

NetScaler® Kubernetes Gateway Controller

Contents

リリースノート	12
リリース 2.2.0	12
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 リリース 2.2.0	12
新機能	12
Helm チャートリリース	13
リリース 2.1.0	13
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 リリース 2.1.0	13
新機能	13
修正された問題	13
Helmチャートのリリース	14
リリース 2.0.0	14
ネットスケーラー® キューバネティス ゲートウェイ コントローラー リリース 2.0.0	14
新機能	14
Helmチャートのリリース	16
リリース 1.4.0	16
NetScaler® クバネティス ゲートウェイ コントローラー バージョン 1.4.0 のリリース	16
修正された問題	16
Helm チャートのリリース	16
リリース 1.3.0	16
ネットスケーラー® キューバネティス ゲートウェイ コントローラー リリース 1.3.0	16
新機能	16
Helm チャートリリース	17
リリース 1.2.0	17
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 リリース 1.2.0	17
新機能	17
Helmチャートリリース	18

リリース 1.1.0	18
ネットスケーラー キューバネティス ゲートウェイ コントローラー リリース 1.1.0	18
新機能	18
Helmチャートリリース	19
リリース 1.0.0	19
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 リリース 1.0.0	19
新機能	19
Helm チャート リリース 1.0.0	20
概要	21
NetScaler® Kubernetes Gateway コントローラー	21
Kubernetes Gateway APIプロバイダーとしてのNetScaler	22
ユースケース	22
はじめに	23
前提条件	23
Kubernetes クラスターに NetScaler® Kubernetes Gateway Controller を展開する方法	23
KubernetesクラスターでNetScaler Kubernetes Gateway ControllerとNetScaler CPXをデプ ロイする方法	25
次は何をしますか	27
サポートされているGateway API CRD	28
ゲートウェイクラスCRD	28
ゲートウェイ CRD	28
HTTPRoute カスタムリソース定義	29
TCPRoute CRDについて	30
UDPRoute カスタムリソース定義	31
注意点	32
サポートされているプラットフォームと展開	32
Kubernetesプラットフォーム	33
NetScalerとの互換性	33

NetScaler オペレーターと NetScaler Kubernetes Gateway コントローラーの互換性	34
インストール	35
NetScaler® Kubernetes Gateway Controller を展開する	35
Gateway および HTTPRoute CRD を使用して HTTP 経由でアプリケーションを公開する	39
Gateway および HTTPRoute CRD を使用した HTTPS/SSL によるアプリケーションの公開	39
Gateway および TCPRoute CRD を使用して TCP アプリケーションを公開する	47
ステップ1: サンプル TCP アプリケーションをデプロイする	47
ステップ2: GatewayClass を作成する	48
ステップ3: TCP リスナーを持つ Gateway を作成する	49
ステップ4: TCPRoute を作成する	50
ステップ5: 構成を確認する	51
Gateway および UDPRoute CRD を使用した UDP アプリケーションの公開	51
ステップ 1: サンプルの UDP アプリケーションを展開する	51
ステップ 2: GatewayClass を作成する	54
ステップ 3: UDP リスナーを持つ Gateway を作成する	55
ステップ 4: UDPRoute を作成する	56
ステップ 5: 構成を確認する	56
NetScaler® CPX Kubernetes Gateway Controller のデプロイ	57
Gateway および HTTPRoute CRD のサンプル	61
Gateway および TCPRoute CRD を使用して TCP アプリケーションを公開する	67
ステップ 1: サンプル TCP アプリケーションをデプロイする	67
ステップ 2: GatewayClass を作成する	69
ステップ 3: TCP リスナーを持つ Gateway を作成する	69
ステップ 4: TCPRoute を作成する	70
ステップ 5: 構成を確認する	71
Gateway および UDPRoute CRD を使用した UDP アプリケーションの公開	72
ステップ 1: サンプル UDP アプリケーションを展開する	72
ステップ 2: GatewayClass を作成する	75

ステップ 3: UDP リスナーを持つ Gateway を作成する	75
ステップ 4: UDPRoute を作成する	76
ステップ5: 設定を確認する	77
NetScaler オペレーター	78
NetScaler Operator リリースノート	78
NetScaler® Operator リリースノート	79
バージョン 3.6.0	79
NetScaler® Operator リリースノート	85
バージョン 3.5.0	85
NetScaler® Operator リリースノート	85
バージョン 3.4.1	85
NetScaler® Operator リリースノート	86
バージョン 3.3.0	86
NetScaler Operator を使用して OpenShift に NetScaler Kubernetes Gateway Controller をデブ ロイする	86
前提条件	86
OpenShift Operator を使用して NetScaler Kubernetes Gateway Controller をデプロイする	87
NetScaler Kubernetes Gateway ControllerをOpenShiftのCPXにNetScaler Operatorを使用して デプロイする	102
前提条件	102
NetScaler Kubernetes Gateway ControllerをNetScaler CPXのサイドカーとしてデプロイす る	102
ネットワーク	120
Kubernetes ノードとKubernetes クラスター外のNetScaler インスタンス間のネットワークを確 立する	120
単一のNetScalerの背後にある複数のクラスター	120
トラフィック管理	122
NetScaler IPAM Controller を介した動的 IP アドレス割り当て	122
前提条件	122

構成	123
ステップ 1: GatewayClass を作成する	123
ステップ 2: Gateway を作成する	123
IPアドレスの割り当てを確認する	125
静的IP 対 IPAM	125
アノテーションリファレンス	126
トラブルシューティング	126
制限事項	127
Gateway内の複数のリスナー	127
設定例	128
名前空間フィルター	130
HTTPRouteフィルターによるトラフィック管理機能の強化	130
extensionRefを使用したNetScaler CRDとの統合	131
参照例	131
標準 HttpRoute API フィルターのネイティブサポート	132
URL書き換え	132
RequestHeaderModifier および ResponseHeaderModifier	133
RequestRedirect	134
RequestMirror	135
NetScaler® プロファイルサポート	136
プロファイルCRDのデプロイ	136
プロファイル仕様	137
NetScaler Kubernetes Gateway ControllerでのNetScalerプロファイルの追加	138
フロントエンドプロファイル	138
バックエンドプロファイル	139
NetScalerプロファイルの種類	141
SSLプロファイル	141
TCPプロファイル	149

HTTPプロファイル	155
<code>spec.gatewayClassName</code> によるコントローラーのスコープ設定	162
トラフィックセキュリティ	164
NetScaler Kubernetes Gateway Controller で事前設定された証明書を使用する	164
利点	164
事前設定された証明書を参照する方法	164
証明書の種類	164
基本設定	165
一般的なシナリオ	167
シナリオ1：単一証明書	167
シナリオ 2: 複数ドメイン証明書	168
シナリオ 3: CA証明書によるクライアント認証	169
シナリオ 4: 証明書バンドルの使用	170
シナリオ 5: 事前設定済み証明書とKubernetes証明書の組み合わせ	170
前提条件	171
トラブルシューティング	171
概要	172
NetScaler® で NetScaler Kubernetes Gateway Controller を使用した証明書キーバンドル	173
注意点	173
「appProtocol」を使用してバックエンドアプリケーション通信プロトコルを定義する	174
AIゲートウェイ	175
概要	175
AI Gateway を使用するタイミング	175
展開オプション	175
アーキテクチャ	176
主要なKubernetesリソース	176
関連トピック	177
リファレンス	177

AI機能を持つNetScaler Kubernetes Gateway Controllerを展開する	177
前提条件	178
NetScaler VPX/MPX/BLX を使用して NetScaler Kubernetes Gateway Controller を展開する	178
NetScaler Kubernetes Gateway Controllerをインストールする	178
GatewayClassを作成する	179
Gateway Controllerを使用してNetScaler CPXをデプロイする	179
Gateway Controllerを使用してNetScaler CPXをインストールする	179
GatewayClassを作成する	180
デプロイの検証	181
参照	181
モデルルーティングを設定する	181
前提条件	181
構成	181
AI Gateway プロファイル	181
AI Gateway プロファイルを持つ Kubernetes Gateway	184
NetScaler AI Gateway ルート	186
参照	188
レート制限とストリームを構成する	189
ゲートウェイごとのレート制限とモデルごとのレート制限	189
設定	190
モデルごとのレート制限を適用する	190
ゲートウェイレベルのレート制限を適用する	195
参照	198
プロンプト管理を設定する	199
ユースケース	199
プロンプト管理のためのRewritePolicy CRD	199
Gatewayごとのリライトポリシーとモデルごとのリライトポリシー	199
設定	200

モデルごとの書き換えポリシー	200
Kubernetes Gatewayでの書き換えポリシー	202
参照	203
代替モデルへのスピルオーバーを設定する	203
仕組み	204
主要な設定要素	204
設定	205
スピルオーバーバックエンドプロファイル	205
スピルオーバー書き換えポリシー	206
スピルオーバーサービスを使用する AI Gateway ルート	207
参照	209
可観測性	210
前提条件	210
設定の概要	210
グローバルプロファイルとルートレベルプロファイル	211
Splunkトークンシークレットを作成する	211
Helm値でSplunk接続を構成する（必須）	211
ルートごとにアナリティクスプロファイルをオーバーライドする	213
カスタムAIゲートウェイメトリクス	215
NetScaler CPX	215
NetScaler VPX/MPX/BLX	216
確認	216
リファレンス	217
NetScaler CRD	218
AI Gateway プロファイル	218
aigatewayprofile CRD を展開する	219
AIゲートウェイプロファイル CRD 属性	219
config 属性	220

authtoken属性	220
プロファイル構成の記述方法	221
プロファイル構成の例	221
フロントエンドプロファイル	221
トークンクォータ付きのバックエンドプロファイル	222
事前構成済みプロファイル	222
関連資料	223
AI Gateway ルート	223
aigatewayroute CRD を展開する	224
aigatewayroute カスタムリソース定義の属性	224
エーアイゲートウェイプロファイル属性	225
parentRefs 属性	226
ルール属性	226
ルート設定の記述方法	230
ルート設定の例	230
複数のバックエンドへのモデルルーティング	230
優先度ベースのモデルスピルオーバー	232
ルールにレート制限ポリシーをアタッチする	233
関連資料	234
レート制限	235
ratelimit CRDを展開する	235
レートリミット CRD 属性	236
targetRef 属性	237
selector_keys 属性	237
ratelimits属性	238
ストリーム属性	239
レート制限設定の記述方法	241
レート制限設定の例	241

トークンベースのレート制限	241
ロギング付きのAPIクライアントごとのリクエストレート制限	242
トラフィック分析用のストリーム識別子	244
関連情報	245
リライトおよびレスポnderポリシー	245
rewritepolicy CRDを展開する	245
リライトポリシー CRD 属性	246
リライトポリシー	247
レスポnderポリシー	249
データセット	250
パットセット	251
文字列マップ	251
HTTPコールアウト	252
ポリシー構成の記述方法	253
ポリシー構成の例	254
応答ヘッダーを追加するリライトポリシー	254
リクエストをリダイレクトするレスポnderポリシー	255
カスタム応答を持つレスポnderポリシー	255
データセットを持つリライトポリシー	256
関連資料	257
fix	258

リリースノート

リリースノートには、特定のビルドでソフトウェアがどのように変更されたか、およびビルドに存在する既知の問題が記載されています。リリースノートドキュメントには、次のセクションのすべてまたは一部が含まれています。

- 新機能: ビルドでリリースされた機能強化およびその他の変更。
- 修正された問題: ビルドで修正された問題。
- 既知の問題: ビルドに存在する問題。

以下は、NetScaler® Kubernetes Gateway Controller のリリースリストです。

- [リリース 2.2.0](#)
- [リリース 2.1.0](#)
- [リリース 2.0.0](#)
- [リリース 1.4.0](#)
- [リリース 1.3.0](#)
- [リリース 1.2.0](#)
- [リリース 1.1.0](#)
- [リリース 1.0.0](#)

リリース 2.2.0

このリリースノートドキュメントでは、リリース 2.2.0 に存在する機能強化と変更点、修正された既知の問題について説明します。

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 リリース 2.2.0

新機能

NetScaler AI Gateway のサポート

NetScaler Kubernetes Gateway Controller リリース 2.2.0 以降、NetScaler AI Gateway 機能がサポートされます。NetScaler AI Gateway の詳細については、[AI Gateway](#) を参照してください。

Helm チャートリリース

Helm チャートリリースに関する情報については、[Helm Chart Release Notes](#) を参照してください。

リリース 2.1.0

このリリースノートドキュメントでは、リリース 2.1.0 の機能強化と変更点、修正済みおよび既知の問題について説明します。

NetScaler® クバネティス ゲートウェイ コントローラー リリース 2.1.0

新機能

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 を使用した Kubernetes Gateway API を介して、TCP/UDP ベースのアプリケーションの公開をサポートします。

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。

修正された問題

- 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` が指定されている場合に、`HTTPProfile` CRD コントローラで問題が確認されました。プロファイルで指定された `GatewayClass` 名により、「ref_manager」属性がないというエラーが発生する可能性があります。
- 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、ヘッダーがすでに存在する場合にのみ機能します。
- `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ルートは、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ています。

- コントローラの再起動後、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` が観測されます。

Helmチャートのリリース

Helmチャートのリリースに関する情報は、[Helm Chart Release Notes](#)を参照してください。

リリース 2.0.0

このリリースノートドキュメントでは、リリース 2.0.0 に存在する機能強化と変更点、修正された問題と既知の問題について説明します。

ネットスケラー® キューバネティス ゲートウェイ コントローラー リリース 2.0.0

新機能

NetScaler Kubernetes Gateway Controller での事前設定された証明書の使用

NetScaler アプライアンスにすでに構成されている証明書または証明書バンドルを Kubernetes Gateway で使用できるようになりました。この機能により、既存の証明書を参照できるようになることで、Kubernetes Secret として管理する代わりに証明書管理が簡素化されます。

詳細については、[NetScaler Kubernetes Gateway Controller での事前設定された証明書の使用](#)を参照してください。

Kubernetes ノードと Kubernetes クラスター外の NetScaler インスタンス間のネットワーク確立のサポート

Kubernetes クラスターにデプロイされたサービスがシームレスに機能するために、Ingress NetScaler デバイスは、Pod が実行されてい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。この接続は、次のいずれかの方法で実現できます。

1. NetScaler Kubernetes Gateway Controller で自動ルート構成を有効にします。

コントローラースペックで `gatewayController.nodeWatch: true` を設定して、各 Pod サブネットの NetScaler 上にルートを作成します。

注:

NetScaler SNIP アドレスが Kubernetes ノードと同じサブネットを共有している場合に、このオプションを使用できます。

2. Kubernetes ノードからサブネットルートを駆動するために NetScaler Node Controller をデプロイします。

注:

`gatewayController.nodeWatch` はデフォルトで `false` です。

詳細については、[Kubernetes ノードと Kubernetes クラスター外の NetScaler インスタンス間のネットワークを確立する](#)を参照してください。

`httproute` で `RequestMirroring` フィルターが定義されている場合、NetScaler Kubernetes Gateway コントローラーはバックエンドサービスグループバインディングを追加しません。

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 Kubernetes Gateway 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 Kubernetes Gateway 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的な名前を保持します。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を超過していました。

NetScaler Kubernetes Gateway Controller 2.0.0 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、完全なトレーサビリティを維持します。

詳細については、[CRD のエンティティ名の変更](#)を参照してください。

Helmチャートのリリース

Helmチャートのリリースに関する情報については、[Helm Chart Release Notes](#)を参照してください。

リリース 1.4.0

このリリースノートドキュメントでは、リリース1.4.0の機能強化と変更点、修正済みおよび既知の問題について説明します。

NetScaler® クバネティス ゲートウェイ コントローラー バージョン 1.4.0 のリリース

修正された問題

NetScaler Kubernetes Gateway controller は、`RequestMirroring` フィルターが `httproute` で定義されている場合、バックエンドサービスグループのバインディングを追加しません。

Helm チャートのリリース

Helm チャートのリリースに関する情報については、[Helm Chart Release Notes](#) を参照してください。

リリース 1.3.0

このリリースノートドキュメントでは、リリース 1.3.0 に存在する機能強化と変更点、修正された既知の問題について説明します。

NetScaler® キューバネティス ゲートウェイ コントローラー リリース 1.3.0

新機能

NetScaler プロファイルのサポート

NetScaler Kubernetes Gateway Controller は、堅牢で一元化された構成を実現するために、プロファイル CRD（SSL プロファイル、HTTP プロファイル、TCP プロファイルを含む）をサポートしています。これらのプロファイルにより、クラスターオペレーターは、TLS 暗号、TCP 最適化、HTTP タイムアウトなどの標準化された再利用可能なインフラストラクチャ設定を定義でき、アプリケーション開発者はそれらをルーティング定義に簡単に適用できます。

詳細については、[NetScaler プロファイルのサポート](#)を参照してください。

バックエンドアプリケーション通信プロトコルを定義するための「appProtocol」の使用のサポート

NetScaler Kubernetes Gateway Controller は、Kubernetes サービスポートの appProtocol フィールドを使用して、NetScaler とバックエンドアプリケーション間の通信を制御する、結果として得られる NetScaler サービスグループのプロトコルを明示的に設定します。

たとえば、`appProtocol: ssl` をサービスポートに設定すると、NetScaler 上の対応するサービスグループが SSL プロトコルで構成されます。明示的に設定されていない場合、デフォルトのサービスグループプロトコルは HTTP です。

詳細については、[バックエンドアプリケーション通信プロトコルを定義するための「appProtocol」の使用](#)を参照してください。

Helm チャートリリース

Helm チャートリリースに関する情報については、[Helm Chart Release Notes](#)を参照してください。

リリース 1.2.0

このリリースノートドキュメントでは、リリース1.2.0の機能強化と変更点、修正済みおよび既知の問題について説明します。

ネットスケラー クバネティス ゲートウェイ コントローラー リリース 1.2.0

新機能

クロスネームスペースルーティングのサポート

Gatewayリソースの各リスナー定義内で `allowedRoutes` フィールドを設定することで、クロスネームスペースルーティングを有効にできるようになりました。`allowedRoutes` フィールドは、特定のリスナーにルートをアタッチすることを許可するネームスペースをきめ細かく制御し、柔軟で安全なマルチテナントまたはマルチチーム展開を可能にします。

詳細については、「[ネームスペースフィルター](#)」を参照してください。

RequestRedirectおよびRequestMirrorフィルターのサポート

NetScaler® Kubernetes Gateway Controllerは、以下のフィルターをサポートするようになりました。

- **RequestRedirect** - HTTP 3xxリダイレクト応答を発行し、ユーザーを異なるURLにリダイレクトしたり、HTTPSを強制したり、移動したコンテンツを適切に管理したりします。
- **RequestMirror** - リクエストのコピーを異なるバックエンドサービスに送信します。これはトラフィックシャドウイングに非常に役立ちます。このフィルターを使用すると、クライアントの応答に影響を与えることなく、新しいサービスバージョンのテスト、分析の実行、問題のデバッグを行うことができます。ミラーリングされたバックエンドからの応答は無視されます。

詳細については、「[RequestRedirect](#)」および「[RequestMirror](#)」を参照してください。

CPX と クバネティス ゲートウェイ コントローラー

Kubernetes Gateway Controllerを使用してNetScaler CPXを展開できるようになりました。詳細については、「[NetScaler CPX Kubernetes Gateway Controllerの展開](#)」を参照してください。

Helmチャートリリース

Helmチャートのリリースに関する情報については、[Helm Chart Release Notes](#)を参照してください。

リリース 1.1.0

このリリースノートドキュメントでは、リリース1.1.0の機能強化と変更点、修正済みおよび既知の問題について説明します。

ネットスケーラー キューバネティス ゲートウェイ コントローラー リリース 1.1.0

新機能

HTTPRouteフィルターによるトラフィック管理機能の強化

NetScaler® Kubernetes Gateway Controllerは、Kubernetes Gateway APIで定義されているHTTPRouteフィルターをサポートしています。この機能により、HTTPRouteリソース内で直接、高度なトラフィック操作ロジックを実装できます。フィルターを活用することで、リクエストとレスポンスを変更し、ヘッダー操作、URL書き換え、リクエストリダイレクトなど、幅広いユースケースを可能にします。

詳細については、「[HTTPRouteフィルターによるトラフィック管理機能の強化](#)」を参照してください。

HTTPルートによる複数のリスナーのサポート

NetScaler Kubernetes Gateway Controllerは、Kubernetes Gateway APIで定義されているように、単一のGatewayリソース内で複数のリスナーをサポートします。この機能により、Kubernetesクラスター内のサービスへの外部トラフィックの管理とルーティングに関して、より高い柔軟性と制御が提供されます。

詳細については、「[HTTPルートによる複数のリスナー](#)」を参照してください。

NetScaler Kubernetes Gateway Controllerを使用したNetScalerでの証明書キーバンドルのサポート

NetScaler Kubernetes Gateway Controllerは、NetScaler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、「[NetScaler Kubernetes Gateway Controllerを使用したNetScalerでの証明書キーバンドル](#)」を参照してください。

Helmチャートリリース

Helmチャートリリースについては、「[Helm Chart Release Notes](#)」を参照してください。

リリース 1.0.0

このリリースノートドキュメントでは、リリース 1.0.0 の機能強化と変更点、修正済みおよび既知の問題について説明します。

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 リリース 1.0.0

新機能

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

NetScaler Kubernetes Gateway Controller は、Gateway API を使用して NetScaler の Application Delivery Controller™ (ADC) 機能を Kubernetes と統合し、Kubernetes 環境向けに高度なトラフィック管理、柔軟性の向上、および安全なアプリケーション配信を提供します。

主要な機能、利点、および Kubernetes 環境での安全で効率的なアプリケーション配信を可能にする方法については、[NetScaler Kubernetes Gateway Controller](#) を参照してください。

Helm チャート リリース 1.0.0

Helm チャートのリリースについては、`<https://github.com/netScaler/netScaler-helm-charts/releases/tag/nsk8sgwctr-1.0.0>` を参照してください。

概要

概要には以下のトピックがあります。

- [Kubernetes における ネットスケーラー クーバネティス ゲートウェイ コントローラー](#)
- [はじめに\(/ja-jp/netScaler-kubernetes-gateway-controller/getting-started.html\)](#)
- [ネットスケーラー クーバネティス ゲートウェイ コントローラー CRD](#)
- [サポートされているプラットフォームと展開\(/ja-jp/netScaler-kubernetes-gateway-controller/support-matrix.html\)](#)

NetScaler® Kubernetes Gateway コントローラー

Kubernetesは、ServicesやIngressなど、アプリケーションを公開するための様々なメカニズムを提供します。Ingressは広く採用されていますが、表現力、拡張性、高度なトラフィック管理機能のサポートにおいて制限があります。

Kubernetes Gateway APIは、これらの制限に対処するために設計された次世代APIです。これは、Kubernetesサービスへの外部アクセスを管理するための、より柔軟で、拡張性があり、ロール指向のアプローチを提供します。Gateway APIの主な利点は次のとおりです。

- ロールベースのリソースモデル: 異なるロール間 (例: インフラストラクチャプロバイダー、クラスターオペレーター、アプリケーション開発者) で設定の責任を分離します。
- 強化された表現力: 高度なルーティングシナリオ、トラフィック分割、ヘッダーベースのルーティングなどをサポートします。
- 拡張性: GatewayClassのパラメーターとポリシーを通じて、カスタム拡張機能と機能を可能にします。
- ポータビリティ: 異なるGateway実装間でのより良いポータビリティを目指します。

As organizations increasingly adopt Kubernetes for deploying and managing their applications, the need for robust and scalable solutions for handling external traffic becomes paramount. NetScaler, a leading Application Delivery Controller™ (ADC), provides a powerful and feature-rich solution, seamlessly integrating with Kubernetes by implementing the Kubernetes Gateway API.

このドキュメントでは、NetScaler Kubernetes Gateway APIの概要を説明し、その主要な機能、利点、およびKubernetes環境で安全かつ効率的なアプリケーション配信を可能にする方法について概説します。

Kubernetes Gateway APIプロバイダーとしてのNetScaler

NetScalerは、主要なADCとしての長年の経験に基づいて、Gateway APIの堅牢で成熟した実装を提供します。この統合により、組織はKubernetesクラスター内でNetScalerの高度な機能を利用できます。NetScaler Kubernetes Gateway APIの実装は、次の主要コンポーネントに依存しています。

- **NetScaler Kubernetes Gateway コントローラー:** NetScaler Ingress ControllerはIngressオブジェクトをNetScaler構成に変換し、NetScaler Kubernetes GatewayはGateway APIオブジェクトをNetScaler構成に変換します。NetScaler Kubernetes Gatewayは、Kubernetes APIサーバーでGateway APIリソースの変更を継続的に監視します。NetScaler Kubernetes Gatewayは、NetScaler Ingress Controllerとは別のデプロイメントです。IngressとGateway APIの両方の機能が必要な場合は、NetScaler Ingress ControllerとNetScaler Kubernetes Gatewayの両方をデプロイする必要があります。
- **カスタムリソース定義 (CRD):** NetScalerは、Kubernetes APIを拡張してNetScaler固有の構成と機能を表現するカスタムリソース定義を導入します。これらのCRDは、標準のGateway APIリソースと連携します。
- **Gateway APIリソース:** NetScalerは、GatewayClass、Gateway、HTTPRoute、TCPRoute、TLSRouteなどのコアGateway APIリソースをサポートします。これらのリソースは、外部トラフィックをどのように処理し、バックエンドサービスにルーティングするかを定義します。

ユースケース

NetScaler Kubernetes Gateway API の実装は、以下を含む幅広いユースケースを可能にします。

- **マイクロサービスを安全に公開する:** NetScaler の高度なセキュリティ機能で、Kubernetes にデプロイされたマイクロサービスを保護します。
- **外部トラフィックの負荷分散:** 高可用性とパフォーマンスのために、Kubernetes サービスの複数のインスタンスに外部トラフィックを分散します。
- **高度なルーティングルールの実装:** ヘッダー、パス、その他の基準に基づいて複雑なルーティングルールを定義します。
- **TLS終端の有効化:** パフォーマンスとセキュリティを向上させるために、TLS の暗号化と復号化を NetScaler にオフロードします。
- **カナリアデプロイメントの実装:** 完全なデプロイメントの前に、新しいアプリケーションバージョンを一部のユーザーに段階的に展開します。
- **A/Bテストのためのトラフィック分割:** A/Bテストのために、異なる割合のトラフィックを異なるアプリケーションバージョンに振り向けます。

- **IDプロバイダーとの統合:** NetScaler とさまざまな ID プロバイダーとの統合を使用して、事前認証と認可を実装します。

はじめに

NetScaler (MPX/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は、NetScaler Kubernetes Gateway Controller を Kubernetes に展開できます。

前提条件

- Kubernetes クラスターがインストールされていること。
- 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、[Helm チャート](#)を参照してください。
- [公式標準チャンネル](#)から Gateway API CRD がインストールされていること。

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

NetScaler MPX、VPX、または BLX を構成する Kubernetes クラスターに NetScaler Kubernetes Gateway Controller を展開するには、次の手順を実行します。

1. ユースケースに基づいて `values.yaml` を生成します。NetScaler Kubernetes Gateway Controller のインストール中に構成できる必須パラメーターとオプションパラメーターについては、[構成](#)を参照してください。

```
netscaler:

  adcCredentialSecret: "nslogin"

  nsIP: "<nsIP>" # NetScaler Management IP or SNIP with management
  access enabled for HAor CLIP for

                # NetScaler Cluster

gatewayController:

  entityPrefix: gwy

  gatewayControllerName: "citrix.com/nsgw-controller"

license:
```

```
accept: yes
```

Kubernetes Gateway には、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が必要です。NetScaler Kubernetes Gateway Controller が NetScaler MPX または VPX を構成できるように、NetScaler に必要な権限を持つデフォルト以外のシステムユーザーアカウントがあることを確認してください。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 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。

ユーザー名とパスワードを直接提供するか、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には、次のコマンドで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>
```

注:

`entityPrefix` と `gatewayControllerName` は必須であり、NetScaler Kubernetes Gateway Controller のすべての展開で一意である必要があります。

2. Helm チャートと前の手順で生成された `values.yaml` を使用して、NetScaler Kubernetes Gateway Controller を展開します。詳細については、[Helm チャートを使用した NetScaler Kubernetes Gateway Controller の展開](#)を参照してください。

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/  
  
helm install gateway-controller netscaler/netscaler-kubernetes-gateway-controller -f values.yaml
```

