

NetScaler® Kubernetes Gateway Controller

Contents

Release notes	12
Release 2.2.0	12
NetScaler® Kubernetes Gateway Controller release 2.2.0	12
What's new	12
Helm chart release	13
Release 2.1.0	13
NetScaler® Kubernetes Gateway Controller release 2.1.0	13
What's new	13
Fixed issues	13
Helm chart release	14
Release 2.0.0	14
NetScaler® Kubernetes Gateway Controller release 2.0.0	14
What's new	14
Helm chart release	15
Release 1.4.0	15
NetScaler® Kubernetes Gateway Controller release 1.4.0	16
Fixed issues	16
Helm chart release	16
Release 1.3.0	16
NetScaler® Kubernetes Gateway Controller release 1.3.0	16
What's new	16
Helm chart release	17
Release 1.2.0	17
NetScaler Kubernetes Gateway Controller release 1.2.0	17
What's new	17
Helm chart release	17

Release 1.1.0	18
NetScaler Kubernetes Gateway Controller release 1.1.0	18
What's new	18
Helm chart release	18
Release 1.0.0	19
NetScaler Kubernetes Gateway Controller release 1.0.0	19
What's new	19
Helm chart release 1.0.0	19
Overview	20
NetScaler® Kubernetes Gateway Controller	20
NetScaler as a Kubernetes Gateway API provider	21
Use Cases	21
Getting started	22
Prerequisites	22
How to deploy NetScaler® Kubernetes Gateway Controller in a Kubernetes cluster	22
How to deploy NetScaler CPX with NetScaler Kubernetes Gateway Controller in a Kubernetes cluster	24
What's next	26
Supported Gateway API CRDs	27
Gateway Class CRD	27
Gateway CRD	27
HTTPRoute CRD	28
TCPRoute CRD	29
UDPRoute CRD	30
Points to note	31
Supported platforms and deployments	31
Kubernetes platforms	32
Compatibility with NetScaler	32

NetScaler Operator compatibility with NetScaler Kubernetes Gateway Controller	33
Install	34
Deploy NetScaler® Kubernetes Gateway Controller	34
Exposing an application by using HTTP Using Gateway and HTTPRoute CRDs	39
Exposing an application by using HTTPS/SSL Using Gateway and HTTPRoute CRDs	43
Exposing a TCP application by using Gateway and TCPRoute CRDs	47
Step 1: Deploy a sample TCP application	47
Step 2: Create a GatewayClass	48
Step 3: Create a Gateway with a TCP listener	49
Step 4: Create a TCPRoute	50
Step 5: Verify the configuration	51
Exposing a UDP Application Using Gateway and UDPRoute CRDs	51
Step 1: Deploy a sample UDP application	51
Step 2: Create a GatewayClass	54
Step 3: Create a Gateway with a UDP listener	54
Step 4: Create a UDPRoute	55
Step 5: Verify the configuration	56
Deploy NetScaler® CPX Kubernetes Gateway Controller	57
Sample Gateway and HTTPRoute CRD	61
Exposing a TCP application by using Gateway and TCPRoute CRDs	67
Step 1: Deploy a sample TCP application	67
Step 2: Create a GatewayClass	69
Step 3: Create a Gateway with a TCP listener	70
Step 4: Create a TCPRoute	71
Step 5: Verify the configuration	72
Exposing a UDP Application Using Gateway and UDPRoute CRDs	72
Step 1: Deploy a sample UDP application	72
Step 2: Create a GatewayClass	75

Step 3: Create a Gateway with a UDP listener	75
Step 4: Create a UDPRoute	76
Step 5: Verify the configuration	77
NetScaler Operator	78
NetScaler Operator release notes	78
NetScaler® Operator release notes	78
Version 3.6.0	78
NetScaler® Operator release notes	84
Version 3.5.0	84
NetScaler® Operator release notes	85
Version 3.4.1	85
NetScaler® Operator release notes	86
Version 3.3.0	86
Deploy NetScaler Kubernetes Gateway Controller in OpenShift using NetScaler Operator	86
Prerequisites	86
Deploy NetScaler Kubernetes Gateway Controller using OpenShift Operator	87
Deploy NetScaler Kubernetes Gateway Controller with CPX in OpenShift using NetScaler Operator	105
Prerequisites	105
Deploy NetScaler Kubernetes Gateway Controller as a sidecar with NetScaler CPX	106
Network	127
Establish a network between Kubernetes nodes and NetScaler instance outside the Kubernetes cluster	127
Multiple clusters behind a single NetScaler	127
Traffic management	129
Dynamic IP address allocation through NetScaler IPAM Controller	129
Prerequisites	129
Configuration	130

Step 1: Create a GatewayClass	130
Step 2: Create a Gateway	130
Verify the IP address allocation	132
Static IP versus IPAM	132
Annotation Reference	133
Troubleshooting	133
Limitations	134
Multiple listeners within a Gateway	134
Sample configuration	134
Namespace filters	136
Enhanced traffic management capabilities with HTTPRoute filters	137
Integration with NetScaler CRDs by using extensionRef	137
Sample reference	138
Native support for standard HttpRoute API filters	139
URLRewrite	139
RequestHeaderModifier and ResponseHeaderModifier	140
RequestRedirect	141
RequestMirror	141
NetScaler® Profiles support	143
Deploy the Profile CRDs	143
Profile specification	144
Add NetScaler profiles in NetScaler Kubernetes Gateway Controller	144
Frontend profiles	144
Backend Profiles	145
Types of NetScaler profiles	147
SSL Profile	148
TCP profile	155
HTTP Profile	162

Controller scoping through <code>spec.gatewayClassName</code>	168
Traffic security	170
Using pre-configured certificates with NetScaler Kubernetes Gateway Controller	170
Benefits	170
How to reference pre-configured certificates	170
Certificate types	170
Basic configuration	171
Common Scenarios	174
Scenario 1: Single certificate	174
Scenario 2: Multiple domains certificates	174
Scenario 3: Client authentication with CA certificate	175
Scenario 4: Using certificate bundles	176
Scenario 5: Combining pre-configured and Kubernetes certificates	177
Prerequisites	177
Troubleshooting	177
Summary	178
Certificate key bundle in NetScaler® by using the NetScaler Kubernetes Gateway Controller	179
Points to note	179
Use "appProtocol" to define backend application communication protocol	180
AI Gateway	182
Overview	182
When to use AI Gateway	182
Deployment options	182
Architecture	183
Key Kubernetes resources	183
Related topics	184
Reference	184
Deploy NetScaler Kubernetes Gateway Controller with AI capabilities	185

Prerequisites	185
Deploy NetScaler Kubernetes Gateway Controller with NetScaler VPX/MPX/BLX	185
Install NetScaler Kubernetes Gateway Controller	185
Create the GatewayClass	186
Deploy NetScaler CPX with Gateway Controller	186
Install NetScaler CPX with Gateway Controller	186
Create the GatewayClass	187
Verify the deployment	188
Reference	188
Configure model routing	188
Prerequisites	188
Configuration	189
AI Gateway profiles	189
Kubernetes Gateway with AI Gateway profile	191
NetScaler AI Gateway route	193
Reference	196
Configure rate limits and streams	196
Per Gateway Vs Per Model rate limits	196
Configuration	197
Apply per-model rate limits	197
Apply Gateway-level rate limits	202
Reference	206
Configure prompt management	206
Use cases	206
RewritePolicy CRD for prompt management	206
Per Gateway Vs Per Model rewrite policies	206
Configuration	207
Per-model rewrite policy	207

Rewrite policy at Kubernetes Gateway	209
Reference	210
Configure spillover to an alternate model	210
How it works	211
Key configuration elements	211
Configuration	211
Spillover backend profile	211
Spillover rewrite policy	213
AI Gateway routes with spillover service	214
Reference	216
Observability	217
Prerequisites	217
Configuration overview	217
Global versus route-level profiles	218
Create the Splunk token secret	218
Configure the Splunk connection in Helm values (mandatory)	218
Override the analytics profile per route	220
Custom AI Gateway metrics	222
NetScaler CPX	222
NetScaler VPX/MPX/BLX	223
Verify	223
Reference	224
NetScaler CRDs	225
AI Gateway profile	225
Deploy the aigatewayprofile CRD	225
aigatewayprofile CRD attributes	226
config attributes	226
authtoken attributes	227

How to write the profile configuration	227
Sample profile configurations	228
Frontend profile	228
Backend profile with a token quota	228
Preconfigured profile	229
Related articles	230
AI Gateway route	230
Deploy the aigatewayroute CRD	231
aigatewayroute CRD attributes	231
aigatewayprofiles attributes	232
parentRefs attributes	232
rules attributes	233
How to write the route configuration	236
Sample route configurations	236
Model routing to multiple backends	236
Priority-based model spillover	238
Attach a rate limit policy to a rule	240
Related articles	241
Rate limit	241
Deploy the ratelimit CRD	242
ratelimit CRD attributes	242
targetRef attributes	243
selector_keys attributes	243
ratelimits attributes	244
streams attributes	245
How to write the rate limit configuration	247
Sample rate limit configurations	248
Token-based rate limiting	248

Per-API-client request rate limiting with logging	249
Stream identifier for traffic analytics	250
Related articles	251
Rewrite and responder policies	251
Deploy the rewritepolicy CRD	251
rewritepolicy CRD attributes	252
Rewrite policy	253
Responder policy	255
Dataset	256
Patset	257
String map	257
HTTP callout	258
How to write the policy configuration	259
Sample policy configurations	260
Rewrite policy to add a response header	260
Responder policy to redirect a request	260
Responder policy with a custom response	261
Rewrite policy with a dataset	262
Related articles	263
fix	264

Release notes

Release notes describe how the software has changed in a particular build, and the issues known to exist in the build. The release notes document includes all or some of the following sections:

- **What's New:** The enhancements and other changes released in the build.
- **Fixed Issues:** The issues that are fixed in the build.
- **Known Issues:** The issues that exist in the build.

The following is the list of NetScaler® Kubernetes Gateway Controller releases:

- [Release 2.2.0](#)
- [Release 2.1.0](#)
- [Release 2.0.0](#)
- [Release 1.4.0](#)
- [Release 1.3.0](#)
- [Release 1.2.0](#)
- [Release 1.1.0](#)
- [Release 1.0.0](#)

Release 2.2.0

This release notes document describes the enhancements and changes, fixed and known issues that exist for release 2.2.0.

NetScaler® Kubernetes Gateway Controller release 2.2.0

What's new

Support for NetScaler AI Gateway

From NetScaler Kubernetes Gateway Controller release 2.2.0 onwards, the NetScaler AI Gateway feature is supported. For more information on NetScaler AI Gateway, see [AI Gateway](#).

Helm chart release

For information on Helm chart release, see [Helm Chart Release Notes](#).

Release 2.1.0

This release notes document describes the enhancements and changes, fixed and known issues that exist for release 2.1.0.

NetScaler® Kubernetes Gateway Controller release 2.1.0

What's new

Exposing a TCP/UDP application by using Gateway and TCPRoute/UDPRoute CRDs

NetScaler Kubernetes Gateway Controller supports exposing a TCP/UDP-based application by using the Kubernetes Gateway API with NetScaler.

For more information, see [Deploy NetScaler Kubernetes Gateway Controller](#) and [Deploy NetScaler CPX Kubernetes Gateway Controller](#).

Fixed issues

- WebSocket is disabled by default. You can now use the `HTTPProfile` CRD for this use case.
- Issues observed in the `HTTPProfile` CRD controller when `GatewayClass` is provided in the profile. The `GatewayClass` name provided in the profile might cause no attribute 'ref_manager' error.
- Host names are not getting the correct precedence over wildcards when they share the same parent domain. As a fix for this issue, routing priority for overlapping host names on the same content switching virtual server is enhanced.
- `RequestHeaderModifier.set` works only if the header already exists.
- `URLRewrite.replacePrefixMatch` produces double slashes, and therefore path normalization is not handled correctly.
- Any two HTTP Routes whose names are the same up to 20 characters create one load balancing virtual server instead of two. Conflicts are observed in load balancing virtual server naming.
- Load balancing virtual server service group binding is lost after controller restart or service event is received after the HTTP route.

- During GET calls to the Kubernetes API, the `429 errors` are observed.

Helm chart release

For information on Helm chart release, see [Helm Chart Release Notes](#).

Release 2.0.0

This release notes document describes the enhancements and changes, fixed and known issues that exist for release 2.0.0.

NetScaler® Kubernetes Gateway Controller release 2.0.0

What's new

Using pre-configured certificates with NetScaler Kubernetes Gateway Controller

You can now use certificates or certificate bundles that are already configured on your NetScaler appliance with your Kubernetes Gateway. This feature simplifies certificate management by allowing you to reference existing certificates instead of managing them as Kubernetes Secrets.

For more information, see [Using pre-configured certificates with NetScaler Kubernetes Gateway Controller](#).

Support to establish a network between Kubernetes nodes and the NetScaler instance outside the Kubernetes cluster

For seamless functioning of services deployed in a Kubernetes cluster, the Ingress NetScaler device must reach the overlay network where Pods run. You can get this connectivity in one of the following ways:

1. Enable automatic route configuration in the NetScaler Kubernetes Gateway Controller.

Set `gatewayController.nodeWatch: true` in the controller spec to create routes on NetScaler for each Pod subnet.

NOTE:

You can use this option when the NetScaler SNIP address shares the same subnet as the Kubernetes nodes.

2. Deploy a NetScaler Node Controller to drive subnet routes from Kubernetes nodes.

NOTE:

`gatewayController.nodeWatch` is `false` by default.

For more information, see [Establish a network between Kubernetes nodes and NetScaler instance outside the Kubernetes cluster](#).

NetScaler Kubernetes Gateway controller does not add the backend service group binding when a `RequestMirroring` filter is defined in `httproute`.

Entity name change for CRDs

When a NetScaler Custom Resource Definition (CRD) instance is created, the NetScaler Kubernetes Gateway Controller generates multiple NetScaler entities associated with that CRD instance. The NetScaler Kubernetes Gateway Controller maintains unique names for each entity to preserve its association with the CRD instance. Since entity naming is based directly on CRD names, some NetScaler entity names exceeded the maximum character limit.

