.yaml`ファイルでCORSポリシー設定を定義できます。`.yaml`ファイルでは、kindフィールドに`corspolicy`を使用し、`spec`セクションでポリシー設定の要件に基づいてCORS CRD属性を追加します。

次のYAMLファイルは、servicenamesフィールドにリストされているサービスに設定されたポリシーを適用します。オリジンが`allow_origins`のいずれかである場合、NetScalerはプリフライトリクエストに対して200 OK応答コードで応答します。応答には、設定された`allow_methods`、`allow_headers`、および`max_age`が含まれます。

```
apiVersion: citrix.com/v1beta1

kind: corspolicy

metadata:

  name: corspolicy-example

spec:

  servicenames:

   - "cors-service"

  allow_origin:

   - "random1234.com"

   - "hotdrink.beverages.com"

  allow_methods:

   - "POST"

   - "GET"

   - "OPTIONS"

  allow_headers:

   - "Origin"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    - "X-Requested-With"

    - "Content-Type"

    - "Accept"

    - "X-PINGOTHER"

  max_age: 86400

  allow_credentials: true
```

ポリシー設定を定義した後、次のコマンドを使用して`.yaml`ファイルをデプロイします。

```
user@master:~/cors$ kubectl create -f corspolicy-example.yaml

corspolicy.citrix.com/corspolicy-example created
```

NetScaler Ingress Controllerは、Ingress NetScalerデバイスにポリシー設定を適用します。

NetScaler Ingress Controller 用の AppQoE を使用したリクエスト再試行機能の有効化

NetScaler アプライアンスが HTTP リクエストを受信し、それをバックエンドサーバーに転送する際、バックエンドサーバーとの接続障害が発生する場合があります。NetScaler でリクエスト再試行機能を構成して、クライアントにリセットを送信する代わりに、リクエストを次に利用可能なサーバーに転送できます。これにより、NetScaler が次に利用可能なサービスに同じリクエストを開始する際に、クライアントはラウンドトリップ時間を節約できます。リクエスト再試行機能の詳細については、NetScaler ドキュメントを参照してください。

NetScaler Ingress Controller を使用して、NetScaler でリクエスト再試行を構成できるようになりました。 カスタムリソース定義 (CRD) は、クラウドネイティブ展開でポリシーを構成する主要な方法です。NetScaler が提供する AppQoE CRD を使用すると、NetScaler Ingress Controller を使用して NetScaler でリクエスト再試行ポリシーを構成できます。AppQoE CRD は、AppQoE ポリシーを適用するために NetScaler Ingress Controller と NetScaler 間の通信を可能にします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

AppQoE CRD の定義

AppQoE CRD は、NetScaler Ingress Controller の GitHub リポジトリの appqoe-crd.yaml で入手できます。AppQoE CRD は、NetScaler で AppQoE ポリシーを定義するために必要なさまざまなオプションの属性を提供します。

AppQoE CRD で提供される属性は次のとおりです。

属性	説明
`servicenames`	AppQoE ポリシーを適用する Kubernetes サービスのリストを指定します。
`on-reset`	接続リセット時に再試行を設定するかどうかを指定します。
`on-timeout`	再試行の時間をミリ秒単位で指定します。
`number-of-retries`	再試行回数を指定します。
`appqoe-criteria`	トラフィックを評価するための式を指定します。
`direction`	AppQoEポリシーをバインドするためのバインドポイントを指定します。

AppQoE CRD を展開する

AppQoE CRD を展開するには、以下を実行します。

1. AppQoE CRDをダウンロードします。
2. 次のコマンドを使用して AppQoE CRD を展開します。

```
kubectl create -f appqoe-crd.yaml
```

AppQoE ポリシー構成の記述方法

NetScaler が提供する AppQoE CRD を Kubernetes クラスターに展開した後、AppQoE ポリシー構成を `.yaml` ファイルで定義できます。`.yaml` ファイルでは、kind フィールドに `appqoepolicy` を使用し、`spec` セクションでポリシー構成の要件に基づいて AppQoE CRD 属性を追加します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

次の YAML ファイルは、servicenames フィールドにリストされているサービスに AppQoE ポリシーを適用します。AppQoE アクションをタイムアウト時に再試行するように構成し、再試行回数を定義する必要があります。

```
apiVersion: citrix.com/v1

kind: appqoepolicy

metadata:

  name: targeturlappqoe

spec:

  appqoe-policies:

    - servicenames:

        - apache

      appqoe-policy:

        operation-retry:

          onReset: 'YES'

          onTimeout: 33

        number-of-retries: 2

        appqoe-criteria: 'HTTP.REQ.HEADER("User-
Agent").CONTAINS("Android")'

        direction: REQUEST
```

ポリシー構成を定義したら、次のコマンドを使用して `.yaml` ファイルを展開します。

$ kubectl create -f appqoe-example.yaml

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Ingress Controller を介したワイルドカード DNS ドメインの設定

ワイルドカード DNS ドメインは、存在しないドメインやサブドメインに対するリクエストを処理するために使用されます。DNS ゾーンでは、ワイルドカードドメインを使用して、存在しないすべてのドメインまたはサブドメインに対するクエリを特定のサーバーにリダイレクトできます。これは、各ドメインに対して個別のリソースレコード (RR) を作成する代わりに利用されます。ワイルドカード DNS ドメインの最も一般的な使用法は、インターネットから他のメールシステムにメールを転送するために使用できるゾーンを作成することです。

ワイルドカード DNS ドメインの詳細については、NetScaler ドキュメント を参照してください。

NetScaler Ingress Controller を使用して、NetScaler 上でワイルドカード DNS ドメインを設定できるようになりました。カスタムリソース定義 (CRD) は、クラウドネイティブデプロイメントでポリシーを設定する主要な方法です。NetScaler が提供するワイルドカード DNS CRD を使用すると、NetScaler Ingress Controller を使用して NetScaler 上でワイルドカード DNS ドメインを設定できます。ワイルドカード DNS CRD は、ワイルドカードドメインをサポートするために NetScaler Ingress Controller と NetScaler 間の通信を可能にします。

使用ガイドラインと制限事項

- 完全修飾ドメイン名 (FQDN) の場合、DNS レコードを追加する方法は複数あります。Ingress リソースの NetScaler Ingress Controller に対して `NS_CONFIG_DNS_REC` 変数を有効にするか、ワイルドカード DNS CRD を使用できます。ただし、同じドメインへの複数の IP マッピングを避けるために、CRD または Ingress のいずれかを通じて設定されていることを確認する必要があります。
- ワイルドカード DNS 設定には、ワイルドカード DNS CRD を使用することをお勧めします。
- NetScaler Ingress Controller で `NS_CONFIG_DNS_REC` が有効になっている場合、Ingress を介して DNS アドレスレコードにワイルドカード DNS エントリを設定することはできません。

ワイルドカード DNS CRD の定義

ワイルドカード DNS CRD は、NetScaler Ingress Controller GitHub リポジトリの wildcarddnsentry.yaml で入手できます。**ワイルドカード DNS CRD は**、NetScaler 上でワイルドカード DNS エントリを設定するために必要なさまざまなオプションの属性を提供します。

ワイルドカード DNS CRD で提供される属性は次のとおりです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

属性	説明
`domain`	ゾーン用に設定されたワイルドカードドメイン名を指定します。
`dnsaddrec`	ワイルドカードドメインのIPv4アドレスを持つDNSアドレスレコードを指定します。
`dnsaaaarec`	ワイルドカードドメインのIPv6アドレスを持つDNS AAAAレコードを指定します。
`soarec`	SOAレコードの構成詳細を指定します。
`nsrec`	ネームサーバーの構成詳細を指定します。

Wildcard DNS CRD をデプロイする

Wildcard DNS CRD をデプロイするには、以下を実行します。

1. Wildcard DNS CRD をダウンロードします。
2. 次のコマンドを使用して Wildcard DNS CD をデプロイします。

```
kubectl create -f wildcarddnsentry.yaml
```

Wildcard DNS 構成ポリシーの記述方法

NetScaler が提供する Wildcard DNS CRD を Kubernetes クラスターにデプロイした後、ワイルドカード DNS 関連の構成を `yaml` ファイルで定義できます。`.yaml` ファイルでは、kind フィールドに `wildcarddnsentry` を使用し、`spec` セクションでポリシー構成の要件に基づいて Wildcard DNS CRD 属性を追加します。

以下は、NetScaler 上で SOA レコード、NS レコード、DNS ゾーン、およびアドレスと AAAA レコードを構成する YAML ファイル定義の例です。

```
apiVersion:

citrix.com/v1

kind: wildcarddnsentry

metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: sample-config
spec:
  zone:
    domain: configexample
    dnsaddrec:
      domain-ip: 1.1.1.1
      ttl: 3600
    dnsaaaarec:
      domain-ip: '2001::.1'
      ttl: 3600
    soarec:
      origin-server: n2.configexample.com
      contact: admin.configexample.com
      serial: 100
      refresh: 3600
      retry: 3
      expire: 3600
    nsrec:
      nameserver: n1.configexample.com
      ttl: 3600
```

DNS設定を定義した後、以下のコマンドを使用して`wildcarddns-example.yaml`ファイルをデプロイします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl create -f wildcarddns-example.yaml
```

メトリクス

メトリクスには、以下のトピックがあります。

- PrometheusおよびGrafanaとの統合
- 分析と可観測性
- ConfigMapを使用した分析構成のサポート

PrometheusとGrafanaを使用してNetScalerのメトリクスを表示する

NetScaler Metrics ExporterとPrometheus-Operatorを使用して、NetScaler VPXまたはCPXイングレスデバイス、およびNetScaler CPX (east-west) デバイスを監視できます。

NetScaler メトリクス エクスポーター

NetScaler Metrics Exporterは、NetScalerの統計情報を収集し、`HTTP`を使用してPrometheusにエクスポートするシンプルなサーバーです。その後、PrometheusをGrafanaのデータソースとして追加し、NetScalerの統計情報をグラフィカルに表示できます。詳細については、NetScaler Metrics Exporterを参照してください。

注:

NetScaler Metrics Exporterは、NetScalerの管理パーティションからのメトリクスエクスポートをサポートしています。

Prometheus Operatorを起動する

Prometheus Operatorは、Kubernetes上のサービスを監視するための広範な方法を提供します。開始するには、このトピックでは`kube-prometheus`とそのマニフェストファイルを使用します。マニフェストファイルは、基本的な動作モデルをデプロイするのに役立ちます。以下のコマンドを使用して、Kubernetes環境にPrometheus Operatorをデプロイします。

```
git clone https://github.com/coreos/kube-prometheus.git

kubectl create -f kube-prometheus/manifests/setup/
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl create -f kube-prometheus/manifests/
```

Prometheus-Operatorをデプロイすると、いくつかのポッドとサービスがデプロイされます。デプロイされたポッドのうち、`prometheus-k8s-xx`ポッドはメトリクス集約とタイムスタンプ用であり、`grafana`ポッドは視覚化用です。クラスターで実行されているすべてのコンテナイメージを表示すると、次の出力が表示されます。

```
$ kubectl get pods -n monitoring

NAME                                  READY     STATUS    RESTARTS   AGE

alertmanager-main-0                   2/2       Running   0          2h

alertmanager-main-1                   2/2       Running   0          2h

alertmanager-main-2                   2/2       Running   0          2h

grafana-5b68464b84-5fvxq              1/1       Running   0          2h

kube-state-metrics-6588b6b755-d6ftg   4/4       Running   0          2h

node-exporter-4hbcp                   2/2       Running   0          2h

node-exporter-kn9dg                   2/2       Running   0          2h

node-exporter-tpxhp                   2/2       Running   0          2h

prometheus-k8s-0                      3/3       Running   1          2h

prometheus-k8s-1                      3/3       Running   1          2h

prometheus-operator-7d9fd546c4-m8t7v  1/1       Running   0          2h
```

注: `manifests`フォルダー内のファイルは相互に依存しているため、作成順序が重要です。特定のシナリオでは、マニフェストファイルが順不同で作成され、Kubernetesからエラーメッセージが表示されることがあります。 このシナリオを解決するには、`kubectl create -f kube-prometheus/manifests/`コマンドを再実行してください。初回に満たされていない依存関係のために作成されなかったYAMLファイルは、これで作成されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

PrometheusおよびGrafanaポッドをNodePort経由で公開することをお勧めします。そのためには、`prometheus-service.yaml`および`grafana-service.yaml`ファイルを次のように変更する必要があります。

Prometheusサービスを変更する

```
apiVersion: v1

kind: Service

metadata:

  labels:

    prometheus: k8s

  name: prometheus-k8s

  namespace: monitoring

spec:

  type: NodePort

  ports:

  - name: web

    port: 9090

    targetPort: web

  selector:

    app: prometheus

    prometheus: k8s
```

`prometheus-service.yaml`ファイルを変更した後、以下のコマンドを使用してKubernetesクラスターに変更を適用します。

```
kubectl apply -f prometheus-service.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Grafanaサービスを変更する

```
apiVersion: v1

kind: Service

metadata:

  name: grafana

  namespace: monitoring

spec:

  type: NodePort

  ports:

  - name: http

    port: 3000

    targetPort: http

  selector:

    app: grafana
```

`grafana-service.yaml`ファイルを変更した後、以下のコマンドを使用してKubernetesクラスターに変更を適用します。

```
kubectl apply -f grafana-service.yaml
```

NetScaler Metrics Exporter の構成

このトピックでは、NetScaler Metrics Exporter を NetScaler VPX または CPX イングレスデバイス、あるいは NetScaler CPX (east-west) デバイスと統合する方法について説明します。

NetScaler VPX イングレスデバイス用の NetScaler Metrics Exporter の構成

イングレス NetScaler VPX デバイスを監視するには、NetScaler Metrics Exporter は Kubernetes クラスター内でポッドとして実行されます。NetScaler VPX イングレスデバイスの IP アドレスは、NetScaler Metrics Exporter への引数として提供されます。ADC にアクセスするためのログイ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ン資格情報を提供するには、シークレットを作成し、ボリュームをマウントパス「/mnt/nslogin」にマウントします。

```
kubectl create secret generic nslogin --from-literal=username=<citrix-adc-user> --from-literal=password=<citrix-adc-password> -n <namespace>
```

エクスポーターをデプロイするための YAML ファイルの例を次に示します。

```
apiVersion: v1

kind: Pod

metadata:

  name: exporter-vpx-ingress

  labels:

    app: exporter-vpx-ingress

spec:

  containers:

    - name: exporter

      image: "quay.io/citrix/citrix-adc-metrics-exporter:1.4.8"

      imagePullPolicy: IfNotPresent

      args:

        - "--target-nsip=<IP_of_VPX>"

        - "--port=8888"

      volumeMounts:

      - name: nslogin

        mountPath: "/mnt/nslogin"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        readOnly: true
      securityContext:
        readOnlyRootFilesystem: true
  volumes:
  - name: nslogin
    secret:
      secretName: nslogin
---
kind: Service
apiVersion: v1
metadata:
  name: exporter-vpx-ingress
  labels:
    service-type: citrix-adc-monitor
spec:
  selector:
    app: exporter-vpx-ingress
  ports:
    - name: exporter-port
      port: 8888
      targetPort: 8888
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler VPX デバイスの IP アドレスとポートは、`--target-nsip` パラメーターで指定する必要があります。例: `--target-nsip=10.0.0.20`。

NetScaler CPX イングレスデバイス用の NetScaler Metrics Exporter の構成

NetScaler CPX イングレスデバイスを監視するには、NetScaler Metrics Exporter は NetScaler CPX のサイドカーとして追加されます。エクスポーターをサイドカーとして持つ NetScaler CPX イングレスデバイスの YAML ファイルの例を次に示します。

```
---

apiVersion: apps/v1

kind: Deployment

metadata:

  labels:

    app: cpx-ingress

  name: cpx-ingress

spec:

  replicas: 1

  selector:

    matchLabels:

      app: cpx-ingress

  template:

    metadata:

      annotations:

        NETSCALER_AS_APP: "True"

      labels:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        app: cpx-ingress
    spec:
      containers:
      - env:
        - name: EULA
          value: "YES"
        - name: NS_PROTOCOL
          value: HTTP
        - name: NS_PORT
          value: "9080"
        #Define the NITRO port here
        image: quay.io/citrix/citrix-k8s-cpx-ingress:13.0-52.24
        imagePullPolicy: IfNotPresent
        name: cpx-ingress
        ports:
        - containerPort: 80
          name: http
          protocol: TCP
        - containerPort: 443
          name: https
          protocol: TCP
        - containerPort: 9080
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        name: nitro-http
        protocol: TCP
      - containerPort: 9443
        name: nitro-https
        protocol: TCP
      securityContext:
        privileged: true
    # Adding exporter as a sidecar
    - args:
      - --target-nsip=192.0.0.2
      - --port=8888
      - --secure=no
      env:
      - name: NS_USER
        value: nsroot
      - name: NS_PASSWORD
        value: nsroot
      image: quay.io/citrix/citrix-adc-metrics-exporter:1.4.8
      imagePullPolicy: IfNotPresent
      name: exporter
      securityContext:
        readOnlyRootFilesystem: true
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      serviceAccountName: cpx

---

kind: Service

apiVersion: v1

metadata:

  name: exporter-cpx-ingress

  labels:

    service-type: citrix-adc-monitor

spec:

  selector:

    app: cpx-ingress

  ports:

    - name: exporter-port

      port: 8888

      targetPort: 8888
```

ここでは、エクスポーターはローカル IP アドレス (`192.0.0.2`) を使用して NetScaler CPX からメトリクスをフェッチします。

NetScaler CPX (east-west) デバイス用の NetScaler Metrics Exporter の構成

NetScaler CPX (east-west) デバイスを監視するには、NetScaler Metrics Exporter は NetScaler CPX のサイドカーとして追加されます。エクスポーターをサイドカーとして持つ NetScaler CPX (east-west) デバイスの YAML ファイルの例を次に示します。

```
apiVersion: apps/v1

kind: DaemonSet
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
metadata:
  annotations:
    deprecated.daemonset.template.generation: "0"
  labels:
    app: cpx-ew
  name: cpx-ew
spec:
  selector:
    matchLabels:
      app: cpx-ew
  template:
    metadata:
      annotations:
        NETSCALER_AS_APP: "True"
      labels:
        app: cpx-ew
      name: cpx-ew
    spec:
      containers:
      - env:
        - name: EULA
          value: "yes"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        - name: NS_NETMODE
          value: HOST
        #- name: "kubernetes_url"
        #  value: "https://10..xx.xx:6443"
        image: quay.io/citrix/citrix-k8s-cpx-ingress:13.0-52.24
        imagePullPolicy: IfNotPresent
        name: cpx
        securityContext:
          privileged: true
      # Add exporter as a sidecar
      - args:
        - --target-nsip=192.168.0.2
        - --port=8888
        - --secure=no
        env:
        - name: NS_USER
          value: nsroot
        - name: NS_PASSWORD
          value: nsroot
        image: quay.io/citrix/citrix-adc-metrics-exporter:1.4.8
        imagePullPolicy: IfNotPresent
        name: exporter
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        securityContext:
          readOnlyRootFilesystem: true
      serviceAccountName: cpx
---
kind: Service
apiVersion: v1
metadata:
  name: exporter-cpx-ew
  labels:
    service-type: citrix-adc-monitor
spec:
  selector:
    app: cpx-ew
  ports:
    - name: exporter-port
      port: 8888
      targetPort: 8888
```

ここでは、エクスポーターはローカル IP (`192.168.0.2`) を使用して NetScaler CPX (east-west) デバイスからメトリクスをフェッチします。

NetScaler を検出するための ServiceMonitor

NetScaler Metrics Exporter は、NetScaler VPX または CPX イングレスデバイス、および NetScaler CPX (east-west) デバイスからデータを収集するのに役立ちます。Prometheus Operator は、メトリクスにタイムスタンプを付け、保存し、Grafana で視覚化できるように、これらのエク

© 1997–2026 Citrix Systems, Inc. All rights reserved.

スポーターを検出する必要があります。Prometheus Operator は、ServiceMonitor の概念を使用して、サービスにアタッチされたラベルを使用して、そのサービスに属するポッドを検出します。

次の YAML ファイルの例は、`service-type: citrix-adc-monitor` ラベルが関連付けられているすべてのエクスポーターサービス (サンプル YAML ファイルで指定) を検出します。

```
apiVersion: monitoring.coreos.com/v1

kind: ServiceMonitor

metadata:

  name: citrix-adc-servicemonitor

  labels:

    servicemonitor: citrix-adc

spec:

  endpoints:

  - interval: 30s

    port: exporter-port

  selector:

    matchLabels:

      service-type: citrix-adc-monitor

  namespaceSelector:

    matchNames:

    - monitoring

    - default
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

`ServiceMonitor`は、Prometheusが`default`と`monitoring`の名前空間にあるExporterのみを検出するように指示します。他の名前空間からExporterを検出するには、それらの名前空間の名前を`namespaceSelector:`フィールドの下に追加します。

注記:

監視する必要があるExporterが`default`または`monitoring`以外の名前空間に存在する場合、Prometheusがそれらの名前空間にアクセスするために追加のRBAC権限を提供する必要があります。以下は、Prometheusにすべての名前空間のリソースへのフルアクセスを提供するサンプルYAML (`prometheus-clusterRole.yaml`) ファイルです。

```
apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

  name: prometheus-k8s

rules:

- apiGroups:

  - ""

  resources:

  - nodes/metrics

  - namespaces

  - services

  - endpoints

  - pods

  verbs: ["*"]

- nonResourceURLs:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  - /metrics

  verbs:  ["*"]
```

Prometheusに追加の権限を付与するには、次のコマンドを使用してサンプルYAMLをデプロイします。

```
kubectl apply -f prometheus-clusterRole.yaml
```

Grafanaでメトリクスを表示する

監視のために検出されたNetScalerインスタンスは、Prometheusコンテナの**ターゲット**ページに表示されます。**ターゲット**ページには、次のURLからアクセスできます: `http://<k8s_cluster_ip>:<prometheus_nodeport>/targets`:

- [ ] メトリクスをグラフで表示するには:

1. `http://<k8s_cluster_ip>:<grafafa_nodeport>`を使用して、デフォルトの認証情報 *admin:admin* でGrafanaにログインします。
2. 左側のパネルで**+**を選択し、**インポート**をクリックしてサンプルGrafanaダッシュボードをインポートします。

   - [ ] 次のようなグラフを含むダッシュボードが表示されます。

   - [ ] Grafanaのドキュメントまたはデモビデオを使用して、ダッシュボードをさらに強化できます。

分析と可観測性

NetScalerインスタンスからの分析は、アプリケーションのパフォーマンスに関する詳細な洞察を提供し、問題を迅速に特定し、必要な措置を講じるのに役立ちます。

NetScaler Ingress Controller YAMLファイルでアノテーションを使用して分析を有効にする

IngressまたはLoadBalancerタイプのサービス構成でスマートアノテーションとして定義されている分析プロファイルを使用して、分析を有効にできます。アプリケーションのIngressまたはサー

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ビス構成で、監視する必要がある特定のパラメータを指定できます。 以下は、HTTPレコード用の分析プロファイルを含むIngressアノテーションのサンプルです。

`ingress.citrix.com/analyticsprofile: '{"webinsight": {"httpurl":"ENABLED", "httpuseragent":"ENABLED", "httpHost":"ENABLED","httpMethod":"ENABLED","httpContentType":"ENABLED"}}'`

以下は、Webアプリケーション用の分析プロファイルを含むIngress構成のサンプルです。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  annotations:

    ingress.citrix.com/analyticsprofile: '{"webinsight":
{"httpurl":"ENABLED", "httpuseragent":"ENABLED",

      "httphost":"ENABLED", "httpmethod":"ENABLED",
"httpcontenttype":"ENABLED"}}'

    ingress.citrix.com/insecure-termination: allow

  name: webserver-ingress

spec:

  rules:

  - http:

      paths:

      - backend:

          service:

            name: webserver

            port:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
              number: 80

        path: /

        pathType: Prefix

  tls:

  - secretName: name
```

以下はサービスアノテーションです。

`service.citrix.com/analyticsprofile: '{"80-tcp":{"webinsight": {"httpurl":"ENABLED", "httpuseragent":"ENABLED"}}}'`

以下は、Apache Webアプリケーションを公開する分析プロファイルを含むサービス構成のサンプルです。

```
apiVersion: v1

kind: Service

metadata:

  name: apache

  annotations:

    service.citrix.com/csvserver: '{"l2conn":"on"}'

    service.citrix.com/lbvserver: '{"80-tcp":
{"lbmethod":"SRCIPDESTIPHASH"}}'

    service.citrix.com/servicegroup: '{"80-tcp":{"usip":"yes"}}'

    service.citrix.com/monitor: '{"80-tcp":{"type":"http"}}'

    service.citrix.com/frontend-ip: "192.0.2.16"

    service.citrix.com/analyticsprofile: '{"80-tcp":{"webinsight":
{"httpurl":"ENABLED", "httpuseragent":"ENABLED"}}}'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    NETSCALER_VPORT: "80"
  labels:
    name: apache
spec:
  externalTrafficPolicy: Local
  type: LoadBalancer
  selector:
    name: apache
  ports:
  - name: http
    port: 80
    targetPort: http
  selector:
    app: apache
```

アノテーションの詳細については、アノテーションのドキュメントを参照してください。

NetScaler Consoleを使用した分析

NetScaler Consoleは、システムで発生するさまざまなイベントに関する分析や、使いやすいユーザーインターフェイスでサービスを監視するためのサービスグラフを含む、包括的な可観測性ソリューションを提供します。

NetScaler Consoleの分析は、NetScalerインスタンスからのデータからさまざまな洞察を得て、アプリケーションのパフォーマンスを記述、予測、改善するための簡単でスケーラブルな方法を提供します。NetScaler Consoleでは、1つ以上の分析機能を同時に使用できます。サービスグラフの詳細については、サービスグラフのドキュメントを参照してください。

ADM分析またはサービスグラフを使用するには：

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- ADMエージェントをインストールし、NetScaler Consoleとデータセンターまたはクラウド内のKubernetesクラスターまたは管理対象インスタンス間の通信が確保されていることを確認する必要があります。これにより、NetScalerインスタンスがNetScaler Consoleによって検出可能になります。
- 適切なライセンスが利用可能であり、ADMで自動ライセンスが有効になっていることを確認してください。

オープンソースツールによる分析

NetScalerは、NetScaler Observability Exporterを使用して、オブザーバビリティのためにさまざまなオープンソースツールと統合できます。NetScaler Observability Exporterは、NetScalerからメトリクスとトランザクションを収集し、サポートされているエンドポイントに適した形式（JSON、AVROなど）に変換するコンテナです。収集されたデータを目的のエンドポイントにエクスポートできます。データを分析することで、NetScalerによってプロキシされたアプリケーションのマイクロサービスレベルで貴重な洞察を得ることができます。 NetScaler Observability Exporterの詳細については、NetScaler Observability Exporterのドキュメントを参照してください。

ConfigMap を使用した分析構成のサポート

NetScaler Observability Exporter を使用して、NetScaler CPX、MPX、またはVPXからメトリクスとトランザクションをエクスポートし、エクスポートされたデータを分析して有意義なインサイトを得ることができます。NetScaler Ingress Controller 内で ConfigMap を使用して、NetScaler Observability Exporter の構成を有効にできるようになりました。

NS_ANALYTICS_CONFIG のスキーマ

`NS_ANALYTICS_CONFIG` のスキーマは次のとおりです。ConfigMap の属性は、このスキーマに準拠している必要があります。

```
type: map

mapping:

  NS_ANALYTICS_CONFIG:

    required: false

    type: map

    mapping:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      endpoint:
        required: false
        type: map
        mapping:
          metrics:
            required: false
            type: map
            mapping:
              service:
                required: false
                type: str
          transactions:
            required: false
            type: map
            mapping:
              service:
                required: false
                type: str
      distributed_tracing:
        required: false
        type: map
        mapping:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        enable:
          required: true
          type: str
          enum:
            - 'true'
            - 'false'
        samplingrate:
          required: false
          type: int
          range:
            max: 100
            min: 0
    timeseries:
      required: false
      type: map
      mapping:
        port:
          required: false
          type: int
        metrics:
          required: false
          type: map
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            mapping:
              enable:
                required: true
                type: str
                enum:
                  - 'true'
                  - 'false'
              mode:
                required: true
                type: str
                enum:
                  - prometheus
                  - avro
                  - influx
              export_frequency:
                required: false
                type: int
                range:
                  max: 300
                  min: 30
              schema_file:
                required: false
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            type: str
          enable_native_scrape:
            required: false
            type: str
            enum:
              - 'true'
              - 'false'
      auditlogs:
        required: false
        type: map
        mapping:
          enable:
            required: true
            type: str
            enum:
              - 'true'
              - 'false'
      events:
        required: false
        type: map
        mapping:
          enable:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            required: true
            type: str
            enum:
              - 'true'
              - 'false'
    transactions:
      required: false
      type: map
      mapping:
        enable:
          required: true
          type: str
          enum:
            - 'true'
            - 'false'
        port:
          required: false
          type: int
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注記:

- NSIC リリース 2.0.x 以降、`endpoint.server` パラメータは `endpoint.metrics.service` パラメータに置き換えられました。
- NSIC リリース 2.0.x 以降、`endpoint.service` パラメータは `endpoint.transactions.service` パラメータに置き換えられました。
- NSIC Helm チャートバージョン 1.x から 2.x にアップグレードする場合、`metrics_endpoint` (analyticsConfig.endpoint.server) と `transactions_endpoint` (analyticsConfig.endpoint.service) の値をメモし、Helm アップグレードで次の設定パラメータを使用してアップグレードします: `--set analyticsConfig.endpoint.metrics.service=<metrics_endpoint>,analyticsConfig.endpoint.transactions.service=<transactions_endpoint>`。

分析構成用の ConfigMap の作成

Helm チャートまたは NetScaler Operator を使用して、NSIC のインストールまたはアップグレード中に分析構成用の ConfigMap を作成できます。ConfigMap の作成に必要なパラメータについては、構成 を参照してください。

以下は、ConfigMap の values.yaml のサンプルです。サポートされている変数については、ConfigMap を使用した分析構成のサポートされている環境変数 を参照してください。

```
analyticsConfig:

required: true

distributedTracing:

  enable: true

  samplingrate: 100

endpoint:

  metrics:

    service: "1.1.1.1"

  transactions:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    service: "1.1.1.2"
timeseries:
  port: 5563
  metrics:
    enable: true
    mode: 'prometheus'
    exportFrequency: 30
    schemaFile: schema.json
    enableNativeScrape: false
  auditlogs:
    enable: true
  events:
    enable: true
transactions:
  enable: true
  port: 5557
```

NetScaler Ingress Controller での ConfigMap サポートの構成方法の詳細については、NetScaler Ingress Controller の ConfigMap サポートの構成 を参照してください。

ConfigMap を使用した分析構成のサポートされている環境変数

ConfigMap を使用して、`NS_ANALYTICS_CONFIG` の下の次のパラメータを構成できます。

- `distributed_tracing`: この変数は NetScaler で OpenTracing を有効または無効にし、次の属性を持ちます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

    - `enable`: OpenTracingを有効にするには、この値を`true`に設定します。デフォルト値は`false`です。
    - `samplingrate`: OpenTracingのサンプリングレートをパーセンテージで指定します。デフォルト値は100です。

詳細については、分散トレースを参照してください。

- `endpoint`: 可観測性エンドポイントのIPアドレスまたはDNSアドレスを指定します。
    - `metrics`:
        - `service`: この値を、可観測性エンドポイントのIPアドレスまたはDNSアドレスとして設定します。
    - `transactions`:
        - `service`: この値を、NetScaler Observability ExporterサービスのIPアドレスまたは`namespace/service`として設定します。
- `timeseries`: NetScalerからの時系列データのエクスポートを有効にします。時系列設定には、次の属性を指定できます。
    - `port`: 分析サーバーの時系列エンドポイントのポート番号を指定します。デフォルト値は5563です。
    - `metrics`: NetScalerからのメトリックのエクスポートを有効にします。
        - `enable`: メトリックの送信を有効にするには、この値を`true`に設定します。デフォルト値は`false`です。
        - `mode`: メトリックエンドポイントのモードを指定します。指定可能な値は`avro`と`prometheus`です。
    - `auditlogs`: NetScalerからの監査ログデータのエクスポートを有効にします。
        - `enable`: 監査ログデータを有効にするには、この値を`true`に設定します。デフォルト値は`false`です。
    - `events`: NetScalerからのイベントのエクスポートを有効にします。
        - `enable`: イベントのエクスポートを有効にするには、この値を`true`に設定します。デフォルト値は`false`です。
- `transactions`: NetScalerからのトランザクションのエクスポートを有効にします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- `enable`: トランザクションの送信を有効にするには、この値を`true`に設定します。デフォルト値は`false`です。
- `port`: アナリティクスサーバーのトランザクションエンドポイントのポート番号を指定します。デフォルト値は5557です。

NetScaler Ingress Controllerの実行中に、ConfigMap設定をランタイムで変更できます。

`NS_ANALYTICS_CONFIG`の属性は、明確に定義されたスキーマに従う必要があります。提供された値がスキーマに準拠していない場合、設定全体が拒否されます。参照については、スキーマファイルns_analytics_config_schema.yamlを参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ユースケース

ユースケースで利用できるトピックは次のとおりです。

- IPAMコントローラーを使用したIPアドレス管理
- イングレスの保護
- TCPユースケース
- HTTPユースケース
- HTTPコールアウト
- セッションアフィニティ
- IPアドレスの許可リストまたはブロックリスト
- ExternalDNSとの相互運用性
- Vaultサーバーに保存されているNetScaler資格情報の使用
- NetScaler資格情報を保存するためのKubernetesシークレットの使用
- UDPベースのアプリケーションへのイングレストラフィックの負荷分散
- 2層展開のセットアップ
- 水平ポッドオートスケーラー
- ダイレクトサーバーリターン(/ja-jp/netscaler-k8s-ingress-controller/how-to/dsr.html)
- アドミッションコントローラーWebhook(/ja-jp/netscaler-k8s-ingress-controller/how-to/webhook-use-case.html)
- GRPCサポートの有効化(/ja-jp/netscaler-k8s-ingress-controller/how-to/grpc.html)
- ポリシーベースルーティング(/ja-jp/netscaler-k8s-ingress-controller/how-to/pbr.html)
- MongoDBソリューション(/ja-jp/netscaler-k8s-ingress-controller/how-to/mongodb-solution.html)
- アジュールパイプラインズを使用したカナリアおよびブルー/グリーンデプロイメント
- 外部サービス向けトラフィック管理(/ja-jp/netscaler-k8s-ingress-controller/how-to/external-load-balance-adc.html)
- 名前空間をまたがる外部ネームサービスサポート(/ja-jp/netscaler-k8s-ingress-controller/how-to/external-name-service-support.html)

© 1997–2026 Citrix Systems, Inc. All rights reserved.