3. インストールを確認します。

```
kubectl get pods
```

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

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

例:

```
kubectl describe pod netscaler-kubernetes-gateway-  
controller-5678d66d9b-nqp6p
```

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

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

例:

```
kubectl logs -f netscaler-kubernetes-gateway-controller-5678d66d9b-  
nqp6p
```

6. (オプション) インストールを削除します。

```
helm uninstall gateway-controller
```

KubernetesクラスターでNetScaler Kubernetes Gateway ControllerとNetScaler CPXをデプロイする方法

NetScaler CPXを構成するKubernetesクラスターにNetScaler Kubernetes Gateway Controllerをデプロイするには、次の手順を実行します。

1. ユースケースに基づいて `values.yaml` を生成します。NetScaler CPXとKubernetes Gateway Controller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ーについては、[構成](#)を参照してください。

```
gatewayController:  
  
  entityPrefix: gwy  
  
  gatewayControllerName: "citrix.com/nscpxgw-controller"  
  
license:  
  
  accept: yes
```

```
service:

  annotations: {}

  spec:

    type: NodePort

    ports:

      - port: 80

        targetPort: 80

        protocol: TCP

        name: http

      - port: 443

        targetPort: 443

        protocol: TCP

        name: https
```

注:

`entityPrefix` と `gatewayControllerName` は必須であり、NetScaler CPXとKubernetes Gateway Controllerのすべてのデプロイメントで一貫である必要があります。

2. Helmチャートと前のステップで生成された `values.yaml` を使用して、NetScaler CPXとKubernetes Gateway Controllerをデプロイします。詳細については、[Helmチャートを使用したNetScaler CPXとKubernetes Gateway Controllerのデプロイ](#)を参照してください。

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

```
helm install cpx-gateway-controller netScaler/netScaler-cpx-with-gateway-controller -f values.yaml
```

3. インストールを確認します。

```
kubectl get pods
```

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

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

例:

```
kubectl describe pod cpx-gateway-controller-netScaler-cpx-with-gateway-controller1z6ghg
```

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

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

例:

```
kubectl logs -f cpx-gateway-controller-netScaler-cpx-with-gateway-controller1z6ghg -c nskgwc
```

6. (オプション) インストールを削除します。

```
helm uninstall cpx-gateway-controller
```

次は何をしますか

- [NetScaler Kubernetes Gateway Controllerを展開する。](#)
- [NetScaler CPXをKubernetes Gateway Controllerと共に展開する。](#)

サポートされているGateway API CRD

以下のセクションでは、最新リリースバージョンにおける各Kubernetes Gateway API CRDでサポートされているパラメータの概要を説明します。このトピックに記載されているサポート対象外のパラメータまたはCRDは、現在サポートされていません。

ゲートウェイクラスCRD

プロパティ	説明	ステータス
spec.controllerName	コントローラーのデプロイ時に指定された名前と一致する必要があるコントローラーの名前。	サポート対象
spec.description	GatewayClassの人間が読める説明。	サポート対象
status.conditions	コントローラーからのこのGatewayClassのステータスは、「Conditions」で確認できます。	サポート対象

ゲートウェイ CRD

プロパティ	説明	ステータス
spec.addresses	IPアドレスのリスト。コントローラーはすべてのアドレスを使用し、ipsetを作成します。コントローラーは、NetScaler上に作成されたコンテンツスイッチング仮想サーバーの仮想IPアドレスとしてipsetを使用します。	サポート対象
spec.gatewayClassName	このGatewayが属するGatewayClassの名前。	サポート対象
spec.listeners	Gatewayが公開するネットワークエンドポイント（リスナー）のリスト。	サポート対象

プロパティ	説明	ステータス
spec.listeners.name	リスナーの名前。この名前は、Gateway内で一意である必要があります。	サポートされています
spec.listeners[].ポート	リスナーのポート番号。コントローラーは、このポートをNetScaler上に作成されたコンテンツスイッチング仮想サーバーポートとして使用します。	サポートされています
spec.listeners[].プロトコル	リスナーが受信を期待するネットワークプロトコル。コントローラーは、このプロトコルをNetScaler上に作成されたコンテンツスイッチング仮想サーバーの仮想サーバーIPアドレスとして使用します。	サポートされています
spec.listeners[].tls.certificateRefs	CertificateRefsには、TLS証明書と秘密鍵を含むKubernetesオブジェクトへの一連の参照が含まれています。現在、コントローラーはKubernetesシークレットのみをサポートしています。	部分的にサポートされています

HTTPRoute カスタムリソース定義

プロパティ	説明	ステータス
spec.hostnames	このルートが一致するホスト名のリスト。コントローラーは、コンテンツスイッチングポリシーのルール式でホスト名を使用します。	サポートされています
spec.parentRefs[]	このルートがアタッチするGatewayへの参照。	部分的にサポートされています

プロパティ	説明	ステータス
spec.parentRefs[].種類	コントローラーは、以下の parentRefsのみをサポートします: Gateway	部分的にサポートされています
spec.parentRefs[].名前	Gatewayリソースの名前。	サポートされています
spec.parentRefs[].ネームスペース	Gatewayリソースの名前空間。	サポートされています
spec.rules[]	コントローラーは、コンテンツスイッチングポリシーのルール式を生成するためにこれらのルールを使用します。	サポートされています
spec.ルール[].一致	コントローラーは、パス、メソッド、クエリパラメータ、ヘッダーなどを使用して、NetScaler 上でコンテンツスイッチングポリシーのルールを作成します。	サポートされています
spec.rules[].backendRefs	現在、コントローラーは以下の backendRefs のみをサポートしています: service、name、namespace、および port。	部分的にサポートされています
spec.ルール[].フィルター	コントローラーは以下のフィルターをサポートしています: urlRewrite、requestRedirect、responseHeaderModifier、requestHeaderModifier、extensionRef	部分的にサポートされています

TCPRoute CRDについて

TCPRoute は、Gateway リスナーからバックエンドの Kubernetes サービスへの TCP トラフィックルーティングを可能にします。

プロパティ	説明	ステータス
spec.parentRefs[]	このルートがアタッチする Gatewayへの参照	部分的にサポート
spec.parentRefs[].種類	現在、コントローラーは以下の parentRefsのみをサポートしています: Gateway	サポート済み
spec.parentRefs[].名前	Gatewayリソースの名前。	サポート済み
spec.親参照[].名前空間	Gatewayリソースの名前空間。	サポート済み
spec.parentRefs[].sectionName	アタッチする特定のGatewayリスナーの名前。	サポートされています
spec.rules[]	TCPトラフィックがバックエンドサービスにルーティングされる方法を定義するルールの一覧。	サポートされています
spec.rules[].backendRefs[]	現在、コントローラーは以下の backendRefsのみをサポートしています: service、name、namespace、port。	部分的にサポートされています

UDPRoute カスタムリソース定義

UDPRouteは、GatewayリスナーからバックエンドのKubernetesサービスへのUDPトラフィックルーティングを可能にします。

プロパティ	説明	ステータス
spec.parentRefs[]	このルートがアタッチする Gatewayへの参照	部分的にサポートされています
spec.parentRefs[].kind フィールド	現在、コントローラーは以下の parentRefsのみをサポートしています: Gateway	対応済み
spec.parentRefs[]のname	Gatewayリソースの名前。	対応済み

プロパティ	説明	ステータス
仕様.親参照[].名前空間	Gatewayリソースの名前空間。	対応済み
spec.parentRefs[].sectionName	アタッチする特定のGatewayリスナーの名前。	対応済み
spec.rules[]	UDPトラフィックがバックエンドサービスにルーティングされる方法を定義するルールの一覧。	対応済み
spec.rules[].backendRefs[]	現在、コントローラーは以下のbackendRefsのみをサポートしています: service, name, namespace, port	部分的にサポートされています

注意点

- コントローラーは、以前にリストされたサポートされているパラメーターのみを考慮し、サポートされていないパラメーターは無視されます。
- 将来のバージョンでは、追加のパラメーターのサポートが拡張される可能性があります。
- コントローラーによって認識されるように、Gatewayリソースに対して `gatewayClassName` が正しく設定されていることを確認してください。
- `HTTPRoute` 内のすべての `parentRefs` は、既存の有効なGatewayを正しく参照している必要があります。
- ステータス更新は動的に管理され、システム負荷に基づいて反映されるまでに時間がかかる場合があります。
- CRDを適用する際には、解析エラーや検証エラーがないかログを監視する必要があります。
- Gateway CRDで使用されるIPアドレスは、IngressまたはCitrix® Listenerリソースと共有することはできません。

サポートされているプラットフォームと展開

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NetScaler Kubernetes Gateway Controller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 Kubernetes Gateway Controller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 v1.27（以降）。
- グーグル・クーバネティス・エンジン (GKE)
- エラスティック・クーバネティス・サービス (EKS)
- アジュール・クーバネティス・サービス (AKS)
- Red Hat OpenShift バージョン 4.15 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・クーバネティス・プラットフォーム
- ミランティス・クーバネティス・エンジン
- ブイエムウェア・タンズ
- ランチャー

NetScalerとの互換性

以下の表は、任意のNetScaler Kubernetes Gateway 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。

NETSCALER KUBERNETES GATEWAY コントローラー	NETSCALERのバージョン
2.2.0	VPX/MPX/CPX 14.1-73.x 以降
2.1.0	VPX/MPX/CPX 14.1-66.59
2.0.0	VPX/MPX/CPX 14.1-60.57
1.4.0	VPX/MPX/CPX 14.1-60.57
1.3.0	VPX/MPX/CPX 14.1-47.48
1.2.0	VPX/MPX/CPX 14.1-47.48
1.1.0	VPX/MPX/CPX 14.1-43.50
1.0.0	VPX/MPX/CPX 14.1-38.53

NetScaler オペレーターと NetScaler Kubernetes Gateway コントローラーの互換性

この表は、任意のNetScaler Operatorリリースを使用して展開できるNetScaler Kubernetes Gateway Controllerリリースを示しています。

OPENSIFTバージョン	NETSCALER OPERATOR バージョン	ネットスケラー クバネティス ゲートウェイ コントローラー
4.11以降	3.6.0	1.4.0
4.11以降	3.5.0	1.4.0
4.11 以降	3.4.1	1.4.0
4.11 以降	3.3.0	1.3.0

インストール

インストールには、次のトピックがあります。

- [NetScaler Kubernetes Gateway Controller をデプロイする](#)
- [NetScaler CPX を Kubernetes Gateway Controller と共にデプロイする](#)

NetScaler® Kubernetes Gateway Controller を展開する

展開プロセスには、トラフィックルーティングルールを定義する Gateway CRD を監視および管理する Gateway Controller の展開が含まれます。Gateway Controller は、外部リクエストをサンプルアプリケーションに誘導するように NetScaler VPX™ を構成します。



1. 開始するには、サンプルアプリケーションを展開します。

```
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

        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

  appProtocol: https

selector:

  name: cnn-website
```

2. Gateway API CRD がまだインストールされていない場合は、[公式標準チャンネル](#) からインストールします。
3. ユースケースに基づいて `values.yaml` を生成します。NetScaler Kubernetes Gateway Controller 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関する情報については、[構成](#) を参照してください。

```
netScaler:

  adcCredentialSecret: "nslogin"

  nsIP: "4.4.4.10" # NetScaler Management IP or SNIP with Management
  access enabled for HA and CLIP for NetScaler Cluster

gatewayController:

  entityPrefix: gwy

  gatewayControllerName: "citrix.com/nsgw-controller"
```

```
license:  
  
  accept: yes
```

Kubernetes Gateway には、NetScaler MPX または NetScaler VPX 上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が必要です。NetScaler Kubernetes Gateway Controller が NetScaler MPX または VPX を構成できるように、NetScaler には必要な権限を持つデフォルト以外のシステムユーザーアカウントがあることを確認してください。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 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。

ユーザー名とパスワードを直接提供することも、Kubernetes シークレットを使用することもできます。Kubernetes シークレットを使用するには、次のコマンドで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>
```

注:

`entityPrefix` と `gatewayControllerName` は必須であり、NetScaler Kubernetes Gateway Controller のすべての展開で一貫である必要があります。

4. Helm チャートと、前のステップで生成された `values.yaml` を使用して、NetScaler Kubernetes Gateway Controller を展開します。詳細については、[Helm Chart を使用した NetScaler Kubernetes Gateway Controller の展開](#) を参照してください。

```
helm repo add netscaler https://netscaler.github.io/netscaler-helm-charts/  
  
helm install gateway-controller netscaler/netscaler-kubernetes-gateway-controller -f values.yaml
```

Gateway および HTTPRoute CRD を使用して HTTP 経由でアプリケーションを公開する

以下は、GatewayClass、Gateway、および HTTPRoute の定義の例です。

1. Gateway オブジェクトを処理するコントローラーを定義するために GatewayClass リソースを作成します。

GatewayClass の例

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

kind: GatewayClass

metadata:

  name: example-gateway-class

spec:

  controllerName: citrix.com/nsgw-controller
```

2. 定義されたリスナーに基づいてサービスを公開するために、Gateway リソースを定義します。

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

kind: Gateway

metadata:

  name: example-gateway

spec:

  gatewayClassName: example-gateway-class

  listeners:

    - name: http

      protocol: HTTP

      port: 80
```

```
addresses:

  - type: IPAddress

    value: 4.4.4.4
```

注記:

- Gateway で定義されている gatewayClassName が、前の手順で作成された GatewayClass リソースと一致することを確認します。

3. HTTPRoute またはその他のルート CRD を適用して、バックエンドサービスにトラフィックをルーティングします。

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

kind: HTTPRoute

metadata:

  name: example-route

spec:

  parentRefs:

    - name: example-gateway

  hostnames:

    - "example.com"

  rules:

    - matches:

        - path:

            type: PathPrefix
```

```

    value: /

    backendRefs:

    - name: cnn-website

      namespace: default

    port: 80

```

注記:

- `parentRefs` で指定されたゲートウェイ名が、前の手順で確立された Gateway CRD と一致することを確認します。
- 現在、HTTPRoute CRD は単一の `parentRef` のみをサポートしています。

4. gatewayClass のステータスを確認します。ステータスが `ACCEPTED: True` と表示されている場合、コントローラーはリソースを受け入れています。

```
kubectl get gatewayclass
```

NAME	CONTROLLER	ACCEPTED	AGE
example-gateway-class	citrix.com/nsgw-controller	True	2d

```
kubectl describe gatewayclass
```

```
...
```

```
Spec:
```

```
  Controller Name: citrix.com/nsgw-controller
```

```
Status:
```

```
  Conditions:
```

```
    Last Transition Time: 2025-03-21T09:15:32Z
```

```

Message:          Accepted by the controller.

Reason:           Accepted

Status:           True

Type:             Accepted

Events:           <none>

```

Gateway リソースのステータスを確認します。ステータスが **PROGRAMMED : True** と表示されている場合、コントローラーは設定を NetScaler に正常にプッシュしています。

```
kubectl get gateway example-gateway
```

NAME	CLASS	ADDRESS	PROGRAMMED	AGE
example-gateway	example-gateway-class	4.4.4.4	True	6d3h

```
kubectl describe gateway example-gateway
```

```
...
```

```
Status:
```

```
Addresses:
```

```
  Type:  IPAddress
```

```
  Value:  4.4.4.4
```

```
Conditions:
```

```
  Last Transition Time:  2025-03-24T08:53:07Z
```

```
  Message:              Accepted by the controller.
```

```
  Reason:               CreatedCRDInstance
```

```
Status:                True

Type:                  Accepted

Last Transition Time:  2025-03-24T08:53:18Z

Message:               Config pushed by the controller.

Reason:               ConfigPushedToNetscaler

Status:                True

Type:                  Programmed

Events:               <none>
```

Gateway および HTTPRoute CRD を使用した HTTPS/SSL によるアプリケーションの公開

以下は、GatewayClass、Gateway、および HTTPRoute の定義の例です。

1. コントローラーによって処理される Gateway オブジェクトを定義するために、GatewayClass リソースを作成します。

GatewayClass の例

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

kind: GatewayClass

metadata:

  name: example-gateway-class

spec:

  controllerName: citrix.com/nsgw-controller
```

2. (オプション) TLS 設定のために Gateway-Listener で使用する自己署名 SSL 証明書とキーを作成します。以下のコマンドは、自己署名証明書を作成するための単なるサンプルコマンドであり、アプリケーションのホスト名が `example.com` であることを前提としています。

```
openssl genrsa -out cnnwebsite_key.pem 2048

openssl req -new -key cnnwebsite_key.pem -out cnnwebsite_csr.pem -subj
"/CN=example.com"

openssl x509 -req -in cnnwebsite_csr.pem -sha256 -days 365 -extensions
v3_ca -signkey cnnwebsite_key.pem -CAcreateserial -out
cnnwebsite_cert.pem
```

注:

既存のSSL証明書がある場合は、それを使用してKubernetesシークレットを作成できます。

3. SSL証明書とキー（TLSペア）を保存するためのKubernetesシークレットを作成します。

```
kubectl create secret tls app1-secret --key cnnwebsite_key.pem --cert
cnnwebsite_cert.pem
```

4. HTTP（ポート 80）および HTTPS（ポート 443）用のリスナーを持つGatewayリソースを作成します。Gatewayはアプリケーションを4.4.4.4で公開します。HTTPSリスナー内で、前の手順で作成したSSL証明書シークレットを参照します。

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

kind: Gateway

metadata:

  name: example-gateway

spec:

  gatewayClassName: example-gateway-class

  listeners:

    - name: http

      protocol: HTTP
```

```
    port: 80

  - name: https

    protocol: HTTPS

    port: 443

    tls:

      mode: Terminate

      certificateRefs:

        - kind: Secret

          name: app1-secret

    addresses:

      - type: IPAddress

        value: 4.4.4.4
```

5. HTTPRouteリソースを適用して、受信トラフィックを1つ以上のバックエンドサービスにルーティングします。

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

kind: HTTPRoute

metadata:

  name: example-route

spec:

  parentRefs:

    - name: example-gateway

  hostnames:
```

```
- "example.com"

rules:

- matches:

  - path:

    type: PathPrefix

    value: /

  backendRefs:

  - name: cnn-website

    namespace: default

    port: 443
```

注:

- アプリケーションはポート80 (HTTP) とポート443 (HTTPS) の両方で公開されます。
 - アプリケーションをHTTPSのみで公開するには、GatewayリソースのHTTPリスナー設定を除外します。
 - HTTPをHTTPSにリダイレクトするには、`requestRedirect` HTTPRouteフィルターを使用します。
- バックエンドサービスがGatewayからのHTTPSトラフィックを期待する場合は、Service定義の対応するポート（例：ポート443）でappProtocolがhttpsに設定されていることを確認してください。参照：バックエンドプロトコルに関する詳細は、Gateway APIドキュメントを参照してください。

GatewayおよびTCPRoute CRDを使用してTCPアプリケーションを公開する

このセクションでは、NetScalerをゲートウェイインフラストラクチャとして使用し、Kubernetes Gateway APIを使用してTCPベースのアプリケーションを公開するための段階的な手順を説明します。

ステップ1：サンプルTCPアプリケーションをデプロイする

KubernetesクラスターにサンプルTCPアプリケーション（例：Redisサーバー）をデプロイします。

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: redis-server

  namespace: default

spec:

  replicas: 2

  selector:

    matchLabels:

      app: redis

  template:

    metadata:

      labels:

        app: redis

    spec:

      containers:
```

```
- name: redis

  image: redis:7.0

  ports:

    - containerPort: 6379

---

apiVersion: v1

kind: Service

metadata:

  name: redis-service

  namespace: default

spec:

  selector:

    app: redis

  ports:

    - port: 6379

      targetPort: 6379

      protocol: TCP
```

設定を適用します。

```
kubectl apply -f redis-deployment.yaml
```

ステップ2：GatewayClassを作成する

NetScaler Gateway Controllerを参照するGatewayClassリソースを作成します。

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

kind: GatewayClass

metadata:

  name: netscaler-gateway-class

spec:

  controllerName: citrix.com/netscaler-gateway-controller
```

GatewayClassを適用します。

```
kubectl apply -f gateway-class.yaml
```

ステップ3：TCPリスナーを持つGatewayを作成する

TCPプロトコルリスナーを持つGatewayリソースを作成します。 `addresses` フィールドは、NetScaler上のVIPアドレスを指定します。

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

kind: Gateway

metadata:

  name: tcp-gateway

  namespace: default

spec:

  gatewayClassName: netscaler-gateway-class

  addresses:

    - type: IPAddress

      value: "192.168.1.100"
```

```
listeners:

  - name: redis-listener

    protocol: TCP

    port: 6379

    allowedRoutes:

      namespaces:

        from: Same
```

Gatewayを適用します。

```
kubectl apply -f tcp-gateway.yaml
```

ステップ4：TCPRouteを作成する

Gatewayリスナーにアタッチし、バックエンドサービスにトラフィックをルーティングする **TCPRoute** リソースを作成します。

```
apiVersion: gateway.networking.k8s.io/v1alpha2

kind: TCPRoute

metadata:

  name: redis-route

  namespace: default

spec:

  parentRefs:

    - name: tcp-gateway

    sectionName: redis-listener
```

```
rules:

  - backendRefs:

    - name: redis-service

      port: 6379
```

`TCPRoute` を適用します。

```
kubectl apply -f tcp-route.yaml
```

ステップ5：構成を確認する

1. Gatewayのステータスを確認します。

```
kubectl get gateway tcp-gateway -o yaml
```

1. `TCPRoute` のステータスを確認します。

```
kubectl get tcproute redis-route -o yaml
```

`Accepted` と `Programmed` の両方の条件が `status: "True"` と表示されていることを確認します。

1. TCP接続をテストします。

```
redis-cli -h 192.168.1.100 -p 6379 ping
```

期待される出力: `PONG`

GatewayおよびUDPRoute CRDを使用したUDPアプリケーションの公開

このセクションでは、NetScaler を使用した Kubernetes Gateway API を使って、UDP ベースのアプリケーション (DNS サーバーなど) を公開するための段階的な手順を説明します。

ステップ 1: サンプルの UDP アプリケーションを展開する

Kubernetes クラスターにサンプルの DNS サーバー (CoreDNS) を展開します。

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: coredns-server

  namespace: default

spec:

  replicas: 2

  selector:

    matchLabels:

      app: coredns

  template:

    metadata:

      labels:

        app: coredns

    spec:

      containers:

        - name: coredns

          image: coredns/coredns:1.11.1

          args: ["-conf", "/etc/coredns/Corefile"]

          ports:

            - containerPort: 5353

            protocol: UDP
```

```
    volumeMounts:

      - name: config-volume

        mountPath: /etc/coredns

    volumes:

      - name: config-volume

        configMap:

          name: coredns-config

---

apiVersion: v1

kind: ConfigMap

metadata:

  name: coredns-config

  namespace: default

data:

  Corefile: |

    .:5353 {

      forward . 8.8.8.8

      log

      errors

    }

---

apiVersion: v1
```

```
kind: Service

metadata:

  name: dns-service

  namespace: default

spec:

  selector:

    app: coredns

  ports:

    - port: 5353

      targetPort: 5353

      protocol: UDP
```

構成を適用します。

```
kubectl apply -f coredns-deployment.yaml
```

ステップ 2: GatewayClass を作成する

まだ作成されていない場合は、GatewayClass を作成します。

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

kind: GatewayClass

metadata:

  name: netscaler-gateway-class

spec:
```

```
controllerName: citrix.com/netScaler-gateway-controller
```

ステップ 3: UDP リスナーを持つ Gateway を作成する

UDP プロトコルリスナーを持つ Gateway リソースを作成します。

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

kind: Gateway

metadata:

  name: udp-gateway

  namespace: default

spec:

  gatewayClassName: netScaler-gateway-class

  addresses:

    - type: IPAddress

      value: "192.168.1.101"

  listeners:

    - name: dns-listener

      protocol: UDP

      port: 53

      allowedRoutes:

        namespaces:

          from: Same
```

Gateway を適用します。

```
kubectl apply -f udp-gateway.yaml
```

ステップ 4: UDPRoute を作成する

Gateway リスナーにアタッチする UDPRoute リソースを作成します。

```
apiVersion: gateway.networking.k8s.io/v1alpha2

kind: UDPRoute

metadata:

  name: dns-route

  namespace: default

spec:

  parentRefs:

    - name: udp-gateway

      sectionName: dns-listener

  rules:

    - backendRefs:

        - name: dns-service

          port: 5353
```

UDPRoute を適用します。

```
kubectl apply -f udp-route.yaml
```

ステップ 5: 構成を確認する

1. Gateway のステータスを確認します。

```
kubectl get gateway udp-gateway -o yaml
```

1. UDPRoute のステータスを確認します。

```
kubectl get udproute dns-route -o yaml
```

1. Gateway経由でDNS解決をテストします:

```
dig @192.168.1.101 -p 53 example.com
```

NetScaler® CPX Kubernetes Gateway Controllerのデプロイ

デプロイプロセスには、トラフィックルーティングルールを定義するGateway CRDを監視および管理するGateway Controllerのデプロイが含まれます。その後、Gateway ControllerはNetScaler CPXを構成して、外部リクエストをサンプルアプリケーションに転送します。



1. 開始するには、サンプルアプリケーションをデプロイします。

```
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

        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:

    name: cnn-website
```

- Gateway API CRDがまだインストールされていない場合は、[公式標準チャネル](#)からインストールします。
- ユースケースに基づいて `values.yaml` を生成します。NetScaler Kubernetes Gateway Controller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。

```
gatewayController:

  entityPrefix: gwy

  gatewayControllerName: "citrix.com/nscpxgw-controller"

license:

  accept: yes

service:
```

```
annotations: {}

spec:

  type: NodePort

  ports:

    - port: 80

      targetPort: 80

      protocol: TCP

      name: http

    - port: 443

      targetPort: 443

      protocol: TCP

      name: https

    - port: 6379

      targetPort: 6379

      protocol: TCP

      name: tcp6379

    - port: 5353

      targetPort: 5353

      protocol: UDP

      name: udp5353
```

注：

`entityPrefix` と `gatewayControllerName` は必須であり、NetScaler CPXとKubernetes Gateway Controllerのすべてのデプロイメントで一貫である必要があります。

4. Helmチャートと、前の手順で生成された `values.yaml` を使用して、NetScaler CPXとKubernetes Gateway Controllerをデプロイします。詳細については、[Helmチャートを使用したNetScaler CPXとKubernetes Gateway Controllerのデプロイ](#)を参照してください。

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

```
helm install cpx-gateway-controller netScaler/netScaler-cpx-with-gateway-controller -f values.yaml
```

GatewayおよびHTTPRoute CRDのサンプル

以下は、GatewayClass、Gateway、およびHTTPRoute定義の例です。

1. Gatewayオブジェクトを処理するコントローラーを定義するために、GatewayClassリソースを作成します。

GatewayClassのサンプル

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

kind: GatewayClass

metadata:

  name: example-gateway-class

spec:

  controllerName: citrix.com/nscpxgw-controller
```

2. (オプション) TLS構成のためにGateway-Listenerで使用する自己署名SSL証明書とキーを作成します。

以下のコマンドは、自己署名証明書を作成するための単なるサンプルコマンドであり、アプリケーションのホスト名が `example.com` であると仮定しています。

```
openssl genrsa -out cnnwebsite_key.pem 2048

openssl req -new -key cnnwebsite_key.pem -out cnnwebsite_csr.pem -subj
"/CN=example.com"

openssl x509 -req -in cnnwebsite_csr.pem -sha256 -days 365 -extensions
v3_ca -signkey cnnwebsite_key.pem -CAcreateserial -out
cnnwebsite_cert.pem
```

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。

3. SSL証明書とキー（TLSペア）を保存するためのKubernetes Secretを作成します。

```
kubectl create secret tls app1-secret --key cnnwebsite_key.pem --cert
cnnwebsite_cert.pem
```

4. 定義されたリスナーに基づいてサービスを公開するためのGatewayリソースを定義します。

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

kind: Gateway

metadata:

  name: example-gateway

spec:

  gatewayClassName: example-gateway-class

  listeners:

    - name: http
```

```
    protocol: HTTP

    port: 80

  - name: https

    protocol: HTTPS

    port: 443

    tls:

      mode: Terminate

      certificateRefs:

        - kind: Secret

          name: app1-secret

          group: ""
```

注:

- Gatewayで定義されているgatewayClassNameが、前の手順で作成されたGatewayClassリソースと一致することを確認します。

5. HTTPRouteまたはその他のルートCRDを適用して、バックエンドサービスにトラフィックをルーティングします。

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

kind: HTTPRoute

metadata:

  name: example-route

spec:
```

```

parentRefs:

- name: example-gateway

hostnames:

- "example.com"

rules:

- matches:

- path:

    type: PathPrefix

    value: /

backendRefs:

- name: cnn-website

  namespace: default

  port: 80

```

注:

- `parentRefs` で指定されたゲートウェイ名が、前の手順で確立されたGateway CRD と一致することを確認してください。
- 現在、HTTPRoute CRDは単一の `parentRef` のみをサポートしています。

- gatewayClassのステータスを確認します。ステータスが `ACCEPTED: True` と表示されている場合、コントローラーはリソースを受け入れています。

```
kubectl get gatewayclass
```

NAME	CONTROLLER	ACCEPTED	AGE
------	------------	----------	-----

```
example-gateway-class    citrix.com/nscpxgw-controller    True    2d
```

```
kubectl describe gatewayclass
```

```
...
```

```
Spec:
```

```
Controller Name:    citrix.com/nscpxgw-controller
```

```
Status:
```

```
Conditions:
```

```
    Last Transition Time:    2025-10-16T06:24:51Z
```

```
    Message:                Accepted by the controller.
```

```
    Reason:                 Accepted
```

```
    Status:                 True
```

```
    Type:                   Accepted
```

```
Events:                  <none>
```

Gatewayリソースのステータスを確認します。ステータスが **PROGRAMMED : True** と表示されている場合、コントローラーは設定をNetScalerに正常にプッシュしています。

```
kubectl get gateway example-gateway
```

NAME	CLASS	ADDRESS	PROGRAMMED	AGE
example-gateway	example-gateway-class	10.244.2.28	True	14m

```
kubect1 describe gateway example-gateway
```

Status:

Addresses:

Type: IPAddress

Value: 10.244.2.28

Conditions:

Last Transition Time: 2025-10-16T06:24:53Z

Message: Accepted by the controller.