Starting with NetScaler Kubernetes Gateway Controller 2.0.0, the naming convention is optimized by using the following approach to generate shorter entity names during CRD creation:

- Hashed naming: A portion of the entity name is hashed to reduce the overall length
- Preserved information: The necessary Kubernetes related metadata is retained in the entity's comment field, if the entity comment is supported by NetScaler
- Improved compatibility: Names comply with NetScaler character restrictions while maintaining full traceability

For more information, see [Entity name change for CRDs](#).

Helm chart release

For information on Helm chart release, see [Helm Chart Release Notes](#).

Release 1.4.0

This release notes document describes the enhancements and changes, fixed and known issues that exist for release 1.4.0.

NetScaler® Kubernetes Gateway Controller release 1.4.0

Fixed issues

NetScaler Kubernetes Gateway controller does not add the backend service group binding when a `RequestMirroring` filter is defined in `httproute`.

Helm chart release

For information on Helm chart release, see [Helm Chart Release Notes](#).

Release 1.3.0

This release notes document describes the enhancements and changes, fixed and known issues that exist for release 1.3.0.

NetScaler® Kubernetes Gateway Controller release 1.3.0

What's new

Support for NetScaler profiles

The NetScaler Kubernetes Gateway Controller supports profile CRDs (including SSL profile, HTTP profile, and TCP profile) to achieve robust, centralized configuration. These profiles allow cluster operators to define standardized, reusable infrastructure settings, such as TLS ciphers, TCP optimizations, or HTTP timeouts, that application developers can effortlessly apply to their routing definitions.

For more information, see [NetScaler Profiles support](#).

Support to use “appProtocol” to define backend application communication protocol

The NetScaler Kubernetes Gateway Controller uses the `appProtocol` field on a Kubernetes service port to explicitly set the protocol for the resulting NetScaler service group, that controls the communication between the NetScaler and the backend applications.

For example, setting `appProtocol: ssl` on the service port ensures the corresponding service group on the NetScaler is configured with the SSL protocol. If not explicitly set, the default service group protocol is HTTP.

For more information, see [Use “appProtocol” to define backend application communication protocol](#)

Helm chart release

For information on Helm chart release, see [Helm Chart Release Notes](#).

Release 1.2.0

This release notes document describes the enhancements and changes, fixed and known issues that exist for release 1.2.0.

NetScaler Kubernetes Gateway Controller release 1.2.0

What's new

Support for cross-namespace routing

You can now enable cross-namespace routing by configuring the `allowedRoutes` field within each listener definition of your Gateway resource. The `allowedRoutes` field provides granular control over which namespaces are permitted to attach routes to a specific listener, allowing for flexible and secure multitenant or multi-team deployments.

For more information, see [Namespace filters](#).

Support for RequestRedirect and RequestMirror filters

NetScaler® Kubernetes Gateway Controller now supports the following filters:

- **RequestRedirect** - Issues HTTP 3xx redirect responses and redirects users to different URLs, enforce HTTPS, or manage moved content gracefully.
- **RequestMirror** - Sends a copy of the request to different backend services, which is highly beneficial for traffic shadowing. This filter allows testing of new service versions, performing analytics, or debugging issues without affecting the client's response. The response from the mirrored backend is ignored.

For more information, see [RequestRedirect](#) and [RequestMirror](#).

CPX with Kubernetes Gateway Controller

You can now deploy the NetScaler CPX with Kubernetes Gateway controller. For more information, see [Deploy NetScaler CPX Kubernetes Gateway Controller](#).

Helm chart release

For information on Helm chart release, see [Helm Chart Release Notes](#).

Release 1.1.0

This release notes document describes the enhancements and changes, fixed and known issues that exist for release 1.1.0.

NetScaler Kubernetes Gateway Controller release 1.1.0

What's new

Enhanced traffic management capabilities with HTTPRoute filters

The NetScaler® Kubernetes Gateway controller supports HTTPRoute filters as defined by the Kubernetes Gateway API. This feature allows you to implement sophisticated traffic manipulation logic directly within the HTTPRoute resources. By leveraging filters, you can modify requests and responses, enabling a wide range of use cases such as header manipulation, URL rewriting, and request redirection.

For more information, see [Enhanced traffic management capabilities with HTTPRoute filters](#).

Support for multiple listeners with HTTP routes

The NetScaler Kubernetes Gateway Controller supports multiple listeners within a single Gateway resource, as defined by the Kubernetes Gateway API. This capability provides greater flexibility and control over how external traffic is managed and routed to services within your Kubernetes cluster.

For more information, see [Multiple Listeners with HTTP routes](#).

Certificate key bundle support in NetScaler by using the NetScaler Kubernetes Gateway Controller

NetScaler Kubernetes Gateway Controller now supports certificate bundle (`certkeybundle`) functionality, which is supported on NetScaler starting from release 14.1 build 21.x. With this functionality, the issue with the certificate chain and the additional handling that is required when two certificates share an intermediate CA are resolved. For more information on certificate key bundle support in NetScaler, see [Certificate key bundle in NetScaler by using the NetScaler Kubernetes Gateway Controller](#).

Helm chart release

For information on Helm chart release, see [Helm Chart Release Notes](#).

Release 1.0.0

This release notes document describes the enhancements and changes, fixed and known issues that exist for release 1.0.0.

NetScaler Kubernetes Gateway Controller release 1.0.0

What's new

NetScaler® Kubernetes Gateway Controller

The NetScaler Kubernetes Gateway Controller integrates NetScaler's Application Delivery Controller™ (ADC) capabilities with Kubernetes by using the Gateway API to provide advanced traffic management, enhanced flexibility, and secure application delivery for Kubernetes environments.

For information on key features, benefits, and how it enables secure and efficient application delivery in Kubernetes environments, see [NetScaler Kubernetes Gateway Controller](#)

Helm chart release 1.0.0

For information on Helm chart release, see `<https://github.com/netScaler/netScaler-helm-charts/releases/tag/nsk8sgwctr-1.0.0>`.

Overview

The following topics are available under Overview:

- [NetScaler Kubernetes Gateway Controller in Kubernetes](#)
- [Getting started](#)
- [NetScaler Kubernetes gateway controller CRDs](#)
- [Supported platforms and deployments](#)

NetScaler® Kubernetes Gateway Controller

Kubernetes offers various mechanisms for exposing applications, including Services and Ingress. While Ingress has been widely adopted, it has limitations in terms of expressiveness, extensibility, and support for advanced traffic management features.

The Kubernetes Gateway API is a next-generation API designed to address these limitations. It provides a more flexible, extensible, and role-oriented approach to managing external access to Kubernetes services. Key advantages of the Gateway API include:

- **Role-Based Resource Model:** Separates configuration responsibilities between different roles (for example, infrastructure provider, cluster operator, application developer).
- **Enhanced Expressiveness:** Supports advanced routing scenarios, traffic splitting, header-based routing, and more.
- **Extensibility:** Allows for custom extensions and features through GatewayClass parameters and policies.
- **Portability:** Aims for better portability across different Gateway implementations.

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.

This document provides an overview of the NetScaler Kubernetes Gateway API, outlining its key features, benefits, and how it enables secure and efficient application delivery in Kubernetes environments.

NetScaler as a Kubernetes Gateway API provider

NetScaler provides a robust and mature implementation of the Gateway API, building upon its years of experience as a leading ADC. This integration allows organizations to benefit from NetScaler's advanced features within their Kubernetes clusters. The NetScaler Kubernetes Gateway API implementation relies on the following key components:

- **NetScaler Kubernetes Gateway Controller:** The NetScaler Ingress Controller converts Ingress objects into NetScaler configurations, while the NetScaler Kubernetes Gateway translates Gateway API objects into NetScaler configurations. NetScaler Kubernetes Gateway continuously monitors the Kubernetes API server for changes in the gateway API resources. The NetScaler Kubernetes Gateway is a separate deployment from the NetScaler Ingress Controller. If both Ingress and Gateway API functionalities are needed, both the NetScaler Ingress Controller and the NetScaler Kubernetes Gateway must be deployed.
- **Custom Resource Definitions (CRDs):** NetScaler introduces custom resource definitions that extend the Kubernetes API to represent NetScaler-specific configurations and features. These CRDs work with the standard Gateway API resources.
- **Gateway API Resources:** NetScaler supports the core Gateway API resources, such as GatewayClass, Gateway, HTTPRoute, TCPRoute, and TLSRoute. These resources define how external traffic must be handled and routed to back-end services.

Use Cases

The NetScaler Kubernetes Gateway API implementation enables a wide range of use cases, including:

- **Securely Exposing Microservices:** Protect microservices deployed in Kubernetes with NetScaler's advanced security features.
- **Load Balancing External Traffic:** Distribute external traffic across multiple instances of Kubernetes services for high availability and performance.
- **Implementing Advanced Routing Rules:** Define complex routing rules based on headers, paths, and other criteria.
- **Enabling TLS Termination:** Offload TLS encryption and decryption to NetScaler for improved performance and security.
- **Implementing Canary Deployments:** Gradually roll out new application versions to a subset of users before a full deployment.
- **Traffic Splitting for A/B Testing:** Direct different percentages of traffic to different application versions for A/B testing.

- **Integrating with Identity Providers:** Implement pre-authentication and authorization using NetScaler's integration with various identity providers.

Getting started

Network administrators who have access to NetScaler (MPX/VPX/BLX) and Kubernetes environment can deploy NetScaler Kubernetes Gateway Controller in Kubernetes.

Prerequisites

- You have installed the Kubernetes cluster.
- You have installed Helm version 3.x or later. To install Helm, see [Helm charts](#).
- You have Installed the Gateway API CRDs from the [official standard channel](#).

How to deploy NetScaler® Kubernetes Gateway Controller in a Kubernetes cluster

To deploy NetScaler Kubernetes Gateway Controller in a Kubernetes Cluster configuring NetScaler MPX, VPX, or BLX, perform the following steps:

1. Generate `values.yaml` based on your use case. For information about the mandatory and optional parameters that you can configure during NetScaler Kubernetes Gateway Controller installation, see [Configurations](#).

```
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
```

The user name and password of a system user account on NetScaler MPX or NetScaler VPX are required for the Kubernetes Gateway. Ensure that NetScaler has a non-default system user account with the necessary privileges to allow the NetScaler Kubernetes Gateway Controller to configure NetScaler MPX or VPX. For instructions on creating a system user account on NetScaler, refer [Create a system user account for NetScaler Ingress Controller in NetScaler](#).

You can provide the user name and password directly or use Kubernetes secrets. To use Kubernetes secrets, create a secret for the user name and password with the following command:

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

NOTE:

`entityPrefix` and `gatewayControllerName` are mandatory and must be unique for every deployment of NetScaler Kubernetes Gateway Controller.

2. Deploy the NetScaler Kubernetes Gateway controller using the Helm charts and the `values.yaml` generated in previous step. For more information, see [NetScaler Kubernetes Gateway Controller deployment using Helm Chart](#).

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

3. Verify the installation.

```
kubectl get pods
```

4. (Optional) View the summary of the NetScaler Kubernetes Gateway Controller pod.

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

Example:

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

5. (Optional) Check the NetScaler Kubernetes Gateway Controller logs.

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

Example:

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

6. (Optional) Delete the installation.

```
helm uninstall gateway-controller
```

How to deploy NetScaler CPX with NetScaler Kubernetes Gateway Controller in a Kubernetes cluster

To deploy NetScaler Kubernetes Gateway Controller in a Kubernetes Cluster configuring NetScaler CPX, perform the following steps:

1. Generate `values.yaml` based on your use case. For information about the mandatory and optional parameters that you can configure during NetScaler CPX with Kubernetes Gateway Controller installation, see [Configurations](#).

```
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
```

NOTE:

`entityPrefix` and `gatewayControllerName` are mandatory and must be unique for every deployment of NetScaler CPX with Kubernetes Gateway Controller.

2. Deploy the NetScaler CPX with Kubernetes Gateway controller using the Helm charts and the `values.yaml` generated in previous step. For more information, see [NetScaler CPX with Kubernetes Gateway Controller deployment using Helm Chart](#).

```
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. Verify the installation.