IPAMコントローラーを使用したIPアドレス管理

IPAMコントローラーは、NetScalerが提供するIPアドレス管理用のコンテナであり、Kubernetesクラスター内のポッドとしてNetScaler Ingress Controllerと並行して実行されます。LoadBalancerタイプのサービスの場合、IPAMコントローラーを使用して、指定されたIPアドレス範囲からサービスにIPアドレスを自動的に割り当てることができます。このIP範囲は、YAMLを使用してIPAMコントローラーを展開する際にYAMLファイルで指定できます。NetScaler Ingress Controllerは、サービスに割り当てられたIPアドレスをNetScaler MPXまたはVPXの仮想IPアドレス（VIP）として構成します。 このIPアドレスを使用して、サービスに外部からアクセスできます。

LoadBalancerタイプのサービスの概要

Kubernetes環境では、マイクロサービスは動的に作成および破棄される一連のポッドとして展開されます。マイクロサービスを参照するポッドのセットは常に変化するため、Kubernetesは、一連のポッドで実行されているマイクロサービスを公開するためのサービスとして知られる論理的な抽象化を提供します。サービスは、論理的なポッドのセットと、それらにアクセスするためのポリシーを定義します。

LoadBalancerタイプのサービスは、Kubernetesクラスター内のマイクロサービスを外部に公開する最も簡単な方法です。LoadBalancerタイプのサービスは、AWS、GCP、Azureなどのパブリッククラウド上のKubernetes展開でネイティブにサポートされています。クラウド展開では、LoadBalancerタイプのサービスを作成すると、クラウド管理のロードバランサーがサービスに割り当てられます。その後、サービスはロードバランサーを使用して公開されます。

LoadBalancerタイプのサービス向けNetScaler IPAMソリューション

Kubernetesクラスターをパブリッククラウドに展開するのではなく、ベアメタルまたはオンプレミスに展開したい状況がいくつかあるかもしれません。ベアメタルKubernetesクラスターでアプリケーションを実行している場合、イングレスを使用するよりも`LoadBalancer`タイプのサービスを使用してTCPまたはUDPトラフィックをルーティングする方がはるかに簡単です。HTTPトラフィックの場合でも、イングレスよりも便利な場合があります。ただし、ベアメタルKubernetesクラスターには、ロードバランサーの実装がネイティブに利用できません。NetScalerは、NetScaler Ingress ControllerとNetScalerを使用して、このようなサービスの負荷分散を行う方法を提供します。

`LoadBalancer`タイプのサービス向けNetScalerソリューションでは、Kubernetesクラスター内に展開されたNetScaler Ingress Controllerが、クラスター外に展開されたNetScalerを構成して、受信トラフィックの負荷分散を行います。NetScalerソリューションを使用すると、展開がベアメタル、オンプレミス、またはパブリッククラウドのいずれであるかに関係なく、Kubernetesクラスターへの受信トラフィックの負荷分散を行うことができます。NetScaler Ingress Controllerは、

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScalerのマルチテナンシーを可能にする柔軟なIPアドレス管理を提供するため、単一のNetScalerを使用して複数のサービスの負荷分散を行うだけでなく、イングレス機能も実行できます。したがって、ロードバランサーリソースの利用率を最大化し、運用コストを大幅に削減できます。

IPAMコントローラーを使用したIPアドレス管理

NetScaler IPAMコントローラーには、NetScalerが提供するVIP CRDが必要です。VIP CRDには、サービス名、名前空間、およびIPアドレスのフィールドが含まれています。VIP CRDは、NetScaler Ingress ControllerとIPAMコントローラー間の内部通信に使用されます。

次の図は、IPAMコントローラーを使用してサービスにIPアドレスを割り当てるサービスタイプロードバランサーの展開を示しています。

`Loadbalancer`タイプの新しいサービスが作成されると、次のイベントが発生します。

1. サービス内の`loadBalancerIP`フィールドが空の場合、NetScaler Ingress Controllerはサービス用のVIP CRDオブジェクトを作成します。
2. IPAMコントローラーは、VIP CRDオブジェクトにIPアドレスを割り当てます。
3. VIP CRDオブジェクトがIPアドレスで更新されると、NetScaler Ingress ControllerはNetScalerを自動的に構成します。

注:

NetScalerが提供するカスタムリソース定義 (CRD) は`LoadBalancer`タイプのサービスもサポートしています。つまり、CRDオブジェクトを作成し、そのCRDをサービスに適用する際に、サービス名として`LoadBalancer`タイプのサービスを指定できます。

IPAMコントローラーによって割り当てられたIPアドレスを持つLoadBalancerタイプのサービスを公開する

このトピックでは、IPAMコントローラーによって割り当てられたIPアドレスを持つLoadBalancerタイプのサービスを公開する方法について説明します。

IPAMコントローラーからのIPアドレスを持つロードバランサタイプのサービスを公開するには、次の手順を実行します。

1. NetScaler Ingress Controllerを展開する。
2. NetScaler IPAMコントローラーを展開する。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

3. サンプルアプリケーションを展開する。
4. LoadBalancerタイプのサービスを使用してサンプルアプリケーションを公開する。
5. サービスにアクセスする。

NetScaler Ingress Controllerを展開する

IPAMコントローラー引数を使用してNetScaler Ingress Controllerを展開するには、次の手順を実行します。

1. 次のコマンドを使用して、NetScaler® Helmチャートリポジトリをローカルレジストリに追加します。

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/
```

2. 次のコマンドを使用して、NetScaler Ingress Controllerをインストールします。

```
helm install netscaler-ingress-controller netscaler/netscaler-ingress-controller --set nsIP=<NS_IP>,license.accept=yes,adcCredentialSecret=<>,ingressClass[0]=netscaler,serviceClass[0]=netscaler,ipam=true -n netscaler
```

Helmチャートを使用してNetScaler Ingress Controllerを展開および構成する方法の詳細については、Helmチャートリポジトリを参照してください。

IPAMコントローラーのデプロイ

1. 次のコマンドを使用して、NetScaler Helmチャートリポジトリをローカルレジストリに追加します。

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/
```

2. 次のコマンドを使用して、NetScaler IPAMコントローラーをインストールします。

```
helm install netscaler-ipam-controller netscaler/netscaler-ipam-controller --set vipRange='[{"Prod": ["10.1.2.0 - 10.1.2.255"]}]' -n netscaler
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サンプルアプリケーションのデプロイ

Kubernetesクラスターに`apache`アプリケーションをデプロイするには、次の手順を実行します。

注:

この例では、`apache`アプリケーションを使用しています。任意のサンプルアプリケーションをデプロイできます。

1. 次のコマンドを使用してサンプルアプリケーションをデプロイします。

```
kubectl apply -f - <<EOF

apiVersion: apps/v1

kind: Deployment

metadata:

  name: apache

  namespace: netscaler

  labels:

    name: apache

spec:

  selector:

    matchLabels:

      app: apache

  replicas: 2

  template:

    metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      labels:
        app: apache
    spec:
      containers:
      - name: apache
        image: httpd:latest
        ports:
        - name: http
          containerPort: 80
        imagePullPolicy: IfNotPresent
  EOF
  ```
```

2. 次のコマンドを使用して、Podが実行されていることを確認します。

```
kubectl get pods
```

LoadBalancerタイプのサービスを使用してサンプルアプリケーションを公開する

`LoadBalancer`タイプのサービスを作成するには、次の手順を実行します。

1. IPAMインストール時に指定された`Prod` VIP範囲からIPアドレスが割り当てられる`apache`アプリケーションを公開するサービスをデプロイします。

```
  kubectl apply -f - <<EOF
  apiVersion: v1
  kind: Service
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.
```

```
  metadata:
    name: apache
    namespace: netscaler
    labels:
      name: apache
    annotations:
      service.citrix.com/class: 'netscaler'
      service.citrix.com/ipam-range: 'Prod'
  spec:
    externalTrafficPolicy: Local
    type: LoadBalancer
    ports:
    - name: http
      port: 80
      targetPort: http
    selector:
      app: apache


  EOF
```

サービスを作成すると、IPAMコントローラーは、IPAMコントローラーのデプロイで定義したIPアドレス範囲から`apache`サービスにIPアドレスを割り当てます。IPAMコントローラーによって割り当てられたIPアドレスは、サービス定義の`status.loadBalancer.ingress:`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フィールドに提供されます。NetScaler Ingress Controllerは、サービスに割り当てられたIPアドレスをNetScalerの仮想IPアドレス（VIP）として構成します。

2. 次のコマンドを使用してサービスを表示します。

```
kubectl get service apache --output yaml
```

出力:

サービスへのアクセス

IPAMコントローラーによってサービスに割り当てられたIPアドレスを使用して、`apache` サービスにアクセスできます。IPアドレスは、サービス定義の`status.loadBalancer.ingress:`フィールドで確認できます。サービスにアクセスするには、以下のcurlコマンドを使用します。

```
curl <IP_address>
```

応答は次のようになります:

```
<html><body><h1>It works!</h1></body></html>
```

Ingress の保護

このトピックでは、NetScaler を使用して Ingress を保護するためのさまざまな方法と、NetScaler Ingress Controller が提供するアノテーションについて説明します。

次の表に、Ingress NetScaler と NetScaler Ingress Controller を使用して Ingress を保護するために使用できる、サンプルアノテーションを含む TLS のユースケースを示します。

ユースケース	サンプルアノテーション
TLSv1.3 プロトコルを有効にする	`ingress.citrix.com/frontend-sslprofile: '{"tls13":"enabled", "tls13sessionticketsperauthcontext":"1", "dhekeyexchangewithpsk":"yes"}'`
HTTP厳格なトランスポートセキュリティ (HSTS)	`ingress.citrix.com/frontend-sslprofile: '{"hsts":"enabled", "maxage" : "157680000", "includesubdomain":"yes"}`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ユースケース	サンプルアノテーション
OCSP ステープリング	`ingress.citrix.com/frontend-sslprofile: '{"ocspstapling":"enabled"}'`
クライアント認証を必須に設定する	`ingress.citrix.com/frontend-sslprofile: '{"clientauth":"enabled", "clientcert" : "mandatory"}'`
TLS セッションチケット拡張	`ingress.citrix.com/frontend-sslprofile: '{"sessionticket" : "enabled", "sessionticketlifetime : "300"}'`
SSLセッション再利用	`ingress.citrix.com/frontend-sslprofile: '{"sessreuse" : "enabled", "sesstimeout : "120"}'`
暗号グループ	`ingress.citrix.com/frontend-sslprofile:'{"snienable": "enabled", "ciphers" : [{"ciphername": "secure", "cipherpriority" :"1"}, {"ciphername": "secure", "cipherpriority" :"21"}]}'`
暗号リダイレクト	`ingress.citrix.com/frontend-sslprofile:'{"snienable": "enabled", "ciphers" : [{"ciphername": "secure", "cipherpriority" :"1"}], "cipherredirect":"enabled", "cipherurl": "https://redirecturl"}'`

TLS v1.3プロトコルを有効にする

SSLプロファイルのアノテーションを使用して、SSLプロファイルでTLS 1.3プロトコルサポートを有効にし、Ingress NetScalerのSSLプロファイルで`tls13SessionTicketsPerAuthContext`と`dheKeyExchangeWithPsk`パラメーターを設定できます。

`tls13SessionTicketsPerAuthContext`パラメーターを使用すると、TLS 1.3がネゴシエートされ、チケットベースの再開が有効になり、ハンドシェイクが完了するか、ハンドシェイク後のクライアント認証が完了するたびに、Ingress NetScalerが発行するチケットの数を設定できます。この値を増やすことで、クライアントが各接続に新しいチケットを使用して複数の並列接続を開く

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ことができるようになります。設定できる最小値は1、最大値は10です。デフォルトでは、値は1に設定されています。

注:

再開が無効になっている場合、チケットは送信されません。

`dheKeyExchangeWithPsk`パラメーターを使用すると、TLS 1.3セッション再開ハンドシェイク中に事前共有キーが受け入れられたときに、`Ingress NetScaler requires a DHE`キー交換が発生するかどうかを指定できます。DHEキー交換は、チケットキーが侵害された場合でも前方秘匿性を確保しますが、`DHE`キー交換を実行するために追加のリソースが必要になります。

以下は、SSLプロファイルでTLS 1.3プロトコルサポートを有効にし、SSLプロファイルで`tls13SessionTicketsPerAuthContext`と`dheKeyExchangeWithPsk`パラメーターを設定するためのHTTPプロファイルのサンプルアノテーションです。

```
ingress.citrix.com/frontend-sslprofile: '{"tls13":"enabled",
"tls13sessionticketsperauthcontext":"1", "dhekeyexchangewithpsk":"yes"}'
```

HTTP厳格トランスポートセキュリティ (HSTS)

Ingress NetScalerアプライアンスは、SSLプロファイルに組み込みオプションとしてHTTP Strict Transport Security (HSTS) をサポートしています。HSTSを使用すると、サーバーはクライアントとのすべての通信にHTTPS接続の使用を強制できます。つまり、サイトにはHTTPSを使用することによってのみアクセスできます。HSTSのサポートは、SSL LabsからのA+認定に必要です。詳細については、NetScalerでのHSTSのサポートを参照してください。

SSLプロファイルのアノテーションを使用して、Ingress NetScalerのSSLフロントエンドプロファイルでHSTSを有効にできます。以下は、Ingressアノテーションの例です。

```
ingress.citrix.com/frontend-sslprofile: '{"hsts":"enabled", "maxage" :
"157680000", "includesubdomain":"yes"}'
```

ここで、:

- `HSTS` - SSLプロファイル上のHTTP Strict Transport Security (HSTS) の状態。HSTSを使用すると、サーバーはクライアントとのすべての通信にHTTPS接続の使用を強制できます。サポートされている値はENABLEDとDISABLEDです。デフォルトでは、値はDISABLEDに設定されています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- `maxage` - クライアントがサーバーにHTTPSリクエストのみを送信する必要がある厳格なトランスポートセキュリティ (STS) ヘッダーの最大時間（秒単位）を設定できます。設定できる最小時間は0で、最大は4294967294です。デフォルトでは値は0です。
- `IncludeSubdomains` - サブドメインのHSTSを有効にできます。`Yes` に設定されている場合、クライアントはサブドメインに対してHTTPSリクエストのみを送信する必要があります。デフォルトでは値はNoに設定されています。

OCSPステープリング

Ingress NetScalerは、OCSPレスポンダーから証明書の状態を検証した後、SSLハンドシェイク時にサーバー証明書の失効ステータスをクライアントに送信できます。サーバー証明書の失効ステータスは、アプライアンスがSSLハンドシェイクの一部としてクライアントに送信する応答に「ステープル」されます。NetScalerでのCRLおよびOCSPレポートの実装の詳細については、OCSPステープリングを参照してください。

OCSPステープリング機能を使用するには、以下のIngressアノテーションを持つSSLプロファイルを使用して有効にできます。

```
ingress.citrix.com/frontend-sslprofile: '{"ocspstapling":"enabled"}'
```

注:

OCSPステープリングを使用するには、NetScalerアプライアンスにOCSPレスポンダーを追加する必要があります。

クライアント認証を必須に設定する

SSLプロファイルのアノテーションを使用すると、クライアント認証を有効にできます。Ingress NetScalerアプライアンスは、SSLハンドシェイク中にクライアント証明書を要求します。

アプライアンスは、発行者の署名や有効期限などの通常の制約について、クライアントから提示された証明書をチェックします。

以下にいくつかのユースケースを示します。

- ウェブサイトのコンテンツが表示される前に有効なクライアント証明書を要求します。これにより、ウェブサイトのコンテンツは認証されたマシンとユーザーのみに制限されます。
- 有効なクライアント証明書を要求します。有効なクライアント証明書が提供されない場合、ユーザーに多要素認証を求めます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

クライアント認証は、必須またはオプションに設定できます。

- 必須に設定されている場合、SSLクライアントが有効なクライアント証明書を送信しないと、接続は切断されます。有効とは、特定の認証局によって署名/発行され、期限切れまたは失効していないことを意味します。
- オプションの場合、NetScalerはクライアント証明書を要求しますが、クライアントが無効な証明書を提示した場合や証明書を提示しなかった場合でも、SSLトランザクションを続行します。この構成は、認証シナリオ（例えば、有効なクライアント証明書が提供されない場合に2要素認証を要求するなど）に役立ちます。

SSLプロファイルの注釈を使用すると、SSL仮想サーバーでクライアント認証を有効にし、クライアント認証を`Mandatory`として設定できます。

以下は、SSLプロファイルの注釈の例です。

```
ingress.citrix.com/frontend-sslprofile: '{"clientauth":"enabled",
"clientcert" : "mandatory"}'
```

注:

クライアント証明書をIngress NetScaler上のSSL仮想サーバーにバインドしていることを確認してください。

TLSセッションチケット拡張

SSLハンドシェイクはCPU負荷の高い操作です。セッションの再利用が有効になっている場合、既存のクライアントに対してはサーバーまたはクライアントの鍵交換操作がスキップされます。これにより、クライアントはセッションを再開できます。これは応答時間を改善し、サーバーがサポートできる1秒あたりのSSLトランザクション数を増加させます。ただし、サーバーは各セッション状態の詳細を保存する必要があり、これはメモリを消費し、リクエストが複数のサーバー間で負荷分散される場合に、それらのサーバー間で共有することが困難です。

Ingress NetScalerアプライアンスは、SessionTicket TLS拡張をサポートしています。この拡張の使用は、セッションの詳細がサーバーではなくクライアントに保存されることを示します。クライアントは、クライアントHelloメッセージにセッションチケットTLS拡張を含めることで、このメカニズムをサポートしていることを示す必要があります。新しいクライアントの場合、この拡張は空です。サーバーはNewSessionTicketハンドシェイクメッセージで新しいセッションチケット

© 1997–2026 Citrix Systems, Inc. All rights reserved.

を送信します。セッションチケットは、サーバーのみが知っているキーペアを使用して暗号化されます。サーバーが現在新しいチケットを発行できない場合、通常のハンドシェイクを完了します。

SSLプロファイルの注釈を使用すると、RFC 5077に従ってセッションチケットの使用を有効にできます。また、Ingress NetScalerによって発行されるセッションチケットの有効期間を、`sessionticketlifetime`パラメーターを使用して設定できます。

以下は、Ingress注釈の例です。

```
ingress.citrix.com/frontend-sslprofile: '{"sessionticket" : "enabled",
"sessionticketlifetime : "300"}'
```

SSLセッションの再利用

NetScalerアプライアンスで既存のSSLセッションを再利用できます。SSL再ネゴシエーションプロセスが完全なSSLハンドシェイクで構成されるのに対し、SSL再利用はクライアントがリクエストとともにSSL IDを送信するため、部分的なハンドシェイクで構成されます。

SSLプロファイルの注釈を使用して、セッションの再利用を有効にし、Ingress NetScalerでセッションタイムアウト値（秒単位）を設定できます。

以下は、Ingressアノテーションの例です。

```
ingress.citrix.com/frontend-sslprofile: '{"sessreuse" : "enabled",
"sesstimeout : "120"}'
```

デフォルトでは、アプライアンスでセッション再利用オプションが有効になっており、そのタイムアウト値は120秒に設定されています。したがって、クライアントが別のTCP接続で、以前のSSLセッションIDを120秒以内に送信した場合、アプライアンスは部分的なハンドシェイクを実行します。

暗号グループの使用

Ingress NetScalerには、組み込みの暗号グループが付属しています。DEFAULT暗号グループの一部ではない暗号を使用するには、それらをSSLプロファイルに明示的にバインドする必要があります。Ingress NetScaler上のSSL仮想サーバーにバインドするために、ユーザー定義の暗号グループを作成することもできます。

組み込みの暗号グループはTier-1およびTier-2 NetScalerで使用でき、ユーザー定義の暗号グループはTier-1 NetScalerでのみ使用できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ユーザー定義の暗号グループを使用するには、NetScalerにユーザー定義の暗号グループがあることを確認してください。次の手順を実行します。

1. ユーザー定義の暗号グループを作成します。例: `testgroup`。
2. 必要なすべての暗号をユーザー定義の暗号グループにバインドします。
3. ユーザー定義の暗号グループ名をメモします。

詳細な手順については、ユーザー定義の暗号グループを構成するを参照してください。

SSLプロファイルの注釈を使用して、組み込みの暗号グループ、ユーザー定義の暗号グループ、またはその両方をSSLプロファイルにバインドできます。

以下は、組み込みの暗号グループとユーザー定義の暗号グループをSSLプロファイルにバインドするために使用できるIngressアノテーションの構文です。

```
ingress.citrix.com/frontend-sslprofile:'{"snienable":"enabled",
"ciphers" : [{"ciphername": "secure", "cipherpriority" :"1"},
{"ciphername": "testgroup", "cipherpriority" :"2"}]}'
```

Ingressアノテーションは、組み込みの暗号グループ`SECURE`とユーザー定義の暗号グループ`testgroup`をSSLプロファイルにバインドします。

暗号リダイレクトの使用

SSLハンドシェイク中に、SSLクライアント（通常はWebブラウザ）は、設定された暗号の優先順位で、サポートする暗号スイートを通知します。そのリストから、SSLサーバーは自身が設定している暗号のリストと一致する暗号を選択します。

クライアントが通知した暗号がSSLサーバーに設定されている暗号と一致しない場合、SSLハンドシェイクは失敗します。失敗は、ブラウザに表示される不可解なエラーメッセージによって通知されます。これらのメッセージがエラーの正確な原因に言及することはほとんどありません。

暗号リダイレクトを使用すると、SSLハンドシェイクが失敗したときに、正確で意味のあるエラーメッセージを配信するようにSSL仮想サーバーを設定できます。SSLハンドシェイクが失敗すると、NetScalerアプライアンスはユーザーを以前に設定されたURLにリダイレクトするか、URLが設定されていない場合は、内部で生成されたエラーページを表示します。

以下は、暗号グループをバインドし、暗号リダイレクトを有効にしてリクエストを`redirecturl`にリダイレクトするために使用できるイングレスアノテーションの構文です。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
ingress.citrix.com/frontend-sslprofile:'{"snienable": "enabled",
"ciphers" : [{"ciphername": "secure", "cipherpriority" :"1"}],
"cipherredirect":"enabled", "cipherurl": "https://redirecturl"}'
```

TCPのユースケース

Kubernetes環境では、イングレスオブジェクトにより、Kubernetesクラスターの外部からKubernetesサービスへのアクセスが可能になります。標準のKubernetesイングレスリソースは、すべてのトラフィックがHTTPベースであると想定しており、TCP、UDP、SSLなどの非HTTPベースのプロトコルには対応していません。したがって、DNS、FTP、LDAPなどの非HTTPアプリケーションは、標準のKubernetesイングレスを使用して公開することはできません。

TCPイングレストラフィックを負荷分散する方法

NetScalerは、イングレスアノテーションを使用してTCPベースのイングレストラフィックを負荷分散するソリューションを提供します。イングレスリソース定義でこれらのアノテーションを指定すると、NetScaler Ingress ControllerはNetScalerを構成してTCPイングレストラフィックを負荷分散します。

TCPベースのイングレストラフィック

TCPベースのイングレストラフィックを負荷分散するには、Kubernetesイングレスリソース定義で次のアノテーションを使用できます。

- `ingress.citrix.com/insecure-service-type`: このアノテーションは、NetScalerでTCPを使用したL4負荷分散を有効にします。
- `ingress.citrix.com/insecure-port`: このアノテーションはTCPトラフィックのポートを構成し、非標準ポートでマイクロサービスアクセスが必要な場合に役立ちます。デフォルトでは、ポート80が構成されています。

イングレスアノテーションの詳細については、Ingress annotationsを参照してください。

サンプル: TCPベースのイングレスのイングレス定義。

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  annotations:

    ingress.citrix.com/frontend-ip: "192.168.1.1" # If you are using IPAM
then use ingress.citrix.com/ipam-range: "<IPAM Range Name>"

    ingress.citrix.com/insecure-port: '6379'

    ingress.citrix.com/insecure-service-type: tcp

  name: redis-master-ingress

spec:

  ingressClassName: guestbook

  defaultBackend:

    service:

      name: redis-master-pods

      port:

        number: 6379

---

apiVersion: networking.k8s.io/v1

kind: IngressClass

metadata:

  name: guestbook

spec:

  controller: citrix.com/ingress-controller

EOF
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

LoadBalancerタイプのサービスに対するTCPベースのトラフィック

サービス定義YAMLで次のサービスアノテーションを使用して、TCPベースのトラフィックを負荷分散できます: `service.citrix.com/service-type-<index>`。サービスアノテーションの詳細については、Service annotationsを参照してください。

サンプル: TCPベースのトラフィックを負荷分散するためのService type LoadBalancer YAML。

```
apiVersion: v1

kind: Service

metadata:

  name: backend

  annotations:

      service.citrix.com/class: 'netscaler' # During deployment of NSIC,
set `serviceClass` with 'netscaler'.

      service.citrix.com/frontend-ip: '192.168.1.10' # If you are using
IPAM then use, service.citrix.com/ipam-range: "<IPAM Range Name>"

      service.citrix.com/service-type-0: TCP

  labels:

      app: backend

spec:

  ports:

    - name: port-6379

      port: 6379

      targetPort: 6379

  type: LoadBalancer

  selector:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    name: backend
```

SSL over TCPに基づいたイングレストラフィックの負荷分散

NetScaler Ingress Controllerは、SSL over TCPに基づいたイングレストラフィックを負荷分散するために使用できる`ingress.citrix.com/secure-service-type: ssl_tcp`アノテーションを提供します。

サンプル: TCPベースのIngressにおけるSSL用Ingress定義。

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  annotations:

    ingress.citrix.com/frontend-ip: "192.168.1.1" # If you are using IPAM
then use ingress.citrix.com/ipam-range: "<IPAM Range Name>"

    ingress.citrix.com/secure-service-type: "ssl_tcp"

    ingress.citrix.com/secure-backend: '{"frontendcolddrinks":"True"}'

  name: colddrinks-ingress

spec:

  ingressClassName: colddrink

  defaultBackend:

    service:

      name: frontendcolddrinks

      port:

        number: 443
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  tls:
  - secretName: "colddrink-secret"
---
apiVersion: networking.k8s.io/v1
kind: IngressClass
metadata:
  name: colddrink
spec:
  controller: citrix.com/ingress-controller
EOF
```

TCPベースのアプリケーションのパフォーマンスを監視し、改善する

アプリケーション開発者は、NetScalerの豊富なモニター（TCP-ECVなど）を通じて、TCPベースのアプリケーションの健全性を綿密に監視できます。ECV（拡張コンテンツ検証）モニターは、アプリケーションが期待されるコンテンツを返すかどうかを確認するのに役立ちます。NetScaler Ingress Controllerは、バックエンドサービスの健全性を監視するために使用できる `ingress.citrix.com/monitor` アノテーションを提供します。NetScalerの組み込みモニターの詳細については、組み込みモニターを参照してください。

また、`Source IP` などの永続化メソッドを使用することで、アプリケーションのパフォーマンスを向上させることができます。これらのNetScaler機能は、Kubernetesのスマートアノテーションを通じて使用できます。

以下のIngressリソースの例では、スマートアノテーションを使用しています。

```
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: Ingress
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
metadata:

  annotations:

    ingress.citrix.com/frontend-ip: "192.168.1.1" # If you are using IPAM
then use ingress.citrix.com/ipam-range: "<IPAM Range Name>"

    ingress.citrix.com/insecure-port: "80"

    ingress.citrix.com/lbvserver: '{"mongodb-svc":
{"lbmethod":"SRCIPDESTIPHASH"}}'

    ingress.citrix.com/monitor: '{"mongodbsvc":{"type":"TCP-ECV"}}'

  name: mongodb

spec:

  rules:

  - host: mongodb.beverages.com

    http:

      paths:

      - backend:

          service:

            name: mongodb-svc

            port:

              number: 80

        path: /

        pathType: Prefix

EOF
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler CPXサービスで非標準HTTPポートを公開する方法

NetScaler CPXサービスで、他のポートでTCPトラフィックを許可するために、80番ポートと443番ポート以外のポートを公開する必要がある場合があります。 このセクションでは、KubernetesクラスターにNetScaler CPXサービスをデプロイする際に、NetScaler CPXサービスで他の非標準HTTPポートを公開する方法について説明します。

Helmチャートのデプロイの場合

Helmチャートを使用してIngress ControllerとともにNetScaler CPXをデプロイする際に非標準HTTPポートを公開するには、Helmチャートインストールガイドを参照してください。

```
servicePorts:

  - port: 80

    protocol: TCP

    name: http

  - port: 443

    protocol: TCP

    name: https

  - port: 6379

    protocol: TCP

    name: tcp
```

OpenShift Operatorを使用するデプロイの場合

OpenShift Operatorを使用するデプロイの場合、NetScaler Operatorを使用してNetScaler CPXをサイドカーとしてNetScaler Ingress Controllerをデプロイするのステップ6で指定されているように、Ingress ControllerとともにCPXを作成するためにYAML定義を編集し、`service`セクションの以下のコードブロックで指定されているようにポートを指定する必要があります。

```
service:

  annotation: {}
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:
  type: NodePort # ClusterIP or LoadBalancer
  ports:
  - port: 80
    targetPort: 80
    protocol: TCP
    name: http
  - port: 443
    targetPort: 443
    protocol: TCP
    name: https
```

TCPプロファイルのサポート

このセクションでは、NetScaler Ingress Controllerのアノテーションを使用して、`LoadBalancer`タイプおよびIngressタイプのサービスに対してスマートアノテーションを使用してNetScalerでTCPパラメーターを構成するさまざまな方法について説明します。

TCPプロファイルは、TCP設定の集合です。各エンティティで設定を構成する代わりに、プロファイルでTCP設定を構成し、必要なすべてのエンティティにプロファイルをバインドできます。フロントエンドTCPプロファイルはクライアント側のコンテンツスイッチング仮想サーバーにアタッチでき、バックエンドTCPプロファイルはサービスグループ用に構成できます。

LoadBalancerタイプのサービスに対するTCPプロファイルのサポート

NetScaler Ingress Controllerは、`LoadBalancer`タイプのサービス向けTCPプロファイルに対して、以下のサービスアノテーションを提供します。これらのアノテーションを使用して、NetScalerのTCP設定を定義できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サービスアノテーション	説明
`service.citrix.com/frontend-tcpprofile`	このアノテーションを使用して、フロントエンドTCPプロファイル（クライアントプレーン）を作成します。
`service.citrix.com/backend-tcpprofile`	このアノテーションを使用して、バックエンドTCPプロファイル（サーバープレーン）を作成します。

ユーザー定義TCPプロファイル

TCPのサービスアノテーションを使用すると、コンテンツスイッチング仮想サーバーまたはサービスグループと同じ名前でカスタムプロファイルを作成し、それらを対応する仮想サーバー (`frontend-tcpprofile`) およびサービスグループ (`backend-tcpprofile`) にバインドできます。

サービスアノテーション	サンプル
`service.citrix.com/frontend-tcpprofile`	`service.citrix.com/frontend-tcpprofile: '{"ws" : "enabled", "sack" : "enabled"}'`
`service.citrix.com/backend-tcpprofile`	`service.citrix.com/backend-tcpprofile: '{"ws" : "enabled", "sack" : "enabled"}'`

組み込みTCPプロファイル

組み込みTCPプロファイルは、プロファイルを作成するのではなく、アノテーションで指定されたプロファイル名を対応する仮想サーバー (frontend-tcpprofile) およびサービスグループ (backend-tcpprofile) にバインドします。

組み込みTCPプロファイルの例：

```
service.citrix.com/frontend-tcpprofile: "tcp_preconf_profile"
```

```
service.citrix.com/backend-tcpprofile: '{"citrix-svc" : "tcp_preconf_profile"}
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

例：TCPプロファイル構成を持つLoadBalancerタイプのサービス

この例では、サンプルアプリケーション `tea-beverage` 用にTCPプロファイルが構成されています。このアプリケーションは、YAMLファイルを使用してLoadBalancerタイプのサービスとしてデプロイおよび公開されます。

注：

`LoadBalancer` タイプのサービスを公開する方法については、「`LoadBalancer` タイプのサービス」を参照してください。

次のコマンドを使用してサンプルアプリケーション (tea-beverage.yaml) をデプロイします。

```
kubectl apply -f - <<EOF

apiVersion: apps/v1

kind: Deployment

metadata:

  name: tea-beverage

  labels:

    name: tea-beverage

spec:

  selector:

    matchLabels:

      name: tea-beverage

  replicas: 2

  template:

    metadata:

      labels:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        name: tea-beverage
    spec:
      containers:
      - name: tea-beverage
        image: quay.io/sample-apps/hotdrinks:latest
        ports:
        - name: tea-80
          containerPort: 80
        - name: tea-443
          containerPort: 443
        imagePullPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: tea-beverage
  annotations:
    service.citrix.com/frontend-ip: 10.105.158.194 # If you are using IPAM
then use, service.citrix.com/ipam-range: "<IPAM Range Name>"
    service.citrix.com/frontend-tcpprofile: '{"ws" : "enabled", "sack" :
"enabled"}'
    service.citrix.com/backend-tcpprofile: '{"ws" : "enabled", "sack" :
"enabled"}'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:
  type: LoadBalancer
  ports:
  - name: tea-80
    port: 80
    targetPort: 80
  selector:
    name: tea-beverage
EOF
```

アプリケーションがデプロイされると、対応するエンティティとプロファイルがNetScaler上に作成されます。NetScalerで次のコマンドを実行して、これを確認します：`show cs vserver k8s-tea-beverage_80_default_svc` および `show servicegroup k8s-tea-beverage_80_sgp_f4lezsannvu7tk2ftpjbhi4hza2tvdnk`。

```
      # show cs vserver k8s-tea-beverage_80_default_svc
        k8s-tea-beverage_80_default_svc (10.105.158.194:80) - TCP Type:
CONTENT
        State: UP
        Last state change was at Wed Apr  3 09:37:59 2024
        Time since last state change: 0 days, 00:00:09.790
        Client Idle Timeout: 9000 sec
        Down state flush: ENABLED
        Disable Primary Vserver On Down : DISABLED
        Comment:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
uid=VIGQWRCYKCM6WFYX2GFKRVT3ZF6JSFISPW6XM24JADBXEYRLITOQ====

        **TCP profile name: k8s-tea-beverage_80_default_svc**

        Appflow logging: ENABLED

        State Update: DISABLED

        Default: k8s-tea-beverage_80_lbv_f4lezsannvu7tk2ftpjbhi4hza2tvdnk
Content Precedence: RULE

       L2Conn: OFF Case Sensitivity: ON

        Authentication: OFF

        401 Based Authentication: OFF

        HTTP Redirect Port: 0 Dtls : OFF

        Persistence: NONE

        Listen Policy: NONE

        IcmpResponse: PASSIVE

        RHIstate:  PASSIVE

        Traffic Domain: 0



        1) Default Target LB: k8s-tea-
beverage_80_lbv_f4lezsannvu7tk2ftpjbhi4hza2tvdnk Hits: 0

        Done
```

```
      # show servicegroup k8s-tea-
beverage_80_sgp_f4lezsannvu7tk2ftpjbhi4hza2tvdnk

        k8s-tea-beverage_80_sgp_f4lezsannvu7tk2ftpjbhi4hza2tvdnk - TCP

        State: ENABLED Effective State: UP Monitor Threshold : 0
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        Max Conn: 0 Max Req: 0 Max Bandwidth: 0 kbits

        Use Source IP: NO

        Client Keepalive(CKA): NO

        Monitoring Owner: 0

        TCP Buffering(TCPB): NO

        HTTP Compression(CMP): NO

        Idle timeout: Client: 9000 sec Server: 9000 sec

        Client IP: DISABLED

        Cacheable: NO

        SC: ???

        SP: OFF

        Down state flush: ENABLED

        Monitor Connection Close : NONE

        Appflow logging: ENABLED

        TCP profile name: k8s-tea-
beverage_80_sgp_f4lezsannvu7tk2ftpjbhi4hza2tvdnk

        ContentInspection profile name: ???

        Process Local: DISABLED

        Traffic Domain: 0

        Comment: "lbsvc:tea-beverage,svcport:80,ns:default"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        1)   10.146.107.38:30524 State: UP Server Name: 10.146.107.38
Server ID: None Weight: 1 Order: Default

          Last state change was at Wed Apr  3 09:38:00 2024

          Time since last state change: 0 days, 00:02:27.660



          Monitor Name: tcp-default State: UP Passive: 0

          Probes: 30 Failed [Total: 0 Current: 0]

          Last response: Success - TCP syn+ack received.

          Response Time: 0.000 millisec

      Done
```

注：

TCPプロファイルは、シングルポートサービスでサポートされています。

Ingressアノテーションを使用したTCPプロファイルの構成

次の表に、サンプルアノテーションを使用したTCPのユースケースの一部を示します。

ユースケース	サンプルアノテーション
アイドルTCP接続をサイレントに破棄する	`ingress.citrix.com/frontend-tcpprofile: '{"drophalfclosedconnontimeout" : "enabled", "dropestconnontimeout" : "enabled"}'` `ingress.citrix.com/backend-tcpprofile: '{"citrix-svc" : {"drophalfclosedconnontimeout" : "enabled",`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ユースケース	サンプルアノテーション	
	`"dropestconnontimeout" : "enabled"}}'`	
TCP接続の確認応答の遅延	`ingress.citrix.com/frontend-tcpprofile: '{"delayedack" : "150"}' ingress.citrix.com/backend-tcpprofile: '{"citrix-svc" : {"delayedack" : "150"}}'`	
クライアント側のマルチパスTCPセッション管理	`ingress.citrix.com/frontend-tcpprofile: '{"mptcp": "enabled", "mptcpsessiontimeout" : "7200"}' ingress.citrix.com/backend-tcpprofile: '{"citrix-svc" : {"mptcp": "enabled", "mptcpsessiontimeout" : "7200"}}'`	
TCP最適化		
選択的確認応答	`ingress.citrix.com/frontend_tcpprofile: '{"sack" : "enabled"}' ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"sack" : "enabled"}}'`	
前方確認応答	`ingress.citrix.com/frontend_tcpprofile: '{"fack" : "enabled"}' ingress.citrix.com/backend_tcpprofile:`	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ユースケース	サンプルアノテーション	
	`'{"citrix-svc" : {"fack" : "enabled"}}'`	
ウィンドウ スケーリング	`ingress.citrix.com/frontend_tcpprofile: '{"ws" : "enabled", "wsval" : "9"}' ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"ws" : "enabled", "wsval" : "9"}}'`	
最大セグメントサイズ	`ingress.citrix.com/frontend_tcpprofile: '{"mss" : "1460", "maxpktpermss" : "512"}' ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"mss" : "1460", "maxpktpermss" : "512"}}'`	
キープアライブ	`ingress.citrix.com/frontend_tcpprofile: '{"ka" : "enabled", "kaprobeupdatelastactivity" : "enabled", "kaconnidletime": "900", "kamaxprobes" : "3", "kaprobeinterval" : "75"}' ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"ka" : "enabled", "kaprobeupdatelastactivity" : "enabled",`	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ユースケース	サンプルアノテーション	
	`"kaconnidletime": "900", "kamaxprobes" : "3", "kaprobeinterval" : "75"}}'`	
bufferSize	`ingress.citrix.com/frontend_tcpprofile: '{"bufferSize" : "8190"}'` `ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"bufferSize" : "8190"}}'`	
MPTCP	`ingress.citrix.com/frontend_tcpprofile: '{"mptcp" : "enabled", "mptcpdropdataonpreestsf" : "enabled", "mptcpfastopen": "enabled", "mptcpsessiontimeout" : "7200"}'` `ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"mptcp" : "enabled", "mptcpdropdataonpreestsf" : "enabled", "mptcpfastopen": "enabled", "mptcpsessiontimeout" : "7200"}}'`	
フレーバー	`ingress.citrix.com/frontend_tcpprofile: '{"flavor" : "westwood"}'` `ingress.citrix.com/`	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ユースケース	サンプルアノテーション	
	`backend_tcpprofile: '{"citrix-svc" : {"flavor" : "westwood"}}'`	
動的受信バッファリング	`ingress.citrix.com/frontend_tcpprofile: '{"dynamicReceiveBuffering" : "enabled"}'` `ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"dynamicReceiveBuffering" : "enabled"}}'`	
スプーフィング攻撃に対するTCPの防御	`ingress.citrix.com/frontend_tcpprofile: '{"rstwindowattenuate" : "enabled", "spoofsyndrop" : "enabled"}` `ingress.citrix.com/backend-tcpprofile: '{"citrix-svc" : {"rstwindowattenuate" : "enabled", "spoofsyndrop" : "enabled"}}'`	

注:

前述のIngressアノテーションは、以前に説明した形式でサービスアノテーションとともに使用することもできます。

アイドル状態のTCP接続をサイレントに切断する

ネットワークにおいて、多くのTCP接続がアイドル状態になると、NetScalerはそれらを閉じるためにRSTパケットを送信します。チャネルを介して送信されるパケットは、それらのチャネルを

© 1997–2026 Citrix Systems, Inc. All rights reserved.