Reason: CreatedCRDInstance

Status: True

Type: Accepted

Last Transition Time: 2025-10-16T06:25:01Z

Message: Config pushed by the controller.

Reason: ConfigPushedToNetscaler

Status: True

Type: Programmed

Events: <none>

7. トラフィックの場合、CPXのサービスは `nodePort` としてデプロイされているため、`nodeIP:NodePort` を使用してトラフィックを送信できます。

```
kubect1 get svc -o wide
```

NAME	EXTERNAL-IP	PORT(S)	TYPE	AGE	CLUSTER-IP	SELECTOR
cpx-gateway-controller-cpx-service			NodePort		10.104.175.74	
<none>		80:31375/TCP, 443:31994/TCP		11m		app=cpx-gateway-controller-netscaler-cpx-with-gateway-controller

```
curl -vik --resolve example.com:<nodePort>:<nodeIP> https://
example.com:<nodePort>
```

```
curl -vik --resolve example.com:31994:<nodeIP> https://
example.com:31994
```

GatewayおよびTCPRoute CRDを使用してTCPアプリケーションを公開する

このセクションでは、NetScalerとKubernetes Gateway APIを使用してTCPベースのアプリケーションを公開するための段階的な手順を説明します。

ステップ 1: サンプル TCP アプリケーションをデプロイする

Kubernetes クラスターにサンプル TCP アプリケーション (例: Redis サーバー) をデプロイします。

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: redis-server

  namespace: default

spec:

  replicas: 2
```

```
  selector:

    matchLabels:

      app: redis

  template:

    metadata:

      labels:

        app: redis

    spec:

      containers:

        - name: redis

          image: redis:7.0

          ports:

            - containerPort: 6379

---

apiVersion: v1

kind: Service

metadata:

  name: redis-service

  namespace: default

spec:

  selector:

    app: redis
```

```
ports:

  - port: 6379

  targetPort: 6379

  protocol: TCP
```

構成を適用します。

```
kubectl apply -f redis-deployment.yaml
```

ステップ 2: GatewayClass を作成する

NetScaler Gateway Controller を参照する GatewayClass リソースを作成します。

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

kind: GatewayClass

metadata:

  name: netscaler-gateway-class

spec:

  controllerName: citrix.com/netscaler-gateway-controller
```

GatewayClass を適用します。

```
kubectl apply -f gateway-class.yaml
```

ステップ 3: TCP リスナーを持つ Gateway を作成する

TCP プロトコル リスナーを持つ Gateway リソースを作成します。 `addresses` フィールドは NetScaler 上の VIP アドレスを指定します。

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

kind: Gateway
```

```
metadata:

  name: tcp-gateway

  namespace: default

spec:

  gatewayClassName: netscaler-gateway-class

  addresses:

    - type: IPAddress

      value: "192.168.1.100"

  listeners:

    - name: redis-listener

      protocol: TCP

      port: 6379

      allowedRoutes:

        namespaces:

          from: Same
```

Gateway を適用します。

```
kubectl apply -f tcp-gateway.yaml
```

ステップ 4: TCPRoute を作成する

Gateway リスナーにアタッチし、バックエンド サービスにトラフィックをルーティングする TCPRoute リソースを作成します。

```
apiVersion: gateway.networking.k8s.io/v1alpha2

kind: TCPRoute

metadata:

  name: redis-route

  namespace: default

spec:

  parentRefs:

    - name: tcp-gateway

      sectionName: redis-listener

  rules:

    - backendRefs:

        - name: redis-service

          port: 6379
```

TCPRoute を適用します。

```
kubectl apply -f tcp-route.yaml
```

ステップ 5: 構成を確認する

1. Gateway のステータスを確認します。

```
kubectl get gateway tcp-gateway -o yaml
```

1. TCPRoute のステータスを確認します。

```
kubectl get tcproute redis-route -o yaml
```

`Accepted` と `Programmed` の両方の条件が `status: "True"` を示していることを確認します。

1. TCP接続をテストします。

```
redis-cli -h 192.168.1.100 -p 6379 ping
```

期待される出力: `PONG`

Gateway および UDPRoute CRD を使用した UDP アプリケーションの公開

このセクションでは、NetScaler を使用して Kubernetes Gateway API を介して UDP ベースのアプリケーション (DNS サーバーなど) を公開するための段階的な手順を説明します。

ステップ 1: サンプル UDP アプリケーションを展開する

Kubernetes クラスターにサンプル DNS サーバー (CoreDNS) を展開します。

```
apiVersion: apps/v1

kind: Deployment

metadata:

  name: coredns-server

  namespace: default

spec:

  replicas: 2

  selector:

    matchLabels:

      app: coredns

  template:

    metadata:
```

```
    labels:

      app: coredns

  spec:

    containers:

      - name: coredns

        image: coredns/coredns:1.11.1

        args: ["-conf", "/etc/coredns/Corefile"]

        ports:

          - containerPort: 5353

            protocol: UDP

        volumeMounts:

          - name: config-volume

            mountPath: /etc/coredns

    volumes:

      - name: config-volume

        configMap:

          name: coredns-config

---

apiVersion: v1

kind: ConfigMap

metadata:

  name: coredns-config
```

```
    namespace: default

data:

  Corefile: |

    .:5353 {

      forward . 8.8.8.8

      log

      errors

    }

  ---

  apiVersion: v1

  kind: Service

  metadata:

    name: dns-service

    namespace: default

  spec:

    selector:

      app: coredns

    ports:

      - port: 5353

        targetPort: 5353

        protocol: UDP
```

構成を適用します。

```
kubectl apply -f coredns-deployment.yaml
```

ステップ 2: GatewayClass を作成する

まだ作成されていない場合は、GatewayClass を作成します。

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

kind: GatewayClass

metadata:

  name: netscaler-gateway-class

spec:

  controllerName: citrix.com/netscaler-gateway-controller
```

ステップ 3: UDP リスナーを持つ Gateway を作成する

UDP プロトコル リスナーを持つ Gateway リソースを作成します。

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

kind: Gateway

metadata:

  name: udp-gateway

  namespace: default

spec:

  gatewayClassName: netscaler-gateway-class

  addresses:

    - type: IPAddress
```

```
    value: "192.168.1.101"

  listeners:

    - name: dns-listener

      protocol: UDP

      port: 53

      allowedRoutes:

        namespaces:

          from: Same
```

Gateway を適用します。

```
kubectl apply -f udp-gateway.yaml
```

ステップ 4: UDPRoute を作成する

Gateway リスナーにアタッチする UDPRoute リソースを作成します。

```
apiVersion: gateway.networking.k8s.io/v1alpha2

kind: UDPRoute

metadata:

  name: dns-route

  namespace: default

spec:

  parentRefs:

    - name: udp-gateway

    sectionName: dns-listener
```

```
rules:

  - backendRefs:

    - name: dns-service

      port: 5353
```

UDPRouteを適用します:

```
kubectl apply -f udp-route.yaml
```

ステップ5: 設定を確認する

1. Gatewayのステータスを確認します:

```
kubectl get gateway udp-gateway -o yaml
```

1. UDPRouteのステータスを確認します:

```
kubectl get udproute dns-route -o yaml
```

1. Gateway経由でDNS解決をテストします:

```
dig @192.168.1.101 -p 53 example.com
```

NetScaler オペレーター

NetScaler Operatorで以下のトピックが利用可能です。

- [NetScaler Operator リリースノート](#)
- [NetScaler Operatorを使用してOpenShiftにNetScaler Kubernetes Gateway Controllerをデプロイする](#)
- [NetScaler Operatorを使用してOpenShiftにCPXと共にNetScaler Kubernetes Gateway Controllerをデプロイする](#)

NetScaler Operator リリースノート

リリースノートには、特定のビルドでソフトウェアがどのように変更されたか、およびビルドに存在する既知の問題が記載されています。リリースノートドキュメントには、次のセクションのすべてまたは一部が含まれています。

- 新機能: ビルドでリリースされた機能強化およびその他の変更点。
- 修正された問題: ビルドで修正された問題。
- 既知の問題: ビルドに存在する問題。

以下は、NetScaler® Operator リリースの一覧です。

- [3.6.0](#)
- [3.5.0](#)
- [3.4.1](#)
- [3.3.0](#)

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 が有効になっている場合、`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自動的に仮想 IP (VIP) を受け取ります。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 を使用した Kubernetes Gateway API を介して、TCP/UDP ベースのアプリケーションの公開をサポートします。

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 設定の永続化

NetScaler CPX の設定永続化により、ポッドのシャットダウン時に実行中の設定を Kubernetes ConfigMap に保存することで、新しいポッドが稼働するまでの時間を大幅に短縮します。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ポッドが起動した場合でも、設定が自動的に復元されます。新しいポッド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 ポッド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設定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

なインGRESS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 を使用した 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、OWASP Top 10 カテゴリに準拠した包括的な WAF およびボット管理保護を単一の Kubernetes カスタムリソースで構成できます。

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GRESS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 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 に関連付けられた 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

サポートしています。OpenShift ノードと Citrix ADC の間に、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。

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` の下の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ます。これにより、単一の GSLB ドメインが異なるホスト名を公開する Ingress によってバックアップされることが可能になります。
- **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) と `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可能にします。

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のエンティティ名変更

NetScaler Custom Resource Definition (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的な名前を保持します。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。
- 情報保持: 必要な Kubernetes 関連のメタデータは、NetScaler がエンティティコメントをサポートしている場合、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。

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を追加しました。
- GslbConfigSyncMonitor は、GSLB サイト監視の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、構成を変更します。

```
"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"
```

```
'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デプロイメントのSSLパススルーサポート

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、ロードバランサーで復号化するのではなく、サーバーに直接渡して復号化できます。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` 名により、「ref_manager」属性がないというエラーが発生する可能性があります。

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。

`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ルートは、2つではなく1つの負荷分散仮想サーバーを作成します。負荷分散仮想サーバーの命名で競合が確認されています。

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、負荷分散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

- イングレスで複数のサービスが指定されており、そのうちの1つに対してのみモニターが作成される場合、カスタムモニターの作成は失敗します。
- CRUD 操作中に、ポッド CRD 内の複数のポッド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、誤った例外処理が確認されています。
- NetScaler Ingress Controller ポッドが再起動すると、コントローラーがダウンしている間に削除されたイングレスの構成をコントローラーが削除できません。
- 名前が20文字まで同じ2つのHTTPルートは、2つではなく1つの負荷分散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。
- When multiple pods share the same pod IP address, deleting one pod can remove the active pod's endpoint state on NetScaler. In some cases, this can cause NetScaler Kubernetes Ingress Controller to crash with a key error.
- Kubernetes entities (namespace, services, etc) containing hyphens (-) caused configuration failures for NetScaler entities such as authpolicy and patset, since these entity names do not permit hyphens on the NetScaler.
- NSIC/GSLB Controller は、負荷がかかると Kubernetes API server から HTTP 429 (Too Many Requests) エラーを受け取り、一部のウォッチイベントが見逃される可能性があります。Helm chart は現在、専用の FlowSchema と PriorityLevelConfiguration (API Priority and Fairness) を提供しており、これにより 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した。

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、TCP のデフォルトプロファイル形式が正しくない問題を解決しました。
- NetScaler IPAM Controller の 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 ライセンスのサポート

以下のパラメータを設定することで、Kubernetes Gateway Controller を使用する NetScaler CPX および Ingress Controller を使用する NetScaler CPX を使用して、NetScaler CPX の LAS ライセンス方式を利用できるようになりました。

```
spec:

  admSettings:

    admIP: ''

    bandwidth: 1000

    licenseEdition: PLATINUM

    licensingMode: las

    loginSecret: ''
```

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

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 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 NETSCALER CPX のサポート

NetScaler Operator を使用して、NetScaler Kubernetes Gateway Controller とともに 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。

NetScaler Operator を使用して OpenShift に NetScaler Kubernetes Gateway Controller をデプロイする

このセットアップでは、NetScaler Kubernetes Gateway Controller は、OpenShift クラスターの外部にある NetScaler MPX、NetScaler VPX、または NetScaler BLX 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 K8s Gateway Controller が NetScaler と通信する必要がある IP アドレスを特定します。この IP アドレスは、NetScaler のデプロイの種類に応じて、次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 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 デプロイのクラスター管理 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、NetScaler MPX、または NetScaler BLX のユーザー名とパスワード。NetScaler K8s Gateway 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。
- 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ことも、OpenShift シークレットを使用することもできます (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
```

- Gateway API CRD がまだインストールされていない場合は、[公式標準チャンネル](#) からインストールします。

OpenShift Operator を使用して NetScaler Kubernetes Gateway Controller をデプロイする

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 CNNwebsite アプリケーションを展開します。
 - a. **Workloads > Deployments > Create Deployment** に移動し、以下の YAML ファイルを使用してデプロイを作成します。

```
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

          ports:
```

```
- name: http-80

  containerPort: 80

- name: https-443

  containerPort: 443

---
```

デプロイの作成 (/en-us/netScaler-kubernetes-gateway-controller/media/create-deployment.png)

- b. **Workloads > Pods** セクションに移動し、CNNwebsite 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。

ポッド (/en-us/netScaler-kubernetes-gateway-controller/media/pods.png)

3. CNNwebsite 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 ファイルを使用します。

```
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
```

4. CNNwebsite アプリケーションの自己署名証明書、ゲートウェイクラス、ゲートウェイ、および HTTP ルートを作成します。
 - a. TLS 設定のために Gateway-Listener で使用する自己署名 SSL 証明書とキーを作成します。以下のコマンドは、自己署名証明書を作成するためのサンプルコマンドであり、アプリケーションのホスト名が example.com であることを前提としています。

```
openssl genrsa -out cnnwebsite_key.pem 2048

openssl req -new -key cnnwebsite_key.pem -out cnnwebsite_csr.pem
-subj "/CN=example.com"

openssl x509 -req -in cnnwebsite_csr.pem -sha256 -days 365
-extensions v3_ca -signkey cnnwebsite_key.pem -CAcreateserial -out
cnnwebsite_cert.pem

oc create secret tls app1-secret --key cnnwebsite_key.pem --cert
cnnwebsite_cert.pem
```

- b. ゲートウェイクラス、ゲートウェイ、および HTTP ルートを展開します。

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

kind: GatewayClass

metadata:

  name: example-gateway-class

spec:

  controllerName: citrix.com/nsgw-controller

---

apiVersion: gateway.networking.k8s.io/v1

kind: Gateway

metadata:

  name: example-gateway

spec:

  gatewayClassName: example-gateway-class

  listeners:

    - name: http

      protocol: HTTP

      port: 80

    - name: https

      protocol: HTTPS

      port: 443

      tls:

        mode: Terminate
```

```
    certificateRefs:

      - kind: Secret

        name: app1-secret

    addresses:

      - type: IPAddress

        value: 4.4.4.4 # VIP

---

apiVersion: gateway.networking.k8s.io/v1

kind: HTTPRoute

metadata:

  name: example-route

spec:

  parentRefs:

    - name: example-gateway

  hostnames:

    - "example.com"

  rules:

    - matches:

      - path:

          type: PathPrefix

          value: /

    backendRefs:
```

```
- name: cnn-website

namespace: default

port: 443
```

ゲートウェイクラス (/en-us/netScaler-kubernetes-gateway-controller/media/gwclass-gateway-route.png)

5. **Operators > Installed Operators** に移動し、**NetScaler Operator** をクリックします。

オペレーター (/en-us/netScaler-kubernetes-gateway-controller/media/installedoperators.png)

6. **NetScaler K8s Gateway Controller** タブをクリックし、**Create NetScalerK8sGatewayController** を選択します。



NetScaler K8s Gateway ControllerのYAML定義が表示されます。必要に応じて、「フォームビュー」ボタンを選択し、フォームでパラメータを更新できます。

ネットスケラー キューバネティス ゲートウェイ コレクター YAML定義(/en-us/netScaler-kubernetes-gateway-controller/media/create-ns-kub-gw-cntrl.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。

```
netScaler:

  adcCredentialSecret: "nslogin"

  nsIP: "4.4.4.10"

  # NetScaler Management IP or SNIP with Management access enabled for HA
  and CLIP for NetScaler Cluster

gatewayController:

  entityPrefix: gwy

  gatewayControllerName: "citrix.com/nsgw-controller"
```

acceptLicense:yes

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	NetScaler K8s Gateway 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	ポッド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。
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。
nodeSelector	オプション	N/A	NetScaler K8s Gateway Controller のデプロイで nodeSelectorオプションに使用される ノードラベルの キー:値ペア。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 」

パラメータ	必須またはオプション	デフォルト値	説明
			を参照してください。
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ピング。
許容	オプション	該当なし	NetScaler K8s Gateway Controller デプロイメントのトレランス。

エクスポーター

パラメータ	必須またはオプション	デフォルト値	説明
extraVolumeMounts	オプション	該当なし	エクスポーターコンテナにマウントされる追加の VolumeMounts。
イメージ	任意	<code>quay.io/netScaler/netScaler-adc-metrics-exporter@sha<sha value of the latest release></code>	Quay.ioでホストされているメトリクスエクスポートイメージ。
必須	任意	偽	NetScaler MPX/VPX/BLXのメトリクスをプルするために、NetScaler K8s Gateway Controller とともにNetScaler Stats用エクスポーターを実行するには、trueに設定します。

パラメータ	必須またはオプション	デフォルト値	説明
pullPolicy	任意	IfNotPresent	エクスポーターイメージのプルポリシー。
ports.containerPort	任意	8888	エクスポーターコンテナポート。
リソース	オプション	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	オプション	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。

ゲートウェイコントローラー

パラメータ	必須またはオプション	デフォルト値	説明
clusterName	オプション	N/A	NetScaler K8s Gateway Controller がデプロイされているKubernetesクラスターの一意的識別子。
disableAPIServerCertificateVerify	オプション	いいえ	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。
enableLivenessProbe	オプション	はい	NetScaler K8s Gateway Controller の活性プローブ設定

パラメータ	必須またはオプション	デフォルト値	説明
			を無効にするには、falseに設定します。
enableReadinessProbe	オプション	はい	NetScaler K8s Gateway Controllerの準備プローブ設定を無効にするには、falseに設定します。
entityPrefix	任意	該当なし	NetScaler MPXまたはNetScaler VPX上のリソースのプレフィックス。
gatewayControllerName	必須	該当なし	ゲートウェイコントローラーの名前。
イメージ	必須	<code>quay.io/netScaler/netScaler-k8s-ingress-controller@sha<sha value of latest release></code>	Quay.ioでホストされているNetScaler K8s Gateway Controllerイメージ。
extraVolumeMounts	任意	該当なし	NetScaler K8s Gateway Controller コンテナにマウントされる追加のVolumeMounts。
jsonLog	オプション	いいえ	ログメッセージをJSON形式で出力する必要がある場合は、この引数をtrueに設定します。
kubernetesURL	オプション	N/A	NetScaler K8s Gateway Controller がイベントを登録するために使用する

パラメータ	必須またはオプション	デフォルト値	説明
			kube-apiserverのURL。指定しない場合、コントローラーは内部kube-apiserver IPアドレスを使用します。
livenessProbe	オプション	N/A	NetScaler K8s Gateway Controllerの活性プローブ設定。
logLevel	オプション	INFO	NetScaler K8s Gateway 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。サポートされているレベル: CRITICAL、ERROR、WARNING、INFO、DEBUG、TRACE、およびNONE。
nodeWatch	オプション	いいえ	NetScaler MPX/VPXからKubernetesクラスター内のポッドへのネットワークルートを自動的に構成するには、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	オプション	いいえ	NetScaler Node Controllerがポリシーベースルーティ

パラメータ	必須またはオプション	デフォルト値	説明
			ング (PBR) を構成していることをコントローラーに通知するには、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	はい	OpenShift環境を使用していない場合は、falseに設定します。
optimizeEndpointBinding	オプション	偽	true に設定すると、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なります。アプリケーションごとに多数のエンドポイントが存在する場合に推奨されます。NetScaler >=13.0–45.7 に適用されます。
podIPsforServiceGroupMembers	オプション	偽	true の場合、ノード IP とノードポートの代わりに、Pod IP と Pod ポートを LoadBalancer/NodePort サービスのサービスグループメンバーとして追加します。NetScaler とクラスター Pod (nodeWatch または

パラメータ	必須またはオプション	デフォルト値	説明
			NSNC) 間のルートが必要です。
pullPolicy	必須	IfNotPresent	NetScaler K8s Gateway Controller のイメージプルポリシー。
readinessProbe	オプション	N/A	NetScaler K8s Gateway Controller のレディネスプローブ設定。
リソース	任意	該当なし	NetScaler K8s Gateway Controller コンテナ向けの CPU およびメモリ リソース。

NetScaler

パラメータ	必須または任意	デフォルト値	説明
adcCredentialSecret	必須	該当なし	NetScaler K8s Gateway Controller が NetScaler に接続するために必要なシークレット。
nitroReadTimeout	任意	20	NITRO 読み取りタイムアウト（秒単位）。
nsIP	必須	N/A	NetScaler IP アドレス。
nsPort	任意	443	NetScaler K8s Gateway Controller が NetScaler と通信するために使用する

パラメータ	必須または任意	デフォルト値	説明
			ポート。HTTPにはポート80を使用できます。
nsProtocol	任意	N/A	NetScaler K8s Gateway Controller がNetScalerと通信するために使用するプロトコル。HTTPにはポート80を使用できます。
nsSNIPS	任意	N/A	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るNetScaler上のサブネットIPアドレスのリスト（PBRサポート）。
nsVIP	任意	N/A	NetScaler上の仮想IPアドレス。

8. 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 K8s Gateway 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。

ネットスケーラー クバネティス ゲートウェイ コレクター YAML定義(/en-us/netscaler-kubernetes-gateway-controller/media/gwcontrollerinstance.png)

9. **ワークロード > Pods**セクションに移動し、NetScaler Kubernetes Gateway Controllerポッドが稼働していることを確認します。

Pod作成成功(/en-us/netscaler-kubernetes-gateway-controller/media/podgwcntrlr.png)

10. トラフィックを送信してデプロイメントを確認します。

```
curl -kv https://example.com --resolve example.com:443:<VIP>
```

注:

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、OpenShiftクラスター内のポッドネットワークがNetScaler VPX、NetScaler MPX、またはNetScaler BLXから到達可能であることを確認してください。NetScaler K8s Gateway 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。

NetScaler Kubernetes Gateway ControllerをOpenShiftのCPXにNetScaler Operatorを使用してデプロイする

このセットアップでは、NetScaler Kubernetes Gateway ControllerがOpenShiftクラスターにデプロイされた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。
- 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。

```
kubectl create -f https://raw.githubusercontent.com/prometheus-operator/prometheus-operator/refs/heads/main/example/prometheus-operator-crd/monitoring.coreos.com_servicemonitors.yaml
```

- [直接Prometheusエクスポート](#)を介して収集されたNetScaler CPXのメトリックを表示する場合は、[Prometheus Operator](#)をインストールします。
- Gateway API CRDがまだインストールされていない場合は、[公式標準チャンネル](#)からインストールします。

NetScaler Kubernetes Gateway ControllerをNetScaler CPXのサイドカーとしてデプロイする

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CNNwebsiteアプリケーションをデプロイします。

- 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メントを作成します。

```
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

ports:

  - name: http-80

    containerPort: 80

  - name: https-443

    containerPort: 443
```



- b. **Workloads > Pods**セクションに移動し、CNNwebsite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。



3. CNNwebsite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: 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
```

4. CNNwebsiteアプリケーション用に、自己署名証明書、ゲートウェイクラス、ゲートウェイ、およびHTTPルートを作成します。
 - a. TLS構成のためにGateway-Listenerで使用する自己署名SSL証明書とキーを作成します。次のコマンドは、自己署名証明書を作成するための単なるサンプルコマンドであり、アプリケーションのホスト名がexample.comであると仮定しています。

```
openssl genrsa -out cnnwebsite_key.pem 2048

openssl req -new -key cnnwebsite_key.pem -out cnnwebsite_csr.pem
-subj "/CN=example.com"

openssl x509 -req -in cnnwebsite_csr.pem -sha256 -days 365
-extensions v3_ca -signkey cnnwebsite_key.pem -CAcreateserial -out
cnnwebsite_cert.pem

oc create secret tls app1-secret --key cnnwebsite_key.pem --cert
cnnwebsite_cert.pem
```

- b. ゲートウェイクラス、ゲートウェイ、およびHTTPルートを展開します。

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

kind: Gateway

metadata:

  name: example-gateway

spec:

  gatewayClassName: example-gateway-class

  listeners:

    - name: http

      protocol: HTTP

      port: 80

    - name: https

      protocol: HTTPS

      port: 443

      tls:

        mode: Terminate

        certificateRefs:

          - kind: Secret

            name: app1-secret

  ---

apiVersion: gateway.networking.k8s.io/v1

kind: HTTPRoute

metadata:
```

```
name: example-route

spec:

  parentRefs:

    - name: example-gateway

  hostnames:

    - "example.com"

  rules:

    - matches:

      - path:

          type: PathPrefix

          value: /

      backendRefs:

        - name: cnn-website

          namespace: default

        port: 443
```

5. **Operators > Installed Operators**に移動し、**NetScaler Operator**をクリックします。



6. **NetScaler K8s Gateway Controller**タブをクリックし、**Create NetScalerCPXWithK8sGatewayController**を選択します。



Kubernetes Gateway Controller YAML定義を含むNetScaler CPXが表示されます。必要に応じて、ボタンを選択し、フォームでパラメーターを更新できます。



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ーを一覧表示する次の表を参照してください。

```
gatewayController:

  entityPrefix: gwy

  gatewayControllerName: "citrix.com/nscpxgw-controller"


accept License: yes


service:

  annotations: {}

  spec:

    type: NodePort

    ports:

      - port: 80

        targetPort: 80

        protocol: TCP

        name: http

      - port: 443

        targetPort: 443

        protocol: TCP

        name: https
```

パラメーター	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 を承諾するには「はい」に設定します。
アフィニティ	任意	該当なし	Pod の割り当て用アフィニティラベル。
nodeSelector	任意	該当なし	NetScaler Ingress Controller の展開における nodeSelector オプションに使用するノードラベルのキーと値のペア。
podAnnotations	任意	N/A	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になります。
許容	オプション	N/A	CPX-NetScaler K8s Gatewayデプロイメントのトレランスを指定します。