```
kubectl get pods
```

4. (Optional) View the summary of the NetScaler Kubernetes Gateway Controller pod.

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

Example:

```
kubectl describe pod cpx-gateway-controller-netScaler-cpx-with-gateway-controllerz6ghg
```

5. (Optional) Check the NetScaler Kubernetes Gateway Controller logs.

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

Example:

```
kubectl logs -f cpx-gateway-controller-netScaler-cpx-with-gateway-controllerz6ghg -c nskgwc
```

6. (Optional) Delete the installation.

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

What's next

- [Deploy NetScaler Kubernetes Gateway Controller.](#)
- [Deploy NetScaler CPX with Kubernetes Gateway Controller.](#)

Supported Gateway API CRDs

The following sections outline the supported parameters for each Kubernetes Gateway API CRD in the latest released version. Any parameters or CRD outside the supported parameters listed in this topic are currently not supported.

Gateway Class CRD

PROPERTY	DESCRIPTION	STATUS
spec.controllerName	The name of the controller that must match with the name provided while deploying the controller.	Supported
spec.description	Human-readable description of the GatewayClass.	Supported
status.conditions	The status of this GatewayClass from the controller can be found in "Conditions."	Supported

Gateway CRD

PROPERTY	DESCRIPTION	STATUS
spec.addresses	The list of IP addresses. The Controller uses all the addresses and creates an ipset. The Controller uses the ipset as virtual IP address of a content switching virtual server created on NetScaler.	Supported
spec.gatewayClassName	The name of the GatewayClass to which this Gateway belongs.	Supported
spec.listeners	The list of network endpoints (listeners) that the Gateway exposes.	Supported

PROPERTY	DESCRIPTION	STATUS
spec.listeners.name	The name of the listener. This name must be unique within a Gateway.	Supported
spec.listeners[].port	The port number for the listener. The Controller uses this port as content switching virtual server port created on NetScaler.	Supported
spec.listeners[].protocol	The network protocol that the listener expects to receive. Controller uses this protocol as the virtual server IP address of a content switching virtual server created on NetScaler.	Supported
spec.listeners[].tls.certificateRefs	The CertificateRefs contains a series of references to Kubernetes objects that contain TLS certificates and private keys. Currently, the Controller only supports Kubernetes secrets.	Partially Supported

HTTPRoute CRD

PROPERTY	DESCRIPTION	STATUS
spec.hostnames	The list of host names this route matches. The Controller uses the hostname in the content switching policy rule expression.	Supported
spec.parentRefs[]	The references to the Gateways this route attaches to.	Partially Supported

PROPERTY	DESCRIPTION	STATUS
spec.parentRefs[].kind	The controller supports only the following parentRefs: Gateway	Partially Supported
spec.parentRefs[].name	The name of the Gateway Resource.	Supported
spec.parentRefs[].namespace	The namespace of the Gateway Resource.	Supported
spec.rules[]	Controller uses these rules for generating the content switching policy rule expression.	Supported
spec.rules[].matches	Controller uses path, method, queryparams, headers and so on to craft the content switching policy rule on NetScaler.	Supported
spec.rules[].backendRefs	Currently, Controller support only the following backendRefs: service, name, namespace, and port.	Partially Supported
spec.rules[].filters	Controller supports the following filters: urlRewrite, requestRedirect, responseHeaderModifier, requestHeaderModifier, extensionRef	Partially Supported

TCPRoute CRD

TCPRoute enables TCP traffic routing from a Gateway listener to backend Kubernetes services.

PROPERTY	DESCRIPTION	STATUS
spec.parentRefs[]	The references to the Gateways this route attaches to	Partially Supported
spec.parentRefs[].kind	Currently, the controller supports only the following parentRefs: Gateway	Supported
spec.parentRefs[].name	The name of the Gateway Resource.	Supported
spec.parentRefs[].namespace	The namespace of the Gateway Resource.	Supported
spec.parentRefs[].sectionName	The name of a specific Gateway listener to attach to.	Supported
spec.rules[]	List of rules defining how TCP traffic is routed to backend services.	Supported
spec.rules[].backendRefs[]	Currently, Controller supports only the following backendRefs: service, name, namespace, port.	Partially Supported

UDPRoute CRD

UDPRoute enables UDP traffic routing from a Gateway listener to backend Kubernetes services.

PROPERTY	DESCRIPTION	STATUS
spec.parentRefs[]	The references to the Gateways this route attaches to	Partially Supported
spec.parentRefs[].kind	Currently, the controller supports only the following parentRefs: Gateway	Supported

PROPERTY	DESCRIPTION	STATUS
<code>spec.parentRefs[].name</code>	The name of the Gateway Resource.	Supported
<code>spec.parentRefs[].namespace</code>	The namespace of the Gateway Resource.	Supported
<code>spec.parentRefs[].sectionName</code>	The name of a specific Gateway listener to attach to.	Supported
<code>spec.rules[]</code>	List of rules defining how UDP traffic is routed to backend services.	Supported
<code>spec.rules[].backendRefs[]</code>	Currently, Controller supports only the following backendRefs: service, name, namespace, port	Partially Supported

Points to note

- The controller considers only the supported parameters listed earlier; any unsupported parameters are ignored.
- Future versions might extend support for additional parameters.
- Ensure that the `gatewayClassName` is correctly set for Gateway resources so that they are recognized by the controller.
- All `parentRefs` in `HTTPRoute` must correctly reference existing and valid Gateways.
- Status updates are dynamically managed and might take time to reflect based on system load.
- Logs must be monitored for any parsing or validation errors when applying CRDs.
- IP addresses used in Gateway CRD can't be shared with Ingresses or Citrix® Listener resources.

Supported platforms and deployments

This topic provides details about supported Kubernetes platforms and compatibility of NetScaler Kubernetes Gateway Controller.

Kubernetes platforms

NetScaler Kubernetes Gateway Controller is supported on the following platforms:

- Kubernetes v1.27 (and later) on bare metal or self-hosted on public clouds such as AWS, GCP, or Azure.
- Google Kubernetes Engine (GKE)
- Elastic Kubernetes Service (EKS)
- Azure Kubernetes Service (AKS)
- Red Hat OpenShift version 4.15 and later
- Pivotal Container Service (PKS)
- Diamanti Enterprise Kubernetes Platform
- Mirantis Kubernetes Engine
- VMware Tanzu
- Rancher

Compatibility with NetScaler

The following table lists the NetScaler versions that are compatible with any NetScaler Kubernetes Gateway Controller release:

NETSCALER KUBERNETES GATEWAY CONTROLLER	NETSCALER VERSIONS
2.2.0	VPX/MPX/CPX 14.1-73.x and later
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 Operator compatibility with NetScaler Kubernetes Gateway Controller

The table lists NetScaler Kubernetes Gateway Controller releases that can be deployed using any NetScaler Operator release.

OPENSIFT VERSION	NETSCALER OPERATOR VERSION	NETSCALER KUBERNETES GATEWAY CONTROLLER
4.11 and later	3.6.0	1.4.0
4.11 and later	3.5.0	1.4.0
4.11 and later	3.4.1	1.4.0
4.11 and later	3.3.0	1.3.0

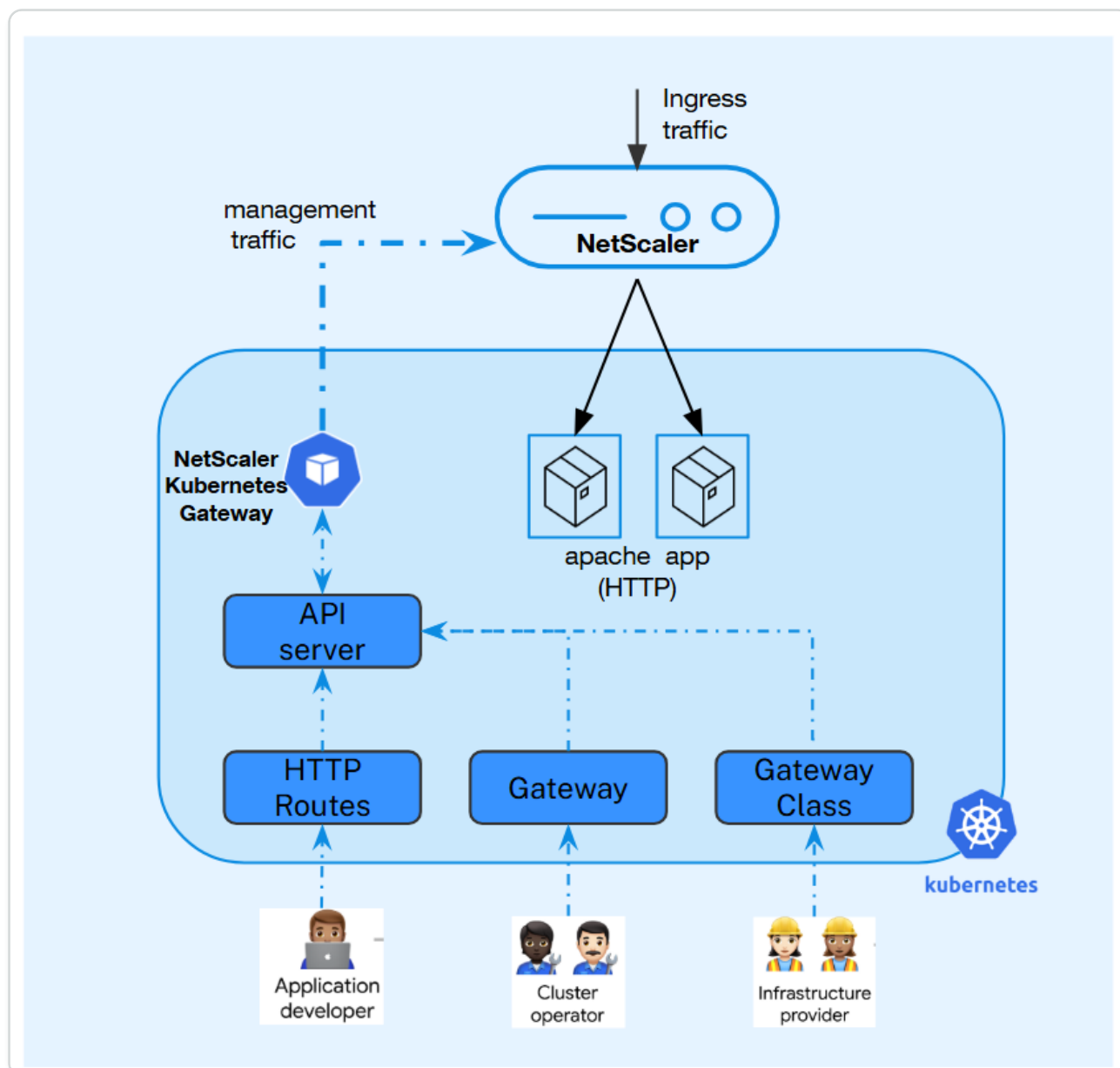
Install

The following topics are available under Install:

- [Deploy NetScaler Kubernetes Gateway Controller](#)
- [Deploy NetScaler CPX with Kubernetes Gateway Controller](#)

Deploy NetScaler® Kubernetes Gateway Controller

The deployment process involves deploying the Gateway Controller, which monitors and manages Gateway CRDs that define traffic routing rules. The Gateway Controller then configures the NetScaler VPX™ to direct external requests to the sample application.



1. Deploy a sample application to get started.

```
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. Install the Gateway API CRDs from the [official standard channel](#) if they are not already installed.
3. Generate `values.yaml` based on your use case. For information about the mandatory and optional parameters that you can configure during NetScaler Kubernetes Gateway Controller installation, see [Configurations](#).

```
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
```

The user name and password of a system user account on NetScaler MPX or NetScaler VPX are required for the Kubernetes Gateway. Ensure that NetScaler has a non-default system user account with the necessary privileges to allow the NetScaler Kubernetes Gateway Controller to configure NetScaler MPX or VPX. For instructions on creating a system user account on NetScaler, refer to the [Create a system user account for NetScaler Ingress Controller in NetScaler](#).

You can provide the user name and password directly or use Kubernetes secrets. To use Kubernetes secrets, create a secret for the user name and password with the following command:

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

NOTE:

`entityPrefix` and `gatewayControllerName` are mandatory and must be unique for every deployment of NetScaler Kubernetes Gateway Controller.

4. Deploy the NetScaler Kubernetes Gateway controller using the Helm charts and the `values.yaml` generated in previous step. For more information, see [NetScaler Kubernetes Gateway Controller deployment using Helm Chart](#).

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

Exposing an application by using HTTP Using Gateway and HTTPRoute CRDs

The following is an example of a GatewayClass, Gateway, and an HTTPRoute definition.

1. Create a GatewayClass resource to define the controller handling Gateway objects.

Sample GatewayClass

```
apiVersion: gateway.networking.k8s.io/v1  
  
kind: GatewayClass  
  
metadata:  
  name: example-gateway-class  
  
spec:  
  controllerName: citrix.com/nsgw-controller
```

2. Define a Gateway resource to expose services based on defined listeners.

```
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
```

NOTES:

- Verify that the gatewayClassName defined in the Gateway matches the GatewayClass resource that was created in the previous step.

3. Apply HTTPRoute or other route CRDs to route traffic to backend services.

```
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
```

NOTES:

- Ensure that the gateway name specified in `parentRefs` matches with the Gateway CRD established in the previous step.
- Currently, the HTTPRoute CRD only supports a single `parentRef`.

4. Verify the gatewayClass status. If the status shows `ACCEPTED: True`, then the controller has accepted the resource.

```
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>

Verify the Gateway resource status. If the status shows **PROGRAMMED : True**, then controller has successfully pushed the configuration to NetScaler.

```
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>
```

Exposing an application by using HTTPS/SSL Using Gateway and HTTPRoute CRDs

The following is an example of a GatewayClass, Gateway, and an HTTPRoute definition.

1. Create a GatewayClass resource to define the Gateway objects handled by the controller.

Sample GatewayClass

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

kind: GatewayClass

metadata:

  name: example-gateway-class

spec:
```

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

2. (Optional) Create a self-signed SSL certificate and a key to be used with the Gateway-Listener for TLS configuration. The following command is just a sample command to create a self-signed certificate and assumes that the host name of the application is `example.com`

```
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
```

NOTE:

If you already have an SSL certificate, you can create a Kubernetes secret using the same.

3. Create a Kubernetes Secret to store the SSL certificate and key (TLS pair).

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

4. Create the Gateway resource with listeners for HTTP (port 80) and HTTPS (port 443). The Gateway exposes the application on 4.4.4.4. Refer the SSL certificate Secret created in the preceding step within the HTTPS listener.

```
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. Apply the HTTPRoute resource to route incoming traffic to one or more backend services.