不必要にアクティブにし、メッセージの洪水を引き起こし、それがNetScalerにサービス拒否メッセージの洪水を生成させます。

TCPプロファイルの`drophalfclosedconnontimeout`および`dropestconnontimeout`パラメータを使用すると、アイドルタイムアウト時にTCPの半閉接続をサイレントに切断したり、アイドルタイムアウト時にTCPの確立済み接続を切断したりできます。デフォルトでは、これらのパラメータはNetScalerで無効になっています。両方を有効にすると、接続がタイムアウトしても、半閉接続も確立済み接続もクライアントにRSTパケットを送信することはありません。NetScalerは単に接続を切断します。

TCPプロファイルの注釈を使用すると、NetScalerで`drophalfclosedconnontimeout`と`dropestconnontimeout`を有効または無効にできます。以下は、これらのパラメータを有効にするTCPプロファイルの注釈の例です。

```
ingress.citrix.com/frontend-tcpprofile: '{"drophalfclosedconnontimeout" : "enable", "dropestconnontimeout" : "enable"}'
```

```
ingress.citrix.com/backend-tcpprofile: '{"citrix-svc" : {"drophalfclosedconnontimeout" : "enable", "dropestconnontimeout" : "enable"}}'
```

遅延TCP接続確認応答

複数のACKパケットの送信を避けるため、NetScalerはTCP遅延確認応答メカニズムをサポートしています。これは、デフォルトのタイムアウト100ミリ秒で遅延ACKを送信します。NetScalerはデータパケットを蓄積し、連続して2つのデータパケットを受信した場合、またはタイマーが期限切れになった場合にのみACKを送信します。TCP展開ACKに設定できる最小遅延は10ミリ秒、最大は300ミリ秒です。デフォルトでは、遅延は100ミリ秒に設定されています。

TCPプロファイルの注釈を使用すると、遅延ACKパラメータを管理できます。以下は、これらのパラメータを有効にするTCPプロファイルの注釈の例です。

```
ingress.citrix.com/frontend-tcpprofile: '{"delayedack" : "150"}'
```

```
ingress.citrix.com/backend-tcpprofile: '{"citrix-svc" : {"delayedack" : "150"}}'
```

クライアント側マルチパスTCPセッション管理

クライアントとNetScaler間のマルチパスTCP (MPTCP) 接続に対して、NetScalerでTCP設定を実行できます。NetScalerとバックエンド通信間ではMPTCP接続はサポートされていません。クライアントとNetScalerアプライアンスの両方が同じMPTCPバージョンをサポートしている必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScalerのTCPプロファイルを使用して、MPTCPを有効にし、MPTCPセッションタイムアウト(`mptcpsessiontimeout`)を秒単位で設定できます。`mptcpsessiontimeout`値が設定されていない場合、MPTCPセッションはクライアントアイドルタイムアウト後にフラッシュされます。設定できる最小タイムアウト値は0、最大は86400です。デフォルトでは、タイムアウト値は0に設定されています。

TCPプロファイルの注釈を使用すると、NetScalerでMPTCPを有効にし、`mptcpsessiontimeout`パラメータ値を設定できます。以下は、NetScalerでMPTCPを有効にし、`mptcpsessiontimeout`パラメータ値を7200に設定するTCPプロファイルの注釈の例です。

```
ingress.citrix.com/frontend-tcpprofile: '{"mptcp" : "enabled",
"mptcpsessiontimeout" : "7200"}'
```

```
ingress.citrix.com/backend-tcpprofile: '{"citrix-svc" : {"mptcp" : "enabled",
"mptcpsessiontimeout" : "7200"}}'
```

TCP最適化

NetScalerの関連するTCP最適化機能のほとんどは、対応するTCPプロファイルを通じて公開されています。TCPプロファイルの注釈を使用すると、NetScalerで以下のTCP最適化機能を有効にできます。

- `Selective acknowledgment (SACK)`: TCP SACKは、複数のパケット損失の問題に対処し、全体的なスループット容量を低下させます。選択的確認応答により、受信側は正常に受信されたすべてのセグメントを送信側に通知でき、送信側は失われたセグメントのみを再送信できます。この手法は、全体的なスループットを向上させ、接続遅延を削減するのに役立ちます。

  以下は、NetScalerでSACKを有効にするTCPプロファイルの注釈の例です。

  ```
  ingress.citrix.com/frontend_tcpprofile: '{"sack" : "enabled"}'
  ```

  ```
  ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"sack" :
  "enabled"}}'
  ```

- `Forward acknowledgment (FACK)`: ネットワーク内の未処理のデータバイトの総数を明示的に測定することでTCP輻輳を回避し、送信者（T1またはクライアントのいずれか）が再送信タイムアウト中にネットワークに注入するデータ量を制御するのに役立ちます。

  以下は、NetScalerでFACKを有効にするTCPプロファイルの注釈の例です。

  ```
  ingress.citrix.com/frontend_tcpprofile: '{"fack" : "enabled"}'
  ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

`ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"fack" : "enabled"}}'`

- `Window Scaling (WS)`: TCPウィンドウのスケーリングにより、TCP受信ウィンドウサイズを65535バイトを超えて増やすことができます。これにより、TCPパフォーマンス全体、特に高帯域幅で遅延の大きいネットワークでのパフォーマンスが向上します。遅延を減らし、TCP経由の応答時間を改善するのに役立ちます。

  以下は、NetScalerでWSを有効にするTCPプロファイルの注釈の例です。

  `ingress.citrix.com/frontend_tcpprofile: '{"ws" : "enabled", "wsval" : "9"}'`

  `ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"ws" : "enabled", "wsval" : "9"}}'`

  ここで、`wsval` は新しいウィンドウサイズを計算するために使用される係数です。この引数は、ウィンドウのスケーリングが有効になっている場合にのみ必須です。設定できる最小値は0、最大値は14です。デフォルトでは、値は4に設定されています。

- `Maximum Segment Size (MSS)`: 単一のTCPセグメントのMSS。この値は、中間ルーターとエンドクライアントのMTU設定に依存します。1460の値は、MTUが1500に対応します。

  NetScalerでMSSを有効にするTCPプロファイルの注釈の例を次に示します。

  `ingress.citrix.com/frontend_tcpprofile: '{"mss" : "1460", "maxpktpermss" : "512"}'`

  `ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"mss" : "1460", "maxpktpermss" : "512"}}'`

  ここで、:

  - `mss` は、TCP接続に使用するMSSです。最小値: 0; 最大値: 9176。
  - `maxpktpermss` は、最大セグメントサイズ（MSS）ごとに許可されるTCPパケットの最大数です。最小値: 0; 最大値: 1460。

- `Keep-Alive (KA)`: ピアがまだ稼働しているかを確認するために、定期的なTCPキープアライブ（KA）プローブを送信します。

  NetScalerでTCPキープアライブ（KA）を有効にするTCPプロファイルの注釈の例を次に示します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
ingress.citrix.com/frontend_tcpprofile: '{"ka" : "enabled",
"kaprobeupdatelastactivity" : "enabled", "kaconnidletime": "900",
"kamaxprobes" : "3", "kaprobeinterval" : "75"}'
```

```
ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"ka" : "enabled",
"kaprobeupdatelastactivity" : "enabled", "kaconnidletime": "900",
"kamaxprobes" : "3", "kaprobeinterval" : "75"}}'
```

ここで、:

- `ka` は、ピアがまだ稼働しているかを確認するために、定期的なTCPキープアライブ（KA）プローブの送信を有効にするために使用されます。可能な値: ENABLED, DISABLED。デフォルト値: DISABLED。
- `kaprobeupdatelastactivity` は、キープアライブ（KA）プローブを受信した後、接続の最終アクティビティを更新します。可能な値: ENABLED, DISABLED。デフォルト値: ENABLED。
- `kaconnidletime` は、キープアライブ（KA）プローブを送信する前に、接続がアイドル状態である期間（秒単位）です。設定できる最小値は1、最大値は4095です。
- `kaprobeinterval` は、ピアが応答しない場合に、次のキープアライブ（KA）プローブまでの時間間隔（秒単位）です。設定できる最小値は1、最大値は4095です。

- `bufferSize:` TCPバッファサイズをバイト単位で指定します。設定できる最小値は8190、最大値は20971520です。デフォルト値は8190です。

  以下は、TCPバッファサイズを指定するためのTCPプロファイルの注釈の例です。

  ```
  ingress.citrix.com/frontend_tcpprofile: '{"bufferSize" : "8190"}'
  ```

  ```
  ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"bufferSize" :
  "8190"}}'
  ```

- `Multipath TCP (MPTCP):` MPTCPを有効にし、オプションのMPTCP設定を行います。以下は、MPTCPを有効にし、オプションのMPTCP設定を行うためのTCPプロファイルの注釈の例です。

  ```
  ingress.citrix.com/frontend_tcpprofile: '{"mptcp" : "enabled",
  "mptcpdropdataonpreestsf" : "enabled", "mptcpfastopen": "enabled",
  "mptcpsessiontimeout" : "7200"}'
  ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

`ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"mptcp" : "enabled", "mptcpdropdataonpreestsf" : "enabled", "mptcpfastopen": "enabled", "mptcpsessiontimeout" : "7200"}}'`

ここで、:

- `mptcpdropdataonpreestsf` は、確立済みのサブフローでデータをサイレントに破棄するために使用されます。有効にすると、確立済みのサブフローでデータが受信されたときに、接続を破棄する代わりにDSSデータパケットがサイレントに破棄されます。可能な値：ENABLED、DISABLED。デフォルト値：DISABLED。
- `mptcpfastopen` を有効にすると、SYNハンドシェイクの3番目のACKを受信する前にDSSデータパケットが受け入れられます。可能な値：ENABLED、DISABLED。デフォルト値：DISABLED。

- `flavor`: TCP輻輳制御アルゴリズムを設定します。可能な値：Default、BIC、CUBIC、Westwood、Nile。デフォルト値：Default。以下は、TCP輻輳制御アルゴリズムを設定するTCPプロファイルの注釈の例です。

  `ingress.citrix.com/frontend_tcpprofile: '{"flavor" : "westwood"}'`

  `ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"flavor" : "westwood"}}'`

- `Dynamic receive buffering`: 動的受信バッファリングを有効または無効にします。有効にすると、メモリとネットワークの状態に基づいて受信バッファを動的に調整できます。可能な値：ENABLED、DISABLED。デフォルト値：DISABLED。

  注:

  動的な調整を行うには、バッファサイズ引数を設定する必要があります。

  `ingress.citrix.com/frontend_tcpprofile: '{"dynamicReceiveBuffering" : "enabled"}'`

  `ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"dynamicReceiveBuffering" : "enabled"}}'`

スプーフィング攻撃からTCPを防御する

NetScalerでTCPプロファイルの`rstWindowAttenuate`パラメータを使用して、スプーフィング攻撃からTCPを防御できます。デフォルトでは、`rstWindowAttenuate`パラメータは無効になって

© 1997–2026 Citrix Systems, Inc. All rights reserved.

います。このパラメータは、NetScalerをスプーフィングから保護するために有効にされます。`rstWindowAttenuate`を有効にすると、無効なシーケンス番号に対して修正確認応答（ACK）で応答します。可能な値：Enabled、Disabled。また、`spoofSynDrop`パラメータを使用して、スプーフィングから保護するために無効なSYNパケットのドロップを有効または無効にできます。無効にすると、SYNパケットを受信したときに確立された接続がリセットされます。このパラメータのデフォルト値はENABLEDです。

以下は、NetScalerで`rstWindowAttenuate`を有効にするTCPプロファイルのサンプルアノテーションです。

`ingress.citrix.com/frontend_tcpprofile: '{"rstwindowattenuate" : "enabled", "spoofsyndrop" : "enabled"}'`

`ingress.citrix.com/backend_tcpprofile: '{"citrix-svc" : {"rstwindowattenuate" : "enabled", "spoofsyndrop" : "enabled"}}'`

Ingressアノテーションを使用したTCPプロファイルの適用例

この例では、TCPプロファイルを適用する方法を示します。

1. TCPプロファイルを使用してフロントエンドIngressリソースを展開します。このIngressリソースでは、バックエンドとTLSは定義されていません。

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: frontend-ingress

  annotations:

    ingress.citrix.com/insecure-termination: "allow"

    ingress.citrix.com/frontend-ip: "10.221.36.190" # If you are using IPAM then use, ingress.citrix.com/ipam-range: "<IPAM Range Name>"

    ingress.citrix.com/frontend-tcpprofile: '{"ws" : "enabled", "sack" : "enabled"}'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:

  tls:

  - hosts:

  rules:

  - host:

EOF
```

2. 同じフロントエンドIPアドレスを持つセカンダリIngressリソースを展開します。バックエンドとTLSが定義されており、これによりロードバランシングリソース定義が作成されます。

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: backend-ingress

  annotations:

    ingress.citrix.com/insecure-termination: "allow"

    ingress.citrix.com/frontend-ip: "10.221.36.190" # If you are using
IPAM then use, ingresss.citrix.com/ipam-range: "<IPAM Range Name>"

spec:

  tls:

  - secretName: <hotdrink-secret>

  rules:

  - host:  hotdrink.beverages.com
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: frontend-hotdrinks
            port:
              number: 80
EOF
```

3. Ingressリソースが展開されると、対応するエンティティ、プロファイルがNetScaler上に作成されます。 NetScalerで次のコマンドを実行します: `show cs vserver k8s-10.221.36.190_443_ssl`。

```
       # show cs vserver k8s-10.221.36.190_443_ssl


         k8s-10.221.36.190_443_ssl (10.221.36.190:443) - SSL Type:
CONTENT
         State: UP
         Last state change was at Wed Apr  3 04:21:38 2024
         Time since last state change: 0 days, 00:00:57.420
         Client Idle Timeout: 180 sec
         Down state flush: ENABLED
         Disable Primary Vserver On Down : DISABLED
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          Comment:
uid=XMX2KPYG2GUJIHGTLVCPA7QVXDUBDRMJFTAWNCPAA2TVXB33EL5A====

          TCP profile name: k8s-10.221.36.190_443_ssl

          Appflow logging: ENABLED

          State Update: DISABLED

          Default: Content Precedence: RULE

          Vserver IP and Port insertion: OFF

          L2Conn: OFF Case Sensitivity: ON

          Authentication: OFF

          401 Based Authentication: OFF

          Push: DISABLED Push VServer:

          Push Label Rule: none

          HTTP Redirect Port: 0 Dtls : OFF

          Persistence: NONE

          Listen Policy: NONE

          IcmpResponse: PASSIVE

          RHIstate:  PASSIVE

          Traffic Domain: 0



          1) Content-Switching Policy: k8s-
backend_80_csp_2k75kfjrr6ptgzwtncozwxdjqrpbvicz Rule:
HTTP.REQ.HOSTNAME.SERVER.EQ("hotdrink.beverages.com") &&
HTTP.REQ.URL.PATH.SET_TEXT_MODE(IGNORECASE).STARTSWITH("/") Priority:
200000008 Hits: 0
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
Done
```

注記：

NetScalerでサポートされているさまざまなTCPパラメータの網羅的なリストについては、サポートされているTCPパラメータを参照してください。JSON形式で渡すキーと値は、NetScaler NITRO形式と一致している必要があります。NetScaler NITRO APIの詳細については、NetScaler 14.1 REST API - TCPプロファイル用NITROドキュメントを参照してください。

HTTPユースケース

このトピックでは、`LoadBalancer`タイプのサービスおよびingressに対してスマートアノテーションを使用してNetScalerで構成できる、さまざまなHTTPユースケースについて説明します。

NetScalerのHTTP構成は、`HTTP profile`と呼ばれるエンティティで指定できます。これはHTTP設定のコレクションです。その後、HTTPプロファイルを、これらのHTTP構成を使用できるサービスまたは仮想サーバーに関連付けることができます。

LoadBalancerタイプのサービスに対するHTTPプロファイルのサポート

NetScaler Ingress Controllerは、`LoadBalancer`タイプのサービスに対するHTTPプロファイルに以下のサービスアノテーションを提供します。これらのアノテーションを使用して、NetScalerのHTTP設定を定義できます。

サービスアノテーション	説明
`service.citrix.com/frontend-httpprofile`	このアノテーションを使用して、フロントエンドHTTPプロファイル（クライアントプレーン）を作成します。
`service.citrix.com/backend-httprofile`	このアノテーションを使用して、バックエンドHTTPプロファイル（サーバープレーン）を作成します。

ユーザー定義HTTPプロファイル

HTTPのサービスアノテーションを使用すると、コンテンツスイッチング仮想サーバーまたはサービスグループと同じ名前のカスタムプロファイルを作成し、これらのプロファイルを対応するコ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ンテンツスイッチング仮想サーバー (`frontend-httpprofile`) またはサービスグループ (`backend-httpprofile`) にバインドできます。

サービスアノテーション	サンプル
`service.citrix.com/frontend-httpprofile`	`service.citrix.com/frontend-httpprofile: '{"http2direct" : "enabled", "altsvc" : "enabled"}'`
`service.citrix.com/backend-httpprofile`	`service.citrix.com/backend-httpprofile: '{"http2direct" : "enabled", "altsvc" : "enabled"}'`

組み込みHTTPプロファイル

組み込みHTTPプロファイルは、プロファイルを作成しませんが、アノテーションで指定されたプロファイル名を、対応する仮想サーバー (frontend-httpprofile) とサービスグループ (backend-httpprofile) にバインドします。

以下は、組み込みHTTPプロファイルの例です。

`service.citrix.com/frontend-httpprofile: "http_preconf_profile"`

`service.citrix.com/backend-httpprofile: '{"apache" : "http_preconf_profile"}'`

例: HTTPプロファイル構成を持つLoadBalancerタイプのサービス

この例では、サンプルアプリケーション `tea-beverage` のHTTPプロファイルが構成されています。このアプリケーションは、以下のYAMLファイルを使用して、LoadBalancerタイプのサービスとしてデプロイおよび公開されます。

注:

LoadBalancerタイプのサービスの公開については、`LoadBalancer` タイプのサービスを参照してください。

以下のコマンドを使用して、サンプルアプリケーション (tea-beverage.yaml) をデプロイします。

```
kubectl apply -f - <<EOF

apiVersion: apps/v1
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kind: Deployment
metadata:
  name: tea-beverage
  labels:
    name: tea-beverage
spec:
  selector:
    matchLabels:
      name: tea-beverage
  replicas: 2
  template:
    metadata:
      labels:
        name: tea-beverage
    spec:
      containers:
      - name: tea-beverage
        image: quay.io/sample-apps/hotdrinks:latest
        ports:
        - name: tea-80
          containerPort: 80
        - name: tea-443
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          containerPort: 443

        imagePullPolicy: Always

---

apiVersion: v1

kind: Service

metadata:

  name: tea-beverage

  annotations:

    service.citrix.com/service-type-0: 'HTTP'

    service.citrix.com/frontend-ip: 10.105.158.194

    service.citrix.com/frontend-httpprofile: '{"http2direct" : "enabled",
"altsvc" : "enabled"}'

    service.citrix.com/backend-httpprofile: '{"http2direct" : "enabled",
"altsvc" : "enabled"}'

spec:

  type: LoadBalancer

  ports:

  - name: tea-80

    port: 80

    targetPort: 80

  selector:

    name: tea-beverage
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
EOF
```

アプリケーションがデプロイされると、対応するエンティティとプロファイルがNetScaler上に作成されます。NetScalerで以下のコマンドを実行して、これらを確認してください: `show cs vserver k8s-tea-beverage_80_default_svc` および `show servicegroup k8s-tea-beverage_80_sgp_f4lezsannvu7tk2ftpjbhi4hza2tvdnk`。

```
  # show cs vserver k8s-tea-beverage_80_default_svc



    k8s-tea-beverage_80_default_svc (10.105.158.194:80) - HTTP Type:
CONTENT

    State: UP

    Last state change was at Wed May  8 06:44:11 2024

    Time since last state change: 0 days, 00:04:24.740

    Client Idle Timeout: 180 sec

    Down state flush: ENABLED

    Disable Primary Vserver On Down : DISABLED

    Comment: uid=2TZGFGG3JKYS2D7KWJIPS7ODTL5D4S7IMU5K6GETMXHXRDE2WIAQ====

    HTTP profile name: k8s-tea-beverage_80_default_svc

    Appflow logging: ENABLED

    Port Rewrite : DISABLED

    State Update: DISABLED

    Default: k8s-tea-beverage_80_lbv_f4lezsannvu7tk2ftpjbhi4hza2tvdnk
Content Precedence: RULE

    Vserver IP and Port insertion: OFF
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    L2Conn: OFF Case Sensitivity: ON

    Authentication: OFF

    401 Based Authentication: OFF

    Push: DISABLED Push VServer:

    Push Label Rule: none

    HTTP Redirect Port: 0 Dtls : OFF

    Persistence: NONE

    Listen Policy: NONE

    IcmpResponse: PASSIVE

    RHIstate:  PASSIVE

    Traffic Domain: 0



  1) Default Target LB: k8s-tea-
beverage_80_lbv_f4lezsannvu7tk2ftpjbhi4hza2tvdnk Hits: 0

  Done
```

```
  # show servicegroup k8s-tea-
beverage_80_sgp_f4lezsannvu7tk2ftpjbhi4hza2tvdnk

      k8s-tea-beverage_80_sgp_f4lezsannvu7tk2ftpjbhi4hza2tvdnk - HTTP

      State: ENABLED  Effective State: UP  Monitor Threshold : 0

      Max Conn: 0  Max Req: 0  Max Bandwidth: 0 kbits

      Use Source IP: NO

      Client Keepalive(CKA): NO
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      Monitoring Owner: 0

      TCP Buffering(TCPB): NO

      HTTP Compression(CMP): NO

      Idle timeout: Client: 180 sec  Server: 360 sec

      Client IP: DISABLED

      Cacheable: NO

      SC: ???

      SP: OFF

      Down state flush: ENABLED

      Monitor Connection Close : NONE

      Appflow logging: ENABLED

      HTTP profile name: k8s-tea-
beverage_80_sgp_f4lezsannvu7tk2ftpjbhi4hza2tvdnk

      ContentInspection profile name: ???

      Process Local: DISABLED

      Traffic Domain: 0

      Comment: "lbsvc:tea-beverage,svcport:80,ns:default"




      1)  10.146.107.146:30092  State: UP  Server Name: 10.146.107.146
 Server ID: None  Weight: 1  Order: Default

        Last state change was at Wed May  8 06:44:25 2024

        Time since last state change: 0 days, 00:04:51.610
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        Monitor Name: tcp-default  State: UP  Passive: 0

        Probes: 59 Failed [Total: 0 Current: 0]

        Last response: Success - TCP syn+ack received.

        Response Time: 0.000 millisec



     2)  10.146.107.147:30092  State: UP  Server Name: 10.146.107.147
 Server ID: None  Weight: 1  Order: Default

        Last state change was at Wed May  8 06:44:45 2024

        Time since last state change: 0 days, 00:04:31.690



        Monitor Name: tcp-default   State: UP Passive: 0

        Probes: 56 Failed [Total: 2 Current: 0]

        Last response: Success - TCP syn+ack received.

        Response Time: 0.000 millisec

Done
```

注記:

HTTPプロファイルは、シングルポートサービスでサポートされています。

Ingressアノテーションを使用してHTTPプロファイルを構成する

次の表に、HTTPユースケースとサンプルアノテーションを示します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ユースケース	サンプルアノテーション	

HTTP/2の構成	`ingress.citrix.com/frontend-httpprofile: '{"http2" : "enabled"}'`
	`ingress.citrix.com/frontend-httpprofile: '{"http2direct" : "enabled"}'`
	`ingress.citrix.com/frontend-httpprofile: '{"http2direct" : "enabled", "altsvc" : "enabled"}'`	

	`ingress.citrix.com/backend-httpprofile: '{"apache" : {"http2" : "enabled"}}'`
	`ingress.citrix.com/backend-httpprofile: '{"apache" : {"http2direct" : "enabled"}}'`
	`ingress.citrix.com/backend-httpprofile: '{"apache" : {"http2direct" : "enabled", "altsvc" : "enabled"}}'`	

HTTPセッションタイムアウトの処理	`ingress.citrix.com/frontend-httpprofile: '{"reqtimeout" : "10", "reqtimeoutaction" : "drop"}'` `ingress.citrix.com/frontend-httpprofile: '{"reqtimeout" : "10", "adpttimeout" : "enabled"}'`	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ユースケース	サンプルアノテーション	
	`ingress.citrix.com/frontend-httpprofile: '{"reusepooltimeout" : "20000"}'`	
	`ingress.citrix.com/backend-httpprofile: '{"apache" : {"reqtimeout" : "10", "reqtimeoutaction" : "drop"}}'` `ingress.citrix.com/backend-httpprofile: '{"apache" : {"reqtimeout" : "10", "adpttimeout" : "enabled"}}'` `ingress.citrix.com/backend-httpprofile: '{"apache" : {"reusepooltimeout" : "20000"}}'`	

注:

上記のIngressアノテーションは、以前に説明した形式のサービスアノテーションでも使用できます。

HTTP/2の構成

NetScalerは、クライアント側とサーバー側の両方でHTTP/2を構成します。詳細については、NetScalerのHTTP/2サポートを参照してください。HTTP負荷分散構成の場合、NetScalerは以下のいずれかの方法を使用して、HTTP/2を使用してクライアント/サーバーとの通信を開始します。

NetScaler は、HTTP/2 メソッド用の構成可能なオプションを HTTP プロファイルで提供します。これらの HTTP/2 オプションは、HTTPS または HTTP 負荷分散設定のクライアント側とサーバー側に適用できます。NetScaler Ingress Controller は、NetScaler 上で HTTP プロファイルを構成するためのアノテーションを提供します。これらのアノテーションを使用して、NetScaler 上のさま

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ざまな HTTP 負荷分散構成を、HTTP/2 を使用してクライアント/サーバーと通信するように構成します。

注:

- 以下のメソッドの説明では、クライアントとサーバーはHTTP/2接続の一般的な用語です。たとえば、HTTP/2を使用するNetScalerの負荷分散設定では、NetScalerはクライアント側ではサーバーとして機能し、サーバー側ではクライアントとして機能します。
- NetScalerでHTTP/2サービス側のグローバルパラメータ(`HTTP2Serverside`)が有効になっていることを確認してください。

HTTP/2アップグレード

この方法では、クライアントはHTTP/1.1リクエストをサーバーに送信します。リクエストにはアップグレードヘッダーが含まれており、接続をHTTP/2にアップグレードするようサーバーに要求します。サーバーがHTTP/2をサポートしている場合、サーバーはアップグレードリクエストを受け入れ、その応答で通知します。クライアントはアップグレード確認応答を受信した後、クライアントとサーバーはHTTP/2を使用して通信を開始します。

HTTPプロファイルのアノテーションを使用して、NetScalerでHTTP/2アップグレードメソッドを構成できます。次のサンプルアノテーションは、NetScalerでHTTP/2アップグレードメソッドを構成します。

```
ingress.citrix.com/frontend-httpprofile: '{"http2" : "enabled"}'
```

```
ingress.citrix.com/backend-httpprofile: '{"apache" : {"http2" : "enabled"}}'
```

直接HTTP/2

この方法では、クライアントはHTTP/2アップグレードメソッドを使用する代わりに、HTTP/2でサーバーと直接通信を開始します。サーバーがHTTP/2をサポートしていないか、HTTP/2リクエストを直接受け入れるように構成されていない場合、クライアントからのHTTP/2パケットを破棄します。この方法は、クライアントデバイスの管理者がサーバーがHTTP/2をサポートしていることをすでに知っている場合に役立ちます。

HTTPプロファイルの注釈を使用すると、NetScalerで直接HTTP/2メソッドを設定できます。NetScalerで直接HTTP/2メソッドを設定するためのHTTPプロファイルの注釈の例を次に示します。

```
ingress.citrix.com/frontend-httpprofile: '{"http2direct" : "enabled"}'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
ingress.citrix.com/backend-httpprofile: '{"apache" : {"http2direct" :
"enabled"}}'
```

代替サービス (ALT-SVC) を使用した直接HTTP/2

この方法では、サーバーはHTTP/1.1応答に代替サービス (ALT-SVC) フィールドを含めることで、HTTP/2をサポートしていることをクライアントに通知します。クライアントがALT-SVCフィールドを理解するように設定されている場合、クライアントは応答を受信した後、クライアントとサーバーはHTTP/2を使用して直接通信を開始します。

NetScalerで代替サービス (ALT-SVC) メソッドを使用して直接HTTP/2を設定するためのHTTPプロファイルの注釈の例を次に示します。

```
ingress.citrix.com/frontend-httpprofile: '{"http2direct" : "enabled", "altsvc" :
"enabled"}'
```

```
ingress.citrix.com/backend-httpprofile: '{"apache" : {"http2direct" : "enabled",
"altsvc" : "enabled"}}'
```

HTTPセッションタイムアウトの処理

さまざまな種類のHTTPリクエストを処理し、クライアントが開始する接続を制限したいSlowloris DDOS攻撃などの攻撃を軽減するために、さまざまなタイムアウトを設定できます。NetScalerでは、このようなシナリオに対して次のタイムアウトを設定できます。

- reqTimeout と reqTimeoutAction
- adptTimeout
- reusePoolTimeout

reqTimeout と reqTimeoutAction

NetScalerでは、HTTPプロファイルの`reqTimeout`および`reqTimeoutAction`パラメータを使用して、HTTPリクエストのタイムアウト値とリクエストのタイムアウトアクションを設定できます。`reqTimeout`値は秒単位で設定され、HTTPリクエストは`reqTimeout`パラメータで指定された時間内に完了する必要があります。HTTPリクエストが定義された時間内に完了しない場合、`reqTimeoutAction`で指定されたリクエストタイムアウトアクションが実行されます。設定できる最小タイムアウト値は0、最大は86400です。デフォルトでは、タイムアウト値は0に設定されています。

`reqTimeoutAction`パラメータを使用して、HTTPリクエストのタイムアウト値 (`reqTimeout`) が経過した場合に実行するアクションの種類を指定できます。次のアクションを指定できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- リセット
- ドロップ

HTTPプロファイルの注釈を使用して、HTTPリクエストのタイムアウトとHTTPリクエストのタイムアウトアクションを設定できます。NetScalerでHTTPリクエストのタイムアウトとHTTPリクエストのタイムアウトアクションを設定するために、HTTPプロファイルの次のサンプル注釈が使用されます。

```
ingress.citrix.com/frontend-httpprofile: '{"reqtimeout" : "10",
"reqtimeoutaction" : "drop"}'
```

```
ingress.citrix.com/backend-httpprofile: '{"apache" : {"reqtimeout" : "10",
"reqtimeoutaction" : "drop"}}'
```

adptTimeout

要求されたセッションに設定されたタイムアウト値を使用する代わりに、`adptTimeout`を有効にすることもできます。`adptTimeout`パラメータは、フロー条件に応じてリクエストのタイムアウトを調整します。有効にすると、リクエストのタイムアウトは内部的に増減され、フローに適用されます。デフォルトでは、このパラメータはDISABLEDに設定されています。

HTTPプロファイルの注釈を使用して、`adpttimeout`パラメータを次のように有効または無効にできます。

```
ingress.citrix.com/frontend-httpprofile: '{"reqtimeout" : "10", "adpttimeout" :
"enabled"}'
```

```
ingress.citrix.com/backend-httpprofile: '{"apache" : {"reqtimeout" : "10",
"adpttimeout" : "enabled"}}'
```

reusePoolTimeout

再利用プールからアイドル状態のサーバー接続をフラッシュするために、再利用プールタイムアウト値を設定できます。サーバーが設定された時間アイドル状態の場合、対応する接続はフラッシュされます。

設定できる最小タイムアウト値は0で、最大は31536000です。デフォルトでは、タイムアウト値は0に設定されています。

HTTPプロファイルの注釈を使用して、次のように必要なタイムアウト値を設定できます。

```
ingress.citrix.com/frontend-httpprofile: '{"reusepooltimeout" : "20000"}'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

`ingress.citrix.com/backend-httpprofile: '{"apache" : {"reusepooltimeout" : "20000"}}'`

Ingressアノテーションを使用したHTTPプロファイルの適用例

この例では、HTTPプロファイルを適用する方法を示します。

1. HTTPプロファイルを使用してフロントエンドのIngressリソースを展開します。このIngressリソースでは、バックエンドとTLSは定義されていません。

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: frontend-ingress

  annotations:

    ingress.citrix.com/insecure-termination: "allow"

    ingress.citrix.com/frontend-ip: "10.221.36.190"

    ingress.citrix.com/frontend-httpprofile: '{"reqtimeout" : "10",
"adpttimeout" : "enabled"}'

spec:

  tls:

  - hosts:

  rules:

  - host:

EOF
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

2. 同じフロントエンドIPアドレスを持つセカンダリIngressリソースを展開します。このIngressリソースでは、バックエンドとTLSが定義されており、ロードバランシングリソース定義が作成されます。

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: backend-ingress

  annotations:

    ingress.citrix.com/insecure-termination: "allow"

    ingress.citrix.com/frontend-ip: "10.221.36.190"

spec:

  tls:

  - secretName: <hotdrink-secret>

  rules:

  - host:  hotdrink.beverages.com

    http:

      paths:

      - path: /

        pathType: Prefix

        backend:

          service:

            name: frontend-hotdrinks
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
            port:
              number: 80
EOF
```

3. Ingressリソースが展開されると、対応するエンティティとプロファイルがNetScaler上に作成されます。 NetScalerで次のコマンドを実行して、これを確認します: `show cs vserver k8s-10.221.36.190_443_ssl`。

```
        # show cs vserver k8s-10.221.36.190_443_ssl

          k8s-10.221.36.190_443_ssl (10.221.36.190:443) - SSL Type:
CONTENT

          State: UP

          Last state change was at Wed Apr  3 04:50:56 2024

          Time since last state change: 0 days, 00:00:31.590

          Client Idle Timeout: 180 sec

          Down state flush: ENABLED

          Disable Primary Vserver On Down : DISABLED

          Comment:
uid=IIFGNN4JZ4LBQ7VCBTX6DN7TIBGGOU3OD2OBIC7MB73O2B2FNL2Q====

          **HTTP profile name: k8s-10.221.36.190_443_ssl**

          Appflow logging: ENABLED

          State Update: DISABLED

          Default: Content Precedence: RULE

          Vserver IP and Port insertion: OFF

          L2Conn: OFF Case Sensitivity: ON
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
          Authentication: OFF

          401 Based Authentication: OFF

          Push: DISABLED Push VServer:

          Push Label Rule: none

          HTTP Redirect Port: 0 Dtls : OFF

          Persistence: NONE

          Listen Policy: NONE

          IcmpResponse: PASSIVE

          RHIstate:  PASSIVE

          Traffic Domain: 0



          1) Content-Switching Policy: k8s-
backend_80_csp_2k75kfjrr6ptgzwtncozwxdjqrpbvicz Rule:
HTTP.REQ.HOSTNAME.SERVER.EQ("hotdrink.beverages.com") &&
HTTP.REQ.URL.PATH.SET_TEXT_MODE(IGNORECASE).STARTSWITH("/") Priority:
200000008 Hits: 0