NetScaler コンソール設定

パラメータ	必須またはオプション	デフォルト値	説明
ADMIP	任意	N/A	NetScaler Console (旧 NetScaler 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	偽	NetScaler CPX で帯域幅ベースのライセンスを使用する場合は、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	オプション	PLATINUM	指定可能な値は、Standard、Platinum、Enterpriseです。
licenseServerIP	オプション	N/A	NetScaler CPXをライセンスするための NetScaler Console (旧NetScaler Console) のIPアドレス。詳細については、「ライセンス」を参照してください。
licenseServerPort	オプション	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NetScaler Console) ポート。

パラメータ	必須またはオプション	デフォルト値	説明
loginSecret	オプション	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	オプション	偽	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	オプション	N/A	NetScaler CPX に 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 <code>pool</code> または <code>las</code> に設定します。licensingMode が <code>las</code> の場合は、ADMIP、loginSecret、bandWidth、および licenseEdition を設定します。 <code>pool</code> の場合は、licenseServerIP と bandWidthLicense または vCPULicense のいずれかを設定します。

エクスポーター

パラメータ	必須またはオプション	デフォルト値	説明
必須	オプション	フォールス	NetScaler CPX のメトリクスをプルするために、NetScaler Ingress Controller とともに NetScaler Stats 用のエクスポーターを実行するには、このパラメーターを true に設定します。
イメージ	オプション	quay.io/netScaler/ netScaler-adc- metrics- exporter@sha256:3 d54b9151c742c001 17be74e1ffd544303 58e4cf628fe1f02d5 03088583f678	Quay.io でホストされているメトリクスエクスポートイメージ。
pullPolicy	オプション	IfNotPresent	エクスポートのイメージプルポリシー。
リソース	オプション	{}	メトリクスエクスポートコンテナの CPU/メモリリソース。
ports.containerPort	オプション	8888	エクスポートコンテナのポート。
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	プライベートな Docker レジストリ または リポジトリ からイメージをプルす

パラメータ	必須またはオプション	デフォルト値	説明
			るために使用される Kubernetes シークレットのリスト。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。

ゲートウェイコントローラー

パラメータ	必須またはオプション	デフォルト値	説明
disableAPIServerCertificateVerify	オプション	偽	APIサーバーの証明書検証を無効にするには、このパラメータを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、このパラメータをfalseに設定します。
enableReadinessProbe	任意	トゥルー	NetScaler Ingress Controllerの Readinessプローブ設定。
entityPrefix	任意	該当なし	NetScaler CPX上のリソースのプレフィックス。
extraVolumeMounts	任意	該当なし	該当なし

パラメータ	必須またはオプション	デフォルト値	説明
gatewayControllerName	必須	該当なし	ゲートウェイコントローラーの名前を指定します。
イメージ	必須	quay.io/netScaler/ netScaler-k8s- ingress- controller@sha256: 2ff38654476234bfe 2c3908932702b90c 8ec6ed3cfb7c7f467 dc6181df0f1adf	Quay.io でホストされている NetScaler Ingress Controller イメージ。
jsonLog	オプション	いいえ	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。指定しない場合、内部の kube-apiserver IP アドレスが使用されます。
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時間（秒単位）。
livenessProbe.periodSeconds	オプション	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。

パラメータ	必須またはオプション	デフォルト値	説明
logLevel	オプション	INFO	NSICによって生成されるログを制御するためのログレベル。サポートされているレベル： CRITICAL、 ERROR、 WARNING、 INFO、DEBUG、 TRACE、および NONE。
OpenShift	オプション	はい	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。
optimizeEndpointBinding	オプション	フォールス	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が多い場合に推奨されます。 (NetScaler >=13.0-45.7)
pullPolicy	必須	IfNotPresent	NetScaler Ingress Controllerのイメージプルポリシー。
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 Ingress Controllerコンテナ

パラメータ	必須またはオプション	デフォルト値	説明
			のCPUおよびメモリのリソース。
routeLabels	任意	N/A	OpenShift クラスター内の routeSharding のため、NetScaler Ingress Controller によって使用される ルートラベルセレクター。

NetScaler CPX 設定

パラメーター	必須または任意	デフォルト値	説明
cpxCommands	任意	N/A	CPX がインスタンス化されるときに適用される、ユーザーが提供する NetScaler 起動設定。動的ではありません。ConfigMap への変更は、ポッドが再起動されるまで反映されません。
cpxLicenseAggregator	任意	N/A	CPXのライセンス供与に使用する CPXライセンスアグリゲーターのIPアドレスまたはFQDN (使用する場合)。
cpxShellCommands	オプション	N/A	CPXがインスタンス化されたときに適用される、ユーザー指定の起動コマンド。動的ではないため、

パラメーター	必須または任意	デフォルト値	説明
			変更にはPodの再起動が必要です。
enableLivenessProbe	オプション	はい	NetScaler CPX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。
enableStartupProbe	オプション	はい	NetScaler CPXのスタートアッププローブ設定を無効にするには、falseに設定します。
extraVolumeMounts	オプション	該当なし	CPX の追加ボリュームマウント。
hostName	オプション	該当なし	CPX に設定するホスト名。
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。
livenessProbe	オプション	該当なし	NetScaler CPX の活性プローブ設定。
mgmtHttpPort	任意	9080	NodePort CPXサービスに使用されるHTTPポート。
mgmtHttpsPort	任意	9443	NodePort CPXサービスに使用されるHTTPSポート。

パラメーター	必須または任意	デフォルト値	説明
nitroReadTimeout	任意	20	NITRO読み取りタイムアウト（秒単位）。
nsHTTP2ServerSide	任意	OFF	NetScalerサービスグループ構成でHTTP/2を有効にするには、ONに設定します。
nsProtocol	オプション	HTTP	NetScaler CPXとNetScaler Ingress Controller間の通信に使用されるプロトコル。
pullPolicy	必須	IfNotPresent	NetScaler CPXイメージのプルポリシー。
service.annotations	オプション	N/A	CPXサービスのアノテーションの辞書（例：アノテーションを介してTLSシークレットを公開するため）。
service.spec	オプション	N/A	CPXのサービス仕様。CPXをNodePortとして公開するには、 service.spec.type: NodePortを設定し、 service.citrix.com/secretアノテーションを使用してTLSシークレットを参照します。

パラメーター	必須または任意	デフォルト値	説明
startupProbe	オプション	N/A	NetScaler CPXのスタートアッププローブ設定。

8. 必要なパラメータの値を更新した後、**作成**をクリックします。Kubernetes Gateway 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名前空間で、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-gateway-controller` ポッドが稼働してい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を検証します。

- 次のコマンドを使用して NodePort の詳細を取得します: `oc get svc`
- cpx-service NodePort を使用し、次のコマンドに示すようにトラフィックを送信します:

```
curl https://example.com:<NodePort> --resolve
example.com:<NodePort>:<Master-Node-IP>
```

上記の cURL コマンドは次の出力を返す必要があります: `<html><body><h1>It works!</h1></body></html>`

ネットワーク

ネットワークには、次のトピックがあります。

- Kubernetes ノードと Kubernetes クラスター外の NetScaler インスタンス間のネットワークを確立する(/ja-jp/netscaler-kubernetes-gateway-controller/network/node-controller.html)

KubernetesノードとKubernetesクラスター外のNetScalerインスタンス間のネットワークを確立する

Kubernetesクラスターにデプロイ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。この接続は、以下のいずれかの方法で実現できます。

1. NetScaler Kubernetes Gateway Controllerで自動ルート設定を有効にします。

各PodサブネットのNetScaler上にルートを作成するには、コントローラースペックで `gatewayController.nodeWatch: true` を設定します。

注:

NetScaler SNIPがKubernetesノードと同じサブネットを共有している場合に、このオプションを選択できます。

2. Kubernetesノードからサブネットルートを駆動するために、[NetScaler Node Controller](#)をデプロイします。

注:

`gatewayController.nodeWatch` はデフォルトで `false` です。

単一のNetScalerの背後にある複数のクラスター

1つのNetScalerが複数のKubernetesクラスターのトラフィックを負荷分散する場合、CNIサブネットの重複によりルーティングの競合が発生し、静的ルートの挿入が失敗する可能性があります。そ

のシナリオでは、[ポリシーベースルーティング \(PBR\)](#)を使用し、各クラスターに個別のSNIP値のセットを提供します。

コントローラーが正しいPBRエントリをインストールするように、Helm設定、環境変数、またはConfigMapを介してSNIPを提供します。

NetScalerでPBR SNIPを設定するには、次のコマンドを使用します。

```
helm install my-release netscaler/netscaler-kubernetes-gateway-controller
--set gatewayController.gatewayControllerName=citrix.com/nsgc-
controller,license.accept=yes,gatewayController.entityPrefix=gwy,netscaler
.nsnCPbr="true",netscaler.adcCredentialSecret=<Secret-for-ADC-
credentials>,netscaler.nsSNIPs='[<NS_SNIP1>\, <NS_SNIP2>\, ...]'
```

トラフィック管理

トラフィック管理で利用できるトピックは次のとおりです。

- [NetScaler IPAM Controller を介した動的 IP アドレス割り当て](#)
- [ゲートウェイ内の複数のリスナー](#)
- [HTTPRoute フィルターによるトラフィック管理機能の強化](#)
- [NetScaler プロファイル](#)

NetScaler IPAM Controller を介した動的 IP アドレス割り当て

NetScaler Kubernetes Gateway Controller は、NetScaler IPAM Controller 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 が有効になっている場合、`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、NetScaler IPAM Controller から仮想 IP (VIP) アドレスを自動的に受け取ります。この機能により、各ゲートウェイに 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。

前提条件

NetScaler IPAM Controller を介した動的 IP アドレス割り当てを構成する前に、以下を確認してください。

- **NetScaler Gateway Controller** が Helm `values.yaml` の `ipam: True` とともに展開されていること。詳細については、「[NetScaler Kubernetes Gateway Controller の展開](#)」を参照してください。
- **NetScaler IPAM Controller** が IP アドレス範囲とともにクラスターに展開されていること。目的の IP アドレス範囲で NetScaler IPAM Controller を展開する方法については、「[NetScaler IPAM Controller インストールガイド](#)」を参照してください。

`VIP_RANGE` を構成する際に、Gateway アノテーションから参照できる名前付き範囲 (例: `Production`、`Staging`) と、デフォルトプールとして機能する名前なし範囲を定義できます。

構成

NetScaler IPAM Controller を介した動的 IP アドレス割り当てを構成するには、次の手順を実行します。

ステップ 1: GatewayClass を作成する

NetScaler Kubernetes Gateway Controller 名で GatewayClass を作成します。

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

kind: GatewayClass

metadata:

  name: citrix-gateway-class

spec:

  controllerName: citrix.com/nsgc-controller
```

ステップ 2: Gateway を作成する

`spec.addresses` なしで Gateway を作成します。NetScaler IPAM Controller は、構成されたプールから IP アドレスを自動的に割り当てます。

基本的な Gateway (デフォルトの IP 範囲)

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

kind: Gateway

metadata:

  name: my-gateway

  namespace: default

spec:

  gatewayClassName: citrix-gateway-class

  listeners:
```

```
- name: http-listener

protocol: HTTP

port: 80

allowedRoutes:

  namespaces:

    from: All
```

名前付き IP 範囲を持つ Gateway

Gatewayに `gateway.citrix.com/ipam-range` アノテーションを使用して、特定の名前付き範囲からIPアドレスを割り当てます。

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

kind: Gateway

metadata:

  name: production-gateway

  annotations:

    gateway.citrix.com/ipam-range: "Production"

spec:

  gatewayClassName: citrix-gateway-class

  listeners:

    - name: https-listener

      protocol: HTTPS

      port: 443

      allowedRoutes:
```

```
namespaces:
```

```
  from: All
```

アノテーション内の範囲名は、NetScaler IPAMコントローラーの `VIP_RANGE` 設定で定義された名前付き範囲と一致する必要があります。

IPアドレスの割り当てを確認する

Gatewayを作成した後、`status.addresses` フィールドを確認して、割り当てられたIPアドレスを確認します。

```
kubectl get gateway my-gateway -o jsonpath='{.status.addresses}'
```

期待される出力:

```
[{"type": "IPAddress", "value": "10.1.2.1"}]
```

静的IP 対 IPAM

設定	動作
<code>spec.addresses</code> が設定されている	NetScaler Kubernetes Gateway Controllerは、指定されたIPアドレスを直接使用します。IPAMは呼び出されません。
<code>spec.addresses</code> が省略されている	IPAMが有効になっている場合、NetScaler Kubernetes Gateway ControllerはNetScaler IPAMコントローラーからIPアドレスを要求します。

Gatewayに特定のIPアドレスが必要な場合は、`spec.addresses` で明示的に設定します。

```
spec:
```

```
  addresses:
```

```
    - type: IPAddress
```

```
value: "10.1.2.50"
```

アノテーションリファレンス

アノテーション	スコープ	説明
<code>gateway.citrix.com/ ipam-range</code>	ゲートウェイ	NetScaler IPAM Controllerがアドレスを割り当てる、名前付きIPアドレス範囲を指定します。省略した場合、デフォルト（名前なし）の範囲が使用されます。

トラブルシューティング

症状	考えられる原因	解決策
ゲートウェイがIPアドレスを割り当てられずに保留状態のままになる	NetScaler Kubernetes Gateway Controllerの展開中に、NetScaler IPAMコントローラーが実行されていないか、有効になっていない	NetScaler IPAMコントローラーポッドが実行されていること、およびNetScaler Kubernetes Gateway Controllerの <code>values.yaml</code> に <code>ipam: True</code> が設定されていることを確認します。
予期された範囲からIPアドレスが割り当てられない	アノテーションの範囲名がNetScaler IPAM構成と一致しない	<code>gateway.citrix.com/ ipam-range</code> アノテーションの値が、NetScaler IPAMコントローラーの <code>VIP_RANGE</code> にある名前付き範囲と一致することを確認します。
IPプールが枯渇しました	設定された範囲内のすべてのIPアドレスが割り当てられています	NetScaler IPAM Controllerの設定で <code>VIP_RANGE</code> を拡張するか、未使用のGatewayを解放してください
<code>spec.addresses</code> が設定されると、GatewayはIPAMなしで動作します	予期される動作	静的アドレスはIPAMをバイパスします。IPAM割り当てを使用するには、

症状	考えられる原因	解決策
		<code>spec.addresses</code> を削除してください

制限事項

- 各Gatewayは、NetScaler IPAM Controllerから単一のVIPを受け取ります。
- `gateway.citrix.com/ipam-range` アノテーションは、NetScaler IPAMコントローラー設定に存在する範囲名を参照する必要があります。無効な範囲名はIP割り当てを妨げます。
- アノテーションが省略されている場合、NetScaler IPAM Controllerは、デフォルトの（名前のない）プールからIPを自動的に割り当てます（設定されている場合）。

Gateway内の複数のリスナー

NetScaler® Kubernetes Gateway Controllerは、Kubernetes Gateway APIで定義されているように、単一のGatewayリソース内で複数のリスナーをサポートします。この機能により、Kubernetes クラスター内のサービスへの外部トラフィックの管理とルーティング方法について、より高い柔軟性と制御が可能になります。

複数のリスナーをサポートすることで、次のことが可能になります。

- 同じゲートウェイIPアドレスのセットで**複数のポートを公開**し、異なるアプリケーションやサービスが個別のポートを介してアクセスできるようにします。
- 同じゲートウェイIPアドレスを使用しながら、HTTPやHTTPSなどの**異なるプロトコルを別々のポートでサポート**します。
- 各リスナーに**固有のTLS構成を適用**し、特定の要件に合わせた安全な通信を可能にします。
- 各リスナーに**個別のルーティングルールを定義**し、異なるインバウンドトラフィックフローを管理および指示できるようにします。

この強化されたリスナー機能により、単一のGatewayの背後で複数のセキュアおよび非セキュアなアプリケーションをホストしたり、コンプライアンスおよびセキュリティ目的でトラフィックをセグメント化したりするなど、高度なトラフィック管理シナリオが可能になります。複数のリスナーを使用することで、アプリケーションの配信を最適化し、Kubernetes環境のスケラビリティとセキュリティを向上させることができます。

設定例

Kubernetesの以下の設定例では、3つのコンテンツスイッチング仮想サーバーと、各コンテンツスイッチング仮想サーバー用のipsetが作成されます。

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

kind: Gateway

metadata:

  name: my-gateway-http

  namespace: default

spec:

  gatewayClassName: my-gateway-class

  addresses:

    - type: IPAddress

      value: <ip1>

    - type: IPAddress

      value: <ip2>

    - type: IPAddress

      value: <ip3>

  listeners:

    - name: http-listener

      protocol: HTTP

      port: 80

      allowedRoutes:
```

```
    namespaces:

      from: All

- name: http-listener1

  protocol: HTTP

  port: 81

  allowedRoutes:

    namespaces:

      from: All

- name: https

  protocol: HTTPS

  port: 443

  tls:

    mode: Terminate

    certificateRefs:

      - kind: Secret

        name: my-secret

        group: ""

  allowedRoutes:

    namespaces:

      from: All
```

名前空間フィルター

デフォルトでは、Gatewayは厳格な名前空間分離を強制します。これは、Gatewayリソースと同じ名前空間に存在する `HTTPRoute` のようなルートのみがそのGatewayにアタッチできることを意味します。このデフォルトの動作は、安全で分離されたルーティング環境を提供し、同じ名前空間内の信頼されたルートのみがGatewayのトラフィック管理に影響を与えることを保証します。

より高度なルーティングシナリオをサポートするために、Gatewayリソースの各リスナー定義内で `allowedRoutes` フィールドを構成することで、クロス名前空間ルーティングを有効にできます。`allowedRoutes` フィールドは、特定のリスナーにルートをアタッチすることを許可する名前空間をきめ細かく制御し、柔軟で安全なマルチテナントまたはマルチチーム展開を可能にします。

`allowedRoutes.namespaces.from` フィールドは、名前空間選択のために3つの異なるモードをサポートしています。

- **Same (デフォルト):** - Gatewayと同じ名前空間からのルートのみがアタッチを許可されます。このモードは最も制限が厳しく安全なオプションであり、厳格な分離が必要な環境に適しています。
- **すべて:** - Kubernetesクラスター内の任意の名前空間からのルートをGatewayリスナーにアタッチできます。このモードは最大限の柔軟性を提供し、集中管理されたGatewayリソースが複数の名前空間からのルートを処理できるようにします。ただし、このモードを使用する際には、セキュリティ上の影響を考慮し、適切なRBAC制御を実装することが重要です。
- **セクター:** - 名前空間ラベルの一致条件に基づいてルートのアタッチが許可されます。このモードはKubernetesラベルセクターを使用して、高度な名前空間ベースのルーティングポリシーを可能にします。例えば、特定のラベル（`team=frontend` など）を持つ名前空間のみがリスナーにルートをアタッチできるようにすることで、きめ細かなマルチテナント制御を提供できます。

注:

`HTTPRoute` が適用された後に名前空間ラベルが変更された場合、その変更はGateway Controllerに反映されません。

HTTPRouteフィルターによるトラフィック管理機能の強化

NetScaler® Kubernetes Gatewayコントローラーは、Kubernetes Gateway APIで定義されているHTTPRouteフィルターをサポートしています。この機能により、HTTPRouteリソース内で直接、高度なトラフィック操作ロジックを実装できます。フィルターを活用することで、リクエストとレ

スポンスを変更し、ヘッダー操作、URL書き換え、リクエストリダイレクトなど、幅広いユースケースを実現できます。

extensionRefを使用したNetScaler CRDとの統合

NetScaler Ingress Controller (NSIC) のカスタムリソース定義 (CRD) は、HTTPRouteフィルター内のextensionRefフィールドを介して参照できます。この強力な機能により、高度なNetScaler機能をGateway API構成に直接シームレスに統合できます。

extensionRefを介して参照できるNSIC CRDには、[rewritepolicy](#)、[ratelimit](#)、[bot](#)、[waf](#)、[appqoeppolicy](#) があります。

参照例

```
rules:
  - filters:
    - type: ExtensionRef

    extensionRef:

      group: "citrix.com"

      kind: "<crd-kind>"

      name: "<crdinstance-name>"
```

注:

- CRDインスタンスはサービス名なしで作成する必要があります。
- HTTPRouteフィルターの `extensionRef` の場合、グループは常に「citrix.com」であり、種類はNetScaler CRDタイプに対応します。有効な種類には、`bot`、`waf`、`rewritepolicy`、`ratelimit`、および `appqoeppolicy` が含まれます。

特殊な処理のために、以下のNSIC CRDを参照できます。

- **ボット管理 (BOT):** NetScalerがサポートする高度なボット検出および軽減技術を適用することで、悪意のあるボットトラフィックからアプリケーションを保護します。

- **Webアプリケーションファイアウォール (WAF):** NetScalerがサポートする堅牢なWAF機能を統合してトラフィックを検査し、既知の攻撃やゼロデイ攻撃をブロックすることで、アプリケーションを保護します。
- **書き換えポリシー:** NetScalerの豊富な書き換えポリシーエンジンを使用することで、標準のGateway APIフィルターを超える高度なリクエストおよびレスポンスの書き換えルールを適用します。
- **レート制限:** 受信リクエストのレートを管理するポリシーを適用し、アプリケーションが過負荷になるのを防ぎます。
- **AppQoeポリシー:** さまざまな基準に基づいてトラフィックを優先または制限するポリシーを適用し、重要なアプリケーションの最適なパフォーマンスとリソースの公平な割り当てを保証します。

`extensionRef` メカニズムは、Kubernetes Gateway API を使用してトラフィック管理を行いながら、NetScaler の広範な機能セットを活用できるようにする橋渡し役として機能します。

標準 HttpRoute API フィルターのネイティブサポート

NetScaler Kubernetes Gateway Controller は、HTTPRoute フィルターのサポートに基づいて、以下の標準 Gateway API フィルターをネイティブに実装しています。

URL書き換え

リクエストがバックエンドサービスに転送される前に、リクエストのパスまたはホスト名を変更します。このフィルターは、ユーザー向け URL を内部サービスパスにマッピングしたり、公開 URL を変更せずにサービスを移行したりするのに役立ちます。

例: `/old-path` を `/new-path` に書き換えたり、リクエストのホスト名を変更したりします。

```
- filters:
  - type: URLRewrite
    urlRewrite:
      path:
        type: ReplacePrefixMatch
        replacePrefixMatch: /new-path
    matches:
```

```
- path:

  type: PathPrefix

  value: /old-path

# Rewrite /old-path/rest-of-the-url to /new-path/rest-of-the-url
```

RequestHeaderModifier および ResponseHeaderModifier

受信リクエストまたは送信レスポンスの HTTP ヘッダーを追加、設定、または削除します。このフィルターは、トレース情報の挿入、セキュリティヘッダーの設定、キャッシュ制御ディレクティブの変更など、さまざまな目的に使用できます。

例: `X-Forwarded-Proto` ヘッダーを追加したり、レスポンスから内部専用ヘッダーを削除したりします。

```
rules:

- filters:

  - type: RequestHeaderModifier

    requestHeaderModifier:

      add:

        - name: X-Forwarded-Proto

          value: http

      remove:

        - Proxy-Authenticate

  - type: ResponseHeaderModifier

    responseHeaderModifier:

      set:

        - name: X-Cache
```

```
        value: HIT

    remove:

    - Server

# Add 'X-Forwarded-Proto' and Remove 'Proxy-Authenticate' in request send
to the backend.

# Set/Replace 'X-Cache' header value and Remove 'Server' header in
response sent back to the client
```

RequestRedirect

HTTP 3xx リダイレクトレスポンスを発行し、ユーザーを異なる URL にリダイレクトしたり、HTTPS を強制したり、移動したコンテンツを適切に管理したりします。

例: HTTP トラフィックを HTTPS にリダイレクトしたり、非正規ホスト名を正規ホスト名にリダイレクトしたりします。

```
rules:

  filters:

    - type: RequestRedirect

    requestRedirect:

      path:

        type: ReplacePrefixMatch

        replacePrefixMatch: /new-path

      statusCode: 301

# Redirect Client request to /new-path with 301 status
```

RequestMirror

リクエストのコピーを異なるバックエンドサービスに送信します。これはトラフィックシャドウイングに非常に役立ちます。このフィルターを使用すると、クライアントの応答に影響を与えることなく、新しいサービスバージョンのテスト、分析の実行、問題のデバッグを行うことができます。ミラーリングされたバックエンドからの応答は無視されます。

注:

複数のミラーバックエンドを指定できます。ただし、ルールごとにトラフィックミラーリングの割合または分数設定は1つのみが尊重されます。

例: リアルタイムのパフォーマンスインサイトを得るために、本番トラフィックを監視および分析サービスにミラーリングする。

```
rules:
  - matches:
      - path:
          type: PathPrefix
          value: /webapp
    filters:
      - type: RequestMirror
        requestMirror:
          percent: 80
          backendRef:
            name: traffic-monitoring-service
            port: 8080
      - type: RequestMirror
        requestMirror:
```

```
        backendRef:

          name: traffic-analytics-service

          port: 9000

        backendRefs:

          - name: webapp-v1-primary

            port: 80

# 80% of traffic sent to `webapp-v1-primary` will be mirrored to both
# `traffic-monitoring-service` and `traffic-analytics-service`.

#If percent/fraction is not specified 100% of the traffic will be cloned/
mirrored
```

NetScaler® プロファイルサポート

NetScaler Kubernetes Gateway Controller は、堅牢で一元化された構成を実現するために、プロファイルCRD (SSLプロファイル、HTTPプロファイル、TCPプロファイルを含む) を導入します。これらのプロファイルにより、クラスターオペレーターは、TLS暗号、TCP最適化、HTTPタイムアウトなどの標準化された再利用可能なインフラストラクチャ設定を定義でき、アプリケーション開発者はそれらをルーティング定義に簡単に適用できます。

プロファイルCRDのデプロイ

NetScalerプロファイルYAMLファイル: [profiles_crd.yaml](#)。

注:

デプロイ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 apply -f profiles_crd.yaml
```

例:

```
root@1:~# kubectl apply -f profiles_crd.yaml

customresourcedefinition.apiextensions.k8s.io/httpprofiles.citrix.com
created

customresourcedefinition.apiextensions.k8s.io/tcpprofiles.citrix.com
created

customresourcedefinition.apiextensions.k8s.io/sslprofiles.citrix.com
created
```

プロファイル仕様

すべてのプロファイルCRDは、目的のNetScaler構成を定義するために、次の2つのフィールドのいずれかを使用します。これらのフィールドは相互に排他的です:

- `spec.preconfigured`
- `spec.config`

フィールド	目的	NETSCALERでのアクション
<code>spec.preconfigured</code>	NetScalerアプライアンス上の既存のプロファイルを使用します。	コントローラーはプロファイルを作成しません。提供された文字列値をバインディングのプロファイル名として使用します。
<code>spec.config</code>	プロファイルを定義します。	コントローラーは、 <code>config</code> ブロックで指定されたキーと値のペアを使用して、NetScaler上に一意の名前を持つプロファイル（CRDのメタデータに基づく）を作成します。

NetScaler Kubernetes Gateway ControllerでのNetScalerプロファイルの追加

NetScalerプロファイル（HTTP、TCP、SSL）を、Kubernetes Gatewayリソースにフロントエンドプロファイルおよびバックエンドプロファイルとして追加できます。

フロントエンドプロファイル

フロントエンドプロファイルは、Gatewayのリスナーに直接設定を適用するために使用されます。たとえば、NetScaler上のフロントエンドコンテンツスイッチング仮想サーバーなどです。これらのプロファイルには、バインディング用の `targetRef` セクションを含める必要があります。

単一のフロントエンドプロファイルCRDを複数のGatewayに効率的に適用できます。

例1 すべてのGatewayリスナーに設定されたフロントエンドSSLプロファイル

次の例では、カスタム構成パラメーターを持つ新しいフロントエンドSSLプロファイルを定義し、「my-gateway」リソースのすべてのリスナーまたはコンテンツスイッチング仮想サーバーに適用します。

```
apiVersion: citrix.com/v1

kind: sslprofile

metadata:

  name: frontend-ssl-custom

  namespace: production

spec:

  targetRef:

    - name: my-gateway

      namespace: production

      kind: Gateway

      group: gateway.networking.k8s.io

  config:
```

```
sslprofiletype: "frontend"

sessreuse: "ENABLED"

zerorttearlydata: "DISABLED"

hsts: "ENABLED"

preload: "YES"
```

バックエンドプロファイル

バックエンドプロファイルは、Gateway（NetScaler）とバックエンドKubernetesアプリケーション間の接続に影響する設定を構成するために使用されます。たとえば、サービスグループやサーバーなどです。

バックエンドプロファイルには `targetRef` を含めることはできません。代わりに、`HTTPRoute` の標準 `ExtensionRef` フィルタータイプを使用して特定のHTTPRouteルールに直接アタッチされ、ルートごとにきめ細かな制御を可能にします。

例1：HTTPRoute用のバックエンド事前設定済みSSLプロファイル

次の例では、「manual_secure_backend」という名前の既存のNetScalerプロファイルを使用し、`HTTPRoutes` による使用を意図したプロファイルを定義します（`targetRef` は存在しません）。

```
apiVersion: citrix.com/v1

kind: sslprofile

metadata:

  name: simple-ssl-profile-backend

spec:

  # Define the existing profile name

  preconfigured: "manual_secure_backend"
```

例2：バックエンドプロファイルをHTTPRouteに適用する

次のHTTPRouteは、`ExtensionRef` フィルターを介して前の例で作成されたバックエンド `sslprofile` を参照します。

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

kind: HTTPRoute

metadata:

  name: my-httproute

spec:

  parentRefs:

    - name: my-gateway-http

      namespace: default

  hostnames:

    - "example123.com"

  rules:

    - filters:

        - type: ExtensionRef

          extensionRef:

            group: "citrix.com"

            kind: "sslprofile"

            name: "simple-ssl-profile-backend" # References the Profile CRD
name

  matches:

    - path:
```

```
    type: PathPrefix

    value: /

  backendRefs:

    - name: app1-svc

      namespace: default

    port: 80
```

NetScalerプロファイルの種類

次のNetScalerプロファイルタイプがサポートされています。

- HTTPプロファイル
- TCPプロファイル
- SSLプロファイル

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。設定の容易さと柔軟性を提供します。各アプリケーション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アプリケーションにプロファイルをバインドできます。

SSLプロファイルCRDスキーマ参照は次のとおりです。

フィールド	タイプ	説明	フロントエンド プロファイルに 必須	バックエンド プロファイルに必 須
事前構成済み	文字列	config と相互排 他的です。使用 する NetScaler アプライアンス 上に存在する、 事前設定済みま たは組み込みの SSL プロファイ ルの名前。	いいえ	いいえ

フィールド	タイプ	説明	フロントエンド プロファイルに 必須	バックエンドプ ロファイルに必 須
構成	オブジェクト	preconfigured と相互排他的で す。完全なカス タムプロファイ ル用の NetScaler NITRO API に 関連するすべて のパラメーター が含まれます。 お客様は、ここ で有効な NITRO SSL プ ロファイルパラ メーターを指定 できます。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。	いいえ	いいえ
targetRef	オブジェクトの 配列	このプロファイ ルが適用される ターゲットリ ソース（通常は Gateway）のリ スト。この フィールドは Frontend Profilesには必 須であり、 Backend Profilesには省	はい	いいえ

フィールド	タイプ	説明	フロントエンド プロファイルに 必須	バックエンドプ ロファイルに必 須
		略する必要があります。		
gatewayClassName	文字列	このSSLプロファイルを適用する必要がある GatewayClass の名前。マルチテナント環境でプロファイルを特定のコントローラーインスタンスにスコープするために使用されます。	いいえ	いいえ
SSLサイファ	文字列の配列	SSLプロファイルにバインドされる暗号のリスト。優先順位はリスト内の順序によって決定され、暗号スイート、事前定義された暗号グループ、またはユーザー作成の暗号グループを含めることができます。	いいえ	いいえ

重要

- 事前設定されたSSLプロファイルを使用する場合、NetScalerを介して手動で暗号をバインドする必要があります、`spec.policies.sslciphers` は無視されます。
- 組み込みの暗号グループはTier-1およびTier-2 NetScalerでサポートされていますが、ユーザー定義の暗号グループはTier-1 NetScalerでのみサポートされています。

例1：事前設定されたフロントエンドSSLプロファイル

```
apiVersion: citrix.com/v1

kind: sslprofile

metadata:

  name: frontend-ssl-preconfigured

spec:

  targetRef:

    - name: my-gateway-https2

    sectionName: https

  preconfigured: "ns_default_ssl_profile_frontend"
```

例2：カスタムフロントエンドSSLプロファイル

```
apiVersion: citrix.com/v1

kind: sslprofile

metadata:

  name: frontend-ssl-custom

spec:

  targetRef:
```

```
- name: my-gateway-https # Applying to all csvservers in the Gateway

sslciphers:

- "HIGH"

config:

  sslprofiletype: "frontend"

  sessreuse: "ENABLED"

  sesstimeout: 300

  clientauth: "DISABLED"

  ssl3: "DISABLED"

  tls1: "DISABLED"

  tls11: "DISABLED"

  includesubdomains: "YES"

  maxage: 31536000

  preload: "YES"
```

例3：事前設定されたバックエンドSSLプロファイルと、そのプロファイルを参照する HTTPRoute

```
apiVersion: citrix.com/v1

kind: sslprofile

metadata:

  name: backend-ssl-preconfigured

spec:
```

```
preconfigured: "ns_default_ssl_profile_backend"
```

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

```
kind: HTTP Route
```

```
metadata:
```

```
  name: my-httproute
```

```
spec:
```

```
  parentRefs:
```

```
    - name: my-gateway-https
```

```
  hostnames:
```

```
    - "example123.com"
```

```
  rules:
```

```
    - filters:
```

```
      - type: ExtensionRef
```

```
        extensionRef:
```

```
          group: "citrix.com"
```

```
          kind: "sslprofile"
```

```
          name: "backend-ssl-preconfigured"
```

```
  matches:
```

```
    - path:
```

```
      type: PathPrefix
```

```
      value: /
```

```
backendRefs:  
  
  - name: app1-svc  
  
    namespace: default  
  
    port: 443
```

例4：カスタムバックエンドSSLプロファイルと、そのプロファイルを参照するHTTPRoute

```
apiVersion: citrix.com/v1  
  
kind: sslprofile  
  
metadata:  
  
  name: backend-ssl-custom  
  
spec:  
  
  sslciphers:  
  
    - "HIGH"  
  
  config:  
  
    sslprofiletype: "backend"  
  
    sessreuse: 'ENABLED'  
  
    sesstimeout: 300  
  
    serverauth: 'ENABLED'  
  
    insertionencoding: 'Unicode'  
  
    ssl3: 'ENABLED'  
  
    tls1: 'DISABLED'  
  
    tls12: 'ENABLED'
```

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

kind: HTTPRoute

metadata:

  name: my-httproute2

spec:

  parentRefs:

    - name: my-gateway-https2

  hostnames:

    - "example123.com"

  rules:

    - filters:

        - type: ExtensionRef

        extensionRef:

          group: "citrix.com"

          kind: "sslprofile"

          name: "backend-ssl-custom"

    matches:

      - path:

          type: PathPrefix

          value: /

    backendRefs:

      - name: app1-svc
```

```
port: 443
```

TCPプロファイル

TCPプロファイルは、トランスポート層の動作を最適化するために使用されるTCP設定の集合体です。このプロファイルを使用すると、TCP Fast OpenやHyStartなどの高度な設定を適用して、接続のパフォーマンスと効率を向上させることができます。

TCPプロファイルCRDスキーマ参照

フィールド	タイプ	説明	フロントエンド プロファイルに 必須	バックエンドプ ロファイルに必 須
事前設定済み	文字列	config とは相互排他的です。使用する NetScaler アプリケーションに存在する、事前設定済みまたは組み込みの TCP プロファイルの名前。	いいえ	いいえ
設定	オブジェクト	preconfigured とは相互排他的です。完全にカスタムなプロファイルのために、NetScaler NITRO API に関連するすべてのパラメーターが含まれます。お客様は、任意の有効な NITRO TCP プロファイルパラメーターを指定できます。TCP プロファイルの	いいえ	いいえ

フィールド	タイプ	説明	フロントエンド プロファイルに 必須	バックエンドプ ロファイルに必 須
		すべての可能な キーと値のペア については、 TCP profile を 参照してくださ い。		
targetRef	オブジェクトの 配列	このプロファイ ルが適用される ターゲットリ ソース（通常は Gateway）のリ スト。この フィールドはフ ロントエンドプ ロファイルに必 須であり、バッ クエンドプロ ファイルでは省 略する必要があります。	はい	いいえ
gatewayClassN ame	文字列	このTCPプロ ファイルを適用 する必要がある GatewayClass の名前。マルチ テナント環境 で、プロファイ ルを特定のコン トローラーイン スタンスにス コープするため に使用されま す。	いいえ	いいえ

例1：事前設定されたフロントエンドTCPプロファイル

```
apiVersion: citrix.com/v1

kind: tcpprofile

metadata:

  name: frontend-tcp-preconfigured

spec:

  targetRef:

    - name: my-gateway-https2

      sectionName: https    # Applying to specific csvserver/Listener in the
Gateway

  preconfigured: "nstcp_default_profile"
```

例2：カスタムフロントエンドTCPプロファイル

```
apiVersion: citrix.com/v1

kind: tcpprofile

metadata:

  name: frontend-tcp-custom

spec:

  targetRef:

    - name: my-gateway-https # Applying to all csvservers in the Gateway

  config:

    mss: 1000

    sack: ENABLED
```

```
    buffersize: 10000

    ka: ENABLED

    name: "dummy"      # Will be ignored, tcpprofile will be generated based
                        # on metadata.name and metadata.namespace
```

例3：事前設定されたバックエンドTCPプロファイルと、そのプロファイルを参照する HTTPRoute

```
apiVersion: citrix.com/v1

kind: tcpprofile

metadata:

  name: backend-tcp-preconfigured

spec:

  preconfigured: "nstcp_default_profile"
```

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

kind: HTTPRoute

metadata:

  name: my-httproute

spec:

  parentRefs:

    - name: my-gateway-https

  hostnames:

    - "example123.com"

  rules:
```

```
- filters:

- type: ExtensionRef

  extensionRef:

    group: "citrix.com"

    kind: "tcpprofile"

    name: "backend-tcp-preconfigured"

  matches:

    - path:

        type: PathPrefix

        value: /

    backendRefs:

      - name: app1-svc

        port: 443
```

例4：カスタムバックエンドTCPプロファイルと、そのプロファイルを参照するHTTPRoute

```
apiVersion: citrix.com/v1

kind: tcpprofile

metadata:

  name: backend-tcp-custom

spec:

  config:

    mss: 1000
```

```
sack: ENABLED
```

```
bufferSize: 10000
```

```
ka: ENABLED
```

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

```
kind: HTTPRoute
```

```
metadata:
```

```
  name: my-httproute2
```

```
spec:
```

```
  parentRefs:
```

```
    - name: my-gateway-https2
```

```
  hostnames:
```

```
    - "example123.com"
```

```
  rules:
```

```
    - filters:
```

```
      - type: ExtensionRef
```

```
        extensionRef:
```

```
          group: "citrix.com"
```

```
          kind: "sslprofile"
```

```
          name: "backend-ssl-custom"
```

```
- type: ExtensionRef

extensionRef:

  group: "citrix.com"

  kind: "tcpprofile"

  name: "backend-tcp-custom"

  # Applying SSL and TCPProfile to the same rule

matches:

- path:

  type: PathPrefix

  value: /

backendRefs:

- name: app1-svc

  namespace: default

  port: 443
```

HTTPプロファイル

HTTPプロファイルは、WebSocketの有効化、HTTPタイムアウトのカスタマイズ、厳格な検証ルールの設定など、さまざまなアプリケーション層の動作を設定するために使用されるHTTP設定の集合体です。

HTTPプロファイルCRDスキーマ参照

フィールド	タイプ	説明	フロントエンド プロファイルに 必須	バックエンドプ ロファイルに必 須
事前設定済み	文字列	config とは相互 排他的です。使	いいえ	いいえ

フィールド	タイプ	説明	フロントエンド プロファイルに 必須	バックエンドプ ロファイルに必 須
		用する NetScaler アプ ライアンスに存 在する、事前設 定済みまたは組 み込みの HTTP プロファイルの 名前。		
設定	オブジェクト	preconfigured とは相互排他的 です。完全にカ スタムなプロ ファイルのため の NetScaler NITRO API に 関連するすべて のパラメーター が含まれます。 顧客は任意の有 効な NITRO TCP プロファ イルパラメー ターを指定でき ます。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。	いいえ	いいえ
targetRef	オブジェクトの 配列	このプロファイ ルが適用される ターゲットリ ソース (通常は Gateway) のリ スト。この	はい	いいえ

フィールド	タイプ	説明	フロントエンド プロファイルに 必須	バックエンドプ ロファイルに必 須
		フィールドはフ ロントエンドプ ロファイルには 必須であり、 バックエンドプ ロファイルには 省略する必要が あります。		
gatewayClassN ame	文字列	このHTTPプロ ファイルを適用 する必要がある GatewayClass の名前。マルチ テナント環境で プロファイルを 特定のコント ローラーインス タンスにスコー プするために使 用されます。	いいえ	いいえ

例1: 事前設定されたフロントエンドHTTPプロファイル

```
apiVersion: citrix.com/v1

kind: httpprofile

metadata:

  name: frontend-http-preconfigured

spec:

  targetRef:

    - name: my-gateway-https

    kind: Gateway
```

```
preconfigured: "frontend1"
```

例2: カスタムフロントエンドHTTPプロファイル

```
apiVersion: citrix.com/v1

kind: httpprofile

metadata:

  name: frontend-http-custom

spec:

  targetRef:

    - name: my-gateway-https

      kind: Gateway

    group: gateway.networking.k8s.io    # Applying to all CSvserver in the
Gateway

  config:

    websocket: "ENABLED"

    http2: ENABLED

    reqtimeout: 10

    dropinvalreqs: ENABLED
```

例3: 事前設定されたバックエンドHTTPプロファイルと、そのプロファイルを参照する HTTPRoute

```
apiVersion: citrix.com/v1

kind: httpprofile

metadata:
```

```
  name: backend-http-preconfigured

spec:

  preconfigured: "nshttp_default_profile"
```

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

kind: HTTPRoute

metadata:

  name: my-httproute

  namespace: default

spec:

  parentRefs:

    - name: my-gateway-https

  hostnames:

    - "example123.com"

  rules:

    - filters:

        - type: ExtensionRef

          extensionRef:

            group: "citrix.com"

            kind: "tcpprofile"

            name: "backend-tcp-preconfigured"

    matches:
```

```
- path:

  type: PathPrefix

  value: /

  backendRefs:

    - name: app1-svc

      namespace: default

      port: 443
```

例4: カスタムバックエンドHTTPプロファイルと、そのプロファイルを参照するHTTPRoute

```
apiVersion: citrix.com/v1

kind: httpprofile

metadata:

  name: backend-http-custom

spec:

  config:

    websocket: "ENABLED"

    http2: ENABLED

    reqtimeout: 10

    dropinvalreqs: ENABLED
```

```
# HTTPRoute referring custom backend sslprofile

apiVersion: gateway.networking.k8s.io/v1

kind: HTTPRoute
```

```
metadata:

  name: my-httproute2

  namespace: default

spec:

  parentRefs:

    - name: my-gateway-https2

      namespace: default

  hostnames:

    - "example123.com"

  rules:

    - filters:

        # Here we are referring to all types of profiles for this rule.

    - type: ExtensionRef

      extensionRef:

        group: "citrix.com"

        kind: "sslprofile"

        name: "backend-ssl-custom"

    - type: ExtensionRef

      extensionRef:

        group: "citrix.com"

        kind: "httpprofile"

        name: "backend-http-custom"
```

```
- type: ExtensionRef

extensionRef:

  group: "citrix.com"

  kind: "tcpprofile"

  name: "backend-tcp-custom"

matches:

- path:

  type: PathPrefix

  value: /

backendRefs:

- name: app1-svc

  namespace: default

  port: 443
```

`spec.gatewayClassName` によるコントローラーのスコープ設定

`spec.gatewayClassName` フィールドは、どのNetScaler Gateway ControllerインスタンスがプロファイルCRDを処理するかを制御します。この制御は、マルチテナントまたはマルチコントローラー環境において非常に重要です。

シナリオ	SPEC.GATEWAYCLASSNAME	コントローラーの動作
スコープ設定済み	存在する（例： <code>netScaler-prod</code> ）	その <code>specific GatewayClass</code> を管理するコントローラーインスタンスのみがプロファイルを処理し、分離とターゲット設定を保証します。

シナリオ	SPEC.GATEWAYCLASSNAME	コントローラーの動作
グローバル	なし	クラスター内のすべての NetScaler Gateway Controller インスタンスは、それぞれの NetScaler アプライアンス上でプロファイルを受け入れ、設定することができます。

プロファイルが、現在クラスターにデプロイされていない `gatewayClassName` を指定している場合、プロファイルCRDはスキップされ、対応する `GatewayClass` が利用可能になるまで待機します。

トラフィックセキュリティ

トラフィックセキュリティには、以下のトピックがあります。

- 事前設定された証明書の使用
- NetScaler Kubernetes Gateway Controller を使用した NetScaler の証明書キーバンドル
- バックエンドアプリケーションの通信プロトコルを定義するための「appProtocol」の使用

NetScaler Kubernetes Gateway Controller で事前設定された証明書を使用する

このトピックでは、NetScaler アプライアンスにすでに設定されている証明書または証明書バンドルを Kubernetes Gateway で使用する方法について説明します。この機能により、既存の証明書を参照できるため、Kubernetes Secrets として管理する手間が省け、証明書管理が簡素化されます。

利点

NetScaler Kubernetes Gateway Controller で事前設定された証明書を使用する利点の一部を以下に示します。

- **管理の簡素化:** NetScaler アプライアンスにすでに設定されている証明書を使用します。
- **シークレット作成不要:** 証明書用の Kubernetes Secrets の作成と維持にかかる手間を省きます。
- **一元管理:** Kubernetes で証明書を設定しながら、NetScaler で直接証明書を管理します。

事前設定された証明書を参照する方法

Gateway を定義する際、リスナー設定の `certificateRefs` セクションで事前設定された証明書または証明書バンドルを参照できます。

証明書の種類

次の2種類の事前設定された証明書または証明書バンドルを参照できます。

1. **ドメイン証明書:** HTTPS 接続用のサーバー証明書。
2. **CA 証明書:** クライアント認証用の認証局証明書。複数の CA 証明書を指定できます。

基本設定

事前設定された証明書を参照する簡単な例を次に示します。

NetScaler のセットアップ

```
add ssl certkey my-default-cert -cert /nsconfig/ssl/server.crt -key /  
nsconfig/ssl/server.key
```

```
add ssl certkeybundle my-cert-bundle -bundlefile /nsconfig/ssl/bundle.pem
```

または、事前設定された証明書バンドルを使用します。

```
apiVersion: gateway.networking.k8s.io/v1  
  
kind: Gateway  
  
metadata:  
  
  name: my-gateway-with-preconf-certs  
  
  namespace: default  
  
spec:  
  
  gatewayClassName: my-gateway-class  
  
  addresses:  
  
    - type: IPAddress  
  
      value: "<IP ADDRESS>"  
  
  listeners:  
  
    - name: https-listener  
  
      protocol: HTTPS  
  
      port: 443  
  
      tls:
```

```
    mode: Terminate

    certificateRefs:

      - group: "netscaler.certkeybundle"

        kind: "default" # Only one domain cert. Should be bound as
        default unless kind is specified.

        name: "my-cert-bundle" # <---- Name of ssl certkeybundle
        configured on netscaler
```

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

kind: Gateway

metadata:

  name: my-gateway-with-preconf-certs

  namespace: default

spec:

  gatewayClassName: my-gateway-class

  addresses:

    - type: IPAddress

      value: "<IP ADDRESS>"

  listeners:

    - name: https-listener

      protocol: HTTPS

      port: 443

      tls:
```

```
mode: Terminate

certificateRefs:

- group: "netscaler.certkey"

    kind: "default"          # Only one domain cert. Should be bound as
default unless kind is specified.

    name: "my-default-cert" # <---- Name of ssl certkey configured on
netscaler
```

設定フィールドの説明は次のとおりです。

- **group** : 証明書のソースを指定します。
 - 事前設定されたNetScaler証明書の場合は「netscaler.certkey」
 - 事前設定されたNetScaler証明書バンドルの場合は「netscaler.certkeybundle」
- **kind** : 証明書の種類を指定します。
 - 「ca」 - 認証局証明書（クライアント認証用）に使用します。複数のCA証明書を指定できます。
 - 「sni」 - SNIとしてバインドする必要があるサーバー証明書に使用します。

注:

kind はサーバー証明書ではオプションです。種類が指定されていないサーバー証明書が1つだけ存在する場合、それはデフォルトとしてバインドされます。複数のサーバー証明書が存在する場合、それらは自動的にSNI証明書としてバインドされます。

- **name** : NetScalerアプライアンスで設定されている証明書の名前

一般的なシナリオ

シナリオ1：単一証明書

HTTPSリスナーに単一の事前設定済み証明書を使用します。

```
listeners:

- name: https

  protocol: HTTPS

  port: 443

  tls:

    mode: Terminate

    certificateRefs:

      - group: "netscaler.certkey"

        name: "company-ssl-cert"
```

シナリオ 2: 複数ドメイン証明書

複数のサーバー証明書が提供される場合、`snienable` を `ENABLED` に設定したSSLプロファイルを作成する必要があります:

```
listeners:

- name: https

  protocol: HTTPS

  port: 443

  tls:

    mode: Terminate

    certificateRefs:

      - group: "netscaler.certkey"

        name: "primary-domain-cert"

      - group: "netscaler.certkey"
```

```
name: "secondary-domain-cert"
```

必要なSSLプロファイル

```
apiVersion: citrix.com/v1

kind: sslprofile

metadata:

  name: frontend-ssl-custom

spec:

  targetRef:

    - name: my-gateway-https

    sectionName: https

  config:

    sslprofiletype: "frontend"

    snienable: "ENABLED"
```

シナリオ 3: CA証明書によるクライアント認証

クライアント認証のためにCA証明書を追加します:

```
listeners:

  - name: https

    protocol: HTTPS

    port: 443

    tls:

      mode: Terminate
```

```
certificateRefs:

- group: "netscaler.certkey"

  kind: "default"

  name: "server-cert"

- group: "netscaler.certkey"

  kind: "ca"

  name: "client-ca-cert"
```

シナリオ 4: 証明書バンドルの使用

NetScalerで証明書バンドルが設定されている場合、`netscaler.certkeybundle` グループを使用します:

```
listeners:

- name: https

  protocol: HTTPS

  port: 443

  tls:

    mode: Terminate

    certificateRefs:

      - group: "netscaler.certkeybundle"

        name: "my-cert-bundle"
```

シナリオ 5: 事前設定済み証明書とKubernetes証明書の組み合わせ

事前設定済み証明書を通常のKubernetesシークレットと組み合わせることができます:

```
listeners:

- name: https

  protocol: HTTPS

  port: 443

  tls:

    mode: Terminate

    certificateRefs:

      - kind: Secret

        name: "k8s-tls-secret"

      - group: "netscaler.certkey"

        name: "additional-domain-cert"
```

前提条件

Gatewayで事前設定済み証明書または証明書バンドルを使用する前に:

1. 証明書がNetScalerアプライアンスにすでに設定されていることを確認してください。
2. NetScalerに表示されているとおりの正確な証明書名をメモし、Gateway CRDでそれを参照してください。
3. Gateway CRDインスタンスのデプロイ後に証明書がNetScalerで作成された場合、適切なバイndディングを確実にするためにGatewayリソースのYAMLを削除して再適用してください。

トラブルシューティング

一般的な問題とその解決策は次のとおりです。

- 問題: 証明書が見つかりません

解決策: 証明書名がNetScalerで設定されている名前と完全に一致していることを確認してください。

- 問題: 誤った証明書が提供されました

解決策: 複数の証明書 (SNI) を使用する場合は、以下を確認してください。

- snienableがENABLEDに設定されたSSLプロファイルが作成され、SSL仮想サーバーにバインドされていること。
- SSLProfile CRDを使用してSSLプロファイルを作成できます - ゲートウェイを参照するSSLProfile構成の例:

```
apiVersion: citrix.com/v1

kind: sslprofile

metadata:

  name: frontend-ssl-custom

spec:

  targetRef:

    - name: my-gateway-https

      sectionName: https

  config:

    sslprofiletype: "frontend"

    snienable: "ENABLED"
```

概要

Kubernetes Gatewayで事前設定された証明書を参照するのは簡単で、次のように要約できます。

1. 通常の証明書には group: "netscaler.certkey" を使用します
2. 証明書バンドルには group: "netscaler.certkeybundle" を使用します
3. CA証明書には kind: "ca" を指定します。サーバー証明書の場合、kindはオプションです
4. NetScalerで設定されている証明書名を提供します

NetScaler® で NetScaler Kubernetes Gateway Controller を使用した証明書キーバンドル

NetScaler Kubernetes Gateway Controller は、NetScaler 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 Kubernetes Gateway Controller は、Gateway リソースで提供される証明書の証明書バンドルを作成します。

この機能を有効にするには、NetScaler Kubernetes Gateway Controller に新しい環境変数 `CERT_BUNDLE` が追加され、Helm Charts から `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 Kubernetes Gateway Controller](#) の Helm Charts を参照してください。NetScaler Kubernetes Gateway 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、次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。

注意点

1. NetScaler での証明書バンドルの作成は、次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。証明書はシーケンスで配置する必要があります、すべての中間証明書がバンドルに存在する必要があります。
- 証明書は PEM 形式のみである必要があります。
- 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が削除され、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。

「appProtocol」を使用してバックエンドアプリケーション通信プロトコルを定義する

NetScaler Kubernetes Gateway Controllerは、Kubernetesサービスポート上の `appProtocol` フィールドを使用して、NetScalerとバックエンドアプリケーション間の通信を制御する、結果として得られるNetScalerサービスグループのプロトコルを明示的に設定します。

例えば、サービスポートで `appProtocol: ssl` を設定すると、NetScaler上の対応するサービスグループがSSLプロトコルで構成されることが保証されます。明示的に設定されていない場合、デフォルトのサービスグループプロトコルはHTTPです。

```
apiVersion: v1

kind: Service

metadata:

  name: my-secure-backend-service

spec:

  selector:

    app: secure-app

  ports:

    - name: https-to-ssl

      protocol: TCP

      port: 443

      targetPort: 443

      # CRITICAL: This line tells NetScaler to use SSL for the backend
      connection

      appProtocol: ssl
```

AIゲートウェイ

NetScaler® AI Gateway は、アプリケーションと大規模言語モデル (LLM) バックエンドの間を流れる AI トラフィックを企業が管理できるようにします。NetScaler Kubernetes Gateway Controller は、AI Gateway の機能を Kubernetes 環境に拡張し、NetScaler インスタンスでの手動 CLI または GUI 設定なしで、Kubernetes ネイティブのリソースを使用して AI Gateway ポリシーを設定および管理できるようにします。

NetScaler Kubernetes Gateway Controller を使用すると、既存のアプリケーションワークロードとともに、CRD とアノテーションベースのポリシーを通じて AI Gateway の設定を自動化できます。

概要

AI Gateway を使用するタイミング

- アプリケーションが Kubernetes 上で実行され、Azure OpenAI LLM を呼び出す場合。
- トークン認識ルーティング、レート制限、プロンプト検査、およびスピルオーバーを備えた、すべての AI トラフィックに対する単一の管理されたエントリポイントが必要な場合。
- Kubernetes マニフェストとして定義され、GitOps ワークフローを通じて管理される AI Gateway ポリシーが必要な場合。
- クラスター外の外部アプリケーションまたはレガシーシステムが、Kubernetes 内で実行されている LLM バックエンドへの管理されたアクセスを必要とする場合。

展開オプション

Gateway Controller は、トラフィックの発生元に応じて、2 つの NetScaler フォームファクターをサポートしています。

展開	フォームファクター	使用条件
クラスター内	NetScaler CPX	クライアントアプリケーションはKubernetesクラスター内で実行されます。自己ホスト型LLMバックエンドへの東西トラフィックと、外部プロバ