```
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
```

NOTE:

- The application is exposed on both port 80 (HTTP) and port 443 (HTTPS).
 - To expose the application on HTTPS only, exclude the HTTP listener configuration in the Gateway resource.
 - To redirect HTTP to HTTPS, use the `requestRedirect` HTTPRoute Filter.
- If your backend service expects HTTPS traffic from the Gateway, ensure the `appProtocol` is set to `https` on the corresponding port (for example, port 443) in your Service definition. Reference: Consult the Gateway API documentation for details on backend protocols.

Exposing a TCP application by using Gateway and TCPRoute CRDs

This section provides step-by-step instructions to expose a TCP-based application by using the Kubernetes Gateway API with NetScaler as the gateway infrastructure.

Step 1: Deploy a sample TCP application

Deploy a sample TCP application (for example, a Redis server) in your Kubernetes cluster:

```
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
```

Apply the configuration:

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

Step 2: Create a GatewayClass

Create a GatewayClass resource that references the NetScaler Gateway Controller:

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

kind: GatewayClass
```

```
metadata:

  name: netScaler-gateway-class

spec:

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

Apply the GatewayClass:

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

Step 3: Create a Gateway with a TCP listener

Create a Gateway resource with a TCP protocol listener. The `addresses` field specifies the VIP address on the NetScaler:

```
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
```

Apply the Gateway:

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

Step 4: Create a TCPRoute

Create a `TCPRoute` resource that attaches to the Gateway listener and routes traffic to the backend service:

```
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
```

Apply the `TCPRoute` :

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

Step 5: Verify the configuration

1. Check the Gateway status:

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

1. Check the `TCPRoute` status:

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

Verify that both `Accepted` and `Programmed` conditions show `status: "True"` .

1. Test the TCP connection:

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

Expected output: `PONG`

Exposing a UDP Application Using Gateway and UDPRoute CRDs

This section provides step-by-step instructions to expose a UDP-based application (such as a DNS server) using the Kubernetes Gateway API with NetScaler.

Step 1: Deploy a sample UDP application

Deploy a sample DNS server (CoreDNS) in your Kubernetes cluster:

```
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
```

Apply the configuration:

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

Step 2: Create a GatewayClass

If not already created, create a GatewayClass:

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

kind: GatewayClass

metadata:

  name: netscaler-gateway-class

spec:

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

Step 3: Create a Gateway with a UDP listener

Create a Gateway resource with a UDP protocol listener:

```
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
```

Apply the Gateway:

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

Step 4: Create a UDPRoute

Create a UDPRoute resource that attaches to the Gateway listener:

```
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
```

Apply the UDPRoute:

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

Step 5: Verify the configuration

1. Check the Gateway status:

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

1. Check the UDPRoute status:

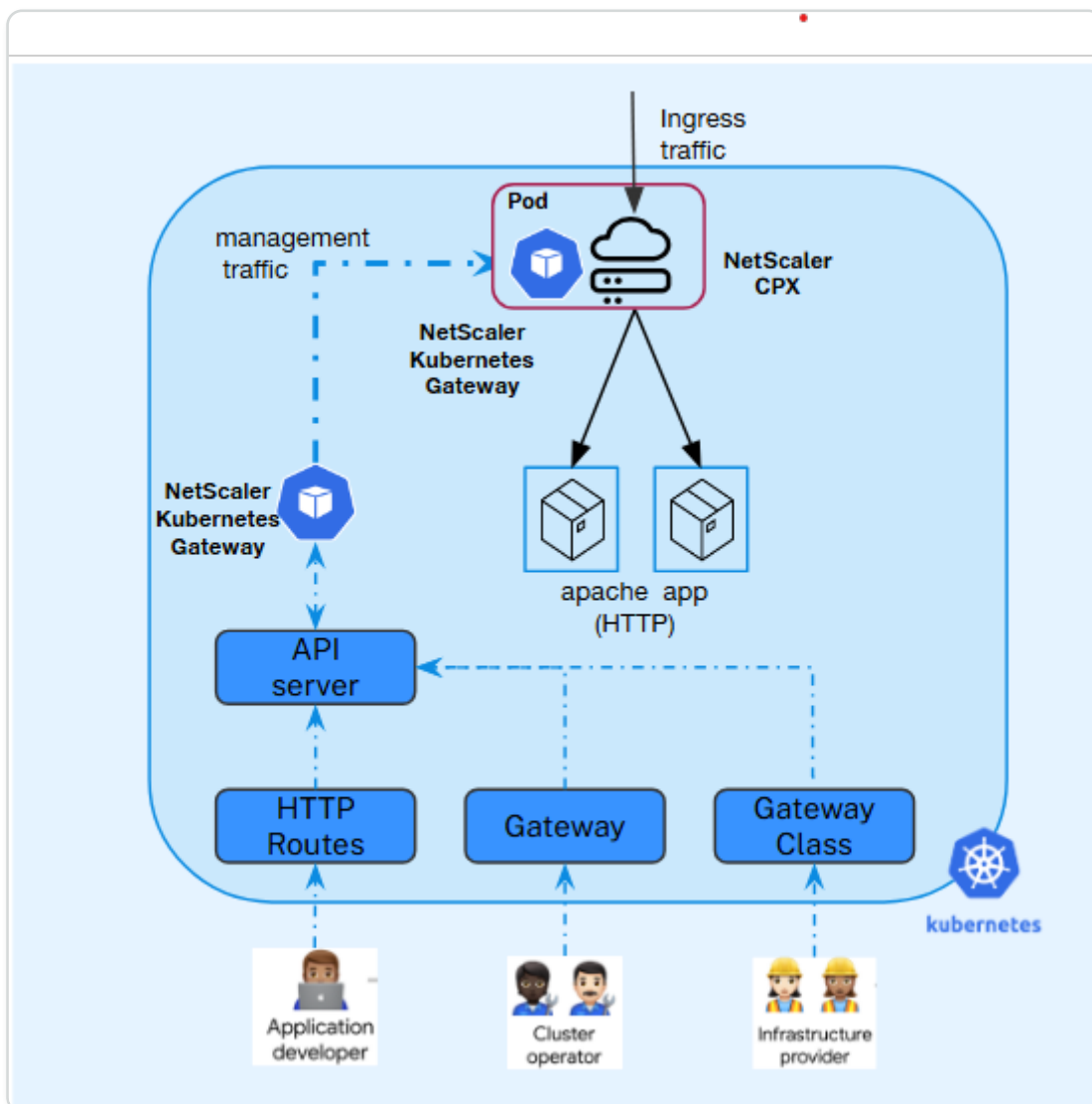
```
kubectl get udproute dns-route -o yaml
```

- ### 1. Test DNS resolution through the Gateway:

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

Deploy NetScaler® CPX Kubernetes Gateway Controller

The deployment process involves deploying the Gateway Controller, which monitors and manages Gateway CRDs that define traffic routing rules. The Gateway Controller then configures NetScaler CPX to direct external requests to the sample application.



1. Deploy a sample application to get started.

```
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
```

2. Install the Gateway API CRDs from the [official standard channel](#) if they are not already installed.

3. Generate `values.yaml` based on your use case. For information about the mandatory and optional parameters that you can configure during NetScaler Kubernetes Gateway Controller installation, see [Configurations](#).

```
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

```

NOTE:

`entityPrefix` and `gatewayControllerName` are mandatory and must be unique for every deployment of NetScaler CPX with Kubernetes Gateway Controller.

4. Deploy NetScaler CPX with Kubernetes Gateway controller using the Helm charts and the `values.yaml` generated in preceding step. For more information, see [NetScaler CPX with Kubernetes Gateway Controller deployment using Helm Chart](#).

```

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

```

Sample Gateway and HTTPRoute CRD

The following is an example of a GatewayClass, Gateway, and an HTTPRoute definition.

1. Create a GatewayClass resource to define the controller handling Gateway objects.

Sample GatewayClass

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

kind: GatewayClass

metadata:

  name: example-gateway-class

spec:

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

2. (Optional) Create a self-signed SSL certificate and a key to be used with the Gateway-Listener for TLS configuration.

The following command is just a sample command to create a self-signed certificate and assumes that the host name of the application is `example.com`.

```
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
```

NOTE:

If you already have an SSL certificate, you can create a Kubernetes secret using the same.

3. Create a Kubernetes Secret to store the SSL certificate and key (TLS pair).

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

4. Define a Gateway resource to expose services based on defined listeners.

```
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: ""
```

NOTES:

- Verify that the `gatewayClassName` defined in the Gateway matches the `GatewayClass` resource that is created in the preceding step.

5. Apply HTTPRoute or other route CRDs to route traffic to backend services.

```
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
```

NOTES:

- Ensure that the gateway name specified in `parentRefs` matches with the Gateway CRD established in the preceding step.
- Currently, the HTTPRoute CRD only supports a single `parentRef`.

6. Verify the gatewayClass status. If the status shows `ACCEPTED: True`, then the controller has accepted the resource.

```
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>

Verify the Gateway resource status. If the status shows **PROGRAMMED : True**, then controller has successfully pushed the configuration to NetScaler.

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

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

```
kubectl 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. For traffic, as service of CPX is deployed as `nodePort`, you can use `nodeIP:NodePort` to send the traffic.

```
kubectl get svc -o wide
```

NAME	EXTERNAL-IP	PORT(S)	TYPE	AGE	CLUSTER-IP	SELECTOR
cpx-gateway-controller-cpx-service		80:31375/TCP, 443:31994/TCP	NodePort	11m	10.104.175.74	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
```

Exposing a TCP application by using Gateway and TCPRoute CRDs

This section provides step-by-step instructions to expose a TCP-based application using the Kubernetes Gateway API with NetScaler.

Step 1: Deploy a sample TCP application

Deploy a sample TCP application (for example, a Redis server) in your Kubernetes cluster:

```
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
```

Apply the configuration:

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

Step 2: Create a GatewayClass

Create a GatewayClass resource that references NetScaler Gateway Controller:

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

kind: GatewayClass

metadata:

  name: netscaler-gateway-class

spec:
```

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

Apply the GatewayClass:

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

Step 3: Create a Gateway with a TCP listener

Create a Gateway resource with a TCP protocol listener. The `addresses` field specifies the VIP address on NetScaler:

```
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
```

Apply the Gateway:

```
kubect1 apply -f tcp-gateway.yaml
```

Step 4: Create a TCPRoute

Create a TCPRoute resource that attaches to the Gateway listener and routes traffic to the backend service:

```
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
```

Apply the TCPRoute:

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

Step 5: Verify the configuration

1. Check the Gateway status:

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

1. Check the TCPRoute status:

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

Verify that both `Accepted` and `Programmed` conditions show `status: "True"`.

1. Test the TCP connection:

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

Expected output: `PONG`

Exposing a UDP Application Using Gateway and UDPRoute CRDs

This section provides step-by-step instructions to expose a UDP-based application (such as a DNS server) using the Kubernetes Gateway API with NetScaler.

Step 1: Deploy a sample UDP application

Deploy a sample DNS server (CoreDNS) in your Kubernetes cluster:

```
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
```

Apply the configuration:

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

Step 2: Create a GatewayClass

If not already created, create a GatewayClass:

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

kind: GatewayClass

metadata:

  name: netscaler-gateway-class

spec:

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

Step 3: Create a Gateway with a UDP listener

Create a Gateway resource with a UDP protocol listener:

```
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
```

Apply the Gateway:

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

Step 4: Create a UDPRoute

Create a UDPRoute resource that attaches to the Gateway listener:

```
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
```

Apply the UDPRoute:

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

Step 5: Verify the configuration

1. Check the Gateway status:

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

1. Check the UDPRoute status:

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

1. Test DNS resolution through the Gateway:

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

NetScaler Operator

The following topics are available under NetScaler Operator:

- [NetScaler Operator release notes](#)
- [Deploy NetScaler Kubernetes Gateway Controller in OpenShift using NetScaler Operator](#)
- [Deploy NetScaler Kubernetes Gateway Controller with CPX in OpenShift using NetScaler Operator](#)

NetScaler Operator release notes

Release notes describe how the software has changed in a particular build, and the issues known to exist in the build. The release notes document includes all or some of the following sections:

- What's New: The enhancements and other changes released in the build.
- Fixed Issues: The issues that are fixed in the build.
- Known Issues: The issues that exist in the build.

The following is the list of NetScaler® Operator releases:

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

NetScaler® Operator release notes

Version 3.6.0

What's new

NOTE:

During reconciliation of old config to the new config, you might observe downtime for Citrix CRDs configuration due to config change.

SUPPORT FOR NETSCALER IPAM CONTROLLER

NetScaler Kubernetes Gateway Controller supports IP address allocation for Kubernetes Gateway resources using our own NetScaler IPAM controller. When IPAM is enabled, Gateways that do not have a static IP address configured in `spec.addresses` automatically receive a Virtual IP (VIP) from the IPAM controller. This eliminates the need to manually assign and track IP addresses for each Gateway.

For more information, see [<https://docs.netscaler.com/en-us/netscaler-kubernetes-gateway-controller/ipam-controller>](https://docs.netscaler.com/en-us/netscaler-kubernetes-gateway-controller/ipam-controller).

EXPOSING A TCP/UDP APPLICATION BY USING GATEWAY AND TCPROUTE/UDPROUTE CRDS

NetScaler Kubernetes Gateway Controller supports exposing a TCP/UDP-based application by using the Kubernetes Gateway API with NetScaler.

For more information, see [Deploy NetScaler Kubernetes Gateway Controller](#) and [Deploy NetScaler CPX Kubernetes Gateway Controller](#).

NETSCALER CPX CONFIGURATION PERSISTENCE DURING UPGRADE

NetScaler CPX configuration persistence significantly reduces the time required for new pods to go live by saving the running configuration to a Kubernetes ConfigMap during pod shutdown. This enhancement ensures that when a replacement pod starts, whether due to a Helm upgrade, rolling update, or any other replacement, the configuration is automatically restored. Because the new pod initializes with most settings pre-loaded, the delay in becoming ready to handle traffic is minimized.

Previously, the running configuration accumulated through ingress and service resources was lost whenever a NetScaler CPX pod restarted. This required the NetScaler Ingress Controller (NSIC) to re-push every resource from scratch through the NITRO API. In large-scale ingress deployments, this reconciliation process caused a lengthy delay before the NetScaler CPX was fully ready to manage traffic.

For more information, see [<https://docs.netscaler.com/en-us/netscaler-k8s-ingress-controller/accelerate-cpx-upgrade-process>](https://docs.netscaler.com/en-us/netscaler-k8s-ingress-controller/accelerate-cpx-upgrade-process).

CONFIGURE OWASP TOP 10 PROTECTION POLICIES WITH THE OWASP CRD

NetScaler Ingress Controller now supports configuring OWASP Top 10 Protection Policies with the OWASP CRD. The OWASP Top 10:2025 represents the most critical security risks to web applications. Using the OWASP CRD (owasppolicy), you can configure comprehensive WAF and

Bot management protections aligned with the OWASP Top 10 categories in a single Kubernetes custom resource.

For more information, see <https://docs.netscaler.com/en-us/netscaler-k8s-ingress-controller/crds/owasp-top-10-protection-policies>.

BIND DNS INTEGRATION WITH IPAM CONTROLLER AND INGRESS CONTROLLER

BIND 9 is a complete implementation of the DNS protocol. BIND 9 can be configured as an authoritative name server, a resolver, and, on supported hosts, a stub resolver. For more information, see Bind 9.

NetScaler Ingress Controller supports assigning IPs to Ingresses and Services of type LoadBalancer by using the ipam-range annotation. The IPAM controller requires the VIP CustomResourceDefinition (CRD) provided by NetScaler. The VIP CRD is used for internal communication between NetScaler Ingress Controller and the IPAM controller. From this release there is a change in the VIP CRD. The VIP CRD now supports the hostname field which is associated with the VIP.

For more information, see <https://docs.netscaler.com/en-us/netscaler-k8s-ingress-controller/bind-dns-integration>.

OPTIMIZED NETSCALER RESOURCE CONFIGURATION DURING NETSCALER INGRESS CONTROLLER REBOOT

NetScaler Ingress Controller (NSIC) now supports an optimized startup sequence to minimize operational lag. This feature accelerates the configuration of NetScaler resources during reboots or upgrades, and enables the controller to reach a running state faster.

OVN-CNI support for OpenShift using NetScaler node controller The node controller and router pod support OVN-Kubernetes, the default CNI in OpenShift 4.x clusters. VXLAN overlay network is established between the OpenShift nodes and the Citrix ADC using the ovn-k8s-mp0 interface.

For more information, see <https://docs.netscaler.com/en-us/netscaler-k8s-ingress-controller/network/ovn-kubernetes-cni-support>.

SUPPORTING `INGRESSHOSTNAME`, `MULTIPLEIPRESPONSE` AND `EMPTYDOWNRESPONSE` IN THE GTP CRD

- `ingressHostName` — A new optional field under `globaltrafficpolicy.spec.hosts` lets you match an Ingress by its rule hostname independently of the GTP's Host (DNS) field, so a single GSLB domain can be backed by an Ingress that publishes a different hostname.

- `multipleIPResponse` (MIR) and `emptyDownResponse` (EDR) — New optional policy fields `multipleIPResponse` (MIR) and `emptyDownResponse` (EDR), that propagate to the corresponding -MIR/-EDR parameters on the auto-created GSLB vserver, enabling multi-IP DNS responses and empty-down behaviour directly from the CRD.

For more information, see [<https://docs.netscaler.com/en-us/netscaler-k8s-ingress-controller/gslb/gslb#gtp-crd>](https://docs.netscaler.com/en-us/netscaler-k8s-ingress-controller/gslb/gslb#gtp-crd).

Entity name change for CRDs

When a NetScaler Custom Resource Definition (CRD) instance is created, the NetScaler Ingress Controller generates multiple NetScaler entities associated with that CRD instance. The NetScaler Ingress Controller maintains unique names for each entity to preserve its association with the CRD instance. Since entity naming is based directly on CRD names, some NetScaler entity names exceeded the maximum character limit.

Starting with NetScaler Ingress Controller 4.0.16, the naming convention is optimized by using the following approach to generate shorter entity names during CRD creation:

- Hashed naming: A portion of the entity name is hashed to reduce the overall length.
- Preserved information: The necessary Kubernetes related metadata is retained in the entity's comment field, if the entity comment is supported by NetScaler.
- Improved compatibility: Names comply with NetScaler character restrictions while maintaining full traceability. For more information, see [<https://docs.netscaler.com/en-us/netscaler-k8s-ingress-controller/entity-name-change-for-crds>](https://docs.netscaler.com/en-us/netscaler-k8s-ingress-controller/entity-name-change-for-crds).

GSLB CONTROLLER IMPROVEMENTS

GSLB controller is enhanced to include the following improvements:

- Added support for GSE auto-creation for all Ingress resource types, including TCP, UDP, SSL, and others.
- `GslbConfigSyncMonitor` is now enabled by default on the master GSLB node to improve GSLB site monitoring efficiency. For more information, see [<https://docs.netscaler.com/en-us/netscaler-k8s-ingress-controller/gslb/gslb>](https://docs.netscaler.com/en-us/netscaler-k8s-ingress-controller/gslb/gslb).

SERVICETYPELB: EVENT MODIFICATION FOR SMART ANNOTATIONS

Starting with NetScaler Ingress Controller release 4.0.16, if you modify any of the following annotations in `ServiceTypeLB`, the NetScaler Ingress Controller modifies the configuration rather than deleting and recreating it in NetScaler.