          Done
```

注:

NetScalerでサポートされているさまざまなHTTPパラメータについては、NetScaler 14.1 REST APIs - NITRO Documentation for HTTP profilesを参照してください。JSON形式で渡すキーと値は、NITRO形式の値と一致する必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

リライトおよびレスポンダーポリシーによるHTTPコールアウト

HTTPコールアウトを使用すると、NetScalerはポリシー評価の一部として、外部サーバー（コールアウトエージェント）にHTTPまたはHTTPSリクエストを生成して送信できます。サーバー（コールアウトエージェント）から取得された情報は、高度なポリシー式によって分析され、適切なアクションが実行されます。HTTPコールアウトの詳細については、NetScalerドキュメントを参照してください。

NetScalerが提供するリライトおよびレスポンダーCRDを使用して、以下の式でHTTPコールアウトを開始できます。

- `sys.http_callout()`: この式は、httpcalloutエージェントの応答を評価する必要がある場合に、呼び出しをブロックするために使用されます。
- `sys.non_blocking_http_callout()`: この式は、非ブロッキング呼び出し（例：トラフィックミラーリング）に使用されます。

これらの式は、CRDで定義された`httpcallout_policy`名をパラメータとして受け入れます。名前は二重引用符で指定する必要があります。

例: `sys.http_callout("callout_name")`。 この式では、 `callout_name`はリライトおよびレスポンダーCRD YAMLファイルで定義されている適切な`httpcallout_policy`を参照します。

次の表は、リライトおよびレスポンダーCRDにおけるHTTPコールアウトリクエストの属性を説明しています。

パラメータ	説明
`name`	コールアウトの名前を指定します。最大32文字です。
`server_ip`	コールアウトが送信されるサーバー（コールアウトエージェント）のIPアドレスを指定します。
`server_port`	コールアウトが送信されるサーバー（コールアウトエージェント）のポートを指定します。
`http_method`	このコールアウトが送信するHTTPリクエストで使用されるメソッドを指定します。デフォルト値はGETです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	説明
`host_expr`	ホストヘッダーを設定するためのテキスト式を指定します。この式はリテラル値（例：192.101.10.11）であるか、値を導出する高度な式（例：http.req.header("Host")）であることができます。リテラル値はIPアドレスまたは完全修飾ドメイン名であることができます。完全なHTTPリクエスト式とは相互に排他的です。
`url_stem_expr`	URLステムを生成するための文字列式を指定します。文字列式には、リテラル文字列（例：「/mysite/index.html」）または値を導出する式（例：http.req.url）を含めることができます。
`headers`	HTTPリクエストに挿入する1つ以上のヘッダーを指定します。各ヘッダーは`name`と`exp`で構成され、`exp`は指定されたヘッダーの値を提供するために実行時に評価される式です。
`parameters`	HTTPリクエストURL（GETリクエストの場合）またはリクエスト本文（POSTリクエストの場合）に挿入する1つ以上のクエリパラメータを指定します。各パラメータは`name`と`expr`で表され、`expr`は指定されたパラメータ（name=value）の値を提供するために実行時に評価される式です。パラメータ値はURLエンコードされます。
`body_expr`	リクエストの本文を生成するための高度な文字列式。この式には、リテラル文字列または値を導出する式（例：client.ip.src）を含めることができます。
`full_req_expr`	NetScalerがコールアウトエージェントに送信する、式形式の正確なHTTPリクエストを指定します。リクエスト式は、コールアウトが使用される機能によって制約されます。たとえば、HTTP.RES式は、リクエスト時ポリシーバンク

© 1997–2026 Citrix Systems, Inc. All rights reserved.

パラメータ	説明
	やTCPコンテンツスイッチングポリシーバンクでは使用できません。
`scheme`	コールアウトサーバーのスキームタイプを指定します。例: HTTP、HTTPS
`return_type`	ターゲットコールアウトエージェントがコールアウトに応答して返すデータのタイプを指定します。利用可能な設定は次のとおりです: TEXT - 返された値をテキスト文字列として扱います。NUM - 返された値を数値として扱います。BOOL - 返された値をブール値として扱います。
`cache_for_secs`	コールアウト応答がキャッシュされる期間を秒単位で指定します。キャッシュされた応答は、`calloutContentGroup` という名前の統合キャッシュコンテンツグループに保存されます。期間が設定されていない場合、通常のキャッシュ設定を使用してキャッシュされない限り、コールアウト応答はキャッシュされません。このパラメーターは、これらの応答に適用される可能性のある通常のキャッシュ設定よりも優先されます。
`result_expr`	HTTPコールアウトエージェントから送信された応答からコールアウト結果を抽出する式を指定します。この式は応答ベースの式である必要があり、つまり `HTTP.RES` で始まる必要があります。この式内の操作は戻り値の型と一致する必要があります。たとえば、戻り値の型を `TEXT` に設定した場合、結果式はテキストベースの式である必要があります。戻り値の型が `NUM` の場合、結果式 (result_expr) は次の例のように数値である必要があります: `http.res.body(10000).length`
`comment`	このHTTPコールアウトに関する情報を保持するためのコメントを指定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

リライトおよびレスポンダーCRDを使用して、クライアントIPアドレスがブロックリストに登録されているかどうかを検証する

このセクションでは、リライトおよびレスポンダーCRDを使用してHTTPコールアウトを開始し、クライアントIPアドレスがブロックリストに登録されているかどうかを検証し、適切なアクションを実行する方法を示します。

次の図は、リクエストのワークフローを説明しており、図中の各番号はワークフローのステップを示しています:

1. クライアントリクエスト
2. クライアントがブロックリストに登録されているかを確認するためのHTTPコールアウトリクエスト（クライアントIPアドレスは `Cip` という名前のクエリパラメーターとして送信されます）
3. HTTPコールアウトサーバーからの応答
4. ステップ3の応答が安全なIPアドレスを示している場合（クライアントIPアドレスがコールアウトサーバー上のブロックリストに登録されたIPアドレスと一致しない場合）、リクエストはサービスに転送されます。
5. ステップ3の応答が不正なIPアドレスを示している場合（クライアントIPアドレスがコールアウトサーバー上のブロックリストに登録されたIPアドレスと一致する場合）、`Access denied` としてクライアントに応答します。

以下は、ブロックリストに登録されたIPアドレスを検証するためのサンプルYAMLファイル（`ip_validate_responder.yaml`）です。

注:

`ip_validate_responder` YAMLファイルをデプロイする前に、rewrite and responder CRD をデプロイする必要があります。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: validateip
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:

  responder-policies:

    - servicenames:

        - frontend

      responder-policy:

        respondwith:

          http-payload-string: '"HTTP/1.1 401 Access denied\r\n\r\n"'

        respond-criteria:
'sys.http_callout("blocklist_callout").CONTAINS("IP Matched")' #Callout
name needs to be given in double quotes to pick httpcallout_policy

        comment: 'Invalid access'



  httpcallout_policy:

    - name: blocklist_callout

      server_ip: "192.2.156.160"

      server_port: 80

      http_method: GET

      host_expr: '"192.2.156.160"'

      url_stem_expr: '"/validateIP.pl"'

      headers:

      - name: X-Request

        expr: '"Callout Request"'

      parameters:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      - name: Cip

        expr: 'CLIENT.IP.SRC'

      return_type: TEXT

      result_expr: 'HTTP.RES.BODY(100)'
```

rewrite and responder CRDを使用して、クライアントが要求した有効なパスでURLを更新する

このセクションでは、セキュリティ上の理由によりクライアントに公開されているパスが実際のパスと異なる場合に、rewrite and responder CRDを使用してHTTPコールアウトを開始する方法について説明します。

リクエストのワークフローを以下の図で説明します。図中の各番号はワークフローのステップを示しています。

1. クライアントリクエスト
2. 有効なパスを取得するためのHTTPコールアウトリクエスト（クライアントから要求されたパスは、pathという名前のクエリパラメータとしてコールアウトサーバーに送信されます）
3. HTTPコールアウトサーバーからの応答
4. URLリクエストは有効なパスで書き換えられ、サービスに転送されます（有効なパスはコールアウト応答の`newpath`タグの間に記載されています）。

以下はサンプルYAML (`path_rewrite`) ファイルです。

注記:

`path_rewrite` YAMLファイルをデプロイする前に、リライトおよびレスポンダーCRD をデプロイする必要があります。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  name: getvalidpath

spec:

  rewrite-policies:

    - servicenames:

        - frontend

      rewrite-policy:

        operation: replace

        target: http.req.url

        modify-expression: 'sys.http_callout("mapping_callout")' #Callout
name needs to be given in double quotes to pick httpcallout_policy

        comment: 'Get the valid path'

        direction: REQUEST

        rewrite-criteria: 'TRUE'



  httpcallout_policy:

    - name: mapping_callout

      server_ip: "192.2.156.160"

      server_port: 80

      http_method: GET

      host_expr: '"192.2.156.160"'

      url_stem_expr: '"/getPath.pl"'

      headers:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      - name: X-Request

        expr: '"Callout Request"'

      parameters:

      - name: path

        expr: 'http.req.url'

      return_type: TEXT

      result_expr:
'"HTTP.RES.BODY(500).AFTER_STR(\"<newpath>\").BEFORE_STR(\"</newpath>\")"'
```

Ingress NetScaler でセッションアフィニティまたはパーシステンスを設定する

Ingress NetScaler のセッションアフィニティまたはパーシステンス設定により、グループ内のどの仮想サーバーがクライアント要求を受信しても、クライアント要求を同じ選択されたサーバーに転送できます。パーシステンスの構成時間が期限切れになると、グループ内の任意の仮想サーバーが受信クライアント要求に対して選択されます。

パーシステンスが設定されている場合、サーバーが選択されるとロードバランシングメソッドを上書きします。これにより、その仮想サーバーによって表されるサーバー上の接続の状態が維持されます。NetScaler は、サーバーの初期選択には構成されたロードバランシングメソッドを使用しますが、同じクライアントからの後続のすべての要求は同じサーバーに転送します。

最も一般的に使用されるパーシステンスタイプは、Cookie に基づくパーシステンスです。

Cookie に基づくパーシステンスを設定する

Cookie に基づくパーシステンスを有効にすると、NetScaler は HTTP 応答の `Set-Cookie` ヘッダーフィールドに HTTP Cookie を追加します。この Cookie には、HTTP 要求を送信する必要があるサービスに関する情報が含まれています。クライアントは Cookie を保存し、後続のすべての要求にそれを含め、ADC はそれを使用してそれらの要求のサービスを選択します。

NetScaler は Cookie `<NSC_XXXX>= <ServiceIP> <ServicePort>` を挿入します。

ここで：

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- `<<NSC_XXXX>` は、仮想サーバー名から派生した仮想サーバー ID です。
- `<<ServiceIP>` は、サービスの IP アドレスの 16 進値です。
- `<<ServicePort>` は、サービスのポートの 16 進値です。

NetScaler は、Cookie を挿入するときに `ServiceIP` と `ServicePort` を暗号化し、Cookie を受信するときにそれらを復号化します。

例：`a.com=ffffffff02091f1045525d5f4f58455e445a4a423660;expires=Fri, 23-Aug-2019 07:01:45`。

NetScaler Ingress Controller が提供する次の Ingress アノテーションを使用して、Ingress NetScaler でパーシステンス設定を構成できます。

```
ingress.citrix.com/lbvserver: '{"apache":
{"persistenceType":"COOKIEINSERT", "timeout":"20",
"cookiename":"k8s_cookie"}}'
```

ここで：

- `timeout` は永続性の期間を指定します。セッションクッキーが `timeout` の値 0 で使用される場合、使用される HTTP クッキーのバージョンに関わらず、NetScaler によって有効期限は指定されません。セッションクッキーはウェブブラウザが閉じられると期限切れになります。
- `cookiename` は最大32文字のクッキー名を指定します。指定しない場合、クッキー名は内部的に生成されます。
- ここでの `persistenceType` は使用する永続性のタイプを指定し、`COOKIEINSERT` はクッキーベースの永続性に用いられます。クッキー以外にも、適切な引数やその他の必要なパラメータとともに他のオプションも使用できます。

可能な値は、SOURCEIP、SSLSESSION、DESTIP、SRCIPDESTIP などです。

送信元IPアドレス永続性

Ingress NetScaler で送信元 IP 永続性が構成されている場合、ロードバランシング仮想サーバーに永続性を設定でき、同じクライアントからの後続のリクエストに対してスティッキネスを作成します。

送信元IPアドレス永続性を構成するためのIngressアノテーションの例を次に示します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
ingress.citrix.com/lbvserver: '{"apache":{"persistenceType":"SOURCEIP",
"timeout":"10"}}'
```

SSLセッションID永続性

SSLセッションID永続性が構成されている場合、NetScalerアプライアンスは、SSLハンドシェイクプロセスの一部であるSSLセッションIDを使用して、最初の要求がサービスに転送される前に永続性セッションを作成します。ロードバランシング仮想サーバーは、同じSSLセッションIDを持つ後続の要求を同じサービスに転送します。このタイプの永続性は、SSLブリッジサービスに使用されます。

SSLセッションID永続性を構成するためのIngressアノテーションの例を次に示します。

```
ingress.citrix.com/lbvserver: '{"apache":
{"persistenceType":"SSLSESSION"}}'
```

宛先IPアドレスベースの永続性

このタイプの永続性では、Ingress NetScaler が新しいクライアントからリクエストを受信すると、仮想サーバーによって選択されたサービス (宛先 IP アドレス) の IP アドレスに基づいて永続性セッションを作成します。その後、同じ宛先 IP へのリクエストを同じサービスに転送します。このタイプの永続性は、リンクロードバランシングで使用されます。

宛先IPアドレスベースの永続性を構成するためのIngressアノテーションの例を次に示します。

```
ingress.citrix.com/lbvserver: '{"apache":{"persistenceType":"DESTIP"}}'
```

送信元および宛先IPアドレスベースの永続性

このタイプの永続性では、NetScalerアプライアンスがリクエストを受信すると、クライアントのIPアドレス（送信元IPアドレス）と仮想サーバーによって選択されたサービスのIPアドレス（宛先IPアドレス）の両方に基づいて永続性セッションを作成します。その後、同じ送信元IPから同じ宛先IPへのリクエストを同じサービスに転送します。

以下は、送信元および宛先IPアドレスベースの永続性を構成するためのIngressアノテーションのサンプルです。

```
ingress.citrix.com/lbvserver: '{"apache":
{"persistenceType":"SRCIPDESTIP"}}'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

IPアドレスの許可リスト登録または拒否リスト登録

`Allowlisting IP addresses` を使用すると、ユーザーがドメインにアクセスできる信頼されたIPアドレスまたはIPアドレス範囲のリストを作成できます。これは、信頼されたユーザーのみにアクセスを制限および制御するためによく使用されるセキュリティ機能です。

`Blocklisting IP addresses` は基本的なアクセス制御メカニズムです。これは、拒否リストに登録したIPアドレスを使用してドメインにアクセスするユーザーからのアクセスを拒否します。

NetScalerが提供するRewrite and Responder CRD を使用すると、データセット、パットセット、および文字列マップを使用して広範なリライトおよびレスポンダーポリシーを定義し、Ingress NetScalerの統計のための監査ログを有効にすることができます。

リライトまたはレスポンダーポリシーを使用すると、ユーザーがドメインにアクセスできるIPアドレス/CIDRを許可リストに登録または拒否リストに登録できます。

以下のセクションでは、リライトまたはレスポンダーポリシーを使用してIPアドレス/CIDRを許可リストに登録または拒否リストに登録するさまざまな方法について説明します。

IPアドレスの許可リスト登録

レスポンダーポリシーを使用すると、IPアドレスを許可リストに登録し、許可リストに登録されたIPアドレスとは異なるIPアドレスを使用しているクライアントからのリクエストをサイレントに破棄できます。

`allowlist-ip.yaml` という名前のファイルを作成し、次のリライトポリシー設定を記述します。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: allowlistip

spec:

  responder-policies:

    - servicenames:

        - frontend
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      responder-policy:

        drop:

        respond-criteria: '!
client.ip.src.TYPECAST_text_t.equals_any("allowlistip")'

        comment: 'Allowlist certain IP addresses'

  patset:

    - name: allowlistip

      values:

        - '10.xxx.170.xx'

        - '10.xxx.16.xx'
```

IPアドレスをリストとして提供することもできます。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: allowlistip

spec:

  responder-policies:

    - servicenames:

        - frontend

      responder-policy:

        drop:

        respond-criteria: '!
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
client.ip.src.TYPECAST_text_t.equals_any("allowlistip")'

        comment: 'Allowlist certain IP addresses'

  patset:

    - name: allowlistip

      values: [ '10.xxx.170.xx', '10.xxx.16.xx' ]
```

次に、次のコマンドを使用してYAMLファイル (`allowlist-ip.yaml`) をデプロイします。

```
kubectl create -f allowlist-ip.yaml
```

IPアドレスを許可リストに登録し、許可リストにないクライアントからのリクエストに403応答を送信する

レスポンダーポリシーを使用すると、IPアドレスのリストを許可リストに登録し、許可リストに登録されたIPアドレスとは異なるIPアドレスを使用しているクライアントからのリクエストに `HTTP/1.1 403 Forbidden` 応答を送信できます。

`allowlist-ip-403.yaml` という名前のファイルを作成し、次のリライトポリシー設定を記述します。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: allowlistip

spec:

  responder-policies:

    - servicenames:

        - frontend

      responder-policy:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        respondwith:

          http-payload-string: '"HTTP/1.1 403 Forbidden\r\n\r\n" +
"Client: " + CLIENT.IP.SRC + " is not authorized to access URL:" +
HTTP.REQ.URL.HTTP_URL_SAFE +"\n"'

        respond-criteria: '!
client.ip.src.TYPECAST_text_t.equals_any("allowlistip")'

        comment: 'Allowlist a list of IP addresses'

  patset:

    - name: allowlistip

      values: [ '10.xxx.170.xx',  '10.xxx.16.xx' ]
```

次に、次のコマンドを使用してYAMLファイル (`allowlist-ip-403.yaml`) をデプロイします。

```
kubectl create -f allowlist-ip-403.yaml
```

CIDRを許可リストに登録する

レスポンダーポリシーを使用してCIDRを許可リストに登録できます。以下は、CIDRを許可リストに登録するためのレスポンダーポリシー設定の例です。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: blocklistips1

spec:

  responder-policies:

    - servicenames:

        - frontend
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      responder-policy:

        respondwith:

          http-payload-string: '"HTTP/1.1 403 Forbidden\r\n\r\n" +
"Client: " + CLIENT.IP.SRC + " is not authorized to access URL:" +
HTTP.REQ.URL.HTTP_URL_SAFE +"\n"'

        respond-criteria: '!client.ip.src.IN_SUBNET(10.xxx.170.xx/24)'

        comment: 'Allowlist certain IPs'
```

IPアドレスをブロックリストに登録する

レスポンダーポリシーを使用すると、IPアドレスをブロックリストに登録し、ブロックリストに登録されたIPアドレスを使用するクライアントからのリクエストをサイレントに破棄できます。

`blocklist-ip.yaml`という名前のファイルを作成し、次のレスポンダーポリシー設定を記述します。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: blocklistips

spec:

  responder-policies:

    - servicenames:

        - frontend

      responder-policy:

        respondwith:

        drop:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        respond-criteria:
'client.ip.src.TYPECAST_text_t.equals_any("blocklistips")'

        comment: 'Blocklist certain IPS'



  patset:

    - name: blocklistips

      values:

        - '10.xxx.170.xx'

        - '10.xxx.16.xx'
```

次に、YAMLファイル (`blocklist-ip.yaml`) を次のコマンドを使用して展開します。

```
kubectl create -f blocklist-ip.yaml
```

CIDRをブロックリストに登録する

レスポンダーポリシーを使用してCIDRをブロックリストに登録できます。以下は、CIDRをブロックリストに登録するためのレスポンダーポリシー設定の例です。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: blocklistips1

spec:

  responder-policies:

    - servicenames:

        - frontend
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
      responder-policy:

        respondwith:

          http-payload-string: '"HTTP/1.1 403 Forbidden\r\n\r\n" +
"Client: " + CLIENT.IP.SRC + " is not authorized to access URL:" +
HTTP.REQ.URL.HTTP_URL_SAFE +"\n"'

        respond-criteria: 'client.ip.src.IN_SUBNET(10.xxx.170.xx/24)'

        comment: 'Blocklist certain IPs'
```

CIDRを許可リストに登録し、IPアドレスをブロックリストに登録する

レスポンダーポリシーを使用して、CIDRを許可リストに登録し、IPアドレスをブロックリストに登録できます。以下は、レスポンダーポリシー設定の例です。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: allowlistsub

spec:

  responder-policies:

    - servicenames:

        - frontend

      responder-policy:

        drop:

        respond-criteria:
'client.ip.src.TYPECAST_text_t.equals_any("blocklistips") || !
client.ip.src.IN_SUBNET(10.xxx.170.xx/24)'

        comment: 'Allowlist a subnet and blocklist few IP's'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  patset:
    - name: blocklistips
      values:
        - '10.xxx.170.xx'
```

CIDRをブロックリストに登録し、IPアドレスを許可リストに登録する

レスポンダーポリシーを使用して、CIDRをブロックリストに登録し、IPアドレスを許可リストに登録できます。以下は、レスポンダーポリシー設定の例です。

```
apiVersion: citrix.com/v1
kind: rewritepolicy
metadata:
  name: blocklistips1
spec:
  responder-policies:
    - servicenames:
        - frontend
      responder-policy:
        drop:
        respond-criteria: 'client.ip.src.IN_SUBNET(10.xxx.170.xx/24) && !
client.ip.src.TYPECAST_text_t.equals_any("allowlistips")'
        comment: 'Blocklist a subnet and allowlist few IP's'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
patset:
  - name: allowlistips
    values:
      - '10.xxx.170.xx'
      - '10.xxx.16.xx'
```

ExternalDNSとの相互運用性

Kubernetes環境では、`LoadBalancer`タイプのサービスを使用してデプロイメントを公開できます。また、NetScaler IPAM Controllerを使用してサービスにIPアドレスを割り当てることができます。NetScaler IPAM Controllerは、定義されたIPアドレスプールからサービスにIPアドレスを割り当てます。詳細については、IPAMコントローラーによって割り当てられたIPアドレスを持つLoadBalancerタイプのサービスを公開するを参照してください。

サービスはIPAMコントローラーによって割り当てられたIPアドレスを使用してアクセスできますが、サービスディスカバリのためには、DNSプロバイダーにIPアドレスを手動で登録する必要があります。サービスに割り当てられたIPアドレスが変更された場合、関連するDNSレコードを手動で更新する必要があり、プロセス全体が煩雑になります。そのような場合、ExternalDNSを使用して、DNSレコードを外部エントリポイントと同期させることができます。また、ExternalDNSを使用すると、DNSプロバイダーに依存しない方法で、Kubernetesリソースを介してDNSレコードを動的に制御できます。

ExternalDNS統合が機能するためには、`external-dns.alpha.kubernetes.io/hostname`アノテーションにホスト名が含まれている必要があります。

注:

ExternalDNSが機能するためには、サービス仕様にアノテーション`external-dns.alpha.kubernetes.io/hostname`を追加し、そのアノテーションを使用してサービスのホスト名を指定してください。

ExternalDNSと統合するには:

1. InfobloxプロバイダーでExternalDNSをインストールします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

この相互運用性ソリューションはInfobloxプロバイダーでテストされており、他のプロバイダーでも機能する可能性があります。

2. ExternalDNS構成でドメイン名を指定します。
3. `LoadBalancer`タイプのサービス仕様で、次のアノテーションを追加し、そのアノテーションを使用してサービスのホスト名を指定します。

```
external-dns.alpha.kubernetes.io/hostname
```

4. 次のコマンドを使用してサービスをデプロイします。

```
kubectl create -f &lt;service-name>.yml
```

NetScaler Ingress ControllerのためにVaultサーバーに保存されたNetScaler資格情報を使用する

ほとんどの組織では、ティア1のNetScaler Ingressデバイスと Kubernetesクラスターは別々のチームによって管理されています。通常、ネットワーク管理者がティア1のNetScaler Ingressデバイスを管理し、開発者がKubernetesクラスターを管理します。NetScaler Ingress Controllerは、NetScalerを設定するためにNetScalerのユーザー名やパスワードなどのNetScaler資格情報を必要とします。NetScaler Ingress Controllerの仕様の一部としてNetScaler資格情報を指定し、ADC資格情報をKubernetesシークレットとして保存できます。ただし、セキュリティリスクを最小限に抑えるために、NetScaler資格情報をVaultサーバーに保存し、NetScaler Ingress Controllerに資格情報を渡すこともできます。このトピックでは、NetScaler Ingress ControllerのためにVaultサーバーに保存されたNetScaler資格情報を使用する方法について説明します。

次の図は、Vaultサーバーに保存されているNetScaler資格情報をNetScaler Ingress Controllerで使用する手順を説明しています。

前提条件

Vaultサーバーをセットアップし、キーバリュー (KV) シークレットストアを有効にしていることを確認してください。詳細については、Vault documentationを参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Ingress ControllerのためにVaultサーバーからNetScaler資格情報を使用する

NetScaler Ingress ControllerのためにVaultサーバーからNetScaler資格情報を使用するには、次のタスクを実行します。

1. Kubernetes認証用のサービスアカウントを作成する。
2. Key Vaultシークレットを作成し、VaultサーバーでKubernetes認証をセットアップする。
3. Vault Auto-Auth機能を利用して、NetScaler Ingress ControllerのNetScaler資格情報を取得する。

Kubernetes認証用のサービスアカウントを作成する

以下の手順を使用して、Kubernetes認証用のサービスアカウントを作成します。

1. サービスアカウント`cic-k8s-role`を作成し、以下のコマンドを使用してKubernetes TokenReview APIにアクセスするために必要な権限をサービスアカウントに付与します。

```
$ kubectl apply -f cic-k8s-role-service-account.yml




serviceaccount/cic-k8s-role created

clusterrole.rbac.authorization.k8s.io/cic-k8s-role configured

clusterrolebinding.rbac.authorization.k8s.io/cic-k8s-role configured

clusterrolebinding.rbac.authorization.k8s.io/role-tokenreview-binding
configured
```

以下は、サンプルcic-k8s-role-service-account.ymlファイルの一部です。

```
apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRoleBinding

metadata:

name: role-tokenreview-binding
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
namespace: default

roleRef:

apiGroup: rbac.authorization.k8s.io

kind: ClusterRole

name: system:auth-delegator

subjects:

- kind: ServiceAccount

name: cic-k8s-role

namespace: default
```

2. `VAULT_SA_NAME`環境変数を、既に作成したサービスアカウントの名前に設定します。

```
export VAULT_SA_NAME=$(kubectl get sa cic-k8s-role -o
jsonpath="{.secrets[*]['name']}")
```

3. `SA_JWT_TOKEN`環境変数を、TokenReview APIへのアクセスに使用したサービスアカウントのJWTに設定します。

```
export SA_JWT_TOKEN=$(kubectl get secret $VAULT_SA_NAME -o
jsonpath="{.data.token}" | base64 --decode; echo)
```

4. Kubernetes APIと通信するために、Kubernetes CA署名付き証明書を取得します。

```
export SA_CA_CRT=$(kubectl get secret $VAULT_SA_NAME -o
jsonpath="{.data['ca\.crt']}" | base64 --decode; echo)
```

キーボールトシークレットを作成し、VaultサーバーでKubernetes認証を設定する

Vaultサーバーにログインし、次の手順を実行してKey Vaultシークレットを作成し、Kubernetes認証を設定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

1. サンプルのVaultポリシーファイルcitrix-adc-kv-ro.hclを確認し、Vaultで読み取り専用ポリシー `citrix-adc-kv-ro` を作成します。

```
$ tee citrix-adc-kv-ro.hcl <<EOF

# If working with K/V v1

path "secret/citrix-adc/*"

{

    capabilities = ["read", "list"]

}

# If working with K/V v2

path "secret/data/citrix-adc/*"

{

    capabilities = ["read", "list"]

}

EOF



#  Create a policy named citrix-adc-kv-ro

$  vault policy write citrix-adc-kv-ro citrix-adc-kv-ro.hcl
```

2. `secret/citrix-adc/` パスにNetScalerの資格情報を含むKVシークレットを作成します。

```
vault kv put secret/citrix-adc/credential   username='&lt;ADC
username>' \

password='&lt;ADC password>' \

ttl='30m'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

3. デフォルトパス(`auth/kubernetes`)でKubernetes認証を有効にします。

```
# $ vault auth enable kubernetes
```

4. Kubernetesクラスターとの通信方法を指定します。

```
$ vault write auth/kubernetes/config \

token_reviewer_jwt="$SA_JWT_TOKEN" \

kubernetes_host="https://&lt;K8S_CLUSTER_URL>:&lt;API_SERVER_PORT>" \

kubernetes_ca_cert="$SA_CA_CRT"
```

5. KubernetesサービスアカウントをVaultポリシーとデフォルトのトークンTTLにマッピングするロールを作成します。このロールは、デフォルトの名前空間の`cic-k8s-role`サービスアカウントを承認し、サービスアカウントを`citrix-adc-kv-ro`ポリシーにマッピングします。

```
$ vault write auth/kubernetes/role/cic-vault-example\

bound_service_account_names=cic-k8s-role \

bound_service_account_namespaces=default \

policies=citrix-adc-kv-ro \

ttl=24h
```

注:

Kubernetes認証バックエンドによる認証はロールベースです。トークンがログインに使用される前に、ロールの一部として設定する必要があります。

NetScaler Ingress ControllerでVaultエージェントの自動認証を活用する

Vaultの自動認証を活用するには、次の手順を実行します。

1. 提供されているVault Agent設定ファイル`vault-agent-config.hcl`を確認します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    exit_after_auth = true

    pid_file = "/home/vault/pidfile"



    auto_auth {

        method "kubernetes" {

            mount_path = "auth/kubernetes"

            config = {

                role = "cic-vault-example"

            }

        }



        sink "file" {

            config = {

                path = "/home/vault/.vault-token"

            }

        }

    }
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注記:

Vaultエージェント`Auto-Auth`は、Vaultサーバーの`auth/kubernetes`パスで有効になっているKubernetes認証方法を使用するように構成されています。Vaultエージェントは`cic-vault-example`ロールを使用して認証します。 シンクブロックは、トークンを書き込むディスク上の場所を指定します。Vaultエージェントがトークンを複数の場所に配置するようにしたい場合は、Vaultエージェント`Auto-Auth`シンクを複数回構成できます。この例では、シンクは`/home/vault/.vault-token`に設定されています。

2. Consulテンプレート consul-template-config.hcl ファイルを確認します。

```
    vault {

        renew_token = false

        vault_agent_token_file = "/home/vault/.vault-token"

        retry {

            backoff = "1s"

        }

    }



    template {

        destination = "/etc/citrix/.env"

        contents = <<EOH

        {{- with secret "secret/citrix-adc/credential" }}

        NS_USER={{ .Data.data.username }}

        NS_PASSWORD={{ .Data.data.password }}

        {{ end }}
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        EOH

    }
```

このテンプレートは、`secret/citrix-adc/credential`パスでシークレットを読み取り、ユーザー名とパスワードの値を設定します。 KVストアバージョン1を使用している場合は、次のテンプレートを使用します。

```
template {

    destination = "/etc/citrix/.env"

    contents = <<EOH

    {{- with secret "secret/citrix-adc/credential" }}

    NS_USER={{ .Data.username }}

    NS_PASSWORD={{ .Data.password }}

    {{ end }}

    EOH

        }
```

3. vault-agent-config.hclとconsul-template-config.hclからKubernetes ConfigMapを作成します。

```
kubectl create configmap example-vault-agent-config --from-file=./
vault-agent-config.hcl --form-file=./consul-template-config.hcl
```

4. VaultとConsulテンプレートをinitコンテナとして持つNetScaler Ingress Controllerポッドcitrix-k8s-ingress-controller-vault.yamlを作成します。VaultはKubernetes認証方法を使用してトークンをフェッチし、共有ボリューム上に`.env`ファイルを作成するConsulテンプレートに渡します。このトークンは、NetScaler Ingress Controllerがティア1 NetScalerとの認証に使用します。

```
kubectl apply citrix-k8s-ingress-controller-vault.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

`citrix-k8s-ingress-controller-vault.yaml` ファイルは次のとおりです。

```
apiVersion: v1

kind: Pod

metadata:

annotations:

name: cic-vault

namespace: default

spec:

containers:

- args:

    - --ingress-classes tier-1-vpx

    - --feature-node-watch true

    env:

    - name: NS_IP

    value: &lt;Tier 1 ADC IP-ADDRESS>

    - name: EULA

    value: "yes"

    image: in-docker-reg.eng.citrite.net/cpx-dev/kumar-cic:latest

    imagePullPolicy: Always

    name: cic-k8s-ingress-controller

    volumeMounts:

    - mountPath: /etc/citrix
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    name: shared-data
initContainers:
- args:
    - agent
    - -config=/etc/vault/vault-agent-config.hcl
    - -log-level=debug
    env:
    - name: VAULT_ADDR
    value: &lt;VAULT URL>
    image: vault
    imagePullPolicy: Always
    name: vault-agent-auth
    volumeMounts:
    - mountPath: /etc/vault
    name: config
    - mountPath: /home/vault
    name: vault-token
- args:
    - -config=/etc/consul-template/consul-template-config.hcl
    - -log-level=debug
    - -once
    env:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    - name: HOME

    value: /home/vault

    - name: VAULT_ADDR

    value: &lt;VAULT_URL>

    image: hashicorp/consul-template:alpine

    imagePullPolicy: Always

    name: consul-template

    volumeMounts:

    - mountPath: /home/vault

    name: vault-token

    - mountPath: /etc/consul-template

    name: config

    - mountPath: /etc/citrix

    name: shared-data

serviceAccountName: vault-auth

volumes:

- emptyDir:

    medium: Memory

    name: vault-token

- configMap:

    defaultMode: 420

    items:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    - key: vault-agent-config.hcl

        path: vault-agent-config.hcl

    - key: consul-template-config.hcl

name: example-vault-agent-config

    name: config

- emptyDir:

    medium: Memory

    name: shared-data
```

構成が成功すると、Vaultサーバーはトークンをフェッチし、Consulテンプレートコンテナに渡します。Consulテンプレートはトークンを使用してNetScalerの資格情報を読み取り、`/etc/citrix/.env`パスに環境変数として書き込みます。NetScaler Ingress Controllerは、これらの資格情報を使用してティア1 NetScalerと通信します。

Vaultサーバーからフェッチされた資格情報を使用して、NetScaler Ingress Controllerが正常に実行されていることを確認します。

NetScalerの資格情報を保存するためにKubernetesシークレットを使用する方法

ほとんどの組織では、Tier 1 NetScaler IngressデバイスとKubernetesクラスターは別々のチームによって管理されています。NetScaler Ingress Controllerは、NetScalerを構成するためにNetScalerのユーザー名とパスワードなどのNetScaler資格情報を必要とします。通常、NetScaler資格情報はNetScaler Ingress ControllerのPod仕様で環境変数として指定されます。しかし、別の安全なオプションとして、Kubernetesシークレットを使用してNetScaler資格情報を保存する方法があります。

このトピックでは、Kubernetesシークレットを使用してADC資格情報を保存する方法と、NetScaler Ingress Controllerのシークレットデータとして保存された資格情報を提供するさまざまな方法について説明します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Kubernetesシークレットを作成する

Kubernetesシークレットを作成するには、以下の手順を実行します。

1. `data`セクションにNetScalerのユーザー名とパスワードを含むKubernetesシークレットYAMLを定義する`adc-credential-secret.yaml`ファイルを作成します。

```
apiVersion: v1

kind: Secret

metadata:

  name: adc-credential

data:

  username: &lt;ADC user name>

  password: &lt;ADC password>
```

2. シークレットを作成するために、`adc-credential-secret.yaml`ファイルを適用します。

```
kubectl apply -f adc-credential-secret.yaml
```

   または、次のように`kubectl`コマンドの`--from-literal`オプションを使用してKubernetesシークレットを作成することもできます。

```
kubectl create secret generic adc-credentials --from-
literal=username=&lt;username> --from-literal=password=&lt;password>
```

Kubernetesシークレットを作成したら、NetScaler Ingress ControllerのPod仕様でシークレットデータを使用するために、以下のいずれかのオプションを使用できます。

- NetScaler Ingress ControllerのPod仕様でシークレットデータを環境変数として使用する
- シークレットボリュームマウントを使用してNetScaler Ingress Controllerに資格情報を渡す

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Ingress ControllerのPod仕様でシークレットデータを環境変数として使用する

Kubernetesシークレットからのシークレットデータを、NetScaler Ingress Controllerのデプロイメント仕様の環境変数の値として使用できます。

YAMLファイルのスニペットを以下に示します。

```
- name: "NS_USER"

  valueFrom:

    secretKeyRef:

      name: adc-credentials

      key: username

# Set user password for Nitro

- name: "NS_PASSWORD"

  valueFrom:

    secretKeyRef:

      name: adc-credentials

      key: password
```

以下は、シークレットオブジェクトから取得した環境変数の値を持つNetScaler Ingress Controllerデプロイメントの例です。

```


apiVersion: apps/v1

kind: Deployment

metadata:

  name: cic-k8s-ingress-controller
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:
  selector:
    matchLabels:
      app: cic-k8s-ingress-controller
  replicas: 1
  template:
    metadata:
      name: cic-k8s-ingress-controller
      labels:
        app: cic-k8s-ingress-controller
      annotations:
    spec:
      serviceAccountName: cic-k8s-role
      containers:
      - name: cic-k8s-ingress-controller
        image: <image location>
        env:
         # Set NetScaler NSIP/SNIP, SNIP in case of HA (mgmt has to be
enabled)
         - name: "NS_IP"
           value: "x.x.x.x"
         # Set username for Nitro
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        - name: "NS_USER"
          valueFrom:
           secretKeyRef:
            name: adc-credentials
            key: username
        # Set user password for Nitro
        - name: "NS_PASSWORD"
          valueFrom:
           secretKeyRef:
            name: adc-credentials
            key: password
        # Set log level
        - name: "EULA"
          value: "yes"
       imagePullPolicy: Always
```