展開	フォームファクター	使用条件
		イダーへのエグレスの両方を処理します。
外部	NetScaler VPX	クライアントアプリケーションはクラスター外で実行されます — 外部アプリ、レガシーシステム、またはLLMバックエンドに到達する他のクラスター。

アーキテクチャ

ゲートウェイコントローラーはコントロールプレーンに配置されます。Kubernetesリソースを監視し、NetScalerをプログラムします — トラフィックパスにはありません。推論リクエストはクライアントアプリケーションからNetScalerへ、そしてLLMバックエンドへと直接流れます。

AIゲートウェイのアーキテクチャ(/en-us/netscaler-kubernetes-gateway-controller/media/ai-gateway-architecture.png)

主要なKubernetesリソース

リソース	定義元	目的
GatewayClass	Gateway API (アップストリーム)	ゲートウェイをNetScalerコントローラー (CPXまたはVPX) にバインドします
Gateway	Gateway API (アップストリーム)	AI Gatewayフロントエンドプロファイルでリスナーを定義します
AIGatewayRoute	NetScaler CRD	モデルルーティングルール — モデル名で一致し、バックエンドサービスにルーティングします
RateLimit	NetScaler CRD	トークンベースのクォータ — ユーザーごと、組織ごと、モデルごと、またはゲートウェイごと

リソース	定義元	目的
<code>RewritePolicy</code>	NetScaler CRD	プロンプトの変更とスピルオーバーのためのURL書き換え

`GatewayClass` と `Gateway` は標準のGateway APIリソースです。 `AIGatewayRoute` 、 `RateLimit` 、 および `RewritePolicy` は、Gateway ControllerとともにインストールされるNetScaler固有のCRDであり、他のGateway API実装には移植できません。

注記：

外部プロバイダー（Azure OpenAI）に送信されるAIトラフィック、およびバックアッププロバイダーへのスピルオーバーは、そのプロバイダーによってリクエストごとに課金されます。トークンクォータ（`aigatewayprofile`）とレート制限（[レート制限とストリームの構成](#)）を使用して、費用を管理してください。

関連トピック

- モデルルーティングの構成(model-routing.html)
- レート制限とストリームの構成(rate-limits-and-streams.html)
- プロンプト管理の構成(prompt-management.html)
- 代替モデルへのスピルオーバーの構成(spillover-to-alternate-model.html)
- 可観測性(..observability.html)

リファレンス

NetScalerレベルでのAI Gateway構成については、[AI機能](#)を参照してください。

AI機能を持つNetScaler Kubernetes Gateway Controllerを展開する

NetScaler Kubernetes Gateway Controller 2.2.0リリース以降、NetScaler AI Gateway機能がサポートされています。NetScaler AI Gatewayの詳細については、[NetScaler AI Gateway](#)を参照してください。

トラフィックの発生元に応じて、以下のいずれかの展開オプションを使用してNetScaler® AI Gateway機能を有効にできます。

- **NetScaler Kubernetes Gateway ControllerとNetScaler VPX/MPX/BLX** — NetScaler Kubernetes Gateway Controllerはクラスター内で実行され、外部のNetScaler VPX/MPX/BLXを構成します。
- **NetScaler CPXとGateway Controller** — NetScaler CPXとNetScaler Kubernetes Gateway Controllerはクラスター内で一緒に実行されます。

前提条件

- `kubectl` が構成されたKubernetesクラスター。
- Helm 3.x以降。
- [公式標準チャネル](#)からインストールされたGateway API CRD。

NetScaler VPX/MPX/BLX を使用して NetScaler Kubernetes Gateway Controller を展開する

NetScaler Kubernetes Gateway Controllerが外部のNetScaler VPX/MPX/BLXを管理する場合に、このオプションを使用します。

NetScaler Kubernetes Gateway Controllerをインストールする

リリース名 `my-release` で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 my-release netscaler/netscaler-kubernetes-gateway-controller
--set
license.accept=yes,gatewayController.gatewayControllerName=<controllerName>,
netscaler.nsIP=<NSIP>,netscaler.adcCredentialSecret=<Secret-for-NetScaler-credentials>
```

注:

デフォルトでは、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インストールします。

このコマンドは、デフォルト設定でKubernetesクラスターにNetScaler Kubernetes Gateway Controllerをデプロイします。[configuration](#)セクションには、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ーがリストされています。

GatewayClassを作成する

NetScaler Kubernetes Gateway Controllerがインストールされたら、コントローラーにバインドする `GatewayClass` を作成します。 `controllerName` は、インストール中に使用された `gatewayController.gatewayControllerName` の値と一致する必要があります。

```
kubectl apply -f - <<EOF

apiVersion: gateway.networking.k8s.io/v1

kind: GatewayClass

metadata:

  name: aigwclass

spec:

  controllerName: citrix.com/nsgw-controller

EOF
```

注:

すべてのAI Gatewayリソース（ `Gateway`、 `aigatewayprofile`、 `aigatewayroute`、 `ratelimit`、および `rewritepolicy` ）の `gatewayClassName` フィールドは、ここで作成された `GatewayClass` の `metadata.name` と一致する必要があります（ `aigwclass` ）。

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

このオプションを使用して、NetScaler CPXとGateway Controllerをクラスター内で一緒に実行します。

Gateway Controllerを使用してNetScaler CPXをインストールする

リリース名 `my-release` でチャートをインストールするには：

```
helm install my-release netscaler/netscaler-cpx-with-gateway-controller --set
```

```
license.accept=yes,gatewayController.gatewayControllerName=<controllerName>
```

注:

デフォルトでは、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インストールします。

このコマンドは、デフォルト設定でKubernetesクラスターにNetScaler Kubernetes Gateway Controllerをサイドカーとして持つNetScaler CPXをデプロイします。[configuration](#)セクションには、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ーがリストされています。

GatewayClassを作成する

Gateway Controllerを搭載したNetScaler CPXがインストールされたら、コントローラーにバインドする `GatewayClass` を作成します。 `controllerName` は、インストール中に使用された `gatewayController.gatewayControllerName` の値と一致する必要があります。

```
kubect1 apply -f - <<EOF

apiVersion: gateway.networking.k8s.io/v1

kind: GatewayClass

metadata:

  name: aigwclass

spec:

  controllerName: citrix.com/nscpxgw-controller

EOF
```

注:

すべてのAI Gatewayリソース(`Gateway` 、 `aigatewayprofile` 、 `aigatewayroute` 、 `ratelimit` 、 および `rewritepolicy`)の `gatewayClassName` フィールドは、ここで作成された `GatewayClass` の `metadata.name` と一致する必要があります(`aigwclass`)。

デプロイの検証

コントローラーが実行中であり、AI Gatewayリソースを適用する前に `GatewayClass` が承認されていることを確認してください。

```
kubect1 get pods  
  
kubect1 get gatewayclass aigwclass
```

コントローラーポッドは `STATUS Running` と表示される必要があります(CPXの場合、ポッドは `READY 2/2` と表示されます)。また、`GatewayClass` は `ACCEPTED True` と表示される必要があります。

参照

- [NetScaler クーバネティス ゲートウェイ コントローラー Helmチャート](#)
- [NetScaler シーピーエックス ゲートウェイ コントローラー Helmチャート](#)

モデルルーティングを設定する

NetScaler® AI Gateway は、受信した推論リクエストを、リクエスト内のモデル名に基づいて適切な大規模言語モデル (LLM) バックエンドにルーティングします。この機能により、すべてのモデルに対して単一のGatewayエンドポイントが提供されます。クライアントはバックエンドサービスのIPアドレスやトポロジを知る必要がありません。

モデルのアップグレード、バックエンドの変更、および新しいモデルの追加は、アプリケーションコードの変更を必要としません。

前提条件

- NetScaler Kubernetes Gateway Controller が必要な構成で展開されていること。
- GatewayClass が作成され、NetScaler インスタンスにバインドされていること。
- AI推論サービスがKubernetesの外部サービスとして展開されていること。

構成

AI Gateway プロファイル

AI Gateway プロファイルは、NetScaler がアプリケーションとLLMバックエンド間のAIトラフィックを処理する際に適用する設定を定義します。プロファイルは `frontend` タイプ（クライアントに面する仮想サーバーに適用）または `backend` タイプ（LLMバックエンドに接続する

サービスグループに適用) のいずれかです。詳細については、[aigatewayprofile CRD ドキュメント](#) を参照してください。

注:

AI Gateway プロファイルが作成された後、それを変更したり、Gateway または AIGatewayRoute にアタッチされている AI Gateway プロファイルを変更したりすると、関連する NetScaler エンティティが再作成されます。その結果、トラフィックに影響が出ます。

フロントエンド AI Gateway プロファイル

フロントエンドプロファイルは、AI Gateway エンドポイントタイプを定義します。

Gateway 用フロントエンド AIGateway プロファイル

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1

kind: aigatewayprofile

metadata:

  name: csaigwyfp

spec:

  gatewayClassName: "aigwclass"

  config:

    endpointtype: "azureopenai"

    profiletype: "frontend"

EOF
```

AIGatewayRoute 用のフロントエンド AIGateway プロファイル

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1
```

```
kind: aigatewayprofile

metadata:

  name: lbaigwyfp

spec:

  gatewayClassName: "aigwclass"

  config:

    endpointtype: "azureopenai"

    profiletype: "frontend"

EOF
```

バックエンド AI Gateway プロファイル

バックエンドプロファイルは、モデルごとのクォータと認証設定を定義します。

AI Gateway エンドポイントの認証トークン/API キーを含む Kubernetes Secret を作成します。

`<auth-token>` を AI Gateway エンドポイントの認証トークンに置き換えます。

```
kubectl create secret generic aigw-auth --from-literal=authToken=<auth-token>
```

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1

kind: aigatewayprofile

metadata:

  name: aigwybp

spec:
```

```
gatewayClassName: "aigwclass"

config:

  authToken:

    name: aigw-auth

    namespace: default

  endpointtype: "azureopenai"

  profiletype: "backend"

  tokenquota: 25000

  quotarefreshfrequency: 1

---

EOF
```

AI Gateway プロファイルを持つ Kubernetes Gateway

Kubernetes Gateway は、AI の受信トラフィックのエントリポイントを定義します。フロントエンド AI Gateway プロファイルを Gateway に追加するには、`infrastructure` 参照の下にフロントエンドプロファイルの名前を追加します。このフロントエンドプロファイルは、NetScaler で AI トラフィックの前面にあるコンテンツスイッチング (CS) 仮想サーバーに適用されます。

```
kubectl apply -f - <<EOF

apiVersion: gateway.networking.k8s.io/v1

kind: Gateway

metadata:

  name: aigw

spec:

  gatewayClassName: aigwclass
```

```
infrastructure:

  parametersRef:

    group: citrix.com

    kind: aigatewayprofile

    name: csaigwyfp

  listeners:

  - name: https

    protocol: HTTPS

    port: 443

    tls:

      mode: Terminate

      certificateRefs:

      - kind: Secret

        name: web-ingress-secret

        group: ""

    allowedRoutes:

      namespaces:

        from: All

  ---

EOF
```

Gateway オブジェクトのフィールドについては、[Gateway Documentation](#) を参照してください。

注:

- Gateway オブジェクトには、フロントエンド AI Gateway プロファイルのみを適用できます。
- Gateway を適用する前に、Gateway リスナーによって参照される TLS シークレット `web-ingress-secret` を作成します。証明書とキーのパスを独自のものに置き換えてください。

```
kubectl create secret tls web-ingress-secret --cert=<path-to-tls.crt> --key=<path-to-tls.key>
```

NetScaler AI Gateway ルート

モデル名でリクエストを照合し、各ルールを対応するバックエンドサービスに転送するルーティングルールを定義します。 `AIGatewayRoute` では、フロントエンド AI Gateway プロファイルがロードバランシング (LB) 仮想サーバーに適用され、バックエンド AI Gateway プロファイルが LLM バックエンドに接続するサービスグループに適用されます。詳細については、[aigatewayroute CRD documentation](#) を参照してください。

NetScaler PolicyExpression を使用した複雑なルーティング

リクエストの任意のパートでルーティングするには、 `matches` の NetScaler `policyExpression` を使用します。たとえば、このサンプルはリクエストボディを検査し、ボディに `gpt` が含まれている場合に `gpt-model-svc` にルーティングします。

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1

kind: aigatewayroute

metadata:

  name: ai-route-model

spec:

  parentRefs:
```

```

- name: aigw

aigatewayprofiles:

  frontend:

    name: lbaigwyfp

  default-backend:

    name: aigwybp

rules:

- matches:

- policyExpression: HTTP.REQ.BODY(30).CONTAINS("gpt")

  backendRefs:

    - name: gpt-model-svc

      port: 80

---

EOF

```

Kubernetes ネイティブな方法でのシンプルなルーティング

標準のGateway API `path` マッチを使用して、簡単なルーティングを行います。例えば、このサンプルは、`/` パスプレフィックス下のすべてのリクエストを `gpt-model-svc` にルーティングします。

```

kubectl apply -f - <<EOF

apiVersion: citrix.com/v1

kind: aigatewayroute

metadata:

```

```
    name: ai-route-model

spec:

  parentRefs:

    - name: aigw

  aigatewayprofiles:

    frontend:

      name: lbaigwyfp

    default-backend:

      name: aigwybp

  rules:

    - matches:

      - path:

          type: PathPrefix

          value: "/"

      backendRefs:

        - name: gpt-model-svc

          port: 80

  ---

EOF
```

参照

[コンテンツスイッチングベースのモデルルーティング](#)

レート制限とストリームを構成する

レート制限は、Gateway および/または AIGatewayRoute でクォータを適用し、ユーザーごとおよび組織ごとの LLM リソース使用量を正確に制御できるようにします。リクエスト数、実際のトークン消費量、またはその両方で制限できます。

トークンベースのレート制限 (`mode: TOKEN_RATE`) は、リクエスト数による制限とは異なり、実際の GPU リソース使用量を反映する実際のトークン消費量を測定します。

設定されたクォータを超えるリクエストは、LLM バックエンドに到達する前に `429 Too Many Requests` 応答を受け取ります。

重要:

レート制限は、LLM の支出を制限するための主要な制御です。トークンベースの制限 (`mode: TOKEN_RATE`) がないと、単一のユーザーまたは組織が有料プロバイダーに対して無制限のトークンを消費し、コストの上限がなくなります。支出を制限するために、ユーザーごとおよび組織ごとに `TOKEN_RATE` 制限を構成してください。

ゲートウェイごとのレート制限とモデルごとのレート制限

レート制限は、Gateway レベルでグローバルに (すべてのモデルルートにわたって) 適用することも、 `AIGatewayRoute` を介して特定の AI モデルに適用することもできます。

タイプ	バインド方法	スコープ
モデルごと	<code>AIGatewayRoute</code> ルール内の <code>filters[].extensionRef</code>	単一のモデルルートのみ
ゲートウェイごと	<code>targetRef</code> が <code>Gateway</code> リソースを指す	ゲートウェイ上のすべてのルート

注:

モデルごとのレート制限とグローバルレート制限の両方を同時にアクティブにできます。モデルごとの制限が最初に評価されます。グローバル制限は、モデルごとのクォータの状態に関係なく、すべてのモデルに適用されます。

設定

モデルごとのレート制限を適用する

これらの `RateLimit` CRDはAIモデルごとに適用され、`filters[].extensionRef` を介して `AIGatewayRoute` ルールから参照されます。

以下の例では、`x-org-id` および `x-user-id` リクエストヘッダーをキーとして、組織ごとおよびユーザーごとのレート制限を適用します。これらは単なる例であり、[ratelimit CRD](#)がサポートする任意の属性でレート制限をキー設定できます。

組織ごとのレート制限

組織ごとの制限は、`x-org-id` ヘッダーで識別される各組織が消費できる量を制限します。たとえば、このサンプル設定では、60秒間隔で組織ごとに2つのクォータを設定します。リクエスト数制限 (`req_threshold: 20`) とトークン制限 (`mode: TOKEN_RATE`, `req_threshold: 4000`) です。いずれかのクォータを超過したリクエストはブロックされます (`throttle_action: RESPOND`)。すべてのパラメータについては、[ratelimit CRD](#)を参照してください。

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1beta1

kind: ratelimit

metadata:

  name: orgaimodel

  namespace: default

spec:

  gatewayClassName: "aigwclass"

  selector_keys:

    basic:

      header_name: "X-org-id"

  ratelimits:
```

```

- req_threshold: 20

  alertsintimeslice: 40

  timeslice: 60000

  throttle_action: "RESPOND"

- mode: "TOKEN_RATE"

  req_threshold: 4000

  alertsintimeslice: 40

  timeslice: 60000

  throttle_action: "RESPOND"

```

EOF

ユーザーごとのレート制限

ユーザーごとの制限は、`X-user-id` ヘッダーで識別される各ユーザーが消費できる量を制限します。たとえば、このサンプル設定では、各ユーザーに1分あたり20リクエストと4000トークンを許可します。すべてのパラメータについては、[ratelimit CRD](#)を参照してください。

```

kubectl apply -f - <<EOF

apiVersion: citrix.com/v1beta1

kind: ratelimit

metadata:

  name: useraimodel

  namespace: default

spec:

  gatewayClassName: "aigwclass"

```

```
selector_keys:

  basic:

    header_name: "X-user-id"

ratelimits:

  - req_threshold: 20

    alertsintimeslice: 40

    timeslice: 60000

    throttle_action: "RESPOND"

  - mode: "TOKEN_RATE"

    req_threshold: 4000

    alertsintimeslice: 40

    timeslice: 60000

    throttle_action: "RESPOND"

EOF
```

可観測性のためのストリーム

ストリームはゲートウェイから消費メトリックを収集およびエクスポートするため、設定された制限に対してリクエストとトークンがどのように使用されているかを確認できます。

`ratelimit` 仕様に `streams` セクションを追加すると、同じ設定がユーザーごとおよび組織ごとのレート制限の両方に適用されます。

たとえば、このサンプル設定では、10分間隔 (`interval: 10` 、分単位) で2つのストリームを定義します。1つはリクエスト数による上位消費者 (`sort: REQUESTS`) を追跡し、もう1つはトークン数による上位消費者 (`sort: TOKENS`) を追跡します。各ストリームは、10分の `logInterval` で最大20エントリ (`logLimit: 20`) をログに記録します。 `logInterval` はストリームの `interval` 以上である必要があります。

これらのストリームが役立つようにするには、分析構成も適用する必要があります — [可観測性を参照してください](#)。

```
streams:

  - logInterval: 10

    logLimit: 20

    sort: "REQUESTS"

    interval: 10

  - logInterval: 10

    logLimit: 20

    sort: "TOKENS"

    interval: 10
```

レート制限フィルターでAIGatewayRouteを適用する

AIGatewayRoute は、以前に定義されたレート制限を、その `filters` セクションを介してモデルルートにバインドします。たとえば、このサンプルは、`orgaimodel` と `useraimodel` の両方のレート制限（上記で定義）を `ExtensionRef` フィルターとして参照しているため、両方のクォータが一致するモデルに適用されます。

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1

kind: aigatewayroute

metadata:

  name: ai-route-model

spec:

  parentRefs:
```

```
- name: aigw

  namespace: default

aigatewayprofiles:

  frontend:

    name: lbaigwyfp

    namespace: default

  default-backend:

    name: aigwybp

    namespace: default

rules:

  - filters:

    - type: ExtensionRef

      extensionRef:

        group: "citrix.com"

        kind: "ratelimit"

        name: "orgaimodel"

    - type: ExtensionRef

      extensionRef:

        group: "citrix.com"

        kind: "ratelimit"

        name: "useraimodel"

  matches:
```

```

      - policyExpression: "HTTP.REQ.URL.PATH.AFTER_STR(\"/openai/
        deployments/\").BEFORE_STR(\"/\") == \"gpt\""

      backendRefs:

        - name: gpt-model-svc

          port: 80

EOF

```

`AIGatewayRoute` 内の各ルールは、タイプ `ExtensionRef` の `filters` を介して `RateLimit` CRDを参照し、レート制限ポリシーをその特定のモデルルートにバインドします。

ゲートウェイレベルのレート制限を適用する

これらの `RateLimit` CRDは、`targetRef` を使用してゲートウェイを直接参照し、どのモデルが一致するかに関係なく、すべてのモデルルートにわたってクォータを適用します。

組織ごとのゲートウェイレベルのレート制限

```

kubectl apply -f - <<EOF

apiVersion: citrix.com/v1beta1

kind: ratelimit

metadata:

  name: orgcs

  namespace: default

spec:

  gatewayClassName: "aigwclass"

  targetRef:

    - name: aigw

      namespace: default

```

```
    kind: Gateway

    group: gateway.networking.k8s.io

selector_keys:

  basic:

    header_name: "X-org-id"

ratelimits:

  - req_threshold: 20

    alertsintimeslice: 40

    timeslice: 60000

    throttle_action: 'RESPOND'

  - mode: "TOKEN_RATE"

    req_threshold: 4000

    alertsintimeslice: 40

    timeslice: 60000

    throttle_action: 'RESPOND'

streams:

  - logInterval: 10

    logLimit: 20

    sort: "REQUESTS"

    interval: 10

  - logInterval: 10

    logLimit: 20
```

```
    sort: "TOKENS"

    interval: 10

EOF
```

ユーザーごとのゲートウェイレベルのレート制限

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1beta1

kind: ratelimit

metadata:

  name: usercs

  namespace: default

spec:

  gatewayClassName: "aigwclass"

  targetRef:

    - name: aigw

      namespace: default

      kind: Gateway

      group: gateway.networking.k8s.io

  selector_keys:

    basic:

      header_name: "X-user-id"

  ratelimits:
```

```
- req_threshold: 20

  alertsintimeslice: 40

  timeslice: 60000

  throttle_action: "RESPOND"

- mode: "TOKEN_RATE"

  req_threshold: 4000

  alertsintimeslice: 40

  timeslice: 60000

  throttle_action: "RESPOND"

streams:

- logInterval: 10

  logLimit: 20

  sort: "REQUESTS"

  interval: 10

- logInterval: 10

  logLimit: 20

  sort: "TOKENS"

  interval: 10
```

EOF

参照

[トークン消費量に基づくレート制限の構成](#)

プロンプト管理を設定する

NetScaler® AI Gatewayは、LLMバックエンドに到達する前に、受信プロンプトコンテンツを傍受および変更できます。`RewritePolicy` CRDは、JSON XPath式を使用してリクエストボディ内のユーザーメッセージを特定し、クライアントアプリケーションに対して透過的にコンテンツを追加または置換します。

ユースケース

- すべてのプロンプトにコンテキストメタデータを挿入する。
- 組織固有の指示をすべてのリクエストにグローバルに追加する。
- モデル推論の前に、プロンプトコンテンツにシステムレベルの制約を適用する。

プロンプト管理のためのRewritePolicy CRD

`RewritePolicy` はリクエストボディを傍受し、JSON XPath式を介してユーザーメッセージフィールドを特定し、バックエンドモデルに転送する前に設定されたコンテンツを追加または置換します。これをGatewayレベルでグローバルに適用することも、`AIGatewayRoute` を介して特定のAIモデルに適用することもできます。

Gatewayごとのリライトポリシーとモデルごとのリライトポリシー

リライトポリシーは、Gatewayレベルでグローバルに（すべてのモデルルートにわたって）適用することも、`AIGatewayRoute` を介して特定のAIモデルに適用することもできます。

タイプ	バインド方法	スコープ
モデルごと	<code>AIGatewayRoute</code> ルール内の <code>filters[].extensionRef</code>	モデルルートは1つのみ
ゲートウェイごと	<code>Gateway</code> リソースを指す <code>targetRef</code>	ゲートウェイ上のすべてのルート

注:

モデルごとおよびグローバルな書き換えポリシーは、同時にアクティブにすることができません。

設定

モデルごとの書き換えポリシー

この `RewritePolicy` CRDは特定のAIモデルに適用され、`filters[].extensionRef` を介して `AIGatewayRoute` ルールから参照されます。

例えば、このサンプル設定は、JSON XPath式 (`operation: insert_after`) を使用して見つめられたリクエストボディ内の最初のユーザーメッセージの後にテキスト `"I am trained till 10-28-2025"` を追加します。すべてのパラメータについては、[rewritepolicy CRD](#)を参照してください。

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: modelpol

spec:

  gatewayClassName: "aigwclass"

  rewrite-policies:

    - goto-priority-expression: "NEXT"

    rewrite-policy:

      operation: insert_after

      target: "HTTP.REQ.BODY(10000).XPATH_JSON_WITH_MARKUP(xp%/messages/Val[1]/content/text(%).UNQUOTE('\\"'))"

      modify-expression: '"I am trained till 10-28-2025"'

      direction: REQUEST

      rewrite-criteria: 'http.req.is_valid'
```

```
comment: 'Append training date info to user prompt'

EOF
```

書き換えポリシーフィルターを持つNetScaler AI Gatewayルート

`AIGatewayRoute` は、その `filters` セクションを通じて、モデルごとの書き換えポリシーをモデルルートにバインドし、それを `ExtensionRef` フィルターとして参照します。

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1

kind: aigatewayroute

metadata:

  name: ai-route-model

spec:

  parentRefs:

    - name: aigw

      namespace: default

  aigatewayprofiles:

    frontend:

      name: lbaigwyfp

      namespace: default

    default-backend:

      name: aigwybp

      namespace: default

  rules:
```

```
- matches:

  - policyExpression: "HTTP.REQ.URL.PATH.AFTER_STR(\"/openai/
    deployments/\").BEFORE_STR(\"/\") == \"gpt\""

  filters:

    - type: ExtensionRef

      extensionRef:

        group: "citrix.com"

        kind: "rewritepolicy"

        name: "modelpol"

      backendRefs:

        - name: gpt-model-svc

          port: 80

EOF
```

Kubernetes Gatewayでの書き換えポリシー

この `RewritePolicy` CRDは、`targetRef` を使用してゲートウェイを直接参照し、どのモデルが一致するかに関わらず、すべてのモデルルートに適用されます。

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: csvspol

spec:
```

```
gatewayClassName: "aigwclass"

targetRef:

  - name: aigw

    namespace: default

    kind: Gateway

    group: gateway.networking.k8s.io

rewrite-policies:

  - goto-priority-expression: "NEXT"

    rewrite-policy:

      operation: insert_after

      target: "HTTP.REQ.BODY(10000).XPATH_JSON_WITH_MARKUP(xp//
messages/Val[1]/content/text(%)).UNQUOTE('\\"')\"

      modify-expression: '"I am trained till 10-28-2025"'

      direction: REQUEST

      rewrite-criteria: 'http.req.is_valid'

      comment: 'Append training date info to user prompt'

EOF
```

参照

[プロンプト管理](#)

代替モデルへのスπιルオーバーを設定する

プライマリモデルのトークンクォータが使い果たされると、NetScaler® AI Gateway は、クライアント側の変更なしにサービス継続性を維持しながら、設定されたバックアップモデルにリクエストを自動的にルーティングします。

警告:

スピルオーバーは、設定したバックアップモデルにオーバーフローしたトラフィックを転送します。そのバックアップが有料の外部プロバイダー（Azure OpenAI、OpenAI、Anthropic、または Google Vertex AI）である場合、プライマリモデルのトークンクォータを超えるすべてのリクエストは、そのプロバイダーによって課金されます。予期せぬ料金を避けるために、プライマリ `tokenquota` のサイズを適切に設定し、スピルオーバーバックエンドを慎重に選択してください。

仕組み

`AIGatewayRoute` の各ルールには、プライマリ `backendRef` と `is_spillover: true` でマークされたスピルオーバー `backendRef` がリストされています。スピルオーバーバックエンドは、独立したクォータ追跡のために独自の `backend-aigatewayprofile` を使用します。

`RewritePolicy` は URL パス内のモデル名を書き換え、バックアップバックエンドが期待する正しいモデル識別子を受け取るようにします。

主要な設定要素

要素	目的
<code>backendRef</code> の <code>is_spillover: true</code>	バックエンドをスピルオーバーとしてマークします。この要素は、プライマリクォータが使い果たされた場合にのみ使用されます。
スピルオーバーバックエンドの <code>backend-aigatewayprofile</code>	スピルオーバーバックエンドに独自の独立したクォータ追跡を与える個別のバックエンドプロファイル。指定されていない場合、 <code>spec.aigatewayprofiles</code> の下の <code>default-backend</code> プロファイルが使用されます。
<code>spec.aigatewayprofiles</code> の下の <code>spillover-frontend</code>	フロントエンドプロファイルは、スピルオーバーバックエンドに常に使用されます。
<code>RewritePolicy</code> から <code>ExtensionRef</code> までのフィルター	URL 内のモデル名を、バックアップサービスが期待するものと一致するように書き換えます。

設定

スピルオーバーバックエンドプロファイル

スピルオーバーモデル用のバックエンド AI Gateway プロファイルを作成し、独自の独立したクォータ追跡ができるようにします。ルートはこのプロファイルを `aigwybp-spill` として参照します。

スピルオーバー AI Gateway エンドポイントの認証トークンまたは API キーを含む Kubernetes Secret を作成します。 `<auth-token>` をスピルオーバーエンドポイントの認証トークンに置き換えます。