```
"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-'
```

SSL PASSTHROUGH SUPPORT FOR NETSCALER MULTI-CLUSTER INGRESS DEPLOYMENT

SSL passthrough feature allows you to pass incoming secure sockets layer (SSL) requests directly to a server for decryption, rather than decrypting the request using a load balancer. SSL passthrough is widely used for web application security, and it uses the TCP mode to pass encrypted data to servers.

Starting with NetScaler Ingress Controller 4.0.16, the SSL passthrough feature is supported for NetScaler multi-cluster ingress deployment. For more information, see <https://docs.netscaler.com/en-us/netscaler-k8s-ingress-controller/configure/ssl-passthrough-multicluster>.

Fixed issues - NetScaler Kubernetes Gateway Controller

- WebSocket is disabled by default. You can now use the `HTTPProfile` CRD for this use case.
- Issues observed in the `HTTPProfile` CRD controller when `GatewayClass` is provided in the profile. The `GatewayClass` name provided in the profile might cause no attribute 'ref_manager' error.
- Host names are not getting the correct precedence over wildcards when they share the same parent domain. As a fix for this issue, routing priority for overlapping host names on the same content switching virtual server is enhanced.
- `RequestHeaderModifier.set` works only if the header already exists.
- `URLRewrite.replacePrefixMatch` produces double slashes, and therefore path normalization is not handled correctly.
- Any two HTTP Routes whose names are the same up to 20 characters create one load balancing virtual server instead of two. Conflicts are observed in load balancing virtual server naming.
- Load balancing virtual server service group binding is lost after controller restart or service event is received after the HTTP route.
- During GET calls to the Kubernetes API, the `429 errors` are observed.

Fixed issues - NetScaler Ingress Controller

- Custom monitor creation fails if more than one service is mentioned in the ingress, and the monitor is created only for one of them.
- During CRUD operations, multiple bot profile parameters in the bot CRD are missed.
- Incorrect exception handling is observed while logging errors.

- When a NetScaler Ingress Controller pod restarts, the controller fails to delete the configuration for ingress(es), which are deleted when the controller is down.
- Any two HTTP Routes whose names are the same up to 20 characters create one load balancing virtual server instead of two.
- NetScaler Ingress Controller tries to bind service group members to a non-existent service group even before IPAM provides IP address for the content switching virtual server.
- 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 could receive HTTP 429 (Too Many Requests) errors from the Kubernetes API server under load, causing some watch events to be missed. The Helm chart now ships a dedicated FlowSchema and PriorityLevelConfiguration (API Priority and Fairness) so NSIC's API traffic is given a reserved concurrency share and is no longer rate-limited alongside other clients.

NetScaler® Operator release notes

Version 3.5.0

What's new

FIXED ISSUES

- Resolved incorrect default profile format for SSL, HTTP, and TCP in NetScaler Ingress Controller configuration.
- Corrected Infoblox grid host settings in NetScaler IPAM Controller.

NetScaler® Operator release notes

Version 3.4.1

What's new

LAS LICENSE SUPPORT

You can now use LAS licensing method for NetScaler CPX by using NetScaler CPX with Kubernetes Gateway Controller and NetScaler CPX with Ingress Controller by setting the following parameters:

```
spec:

  admSettings:

    admIP: ''

    bandwidth: 1000

    licenseEdition: PLATINUM

    licensingMode: las

    loginSecret: ''
```

SUPPORT FOR NETSCALER INGRESS CONTROLLER/GSLB CONTROLLER VERSION 3.4.4

NetScaler Operator version 3.4.1 supports [NetScaler Ingress Controller/GSLB Controller version 3.4.4](#).

SUPPORT FOR NETSCALER KUBERNETES GATEWAY CONTROLLER VERSION 1.4.0

NetScaler Operator version 3.4.1 supports [NetScaler Kubernetes Gateway Controller version 1.4.0](#).

NetScaler® Operator release notes

Version 3.3.0

What's new

SUPPORT FOR NETSCALER KUBERNETES GATEWAY CONTROLLER

You can now deploy NetScaler Kubernetes Gateway Controller by using NetScaler Operator. For more information, see [Deploy NetScaler Kubernetes Gateway Controller in OpenShift using NetScaler Operator](#).

SUPPORT FOR NETSCALER CPX WITH NETSCALER KUBERNETES GATEWAY CONTROLLER

You can now deploy NetScaler CPX with NetScaler Kubernetes Gateway Controller by using NetScaler Operator. For more information, see [Deploy NetScaler Kubernetes Gateway Controller with CPX in OpenShift using NetScaler Operator](#).

Deploy NetScaler Kubernetes Gateway Controller in OpenShift using NetScaler Operator

In this setup, NetScaler Kubernetes Gateway Controller configures NetScaler MPX or NetScaler VPX or NetScaler BLX residing outside the OpenShift cluster.

Prerequisites

- [Red Hat OpenShift Cluster](#) (version 4.1 or later).
- Deploy NetScaler Operator. For information on how to deploy NetScaler Operator, see [Deploy NetScaler Operator](#).
- Identify the IP address that NetScaler K8s Gateway Controller must communicate with NetScaler. This IP address might be any one of the following IP addresses depending on the type of NetScaler deployment:
 - NSIP (for standalone appliances) - The management IP address of a standalone NetScaler appliance. For more information, see [IP Addressing in NetScaler](#).
 - SNIP (for appliances in High Availability mode) - The subnet IP address. For more information, see [IP Addressing in NetScaler](#).
 - CLIP (for appliances in Cluster mode) - The cluster management IP (CLIP) address for a clustered NetScaler deployment. For more information, see [IP addressing for a cluster](#).

- The user name and password of NetScaler VPX or NetScaler MPX or NetScaler BLX used as the ingress device. NetScaler must have a system user account (non-default) with certain privileges so that the NetScaler K8s Gateway Controller can configure NetScaler VPX or NetScaler MPX. For instructions to create a system user account on NetScaler, see [Create system user account for NetScaler Ingress Controller in NetScaler](#).
- You can directly pass the user name and password as environment variables to the controller, or use OpenShift secrets (recommended). To create a secret for the user name and password using the following command, modify the `<username>` and `<password>` to required values:

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

- To export to any service monitor, install the service monitor CRD by using the following command:

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

- Install the Gateway API CRDs from the [official standard channel](#) if they are not already installed.

Deploy NetScaler Kubernetes Gateway Controller using OpenShift Operator

Perform the following steps:

1. Log in to the OpenShift 4.x Cluster console.
2. Deploy a CNNwebsite application using the console.
 - a. Navigate to **Workloads > Deployments > Create Deployment** and use the following YAML file to create the deployment.

```
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

Project: default ▾

Create Deployment

Configure via: ☐ Form view ☒ YAML view

☒ Sidebar

```
1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: cnn-website
5    labels:
6      name: cnn-website
7      app: cnn-website
8  spec:
9    selector:
10     matchLabels:
11       app: cnn-website
12    replicas: 2
13    template:
14      metadata:
15        labels:
16          name: cnn-website
17          app: cnn-website
18      spec:
19        containers:
20          - name: cnn-website
21            image: quay.io/sample-apps/cnn-website:v1.0.0
22            ports:
23              - name: http-80
24                containerPort: 80
25              - name: https-443
26                containerPort: 443
```

- b. Navigate to **Workloads > Pods** section and ensure that the CNNwebsite application pods are up and running.

Project: cnnwebsite ▾

Pods

Filter ▾ Name ▾ Search by name... 

Name ↑	Status ↑	Ready ↑	Restarts ↑	Owner ↑	Memory ↑	CPU ↑
 cnn-website-85d45f9b5d-gj64b	 Running	1/1	0	 cnn-website-85d45f9b5d	41.4 MiB	0.000 cores
 cnn-website-85d45f9b5d-llq44	 Running	1/1	0	 cnn-website-85d45f9b5d	41.4 MiB	0.000 cores

3. Create a service for the CNNwebsite application. Navigate to **Networking > Services > Create Service** and use the following YAML file.

```
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. Create Self-Signed Certificate, Gateway Class, Gateway, and HTTP Route for CNNwebsite application.

- a. Create a self-signed SSL certificate and a key to be used with the Gateway-Listener for TLS configuration. The following command is just a sample command to create a self-signed certificate and assumes that the host name of the application is example.com.

```
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. Deploy Gateway Class, Gateway, and HTTP Route.

```
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
```

Import YAML

Drag and drop YAML or JSON files into the editor, or manually enter files and use `---` to separate each definition.

☒ Tooltips

```

1  apiVersion: gateway.networking.k8s.io/v1
2  kind: GatewayClass
3  metadata:
4    name: example-gateway-class
5  spec:
6    controllerName: citrix.com/nsgw-controller
7  ---
8  apiVersion: gateway.networking.k8s.io/v1
9  kind: Gateway
10 metadata:
11   name: example-gateway
12 spec:
13   gatewayClassName: example-gateway-class
14   listeners:
15   - name: http
16     protocol: HTTP
17     port: 80
18   - name: https
19     protocol: HTTPS
20     port: 443
21     tls:
22       mode: Terminate
23       certificateRefs:
24       - kind: Secret
25         name: appl-secret
26   addresses:
27   - type: IPAddress
28     value: 4.4.4.4
29   ---
30 apiVersion: gateway.networking.k8s.io/v1
31 kind: HTTPRoute
32 metadata:
33   name: example-route
34 spec:
35   parentRefs:
36   - name: example-gateway
37   hostnames:
38   - "example.com"
39   rules:
40   - matches:
41     - path:
42       type: PathPrefix
43       value: /
44   backendRefs:
45   - name: cnp-website

```

5. Navigate to **Operators > Installed Operators** and click **NetScaler Operator**.

Project: default ▾

Installed Operators

Installed Operators are represented by ClusterServiceVersions within this Namespace. For more information, see the [Understanding Operators documentation](#) or create an Operator and ClusterServiceVersion using the [Operator SDK](#).

Name ▾ Search by name... 

Name	Managed Namespaces	Status	Last updated	Provided APIs
 NetScaler Operator 3.3.0 provided by NetScaler	NS default The operator is running in openshift-operators but is managing this namespace	 Succeeded Up to date	 26 Nov 2025, 01:46	Netscaler K8s Gateway Controller Netscaler CPX With K8s Gateway Controller Netscaler Ingress Controller Netscaler CPX With Ingress Controller View 4 more...