NetScaler Ingress Controller に資格情報を渡すためにシークレットボリュームマウントを使用する

あるいは、NetScaler の資格情報のソースとしてシークレットオブジェクトを使用するボリュームマウントも利用できます。NetScaler Ingress Controller は、シークレットがパス `/etc/citrix` にマウントされていることを想定しており、ファイル `username` と `password` で資格情報を探します。

以下の展開例に示すように、シークレットオブジェクトからボリュームを作成し、`/etc/citrix` で volumeMounts を使用してそのボリュームをマウントできます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: cic-k8s-ingress-controller

spec:

  selector:

    matchLabels:

      app: cic-k8s-ingress-controller

  replicas: 1

  template:

    metadata:

      name: cic-k8s-ingress-controller

      labels:

        app: cic-k8s-ingress-controller

      annotations:

    spec:

      serviceAccountName: cic-k8s-role

      containers:

      - name: cic-k8s-ingress-controller

        image: <image location>

        env:

         # Set NetScaler NSIP/SNIP, SNIP in case of HA (mgmt has to be
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
enabled)

          - name: "NS_IP"

            value: "x.x.x.x"

          # Set log level

          - name: "EULA"

            value: "yes"

        volumeMounts:

        # name must match the volume name below

          - name: secret-volume

            mountPath: /etc/citrix

        imagePullPolicy: Always

      # The secret data is exposed to Containers in the Pod through a
Volume.

      volumes:

      - name: secret-volume

        secret:

          secretName: adc-credentials
```

Hashicorp Vault サーバーに保存されている NetScaler 資格情報を使用する

NetScaler Ingress Controller のために、Hashicorp Vault サーバーに保存されている NetScaler 資格情報を使用し、サイドカーコンテナを介して資格情報をプッシュすることもできます。

詳細については、「Vault サーバーに保存されている NetScaler 資格情報を使用する」を参照してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

UDPベースのアプリケーションへのイングレストラフィックをロードバランスする方法

Kubernetes環境では、イングレスオブジェクトは、Kubernetesクラスターの外部からKubernetesサービスへのアクセスを許可します。標準のKubernetesイングレスリソースは、すべてのトラフィックがHTTPベースであると想定しており、TCP、UDP、SSLなどの非HTTPベースのプロトコルには対応していません。したがって、DNS、FTP、LDAPなどの非HTTPアプリケーションは、標準のKubernetesイングレスを使用して公開することはできません。

NetScalerは、イングレスアノテーションを使用してUDPベースのイングレストラフィックをロードバランスするソリューションを提供します。イングレスリソース定義でこれらのアノテーションを指定すると、NetScaler Ingress ControllerはNetScalerを構成して、UDPベースのイングレストラフィックをロードバランスします。

Kubernetesイングレスリソース定義で次のアノテーションを使用して、UDPベースのイングレストラフィックをロードバランスできます。

- `ingress.citrix.com/insecure-service-type`: このアノテーションは、NetScalerのプロトコルとしてUDPまたはANYを使用してL4ロードバランシングを有効にします。
- `ingress.citrix.com/insecure-port`: このアノテーションはUDPトラフィックのポートを構成します。これは、非標準ポートでマイクロサービスアクセスが必要な場合に役立ちます。デフォルトでは、ポート80が構成されています。

アノテーションの詳細については、アノテーションを参照してください。

NetScalerで`LoadBalancer`タイプのサービスを作成する標準的なKubernetesソリューションを使用することもできます。NetScalerのサービスタイプLoadBalancerについて詳しく知ることができます。

例: UDPベースのイングレスのイングレス定義。

```
kubectl apply -f - <<EOF

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  annotations:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    ingress.citrix.com/frontend-ip: "192.168.1.1" # If you are using IPAM
then use ingress.citrix.com/ipam-range: "<IPAM Range Name>"

    ingress.citrix.com/insecure-port: "5084"

    ingress.citrix.com/insecure-service-type: "udp"

  name: udp-ingress

spec:

  defaultBackend:

    service:

      name: frontend

      port:

        name: udp-53  # Service port name defined in the service
defination

EOF
```

例: サービスポート名が`udp-53`として定義されているサービス定義:

```
kubectl apply -f - <<EOF

apiVersion: v1

kind: Service

metadata:

  name: bind

  labels:

    app: bind

spec:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  ports:
  - name: udp-53
    port: 53
    targetPort: 53
    protocol: UDP
  selector:
    name: bind
EOF
```

UDPベースのアプリケーションのパフォーマンスを監視および改善する

アプリケーション開発者は、NetScalerの豊富なモニター（UDP-ECVなど）を通じて、UDPベースのアプリケーションの健全性を綿密に監視できます。ECV（拡張コンテンツ検証）モニターは、アプリケーションが期待されるコンテンツを返すかどうかを確認するのに役立ちます。NetScaler Ingress Controllerは、バックエンドサービスの健全性を監視するために使用できる `ingress.citrix.com/monitor` アノテーションを提供します。NetScalerの組み込みモニターの詳細については、組み込みモニターを参照してください。

また、`Source IP` などの永続化メソッドを使用することで、アプリケーションのパフォーマンスを向上させることができます。これらのNetScaler機能は、Kubernetesのスマートアノテーションを通じて使用できます。

次のイングレスリソースの例では、スマートアノテーションを使用しています。

```
kubectl apply -f - <<EOF
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  annotations:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    ingress.citrix.com/frontend-ip: "192.168.1.1" # If you are using IPAM
then use ingress.citrix.com/ipam-range: "<IPAM Range Name>"

    ingress.citrix.com/insecure-port: "80"

    ingress.citrix.com/lbvserver: '{"mongodb-svc":
{"lbmethod":"SRCIPDESTIPHASH"}}'

    ingress.citrix.com/monitor: '{"mongodbsvc":{"type":"UDP-ECV"}}'

  name: mongodb

spec:

  rules:

  - host: mongodb.beverages.com

    http:

      paths:

      - backend:

          service:

            name: mongodb-svc

            port:

              number: 80

        path: /

        pathType: Prefix

EOF
```

NetScaler CPXサービスで非標準HTTPポートを公開する方法

NetScaler CPXサービスで、UDPトラフィックを他のポートで許可するために、80番および443番以外のポートを公開する必要がある場合があります。 このセクションでは、Kubernetesクラス

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ターにデプロイする際に、NetScaler CPXサービスで他の非標準HTTPポートを公開する方法について説明します。

Helmチャートの展開の場合

Helmチャートを使用してイングレスコントローラーとともにNetScaler CPXをデプロイする際に、非標準HTTPポートを公開するには、Helmチャートのインストールガイドを参照してください。

```
servicePorts:

  - port: 8080

    protocol: UDP

    name: udp0

  - port: 6798

    protocol: UDP

    name: udp1

  - port: 6379

    protocol: UDP

    name: udp3
```

OpenShift Operatorを使用する展開の場合

OpenShift Operatorを使用する展開の場合、NetScaler Operatorを使用してNetScaler CPXのサイドカーとしてNetScaler Ingress Controllerを展開するのステップ6で指定されているように、イングレスコントローラーとともにCPXを作成するためにYAML定義を編集し、`service`セクションの下にポートを次のように指定する必要があります。

```
service:

  annotation: {}

  spec:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    type: NodePort  # ClusterIP or LoadBalancer
    ports:
    - port: 6790
      targetPort: 6790
      protocol: UDP
      name: udp1
    - port: 6379
      targetPort: 6379
      protocol: UDP
      name: udp0
```

デュアルティア展開のセットアップ方法

デュアルティア展開では、NetScaler VPXまたはMPXがKubernetesクラスターの外部（Tier-1）に展開され、NetScaler CPXがKubernetesクラスターの内部（Tier-2）に展開されます。

Tier-1のNetScaler MPXまたはVPXデバイスは、クライアントからTier-2のNetScaler CPXへのトラフィック（南北トラフィック）をプロキシします。Tier-2のNetScaler CPXは、そのトラフィックをKubernetesクラスター内のマイクロサービスにルーティングします。スタンドアロンポッドとして展開されたNetScaler Ingress Controllerは、Tier-1 NetScalerを構成します。そして、1つ以上のNetScaler CPXポッド内のサイドカーNetScaler Ingress Controllerは、同じポッド内の関連するNetScaler CPXを構成します。

デュアルティア展開は、ベアメタル環境のKubernetes、またはAWS、GCP、Azureなどのパブリッククラウドにセットアップできます。

次の図は、デュアルティア展開を示しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

セットアッププロセス

NetScaler Ingress Controllerのリポジトリには、デモンストレーション目的で、Tier-2用のサンプルApacheマイクロサービスとNetScaler CPXのマニフェスト、Tier-2 NetScaler CPX用のイングレスオブジェクト、NetScaler Ingress Controller、およびTier-1 NetScaler用のイングレスオブジェクトが用意されています。これらのサンプルは、デュアルティアトポロジを展開するためのセットアッププロセスで使用されます。

次の操作を実行します。

1. クラウドまたはオンプレミスにKubernetesクラスターを作成します。クラウド内のKubernetesクラスターは、マネージドKubernetes（例：GKE、EKS、AKS）またはカスタム作成されたKubernetes展開のいずれかです。
2. Kubernetesクラスターの外部で、マルチNIC展開モードでNetScaler MPXまたはVPXを展開します。

   - NetScaler MPXの展開手順については、NetScalerドキュメントを参照してください。
   - NetScaler VPXの展開手順については、NetScaler VPXインスタンスの展開を参照してください。

   NetScaler VPXまたはMPXを展開した後、次の操作を実行します。

   a. KubernetesクラスターのサブネットからIPアドレスをNetScalerのSNIPとして構成します。NetScalerでのSNIPの構成については、サブネットIPアドレス（SNIP）の構成を参照してください。

   b. Kubernetesクラスターと同じサブネットにあるSNIPの管理アクセスを有効にします。SNIPは、NetScaler Ingress ControllerがTier-1 NetScalerを構成できるように、NetScaler Ingress Controller YAMLファイルで`NS_IP`変数として使用する必要があります。

   > **注:**
   >
   > SNIPを`NS_IP`として使用することは必須ではありません。NetScalerの管理IPアドレスがNetScaler Ingress Controllerから到達可能な場合は、管理IPアドレスを`NS_IP`として使用できます。

   c. クラウド展開では、Tier-1 NetScaler VPXでMACベース転送モードを有効にします。NetScaler VPXはマルチNICモードで展開されているため、POD CNIネットワークまたはクライアントネットワークに到達するための戻りルートがありません。したがって、このシ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ナリオを処理するために、Tier-1 NetScaler VPXでMACベース転送モードを有効にする必要があります。

d. NetScaler Ingress Controller専用のNetScalerシステムユーザーアカウントを作成します。NetScaler Ingress Controllerは、システムユーザーアカウントを使用してTier-1 NetScalerを自動的に構成します。

e. オンプレミスのファイアウォールまたはクラウド上のセキュリティグループを設定して、NetScalerに必要なポートへのインバウンドトラフィックを許可します。セットアッププロセスではポート80とポート443を使用しますが、要件に基づいてこれらのポートを変更できます。

3. サンプルマイクロサービスを展開します。次のコマンドを使用します。

```
kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/dual-tier/manifest/apache.yaml
```

4. NetScaler CPXをTier-2イングレスとして展開します。次のコマンドを使用します。

```
kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/dual-tier/manifest/tier-2-cpx.yaml
```

5. Tier-2 NetScaler CPXのイングレスオブジェクトを作成します。次のコマンドを使用します。

```
kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/dual-tier/manifest/ingress-tier-2-cpx.yaml
```

6. Tier-1 NetScaler用のNetScaler Ingress Controllerを展開します。次の手順を実行します。

a. NetScaler Ingress Controllerマニフェストファイルをダウンロードします。次のコマンドを使用します。

```
wget https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/dual-tier/manifest/tier-1-vpx-cic.yaml
```

b. NetScaler Ingress Controllerマニフェストファイルを編集し、次の環境変数の値を入力します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

環境変数	必須またはオプション	説明
NS_IP	必須	NetScalerアプライアンスのIPアドレスです。詳細については、前提条件を参照してください。
NS_USER および NS_PASSWORD	必須	Ingressデバイスとして使用されるNetScaler VPXまたはMPXアプライアンスのユーザー名とパスワードです。詳細については、前提条件を参照してください。
EULA	必須	エンドユーザーライセンス契約です。値を`Yes`として指定します。
LOGLEVEL	オプション	NetScaler Ingress Controllerによって生成されるログを制御するためのログレベルです。デフォルトでは、値はDEBUGに設定されています。サポートされている値は、CRITICAL、ERROR、WARNING、INFO、およびDEBUGです。詳細については、ログレベルを参照してください。
NS_PROTOCOL および NS_PORT	オプション	NetScaler Ingress ControllerがNetScalerと通信するために使用する必要があるプロトコルとポートを定義します。デフォルトでは、NetScaler Ingress Controllerはポート443でHTTPSを使用します。ポート80でHTTPを使用することもできます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

環境変数	必須またはオプション	説明
イングレスクラス	任意	複数のイングレスロードバランサーが異なるイングレスリソースの負荷分散に使用される場合。この環境変数を使用して、特定のイングレスクラスに関連付けられたNetScalerを構成するようにNetScaler Ingress Controllerを指定できます。イングレスクラスの詳細については、イングレスクラスのサポートを参照してください。
NS_VIP	任意	NetScaler Ingress Controllerは、この環境変数で提供されるIPアドレスを使用して、イングレストラフィックを受信するNetScalerに仮想IPアドレスを構成します。

c. 更新されたNetScaler Ingress Controllerマニフェストファイルをデプロイします。次のコマンドを使用します。

```
kubectl create -f tier-1-vpx-cic.yaml
```

7. Tier-1 NetScaler用のイングレスオブジェクトを作成します。次のコマンドを使用します。

```
kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/dual-tier/manifest/ingress-tier-1-vpx.yaml
```

8. クラウドまたはオンプレミスのDNSサーバーの詳細を更新して、WebサイトをTier-1 NetScalerのVIPにポイントします。

例: `citrix-ingress.com 192.250.9.1`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ここで、`192.250.9.1`はTier-1 NetScalerのVIPであり、`citrix-ingress.com`はKubernetesクラスターで実行されているマイクロサービスです。

9. マイクロサービスのURLにアクセスして、デプロイメントを確認します。

ワンステップデプロイメントマニフェストファイルを使用したデュアルティアデプロイメントのセットアップ

簡単にデプロイできるように、NetScaler Ingress Controllerのリポジトリには、オールインワンのデプロイマニフェストが含まれています。ファイルをダウンロードし、以下の環境変数の値で更新して、マニフェストファイルをデプロイできます。

注:

「セットアッププロセス」のステップ1〜2を完了していることを確認してください。(#setup-process).

次の操作を実行します。

1. オールインワン展開マニフェストファイルをダウンロードします。次のコマンドを使用します。

```
wget https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/dual-tier/manifest/all-in-one-dual-tier-demo.yaml
```

2. オールインワン展開マニフェストファイルを編集し、次の環境変数の値を入力します。

環境変数	必須またはオプション	説明
NS_IP	必須	NetScalerアプライアンスのIPアドレスです。詳細については、「前提条件」を参照してください。(/ja-jp/netscaler-k8s-ingress-controller/deploy/unified-ingress-deployment.html#prerequisites).

© 1997–2026 Citrix Systems, Inc. All rights reserved.

環境変数	必須またはオプション	説明
NS_USER および NS_PASSWORD	必須	Ingressデバイスとして使用されるNetScaler VPXまたはMPXアプライアンスのユーザー名とパスワードです。詳細については、「前提条件」を参照してください。(/ja-jp/netscaler-k8s-ingress-controller/deploy/unified-ingress-deployment.html#prerequisites).
EULA	必須	エンドユーザーライセンス契約。値を`Yes`として指定します。
LOGLEVEL	オプション	NetScaler Ingress Controllerによって生成されるログを制御するログレベル。デフォルトでは、値はDEBUGに設定されています。サポートされている値は、CRITICAL、ERROR、WARNING、INFO、DEBUGです。詳細については、ログレベルを参照してください。
NS_PROTOCOL および NS_PORT	オプション	NetScaler Ingress ControllerがNetScalerと通信するために使用する必要があるプロトコルとポートを定義します。デフォルトでは、NetScaler Ingress Controllerはポート443でHTTPSを使用します。ポート80でHTTPを使用することもできます。
イングレスクラス	オプション	複数のイングレスロードバランサーが異なるイングレ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

環境変数	必須またはオプション	説明
		スリソースの負荷分散に使用される場合。この環境変数を使用して、特定のイングレスクラスに関連付けられたNetScalerを構成するようにNetScaler Ingress Controllerを指定できます。イングレスクラスの詳細については、(../configure/ingress-classes.htmlを参照してください。
NS_VIP	オプション	NetScaler Ingress Controllerは、この環境変数で提供されるIPアドレスを使用して、イングレストラフィックを受信するNetScalerに仮想IPアドレスを構成します。**注:** NS_VIPは、Ingress yamlでfrontend-ipアノテーションが提供されていない場合のフォールバックとして機能します。タイプLoadbalancerサービスではサポートされていません。

3. 更新されたオールインワンデプロイメントマニフェストファイルをデプロイします。次のコマンドを使用します。

```
kubectl create -f all-in-one-dual-tier-demo.yaml
```

NetScaler CPXのカスタムメトリックによる水平Podオートスケーラー

Kubernetesクラスターにワークロードを初めてデプロイする場合、リソース要件や、それらの要件が本番環境でどのように変化するかを正確に予測することは困難です。水平Podオートスケーラー (HPA) を使用すると、実際のリソース使用量など、さまざまなメトリックに基づいてワークロード内のPodの数を自動的にスケーリングできます。HPAは、デプロイ、レプリカセット、レプ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

リケーションコントローラーなど、Kubernetesベースのリソースをスケーリングするkubernetesが提供するリソースです。

従来、HPAは必要なメトリックをメトリックサーバーから取得します。その後、指定したターゲットに観測された平均メトリックを一致させるために、デプロイ内のレプリカ数を定期的に調整します。

NetScalerは、NetScaler CPX向けにカスタムメトリックベースのHPAソリューションを提供します。 デフォルトでは、メトリックサーバーはPodのCPUとメモリのメトリックのみを提供します。 NetScalerは、アプリケーションパフォーマンスを分析するための豊富な組み込みメトリックセットを提供しており、これらのメトリックに基づいてより適切なオートスケーリング判断を下すことができます。カスタムメトリックベースのHPAは、HTTPリクエストレート、SSLトランザクション、またはADC帯域幅に基づいたオートスケーリングのような、より優れたソリューションです。

NetScaler CPX HPAソリューション

NetScaler CPX HPAソリューションは、以下のコンポーネントで構成されています。

- NetScaler VPX: NetScaler VPXまたはMPXはTier-1にデプロイされ、クラスター内のNetScaler CPX Pod間でクライアント要求をロードバランシングします。
- NetScaler CPX: クラスター内にデプロイされたNetScaler CPXは、エンドポイントアプリケーションPodのTier-2ロードバランサーとして機能します。NetScaler CPX Podは、NetScaler Ingress ControllerおよびNetScalerメトリックエクスポーターと共にサイドカーとして実行されます。
- NetScaler Ingress Controller: NetScaler Ingress Controllerは、Kubernetes Ingressを中心に構築されたIngressコントローラーであり、Ingressリソース構成に基づいてNetScalerを自動的に構成します。スタンドアロンPodとしてデプロイされたNetScaler Ingress Controllerは NetScaler VPXを構成し、他のインスタンスはNetScaler CPXを構成します。
- NetScaler Metrics Exporter: NetScaler Metrics Exporter)は、アプリケーションパフォーマンスメトリックをオープンソースの監視システムPrometheusにエクスポートします。NetScaler Metrics ExporterはNetScaler CPXからメトリックを収集し、Prometheusが理解できる形式で公開します。
- Prometheus: Prometheusは、オープンソースのシステム監視およびアラートツールキットです。Prometheusは、NetScaler CPXからメトリックを収集し、HPAコントローラーがメトリックをチェックするためにクエリするPrometheusアダプターを使用してそれらを公開するために使用されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- Prometheusアダプター: Prometheusアダプターには、KubernetesリソースメトリックAPIとカスタムメトリックAPIの実装が含まれています。このアダプターは、Kubernetesバージョン1.6以降のautoscaling/v2 HPAでの使用に適しています。また、すでにPrometheusを実行し、適切なメトリックを収集しているクラスター上のメトリックサーバーを置き換えることもできます。

次の図は、NetScaler CPX HPAソリューションがどのように機能するかを視覚的に表したものです。

Tier-1 NetScaler VPX は、Tier-2 の NetScaler CPX をロードバランスします。NetScaler CPX はアプリケーションをロードバランスします。Prometheus、Prometheus-adapter、HPA コントローラーなどの他のコンポーネントもデプロイされます。

HPA コントローラーは、HTTP リクエストレートや帯域幅などのカスタムメトリクスについて、Prometheus-adapter を継続的にポーリングします。HPA でユーザーが定義した制限に達すると、HPA コントローラーは NetScaler CPX デプロイメントをスケーリングし、負荷を処理するために別の NetScaler CPX ポッドを作成します。

NetScaler CPX HPA ソリューションのデプロイ

NetScaler CPX HPA ソリューションをデプロイするには、次の手順を実行します。

1. 次のコマンドを使用して、GitHub から citrix-k8s-ingress-controller リポジトリをクローンします。

   ```
   git clone https://github.com/citrix/citrix-k8s-ingress-controller.git
   ```

   クローン後、次のコマンドでディレクトリを HPA フォルダーに変更します。

   ```
   cd citrix-k8s-ingress-controller/blob/master/docs/how-to/hpa
   ```

2. HPA ディレクトリから、`values.sh` ファイルを開いて編集し、NetScaler VPX に次の値を設定します。

   - `VPX_IP`: NetScaler VPX の IP アドレス
   - `VPX_PASSWORD`: NetScaler VPX 上の `nsroot` ユーザーのパスワード
   - `VIRTUAL_IP_VPX`: サンプルのゲストブックアプリケーションにアクセスする IP アドレス。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

3. `create_all.sh` ファイルを実行して、必要なすべてのリソースを作成します。

```
./create_all.sh
```

この手順では、次のリソースを作成します。

- 監視用の Prometheus と Grafana
- NetScaler Ingress Controller とメトリクスエクスポーターをサイドカーとして持つ NetScaler CPX
- NetScaler VPX を構成するためのスタンドアロンポッドとしての NetScaler Ingress Controller
- サンプルguestbookアプリケーション
- NetScaler CPXオートスケール展開を監視するためのHPAコントローラー
- カスタムメトリクスを公開するためのPrometheusアダプター

4. `hosts`ファイルにエントリを追加します。 guestbookアプリケーションのトラフィックをNetScaler VPX仮想IPアドレスにルーティングするには、`hosts`ファイルにルートを追加する必要があります。 ほとんどのLinuxディストリビューションでは、`hosts`ファイルは`/etc`フォルダーにあります。
5. 生成されたトラフィックを送信し、NetScaler CPXオートスケール展開を確認します。

NetScaler CPX展開HPAは、NetScaler CPXの平均HTTPリクエストレートが1秒あたり20リクエストを超えると、自動的にスケールするように構成されています。トラフィックを送信するには、HPAフォルダーに用意されている以下のスクリプトを使用できます。

- `16_curl.sh` - 1秒あたり16のHTTPリクエストを送信します（しきい値より少ない）
- `30_curl.sh` - 1秒あたり30のHTTPリクエストを送信します（しきい値より多い）

a. `16_curl.sh`スクリプトを実行して、NetScaler CPXに1秒あたり16のHTTPリクエストを送信します。

```
./16_curl.sh
```

次の図は、1秒あたりのHTTPリクエストを表示するGrafanaダッシュボードです。

次の出力は、16 HTTP RPSでのHPAの状態を示しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

b. `30_curl.sh`スクリプトを実行して、NetScaler CPXに1秒あたり30のHTTPリクエストを送信します。

```
./30_curl.sh
```

このスクリプトを実行すると、設定された20リクエストのしきい値を超え、NetScaler CPX展開は1つのポッドから2つのポッドに自動的にスケールします。2つのNetScaler CPXポッドがあるため、メトリック`HTTP request rate`の平均値も30から15に減少します。

次の出力は、ターゲットを超えたときのHPAの状態を示しています。

以下の出力は、NetScaler CPXのレプリカ数が2に増加し、HTTP RPSの平均値が15に減少したことを示しています。

以下の図は、2つのNetScaler CPXがトラフィックをロードバランシングしているGrafanaダッシュボードを示しています。

6. `delete_all.sh`スクリプトを実行してクリーンアップします。

```
./delete_all.sh
```

注:

Tier-1 NetScaler VPXが存在しない場合は、NodePortを使用してNetScaler CPXサービスを公開します。

ダイレクトサーバーリターンを展開する

一般的なロードバランシングシステムでは、ロードバランサーが Webサーバーとクライアント間の仲介役を務めます。受信したクライアントリクエストはロードバランサーによって受け取られ、データパケットにわずかな変更を加えて適切なサーバーに渡されます。サーバーは必要なデータでロードバランサーに応答し、その後ロードバランサーがその応答をクライアントに転送します。

ダイレクトサーバーリターン (DSR) 展開では、ロードバランサーはクライアントリクエストをサーバーに転送しますが、バックエンドサーバーは応答を直接クライアントに送信します。リクエストと応答に異なるネットワークパスを使用することで、余分なホップを回避し、遅延を削減で

© 1997–2026 Citrix Systems, Inc. All rights reserved.

きます。サーバーがクライアントに直接応答するため、DSRはクライアントとサーバー間の応答時間を高速化し、ロードバランサーからの余分な負荷も軽減します。DSRを使用することは、インフラストラクチャの変更をほとんどまたはまったく行わずに、アプリケーションのネットワークパフォーマンスを向上させる透過的な方法です。 NetScalerを使用したDSRの詳細については、NetScalerドキュメントを参照してください。

DSRソリューションは、以下の状況で役立ちます。

- 低遅延（応答時間）が重要なビデオストリーミングを提供するアプリケーションを処理する場合。
- インテリジェントなロードバランシングが不要な場合
- ロードバランサーの出力容量がボトルネックになる可能性がある場合

ただし、DSRを使用する場合、高度なレイヤー7ロードバランシング機能はサポートされません。

NetScalerを使用したKubernetes向けDSRネットワークトポロジ

このトポロジでは、外部ロードバランサー（Tier-1 ADC）が、オーバーレイ（L3 DSR IPIP）を介してKubernetesクラスター内に展開されたイングレスADC（Tier 2 ADC）にトラフィックを分散します。Tier-2 ADCはパケットを受け取り、パケットをデカプセル化し、サービス間でロードバランシングを実行します。Tier-2 ADCは、Tier-1 ADCを介して送信する代わりに、サービスからの戻りトラフィックをクライアントに送信します。

NetScalerを使用したクラウドネイティブアプリケーション向けDSRの展開

Kubernetesクラスターに展開されたアプリケーション向けにDSRを展開するには、以下のセクションの手順を実行します。

NetScaler CPXをTier-2 ADCとして展開する

このセクションでは、DSRトポロジのイングレスデバイスで必要となる構成を作成する手順について説明します。

1. 以下のコマンドを使用してDSR用の名前空間を作成します。

```
kubectl apply -f  https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/example/dsr/KubernetesConfig/dsr_namespace.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

2. 以下のコマンドを使用してConfigMapを作成します。

```
kubectl apply -f  https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/example/dsr/KubernetesConfig/cpx_config.yaml
 -n dsr
```

注:

この例では、ノードコントローラーネットワークは192.168.1.0/24として構成されています。したがって、IPトンネルを作成するコマンドは`add iptunnel dsr 192.168.1.254 255.255.255.0 *`として提供されます。CNC構成に従って値を指定する必要があります。

3. 名前空間`dsr`にNetScaler CPXを展開します。

```
kubectl apply -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/example/dsr/KubernetesConfig/citrix-k8s-cpx-ingress.yml  -n dsr
```

Kubernetesクラスターへのサンプルアプリケーションの展開

このセクションの手順を実行して、Kubernetesクラスターにサンプルアプリケーションを展開します。

1. 以下のコマンドを使用してゲストブックアプリケーションを展開します。

```
kubectl apply -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/example/dsr/KubernetesConfig/guestbook-all-in-one.yaml  -n dsr
```

2. Ingressを使用してゲストブックアプリケーションを公開します。

   a. 以下のコマンドを使用してゲストブックIngress YAMLファイルをダウンロードします。

```
wget https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/example/dsr/KubernetesConfig/guestbook-all-in-one.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

b. `ingress.citrix.com/frontend-ip:` アノテーションを使用して、アプリケーションにアクセスするためのDSR IPまたはパブリックIPアドレスを編集して指定します。

```
ingress.citrix.com/frontend-ip: "&lt;ip-address>"
```

c. YAMLファイルを保存し、以下のコマンドを使用してIngressリソースを展開します。

```
kubectl apply -f guestbook-ingress.yaml -n dsr
```

Tier-1とTier-2 ADC間のネットワーク接続を確立する

このセクションの手順を実行して、Tier-1とTier-2 ADC間のネットワーク接続を確立します。

1. 以下のコマンドを使用してノードコントローラーを展開するためのYAMLをダウンロードします。

```
wget https://raw.githubusercontent.com/citrix/citrix-k8s-node-
controller/master/deploy/citrix-k8s-node-controller.yaml
```

2. YAMLファイルを編集し、NS_IP、NS_USER、NS_PASSWORD、およびREMOTE_VTEPIP引数の値を指定します。詳細については、ノードコントローラーを参照してください。
3. YAMLファイルを保存し、ノードコントローラーを展開します。

```
kubectl create -f citrix-k8s-node-controller.yaml -n dsr
```

Tier-1 ADC 用のNetScaler Ingress Controllerを展開し、NetScaler CPXをサービスとして公開する

NetScaler Ingress Controllerをスタンドアロンポッドとして展開し、Tier-2 NetScaler CPXのIngressリソースを作成するには、次の手順を実行します。

1. 次のコマンドを使用して、NetScaler Ingress ControllerのYAMLファイルをダウンロードします。

```
wget https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-
controller/master/example/dsr/KubernetesConfig/citrix-k8s-ingress-
controller.yaml
```

2. YAMLファイルを編集し、NetScaler Ingress Controllerの次の値を更新します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- NS_IP
- NS_USER
- NS_PASSWORD

詳細については、「YAMLを使用してNetScaler Ingress Controllerを展開する」を参照してください。

3. YAMLファイルを保存し、NetScaler Ingress Controllerを展開します。

```
kubectl create -f citrix-k8s-ingress-controller.yaml -n dsr
```

4. Tier-2 NetScaler CPXのIngressリソースを作成して、Tier-1 ADCでDSR構成を作成します。

```
wget https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-
controller/master/example/dsr/KubernetesConfig/vpx-ingress.yaml
```

5. YAMLファイルを編集し、`ingress.citrix.com/frontend-ip:`アノテーションを使用してユーザーがアプリケーションにアクセスするDSRまたはパブリックIPアドレスを指定します。このIPアドレスは、手順2で指定したIPアドレスと同じである必要があります。

```
kubectl apply -f vpx-ingress.yaml  -n dsr
```

DSR展開をテストする

DSR展開をテストするには、`ingress.citrix.com/frontend-ip:`アノテーションで指定されたIPアドレスを使用してブラウザからアプリケーションにアクセスします。ゲストブックページが表示されます。

サンプル出力は以下のとおりです。

トラブルシューティング

アプリケーションをテストする際、必要な設定がすべて作成されていても、ページが表示されない場合があります。これは、ホスト上の`rp_filter`ルールが原因です。このような問題が発生した場合は、すべてのホストで以下のコマンドを使用してルールを無効にしてください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
sysctl -w net.ipv4.conf.all.rp_filter=0

sysctl -w net.ipv4.conf.cni0.rp_filter=0

sysctl -w net.ipv4.conf.eth0.rp_filter=0

sysctl -w net.ipv4.conf.cni0.rp_filter=0

sysctl -w net.ipv4.conf.default.rp_filter=0
```

アドミッションコントローラーWebフックのサポート

アドミッションコントローラーは、オブジェクトの永続化に先立ってKubernetes APIサーバーへのリクエストを傍受するための強力なツールです。Kubernetesアドミッションコントローラーを使用すると、クラスターで実行を許可するものを定義し、カスタマイズできます。そのため、クラスター管理者がクラスターに予防的なセキュリティ制御を展開するための便利なツールです。ただし、アドミッションコントローラーを`kube-apiserver`バイナリにコンパイルする必要があり、柔軟性が限られています。

この制限を克服するために、Kubernetesは、拡張機能として開発し、実行時に構成されたWebフックとして実行できる動的アドミッションコントローラーをサポートしています。

アドミッションコントローラーWebフックを使用すると、Kubernetesクラスター管理者は、再コンパイルすることなく、APIサーバーのアドミッションチェーンに追加のプラグインを作成できます。アドミッションコントローラーWebフックは、リソースが作成、更新、または削除されるたびに実行できます。

アドミッションコントローラーWebフックには、次の2種類を定義できます。

- 検証アドミッションWebフック
- 変更アドミッションWebフック

変更アドミッションWebフックが最初に呼び出され、カスタムのデフォルトを適用するためにAPIサーバーに送信されたオブジェクトを変更できます。すべてのオブジェクトの変更が完了し、受信オブジェクトがAPIサーバーによって検証されると、検証アドミッションWebフックが呼び出されます。検証アドミッションフックはリクエストを処理し、カスタム ポリシーを適用するためにリクエストを承認または拒否します。

次の図は、アドミッションコントローラーWebフックの仕組みを示しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アドミッションWebフックが役立つシナリオをいくつか示します。

- 名前空間またはクラスター全体で合理的なセキュリティベースラインを義務付けるため。たとえば、コンテナがrootとして実行されるのを禁止したり、コンテナのルートファイルシステムが常に読み取り専用としてマウントされるようにしたりします。
- ラベル、アノテーション、またはリソース制限に関する特定の標準とプラクティスへの準拠を強制するため。たとえば、さまざまなオブジェクトに適切なラベルが使用されていることを確認するために、異なるオブジェクトに対してラベル検証を強制します。
- クラスターで実行されているオブジェクトの構成を検証し、クラスターに影響を与える可能性のある明らかな誤構成を防ぐため。 たとえば、セマンティックタグなしでデプロイされたイメージを検出して修正するため。

アドミッションコントローラーの適用方法

特定のユースケースごとにアドミッションコントローラーを記述することはスケーラブルではなく、さまざまなリソースタイプとフィールドをカバーする複数の構成をサポートするシステムがあることが役立ちます。Kubernetes 用のカスタマイズ可能なアドミッションWebhookを実装するには、Open policy agent (OPA) と Gatekeeper を使用できます。

OPAは、スタック全体でポリシー適用を統一するオープンソースの汎用ポリシーエンジンです。Gatekeeperは、OPAによって実行されるCRDベースのポリシーを適用するカスタマイズ可能な検証Webhookです。

(画像 クレジット)

Gatekeeperは以下の機能を提供します

- 拡張可能でパラメータ化されたポリシーライブラリ
- ポリシーライブラリをインスタンス化するためのネイティブKubernetes CRD（制約）
- ポリシーライブラリを拡張するためのネイティブKubernetes CRD（制約テンプレート）
- 監査機能

アドミッションコントローラーWebhookの記述とデプロイ

前提条件

- admissionregistration.k8s.io/v1beta1 APIが有効になっているKubernetes 1.14.0以降。APIが有効になっているかどうかは、次のコマンドを使用して確認できます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl api-versions | grep admissionregistration.k8s.io/v1beta1
```

次の出力は、APIが有効になっていることを示しています。

```
admissionregistration.k8s.io/v1beta1
```

- ミューティングアドミッションWebhookと検証アドミッションWebhookアドミッションコントローラーは、`kube-apiserver` の admission-control フラグに正しい順序で追加され、リストされている必要があります。

Minikubeを使用すると、次のコマンドでMinikubeを起動することでこのタスクを実行できます。

```
minikube start --extra-config=apiserver.enable-admission-
plugins=NamespaceLifecycle,LimitRanger,ServiceAccount,DefaultStorageClass,
DefaultTolerationSeconds,NodeRestriction,MutatingAdmissionWebhook,Validati
ngAdmissionWebhook`
```

- クラスター管理者権限があることを確認してください。