```
kubectl create secret generic aigw-auth-spill --from-literal=authToken=<auth-token>
```

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1

kind: aigatewayprofile

metadata:

  name: aigwybp-spill

spec:

  gatewayClassName: "aigwclass"

  config:

    authtoken:

      name: aigw-auth-spill

      namespace: default

    endpointtype: "azureopenai"

    profiletype: "backend"

    tokenquota: 25000
```

```
quotarefreshfrequency: 1
```

```
EOF
```

スピルオーバー書き換えポリシー

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: spilloverrewrite

spec:

  gatewayClassName: "aigwclass"

  rewrite-policies:

    - goto-priority-expression: "NEXT"

    rewrite-policy:

      operation: replace

      target: 'http.req.url.path.get(3)'

      modify-expression: '"gpt2.5"'

      comment: 'Replace model name in URL path for spillover backend'

      direction: REQUEST

      rewrite-criteria: 'http.req.is_valid'

EOF
```

この `RewritePolicy` は、URL パス (3 番目のセグメント) 内のモデル名をバックアップモデル名に置き換えます。たとえば、元々 `/openai/deployments/gpt/chat/completions` をターゲットとするリクエストは、バックアップサービスに転送される前に `/openai/deployments/gpt2.5/chat/completions` に書き換えられます。

スピルオーバーサービスを使用する AI Gateway ルート

`AIGatewayRoute` は、モデルルールごとにプライマリバックエンドとスピルオーバーバックエンドを定義します。スピルオーバーバックエンド (`is_spillover: true`) は、`filters[].extensionRef` を介してスピルオーバー書き換えポリシーをアタッチします。

スピルオーバーバックエンドの `backend-aigatewayprofile` は、スピルオーバーモデルが独自の独立したクォータ追跡を取得できるように、個別のバックエンドプロファイルを定義します。

`backend-aigatewayprofile` が指定されていない場合、スピルオーバーバックエンドは `spec.aigatewayprofiles` で定義された `default-backend` プロファイルにフォールバックします。

スピルオーバーバックエンドのフロントエンドプロファイルは、常に `spec.aigatewayprofiles.spillover-frontend` から取得されます。

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1

kind: aigatewayroute

metadata:

  name: ai-route-model

spec:

  parentRefs:

    - name: aigw

      namespace: default

  aigatewayprofiles:

    frontend:
```

```
    name: lbaigwyfp

    namespace: default

  default-backend:

    name: aigwybp

    namespace: default

  spillover-frontend:

    name: lbaigwyfp

    namespace: default

  rules:

    - matches:

        - policyExpression: "HTTP.REQ.URL.PATH.AFTER_STR(\"/openai/
deployments/\").BEFORE_STR(\"/\") == \"gpt\""

    backendRefs:

      - name: gpt-model-svc

        port: 80

      - name: gpt-2-5-model-svc

        port: 80

      is_spillover: true

      backend-aigatewayprofile:

        name: aigwybp-spill

        namespace: default

      filters:
```

```
- type: ExtensionRef

  extensionRef:

    group: "citrix.com"

    kind: "rewritepolicy"

    name: "spilloverrewrite"
```

EOF

参照

[スピルオーバールートリクエストを別の言語モデルに構成する](#)

可観測性

NetScaler® AI Gateway は、可観測性が設定されると、トークン数、モデルID、クライアントID、レイテンシー、コスト配分など、リクエストごとのAIメトリクスをSplunkにエクスポートします。アプリケーションの計測は不要です。

NetScaler は、Splunk HTTP Event Collector (HEC) を介してこれらのメトリクスをSplunkにエクスポートします。HEC接続と分析プロファイルは、このページで説明されているように設定します。

前提条件

- NetScaler Kubernetes Gateway Controller がすでにデプロイされていること。「[AI機能を持つ Kubernetes Gateway Controllerのデプロイ](#)」を参照してください。
- HTTP Event Collector (HEC) が有効になっている、アクセス可能なSplunkインスタンス、およびHECトークンとターゲットインデックス。
- クラスターとNetScalerからアクセス可能な、Splunk HECエンドポイントのホストとポート (HECのデフォルトポートは `8088` です)。
- `kubectl` と Helm 3.x があり、ターゲットネームスペースにシークレットとConfigMapを作成する権限があること。

注:

SplunkトークンシークレットとNetScaler CPXカスタムメトリクス用のConfigMapを、コントローラーまたはCPXと同じネームスペースに作成し、`authToken.namespace` をそのネームスペースに設定します。

設定の概要

可観測性は2つの部分で設定されます。

項目	場所	必須
Splunk サーバー接続	Helm <code>values.yaml</code> の <code>analyticsConfig</code>	必須

項目	場所	必須
ルートごとの分析プロファイル	<code>AIGatewayRoute</code> の <code>aigatewayroute.citrix.com/analytics-profile</code> アノテーション	オプション — ルートごとにグローバル設定を上書きします

グローバルプロファイルとルートレベルプロファイル

プロファイルタイプ	設定場所	スコープ
時系列メトリクス	<code>values.yaml</code> の <code>analyticsConfig.splunk.servers[].profiles.timeSeries</code>	グローバルのみ — ルートごとにはできません
ストリームインサイト	<code>values.yaml</code> の <code>analyticsConfig.splunk.servers[].profiles.streaminsight</code>	グローバルデフォルト
ルートごとのオーバーライド	<code>aigatewayroute.citrix.com/analytics-profile</code> アノテーション	<code>AIGatewayRoute</code> ごと

Splunkトークンシークレットを作成する

```
kubectl create secret generic splunktoken \
  --from-literal=splunk-token=<your-splunk-token> -n <namespace>
```

チャートはこのシークレットの `splunk-token` キーからトークンを読み取ります。Splunk HEC トークンを値として使用してください。 `analyticsConfig`（下記）の `authToken` 参照は、`name` と `namespace` によってこのシークレットを指します。

Helm値でSplunk接続を構成する（必須）

Helmチャートの `values.yaml` にある `analyticsConfig` セクションは、Splunkサーバー接続とデフォルトのアナリティクスプロファイルを作成します。

```
analyticsConfig:

  required: true

  logAllJsonFields: "true"

  splunk:

    enable: "true"

    servers:

      - name: "my-splunk"

        endpoint: "<splunk-endpoint>"

        port: 8088    # Splunk HEC port (default 8088); match your HEC
listener
        authToken:

          name: splunktoken

          namespace: <namespace>

        profiles:

          timeseries:

            metricsexportfrequency: 60    # seconds

            analyticsendpointurl: "netscaler/metric"

          streaminsight:

            analyticsendpointurl: "/services/collector/event"

            analyticsendpointcontenttype: "application/json"
```

注：

`logAllJsonFields: "true"` を設定すると、トランザクションタイプのメトリクスに関するすべてのカウンターが有効になり、AI Gateway固有の詳細がエクスポートデータに含まれるようになります。ただし、すべてのトランザクションでJSONフィールドがすべてエクスポートされるため、Splunkの取り込みとライセンス費用が大幅に増加する可能性があります。完全なAIメトリクスが必要な場合にのみ有効にし、それ以外の場合は無効のままにしてください。

`values.yaml` を編集した後、デプロイメントを更新するために適用します。

```
helm upgrade <release> <chart> -f values.yaml
```

あるいは、`-f values.yaml` を最初の `helm install` コマンドに追加します。

ルートごとにアナリティクスプロファイルをオーバーライドする

特定のモデルルートのコレクターとインサイトタイプをカスタマイズするには、

`AIGatewayRoute` に `aigatewayroute.citrix.com/analytics-profile` アノテーションを使用します。この完全な例に示すように、`AIGatewayRoute` メタデータにアノテーションを追加します。

```
apiVersion: citrix.com/v1

kind: aigatewayroute

metadata:

  name: ai-route-model

  annotations:

    aigatewayroute.citrix.com/analytics-profile: |

      [

        {
```

```
      "collectors": {"splunk": ["my-splunk"]},

      "type": "webinsight",

      "analyticsEndpointUrl": "/services/collector/event",

      "allHttpHeaders": "enabled"

    },

    {

      "collectors": {"splunk": ["my-splunk"]},

      "type": "streaminsight",

      "analyticsEndpointUrl": "/services/collector/event",

      "analyticsEndpointContentType": "application/json",

      "topn": "enabled"

    }

  ]

spec:

  parentRefs:

    - name: aigw

  aigatewayprofiles:

    frontend:

      name: lbaigwyfp

    default-backend:

      name: aigwybp

  rules:
```

```

- matches:

  - path:

    type: PathPrefix

    value: "/"

  backendRefs:

    - name: gpt-model-svc

      port: 80

```

重要：

アノテーションの `collectors` フィールドのサーバー名（例: `"my-splunk"`）は、`values.yaml` の `analyticsConfig` で定義されているサーバー `name` と一致する必要があります。 `values.yaml` の `analyticsConfig` セクションはサーバー接続を作成し、アノテーションは名前ですそれを参照します。

カスタムAIゲートウェイメトリクス

AIゲートウェイ固有のメトリクスカウンターをエクスポートするには、`metrics.json` という名前のカスタムメトリクス構成ファイルを作成します。 [AI Gateway observability](#) に記載されているAIゲートウェイカウンターを使用して、このファイルを構築します。このファイルをNetScalerに追加する方法は、フォームファクター（クラスター内で実行されるコンテナ化されたNetScalerであるNetScaler CPX、またはクラスター外で実行される仮想アプライアンスであるNetScaler VPX/MPX/BLX）によって異なります。

NetScaler CPX

NetScaler CPX（コンテナ化されたNetScaler）の場合、ConfigMapを使用して `metrics.json` ファイルをCPXコンテナに追加します。 `metrics.json` ファイルは `https://raw.githubusercontent.com/netScaler/netScaler-k8s-ingress-controller/refs/heads/master/example/metrics.json` で利用できます。

`metrics.json` ファイルからConfigMapを作成します。

```
kubectl create configmap my-custom-metrics-configmap \
  --from-file=metrics.json -n <namespace>
```

`values.yaml` で `extraVolumeMounts` と `extraVolumes` を設定して、ConfigMapをCPXコンテナにマウントします。

```
extraVolumeMounts:
  - name: custom-metrics-conf

  mountPath: /cpx/custom_metrics_conf

extraVolumes:
  - name: custom-metrics-conf

    configMap:

      name: "my-custom-metrics-configmap"

      defaultMode: 420
```

`values.yaml` を編集した後、それを適用してデプロイメントを更新します。

```
helm upgrade <release> <chart> -f values.yaml
```

NetScaler VPX/MPX/BLX

NetScaler VPX/MPX/BLX（クラスター外で実行される仮想アプライアンス）の場合、[AI Gateway observability](#)で説明されているように、`metrics.json` ファイルをNetScaler VPX/MPX/BLX内に直接配置します。`metrics.json` ファイルは <https://raw.githubusercontent.com/netScaler/netScaler-k8s-ingress-controller/refs/heads/master/example/metrics.json> で利用できます。

確認

設定を適用した後、メトリクスがSplunkに到達していることを確認します。

- Splunkで、設定されたインデックスまたはエンドポイント（例えば、`netScaler/metric` を含むソース）を検索します。
- 1つの `metricexportfrequency` 間隔（例では60秒）以内に、トークン数とモデルIDを含むリクエストごとのAIイベントが表示されるはずです。

リファレンス

[AI Gateway の可観測性](#)

NetScaler CRD

NetScaler CRDで利用可能なトピックは以下のとおりです。

- [AI Gateway プロファイル](#)
- [AI Gateway ルート](#)
- [レート制限](#)
- [リライトおよびレスポンスポリシー](#)

AI Gateway プロファイル

`aigatewayprofile` カスタムリソース定義 (CRD) を使用すると、Kubernetesネイティブ構成を使用してNetScaler®上にAI Gatewayプロファイルを定義できます。AI Gatewayプロファイルには、NetScalerがアプリケーションと大規模言語モデル (LLM) バックエンド間のAIトラフィックを処理する際に適用する設定が含まれています。

`aigatewayprofile` CRD を使用して、次のことができます。

- 推論リクエストを受信するクライアント向けロードバランシング仮想サーバーにフロントエンドプロファイルを適用します。
- アップストリームのLLMプロバイダーに接続するサービスグループにバックエンドプロファイルを適用し、モデルごとのトークンクォータとクォータ更新頻度を設定します。
- Kubernetes Secret からAIエンドポイントの認証トークン (APIキー) を参照します。
- NetScalerにすでに存在する、事前設定された組み込みのAI Gatewayプロファイルを再利用します。

プロファイルは `GatewayClass` にアタッチされ、その `aigatewayprofiles` フィールドを介して `aigatewayroute` によって使用されます。

注:

`preconfigured` フィールドと `config` フィールドは相互に排他的です。いずれか一方を指定する必要があります。

aigatewayprofile CRD を展開する

[aigatewayprofile CRD](#) をダウンロードし、次のコマンドを使用して展開します。

```
kubect1 create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/crd/aigateway/aigatewayprofile-crd.yaml
```

注:

CRD展開YAMLファイルを変更しないでください。

AIゲートウェイプロファイル CRD 属性

次の表は、`aigatewayprofile` CRD の `spec` で利用可能な属性を示しています。

属性	説明	サポートされている値
<code>gatewayClassName</code>	この AI Gateway プロファイルが適用される <code>GatewayClass</code> の名前。このプロファイルは、この値がコントローラーによって管理される <code>GatewayClass</code> と一致する場合にのみ適用されます。このフィールドは必須です。	文字列
<code>preconfigured</code>	使用する NetScaler 上の事前設定された組み込み AI Gateway プロファイルの名前。 <code>config</code> とは相互に排他的です。	文字列
<code>config</code>	完全にカスタムな AI Gateway プロファイルのための NetScaler に関連するすべてのパラメーターを含みます。 <code>preconfigured</code> とは相互に排他的です。	オブジェクト

config 属性

カスタムプロファイルを定義する場合、`spec.config` の下に以下の属性が利用可能です。

属性	説明	サポートされている値
<code>profiletype</code>	AI Gatewayプロファイルのタイプ。この値は設定後に変更できません。	<code>backend</code> , <code>frontend</code>
<code>endpointtype</code>	AIエンドポイントのタイプ。この値は設定後に変更できません。	<code>azureopenai</code>
<code>tokenquota</code>	バックエンドプロファイルのトークンクォータ制限。 <code>profiletype</code> が <code>backend</code> の場合にのみ適用されます。	整数 (0–2147483647)
<code>quotarefreshfrequency</code>	クォータ更新レート（分単位）。 <code>profiletype</code> が <code>backend</code> の場合にのみ適用されます。	整数 (1–65535)
<code>authtoken</code>	AI Gatewayエンドポイント用の認証トークンまたはAPIキーで、Kubernetes Secretから取得されます。この値は設定後に変更できません。 <code>profiletype</code> が <code>backend</code> の場合にのみ適用されます。	オブジェクト

authtoken属性

属性	説明	サポートされる値
<code>name</code>	認証トークンを含むKubernetes Secretオブジェクトの名前。このフィールドは必須です。	<code>\^a-z0-9?\$</code> に一致する文字列
<code>namespace</code>	Kubernetes Secretオブジェクトの名前空間。デフォルト	<code>\^a-z0-9?\$</code> に一致する文字列

属性	説明	サポートされる値
	はAI Gatewayプロファイルの名前空間です。	

プロファイル構成の記述方法

`aigatewayprofile` YAML定義で、`kind` を `aigatewayprofile` として設定します。`spec` セクションで、`gatewayClassName` と、`config` ブロック（カスタムプロファイルの場合）または `preconfigured` 値（組み込みプロファイルを再利用する場合）のいずれかを追加します。

以下のガイドラインに留意してください。

- `config` または `preconfigured` のいずれか一方のみを指定してください。
- `tokenquota` は、`profiletype` が `backend` であるプロファイルでのみ設定してください。
- プロファイル作成後、`profiletype`、`endpointtype`、および `authtoken` の値は変更できません。
- APIキーはKubernetes Secretに保存し、`authtoken` を介して参照してください。

プロファイル構成の例

フロントエンドプロファイル

次の構成は、クライアント向けのAI Gatewayプロファイルを作成します。

```
apiVersion: citrix.com/v1

kind: aigatewayprofile

metadata:

  name: frontend-profile

  namespace: default

spec:

  gatewayClassName: aigwclass

  config:

    profiletype: frontend
```

```
endpointtype: azureopenai
```

トークンクォータ付きのバックエンドプロファイル

次の構成は、60分ごとに更新されるトークンクォータを適用するバックエンドAI Gatewayプロファイルを作成します。

```
apiVersion: citrix.com/v1

kind: aigatewayprofile

metadata:

  name: backend-profile

  namespace: default

spec:

  gatewayClassName: aigwclass

  config:

    profiletype: backend

    endpointtype: azureopenai

    tokenquota: 1000000

    quotarefreshfrequency: 60

    authtoken:

      name: azure-openai-key

      namespace: default
```

事前構成済みプロファイル

次の構成は、NetScalerにすでに存在する組み込みのAI Gatewayプロファイルを再利用します。

```
apiVersion: citrix.com/v1

kind: aigatewayprofile

metadata:

  name: builtin-profile

  namespace: default

spec:

  gatewayClassName: aigwclass

  preconfigured: default-ai-frontend
```

関連資料

- [AI Gatewayの概要](#)
- [モデルルーティングの構成](#)
- [AI Gateway ルート](#)
- [NetScalerでのAI機能](#)

AI Gateway ルート

`aigatewayroute` カスタムリソース定義 (CRD) は、複数の AI サービスバックエンドを結合し、NetScaler® Kubernetes Gateway Controller によって管理される `Gateway` リソースにアタッチします。これにより、`Gateway` のための 統合された AI API を定義する方法が提供され、ダウンストリームクライアントは単一のスキーマを使用して、プロバイダーをまたがる複数の AI バックエンドと対話できます。

NetScaler Kubernetes Gateway Controller は、`aigatewayroute` に対して以下の機能を提供します。

- Gateway API の `HTTPRoute` を介した、ヘッダーベースおよびポリシーベースの AI バックエンドへのルーティング。生成された `HTTPRoute` は `aigatewayroute` と同じ名前を持ちます。
- プライマリ AI バックエンドが利用できないか過負荷状態の場合に自動フェイルオーバーを可能にする、優先度ベースのモデルスピルオーバーをバックアップバックエンドに適用します。

- マルチプロバイダー展開のためのフロントエンドとバックエンドの AI Gateway プロファイルを分離し、クライアント向けおよびアップストリームの AI サービスポリシーを個別に構成できるようにします。
- きめ細かなトラフィック制御とリクエストまたはレスポンスの変換のための `ratelimit` および `rewritepolicy` CRD のサポート。
- AI バックエンド接続のプロトコルレベルチューニングのための `sslprofile`、`tcpprofile`、および `httpprofile` CRD のサポート。
- AI トラフィックパターン、トークン使用量、およびバックエンドパフォーマンスの可観測性のための、組み込みの TCP、HTTP、およびストリーミング分析。

これらのリソースはすべて、`aigatewayroute` と同じ名前空間に作成されます。

`ExtensionRef` フィルターを介して生成された `HTTPRoute` に NetScaler ポリシー CRD をアタッチすることで、デフォルトの動作をカスタマイズできます。

注:

マッチング条件では、モデル名に基づいたルーティング決定のために `x-ai-eg-model` ヘッダーが利用可能です。モデル名はルーティング前にリクエストコンテンツから抽出されます。

aigatewayroute CRD を展開する

`aigatewayroute` CRD をダウンロードし、以下のコマンドを使用して展開します。

```
kubectl create -f https://raw.githubusercontent.com/citrix/citrix-k8s-ingress-controller/master/crd/aigateway/aigatewayroute-crd.yaml
```

注:

CRDデプロイメントYAMLファイルを変更しないでください。

aigatewayroute カスタムリソース定義の属性

次の表は、`aigatewayroute` CRDの `spec` で利用可能なトップレベルの属性を一覧表示しています。

属性	説明	サポートされている値
<code>rules</code>	このルートがトラフィックを照合するルーティングルール のリスト。このフィールドは 必須です。	ルールの配列（最大128）
<code>aigatewayprofiles</code>	生成されたロードバランシング 仮想サーバーおよびサービス グループに適用するデフォ ルトのAI Gatewayプロファイル。	オブジェクト
<code>hostnames</code>	このルートを選択するために HTTP <code>Host</code> ヘッダーと照合 されるホスト名のセット。ワ イルドカードプレフィックス (<code>.*</code>) はサフィックスマッチ として解釈されます。IPアド レスは許可されていません。	RFC 1123ホスト名の配列 (最大16)
<code>parentRefs</code>	このルートがアタッチされて いるリソース（通常は <code>Gateway</code> オブジェクト）へ の参照。	配列（最大32）

エーアイゲートウェイプロファイル属性

以下の各プロファイル参照は、`name`（必須）と、`aigatewayroute` 名前空間にデフォルト設定
されるオプションの `namespace` を指定します。

属性	説明
<code>frontend</code>	プライマリロードバランシング仮想サーバー に適用されるAI Gatewayプロファイル。
<code>spillover-frontend</code>	スピルオーバーロードバランシング仮想サー バーに適用されるAI Gatewayプロファイル。
<code>default-backend</code>	バックエンド参照ごとの <code>backend-</code> <code>aigatewayprofile</code> によって上書きされない

属性	説明
	限り、すべてのバックエンドサービスグループに適用されるAI Gatewayプロファイル。

parentRefs 属性

次の表に `parentRefs` 属性を示します。

属性	説明	サポートされている値
<code>name</code>	参照先の名前。このフィールドは必須です。	文字列
<code>group</code>	参照先のグループ。指定されていない場合、 <code>gateway.networking.k8s.io</code> が推測されます。	文字列
<code>kind</code>	参照先の種類。	デフォルト: <code>Gateway</code>
<code>namespace</code>	参照先の名前空間。デフォルトはルートローカル名前空間です。	文字列
<code>port</code>	このルートがターゲットとするネットワークポート。	整数 (1–65535)
<code>sectionName</code>	ターゲットリソース内のセクション名 (<code>Gateway</code> のリセクサー名)。	文字列

ルール属性

`rules` の各エントリは、以下の属性をサポートします。

属性	説明	サポートされる値
<code>backendRefs</code>	このルールがトラフィックをルーティングするAIサービスバックエンドのリスト。	配列 (最大128)

属性	説明	サポートされる値
<code>matches</code>	このルールがいつ適用されるかを決定する条件のリスト。パスベース、ヘッダーベース、およびポリシー式の一致をサポートします。	配列 (最大128)
<code>filters</code>	このルールに <code>ExtensionRef</code> を介してアタッチするNetScalerポリシーCRD。	配列 (最大16)
<code>timeouts</code>	HTTPリクエストに対して設定できるタイムアウト。	オブジェクト

backendRefs属性

次の表に `backendRefs` 属性を示します。

属性	説明	サポートされている値
<code>name</code>	バックエンド <code>Service</code> リソースの名前。このフィールドは必須です。	文字列
<code>group</code>	参照先のAPIグループ。空の場合、コアAPIグループが推論されます。	文字列
<code>kind</code>	参照先のKubernetesリソースの種類。	デフォルト: <code>Service</code>
<code>namespace</code>	バックエンドリソースの名前空間。デフォルトはルートのローカル名前空間です。	文字列
<code>port</code>	このバックエンドに使用する宛先ポート番号。	整数 (1–65535)
<code>weight</code>	同じルール内の他のバックエンドと比較して、このバック	整数 (デフォルト 1)

属性	説明	サポートされている値
	エンドに送信するトラフィックの割合。	
<code>is_spillover</code>	このバックエンドが、プライマリバックエンドが利用できないか過負荷の場合にのみトラフィックを受信するスピルオーバーバックエンドであるかどうかを示します。1つの <code>backendRef</code> のみがこれを <code>true</code> に設定できます。	ブール値 (デフォルト <code>false</code>)
<code>backend-aigatewayprofile</code>	このバックエンドのサービスグループに適用するバックエンドAI Gatewayプロファイル。スペックレベルの <code>default-backend</code> プロファイルを上書きします。	オブジェクト (<code>name</code> , <code>namespace</code>)
<code>filters</code>	<code>ExtensionRef</code> を介してアタッチする、バックエンドごとのNetScalerポリシーCRD。	配列 (最大 16)

一致属性

次の表に一致属性を示します。

属性	説明	サポートされる値
<code>path</code>	HTTPリクエストパスのマッチャー。デフォルトは <code>/</code> に対する <code>PathPrefix</code> マッチです。	<code>type</code> : <code>Exact</code> , <code>PathPrefix</code> , <code>RegularExpression</code> ; <code>value</code> : 文字列
<code>headers</code>	HTTPリクエストヘッダーマッチャー。複数の値にはANDを使用します。	<code>name</code> 、 <code>value</code> 、 <code>type</code> : <code>Exact</code> 、 <code>RegularExpression</code>
<code>policyExpression</code>	高度なマッチングのためのNetScalerポリシー式。ヘッ	文字列

属性	説明	サポートされる値
	ダーマッチに加えて評価されます。	

フィルター属性

次の表にフィルター属性を示します。

属性	説明	サポートされている値
<code>type</code>	フィルターの種類。現在、 <code>ExtensionRef</code> のみがサポートされています。	<code>ExtensionRef</code>
<code>extensionRef</code>	NetScalerポリシーCRDへの参照 (<code>group</code> 、 <code>kind</code> 、 <code>name</code> — すべて必須)。サポートされている種類には、 <code>ratelimit</code> 、 <code>rewritepolicy</code> 、 <code>aigatewayprofile</code> 、 <code>sslprofile</code> 、 <code>tcpprofile</code> 、および <code>httpprofile</code> が含まれます。	オブジェクト

タイムアウト属性

次の表に、タイムアウト属性を示します。

属性	説明	サポートされている値
<code>request</code>	ゲートウェイがHTTPリクエストに応答する最大期間。	期間。例: <code>30s</code>
<code>backendRequest</code>	ゲートウェイからバックエンドへの個々のリクエストのタイムアウト。 <code>request</code> より長くすることはできません。	期間、例えば <code>20s</code>

注:

ストリーミング応答（`stream=true` を使用したチャット補完など）の場合、応答の完了に時間がかかる可能性があるため、タイムアウトを長く設定してください。

ルート設定の記述方法

`aigatewayroute` YAML定義で、`kind` を `aigatewayroute` に設定します。`spec` セクションで、ルートを `Gateway` にアタッチする `parentRefs`、フロントエンドとバックエンドのプロファイルを選択する `aigatewayprofiles`、およびトラフィックを照合してルーティングする1つ以上の `rules` を追加します。

以下のガイドラインに留意してください。

- `rules` フィールドは必須です。
- 優先度の低い値を持つバックエンドが優先されます。ルール内の最大1つのバックエンドに `is_spillover: true` を設定して、フェイルオーバーバックエンドを指定します。
- モデル名でルーティングするには、`x-ai-eg-model` ヘッダーまたは `matches` 内の `policyExpression` を使用します。
- ポリシーCRD（`ratelimit`、`rewritepolicy` など）をルールレベルまたはバックエンドごとに `ExtensionRef` フィルターを介してアタッチします。