6. Click the **NetScaler K8s Gateway Controller** tab and select **Create NetScalerK8sGatewayController**.

[Installed Operators](#) > Operator details

 **NetScaler Operator**
3.3.0 provided by NetScaler

< Details | YAML | Subscription | Events | All instances | **Netscaler K8s Gateway Controller** | Netscaler CPX With K8s Gateway Controller

Provided APIs

NKGC Netscaler K8s Gateway Controller

To install NetScaler Kubernetes Gateway Controller for NetScaler VPX/MPX/SDX.

 [Create instance](#)

NCWK Netscaler CPX With K8s Gateway Controller

To Install NetScaler CPX with Kubernetes Gateway Controller.

 [Create instance](#)

NIC Netscaler Ingress Controller

To install NetScaler Ingress Controller for NetScaler VPX/MPX/SDX.

 [Create instance](#)

NCWI Netscaler CPX With Ingress Controller

To install NetScaler CPX with Ingress Controller.

 [Create instance](#)

NGC Netscaler GSLB Controller

To install NetScaler GSLB Controller for NetScaler VPX/MPX/SDX.

 [Create instance](#)

NIC Netscaler IPAM Controller

To install NetScaler IPAM Controller for NetScaler CPX/VPX/MPX/SDX.

 [Create instance](#)

NOE Netscaler Observability Exporter

To install NetScaler Observability Exporter for NetScaler.

 [Create instance](#)

NCC Netscaler Cloud Controller

To install NetScaler Cloud Controller for NetScaler VPX/MPX/SDX with Azure Redhat OpenShift in Azure.

 [Create instance](#)

The NetScaler K8s Gateway Controller YAML definition is displayed. Optionally, you can select Form view button and update the parameters in a form.

Create NetscalerK8sGatewayController

Create by completing the form. Default values may be provided by the Operator authors.

Configure via: ☐ Form view ☒ YAML view



Tooltips



Sidebar



Shortcuts

```

1 kind: NetscalerK8sGatewayController
2 apiVersion: netscaler.com/v1
3 metadata:
4   labels:
5     app.kubernetes.io/managed-by: kustomize
6     app.kubernetes.io/name: netscaler-operator
7   name: netscalerk8sgatewaycontroller-sample
8   namespace: openshift-operators
9 spec:
10  acceptLicense: 'No'
11  affinity: {}
12  exporter:
13    extraVolumeMounts: []
14    image: 'quay.io/netscaler/netscaler-adc-metrics-exporter@sha256:558ecc754698f79769197b7dc66c
15    ports:
16      containerPort: 8888
17    pullPolicy: IfNotPresent
18    required: false
19    resources: {}
20    serviceMonitorExtraLabels: {}
21  extraVolumes: []
22  gatewayController:
23    certBundle: false
24    clusterName: ''
25    disableAPIServerCertVerify: false
26    enableLivenessProbe: true
27    enableReadinessProbe: true
28    entityPrefix: ''
29    extraVolumeMounts: []
30    gatewayControllerName: ''
31    image: 'quay.io/netscaler/netscaler-k8s-ingress-controller@sha256:fffa0c8790a9e1f299320ae905
32    jsonLog: false
33    kubernetesURL: ''
34    livenessProbe:
35      initialDelaySeconds: 30
36      periodSeconds: 60
37    logLevel: INFO
38    nodeWatch: false
39    nsValidateCert: {}
40    nsncPbr: false
41    openshift: true
42    optimizeEndpointBinding: false
43    podIPsforServiceGroupMembers: false

```

Create

Cancel



Download

7. Refer the following table that lists the mandatory and optional parameters that you can configure during installation.

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
```

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
acceptLicense	Mandatory	no	Set this value to Yes to accept the NetScaler K8s Gateway Controller EULA.
affinity	Optional	N/A	Affinity labels for pod assignment.
extraVolumes	Optional	N/A	Additional VolumeMounts to be mounted.
imagePullSecrets	Optional	N/A	List of Kubernetes secrets to be used for pulling images from a private Docker registry or repository.
nodeSelector	Optional	N/A	The node label key:value pair to be used for nodeSelector option

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
			in NetScaler K8s Gateway Controller deployment. For more information on how to create this secret, see Pull an Image from a Private Registry .
podAnnotations	Optional	N/A	Map of annotations to add to the pods.
tolerations	Optional	N/A	The tolerations for the NetScaler K8s Gateway Controller deployment.

Exporter

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
extraVolumeMounts	Optional	N/A	Additional VolumeMounts to be mounted in the exporter container.
image	Optional	<code>quay.io/netscaler/netscaler-adc-metrics-exporter@sha<sha value of the latest release></code>	The metrics exporter image hosted on Quay.io.
required	Optional	false	Set to true to run the Exporter for NetScaler Stats along with NetScaler K8s

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
			Gateway Controller to pull metrics for NetScaler MPX/VPX/BLX.
pullPolicy	Optional	IfNotPresent	The exporter image pull policy.
ports.containerPort	Optional	8888	The exporter container port.
resources	Optional	N/A	CPU/memory resources for a metrics exporter container.
serviceMonitorExtra Labels	Optional	N/A	Extra labels for a service monitor when exporter is enabled.

Gateway Controller

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
clusterName	Optional	N/A	A unique identifier of the Kubernetes cluster on which NetScaler K8s Gateway Controller is deployed.
disableAPIServerCertificateVerify	Optional	false	Set this parameter to True for disabling API Server certificate verification.

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
enableLivenessProbe	Optional	true	Set to false to disable liveness probe settings for NetScaler K8s Gateway Controller.
enableReadinessProbe	Optional	true	Set to false to disable readiness probe settings for NetScaler K8s Gateway Controller.
entityPrefix	Optional	N/A	The prefix for the resources on NetScaler MPX or NetScaler VPX.
gatewayControllerName	Mandatory	N/A	Name of Gateway Controller.
image	Mandatory	<code>quay.io/netscaler/netscaler-k8s-ingress-controller@sha256:latest</code>	The NetScaler K8s Gateway Controller image hosted on Quay.io.
extraVolumeMounts	Optional	N/A	Additional VolumeMounts to be mounted in the NetScaler K8s Gateway Controller container.
jsonLog	Optional	false	Set this argument to true if log messages are required in the JSON format.

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
kubernetesURL	Optional	N/A	The kube-apiserver URL that NetScaler K8s Gateway Controller uses to register events. If not specified, the controller uses the internal kube-apiserver IP address .
livenessProbe	Optional	N/A	Liveness probe settings for NetScaler K8s Gateway Controller.
logLevel	Optional	INFO	The log level to control the logs generated by NetScaler K8s Gateway Controller. Supported levels: CRITICAL, ERROR, WARNING, INFO, DEBUG, TRACE, and NONE.
nodeWatch	Optional	false	Set to true to automatically configure network route from NetScaler MPX/VPX to the pods in the Kubernetes cluster. For more information, see Automatically configure route on the NetScaler instance .

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
nsncPbr	Optional	false	Set to true to inform the controller that NetScaler Node Controller is configuring Policy Based Routes (PBR). For more information, see NSNC-PBR-SUPPORT .
OpenShift	Optional	true	Set to false if the OpenShift environment is not being used.
optimizeEndpointBinding	Optional	false	Set to true to enable binding of back-end endpoints to a service group in a single API call. Recommended when many endpoints per application exist. Applicable for NetScaler >=13.0–45.7.
podIPsforServiceGroupMembers	Optional	false	If true, adds pod IP and pod port as service group members for LoadBalancer/NodePort services instead of node IP and node port. Requires a route between NetScaler and cluster pods

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
			(nodeWatch or NSNC).
pullPolicy	Mandatory	IfNotPresent	The NetScaler K8s Gateway Controller image pull policy.
readinessProbe	Optional	N/A	Readiness probe settings for NetScaler K8s Gateway Controller.
resources	Optional	N/A	CPU/memory resources for the NetScaler K8s Gateway Controller container.

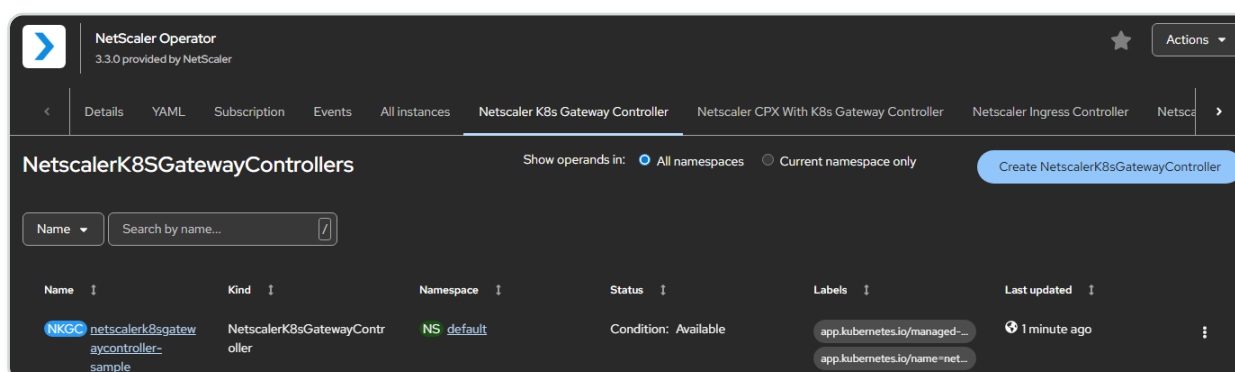
NetScaler

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
adcCredentialSecret	Mandatory	NA	Secret required for NetScaler K8s Gateway Controller to connect to NetScaler.
nitroReadTimeout	Optional	20	The NITRO read timeout in seconds.
nsIP	Mandatory	N/A	NetScaler IP address.
nsPort	Optional	443	The port used by NetScaler K8s Gateway Controller to communicate with NetScaler. You

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
			can use port 80 for HTTP.
nsProtocol	Optional	N/A	The protocol used by NetScaler K8s Gateway Controller to communicate with NetScaler. You can use HTTP on port 80.
nsSNIPS	Optional	N/A	List of subnet IP addresses on the NetScaler used to create PBR routes instead of static routes (PBR support).
nsVIP	Optional	N/A	The virtual IP address on NetScaler.

8. After updating the values for the required parameters, click **Create**.

Ensure that NetScaler K8s Gateway Controller is successfully deployed and initialized.



9. Navigate to **Workloads > Pods** section and verify whether the NetScaler Kubernetes Gateway Controller pod is up and running.

Project: default ▾

Pods ★ Create Pod

Filter ▾ Name ▾ Search by name...

Name ↑	Status ↑	Ready ↑	Restarts ↑	Owner ↑	Memory ↑	CPU ↑	Created ↑
netscalercpxwithk8sgatewaycontroller-sample-556656745d-rpjbq	Running	2/2	0	RS netscalercpxwithk8sgatewaycontroller-sample-556656745d	746.9 MiB	0.068 cores	1 minute ago
netscalerk8sgatewaycontroller-sample-6fbfc8555c-8l6hl	Running	1/1	0	RS netscalerk8sgatewaycontroller-sample-6fbfc8555c	206.5 MiB	0.002 cores	6 minutes ago

10. Verify the deployment by sending traffic:

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

NOTE:

Ensure that the pod network in OpenShift cluster is reachable from NetScaler VPX or NetScaler MPX or NetScaler BLX if you are using a service of type ClusterIP for your application. To configure static route automatically using NetScaler K8s Gateway Controller, see [Configure static route](#).

Deploy NetScaler Kubernetes Gateway Controller with CPX in OpenShift using NetScaler Operator

In this setup, NetScaler Kubernetes Gateway Controller configures NetScaler CPX deployed in the OpenShift cluster.

Prerequisites

- Access to [Red Hat OpenShift Cluster](#) (version 4.1 or later).
- Deploy NetScaler Operator. For information on how to deploy NetScaler Operator, see [Deploy NetScaler Operator](#).
- To export to any service monitor, install the service monitor CRD by using the following command:

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

- Install [Prometheus Operator](#) if you want to view the metrics of the NetScaler CPX collected through the [direct Prometheus export](#).
- Install the Gateway API CRDs from the [official standard channel](#) if they are not already installed.

Deploy NetScaler Kubernetes Gateway Controller as a sidecar with NetScaler CPX

Perform the following steps:

1. Log in to the OpenShift 4.x Cluster console.
2. Deploy a CNNwebsite application using the console.
 - a. Navigate to **Workloads > Deployments > Create Deployment** and use the following YAML file to create the deployment.

```
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
```

Project: default ▾

Create Deployment

Configure via: ☐ Form view ☒ YAML view

☒ Sidebar

```

1  apiVersion: apps/v1
2  kind: Deployment
3  metadata:
4    name: cnn-website
5    labels:
6      name: cnn-website
7      app: cnn-website
8  spec:
9    selector:
10     matchLabels:
11       app: cnn-website
12    replicas: 2
13    template:
14      metadata:
15        labels:
16          name: cnn-website
17          app: cnn-website
18      spec:
19        containers:
20          - name: cnn-website
21            image: quay.io/sample-apps/cnn-website:v1.0.0
22            ports:
23              - name: http-80
24                containerPort: 80
25              - name: https-443
26                containerPort: 443

```

- b. Navigate to **Workloads > Pods** section and ensure that the CNNwebsite application pods are up and running.

Project: cnnwebsite ▾

Pods

Filter ▾ Name ▾ Search by name...  

Name ↑	Status ↑	Ready ↑	Restarts ↑	Owner ↑	Memory ↑	CPU ↑
 cnn-website-85d45f9b5d-gj64b	 Running	1/1	0	 cnn-website-85d45f9b5d	41.4 MiB	0.000 cores
 cnn-website-85d45f9b5d-llq44	 Running	1/1	0	 cnn-website-85d45f9b5d	41.4 MiB	0.000 cores

3. Create a service for the CNNwebsite application. Navigate to **Networking > Services > Create Service** and use the following YAML file.

```
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. Create Self-Signed Certificate, Gateway Class, Gateway, and HTTP Route for CNNwebsite application.