```
kubectl create clusterrolebinding cluster-admin-binding --clusterrole
cluster-admin --user &lt;YOUR USER NAME>
```

ミューティングアドミッションWebhook構成

ミューティングアドミッションWebhook構成の詳細については、ingress-admission-webhookを参照してください。

ミューティングアドミッションWebhookの例では、以下のユースケースがカバーされています。

- Ingress名に基づいてIngressのポートを更新する
- 名前空間に基づいてセキュアなバックエンドを強制的に有効にする

Gatekeeperを使用したバリデーションアドミッションWebhook構成

Gatekeeperは、Kubernetesリソースとして制約を作成できるCRDを使用します。このCRDはGatekeeperでは`ConstraintTemplate`と呼ばれます。制約のスキーマにより、管理者は関数の引数と同様に、制約の動作を微調整できます。制約は、管理者が制約テンプレートをどのように適用したいかをGatekeeperに通知するために使用されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

制約テンプレートを使用して、さまざまなポリシーを適用できます。さまざまな例がGatekeeperライブラリに記載されています。

サンプルポリシーの展開

Gatekeeperを使用して`HttpsOnly`をサンプルポリシーとして展開するには、次の手順を実行します。`HttpsOnly`ポリシーは、HTTPSを使用したIngress構成のみを許可します。

1. 次のコマンドを使用してGatekeeperをインストールします。

   注:

   この手順では、Gatekeeperは事前構築済みのイメージを使用してインストールされます。Gatekeeperのインストールに記載されているさまざまな方法を使用してGatekeeperをインストールできます。

   ```
   # kubectl apply -f https://raw.githubusercontent.com/open-policy-agent/gatekeeper/master/deploy/gatekeeper.yaml
   ```

   次のコマンドを使用してインストールを確認できます。

   ```
   kubectl get crd | grep -i constraintsonstrainttemplates.templates.gatekeeper.sh
   ```

   次のコマンドを使用して、すべての制約テンプレートを確認できます。

   ```
   kubectl get constrainttemplates.templates.gatekeeper.sh
   ```

2. `httpsonly`制約テンプレートを適用します。

   ```
   kubectl apply -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/docs/how-to/webhook/httpsonly/template.yaml
   ```

3. `httpsonly`ポリシーを適用するために制約を適用します。

   ```
   kubectl apply -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/docs/how-to/webhook/httpsonly/constraint.yaml
   ```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

4. ポリシーに違反するサンプルIngressを展開して、ポリシーを検証します。Ingressの作成中にエラーが表示されるはずです。

```
kubectl apply -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/docs/how-to/webhook/httpsonly/bad-example-ingress.yaml



Error from server ([denied by ingress-https-only] Ingress must be https. tls configuration is required for test-ingress): error when creating "ingress.yaml": admission webhook "validation.gatekeeper.sh" denied the request: [denied by ingress-https-only] Ingress must be https. tls configuration is required for test-ingress
```

5. 次に、Ingressに必要なTLSセクションを持つIngressを展開します。

```
 # kubectl apply -f  https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/docs/how-to/webhook/httpsonly/good-example-ingress.yaml



 ingress.networking.k8s.io/test-ingress created
```

6. Gatekeeperポリシーの検証が完了したら、以下のコマンドを使用してインストールをクリーンアップします。

```
Uninstall all packages and template installed.

kubectl delete -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/docs/how-to/webhook/httpsonly/good-example-ingress.yaml

kubectl delete -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/docs/how-to/webhook/httpsonly/constraint.yaml

kubectl delete -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/docs/how-to/webhook/httpsonly/template.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl delete -f https://raw.githubusercontent.com/open-policy-agent/
gatekeeper/master/deploy/gatekeeper.yaml
```

その他のサンプルユースケース

webhookディレクトリの下に複数のユースケースがリストされています。 手順は例で指定されているものと同様であり、次のように要約できます。

1. 各ユースケースディレクトリに記載されているテンプレートYAMLファイルを適用します。
2. 制約YAMLファイルを適用します。
3. 不正な、または適切なサンプルYAMLファイルを適用して、ユースケースを検証します。

その他のユースケースについては、Gatekeeperライブラリを参照してください。

NetScaler Ingress Controller を使用して gRPC サポートを有効にする

gRPC は、Google が開発した高性能なオープンソースのユニバーサルRPCフレームワークです。 gRPCでは、クライアントアプリケーションは、ローカルメソッドを呼び出すのと同じ方法で、別のサーバーからサーバーアプリケーション上のメソッドを直接呼び出すことができます。 GRPC を使用すると、分散アプリケーションやサービスを簡単に作成できます。

gRPC サポートを有効にする

HTTP2 を使用して GRPC サポートを有効にするには、次の手順を実行します。

1. YAML ファイル `cic-configmap.yaml` を作成し、ConfigMap の次のエントリを使用して HTTP2 サーバー側サポートのグローバルパラメータを有効にします。ConfigMap の使用方法の詳細については、ConfigMap ドキュメント を参照してください。

   ```
   NS_HTTP2_SERVER_SIDE: 'ON'
   ```

2. 次のコマンドを使用して ConfigMap を適用します。

   ```
   kubectl apply -f cic-configmap.yaml
   ```

3. ConfigMap をサポートするように NetScaler Ingress Controller を展開するための `cic.yaml` ファイルを編集します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
args:

  - --ingress-classes

    citrix®

  - --configmap

    default/cic-configmap
```

4. 編集した YAML ファイルを適用して、NetScaler Ingress Controller をスタンドアロンポッドとして展開します。

```
kubectl apply -f cic.yaml
```

5. gRPC トラフィックをテストするには、`grpcurl` をインストールする必要がある場合があります。Linux マシンに `grpcurl` をインストールするには、次の手順を実行します。

```
go get github.com/fullstorydev/grpcurl

go install github.com/fullstorydev/grpcurl/cmd/grpcurl
```

6. gRPC テストサービス YAML ファイル (`grpc-service.yaml`) を適用します。

```
kubectl apply -f grpc-service.yaml
```

`grpc-service.yaml` ファイルのサンプルコンテンツは次のとおりです。

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: grpc-service

spec:

  replicas: 1
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  selector:
    matchLabels:
      app: grpc-service
  template:
    metadata:
      labels:
        app: grpc-service
    spec:
      containers:
      - image: registry.cn-hangzhou.aliyuncs.com/acs-sample/grpc-
server:latest
        imagePullPolicy: Always
        name: grpc-service
        ports:
        - containerPort: 50051
          protocol: TCP
      restartPolicy: Always
---
apiVersion: v1
kind: Service
metadata:
  name: grpc-service
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:

  ports:

  - port: 50051

    protocol: TCP

    targetPort: 50051

  selector:

    app: grpc-service

  sessionAffinity: None

  type: NodePort
```

7. gRPC Ingress 構成の証明書を作成します。

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout tls.key
-out tls.crt -subj "/CN=grpc.example.com/O=grpc.example.com"



kubectl create secret tls grpc-secret --key tls.key --cert tls.crt



secret "grpc-secret" created
```

8. Ingress アノテーションを使用して HTTP2 を有効にします。NetScaler Ingress Controller を使用して HTTP2 を有効にする手順については、HTTP/2 サポート を参照してください。

   - フロントエンド Ingress 構成用の YAML ファイルを作成し、それを適用してコンテンツスイッチング仮想サーバーで HTTP/2 を有効にします。

     kubectl apply -f frontend-ingress.yaml を実行します。

`frontend-ingress.yaml` ファイルの内容は以下のとおりです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  annotations:

    ingress.citrix.com/frontend-httpprofile: '{"http2":"enabled",
"http2direct" : "enabled"}'

    ingress.citrix.com/frontend-ip: 192.0.2.1

    ingress.citrix.com/secure-port: "443"

  name: frontend-ingress

spec:

  ingressClassName: citrix

  rules:

  - {}

  tls:

  - {}
```

- 以下の内容でバックエンドIngress構成用のYAMLファイルを作成し、バックエンド（サービスグループ）でHTTP2を有効にするために適用します。

  kubectl apply -f backend-ingress.yaml を実行します。

`backend-ingress.yaml`ファイルの内容は以下のとおりです。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  annotations:
    ingress.citrix.com/backend-httpprofile: '{"grpc-service":{"http2":
"enabled", "http2direct" : "enabled"}}'
    ingress.citrix.com/frontend-ip: 192.0.2.2
    ingress.citrix.com/secure-port: "443"
  name: grpc-ingress
spec:
  ingressClassName: citrix
  rules:
  - host: grpc.example.com
    http:
      paths:
      - backend:
          service:
            name: grpc-service
            port:
              number: 50051
        path: /
        pathType: Prefix
  tls:
  - hosts:
    - grpc.example.com
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    secretName: grpc-secret
```

9. `grpcurl`コマンドを使用してgRPCトラフィックをテストします。

```
grpcurl -v -insecure -d '{"name": "gRPC"}' grpc.example.com:443
helloworld.Greeter.SayHello
```

コマンドの出力は以下のとおりです。

```
Resolved method descriptor:

rpc SayHello ( .helloworld.HelloRequest ) returns
( .helloworld.HelloReply );




Request metadata to send:

(empty)




Response headers received:

content-type: application/grpc




Response contents:

{

    "message": "Hello gRPC"

}
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
Response trailers received:
(empty)
Sent 1 request and received 1 response
```

レート制限CRDを検証する

レート制限CRDを検証するには、以下の手順を実行します。

1. ratelimit-crd.yamlファイルを使用してレート制限CRDを適用します。

   kubectl create -f ratelimit-crd.yaml を実行します。

2. レート制限ポリシーのために、以下の内容でYAMLファイル (ratelimit-crd-object.yaml) を作成します。

```
apiVersion: citrix.com/v1beta1
kind: ratelimit
metadata:
  name: throttle-req-per-clientip
spec:
  servicenames:
    - grpc-service
  selector_keys:
  basic:
    path:
    - "/"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    per_client_ip: true

  req_threshold: 5

  timeslice: 60000

  throttle_action: "RESPOND"
```

3. 以下のコマンドを使用してYAMLファイルを適用します。

```
kubectl create -f ratelimit-crd-object.yaml
```

4. `grpcurl`コマンドを使用してgRPCトラフィックをテストします。

```
grpcurl -v -insecure -d '{"name": "gRPC"}' grpc.example.com:443
helloworld.Greeter.SayHello
```

レート制限に達した後、コマンドは応答として以下のエラーを返します。

```
Error invoking method "helloworld.Greeter.SayHello": failed to query
for service descriptor "helloworld.Greeter": rpc error: code =
Unavailable desc =



Too Many Requests: HTTP status code 429; transport: missing content-
type field
```

gRPCでリライトおよびレスポンダーCRDを検証する

RewriteおよびResponder CRDを検証するには、次の手順を実行します。

1. rewrite-responder-policies-deployment.yamlファイルを使用して、RewriteおよびResponder CRDを適用します。

   kubectl create -f rewrite-responder-policies-deployment.yaml

2. リライトポリシーのYAMLファイル (rewrite-crd-object.yaml) を次の内容で作成します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: addcustomheaders

spec:

  rewrite-policies:

    - servicenames:

        - grpc-service

      rewrite-policy:

        operation: insert_http_header

        target: 'sessionID'

        modify-expression: '"48592th42gl24456284536tgt2"'

        comment: 'insert SessionID in header'

        direction: RESPONSE

        rewrite-criteria: 'http.res.is_valid'
```

3. 次のコマンドを使用してYAMLファイルを適用します。

```
kubectl create -f rewrite-crd-object.yaml
```

4. `grpcurl`コマンドを使用してgRPCトラフィックをテストします。

```
grpcurl -v -insecure -d '{"name": "gRPC"}' grpc.example.com:443
helloworld.Greeter.SayHello
```

このコマンドは、gRPCリクエスト応答にセッションIDを追加します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
Resolved method descriptor:

rpc SayHello ( .helloworld.HelloRequest ) returns
( .helloworld.HelloReply );



Request metadata to send:

(empty)



Response headers received:

content-type: application/grpc

sessionid: 48592th42gl24456284536tgt2



Response contents:

{

  "message": "Hello gRPC"

}



Response trailers received:

(empty)

Sent 1 request and received 1 response
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

複数のKubernetesクラスターに対するポリシーベースルーティング

複数のKubernetesクラスターの負荷分散に単一のNetScalerを使用する場合、NetScaler Ingress Controllerは静的ルートを介してNetScalerにPod CIDRネットワークを追加します。これらのルートは、Kubernetes PodとNetScaler間のネットワーク接続を確立します。ただし、Pod CIDRが重複すると、ルートの競合が発生する可能性があります。NetScalerは、このようなシナリオでのネットワーク競合に対処するために、ポリシーベースルーティング (PBR) をサポートしています。PBRでは、指定した基準に基づいてルーティング決定が行われます。通常、NetScaler VPXまたはNetScaler MPXが選択されたパケットを送信するネクストホップが指定されます。GSLB Kubernetes環境では、各KubernetesクラスターまたはNetScaler Ingress Controller用にサブネットIPアドレス (SNIP) を予約することでPBRが実装されます。ネットプロファイルを使用すると、SNIPは同じNetScaler Ingress Controllerによって作成されたすべてのサービスグループにバインドされます。同じクラスターに属するサービスグループから生成されるすべてのトラフィックについて、送信元IPアドレスは同じSNIPになります。

次のサンプルとポロジでは、NetScaler MPXまたはNetScaler VPXを使用して負荷分散される2つのKubernetesクラスターに対してPBRが構成されています。

PBR構成(/en-us/netscaler-k8s-ingress-controller/media/pbr.png)

NetScaler Ingress Controllerを使用したPBRの構成

PBRを構成するには、Kubernetesクラスターごとに1つ以上のSNIPが必要です。NSIC Helmインストールコマンドの`nsSNIPS`パラメーターを使用してSNIP値を指定できます。

Helmインストールコマンドで次の`values.yml`ファイルを使用して、NetScaler Ingress Controllerを展開し、PBRを構成します。

```
nsIP: <NS_IP>

license:

  accept: yes

adcCredentialSecret: <secret>

nodeWatch: True

clusterName: <name of cluster>
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
nsSNIPS: '["1.2.3.4", "5.6.7.8"]'
```

NetScaler Ingress Controller展開後のNetScalerでのPBR構成の検証

この検証例では、NetScaler Ingress Controllerがデプロイされた2ノードのKubernetesクラスターと、2つのSNIPを持つ次のConfigMapを使用します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cic-configmap
  namespace: default
data:
   NS_SNIPS: '["1.2.3.4","5.6.7.8"]'
```

NetScaler Ingress ControllerがNetScalerに次の構成を追加することを確認できます。

1. すべてのNS_SNIPのIPセットが追加されているかどうかを確認するには、次のコマンドを実行します: `show ipset k8s-pbr_ipset`。

   イメージ(https://user-images.githubusercontent.com/46886297/117246342-19519a00-ae5a-11eb-8e65-70944c24ef51.png)

2. ネットプロファイルが追加され、`SrcIP`が`IPset`に設定されます。

   ```
   > show netprofile k8s-pbr_netprof

   1)      Name: k8s-pbr_netprof

           SrcIP: k8s-pbr_ipset
           SrcIPPersistency: DISABLED
           OverrideLSN: DISABLED
           MBF: DISABLED
           Proxy Protocol: DISABLED
           Proxy Protocol version: V1
           Proxy Protocol After TLS: DISABLED
    Done
   ```

3. NetScaler Ingress Controllerによって追加されたサービスグループにネットプロファイルセットが含まれているかどうかを確認するには、`show servicegroup`を実行します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
> show servicegroup
1)      k8s-frontend_80_sgp_5udfadwklqkfukj3rsuoao5b6wxauf5j - HTTP
        State: ENABLED  Effective State: UP     Monitor Threshold : 0
        Max Conn: 0     Max Req: 0      Max Bandwidth: 0 kbits
        Use Source IP: NO
        Client Keepalive(CKA): NO
        Monitoring Owner: 0
        TCP Buffering(TCPB): NO
        HTTP Compression(CMP): NO
        Idle timeout: Client: 180 sec   Server: 360 sec
        Client IP: DISABLED
        Cacheable: NO
        SC: OFF
        SP: OFF
        Down state flush: ENABLED
        Monitor Connection Close : NONE
        Appflow logging: ENABLED
        ContentInspection profile name: ???
        Network profile name: k8s-pbr_netprof
        Process Local: DISABLED
        Traffic Domain: 0
        Comment: "ing:web-ingress,ingport:80,ns:default,svc:frontend,svcport:80"
 Done
```

4. PBRが追加されているかを確認するには、次のコマンドを実行します: `shpbr`。

```
> shpbr
1)      Name: k8s-1.2.3.4_10.106.175.86
        Action: ALLOW                           Hits: 0
        srcIP = 1.2.3.4
        destIP = 192.168.0.0-192.168.0.255
        srcMac:
        Protocol:
        Vlan:                                   Interface:
        Active Status: ENABLED                  Applied Status: APPLIED
        Priority: 30
        NextHop: 10.106.175.86

2)      Name: k8s-5.6.7.8_10.106.175.86
        Action: ALLOW                           Hits: 0
        srcIP = 5.6.7.8
        destIP = 192.168.0.0-192.168.0.255
        srcMac:
        Protocol:
        Vlan:                                   Interface:
        Active Status: ENABLED                  Applied Status: APPLIED
        Priority: 40
        NextHop: 10.106.175.86

3)      Name: k8s-1.2.3.4_10.106.175.87
        Action: ALLOW                           Hits: 0
        srcIP = 1.2.3.4
        destIP = 192.168.16.0-192.168.16.255
        srcMac:
        Protocol:
        Vlan:                                   Interface:
        Active Status: ENABLED                  Applied Status: APPLIED
        Priority: 50
        NextHop: 10.106.175.87

4)      Name: k8s-5.6.7.8_10.106.175.87
        Action: ALLOW                           Hits: 0
        srcIP = 5.6.7.8
        destIP = 192.168.16.0-192.168.16.255
        srcMac:
        Protocol:
        Vlan:                                   Interface:
        Active Status: ENABLED                  Applied Status: APPLIED
        Priority: 60
        NextHop: 10.106.175.87
```

ここで:

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- PBRの数は、（SNIPの数）`*`（Kubernetesノードの数）に相当します。この場合、4つ（2`*`2）のPBRが追加されます。
- PBRの`srcIP`は、ConfigMapによってNetScaler Ingress Controllerに提供される`NS_SNIPS`です。`destIP`は、KubernetesノードのCNIオーバーレイサブネット範囲です。
- `NextHop`はKubernetesノードのIPアドレスです。

5. NetScaler Ingress Controllerのログを使用して、構成を検証することもできます。

```
root@flannel-master:~/pbr-demo# logs -f cic-vpx | grep pbr
2021-05-06 05:46:41,506  - INFO - [NSPBRConfig.py:configure_snips_netprofile_for_pbr:95] (MainThread) Successfully Added IP 5.6.7.8 to ADC
2021-05-06 05:46:41,539  - INFO - [NSPBRConfig.py:configure_snips_netprofile_for_pbr:95] (MainThread) Successfully Added IP 1.2.3.4 to ADC
2021-05-06 05:46:41,701  - INFO - [nitrointerface.py:add_ns_pbroute:4709] (MainThread) PBRoute for 192.168.0.0-192.168.0.255 with gateway:10.106.175.86 srcip:5.6.7.8 added to ADC
2021-05-06 05:46:41,739  - INFO - [nitrointerface.py:add_ns_pbroute:4709] (MainThread) PBRoute for 192.168.0.0-192.168.0.255 with gateway:10.106.175.86 srcip:1.2.3.4 added to ADC
2021-05-06 05:46:41,771  - INFO - [nitrointerface.py:add_ns_pbroute:4709] (MainThread) PBRoute for 192.168.16.0-192.168.16.255 with gateway:10.106.175.87 srcip:5.6.7.8 added to ADC
2021-05-06 05:46:41,802  - INFO - [nitrointerface.py:add_ns_pbroute:4709] (MainThread) PBRoute for 192.168.16.0-192.168.16.255 with gateway:10.106.175.87 srcip:1.2.3.4 added to ADC
```

NetScalerノードコントローラーを使用してPBRを構成する

複数のKubernetesクラスターに対して、NetScalerノードコントローラーを使用してPBRを構成できます。ネットワーキングにNetScalerノードコントローラーを使用し、単一のNetScalerで複数のKubernetesクラスターの負荷分散を行う場合、VXLANトンネルインターフェースのIPアドレスにパケットを転送するために追加された静的ルートが、ルーティングの競合を引き起こす可能性があります。PBRをサポートするには、NetScalerノードコントローラーがNetScaler Ingress Controllerと連携して、ネットプロファイルをサービスグループにバインドする必要があります。

NetScalerノードコントローラーを使用してPBRを構成するには、次の手順を実行します。

1. NetScalerノードコントローラーの起動時に、`CLUSTER_NAME`を環境変数として指定します。この変数を指定すると、それがマルチクラスター展開であり、NetScalerノBRが静的ルートではなくPBRを構成することを示します。

   例:

   ```
   helm install nsnc netscaler/netscaler-node-controller --set license.accept=yes,nsIP=<NSIP>,vtepIP=<NetScaler SNIP>,vxlan.id=<VXLAN ID>,vxlan.port=<VXLAN PORT>,network=<IP-address-range-for-VTEP-overlay>,adcCredentialSecret=<Secret-for-NetScaler-credentials>,cniType=<CNI-overlay-name>,clusterName=<cluster-name>
   ```

2. NetScaler Ingress Controllerを展開する際、`CLUSTER_NAME`を環境変数として提供します。この値はノードコントローラーで提供された値と同じである必要があり、`nsncPbr=True`引数を使用してPBRを有効にし、NetScalerでPBRを設定します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注記:

- NetScalerノードコントローラーとNetScaler Ingress Controllerのデプロイファイルで `CLUSTER_NAME` に提供される値は、同じKubernetesクラスターにデプロイされている場合、一致する必要があります。
- `CLUSTER_NAME` は、ネットプロファイルエンティティを作成し、それをNetScaler VPXまたはNetScaler MPX上のサービスグループにバインドする際に使用されます。

NetScalerノードコントローラーのデプロイ後にNetScalerでのPBR構成を検証する

この検証例では、NetScalerノードコントローラーとNetScaler Ingress Controllerがデプロイされた2ノードのKubernetesクラスターを使用します。

NetScalerノードコントローラーによってNetScalerに以下の構成が追加されていることを確認できます。

1. 次のコマンドを実行します: `show netprofile`。

   ネットプロファイルが追加され、 `srcIP` の値は、NetScalerとKubernetesノード間のVXLANトンネルネットワークを作成する際にノードコントローラーによって追加されたSNIPに設定されます。

   ```
   > show netprofile

   1)      Name: cnc-cluster1_netprof

           SrcIP: 162.1.11.254
           SrcIPPersistency: DISABLED
           OverrideLSN: DISABLED
           MBF: DISABLED
           Proxy Protocol: DISABLED
           Proxy Protocol version: V1
    Done
   ```

2. 次のコマンドを実行します: `shservicegroup`。

   NetScaler Ingress Controllerは、ネットプロファイルを作成するサービスグループにバインドします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
> shservicegroup
1)      k8s-frontend_80_sgp_5udfadwklqkfukj3rsuoao5b6wxauf5j - HTTP
        State: ENABLED  Effective State: UP     Monitor Threshold : 0
        Max Conn: 0     Max Req: 0      Max Bandwidth: 0 kbits
        Use Source IP: NO
        Client Keepalive(CKA): NO
        Monitoring Owner: 0
        TCP Buffering(TCPB): NO
        HTTP Compression(CMP): NO
        Idle timeout: Client: 180 sec   Server: 360 sec
        Client IP: DISABLED
        Cacheable: NO
        SC: OFF
        SP: OFF
        Down state flush: ENABLED
        Monitor Connection Close : NONE
        Appflow logging: ENABLED
        ContentInspection profile name: ???
        Network profile name: cnc-cluster1_netprof
        Process Local: DISABLED
        Traffic Domain: 0
        Comment: "ing:web-ingress,ingport:80,ns:default,svc:frontend,svcport:80"
Done
```

3. 次のコマンドを実行します: `show pbr`。

   NetScalerノードコントローラーはPBRを追加します。

```
> show pbr
1)      Name: cnc-cluster1_192.168.16.0-192.168.16.255
        Action: ALLOW                           Hits: 337
        srcIP = 162.1.11.254
        destIP = 192.168.16.0-192.168.16.255
        srcMac:
        Protocol:
        Vlan:                                 Interface:
        Active Status: ENABLED                 Applied Status: APPLIED
        Priority: 10
        NextHop: 162.1.11.2

2)      Name: cnc-cluster1_192.168.0.0-192.168.0.255
        Action: ALLOW                           Hits: 674
        srcIP = 162.1.11.254
        destIP = 192.168.0.0-192.168.0.255
        srcMac:
        Protocol:
        Vlan:                                 Interface:
        Active Status: ENABLED                 Applied Status: APPLIED
        Priority: 20
        NextHop: 162.1.11.1

Done
```

ここでは、:

- PBRの数はKubernetesノードの数と同じです。この場合、2つのPBRが追加されます。
- PBRの`srcIP`は、トンネルネットワークでNetScalerノードコントローラーによって追加された`SNIP`です。`destIP`は、KubernetesノードのCNIオーバーレイサブネット範囲です。`NextHop`は、KubernetesノードのVXLANトンネルインターフェースのIPアドレスです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

NetScalerノードコントローラーは、静的ルートの代わりにPBRを追加します。VXLANとブリッジテーブルの残りの設定は同じままです。詳細については、NetScalerノードコントローラーの構成を参照してください。

MongoDB向け単一階層NetScaler® Ingressソリューション

MongoDBは、大量の非構造化データを処理および保存するように設計された、最も人気のあるNoSQLデータベースの1つです。クラウドネイティブアプリケーションは、KubernetesプラットフォームでNoSQLデータベースとしてMongoDBを広く使用しています。

Kubernetes環境では、アプリケーション展開の規模が大きいため、パフォーマンスの問題を特定してトラブルシューティングすることは課題です。MongoDBのようなデータベース展開では、高可用性と高性能の要件が満たされていることを確認するために、監視はデータベース管理の重要な要素です。

NetScalerは、NetScalerの高度な負荷分散およびパフォーマンス監視機能を使用して、Kubernetesプラットフォーム上のMongoDBデータベースの負荷分散と監視のためのIngressソリューションを提供します。MongoDB向けNetScaler Ingressソリューションは、MongoDBトランザクションへのより深い可視性を提供し、パフォーマンスの問題が発生したときに迅速に特定して対処するのに役立ちます。NetScaler Observability Exporterを使用して、MongoDBトランザクションをElasticsearchにエクスポートし、Kibanaダッシュボードを使用して視覚化することで、より深い洞察を得ることができます。

次の図は、NetScalerの単一階層展開を使用したMongoDB向けNetScaler Ingressソリューションについて説明しています。

このソリューションでは、NetScaler VPXはKubernetesクラスターの外部（Tier-1）に展開され、NetScaler Observability ExporterはKubernetesクラスターの内部に展開されます。

Tier-1 NetScaler VPXは、MongoDBクライアントからのトラフィック（南北）をKubernetesクラスター内のMongoDBクエリールーター（Mongos）にルーティングします。NetScaler Observability ExporterはKubernetesクラスターの内部に展開されます。

この展開の一部として、NetScaler VPX（Tier-1 Ingress）用のIngressリソースが作成されます。Tier-1 Ingressリソースは、NetScaler VPX上のMongoDBトラフィックの負荷分散を有効にするためのルールを定義し、Mongoのポートを指定します。MongoDBトラフィックがNetScaler VPX上の指定されたポートに到着すると、このトラフィックはIngressルールで言及されているMongo

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サービスインスタンスの1つにルーティングされます。MongoサービスはMongoDB管理者によって公開され、同じサービスインスタンスがIngressで指定されます。

NetScaler Observability Exporterインスタンスは、NetScaler VPXからのトランザクションを集約し、Elasticsearchサーバーにアップロードします。Kibanaダッシュボードを設定して、必要なデータ（たとえば、クエリ応答時間、最もクエリされたコレクション名）を視覚化し、それらを分析して意味のある洞察を得ることができます。挿入、更新、削除、検索、および応答操作のみが解析され、メトリックはNetScaler Observability Exporterに送信されます。

前提条件

MongoDB向けNetScaler Ingressソリューションを展開する前に、次の手順を完了する必要があります。

- クラウドまたはオンプレミスでKubernetesクラスターをセットアップする
- 展開モードをsharded replica setとして、KubernetesクラスターにMongoDBを展開します。MongoDBの他の展開モードはサポートされていません。
- Elasticsearchがインストールされ、構成されていることを確認してください。Elasticsearchを展開するには、elasticsearch.yamlファイルを使用します。
- トランザクションデータを視覚化するためにKibanaがインストールされていることを確認してください。Kibanaをデプロイするには、kibana.yamlファイルを使用します。
- Kubernetesクラスターの外部にNetScaler VPXインスタンスをデプロイします。NetScaler VPXのデプロイ方法については、NetScaler VPXインスタンスのデプロイを参照してください。

NetScaler VPXをデプロイした後、以下を実行します。

1. KubernetesクラスターのサブネットからIPアドレスをNetScalerのSNIPとして構成します。NetScalerでのSNIPの構成については、サブネットIPアドレス（SNIP）の構成を参照してください。
2. Kubernetesクラスターと同じサブネットであるSNIPの管理アクセスを有効にします。NetScaler Ingress ControllerがTier-1 NetScalerを構成できるように、SNIPはNetScaler Ingress Controller YAMLファイルで`NS_IP`変数として使用する必要があります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

SNIPを`NS_IP`として使用することは必須ではありません。NetScalerの管理IPアドレスがNetScaler Ingress Controllerから到達可能な場合、管理IPアドレスを`NS_IP`として使用できます。

3. NetScaler Ingress Controller専用のNetScalerシステムユーザーアカウントを作成します。NetScaler Ingress Controllerは、システムユーザーアカウントを使用してTier-1 NetScalerを自動的に構成します。
4. NetScaler VPXを構成して、DNSクエリをKubernetesクラスター内のCoreDNSポッドIPアドレスに転送します。

```
add dns nameServer &lt;core-dns-pod-ip-address>
```

たとえば、ポッドIPアドレスが192.244.0.2と192.244.0.3の場合、NetScaler VPXでネームサーバーを次のように構成します。

```
add dns nameServer 192.244.0.3

add dns nameServer 192.244.0.2
```

MongoDB向けNetScaler Ingressソリューションのデプロイ

MongoDB向けNetScaler Ingressソリューションをデプロイすると、Kubernetesクラスターに以下のコンポーネントがデプロイされます。

- NetScaler VPX用のスタンドアロンNetScaler Ingress Controller
- NetScaler VPX用のIngressリソース
- NetScaler 可観測性エクスポーター

MongoDB 用 NetScaler Ingress ソリューションを展開するには、次の手順を実行します。

1. NetScaler VPX のユーザー名とパスワードで Kubernetes シークレットを作成します。

```
kubectl create secret  generic nslogin --from-
literal=username='username' --from-literal=password='mypassword'
```

2. cic-configmap.yaml ファイルをダウンロードし、次のコマンドを使用して展開します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl create -f cic-configmap.yaml
```

3. 次の手順を使用して、NetScaler Ingress Controller をポッドとして展開します。

   a. NetScaler Ingress Controller マニフェストファイルをダウンロードします。次のコマンドを使用します。

   ```
   wget https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/deployment/dual-tier/manifest/tier-1-vpx-cic.yaml
   ```

   b. NetScaler Ingress Controller マニフェストファイルを編集し、次の環境変数の値を入力します。

環境変数	必須またはオプション	説明
NS_IP	必須	NetScaler アプライアンスの IP アドレス。詳細については、前提条件 を参照してください。
NS_USER と NS_PASSWORD	必須	Ingress デバイスとして使用される NetScaler VPX または MPX アプライアンスのユーザー名とパスワード。
EULA	必須	エンドユーザーライセンス契約。値を `Yes` として指定します。
LOGLEVEL	オプション	NetScaler Ingress Controllerによって生成されるログを制御するためのログレベル。デフォルトでは、値はDEBUGに設定されています。サポートされている値は、CRITICAL、ERROR、WARNING、INFO、DEBUGです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

環境変数	必須またはオプション	説明
NS_PROTOCOL および NS_PORT	オプション	NetScaler Ingress ControllerがNetScalerと通信するために使用する必要があるプロトコルとポートを定義します。デフォルトでは、NetScaler Ingress Controllerはポート443でHTTPSを使用します。ポート80でHTTPを使用することもできます。
イングレスクラス	オプション	複数のIngressロードバランサーが異なるIngressリソースの負荷分散に使用される場合。この環境変数を使用して、特定のIngressクラスに関連付けられたNetScalerを構成するようにNetScaler Ingress Controllerを指定できます。Ingressクラスの詳細については、Ingressクラスのサポートを参照してください。
NS_VIP	オプション	NetScaler Ingress Controllerは、この環境変数で提供されるIPアドレスを使用して、Ingressトラフィックを受信するNetScalerに仮想IPアドレスを構成します。

c. NetScaler Ingress Controller YAMLファイルで、以下の引数を指定または変更します。

```
args:

 - --configmap

    default/cic-configmap
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  - --ingress-classes

     tier-1-vpx
```

d. 以下のコマンドを使用して、更新されたNetScaler Ingress Controllerマニフェストファイルをデプロイします。

```
kubectl create -f tier-1-vpx-cic.yaml
```

4. tier-1-vpx-ingress.yamlファイルを使用して、Tier-1 NetScalerのIngressオブジェクトを作成します。

```
kubectl apply -f tier-1-vpx-ingress.yaml
```

`tier-1-vpx-ingress.yaml`ファイルの内容は次のとおりです。このIngressリソースで指定されたルールに従い、NetScaler Ingress ControllerはNetScaler VPXを構成して、ポート27017でMongoDBトラフィックをリッスンします。この例に示すように、NetScaler VPXがトラフィックをルーティングできるように、MongoDbクエリールーター用に作成したサービス（例:`serviceName: mongodb-mongos`）を指定する必要があります。ここで、`mongodb-mongos`はMongoDBクエリールーターのサービスです。

```
apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  annotations:

    ingress.citrix.com/analyticsprofile: '{"tcpinsight":
{"tcpBurstReporting":"DISABLED"}}'

    ingress.citrix.com/insecure-port: "27017"

    ingress.citrix.com/insecure-service-type: mongo

    ingress.citrix.com/insecure-termination: allow

  name: vpx-ingress
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
spec:
  ingressClassName: tier-1-vpx
  defaultBackend:
      service:
        name: mongodb-mongos
        port:
          number: 27017
```

5. coe-es-mongo.yamlファイルを使用して、ElasticsearchをエンドポイントとしてNetScaler Observability Exporterをデプロイします。

```
kubectl apply -f coe-es-mongo.yaml
```

注:

ELKServer環境変数に、IPアドレスまたはDNS名に基づいて、ポート情報とともにElasticsearchサーバーの詳細を設定する必要があります。

以下はConfigMapファイルのサンプルです。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: coe-config-es
data:
  lstreamd_default.conf: |
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    {

        "Endpoints": {

            "ES": {

                "ServerUrl":
"elasticsearch.default.svc.cluster.local:9200",

                "IndexPrefix":"adc_coe",

                "IndexInterval": "daily",

                "RecordType": {

                    "HTTP": "all",

                    "TCP": "all",

                    "SWG": "all",

                    "VPN": "all",

                    "NGS": "all",

                    "ICA": "all",

                    "APPFW": "none",

                    "BOT": "none",

                    "VIDEOOPT": "none",

                    "BURST_CQA": "none",

                    "SLA": "none",

                    "MONGO": "all"

                },

                "ProcessAlways": "no",
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
                "ProcessYieldTimeOut": "500",

                "MaxConnections": "512",

                "ElkMaxSendBuffersPerSec": "64",

                "JsonFileDump": "no"

            }

        }

    }
```

NetScaler Ingressソリューションのデプロイを検証する

以下の例に示すコマンドを使用して、すべてのアプリケーションがデプロイされていることを確認し、すべてのサービスとポートを一覧表示できます。

以下の例に示す`kubectl get ingress`コマンドを使用して、デプロイされたIngressオブジェクトに関する情報を取得できます。

```
    # kubectl get ingress


     NAME               HOSTS    ADDRESS    PORTS    AGE

     vpx-ingress        *                   80       22d
```

MongoDBトラフィックの可観測性を検証する

このトピックでは、NetScaler Ingressソリューションを使用してMongoDBトランザクションの可視性を得る方法について説明し、Kibanaダッシュボードを使用してデータベースのパフォーマンス統計を視覚化します。

このトピックの手順を実行する前に、以下を確認してください。

- Kubernetesクラスターにシャードレプリカセットとして MongoDBをデプロイ済みであること。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- MongoDB用のNetScaler Ingressソリューションがデプロイされていること。
- MongoDBにトラフィックを送信するためのMongoDBクライアントアプリケーションがインストールされていること。
- 可視化のためにKibanaがインストールされていること。

MongoDBトラフィックの可観測性を検証するには、以下の手順を実行します。

1. MongoDB用のクライアントアプリケーションを、Tier-1 NetScaler VPXの仮想IPアドレスを指すように構成します。

   例:

   ```
   mongodb://&lt;vip-of-vpx>:27017/
   ```

2. MongoDBクライアントアプリケーションを使用して、複数のリクエスト（挿入、更新、削除など）をMongoDBデータベースに送信します。トランザクションはElasticsearchサーバーにアップロードされます。
3. MongoDBトランザクションを可視化するためにKibanaダッシュボードを設定します。以下のサンプルKibanaダッシュボードを使用できます。

MongoDB用Kibanaダッシュボード(/en-us/netscaler-k8s-ingress-controller/media/mongodb-kibana.png)

このダッシュボードでは、さまざまな種類のクエリやクエリ応答時間を含む、MongoDBデプロイのパフォーマンス統計を確認できます。このデータを分析することで、トランザクションの遅延などの異常を発見し、即座に対処することができます。

テレメトリデータをPrometheusにエクスポートする

Kubernetesデプロイの場合、Prometheusサーバーが同じKubernetesクラスターにデプロイされている場合、PrometheusがNetScaler Observability Exporterをスクレイプターゲットとして自動的に追加できるようにアノテーションを構成できます。