ルート設定の例

複数のバックエンドへのモデルルーティング

次の構成は、`x-ai-eg-model` ヘッダーに含まれるモデル名に基づいて、リクエストを異なるバックエンドにルーティングします。

```
apiVersion: citrix.com/v1

kind: aigatewayroute

metadata:

  name: ai-model-route

  namespace: default

spec:
```

```
parentRefs:

- name: ai-gateway

  sectionName: https

aigatewayprofiles:

  frontend:

    name: frontend-profile

  default-backend:

    name: backend-profile

rules:

- matches:

  - headers:

    - name: x-ai-eg-model

      value: gpt-5

  backendRefs:

    - name: gpt-5-backend

      port: 80

- matches:

  - headers:

    - name: x-ai-eg-model

      value: gpt-5-codex

  backendRefs:

    - name: gpt-5-codex-backend
```

```
port: 80
```

優先度ベースのモデルスピルオーバー

次の構成は、プライマリバックエンドにトラフィックを送信し、プライマリが利用できないか過負荷の場合にバックアップバックエンドにスピルオーバーします。

```
apiVersion: citrix.com/v1

kind: aigatewayroute

metadata:

  name: ai-spillover-route

  namespace: default

spec:

  parentRefs:

    - name: ai-gateway

      sectionName: https

  aigatewayprofiles:

    frontend:

      name: frontend-profile

    spillover-frontend:

      name: spillover-frontend-profile

    default-backend:

      name: backend-profile

  rules:

    - matches:
```

```
- path:

  type: PathPrefix

  value: /v1/chat/completions

backendRefs:

- name: primary-backend

  port: 80

  weight: 1

- name: backup-backend

  port: 80

  is_spillover: true

  backend-aigatewayprofile:

    name: backup-backend-profile

timeouts:

  request: 120s

  backendRequest: 110s
```

ルールにレート制限ポリシーをアタッチする

次の構成は、`ExtensionRef` フィルターを介して、`ratelimit` CRD をルーティングルールにアタッチします。

```
apiVersion: citrix.com/v1

kind: aigatewayroute

metadata:

  name: ai-ratelimited-route
```

```
namespace: default

spec:

  parentRefs:

    - name: ai-gateway

      sectionName: https

  rules:

    - matches:

      - path:

          type: PathPrefix

          value: /v1/chat/completions

      filters:

        - type: ExtensionRef

          extensionRef:

            group: citrix.com

            kind: ratelimit

            name: token-ratelimit

          backendRefs:

            - name: llama-backend

              port: 80
```

関連資料

- [AI Gateway の概要](#)
- [モデルルーティングの構成](#)

- 代替モデルへのスピンオーバーの構成
- AI Gateway プロファイル
- レート制限
- リライトおよびレスポンスポリシー

レート制限

`ratelimit` カスタムリソース定義 (CRD) を使用すると、Kubernetesネイティブ構成を使用して NetScaler®上でレート制限およびスロットリングポリシーを設定できます。`ratelimit` CRDを使用すると、リクエスト、接続、またはトークンのレートを制限することで、サービスとAIバックエンドを過負荷から保護できます。

`ratelimit` CRDを使用して、次のことができます。

- リクエストレート、接続数、またはトークンレートに基づいてトラフィックをスロットルします。
- クライアントIPアドレスごと、APIパスごと、HTTPメソッドごと、またはHTTPヘッダーによって識別されるAPIクライアントごとに制限を適用します。
- 制限を超過した場合に実行するアクション（リクエストのドロップ、リセット、リダイレクト、応答など）を選択します。
- ストリーム識別子を使用してトラフィック分析を収集し、リクエスト、接続、応答時間、帯域幅、およびトークン使用量の可観測性を実現します。

`ratelimit` CRDは、`ExtensionRef` フィルターを介して `aigatewayroute` から参照されるか、`servicenames` または `targetRef` を介してサービスに直接適用されます。

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。

ratelimit CRDを展開する

`ratelimit` CRD をダウンロードし、次のコマンドを使用して展開します。

```
kubect1 create -f https://raw.githubusercontent.com/netScaler/netScaler-k8s-ingress-controller/refs/heads/master/crd/ratelimit/ratelimit-crd.yaml
```

注:

CRDデプロイメントYAMLファイルを変更しないでください。

レートリミット CRD 属性

注記:

NetScaler Kubernetes Gateway Controller 2.2.0以降のリリースでは、`ratelimit` CRDスキーマの属性は`ratelimits` spec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ています。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、`ratelimit` CRDの`spec`の下で利用可能なトップレベルの属性を示します。

属性	説明	サポートされる値
<code>ingressclass</code>	インGRESSクラス。指定されていない場合、クラスター内のすべてのNetScaler Ingress Controllerがリソースを処理します。それ以外の場合、そのインGRESSクラスを持つコントローラーのみが処理します。	文字列
<code>servicenames</code>	レート制限ポリシーが適用されるサービスの名称。	文字列の配列（最大長127）
<code>gatewayClassName</code>	このプロファイルが適用される <code>GatewayClass</code> の名称。	文字列
<code>targetRef</code>	このプロファイルが適用されるターゲットリソースのリスト。	配列
<code>selector_keys</code>	レート制限またはスロットリングが適用されるトラフィック一致条件。	オブジェクト

属性	説明	サポートされる値
<code>ratelimits</code>	レート制限設定のリスト。	配列
<code>streams</code>	トラフィック分析および追跡のためのストリーム識別子設定のリスト。	配列

targetRef 属性

属性	説明
<code>name</code>	ターゲットリソースの名前。
<code>namespace</code>	ターゲットリソースの名前空間。
<code>group</code>	ターゲットリソースのグループ。
<code>kind</code>	ターゲットリソースの種類。
<code>sectionName</code>	ターゲットリソース内の特定のセクション。

selector_keys 属性

`selector_keys.basic` オブジェクトは、トラフィックストリームの選択基準を定義します。すべてのキーはAND条件として適用されます。キーが指定されていない場合、レート制限はサービスレベルで適用されます。

属性	説明	サポートされている値
<code>path</code>	APIリソースパスのプレフィックス一致。例えば <code>/api/v1/products</code> 。	文字列の配列
<code>method</code>	一致させるHTTPメソッド。	<code>GET</code> , <code>PUT</code> , <code>POST</code> , <code>DELETE</code> , <code>HEAD</code> , <code>OPTIONS</code> , <code>TRACE</code> , <code>CONNECT</code> , <code>PATCH</code> , <code>UNKNOWN_METHOD</code>
<code>header_name</code>	一意のAPIクライアントを識別するHTTPヘッダー。例えば <code>X-apikey</code> 。	文字列

属性	説明	サポートされている値
<code>per_client_ip</code>	設定されている場合、APIリソースにアクセスしている各一意のクライアントIPアドレスにスロットリング制限を適用します。	ブール値

ratelimits属性

`ratelimits` の各エントリは、以下の属性をサポートします。 `req_threshold` 属性は必須です。

属性	説明	サポートされている値
<code>req_threshold</code>	タイムスライスごとに許可される最大リクエスト数。このフィールドは必須です。	整数
<code>timeslice</code>	タイムスライス（ミリ秒単位、10の倍数）。デフォルトは1000ミリ秒です。	整数
<code>limittype</code>	制限タイプ。指定しない場合、デフォルトは <code>SMOOTH</code> です。	<code>BURSTY</code> , <code>SMOOTH</code>
<code>mode</code>	制限が適用されるメトリック。	<code>REQUEST_RATE</code> , <code>CONNECTION</code> , <code>TOKEN_RATE</code>
<code>alertsintimeslice</code>	タイムスライス内で発生させるアラートの数。	整数
<code>throttle_action</code>	制限を超過した場合に実行するアクション。 <code>DROP</code> は制限を超過したリクエストをドロップし、 <code>RESET</code> はクライアント接続をリセットし、 <code>REDIRECT</code> は指定されたURLにリダイレクトし、 <code>RESPOND</code> は <code>429 Exceeded</code>	<code>DROP</code> , <code>RESET</code> , <code>REDIRECT</code> , <code>RESPOND</code> , <code>NOOP</code>

属性	説明	サポートされている値
	<code>allowed rate of requests</code> で応答します。	
<code>redirect_url</code>	<code>throttle_action</code> が <code>REDIRECT</code> の場合に使用されるリダイレクトURL。	文字列
<code>logpackets</code>	メッセージをログに記録するかどうか、およびどのログに記録するかを指定する監査メッセージアクションを追加します。	オブジェクト (<code>logexpression</code> , <code>loglevel</code>)

The `logpackets` オブジェクトは次の属性をサポートします。 `logpackets` が使用される場合、両方とも必須です。

属性	説明	サポートされる値
<code>logexpression</code>	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の式。	文字列 (最大長 7991)
<code>loglevel</code>	生成されるログメッセージの重大度を指定する監査ログレベル。	<code>EMERGENCY</code> , <code>ALERT</code> , <code>CRITICAL</code> , <code>ERROR</code> , <code>WARNING</code> , <code>NOTICE</code> , <code>INFORMATIONAL</code> , <code>DEBUG</code>

ストリーム属性

`streams` の各エントリは次の属性をサポートします。 `sort` 属性は必須です。

属性	説明	サポートされている値
<code>interval</code>	セッション統計 (リクエスト数、帯域幅、応答時間) を計算する際に使用するデータの分数。	整数 (1~10080)
<code>sampleCount</code>	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	整数 (1~65535)

属性	説明	サポートされている値
	には、サンプル数を1に設定します。	
<code>sort</code>	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。このフィールドは必須です。	<code>REQUESTS</code> (デフォルト), <code>CONNECTIONS</code> , <code>RESPTIME</code> , <code>BANDWIDTH</code> , <code>RESPTIME_BREACHES</code> , <code>TOKENS</code> , <code>NONE</code>
<code>snmpTrap</code>	ストリーム識別子に対するSNMPトラップを有効または無効にします。	<code>ENABLED</code> , <code>DISABLED</code>
<code>appflowLog</code>	ストリーム識別子に対するAppFlowロギングを有効または無効にします。	<code>ENABLED</code> , <code>DISABLED</code>
<code>trackAckOnlyPackets</code>	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。	<code>ENABLED</code> , <code>DISABLED</code>
<code>trackTransactions</code>	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。 <code>TOKENS</code> 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。	<code>RESPTIME</code> , <code>TOKENS</code> , <code>NONE</code>
<code>maxTransactionThreshold</code>	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。	整数 (最小値 0)
<code>minTransactionThreshold</code>	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。	整数 (最小値 0)
<code>acceptanceThreshold</code>	違反しないトランザクションの合計トランザクションに対するしきい値。パーセンテージで表されます。小数点以下6桁までサポートされます。	文字列 (最大長 10)

属性	説明	サポートされている値
<code>breachThreshold</code>	間隔で計算された違反トランザクションのしきい値。	整数 (最小値 0)
<code>log</code>	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。	<code>SYSLOG</code> , <code>NONE</code>
<code>logInterval</code>	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。	整数 (1–10080)
<code>logLimit</code>	ログ間隔でログに記録されるオブジェクトの最大数。	整数 (1–1000)

レート制限設定の記述方法

`ratelimit` YAML定義で、`kind` を `ratelimit` として設定します。`spec` セクションで、`selector_keys` でスロットルするトラフィックを選択し、`ratelimits` の下に制限を、`streams` の下に分析を、またはその両方を定義します。

以下のガイドラインを念頭に置いてください。

- `ratelimits` または `streams` の少なくともいずれか1つを指定します。
- 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、`mode: TOKEN_RATE` を使用します。
- クライアントごとの制限を適用するには、`selector_keys.basic.per_client_ip` または `header_name` を使用します。
- `ratelimit` CRDを `ExtensionRef` フィルターを介して `aigatewayroute` にアタッチするか、`servicenames` または `targetRef` を介してサービスにバインドします。

レート制限設定の例

トークンベースのレート制限

以下の設定では、クライアントIPアドレスごとのトラフィックを1分あたり100,000トークンに制限し、制限を超過した場合、`429` ステータスで応答します。

```
apiVersion: citrix.com/v1beta1

kind: ratelimit

metadata:

  name: token-ratelimit

  namespace: default

spec:

  selector_keys:

    basic:

      path:

        - /v1/chat/completions

      method:

        - POST

      per_client_ip: true

  ratelimits:

    - req_threshold: 100000

      timeslice: 60000

      mode: TOKEN_RATE

      limittype: SMOOTH

      throttle_action: RESPOND
```

ロギング付きのAPIクライアントごとのリクエストレート制限

以下の設定では、各APIクライアント（`x-apikey` ヘッダーで識別）を1秒あたり500リクエストに制限し、制限に違反した場合にログを記録します。

```
apiVersion: citrix.com/v1beta1

kind: ratelimit

metadata:

  name: apikey-ratelimit

  namespace: default

spec:

  servicenames:

    - ai-backend-service

  selector_keys:

    basic:

      header_name: X-apikey

  ratelimits:

    - req_threshold: 500

      timeslice: 1000

      mode: REQUEST_RATE

      limittype: BURSTY

      throttle_action: DROP

      logpackets:

        logexpression: "\"Rate limit exceeded for client: \" +
HTTP.REQ.HEADER(\"X-apikey\")\"

        loglevel: WARNING
```

トラフィック分析用のストリーム識別子

以下の設定では、制限を適用せずにトークン使用量に関する分析を収集します。

```
apiVersion: citrix.com/v1beta1

kind: ratelimit

metadata:

  name: token-analytics

  namespace: default

spec:

  selector_keys:

    basic:

      path:

        - /v1/chat/completions

  streams:

    - interval: 5

      sampleCount: 1

      sort: TOKENS

      appflowLog: ENABLED

      trackTransactions: TOKENS

      log: SYSLOG

      logInterval: 5

      logLimit: 100
```

関連情報

- [レート制限とストリームの設定](#)
- [AI Gatewayルート](#)
- [可観測性](#)
- [NetScalerのAI機能](#)

リライトおよびレスポンスポリシー

`rewritepolicy` カスタムリソース定義（CRD）を使用すると、Kubernetesネイティブ構成を使用してNetScaler®でリライトおよびレスポンスポリシーを構成できます。これらのポリシーにより、リクエストとレスポンスの情報をリライトしたり、リクエストの特性に基づいてリクエストに応答したりできます。

`rewritepolicy` CRDは、次のようなシナリオで使用できます。

- リクエストまたはレスポンスのURL、ホスト名、またはヘッダーをリライトする。
- HTTPヘッダーを挿入、削除、または置換する。
- リクエストを別のURLにリダイレクトする、カスタムペイロードで応答する、または接続をドロップしてリセットする。
- ポリシー式でデータセット、パットセット、および文字列マップを参照する。
- ポリシー決定で使用される情報を取得するために、外部エージェントへのHTTPコールアウトを行う。

`rewritepolicy` CRDは、`ExtensionRef` フィルターを介して `aigatewayroute` から参照されるか、`servicenames` または `targetRef` を介してサービスに直接適用されます。

注:

`rewrite-policies` または `responder-policies` の少なくともいずれか1つを指定する必要があります。

`rewritepolicy` CRDを展開する

`rewritepolicy` CRDをダウンロードし、次のコマンドを使用して展開します。

```
kubectl create -f https://raw.githubusercontent.com/netScaler/netScaler-k8s-ingress-controller/refs/heads/master/crd/rewrite-policy/rewrite-responder-policies-deployment.yaml
```

注:

CRD展開YAMLファイルを変更しないでください。

リライトポリシー CRD 属性

以下の表に、`rewritepolicy` CRD の `spec` で利用可能なトップレベルの属性を示します。

属性	説明	サポートされている値
<code>ingressclass</code>	インGRESSクラス。指定されていない場合、クラスター内のすべてのNetScaler Ingress Controllerがリソースを処理します。それ以外の場合、そのインGRESSクラスを持つコントローラーのみがそれを処理します。	文字列
<code>gatewayClassName</code>	このプロファイルが適用される <code>GatewayClass</code> の名前。	文字列
<code>targetRef</code>	このプロファイルが適用されるターゲットリソースのリスト (<code>name</code> , <code>namespace</code> , <code>group</code> , <code>kind</code> , <code>sectionName</code>)。	配列
<code>rewrite-policies</code>	リライトポリシーのリスト。	配列
<code>responder-policies</code>	レスポンスポリシーのリスト。	配列
<code>dataset</code>	ポリシー式によって参照されるデータセットのリスト。	配列

属性	説明	サポートされている値
<code>patset</code>	ポリシー式によって参照されるパターンセットのリスト。	配列
<code>stringmap</code>	ポリシー式によって参照される文字列マップのリスト。	配列
<code>httpcallout_policy</code>	HTTPコールアウト設定のリスト。	配列

リライトポリシー

`rewrite-policies` の各エントリは、次の属性をサポートしています。 `rewrite-policy` オブジェクトは必須です。

属性	説明	サポートされている値
<code>servicenames</code>	リライトポリシーにバインドするサービスの名前。	文字列の配列（最大長127）
<code>goto-priority-expression</code>	現在のポリシーが <code>TRUE</code> と評価された場合に、次に評価するポリシーを指定する式または値。 <code>NEXT</code> は次に高い優先度番号を持つポリシーを評価し、 <code>END</code> はポリシー評価を終了します。デフォルトは <code>END</code> です。	文字列（最大長1499）
<code>logpackets</code>	監査メッセージアクションを追加します (<code>logexpression</code> , <code>loglevel</code>)。	オブジェクト
<code>rewrite-policy</code>	リライトポリシーの定義。このオブジェクトは必須です。	オブジェクト

`rewrite-policy` オブジェクトは次の属性をサポートしています。 `rewrite-criteria` 、 `operation` 、 `target` 、および `direction` 属性は必須です。

属性	説明	サポートされる値
<code>rewrite-criteria</code>	トラフィックが評価される式。必須。	文字列 (最大長 1299)
<code>operation</code>	ユーザー定義のリライトアクションのタイプ。必須。	<code>noop</code> 、 <code>delete</code> 、 <code>insert_http_header</code> 、 <code>delete_http_header</code> 、 <code>corrupt_http_header</code> 、 <code>insert_before</code> 、 <code>insert_after</code> 、 <code>replace</code> 、 <code>replace_http_res</code> 、 <code>delete_all</code> 、 <code>replace_all</code> 、 <code>insert_before_all</code> 、 <code>insert_after_all</code> 、およびその他のサポートされる操作
<code>target</code>	リクエストまたはレスポンスのどの部分を書き換えるかを指定するデフォルト構文の式。必須。	文字列 (最大長 1229)
<code>direction</code>	ポリシーをバインドするバインドポイント。必須。	<code>REQUEST</code> 、 <code>RESPONSE</code>
<code>default-action</code>	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。	<code>NOREWRITE</code> 、 <code>RESET</code> 、 <code>DROP</code>
<code>modify-expression</code>	挿入するコンテンツ、または指定された文字列を置き換えるコンテンツを指定するデフォルト構文の式。	文字列 (最大長 7991)
<code>multiple-occurrence-modify</code>	リクエストまたはレスポンス内の複数の文字列を照合するために使用される検索機能。	文字列 (最大長 171)
<code>additional-multiple-occurrence-modify</code>	検索結果を絞り込むための追加条件。 <code>extend(m,n)</code> 操作で始まります。	文字列 (最大長 1299)

属性	説明	サポートされる値
<code>comment</code>	このリライトポリシーに関する情報を保持するためのコメント。	文字列 (最大長 255)

レスポnderポリシー

`responder-policies` の各エントリは、次の属性をサポートしています。 `responder-policy` オブジェクトは必須です。

属性	説明	サポートされている値
<code>servicenames</code>	レスポnderポリシーにバインドするサービスの名前。	文字列の配列 (最大長 127)
<code>goto-priority-expression</code>	現在のポリシーが <code>TRUE</code> と評価された場合に、評価する次のポリシーを指定する式または値。デフォルトは <code>END</code> です。	文字列 (最大長 1499)
<code>logpackets</code>	監査メッセージアクションを追加します (<code>logexpression</code> , <code>loglevel</code>)。	オブジェクト
<code>responder-policy</code>	レスポnderポリシーの定義。このオブジェクトは必須です。	オブジェクト

`responder-policy` オブジェクトは次の属性をサポートします。 `respond-criteria` 属性は必須であり、 `redirect` 、 `respondwith` 、 `noop` 、 `reset` 、または `drop` のいずれか1つを指定する必要があります。

属性	説明	サポートされる値
<code>respond-criteria</code>	ポリシーがリクエストに応答するかどうかを決定するために使用するデフォルト構文の式。必須。	文字列 (最大長 1299)

属性	説明	サポートされる値
<code>redirect</code>	ポリシーに一致した場合にリクエストをリダイレクトする (<code>url</code> 必須, <code>redirect-status-code</code> , <code>redirect-reason</code>)。	オブジェクト
<code>respondwith</code>	ポリシーに一致した場合にリクエストに応答する (<code>http-payload-string</code> 必須)。	オブジェクト
<code>noop</code>	リクエストに応答する代わりに、保護されたサーバーにリクエストを送信します。	オブジェクト
<code>reset</code>	リクエストがポリシーに一致した場合、クライアント接続を閉じてリセットします。	オブジェクト
<code>drop</code>	ポリシーに一致した場合、応答を送信せずにリクエストを破棄します。	オブジェクト
<code>default-action</code>	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。	<code>NOOP</code> , <code>RESET</code> , <code>DROP</code>
<code>comment</code>	このレスポンスポリシーに関する情報を保持するためのコメント。	文字列 (最大長 255)

データセット

`dataset` の各エントリは、次の属性をサポートしています。 `name` 、 `type` 、および `values` 属性は必須です。

属性	説明	サポートされている値
<code>name</code>	データセットの名前。	文字列 (最大長 32)
<code>type</code>	データセットにバインドする値のタイプ。	<code>ipv4</code> , <code>number</code> , <code>ipv6</code> , <code>ulong</code> , <code>double</code> , <code>mac</code>

属性	説明	サポートされている値
<code>values</code>	このデータセットに関連付けられた、指定された型の値。	文字列の配列
<code>comment</code>	このデータセットに関する情報を保持するためのコメント。	文字列（最大長255）

パットセット

`patset` の各エントリは、次の属性をサポートしています。 `name` および `values` 属性は必須です。

属性	説明	サポートされている値
<code>name</code>	パットセットの名前。	文字列（最大長32）
<code>values</code>	このパットセットに関連付けられたパターンを構成する文字列。	文字列の配列
<code>comment</code>	このパットセットに関する情報を保持するためのコメント。	文字列 (最大長 255)

文字列マップ

`stringmap` の各エントリは、次の属性をサポートします。 `name` および `values` 属性は必須です。

属性	説明	サポートされる値
<code>name</code>	文字列マップの名前。	文字列 (最大長 32)
<code>values</code>	この文字列マップにバインドするキーと値のペアのリスト。各ペアには <code>key</code> と <code>value</code> があります (それぞれ最大長2047文字)。	オブジェクトの配列

属性	説明	サポートされる値
<code>comment</code>	この文字列マップに関する情報を保持するためのコメント。	文字列 (最大長255)

HTTPコールアウト

`httpcallout_policy` の各エントリは、次の属性をサポートしています。 `name`、`server_ip`、および `server_port` 属性は必須です。

属性	説明	サポートされている値
<code>name</code>	HTTPコールアウト名。	文字列 (最大長32)
<code>server_ip</code>	コールアウトが送信されるコールアウトエージェントサーバーのIPアドレス。	文字列
<code>server_port</code>	コールアウトエージェントサーバーのポート。	整数 (1～65535)
<code>http_method</code>	HTTPリクエストで使用されるメソッド。デフォルトは <code>GET</code> です。	<code>GET</code> 、 <code>POST</code>
<code>host_expr</code>	<code>Host</code> ヘッダーを設定するための文字列式。完全なHTTPリクエスト式とは相互排他的です。	文字列 (最大長 255)
<code>url_stem_expr</code>	URLステムを生成するための文字列式。	文字列 (最大長 8191)
<code>headers</code>	HTTPリクエストに挿入する1つ以上のヘッダー (<code>name</code> , <code>expr</code>)。最大8つのヘッダー。	オブジェクトの配列
<code>parameters</code>	リクエストに挿入する1つ以上のクエリパラメータ (<code>name</code> , <code>expr</code>)。	オブジェクトの配列

属性	説明	サポートされている値
<code>body_expr</code>	リクエストのボディを生成するための高度な文字列式。	文字列
<code>full_req_expr</code>	NetScalerがコールアウトエージェントに送信する、式としての正確なHTTPリクエスト。	文字列
<code>scheme</code>	コールアウトサーバーのスキーム。デフォルトは <code>HTTP</code> です。	<code>HTTP</code> , <code>HTTPS</code>
<code>cache_for_secs</code>	コールアウト応答がキャッシュされる期間（秒単位）。	整数 (1～31536000)
<code>return_type</code>	コールアウトエージェントが返すデータのタイプ。 <code>TEXT</code> は値をテキストとして、 <code>NUM</code> は数値として、 <code>BOOL</code> はブール値として扱います。	<code>TEXT</code> , <code>NUM</code> , <code>BOOL</code>
<code>result_expr</code>	応答からコールアウト結果を抽出する式。応答ベースの式である必要があります (<code>HTTP.RES</code> で始まります)。	文字列 (最大長 8191)
<code>comment</code>	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。	文字列 (最大長 255)

ポリシー構成の記述方法

`rewritepolicy` YAML定義で、`kind` を `rewritepolicy` に設定します。`spec` セクションで、次のセクションを1つ以上追加します: `rewrite-policies`、`responder-policies`、`dataset`、`patset`、`stringmap`、および `httpcallout_policy`。

次のガイドラインに留意してください。

- `rewrite-policies` または `responder-policies` の少なくともいずれか1つを指定します。

- リライトポリシーの場合、`rewrite-criteria`、`operation`、`target`、および `direction` 属性は必須です。
- レスポンダーポリシーの場合、`respond-criteria` 属性は必須であり、`redirect`、`respondwith`、`noop`、`reset`、または `drop` のいずれか1つのアクションを正確に指定する必要があります。
- `rewritepolicy` CRDを `ExtensionRef` フィルターを介して `aigatewayroute` にアタッチするか、または `servicenames` または `targetRef` を介してサービスにバインドします。

ポリシー構成の例

応答ヘッダーを追加するリライトポリシー

次の構成は、HTTP応答にカスタムヘッダーを挿入します。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: add-response-header

  namespace: default

spec:

  rewrite-policies:

    - servicenames:

      - ai-backend-service

      rewrite-policy:

        operation: insert_http_header

        target: "x-ai-gateway"

        modify-expression: "\"netscaler\""

        rewrite-criteria: "HTTP.RES.IS_VALID"
```

```
direction: RESPONSE
```

```
comment: "Add AI gateway response header"
```

リクエストをリダイレクトするレスポnderポリシー

次の構成は、一致するリクエストを別のURLにリダイレクトします。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: redirect-policy

  namespace: default

spec:

  responder-policies:

    - servicenames:

      - ai-backend-service

    responder-policy:

      redirect:

        url: "\"https://backup.example.com\" + HTTP.REQ.URL"

        redirect-status-code: 302

      respond-criteria: "HTTP.REQ.URL.CONTAINS(\"/deprecated\")"

      comment: "Redirect deprecated endpoint"
```

カスタム応答を持つレスポnderポリシー

次の構成は、カスタムペイロードを使用して、一致するリクエストに直接応答します。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: block-policy

  namespace: default

spec:

  responder-policies:

    - servicenames:

      - ai-backend-service

    responder-policy:

      respondwith:

        http-payload-string: "\"HTTP/1.1 429 Too Many Requests\\r\\n\\r\\nRate limit exceeded\""

        respond-criteria: "HTTP.REQ.HEADER(\"x-ai-eg-model\").EQ(\"restricted-model\")"

        comment: "Block restricted model requests"
```

データセットを持つリライトポリシー

次の構成は、データセットを定義し、それをリライトポリシーで使します。

```
apiVersion: citrix.com/v1

kind: rewritepolicy

metadata:

  name: dataset-policy
```

```
namespace: default

spec:

  dataset:

    - name: blocked_ips

    type: ipv4

    values:

      - "192.0.2.1"

      - "192.0.2.2"

  responder-policies:

    - servicenames:

      - ai-backend-service

    responder-policy:

      drop: {}

      respond-criteria: "CLIENT.IP.SRC.EQUALS_ANY(\"blocked_ips\")"

      comment: "Drop requests from blocked IPs"
```

関連資料

- [プロンプト管理を構成する](#)
- [代替モデルへのスピルオーバーを構成する](#)
- [AI Gatewayルート](#)
- [NetScaler のリライトおよびレスポンスポリシー](#)

fix

This is the content for the sample heading topic. It is hidden from the main table of contents but remains accessible via links.