- a. Create a self-signed SSL certificate and a key to be used with the Gateway-Listener for TLS configuration. The following command is just a sample command to create a self-signed certificate and assumes that the host name of the application is example.com

```
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. Deploy Gateway Class, Gateway, and HTTP Route.

```
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. Navigate to **Operators > Installed Operators** and click **NetScaler Operator**.

Project: default ▾

Installed Operators

Installed Operators are represented by ClusterServiceVersions within this Namespace. For more information, see the [Understanding Operators documentation](#) or create an Operator and ClusterServiceVersion using the [Operator SDK](#).

Name ▾ Search by name... 🔍

Name	Managed Namespaces	Status	Last updated	Provided APIs
 NetScaler Operator 3.3.0 provided by NetScaler	NS default The operator is running in openshift-operators but is managing this namespace	✔ Succeeded Up to date	🕒 26 Nov 2025, 01:46	Netscaler K8s Gateway Controller Netscaler CPX With K8s Gateway Controller Netscaler Ingress Controller Netscaler CPX With Ingress Controller View 4 more...

6. Click the **NetScaler K8s Gateway Controller** tab and select **Create NetScalerCPXWithK8sGatewayController**.

Installed Operators > Operator details

NetScaler Operator
3.3.0 provided by NetScaler

< Details YAML Subscription Events All instances Netscaler K8s Gateway Controller Netscaler CPX With K8s Gateway Controller

Provided APIs

NKGC Netscaler K8s Gateway Controller To install NetScaler Kubernetes Gateway Controller for NetScaler VPX/MPX/SDX. Create instance	NCWK Netscaler CPX With K8s Gateway Controller To Install NetScaler CPX with Kubernetes Gateway Controller. Create instance	NIC Netscaler Ingress Controller To install NetScaler Ingress Controller for NetScaler VPX/MPX/SDX. Create instance
NCWI Netscaler CPX With Ingress Controller To install NetScaler CPX with Ingress Controller. Create instance	NGC Netscaler GSLB Controller To install NetScaler GSLB Controller for NetScaler VPX/MPX/SDX. Create instance	NIC Netscaler IPAM Controller To install NetScaler IPAM Controller for NetScaler CPX/VPX/MPX/SDX. Create instance
NOE Netscaler Observability Exporter To install NetScaler Observability Exporter for NetScaler. Create instance	NCC Netscaler Cloud Controller To install NetScaler Cloud Controller for NetScaler VPX/MPX/SDX with Azure Redhat OpenShift in Azure. Create instance	

The NetScaler CPX with Kubernetes Gateway Controller YAML definition is displayed. Optionally, you can select Form view button and update the parameters in a form.

Create NetscalerCpxWithIngressController

Create by completing the form. Default values may be provided by the Operator authors.

Configure via: ☐ Form view ☒ YAML view

☒ Tooltips ☒ Sidebar

```

1  kind: NetscalerCpxWithIngressController
2  apiVersion: netscaler.com/v1
3  metadata:
4    labels:
5      app.kubernetes.io/managed-by: kustomize
6      app.kubernetes.io/name: netscaler-operator
7      name: netscalercpxwithingresscontroller-sample
8      namespace: cnnwebsite
9  spec:
10   acceptLicense: 'No'
11   admSettings:
12     admIP: ''
13     bandwidth: 1000
14     bandwidthLicense: false
15     cpxCores: 1
16     licenseEdition: PLATINUM
17     licenseServerIP: ''
18     licenseServerPort: 27000
19     licensingMode: pool
20     loginSecret: ''
21     platform: false
22     vCPULicense: false
23   affinity: {}
24   analyticsConfig:
25     distributedTracing:
26       enable: false
27       samplingRate: 100
28     endpoint:
29       metrics:
30         service: ''
31       transactions:
32         service: ''
33     required: false
34     resources: {}
35     timeseries:
36       auditLogs:
37         enable: false
38       events:
39         enable: false
40       metrics:
41         enable: false
42       enableNativeScrape: false

```

7. Refer the following table that lists the mandatory and optional parameters that you can configure during installation.

```
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
```

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
acceptLicense	Mandatory	no	Set Yes to accept the NetScaler Ingress Controller EULA.
affinity	Optional	N/A	Affinity labels for pod assignment.
nodeSelector	Optional	N/A	The node label key:value pair to be used for nodeSelector option in NetScaler Ingress Controller deployment.
podAnnotations	Optional	N/A	Map of annotations to add to the pods.
replicaCount	Optional	1	The number of CPX-NSIC pods to be deployed. With cpxBgpRouter : true, replicaCount is 1 because CPX is deployed as a DaemonSet.
tolerations	Optional	N/A	Specify the tolerations for the CPX-NetScaler K8s Gateway deployment.

NetScaler Consolse settings

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
ADMIP	Optional	N/A	The NetScaler Console (formerly NetScaler Console) IP address.
bandWidth	Optional	1000	The desired bandwidth capacity for NetScaler CPX in Mbps.
bandWidthLicense	Optional	false	Set this parameter to true if you want to use bandwidth-based licensing for NetScaler CPX.
cpxCores	Optional	1	The desired number of vCPUs for NetScaler CPX.
licenseEdition	Optional	PLATINUM	The possible values are Standard, Platinum, and Enterprise.
licenseServerIP	Optional	N/A	NetScaler Console (formerly NetScaler Console) IP address to license NetScaler CPX. For more information, see Licensing.
licenseServerPort	Optional	27000	NetScaler Console (formerly NetScaler Console) port if a non-default port is used.

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
loginSecret	Optional	N/A	The secret key to log in to NetScaler Console.
platform	Optional	false	Set to true to enable CP1000 platform license.
vCPULicense	Optional	N/A	Set the value to true if you want to use a vCPU-based license for NetScaler CPX.
licensingMode	Optional	pool	Set the licensing mode as <code>pool</code> or <code>las</code> . For <code>licensingMode las</code> , set ADMIP, loginSecret, bandWidth, and licenseEdition. For <code>pool</code> , set licenseServerIP and either bandWidthLicense or vCPULicense.

Exporter

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
required	Optional	false	Set this parameter to true to run the Exporter for NetScaler Stats along with NetScaler Ingress Controller to pull

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
			metrics for NetScaler CPX.
image	Optional	quay.io/netScaler/ netScaler-adc- metrics- exporter@sha256:3 d54b9151c742c001 17be74e1ff544303 58e4cf628fe1f02d5 03088583f678	The metrics exporter image hosted on Quay.io.
pullPolicy	Optional	IfNotPresent	The exporter image pull policy.
resources	Optional	{}	CPU/memory resources for the metrics exporter container.
ports.containerPort	Optional	8888	The exporter container port.
extraVolumes	Optional	N/A	Additional volumes for the additional volumeMounts.
imagePullSecrets	Optional	N/A	A list of Kubernetes secrets to be used for pulling the images from a private Docker registry or repository. For more information, see Pull an Image from a Private Registry .

Gateway Controller

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
disableAPIServerCertificateVerify	Optional	false	Set this parameter to true for disabling API server certificate verification.
enableLivenessProbe	Optional	true	Set this parameter to false to disable liveness probe settings for NetScaler Ingress Controller.
enableReadinessProbe	Optional	true	Readiness probe settings for NetScaler Ingress Controller.
entityPrefix	Optional	N/A	The prefix for the resources on NetScaler CPX.
extraVolumeMounts	Optional	N/A	N/A
gatewayControllerName	Mandatory	N/A	Provide the name for the Gateway Controller.
image	Mandatory	quay.io/netScaler/netScaler-k8s-ingress-controller@sha256:2ff38654476234bfe2c3908932702b90c8ec6ed3cfb7c7f467dc6181df0f1adf	The NetScaler Ingress Controller image hosted on Quay.io.
jsonLog	Optional	false	Set this parameter to true if log messages are

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
			required in the JSON format.
kubernetesURL	Optional	N/A	The kube-apiserver URL that NetScaler Ingress Controller uses to register events. If not specified, it uses the internal kube-apiserver IP address.
livenessProbe.initialDelaySeconds	Optional	30	The time period in seconds for which Kubernetes waits before performing the first liveness check.
livenessProbe.periodSeconds	Optional	60	After the initial delay, Kubernetes performs a liveness check every 60 seconds.
logLevel	Optional	INFO	The log level to control logs generated by NSIC. Supported levels: CRITICAL, ERROR, WARNING, INFO, DEBUG, TRACE, and NONE.
OpenShift	Optional	true	Set this parameter to false if the OpenShift environment is not being used.

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
optimizeEndpointBinding	Optional	false	Set to true to enable binding of back-end endpoints to a service group in a single API call. Recommended when many endpoints per application. (NetScaler >=13.0-45.7)
pullPolicy	Mandatory	IfNotPresent	The NetScaler Ingress Controller image pull policy.
readinessProbe	Optional	N/A	Readiness probe settings.
resources	Optional	{}	CPU/memory resources for the NetScaler Ingress Controller container.
routeLabels	Optional	N/A	Route label selectors used by NetScaler Ingress Controller for routeSharding in the OpenShift cluster.

NetScaler CPX settings

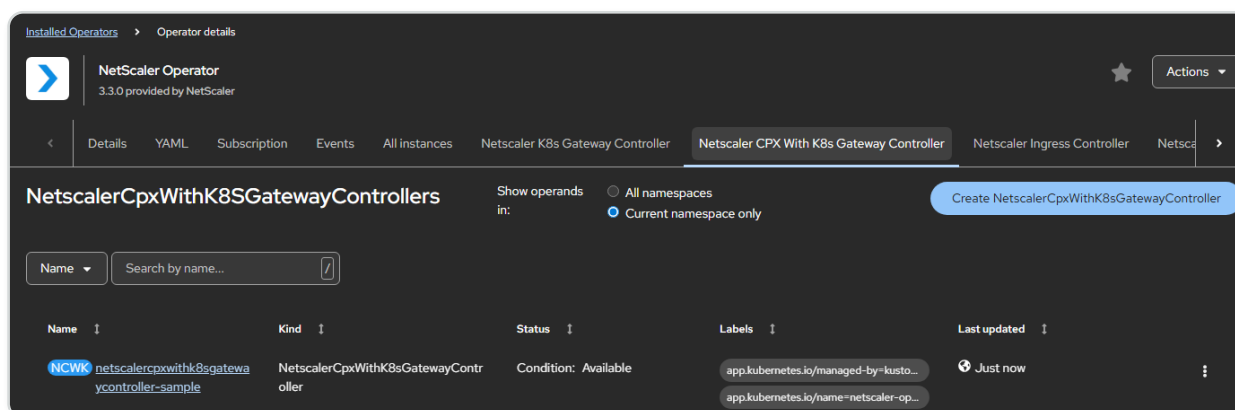
PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
cpxCommands	Optional	N/A	User-provided NetScaler bootstrap config applied when CPX is instantiated.

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
			Not dynamic — changes to the ConfigMap do not reflect until the pod is restarted.
cpxLicenseAggregator	Optional	N/A	IP address or FQDN of the CPX License Aggregator (if used) to license CPX.
cpxShellCommands	Optional	N/A	User-provided bootup commands applied when CPX is instantiated. Not dynamic — changes require pod restart.
enableLivenessProbe	Optional	true	Set to false to disable liveness probe settings for NetScaler CPX.
enableStartupProbe	Optional	true	Set to false to disable startupProbe settings for NetScaler CPX.
extraVolumeMounts	Optional	N/A	Additional volume mounts for CPX.
hostName	Optional	N/A	Host name to set for the CPX.
image	Mandatory	quay.io/netScaler/netScaler-cpx@sha256:04defadf6311259d6ada5	NetScaler CPX image hosted on Quay.io.

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
		380498ed7afe8035 b11ee3c1d7b456fa a94472b5550	
livenessProbe	Optional	N/A	Liveness probe settings for NetScaler CPX.
mgmtHttpPort	Optional	9080	HTTP port to be used for a NodePort CPX service.
mgmtHttpsPort	Optional	9443	HTTPS port to be used for a NodePort CPX service.
nitroReadTimeout	Optional	20	NITRO read timeout in seconds.
nsHTTP2ServerSide	Optional	OFF	Set to ON to enable HTTP/2 for NetScaler service group configurations.
nsProtocol	Optional	HTTP	Protocol used for communication between NetScaler CPX and NetScaler Ingress Controller.
pullPolicy	Mandatory	IfNotPresent	NetScaler CPX image pull policy.
service.annotations	Optional	N/A	Dictionary of annotations for the CPX service (for example, for exposing TLS

PARAMETERS	MANDATORY OR OPTIONAL	DEFAULT VALUE	DESCRIPTION
			secret through annotations).
service.spec	Optional	N/A	Service specification for CPX. To expose CPX as NodePort set service.spec.type: NodePort and use service.citrix.com/secret annotation to reference TLS secret.
startupProbe	Optional	N/A	Startup probe settings for NetScaler CPX.

8. After updating the values for the required parameters, click **Create**. Ensure that the NetScaler CPX with Kubernetes Gateway Controller is successfully deployed and initialized.



9. Attach privileged security context constraints to the service account of NetScaler CPX (as it runs as a privileged pod) by using the following command:
- Get the service account name used by NetScaler CPX using the following command in the namespace where NetScaler CPX has been deployed: `oc get sa`
 - Attach privileged SCC to the service account of NetScaler CPX:

```
oc adm policy add-scc-to-user privileged
```

```
system:serviceaccount:<namespace>:<CPX-ServiceAccount-Name retrieved in the  
previous step>
```

10. Navigate to **Workloads > Pods** section and verify that the `netScaler-cpx-with-gateway-controller` pod is up and running.