以下は、これらのアノテーションを含むNetScaler Observability Exporter YAMLファイル（coe-es-mongodb.yaml）の抜粋です。

```
template:

  metadata:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    name: coe-es
    labels:
      app: coe-es
    annotations:
      prometheus.io/scrape: "true"
      prometheus.io/port: "5563"
```

または、Prometheusサーバー構成ファイルでNetScaler Observability Exporterをスクレイプターゲットとして手動で追加することもできます。

また、次の YAML ファイルに示すように、`cic-configmap.yaml` ファイルで Prometheus のメトリクスが有効になっていることを確認してください。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cic-configmap
  namespace: default
data:
   NS_ANALYTICS_CONFIG: |
     distributed_tracing:
       enable: 'false'
       samplingrate: 0
     endpoint:
       server: 'coe-es.default.svc.cluster.local'
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    timeseries:
      port: 5563
      metrics:
        enable: 'true'
        mode: 'prometheus'
      auditlogs:
        enable: 'false'
      events:
        enable: 'false'
    transactions:
      enable: 'true'
      port: 5557
```

この YAML ファイルでは、次の構成により Prometheus のメトリクスが有効になります。

```
metrics:
   enable: 'true'
   mode: 'prometheus'
```

KubernetesベースのアプリケーションNetScaler VPXとAzure Pipelinesを使用したカナリアおよびブルー/グリーンデプロイメント

このトピックでは、NetScaler VPXとAzure Pipelinesを使用してKubernetesアプリケーションのカナリアおよびブルー/グリーンデプロイメントを実現する方法について説明します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

KubernetesベースのアプリケーションNetScaler VPXとAzure Pipelinesを使用したカナリアデプロイメント

カナリアは、アプリケーションの新しいバージョンを小規模かつ段階的な増分ステップでデプロイするデプロイ戦略です。カナリアの考え方は、まず新しい変更を少数のユーザーにデプロイし、新しいデプロイを拒否するか昇格させるかを決定し、その後残りのユーザーに変更を展開することです。この戦略により、本番環境にアプリケーションの新しいバージョンをデプロイする際のリスクが制限されます。

Azure pipelines は、Azure DevOps が提供するクラウドサービスで、ビルドの自動実行、テストの実行、さまざまな開発環境および本番環境へのコードのデプロイを可能にします。

このセクションでは、NetScaler VPXとNetScaler Ingress ControllerをAzure Pipelinesと組み合わせて使用し、Kubernetesベースのアプリケーションのカナリアデプロイメントを実現する方法について説明します。

カナリアデプロイメントの利点

- アプリケーションのカナリアバージョンは、新しいコードに存在する可能性のある潜在的な問題やデプロイメントの問題に対する早期警告として機能します。
- カナリアバージョンは、スモークテストやA/Bテストに使用できます。
- カナリアは、簡単なロールバックとゼロダウンタイムアップグレードを提供します。
- 複数のバージョンのアプリケーションを同時に実行できます。

このソリューションでは、NetScaler VPXがAzureプラットフォームにデプロイされ、アプリケーションのロードバランシングを可能にし、NetScaler VPXを使用してカナリアデプロイメントを実現します。Microsoft AzureへのNetScalerのデプロイ方法の詳細については、NetScalerドキュメントのリンクを参照してください。

NetScalerを使用したカナリアデプロイメント

ルールベースのカナリアデプロイメントであるIngressアノテーションを使用して、NetScalerでカナリアデプロイメントを実現できます。このアプローチでは、ルールベースのカナリアデプロイメント戦略に基づいてアプリケーションリクエストを処理する必要があることを示すために、特定のIngressオブジェクトを追加で定義する必要があります。Citrixソリューションでは、Ingressレベルでのカナリアベースのトラフィックルーティングは、次のようにさまざまなルールセットを定義することで実現できます。

- 重みに基づいてカナリアルールを適用する
- HTTPリクエストヘッダーに基づいてカナリールールを適用する

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- HTTPヘッダー値に基づいてカナリールールを適用する

詳細については、「Ingressアノテーションを使用した簡素化されたカナリアデプロイ」(/ja-jp/netscaler-k8s-ingress-controller/canary/canary.html#simplified-canary-deployment-using-ingress-annotations)を参照してください。

NetScaler VPXとAzureパイプラインを使用したカナリアデプロイ

Citrixは、Kubernetesベースのアプリケーション向けに、NetScaler VPXとNetScaler Ingress ControllerをAzureパイプラインと組み合わせて使用するカナリアデプロイのソリューションを提案しています。

このソリューションには、3つの構成ディレクトリがあります。

- `kubernetes_configs`
- `deployment_configs`
- `pipeline_configs`

kubernetes_configs

このディレクトリには、バージョンベースのアプリケーション固有のデプロイYAMLファイルと、カナリアデプロイを実現するためのNetScaler構成をプッシュする役割を担うNetScaler Ingress Controllerをデプロイするための Helmベースの構成ファイルが含まれています。

注:

最新のHelmチャートは、NetScaler Ingress Controller Helm charts repositoryからダウンロードし、 `cic_helm` ディレクトリに配置できます。

deployment_configs

このディレクトリには、カナリアデプロイ中にデプロイまたは停止されるアプリケーションの特定のバージョンで利用可能なYAMLファイルのパスを指定する `setup_config` と `teardown_config` のJSONファイルが含まれています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

pipeline_configs

このディレクトリには、ユーザー設定を読み取り、アプリケーションの新しいバージョンを導入するか、アプリケーションのバージョンを停止するためのユーザーのリクエストに基づいてパイプラインをトリガーするAzureパイプラインスクリプトとPythonスクリプトが含まれています。アプリケーションのイングレスYAMLにおけるトラフィックウェイトの割合の変更は、利用可能なアプリケーションバージョン間でトラフィックを切り替えるようにパイプラインをトリガーします。

3つの設定ファイルがすべて揃っている状態で、GitHubの`deployment_configs`および`kubernetes_configs`ディレクトリ内のファイルへの更新は、Azureのパイプラインをトリガーします。

アプリケーションのイングレスYAMLで、`ingress.citrix.com/canary-weight`アノテーションを使用してトラフィック分割の割合を調整できます。

Azureパイプラインでカナリアにサンプルアプリケーションをデプロイする

このトピックでは、NetScalerとAzureパイプラインを使用してカナリアモードでサンプルアプリケーションをデプロイする方法を説明します。

前提条件

以下を確認してください。

- NetScaler VPXがAzureプラットフォームにすでにデプロイされており、サンプルアプリケーションで使用できる状態になっていること。
- Azureパイプライン用にKubernetes service connectionが設定されたAKSクラスター。

次の手順を実行します。

1. GitHubリポジトリをクローンし、`cd/canary-azure-devops`ディレクトリに移動します。
2. `kubernetes_configs`ディレクトリ内のバージョン管理されたフォルダー`v1`に、アプリケーションデプロイメント固有のYAML（イングレスファイルを含む）を配置します。
3. 既存のYAMLファイル`deploy_cic.yaml`、`deploy.yaml`、および`teardown.yaml`を使用して、NetScaler Ingress Controllerのデプロイ、およびアプリケーションのデプロイと破棄を行う3つのAzureパイプラインを作成します。パイプラインの作成については、Azure pipelinesを参照してください。
4. パイプラインYAMLのサブスクリプション、エージェントプール、サービス接続、およびNetScalerの詳細を更新します。
5. パイプラインを保存します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

6. `deploy_config.json`のパスを、アプリケーションYAMLが配置されているディレクトリを指定するパスで更新します。

```
{


 "K8S_CONFIG_PATH" : "cd/canary-azure-devops/kubernetes_configs/v1"


}
```

7. Gitを使用して`deploy_config.json`ファイルとv1ディレクトリをコミットし、パイプラインをトリガーしてアプリケーションのv1バージョンをデプロイします。
8. NetScalerを介してアプリケーションにアクセスします。
9. `kubernetes_configs`の下に`v2`ディレクトリを作成して、アプリケーションのv2バージョンを導入します。このバージョンのイングレスに、トラフィック分割のために適切な重みが設定されたカナリアアノテーションが指定されていることを確認してください。
10. `deploy_config.json`をv2ディレクトリを指定するパスで更新して、アプリケーションのv2バージョンをデプロイします。これで、イングレスアノテーションで設定されたカナリアの重みに基づいて、v1とv2のバージョン間でトラフィックが分割されます（例: `ingress.citrix.com/canary-weight: "40"`）。
11. 新しいバージョンがすべてのトラフィックを処理できるようになるまで、イングレスアノテーションのトラフィックの重みを段階的に増やし続けます。

KubernetesベースのアプリケーションNetScaler VPXとAzureパイプラインを使用したブルー/グリーンデプロイ

ブルー/グリーンデプロイは、ブルーとグリーンと呼ばれる2つの同一の運用環境を実行することで、ダウンタイムとリスクを削減する手法です。常に、いずれかの環境のみが稼働しており、すべての運用トラフィックを処理します。ブルー/グリーン方式の基本は、2つの別個で同一の環境を並行してデプロイすることです。両方の環境にアプリケーションをデプロイすることは、ジョブとタスクを使用することで完全に自動化できます。このアプローチでは、アプリケーションのすべてのリソースの重複が強制されます。ただし、さまざまな継続的デプロイツールでブルー/グリーンデプロイを実行する方法は多数あります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler VPXとAzureパイプラインを使用すると、トラフィックの重みをゼロまたは100に調整することで、同じカナリアベースのソリューションを使用してブルー/グリーンデプロイを実現できます。

外部サービス向けのトラフィック管理

アプリケーションの利用可能なすべてのサービスが、単一のKubernetesクラスターに完全にデプロイされていない場合があります。1つのクラスター外のサービスに依存するアプリケーションがあるかもしれません。この場合、マイクロサービスはドメイン名を解決するためにExternalNameサービスを定義する必要があります。ただし、このアプローチでは、トラフィック管理、ポリシー適用、フェイルオーバー管理などの機能を利用できません。代替策として、NetScalerを構成してドメイン名を解決し、NetScalerの機能を活用できます。

外部サービスに到達するようにNetScalerを構成する

NetScaler Ingress Controllerを使用して、NetScalerをドメイン名リゾルバーとして構成できます。NetScalerをドメイン名リゾルバーとして構成する場合、以下を解決する必要があります。

- マイクロサービスからNetScalerへの到達可能性
- 外部サービスに到達するためのNetScalerでのドメイン名解決

KubernetesクラスターからNetScalerへの到達可能性のためのサービスを構成する

マイクロサービスからNetScalerに到達するには、NetScalerサービスに解決されるヘッドレスサービスを定義する必要があります。これにより、マイクロサービスとNetScaler間の接続が確立されます。

```
apiversion: v1

kind: Service

metadata:

  name: external-svc

spec:

  selector:

    app: cpx

  ports:
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    - protocol: TCP

      port: 80
```

NetScaler Ingress Controllerを使用してNetScalerをドメイン名リゾルバーとして構成する

NetScaler Ingress Controllerを介してNetScalerを構成し、イングレスアノテーション `ingress.citrix.com/external-service` を使用してドメインベースのサービスグループを作成できます。`ingress.citrix.com/external-service` の値は、対応するドメイン名を持つ外部名サービスのリストです。NetScaler VPXの場合、ネームサーバーはConfigMapを使用してNetScaler上に構成されます。

注:

ConfigMapは、NetScaler VPXの場合にのみNetScaler上のネームサーバーを構成するために使用されます。NetScaler CPXの場合、CoreDNSは名前解決要求をアップストリームDNSサーバーに転送します。

NetScaler CPXを使用したトラフィック管理

次の図は、外部サービスに到達するためのNetScaler CPXのデプロイメントについて説明しています。NetScaler CPXでDNSを構成するために外部サービスアノテーションが指定されたIngressがデプロイされます。

注: ConfigMapは、NetScaler VPX上のネームサーバーを構成するために使用されます。

このデプロイでは:

1. マイクロサービスは、www.externalsvc.com のDNSクエリを送信し、それはNetScaler CPXサービスに解決されます。
2. NetScaler CPXは www.externalsvc.com を解決し、外部サービスに到達します。

NetScaler CPXを構成して外部サービスをロードバランシングする手順は次のとおりです。

1. NetScalerに到達するためのヘッドレスサービスを定義します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
apiVersion: v1

kind: Service

metadata:

  name: external-svc

spec:

  selector:

    app: cpx

  ports:

    - protocol: TCP

      port: 80
```

2. イングレスを定義し、dbs-ingress.yaml ファイルで指定されているように external-service アノテーションを指定します。このアノテーションを指定すると、NetScaler Ingress ControllerはNetScaler上にDNSサーバーを作成し、そのサーバーを対応するサービスグループにバインドします。

```
annotations:

  ingress.citrix.com/external-service: '{"external-svc": {"domain":
"www.externalsvc.com"}}'
```

3. ConfigMapを使用して、NetScaler上のDNSサーバーのIPアドレスを追加します。

注:

この手順はNetScaler VPXにのみ適用されます。

```
apiVersion: v1

kind: ConfigMap
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
metadata:

  name: nameserver-cmap

  namespace: default

data:

  NS_DNS_NAMESERVER: '[]'
```

名前空間をまたがるExternalNameサービスのサポート

名前空間は、Kubernetesクラスター内のリソースを分離するために使用されます。時には、異なる名前空間のサービスが、別の名前空間にあるサービスにアクセスする必要がある場合があります。そのようなシナリオでは、Kubernetesが提供する`ExternalName`サービスを使用できます。`ExternalName`サービスは、セレクターを持たず、代わりにDNS名を使用する特別なサービスです。

サービス定義では、`externalName`フィールドは名前空間と、その名前空間でアクセスしようとしているサービスを指す必要があります。Citrix ingress controllerは、クラスター内のサービスにアクセスする必要がある場合に、`ExternalName`タイプのサービスをサポートします。

`ExternalName`サービスを作成する際には、以下の基準を満たす必要があります。

- サービス定義の`externalName`フィールドは、以下の形式に従う必要があります。`svc:<name-of-the-service>.<namespace-of-the-service>.svc.cluster.local`
- `ExternalName`サービス内のポート番号は、ターゲットとなるサービスのポート番号と完全に一致する必要があります。

注:

アプリケーションのサービスがKubernetesクラスターの外部にあり、`ExternalName`サービスを作成した場合、外部サービス向けトラフィック管理機能を使用してドメイン名を解決できます。

ExternalNameサービスの例

この例では、デフォルトの名前空間で`mysql`サービスが実行されており、`namespace1`名前空間から`mysql`サービスにアクセスするために、サンプル`ExternalName`サービスが作成されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

以下は、デフォルトの名前空間で実行されているMySQLサービスのサンプルサービス定義です。

```
apiVersion: v1

kind: Service

metadata:

  name: mysql

  namespace: default

spec:

  clusterIP: None

  ports:

  - port: 3306

    protocol: TCP

    targetPort: 3306

  selector:

    app: mysql

  type: ClusterIP
```

以下は、`namespace1`名前空間から`mysql`サービスにアクセスするためのサンプル`ExternalName`サービス定義です。

```
kind: Service

apiVersion: v1

metadata:

  name: dbservice
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
  namespace: namespace1

spec:

  type: ExternalName

  externalName: mysql.default.svc.cluster.local

  ports:

  - port: 3306

    protocol: TCP

    targetPort: 3306
```

この例では、サービスは`externalName: mysql.default.svc.cluster.local`フィールドで指定されているように、`mysql`がデプロイされている名前空間を指しています。ここで、`mysql`はサービス名、`default`は名前空間です。ポート名も`mysql`サービスと同じであることがわかります。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

アップグレードとトラブルシューティング

アップグレードとトラブルシューティングには、次のトピックがあります。

- アップグレード中のNetScaler CPX構成の永続性](/ja-jp/netscaler-k8s-ingress-controller/accelerate-cpx-upgrade-process.html)
- NetScaler Ingress Controllerのトラブルシューティング](/ja-jp/netscaler-k8s-ingress-controller/troubleshooting/troubleshooting-cic.html)
- NetScalerノードコントローラーのトラブルシューティング](/ja-jp/netscaler-k8s-ingress-controller/troubleshooting/troubleshooting-node-controller.html)
- NetScaler Ingress Controllerのアップグレード](/ja-jp/netscaler-k8s-ingress-controller/upgrade.html)

アップグレード中のNetScaler CPX構成の永続性

Helmアップグレード、ローリングアップデート、またはその他のPodの置き換え中にNetScaler® CPX Podが再起動すると、Ingressおよびサービスリソースを通じて蓄積された実行中の構成は永続化されません。NetScaler Ingress Controller (NSIC) は、NITRO APIを介してすべてのリソースを最初から再プッシュする必要があります。大規模なIngressデプロイメントでは、NetScaler CPXがトラフィックを処理する準備が整うまでに、この調整に時間がかかる場合があります。

NetScaler CPXの構成永続化は、Podシャットダウン時にNetScaler CPXの実行中の構成をKubernetes ConfigMapに保存することで、この遅延を解消します。置き換えPodが起動すると、実行中の構成は自動的に復元されます。新しいPodはほとんどの構成が事前にロードされた状態で起動するため、新しいPodが稼働するまでの時間が短縮されます。この機能は、NetScaler Ingress Controllerバージョン4.1.x以降でサポートされています。

利点

- Podの再起動またはアップグレード後のほぼ瞬時の構成復元。
- 手動での介入は不要 — バックアップと復元は完全に自動化されています。
- 既存のHelmチャートワークフローと互換性があります (`helm upgrade`)。
- 自動オーバーフロー圧縮により、大規模な構成を処理します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- NetScaler CPX自体に変更は不要 — ネイティブのNetScaler CPX起動構成メカニズムを使用します。

構成の永続化を有効にする

Helmのインストール

インストールまたはアップグレード中に`cpxConfigPersistenceEnabled=true`を設定して、この機能を有効にします。

```
helm install my-release netscaler/netscaler-cpx-with-ingress-controller --
set cpxConfigPersistenceEnabled=true,license.accept=yes
```

または

既存のリリースで`cpxConfigPersistenceEnabled=true`を設定して、この機能を有効にします。

```
helm upgrade my-release netscaler/netscaler-cpx-with-ingress-controller \

  --set cpxConfigPersistenceEnabled=true,license.accept=yes
```

NetScaler CPXがバージョン13.1の場合、`enableStartupProbe`を`false`に設定します。

NetScaler Ingress Controller のトラブルシューティング

NetScaler Ingress Controller は、以下の方法でデバッグできます。まず、イベントベースのデバッグ方法を使用し、次にログベースのデバッグ方法を使用します。高度なデバッグには、NetScaler kubectl プラグインと NSIC 診断ツールを使用してください。

- イベントベースのデバッグ
- ログベースのデバッグ
- NetScaler® Kubectl プラグイン
- NSIC 診断ツール

イベントベースのデバッグ

Kubernetes のイベントは、他の Kubernetes エンティティの運用フローに関する洞察を提供するエンティティです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScaler Ingress Controller のイベントベースのデバッグは、Pod レベルで有効になっています。NetScaler Ingress Controller のイベントを表示するには、次のコマンドを使用します。

```
kubectl describe pods <citrix-k8s-ingress-controller pod name> -n
<namespace of pod>
```

イベントは`Events`セクションで確認できます。

以下の例では、NetScaler は意図的に到達不能にされており、同じ情報が`Events`セクションで確認できます。

```
    kubectl describe pods cic-vpx-functionaltest -n functionaltest



    Name:         cic-vpx-functionaltest

    Namespace:    functionaltest



    Events:

    Type     Reason     Age   From                                Message

    ----     ------     ----  ----                                -------

    Normal   Pulled     33m   kubelet, rak-asp4-node2
Container image "citrix-ingress-controller:latest" already present on
machine

    Normal   Created    33m   kubelet, rak-asp4-node2             Created
container cic-vpx-functionaltest

    Normal   Started    33m   kubelet, rak-asp4-node2             Started
container cic-vpx-functionaltest

    Normal   Scheduled  33m   default-scheduler
Successfully assigned functionaltest/cic-vpx-functionaltest to rak-asp4-
node2
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    Normal   Created    33m   CIC ENGINE, cic-vpx-functionaltest
 CONNECTED: NetScaler:&lt;NetScaler IP>:80

    Normal   Created    33m   CIC ENGINE, cic-vpx-functionaltest  SUCCESS:
Test LB Vserver Creation on NetScaler:

    Normal   Created    33m   CIC ENGINE, cic-vpx-functionaltest  SUCCESS:
ENABLING INIT features on NetScaler:

    Normal   Created    33m   CIC ENGINE, cic-vpx-functionaltest  SUCCESS:
GET Default VIP from NetScaler:

    Warning  Created    17s   CIC ENGINE, cic-vpx-functionaltest
 UNREACHABLE: NetScaler: Check Connectivity::&lt;NetScaler IP>:80
```

さらにデバッグするには、NetScaler Ingress Controller Pod のログを確認してください。

ログベースのデバッグ

ConfigMap 機能を使用して、NetScaler Ingress Controller のログレベルをランタイムで変更できます。ランタイム中にログレベルを変更する方法については、ConfigMapドキュメントを参照してください。

NetScaler Ingress Controller のログを確認するには、次のコマンドを使用します。

```
kubectl logs <citrix-k8s-ingress-controller> -n namespace
```

次の表に、一般的な問題とその回避策をいくつか示します。

問題	ログ	回避策
NetScaler インスタンスに到達できません	**2019-01-10 05:05:27,250 - ERROR - [nitrointerface.py:login_logout:94] (MainThread) 例外: HTTPConnectionPool(host='10.106.76.200', port=80): URLでの最大再試行回数を超過しました: /nitro/v1/config/login (原因: NewConnectionError('<urlli**	NetScaler が稼働しており、NSIP アドレスに ping できることを確認してください。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

問題	ログ	回避策
	b3.connection.HTTPConnection object at 0x7f4d45bd63d0>: 新しい接続の確立に失敗しました: [Errno 113] ホストへのルートがありません',))	
ユーザー名またはパスワードが間違っています	**2019-01-10 05:03:05,958 - エラー - [nitrointerface.py:login_logout:90] (メインスレッド) Nitro 例外::ログイン_ログアウト::エラーコード=354,メッセージ=無効なユーザー名またはパスワード**	
SNIP で管理アクセスが有効になっていません	**2019-01-10 05:43:03,418 - ERROR - [nitrointerface.py:login_logout:94] (MainThread) 例外: HTTPConnectionPool(host='10.106.76.242', port=80): URLでの最大再試行回数を超過しました: /nitro/v1/config/login (原因: NewConnectionError('<urllib3.connection.HTTPConnection object at 0x7f302a8cfad0>: 新しい接続の確立に失敗しました: [Errno 110] 接続がタイムアウトしました',))**	NetScaler で管理アクセスが有効になっていること (NetScaler VPX 高可用性の場合)、および管理アクセスが有効になっている IP アドレス **NSIP** が設定されていることを確認してください。
アノテーションの解析中にエラーが発生しました	**2019-01-10 05:16:10,611 - ERROR - [kubernetes.py:set_annotations_to_csapp:1040] (MainThread) set_annotations_to_csapp: エラーメッセージ=No JSON object could be**	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

問題	ログ	回避策
	decodedInvalid Annotation $service_weights 修正して適用してください $ {"frontend":, "catalog":95}	
NITRO アクセスのポートが間違っています	**2019-01-10 05:18:53,964 - ERROR - [nitrointerface.py:login_logout:94] (MainThread) 例外: HTTPConnectionPool(host='10.106.76.242', port=34438): URL: /nitro/v1/config/login で最大再試行回数を超過しました (NewConnectionError('<urllib3.connection.HTTPConnection object at 0x7fc592cb8b10>: 新しい接続の確立に失敗しました: [Errno 111] 接続が拒否されました',))**	NITROアクセスに正しいポートが指定されているか確認してください。デフォルトでは、NetScaler Ingress Controller は通信にポート **80** を使用します。
Ingressクラスが間違っています	**2019-01-10 05:27:27,149 - INFO - [kubernetes.py:get_all_ingresses:1329] (MainThread) ingressオブジェクト web-ingress.default のIngressクラスがサポートされていません**	Ingressファイルが、NetScaler Ingress Controllerが監視するIngressクラスに属していることを確認してください。
Kubernetes APIに到達できません	**2019-01-10 05:32:09,729 - ERROR - [kubernetes.py:_get:222] (Thread-1) /services の呼び出し中にエラーが発生しました: HTTPSConnectionPool(host='10.106.76.237', port=6443): URL: /api/v1/services で最大再試行回数**	kubernetes_url が正しいか確認してください。コマンド `kubectl cluster-info` を使用してURL情報を取得します。Kubernetesメインノードが `https://kubernetes_master_address:6443` で実行されており、Kubernetes APIサーバーポッ

© 1997–2026 Citrix Systems, Inc. All rights reserved.

問題	ログ	回避策
	を超過しました (NewConnectionError('<urllib3.connection.VerifiedHTTPSConnection object at 0x7fb3013e7dd0>: 新しい接続の確立に失敗しました: [Errno 111] 接続が拒否されました',))	ドが稼働していることを確認してください。
YAMLファイルに指定されたサービスポートが正しくありません	NA	Ingress YAMLファイルに正しいポートの詳細を指定し、Ingress YAMLを再適用して問題を解決してください。
ロードバランシング仮想サーバーとサービスグループは作成されていますが、ダウンしています	NA	YAMLファイルで使用されているサービス名とポートを確認してください。NetScaler VPXの場合、NetScaler Ingress Controller を起動する際に `--feature-node-watch` が `true` に設定されていることを確認してください。
NetScaler VPXのコンテンツスイッチング (CS) 仮想サーバーが作成されません。	該当なし	NetScaler VPX の Ingress YAML ファイルで、アノテーション `ingress.citrix.com/frontend-ip` を使用します。
Ingress YAML ファイルの TLS セクションで不適切なシークレットが提供されました	**2019-01-10 09:30:50,673 - INFO - [kubernetes.py:_get:231] (MainThread) リソースが見つかりません: /secrets/default-secret12345 namespace default**	YAML ファイルの値を修正し、YAML を再適用してこの問題を解決します。
	2019-01-10 09:30:50,673 - INFO -	

© 1997–2026 Citrix Systems, Inc. All rights reserved.

問題	ログ	回避策
	[kubernetes.py:get_secret: 1712] (MainThread) アプリケーション default-secret12345.default のシークレットの取得に失敗しました	
`feature-node-watch` 引数が指定されていますが、NetScaler VPX に静的ルートが追加されていません	**エラー - [nitrointerface.py:add_ns_route:4495] (MainThread) Nitro Exception::add_ns_route::errorcode=604,message= ゲートウェイに直接到達できません**	このエラーは、`feature-node-watch` が有効になっており、NetScaler VPX と Kubernetes クラスターが同じネットワークにない場合に発生します。NetScaler Ingress Controller YAML ファイルから `- --feature-node-watch` 引数を削除する必要があります。NetScaler VPX と Kubernetes クラスターが異なるネットワークにある場合、静的ルートは機能しません。NetScaler VPX とクラスターノード間のトンネルを作成するには、ノードコントローラーを使用します。
CRD ステータスが更新されていません	**ERROR - [crdinfrautils.py:update_crd_status:42] (MainThread) negrwaddmuloccmod の CRD ステータス更新中に例外が発生しました: 403 Client Error: Forbidden for url: `https://10.96.0.1:443/apis/citrix.com/v1/namespaces/default/rewritepolicies/negrwaddmuloccmod/status`**	RBAC で CRD ステータスをプッシュする権限が提供されていることを確認します。権限は次の YAML と同様である必要があります。 • APIグループ: ["citrix.com"] リソース: ["rewritepolicies/status", "canarycrds/status", "authpolicies/status", "ratelimits/status", "listeners/status",

© 1997–2026 Citrix Systems, Inc. All rights reserved.

問題	ログ	回避策
		"httproutes/status", "wafs/status"]
NetScaler Ingress Controller イベントが更新されません	**ERROR - [clienthelper.py:post:94] (MainThread) APIサーバーへの/eventsリクエストが禁止されています**	NetScaler Ingress Controller Podイベントを更新する権限がRBACルールで提供されていることを確認してください。
		• APIグループ: [""] リソース: ["events"] 動詞: ["create"]
Rewrite-responder ポリシーが追加されていません	**エラー - [config_dispatcher.py:__dispatch_config_pack:324] (Dispatcher) Status: 104, ErrorCode: 3081, 理由: Nitro例外: 式の構文エラー [D(10, 20).\^RE_SELECT(, Offset 15] <**	このようなエラーは、rewrite-responder CRD内の不正な式が原因です。式を修正し、CRDを再適用してください。
	ERROR - [config_dispatcher.py:__dispatch_config_pack:324] (Dispatcher) Status: 104, ErrorCode: 3098, Reason: Nitro Exception: 無効な式のデータ型 [ent.ip.src\^, Offset 13]	
CRDの適用に失敗しました。NetScaler Ingress Controllerは、CRDを構成のセットに変換し、指定されたCRDに従ってNetScalerを目的の状態に構成します。構成が失敗した場合、CRDインスタンスは	**2020-07-13 08:49:07,620 - ERROR - [config_dispatcher.py:__dispatch_config_pack:256] (Dispatcher) ConfigPack 'default.k8service.kuard-service.add_spec' からの構成**	ログには、NITRO コマンドが失敗したことが示されています。同じログがNetScalerにも表示されます。NetScaler `ns.log` を確認し、`grep` コマンドを使用してエラー文字列を検索して、CRD の適用中に失敗し

© 1997–2026 Citrix Systems, Inc. All rights reserved.

問題	ログ	回避策
NetScalerに適用されない可能性があります。	**ADD_sslprofile_k8s_crd_k8service_kuard-service_default_80_tcp_backend_{name:k8s_crd_k8service_kuard-service_default_80_tcp_backend sslprofiletype:BackEnd tls12:enabled } の実行に失敗しました**	た NetScaler コマンドを特定します。CRD を削除して、再度追加してみてください。
	2020-07-13 08:49:07,620 - ERROR - [config_dispatcher.py:__dispatch_config_pack:257] (Dispatcher) Status: 104, ErrorCode: 1074, Reason: Nitro Exception: 無効な値 [sslProfileType, value differs from existing entity and it cant be updated.]	
	2020-07-13 08:49:07,620 - INFO - [config_dispatcher.py:__dispatch_config_pack:263] (Dispatcher) ConfigPack 'default.k8service.kuard-service.add_spec' の処理に失敗しました	

NetScaler Kubernetes kubectl プラグイン

NetScaler は、NetScaler Ingress Controller のデプロイを検査し、トラブルシューティング操作を実行するための kubectl プラグインを提供します。このプラグインで利用可能なサブコマンドを使用して、トラブルシューティング操作を実行できます。

注:

このプラグインは、NSIC バージョン 1.32.7 以降でサポートされています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

curl を使用したインストール

`kubectl` プラグインは、NetScaler Modern Apps tool kit repository から curl を使用して次のようにダウンロードすることでインストールできます。

Linux の場合:

```
curl -LO https://github.com/netscaler/modern-apps-toolkit/releases/
download/v1.0.0-netscaler-plugin/netscaler-plugin_v1.0.0-netscaler-
plugin_Linux_x86_64.tar.gz

gunzip netscaler-plugin_v1.0.0-netscaler-plugin_Linux_x86_64.tar.gz

tar -xvf netscaler-plugin_v1.0.0-netscaler-plugin_Linux_x86_64.tar

chmod +x kubectl-netscaler

sudo mv kubectl-netscaler /usr/local/bin/kubectl-netscaler
```

Macの場合:

```
curl -s -L https://github.com/netscaler/modern-apps-toolkit/releases/
download/v1.0.0-netscaler-plugin/netscaler-plugin_v1.0.0-netscaler-
plugin_Darwin_x86_64.tar.gz | tar xvz -

chmod +x kubectl-netscaler

sudo mv kubectl-netscaler /usr/local/bin/kubectl-netscaler
```

注:

Macの場合、開発者アプリを許可を有効にする必要があります。

Windowsの場合:

```
curl.exe -LO https://github.com/netscaler/modern-apps-toolkit/releases/
download/v1.0.0-netscaler-plugin/netscaler-plugin_v1.0.0-netscaler-
plugin_Windows_x86_64.zip | tar xvz
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

注:

Windowsの場合、`$PATH`変数を、`kubectl-netscaler.exe`ファイルが展開されているディレクトリに設定する必要があります。

Krewを使用したインストール

Krewは、お使いのマシンでkubectlプラグインを見つけてインストールするのに役立ちます。Krewクイックスタートガイドに従ってKrewをインストールおよびセットアップしてください。

1. お使いのマシンにKrewをインストールしてセットアップします。
2. プラグインリストをダウンロードします。

```
kubectl krew update
```

3. Krewで利用可能なプラグインを見つけます。

```
kubectl krew search netscaler
```

```
NAME       DESCRIPTION                 INSTALLED

netscaler  Inspect NetScaler Ingresses  no
```

4. プラグインをインストールします。

```
kubectl krew install netscaler
```

注:

Macの場合、開発者アプリを許可を有効にする必要があります。

Kubectlプラグインのサブコマンド使用例

このプラグインで利用できるサブコマンドは次のとおりです。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

サブコマンド	説明
`help`	さまざまなオプションに関する情報を提供します。インストール後にこのコマンドを実行して、インストールが成功したかどうかを確認し、利用可能なコマンドを確認することもできます。
`status`	指定されたプレフィックス入力（プレフィックスのデフォルト値は`k8s`）に対するNetScalerエンティティのステータス（稼働中、停止中、またはアクティブ）を表示します。
`conf`	NetScaler構成（show run出力）を表示します。
`support`	NetScaler (`show techsupport`) およびNetScaler Ingress Controllerのサポートバンドルを取得します。サポート関連情報は2つのtar.gzファイルとして抽出されます。これら2つのtarファイルは、NetScaler ADCからの`show tech support`情報と、イングレスコントローラーが展開されているトラブルシューティング用のKubernetes関連情報です。

サブコマンドの使用例

`Help`コマンド

`help`コマンドは、利用可能なコマンドを知るために使用されます。

```
# kubectl netscaler  --help
```

サブコマンドの詳細については、次のように`help`コマンドを使用してください。

```
# kubectl netscaler  &lt;command> --help
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

`Status` コマンド

`status` サブコマンドは、Kubernetes環境でNetScaler Ingress Controllerによって作成および管理されるNetScalerのさまざまなコンポーネントのステータスを表示します。

コンポーネントは、アプリケーションプレフィックス（NetScaler Ingress Controllerポッドの NS_APPS_NAME_PREFIX環境変数、またはHelmチャートのエンティティプレフィックス値）とイングレス名の両方、またはそのいずれかに基づいてフィルタリングできます。デフォルトの検索プレフィックスは `k8s` です。

フラグ	省略形	説明
デプロイメント		イングレスコントローラーデプロイメントの名前。
--ingress	-i	特定のKubernetesイングレスリソースの構成ステータスを取得するオプションを指定します。
--ラベル	-l	イングレスコントローラーデプロイメントのラベル。
--出力		出力形式。サポートされている形式は、表形式（デフォルト）とJSONです。
--ポッド		イングレスコントローラーポッドの名前。
--プレフィックス	-p	NetScaler Ingress Controllerのデプロイ時に指定されたプレフィックスの名前を指定します。
--詳細	-v	このオプションが設定されている場合、NetScalerの構成タイプやサービスポートなどの追加情報が表示されます。

次の例は、NetScaler名前空間内で、ラベル `app=cic-tier2-citrix-cpx-with-ingress-controller` とプレフィックス `plugin2` を持つNetScaler Ingress Controllerによって作成されたNetScalerコンポーネントのステータスを示しています。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    # kubectl netscaler status -l app=cic-tier2-citrix-cpx-with-ingress-
controller -n netscaler -p plugin



    Showing NetScaler components for prefix: plugin2

    NAMESPACE  INGRESS         PORT  RESOURCE          NAME
                                     STATUS

    --         --              --    Listener
 plugin-198.168.0.1_80_http                              up

    default    --              --    Traffic Policy    plugin-
apache2_80_csp_mqwmhc66h3bkd5i4hd224lve7hjfzvoi  active

    default    --              --    Traffic Action    plugin-
apache2_80_csp_mqwmhc66h3bkd5i4hd224lve7hjfzvoi  attached

    default    plugin-apache2  80    Load Balancer     plugin-
apache2_80_lbv_mqwmhc66h3bkd5i4hd224lve7hjfzvoi  up

    default    plugin-apache2  80    Service           plugin-
apache2_80_sgp_mqwmhc66h3bkd5i4hd224lve7hjfzvoi  --

    default    plugin-apache2  80    Service Endpoint  198.168.0.2
                                    up

    netscaler  --              --    Traffic Policy    plugin-
apache2_80_csp_lhmi6gp3aytmvmww3zczp2yzlyoacebl  active

    netscaler  --              --    Traffic Action    plugin-
apache2_80_csp_lhmi6gp3aytmvmww3zczp2yzlyoacebl  attached

    netscaler  plugin-apache2  80    Load Balancer     plugin-
apache2_80_lbv_lhmi6gp3aytmvmww3zczp2yzlyoacebl  up

    netscaler  plugin-apache2  80    Service           plugin-
apache2_80_sgp_lhmi6gp3aytmvmww3zczp2yzlyoacebl  --

    netscaler  plugin-apache2  80    Service Endpoint  198.168.0.3
                                    up
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

`Conf` コマンド

`conf` サブコマンドは、NetScaler (`show run output`) 上の実行中の構成情報を表示します。 `l` オプションは、NetScaler Ingress Controllerポッドのラベルを照会するために使用されます。

フラグ	短縮形	説明
--デプロイメント		イングレスコントローラーデプロイメントの名前。
--ラベル	-l	イングレスコントローラーデプロイメントのラベル。
--ポッド		イングレスコントローラーポッドの名前。

kubectl NetScaler conf サブコマンドの出力例は次のとおりです。

```
    # kubectl netscaler conf -l app=cic-tier2-citrix-cpx-with-ingress-
controller -n netscaler



    set ns config -IPAddress 198.168.0.4 -netmask 255.255.255.255

    set ns weblogparam -bufferSizeMB 3

    enable ns feature LB CS SSL REWRITE RESPONDER AppFlow® CH

    enable ns mode L3 USNIP PMTUD

    set system user nsroot -encrypted

    set rsskeytype -rsstype ASYMMETRIC

    set lacp -sysPriority 32768 -mac 8a:e6:40:7c:7f:47

    set ns hostName cic-tier2-citrix-cpx-with-ingress-
controller-7bf9c46cb9-xpwvm

    set interface 0/1 -haHeartbeat OFF -throughput 0 -bandwidthHigh 0
-bandwidthNormal 0 -intftype Linux -ifnum 0/1
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    set interface 0/2 -speed 1000 -duplex FULL -throughput 0
-bandwidthHigh 0 -bandwidthNormal 0 -intftype Linux -ifnum 0/2
```

`Support` コマンド

`support` サブコマンドは、NetScaler (`show techsupport`) および NetScaler Ingress Controller のサポートバンドルを取得します。

警告：

NetScaler CPX の場合、テクニカルサポートバンドルファイルは指定した場所にコピーされます。セキュリティ上の理由から、NetScaler Ingress Controller が NetScaler VPX または NetScaler MPX を管理している場合、テクニカルサポートバンドルは抽出されるだけでコピーされません。NetScaler からテクニカルサポートバンドルファイルを手動で取得する必要があります。

`support` サブコマンドのフラグ：

フラグ	省略形	説明
--デプロイメント		イングレスコントローラーデプロイメントの名前。
--ラベル	-l	イングレスコントローラーデプロイメントのラベル。
--ポッド		イングレスコントローラーポッドの名前。
--アプリケーションネームスペース		イングレス、サービス、ポッド、CRDなどのKubernetesリソースの詳細が抽出される、スペース区切りの名前空間のリスト（引用符で囲む）。（例: default "namespace1" "namespace2"）（デフォルト: "default"）。
--ディレクトリ	-d	サポートファイルを保存するディレクトリの絶対パスを指定します。指定しない場合、

© 1997–2026 Citrix Systems, Inc. All rights reserved.

フラグ	省略形	説明
		現在のディレクトリが使用されます。
--unhideIP		Kubernetes情報を収集する際にIPアドレスを非表示にしないように、このフラグを設定します。デフォルトでは、このフラグは`false`に設定されています。
--nsbundleをスキップ		このオプションは、NetScalerからのテクニカルサポートの抽出を無効にします。デフォルトでは、このフラグは`false`に設定されています。

以下は、`kubectl netscaler support`コマンドの出力例です。

```
    # kubectl netscaler support -l app=cic-tier2-citrix-cpx-with-ingress-
controller -n plugin

    Extracting show tech support information, this may take

    minutes.............



    Extracting Kubernetes information

    The support files are present in /root/nssupport_20230410032954
```

NSIC診断ツール

NSIC診断ツールは、NetScaler Ingress ControllerとKubernetesクラスターにデプロイされたアプリケーションに関する情報を収集するシェルスクリプトです。このツールは、必要な情報を抽出するために、名前空間、CNI、および出力ディレクトリパスを入力として受け取り、出力ファイルをtar形式で保存します。ユーザーが機密情報であり共有すべきではないと考える情報がある場合

© 1997–2026 Citrix Systems, Inc. All rights reserved.

は、ユーザーが指定した出力ディレクトリパスの下にある出力_ ディレクトリをスキャンし、tar ファイルを再作成して共有してください。

NSIC診断ツールスクリプトをダウンロードし、`nsic_diagnostics_tool.sh`を実行します。

NetScalerノードコントローラーの問題のトラブルシューティング

このトピックでは、ノードコントローラーの使用中に発生する可能性のある問題のトラブルシューティング方法について説明します。ログを収集して原因を特定し、ノードコントローラーの構成に関連する一般的な問題の回避策を適用できます。

ノードコントローラーと基本的なノード構成を検証するには、展開 ページの画像を参照してください。

ルーターポッドが起動しない

`kube-nsnc-router` ポッドが起動しない場合、非管理者ユーザー/権限による特権ポッドのデプロイを妨げる特定のクラスター制限が原因である可能性があります。

回避策として、次のいずれかを実行してください。

1. ノードコントローラーをデプロイするには、`kube-system` 名前空間を使用します。
2. ノードコントローラー `clusterrolebinding` に `cluster-admin` ロールを割り当てます。

注:

オプション1を選択した場合、クラスターごとに利用可能な `kube-system` 名前空間は1つしかないため、単一のクラスターでノードコントローラーの複数のインスタンスを作成することはできません。

サービスステータスがDOWN

サービスが「ダウン」状態にあるときに問題をデバッグするには、次の手順を実行します。

1. 次のコマンドを使用して、ノードコントローラーポッドのログを確認します。

```
kubectl logs <nsnc-pod> -n <namespace>
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ログに「permission」エラーがないか確認します。ノードコントローラーは `kube-nsnc-router` ポッドを作成しますが、これはノードで構成を実行するために `NET_ADMIN` 特権を必要とします。したがって、ノードコントローラーサービスアカウントは `NET_ADMIN` 特権とホストモード `kube-nsnc-routerpods` を作成する能力を持っている必要があります。

2. 次のコマンドを使用して、`kube-nsnc-router` ポッドのログを確認します。

```
kubectl logs <kube-nsnc-pod> -n <namespace>
```

ノード構成にエラーがないか確認します。以下はルーターポッドログのサンプルです。

3. 次のコマンドを使用して、`kube-nsnc-router` ConfigMap の出力を確認します。

```
kubectl get configmaps -n <namespace> kube-nsnc-router -o yaml
```

ConfigMap の「data」セクションに空のフィールドがないか確認します。以下は2ノードのデータセクションのサンプルです。

4. ノード構成を確認し、以下を確実にします。

- ノードコントローラーインターフェイス `nsncvxlan<md5_of_namespace>` が作成されていること。
    - 割り当てられたVTEP IPアドレスが、NetScalerの対応するルーターゲートウェイエントリと同じであること。
    - インターフェイスのステータスが機能していること。
- `iptable` ルールポートが作成されていること。
    - ポートがNetScalerで作成されたVXLANと同じであること。

サービスステータスは稼働中だが、NetScalerからのpingが機能しない

この問題は、SRCIPをNSIPとしてNetScalerからのパケットをデフォルトゲートウェイに転送するPBRエントリの存在が原因で発生する可能性があります。この問題は、いかなる機能にも影響を与えません。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

NetScalerコマンドラインインターフェイスでpingコマンドの`-S`オプションを使用して、NetScalerのVTEPを送信元IPアドレスとして使用できます。例：

```
ping <serviceIP> -S <vtepIP>
```

注:

NSIP自体でpingする必要がある場合は、PBRエントリを削除するか、優先度の高いエンドポイントのPBRエントリを追加する必要があります。

ポッドエンドポイントまたはVIPへのcURLが機能しない

サービスが稼働状態であっても、ポッドエンドポイントにcURLできない場合、エンドポイントへのステートフルTCPセッションが失敗していることを意味します。1つの理由として、nsモード「MBF」が有効に設定されている可能性があります。この問題はデプロイメントに依存し、NetScalerの特定のバージョンでのみ発生する可能性があります。

この問題を解決するには、`MBF ns`モードを無効にするか、`netprofile`が無効になっている`netprofile`をサービスグループにバインドする必要があります。

注:

MBFを無効にすることで問題が解決する場合は、MBFを無効に設定し続ける必要があります。

カスタマーサポート

一般的なサポートについては、カスタマーサポートに問題を提起する際に、より迅速なデバッグに役立つ以下の詳細情報を提供してください。NetScalerからエンドポイントへのcURLまたはpingを実行し、以下の詳細情報を取得してください。

ノードについては、以下のコマンドの詳細情報を提供してください。

1. ノード上のノードコントローラーインターフェースでの`tcpdump`キャプチャ。

   ```
   tcpdump -i nsncvxlan<hash_of_namesapce> -w nsncvxlan.pcap
   ```

2. ノードMgmtインターフェースでの`tcpdump`キャプチャ（例: "eth0"）。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
tcpdump -i eth0 -w mgmt.pcap
```

3. CNIインターフェースでの`tcpdump`キャプチャ（例: "vxlan.calico"）。

```
tcpdump -i vxlan.calico -w cni.pcap
```

4. ノード上の`ifconfig -a`の出力。
5. ノード上の`iptables -L`の出力。

NetScalerについては、以下のshowコマンドの詳細を提供してください。

1. `show ip`
2. `show vxlan <vxlan_id>`
3. `show route`
4. `show arp`
5. `show bridgetable`
6. `show ns pbrs`
7. `show ns bridgetable`
8. `show ns mode`
9. ping/cURLを使用して`nstrace`をキャプチャしてみてください。

```
start nstrace -size 0 -mode rx new_rx txb tx -capsslkeys enABLED
```

```
stop nstrace
```

NetScaler Ingress Controller をアップグレードする

このセクションでは、スタンドアロンデプロイとサイドカーデプロイ (NetScaler CPX と NetScaler Ingress Controller) の両方で NetScaler Ingress Controller インスタンスをアップグレードする方法について説明します。アップグレード手順により、アップグレードプロセス中にデータトラフィックが影響を受けないことが保証されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

前提条件

- **シークレット**: アップグレードされた NSIC バージョンを別の名前空間に配置する場合は、その名前空間にシークレットを配置します。
- **デプロイ名**: YAML ベースのデプロイと Helm ベースのデプロイで異なる名前を使用しても、機能に影響はありません。ただし、`fullnameOverride` 引数を使用することで、同じ名前を保持できます。
- Helm ベースの NSIC デプロイの場合、`helm get values <release_name> -a` コマンドを実行して、既存のデプロイの values.yaml を取得します。

YAML ファイルを使用してデプロイされた NSIC をアップグレードする

YAML ファイルを使用してデプロイされた NSIC インスタンスがある場合は、この手順に従って Helm チャートを使用して NSIC をアップグレードします。

1. 既存のデプロイの CRD インスタンスをバックアップします。CRD インスタンスを YAML ファイルにバックアップするには、次のコマンドを実行します。

   ```
   for crd in $(kubectl get crds -o jsonpath='{.items[*].metadata.name}' |
   tr ' ' '\n' | grep 'citrix.com'); do   echo "Getting instances for CRD:
   $crd";   kubectl get $crd --all-namespaces -o yaml >>
   "all_crd_instances.yaml"; done
   ```

2. 現在のデプロイと ConfigMap YAML ファイルに基づいて、`values.yaml` (スタンドアロンデプロイの場合) または values.yaml (サイドカーデプロイの場合) で正しい引数を設定します。

   注:

   - `entityPrefix` には一意の値を指定することをお勧めします。
   - NSIC インスタンスを特定のイングレスリソースに関連付けるために、必要に応じて `ingressClass` を指定することをお勧めします。
   - NSIC インスタンスを LoadBalancer タイプの特定のサービスに関連付けるために、必要に応じて `serviceClass` を指定することをお勧めします。

3. 既存の NSIC デプロイメントと、既存の NSIC デプロイメントの一部として作成された ClusterRole、ClusterRoleBinding、ServiceAccount などの関連リソースを削除します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
kubectl delete deployment <NSIC_DEPLOYMENT_NAME>

kubectl delete clusterrole <NSIC_CLUSTERROLE_NAME>

kubectl delete clusterrolebinding <NSIC_CLUSTERROLEBINDING_NAME

kubectl delete serviceaccount <NSIC_SERVICEACCOUNT_NAME>
```

4. Configmapを削除します。

```
kubectl delete configmap <CONFIGMAP_NAME>
```

5. 次のコマンドを使用して、NetScaler Helmチャートリポジトリをローカルレジストリに追加します。

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-
charts/
```

NetScaler Helmチャートリポジトリがローカルレジストリに既に追加されている場合は、次のコマンドを使用してリポジトリを更新します。

```
helm repo update netscaler
```

6. 変更された`values.yaml`を使用してNetScaler Ingress Controllerを展開します。

```
helm install <existing NSIC release name deployed using YAML>
netscaler/netscaler-ingress-controller -f values.yaml
```

7. 次のコマンドを使用してCRD仕様を展開します。

```
kubectl apply -f https://raw.githubusercontent.com/netscaler/netscaler-
helm-charts/refs/heads/master/netscaler-ingress-controller/crds/
crds.yaml
```

8. 必要に応じて、ステップ1でバックアップされたCRDインスタンスを展開します。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

Helmチャートを使用して展開されたNSICをアップグレードする

Helmチャートを使用して展開されたNSICをアップグレードするには、この手順に従ってください。

1. 既存の展開のCRDインスタンスをバックアップします。CRDインスタンスをYAMLファイルにバックアップするには、次のコマンドを実行します。

```
for crd in $(kubectl get crds -o jsonpath='{.items[*].metadata.name}' |
tr ' ' '\n' | grep 'citrix.com'); do   echo "Getting instances for CRD:
$crd";   kubectl get $crd --all-namespaces -o yaml >>
"all_crd_instances.yaml"; done
```

2. (オプション) 次のコマンドを使用して、NetScaler Helmチャートリポジトリをローカルレジストリに追加します。

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-
charts/
```

3. NetScaler Helmチャートリポジトリがローカルレジストリに既に追加されている場合は、次のコマンドを使用してリポジトリを更新します。

```
helm repo update netscaler
```

4. `cic.values`を`nsic.values`に置き換えます。
5. NSICをアップグレードするには:

   a. (オプション) 既存のNSICが`Citrix`リポジトリで利用可能なチャートを使用してインストールされている場合は、次の手順を実行します。それ以外の場合は、次のステップに進みます。

      - スタンドアロンNSICの場合、values.yamlを次の属性で更新します。
        - `imageRepository`の値を`netscaler/netscaler-k8s-ingress-controller`として指定します。
        - (オプション) NSIC バージョン 1.35.6 以前を 1.36.5 以降のバージョンにアップグレードするには、NetScaler Helm チャートのパラメータ命名に一致するように、`cncPbr`を`nsncPbr`に置き換えます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- `helm get values <release_name> -a` コマンドを実行し、それに応じて `values.yaml` ファイルを更新します。

- サイドカーとしての NSIC の場合、values.yaml を次の属性で更新します。
  - `nsic.imageRepository` の値を `netscaler/netscaler-k8s-ingress-controller` として指定します。
  - (オプション) NSIC バージョン 1.35.6 以前を 1.36.5 以降のバージョンにアップグレードするには、NetScaler Helm チャートのパラメータ命名に一致するように、`cncPbr` を `nsncPbr` に置き換えます。
  - `helm get values <release_name> -a` コマンドを実行し、それに応じて `values.yaml` ファイルを更新します。

注:

- `entityPrefix` には一意の値を指定することをお勧めします。
- NSIC インスタンスを特定のイングレスリソースに関連付けるために、必要に応じて `ingressClass` を指定することをお勧めします。
- NSIC インスタンスを LoadBalancer タイプの特定のサービスに関連付けるために、必要に応じて `serviceClass` を指定することをお勧めします。

b. 次のコマンドを実行します。

```
helm upgrade netscaler-ingress-controller netscaler/netscaler-ingress-controller -f values.yaml [--version=<supported nsic version>]
```

注:

サポートされている NSIC バージョンのリストについては、このページ を参照してください。

6. 次のコマンドを使用して CRD 仕様を展開します。

```
kubectl apply -f https://raw.githubusercontent.com/netscaler/netscaler-helm-charts/refs/heads/master/netscaler-ingress-controller/crds/crds.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

7. 必要に応じて、ステップ1でバックアップされたCRDインスタンスをデプロイします。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

リファレンス

リファレンスには、以下のトピックがあります。

- エンティティ名の変更
- CRDのエンティティ名の変更
- ライセンス

エンティティ名の変更

NetScalerエンティティを追加する際、NetScaler Ingress ControllerはIngress、サービス、または名前空間ごとに一意の名前を維持します。これにより、NetScalerの命名制限を超える長い名前のNetScalerエンティティが生成されることがあります。

現在、NetScaler Ingress Controllerの命名形式は、エンティティ名を短縮するために更新されています。更新された命名形式では、エンティティ名の一部がハッシュ化され、必要なすべての情報がエンティティコメントの一部として提供されます。

この更新後、`lbvserver` および `servicegroup` エンティティ名で利用可能なコメントには、Ingress名、Ingressポート、サービス名、サービスポート、アプリケーションの名前空間など、必要なすべての詳細が提供されます。

コメントの形式

イングレス: `ing:<ingress-name>,ingport:<ingress-port>,ns:<k8s-namespace>,svc:<k8s-servicename>,svcport:<k8s-serviceport>`

`type LoadBalancer` のサービス: `lbsvc:<k8s-servicename>,svcport:<k8s-serviceport>,ns:<k8s-namespace>`

次の表は、NetScaler Ingress Controllerバージョン1.12で導入されたエンティティ名の変更について説明しています。

エンティティ	古い命名形式	新しい命名形式	説明/コメント
`csvserver` (イングレス)	`k8s-192.2.170.67_80_http`	`k8s-192.2.170.67_80_http`	変更なし

© 1997–2026 Citrix Systems, Inc. All rights reserved.

エンティティ	古い命名形式	新しい命名形式	説明/コメント
`csvserver` (`type LoadBalancer`)	`k8s-apache_default_80_svc`	`k8s-apache_80_default_svc`	現在、ポートの後に名前空間が続きます
`lbvserver` (`type LoadBalancer`)	`k8s-apache_default_80_svc_k8s-apache_default_80_svc`	`k8s-apache_80_lbv_wlikeqxno5vunbthsoj4lxegk7cddh6p` コメント: `lbsvc:apache,svcport:80,ns:default`	`type LoadBalancer` のコメントが変更されました
`servicegroup` (`type LoadBalancer`)	`k8s-apache_default_80_svc_k8s-apache_default_80_svc`	`k8s-apache_80_sgp_wlikeqxno5vunbthsoj4lxegk7cddh6p`	サフィックス `sgp` が追加されます
`cspolicy` または `csaction` または `responder policy`	`k8s-web-ingress_default_443_k8s-frontend_default_80_svc`	`k8s-frontend_80_csp_267pneiak5rw6hoygvrqrzpm4k6thz2p`	service-name、service-portを先頭に移動し、csのサフィックスを追加し、ingress-name、ingress-port、およびnamespaceをハッシュ化しました
`lbvserver` (イングレス)	`k8s-web-ingress_default_443_k8s-frontend_default_80_svc`	`k8s-frontend_80_lbv_267pneiak5rw6hoygvrqrzpm4k6thz2p` コメント: `ing:web-ingress,ingport:5080,ns:default,svc:frontend,svcport:80`	エンティティにサフィックス `lbv` とコメントが追加されました
`servicegroup` (イングレス)	`k8s-web-ingress_default_4`	`k8s-frontend_80_sgp_2`	サフィックス `sgp` が追加されます。

© 1997–2026 Citrix Systems, Inc. All rights reserved.

エンティティ	古い命名形式	新しい命名形式	説明/コメント
	`43_k8s-frontend_default_80_svc`	`67pneiak5rw6hoygvrqrzpm4k6thz2p`	
`lbvserver` (UDP)	`k8s-web-ingress_default_9053-udp_k8s-bind_default_53-udp_svc`	`k8s-bind_53-udp_lbv_uyomblblagixrtw3cxrf23tak6wkpfmw`	`-udp`は以前と同様にポートに追加されます。

NetScaler Ingress Controllerの古いバージョンから最新バージョンにアップグレードすると、NetScaler Ingress Controllerはすべてのエンティティを新しい命名形式で名前変更します。ただし、NetScaler Ingress Controllerは最新バージョンから古いバージョンへのダウングレードを処理しません。

CRDのエンティティ名の変更

NetScalerカスタムリソース定義 (CRD) インスタンスが作成されると、NetScaler Ingress Controllerは、そのCRDインスタンスに関連付けられた複数のNetScalerエンティティを生成します。NetScaler Ingress Controllerは、CRDインスタンスとの関連付けを維持するために、各エンティティに一意の名前を保持します。エンティティの命名はCRD名に直接基づいているため、一部のNetScalerエンティティ名が最大文字数制限を超えていました。

NetScaler Ingress Controller 4.0.x以降、CRD作成時に短いエンティティ名を生成するために、次のアプローチを使用して命名規則が最適化されています。

- **ハッシュ化された命名**: エンティティ名の一部がハッシュ化され、全体の長さが短縮されます
- **保持される情報**: エンティティコメントがNetScalerでサポートされている場合、必要なKubernetes関連のメタデータはエンティティのコメントフィールドに保持されます
- **互換性の向上**: 名前はNetScalerの文字制限に準拠し、完全なトレーサビリティを維持します

NetScaler Kubernetes Ingress Controller 4.1.x以降、エンティティ名に関して以下の機能強化が行われています。

1. **ハイフンをアンダースコアに置換**: CRD名と名前空間内のすべてのハイフンは、エンティティ名ではアンダースコアに変換されます (GSLBエンティティを除く)。
    - 例: `route-for-all` → `route_for_all`

© 1997–2026 Citrix Systems, Inc. All rights reserved.

- 例: `other-namespace` → `other_namespace`

2. **LBvserverの命名を簡素化**: 名前には一意性を確保するためのハッシュが含まれます。
   - 旧: `k8s_crd_k8shttproute_route-for-all_other-namespace_0_lb`
   - 新: `k8s_c15_route_for_all_other_name_http_MFRGGZDFMY4TMZLDGE4TCMZWGIZQYNR2GEZDGNBVGY3TQO`
3. **GSLBエンティティはハイフンを保持**: GSLB関連エンティティ (`gslbservicegroup`, `gslbvserver`) はハイフンを保持します。
   - 例: `cocacola-com` は `cocacola-com` のままです (`cocacola_com`には変換されません)

以下の表は、NetScaler Ingress Controller バージョン 4.0.x リリースで導入されたエンティティ名の変更について説明しています。

エンティティ名	古い命名形式	新しい命名形式
アップエフダブリューポリシー	k8s_crd_waf_wafbasicdefault	k8S_c21_wafbasic_default_DTUHVZOZSCSV4SPAPPQPZSON6VF656I6JSHGRVAB5SCMU6AC5HNA
アップエフダブリュープロファイル	k8s_crd_waf_wafbasicdefault	k8s_c21_wafbasic_default_UXHDUNMOOGVVLEMJ3QRBQSZE24UN2NYR223J365CFRGKLQUZYJQQ
アップエフダブリューシグネチャ	k8s_crd_waf_wafbasicdefault	k8s_c21_wafbasic_default_WNORDMGMY2XDLGL6QBJB27EIM5YWKRGECIXDESDKYJ23IBIUQ6HA
アップキューオーイーアクション	k8s_crd_appqoepolicy_aqaction_targeturlappqoe_0_default	k8s_c02_targeturlappqoe_default_IYTOAY6HRWAJQDMHDS3KNLTACGGJEZXZTW4CKQRCD33ALSFZTALQ
アプリケーションQoEポリシー	k8s_crd_appqoepolicy_aqpolicy_targeturlappqoe_0_default	k8s_c02_targeturlappqoe_default_L3LV3IFDRMXAMDK3IIKAR3HLIM5CNRUYDS36ONBGX5TV7XXWTYFA

© 1997–2026 Citrix Systems, Inc. All rights reserved.

エンティティ名	古い命名形式	新しい命名形式
監査メッセージアクション	k8s_crd_rewritepolicy_rwaudmesaction_multipolicy1_1_default	k8s_c17_multipolicy1_default_P6NUHC62V4KJE442YEVGOMTBJWANVCQ4KTXUURS4SEVD4V2PQUQQ
認証LDAPアクション	k8s_crd_authpolicy_authhotdrinks_ldap-auth-provider_1_default	k8s_c03_authhotdrinks_default_TZZOEQ2I5R5U7RFHNK5IPPOQCIXP2O4R6P4ZMCMZMXIOLCPTCMFA
認証ログインスキーマ	k8s_crd_authpolicy_autht1_lp_jwt-auth-provider_1_default	k8s_c03_autht1_default_2XOCNX67BWBEASZ7V3TT3BJTUOFILRZ6UYTRUH6DS3CRUPATPOMA
認証ログインスキームポリシー	k8s_crd_authpolicy_autht1_lp_jwt-auth-provider_1_default	k8s_c03_autht1_default_2XOCNX67BWBEASZ7V3TT3BJTUOFILRZ6UYTRUH6DS3CRUPATPOMA
認証OAuthアクション	k8s_crd_authpolicy_autht1_authpolicy_1_default	k8s_c03_autht1_default_LOJ53HJPDB56SFB67CJ6NBBTO3XC7NHQ54SCP4MMWSBGJR5QS3LQ
認証ポリシー	k8s_crd_authpolicy_authhotdrinks_authpolicy_1_default	k8s_c03_authhotdrinks_default_GMA6HKR4ZM6LGWQZ7RNLDCRB3TXWSUFBEHEE3TVOJTXZO6DKYDTQ
認証SAMLアクション	k8S_crd_authpolicy_authhotdrinks_saml-auth-provider_1_default	k8s_c03_authhotdrinks_default_UODCZMVDCPIFBZXWP3BBV2RCWWXGF3ZRZ4V3UED2W4HNI6POBVXQ
認証vServer	k8s_crd_authpolicy_aaa_FormsName_authvsrv	k8s_c03_authhotdrinks_default_4X6TDA5WAIGTDSFED56OC526NQQAMV76DHKZOETVJRWMURP4AJ4Q

© 1997–2026 Citrix Systems, Inc. All rights reserved.

エンティティ名	古い命名形式	新しい命名形式
認可ポリシー	k8s_crd_authpolicy_authhotdrinks_authzpolicy_1_default	k8s_c03_authhotdrinks_default_C7HVMV336LZ2IS6WRQQLY2N3ETNQN22DBY4LD7AHSXV7YKQ2JLYQ
ボットポリシー	k8s_crd_bot_btdefault	k8s_c04_SBKZMNXAVR4HHP
ボットポリシーラベル	k8s_crd_bot_btdefault	k8s_c04_SBKZMNXAVR4HHP
ボットプロファイル	k8s_crd_bot_btdefault	k8s_c04_SBKZMNXAVR4HHP
ボットシグネチャ	k8s_crd_bot_btdefault	k8s_c04_SBKZMNXAVR4HHP
コンテンツインスペクションアクション	k8s_crd_icappolicy_exampleicappolicy_ci-action_0_default	k8s_c12_exampleicappolicy_default_MCQ3EFMGPJXSE2O7FFCJNL347VB6PNZTBAQZMGW4DOHBBE4MNE
コンテンツインスペクションポリシー	k8s_crd_icappolicy_exampleicappolicy_ci-policy_0_default	k8s_c12_exampleicappolicy_default_MI33BP5BZP7ZKXMIBXD4A6ZZNRQG5YS4ZRXUQI3N34KVDZ4UG7
CSアクション	k8s_crd_k8shttproute_route-for-all_other-namespace_0	k8s_c15_route_for_all_other_name_NSOCBGCEIEMS3U2K..._0
CSポリシー	k8s_crd_k8shttproute_route-for-all_other-namespace_0	k8s_c15_route_for_all_other_name_D74FE2SLZD7KJP..._0
CS仮想サーバー	k8s_crd_Listener_10.1.3.1_80_http	k8s_c14_my_listener_default_10.1.3.1_80_http
GSLBサービスグループ	k8s_crd_globaltrafficpolicy_cid_east_cluster1_default_cold1_cocacola-com	k8s_c10_cid_east_cluster1_cold1_default_cocacola_com_0_HTTP

© 1997–2026 Citrix Systems, Inc. All rights reserved.

エンティティ名	古い命名形式	新しい命名形式
GSLBvサーバー	k8s_crd_globaltrafficpolicy_default_cold1_cocacola-com	k8s_c10_cold1_default_cocacola_com_0_HTTP
IPセット	k8s_crd_Listener_1.1.1.1_443_ssl_ipset	k8s_c14_my-listener_default_HTDWUDXLSZW2UM6UX4X3MGVUPWTINT7YVANRWA_ipset
LBモニター	k8s_crd_globaltrafficpolicy_RZHNBQRQKJYYRRY6NXQ2MM6QVMQPCJ6C3PL	k8s_c10_cold1_default_ATG3IPNGAVACEEMIYRZIVYYFNKXVCKXI66C5RISIXAHXQIGGS5RA
LBvサーバー	k8s_crd_k8shttproute_route-for-all_other-namespace_0_lb	k8s_c15_route_for_all_other_name_http_MFRGGZDFMY4TMZLDGE4TCMZWGIZQYNR2GEZDGNBVGY3TQOJQ
NS HTTPプロファイル	k8s_crd_Listener_10.1.3.1_80_http	k8s_c14_my-listener_default_10.1.3.1_80_http
エヌエスアイシーアッププロファイル	k8s_crd_icappolicy_exampleicappolicy_icap-profile_0_default	k8s_c12_example_default_PJZLCPKQX6PPR6Y57O37PYBEHD6PT2YSRKPC4JNPBLTD2
エヌエスリミットアイデンティファイア	k8s_crd_ratelimit_default_throttle_apis_codecov	k8s_c17_throttle_default_HUADSDGHAHJKSHUBSD
エヌエスティーシーピープロファイル	k8s_crd_Listener_10.1.3.1_80_http	k8s_c14_my-listener_default_10.1.3.1_80_http
ポリシーデータセット	k8s_crd_rewritepolicy_dataset_multipolicy3_0_default_redirectIPs	k8s_c17_multipolicy3_default_6NGFYEUJWIQOOO5ZUUKSGVMIP6O3BNPJIZ5OTWAXPVCLCC5GEZRA
ポリシーエイチティーティーピーコールアウト	k8s_crd_httpcallout_dbf2bfba	k8s_c17_DD6U4WGWATNEL7

© 1997–2026 Citrix Systems, Inc. All rights reserved.

エンティティ名	古い命名形式	新しい命名形式
ポリシーパターンセット	k8s_crd_authpolicy_authhotdrinks_authpatset_1_default	k8s_c03_authhotdrinks_default_SHZXI2Z3WFHEZVA5BDAJ53R7TZSEUFJXSKRN5YQCRAZSO5V4N4SA
ポリシーストリングマップ	k8s_crd_rewritepolicy_stringmap_basicconfig_0_default_s1	k8s_c17_basicconfig_default_XBMDW6QHZINFHIWFRLWHHGAAWOXTDIGJQTLNZST3MVYOY2QYFYSQ
レスポンダーアクション	k8s_crd_rewritepolicy_respaction_basicconfig_0_default	k8s_c17_basicconfig_default_ZOHREQM7YYCY3CZ7IOP7YLSVDKVDRPMJYAHG4ODFY76R3ZFOR74Q_rw
レスポンダーポリシー	k8s_crd_rewritepolicy_respolicy_basicconfig_0_default	k8s_c17_basicconfig_default_YTSRW4EDTQ5HEQKXD6SDS2KOBO7N26P5HQCIQHCS5GV3YXOI473A
リライトアクション	k8s_crd_k8shttproute_my-httproute_default_urlrewrite_host_0_0	k8s_c15_my_httproute_default_YDWIG6X..._urlrw_host_0_0
リライトポリシー	k8s_crd_rewritepolicy_rwpolicy_multipolicy1_0_default	k8s_c17_multipolicy1_default_AY2LPEUWN6JPESA4L4RO7FZTQJ7P66CVA4NW2LHHR3IJF6GJWKVA
サービスグループ	k8s_crd_icappolicy_exampleicappolicy_icap-sg_0_default	k8s_c12_exampleicappolicy_default_GD3T7QZNKV5EDOYDI2KBB5JTPM4EMBLATOUJVZMGVX7WJNCQAN
SSL暗号	k8s_crd_Listener_1.1.1.1_443_ssl	k8s_c14_DZ5VMPYNSHVTTYF26BTLAS
SSLプロファイル	k8s_crd_Listener_10.1.3.1_443_ssl	k8s_c14_my_listener_default_10.1.3.1_443_ssl

© 1997–2026 Citrix Systems, Inc. All rights reserved.

エンティティ名	古い命名形式	新しい命名形式
SSL仮想サーバー	k8s_crd_Listener_10.1.3.1_443_ssl	k8s_c14_my_listener_default_10.1.3.1_443_ssl
ストリームセレクター	k8s_crd_ratelimit_default_throttlecoffeeperclientip	k8s_c17_throttle_default_HUADSDGHAHJKSHUBSD

アップグレードシナリオ

NetScaler Ingress Controllerの以前のバージョンからバージョン4.0.xにアップグレードすると、NetScaler Ingress Controllerは、すべてのエンティティを新しい命名形式で名前変更します。アップグレード中、エンティティが削除および再作成されるため、トラフィックが中断され、ダウンタイムが発生します。

注：

NetScaler Ingress Controllerは、最新バージョンから以前のバージョンへのダウングレードを処理しません。

NetScaler Ingress Controllerをバージョン4.0.xからバージョン3.x.xにダウングレードする

Citrix CRD構成を保持しながらNetScaler Ingress Controllerを安全にダウングレードするには、次の手順を実行します。

1. すべてのCitrix CRDインスタンスをバックアップします。このコマンドは、すべての名前空間からすべてのCitrix CRDインスタンスを、適切なドキュメント区切り文字を含む単一のYAMLファイルにバックアップします。

```
rm -f all_crd_instances.yaml

for crd in $(kubectl get crds -o jsonpath='{.items[*].metadata.name}' |
tr ' ' '\n' | grep 'citrix.com'); do

echo "Getting instances for CRD: $crd"

kubectl get $crd --all-namespaces -o name | while read resource; do

    kubectl get $resource -o yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    echo "---"

done >> all_crd_instances.yaml

done
```

バックアップファイルが作成されたことを確認します。

```
ls -lh all_crd_instances.yaml
```

2. 次のコマンドを使用して、既存の展開からCRDインスタンスを削除します。

```
kubectl delete -f all_crd_instances.yaml
```

3. NetScaler Ingress Controllerが削除イベントを処理し、関連リソースをクリーンアップできるようにします。

```
kubectl logs -n <nsic-namespace> <nsic-pod-name> -f
```

4. NetScaler Ingress Controllerの展開を目的の3.x.xバージョンに更新して、NetScaler Ingress Controllerをダウングレードします。

```
# Example: using Helm

helm upgrade nsic citrix/citrix-ingress-controller --version <3.x.x-
version>

# Or: using kubectl

kubectl set image deployment/nsic nsic-container=<image:3.x.x-tag> -n
<nsic-namespace>
```

5. ダウングレードされたNetScaler Ingress Controllerポッドが稼働し、準備が整ったら、バックアップされたCRDインスタンスを復元します。

```
kubectl apply -f all_crd_instances.yaml
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

ライセンス

NetScaler CPXのライセンスのために、NetScaler Application Delivery Management (ADM)がライセンス情報を自動的に取得できるように、YAMLで以下の情報を提供する必要があります。

- LS_IP (ライセンスサーバーIP) – NetScaler ConsoleのIPアドレスを指定します。
- LS_PORT (ライセンスサーバーポート) – これは必須フィールドではありません。ADMポートは、変更した場合にのみ指定する必要があります。デフォルトポートは27000です。
- PLATFORM – プラットフォームライセンスを指定します。プラットフォームは`CP1000`です。

以下はYAMLファイルのサンプルです。

```
apiVersion: apps/v1

kind: Deployment

metadata:

  labels:

    name: cpx-ingress

  name: cpx-ingress

spec:

  replicas: 1

  selector:

    matchLabels:

      name: cpx-ingress

  template:

    metadata:

      annotations:

        NETSCALER_AS_APP: "True"
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
    labels:
      name: cpx-ingress
  spec:
    serviceAccountName: cpx
    containers:
    - args:
      - --ingress-classes citrix-ingress
      env:
      - name: EULA
        value: "YES"
      - name: NS_PROTOCOL
        value: HTTP
      - name: NS_PORT
        value: "9080"
      - name: LS_IP
        value: <ADM IP>
      - name: LS_PORT
        value: "27000"
      - name: PLATFORM
        value: CP1000
      image: cpx-ingress:latest
      imagePullPolicy: Always
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

```
        name: cpx-ingress
        ports:
        - containerPort: 80
          name: http
          protocol: TCP
        - containerPort: 443
          name: https
          protocol: TCP
        - containerPort: 9080
          name: nitro-http
          protocol: TCP
        - containerPort: 9443
          name: nitro-https
          protocol: TCP
        securityContext:
          privileged: true
```

© 1997–2026 Citrix Systems, Inc. All rights reserved.

fix

This is the content for the sample heading topic. It is hidden from the main table of contents but remains accessible via links.

© 1997–2026 Citrix Systems, Inc. All rights reserved.