Project: default ▼

Pods

★ Create Pod

Filter ▼ Name ▼ Search by name... [🔍] [📄]

Name ↑	Status ↓	Ready ↓	Restarts ↓	Owner ↓	Memory ↓	CPU ↓	Created ↓	
 netScalercpxwithk8sgate waycontroller-sample- 556656745d-rpjbq	 Running	2/2	0	 netScalercpxwithk8sgate waycontroller-sample- 556656745d	746.9 MiB	0.068 cores	🕒 1 minute ago	⋮
 netScalerk8sgatewaycon troller-sample- 6fbfc8555c-8l6hl	 Running	1/1	0	 netScalerk8sgatewaycon troller-sample- 6fbfc8555c	206.5 MiB	0.002 cores	🕒 6 minutes ago	⋮

11. Verify the deployment by sending traffic.

- a. Obtain the NodePort details using the following command: `oc get svc`
- b. Use cpx-service NodePort and send the traffic as shown in the following command: `curl https://example.com:<NodePort> --resolve example.com:<NodePort>:<Master-Node-IP>`

The preceding cURL command must return the following output: `<html><body><h1>It works!</h1></body></html>`

Network

The following topics are available under Network:

- [Establish a network between Kubernetes nodes and NetScaler instance outside the Kubernetes cluster](#)

Establish a network between Kubernetes nodes and NetScaler instance outside the Kubernetes cluster

For seamless functioning of services deployed in a Kubernetes cluster, the Ingress NetScaler device must reach the overlay network where Pods run. You can get this connectivity in one of the following ways:

1. Enable automatic route configuration in the NetScaler Kubernetes Gateway Controller.

Set `gatewayController.nodeWatch: true` in the controller spec to create routes on NetScaler for each Pod subnet.

NOTE:

You can choose this option when NetScaler SNIP shares the same subnet as Kubernetes nodes.

2. Deploy a [NetScaler Node Controller](#) to drive subnet routes from Kubernetes nodes.

NOTE:

`gatewayController.nodeWatch` is `false` by default.

Multiple clusters behind a single NetScaler

If one NetScaler balances traffic for multiple Kubernetes clusters, overlapping CNI subnets might cause route conflicts and static route injection can fail. In that scenario, use [Policy-Based Routing \(PBR\)](#) and provide a distinct set of SNIP values for each cluster.

Provide SNIPs through Helm settings, environment variable, or ConfigMap so the controller installs the correct PBR entries.

Use the following command to configure PBR SNIPs in NetScaler:

```
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>\, ...]'
```

Traffic management

The following topics are available under Traffic management:

- [Dynamic IP address allocation through NetScaler IPAM Controller](#)
- [Multiple listeners within a Gateway](#)
- [Enhanced traffic management capabilities with HTTPRoute filters](#)
- [NetScaler profiles](#)

Dynamic IP address allocation through NetScaler IPAM Controller

NetScaler Kubernetes Gateway Controller supports IP address allocation for Kubernetes Gateway resources using NetScaler IPAM Controller. When IPAM is enabled, Gateways that do not have a static IP address configured in `spec.addresses` automatically receive a Virtual IP (VIP) address from NetScaler IPAM controller. With this feature, the need to manually assign and track IP addresses for each Gateway is eliminated.

Prerequisites

Before configuring dynamic IP address allocation through NetScaler IPAM Controller, ensure the following:

- **NetScaler Gateway Controller** is deployed with `ipam: True` in the Helm `values.yaml`. For more information, see [Deploy NetScaler Kubernetes Gateway Controller](#).
- **NetScaler IPAM Controller** is deployed in the cluster with the IP address ranges. For information on how to deploy NetScaler IPAM controller with the desired IP address ranges, see [NetScaler IPAM Controller Installation Guide](#).

When configuring the `VIP_RANGE`, you can define **named ranges** (for example, `Production`, `Staging`) that can be referenced from Gateway annotations, and **unnamed ranges** that serve as the default pool.

Configuration

Perform the following steps to configure dynamic IP address allocation through NetScaler IPAM Controller.

Step 1: Create a GatewayClass

Create a GatewayClass with NetScaler Kubernetes Gateway Controller name.

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

kind: GatewayClass

metadata:

  name: citrix-gateway-class

spec:

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

Step 2: Create a Gateway

Create a Gateway without `spec.addresses`. NetScaler IPAM controller automatically allocates an IP address from the configured pool.

Basic Gateway (default IP range)

```
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
```

Gateway with a named IP range

Use the `gateway.citrix.com/ipam-range` annotation on the Gateway to allocate an IP address from a specific named range:

```
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
```

The range name in the annotation must match a named range defined in `VIP_RANGE` configuration of NetScaler IPAM controller.

Verify the IP address allocation

After creating the Gateway, check the `status.addresses` field to confirm the allocated IP address:

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

Expected output:

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

Static IP versus IPAM

CONFIGURATION	BEHAVIOR
<code>spec.addresses</code> is set	NetScaler Kubernetes Gateway Controller uses the specified IP address directly. IPAM is not invoked.
<code>spec.addresses</code> is omitted	NetScaler Kubernetes Gateway Controller requests an IP address from NetScaler IPAM controller if IPAM is enabled.

If you need a specific IP address for a Gateway, set it explicitly in `spec.addresses`:

```
spec:
```

```
  addresses:
```

```
    - type: IPAddress
```

```
value: "10.1.2.50"
```

Annotation Reference

ANNOTATION	SCOPE	DESCRIPTION
<code>gateway.citrix.com/ ipam-range</code>	Gateway	Specifies the named IP address range from which NetScaler IPAM Controller allocates an address. If omitted, the default (unnamed) range is used.

Troubleshooting

SYMPTOM	POSSIBLE CAUSE	RESOLUTION
Gateway stays in pending state with no IP address assigned	NetScaler IPAM controller is not running or not enabled while deploying NetScaler Kubernetes Gateway Controller	Verify that NetScaler IPAM controller pod is running and <code>ipam: True</code> is set in <code>values.yaml</code> of NetScaler Kubernetes Gateway Controller
IP address not allocated from expected range	Range name in the annotation does not match NetScaler IPAM configuration	Check that the <code>gateway.citrix.com/ipam-range</code> annotation value matches a named range in <code>VIP_RANGE</code> of NetScaler IPAM controller
IP pool exhausted	All IP addresses in the configured range are allocated	Expand the <code>VIP_RANGE</code> in NetScaler IPAM Controller configuration or free unused Gateways
Gateway works without IPAM after <code>spec.addresses</code> is set	Expected behavior	Static addresses bypass IPAM. Remove <code>spec.addresses</code> to use IPAM allocation

Limitations

- Each Gateway receives a single VIP from NetScaler IPAM Controller.
- The `gateway.citrix.com/ipam-range` annotation must reference a range name that exists in NetScaler IPAM controller configuration. An invalid range name prevents IP allocation.
- When the annotation is omitted, NetScaler IPAM Controller automatically assigns an IP from the default (unnamed) pool (if configured).

Multiple listeners within a Gateway

The NetScaler® Kubernetes Gateway Controller supports multiple listeners within a single Gateway resource, as defined by the Kubernetes Gateway API. This capability provides greater flexibility and control over how external traffic is managed and routed to services within your Kubernetes cluster.

With multiple listener support, you can:

- **Expose multiple ports** on the same set of gateway IP addresses, so that different applications or services can be accessed through distinct ports.
- **Support different protocols**, such as HTTP and HTTPS, on separate ports while using the same gateway IP address.
- **Apply unique TLS configurations** for each listener, enabling secure communication tailored to specific requirements.
- **Define separate routing rules** for each listener, allowing you to manage and direct different inbound traffic flows.

This enhanced listener functionality enables advanced traffic management scenarios, such as hosting multiple secure and non-secure applications behind a single Gateway, or segmenting traffic for compliance and security purposes. By using multiple listeners, you can optimize your application delivery and improve the scalability and security of your Kubernetes environment.

Sample configuration

The following sample configuration in Kubernetes results in creating three content switching virtual servers and an ipset for each content switching virtual server.

```
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
```

Namespace filters

By default, Gateways enforce strict namespace isolation. This means only routes such as `HTTPRoute` that exist in the same namespace as the Gateway resource can attach to that Gateway. This default behavior provides a secure and isolated routing environment, ensuring that only trusted routes within the same namespace can influence the Gateway's traffic management.

To support more advanced routing scenarios, you can enable cross-namespace routing by configuring the `allowedRoutes` field within each listener definition of your Gateway resource. The `allowedRoutes` field provides granular control over which namespaces are permitted to

attach routes to a specific listener, allowing for flexible and secure multitenant or multi-team deployments.

The `allowedRoutes.namespaces.from` field supports three distinct modes for namespace selection:

- **Same (Default):** - Only routes from the same namespace as the Gateway are allowed to attach. This mode is the most restrictive and secure option, suitable for environments where strict isolation is required.
- **All:** - Routes from any namespace in the Kubernetes cluster can attach to the Gateway listener. This mode offers maximum flexibility, enabling centralized Gateway resources to serve routes from multiple namespaces. However, it is important to consider the security implications and implement appropriate RBAC controls when using this mode.
- **Selector:** - Routes are allowed to attach based on namespace label matching criteria. This mode uses Kubernetes label selectors to enable sophisticated namespace-based routing policies. For example, you can allow only namespaces with a specific label (such as `team=frontend`) to attach routes to a listener, providing fine-grained multitenant control.

NOTE:

If the namespace label is changed after `HTTPRoute` is applied, the changes do not get reflected in Gateway Controller.

Enhanced traffic management capabilities with HTTPRoute filters

The NetScaler® Kubernetes Gateway controller supports HTTPRoute filters as defined by the Kubernetes Gateway API. This feature allows you to implement sophisticated traffic manipulation logic directly within the HTTPRoute resources. By leveraging filters, you can modify requests and responses, enabling a wide range of use cases such as header manipulation, URL rewriting, and request redirection.

Integration with NetScaler CRDs by using extensionRef

The Custom Resource Definitions (CRDs) of NetScaler Ingress Controller (NSIC) can be referred through the `extensionRef` field within HTTPRoute filters. This powerful feature allows you to seamlessly integrate advanced NetScaler functionalities directly into your Gateway API configurations.

NSIC CRDs that can be referred through extensionRef include: [rewritepolicy](#), [ratelimit](#), [bot](#), [waf](#), [appqoeppolicy](#).

Sample reference

```
rules:
  - filters:
      - type: ExtensionRef
        extensionRef:
          group: "citrix.com"
          kind: "<crd-kind>"
          name: "<crdinstance-name>"
```

NOTE:

- CRD instances must be created without service names.
- For `extensionRef` in HTTPRoute filters, the group is always "citrix.com" and the kind corresponds to the NetScaler CRD type. Valid kinds include: `bot`, `waf`, `rewritepolicy`, `ratelimit`, and `appqoeppolicy`.

You can refer to NSIC CRDs for specialized processing, including:

- **Bot Management (BOT):** Protect your applications from malicious bot traffic by applying the sophisticated bot detection and mitigation techniques that NetScaler supports.
- **Web Application Firewall (WAF):** Secure your applications by integrating the robust WAF capabilities that NetScaler supports to inspect traffic and block known and zero-day attacks.
- **Rewrite Policies:** Apply advanced request and response rewriting rules beyond the standard Gateway API filters by using the rich rewrite policy engine of NetScaler.
- **Ratelimit:** Apply policies to manage the rate of incoming requests protecting your applications from being overwhelmed.

- **AppQoe Policy:** Apply policies that prioritize or limit traffic based on various criteria, ensuring optimal performance for critical applications and a fair allocation of resources.

The `extensionRef` mechanism acts as a bridge, allowing users to tap into the extensive feature set of NetScaler while using the Kubernetes Gateway API for traffic management.

Native support for standard HttpRoute API filters

Building upon its HTTPRoute filter support, the NetScaler Kubernetes Gateway Controller natively implements the following standard Gateway API filters.

URLRewrite

Modify the path or host name of requests before they are forwarded to the back-end service. This filter is useful for mapping user-facing URLs to internal service paths or for migrating services without changing the public URL.

Example: Rewriting `/old-path` to `/new-path` or changing the request's host name.

```
- 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 and ResponseHeaderModifier

Add, set, or remove HTTP headers for incoming requests or outgoing responses. This filter can be used for various purposes, such as injecting tracing information, setting security headers, or modifying cache-control directives.

Example: Adding an `X-Forwarded-Proto` header or removing an internal-only header from responses.

```
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

Issue HTTP 3xx redirect responses and redirect users to different URLs, enforce HTTPS, or manage moved content gracefully.

Example: Redirect HTTP traffic to HTTPS, or redirect a non-canonical host name to a canonical host name.

```
rules:

  filters:

    - type: RequestRedirect

      requestRedirect:

        path:

          type: ReplacePrefixMatch

          replacePrefixMatch: /new-path

        statusCode: 301

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

RequestMirror

Send a copy of the request to different backend services, which is highly beneficial for traffic shadowing. This filter allows testing of new service versions, performing analytics, or debugging issues without affecting the client's response. The response from the mirrored backend is Ignored.

NOTE:

You can specify multiple mirror backends. However, only one traffic mirroring percentage or fraction configuration is honored per rule.

Example: Mirroring production traffic to a monitoring and analytics service for real-time performance insights.

```
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® Profiles support

The NetScaler Kubernetes Gateway Controller introduces Profile CRDs (including SSL profile, HTTP profile, and TCP profile) to achieve robust, centralized configuration. These profiles allow cluster operators to define standardized, reusable infrastructure settings, such as TLS ciphers, TCP optimizations, or HTTP timeouts, that application developers can effortlessly apply to their routing definitions.

Deploy the Profile CRDs

The NetScaler Profiles YAML file: [profiles_crd.yaml](#).

NOTE:

Ensure that you do not modify the deployment YAML file.

Deploy the CRD, using the following command:

```
kubectl apply -f profiles_crd.yaml
```

For example,

```
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
```

Profile specification

All Profile CRDs use one of the following two fields to define the desired NetScaler configuration. These fields are mutually exclusive:

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

FIELD	PURPOSE	ACTION ON NETSCALER
<code>spec.preconfigured</code>	Use an existing profile on the NetScaler appliance.	The controller does not create a profile. It uses the provided string value as the profile name for binding.
<code>spec.config</code>	Define a profile.	The controller creates a uniquely named profile (based on the CRD's metadata) on the NetScaler, using the key-value pairs you provide in the config block.

Add NetScaler profiles in NetScaler Kubernetes Gateway Controller

You can add NetScaler profiles (HTTP, TCP, and SSL) in the Kubernetes Gateway resources as Frontend profile and Backend profile.

Frontend profiles

Frontend profiles are used to apply settings directly to the Gateway's listeners. For example, the frontend content switching virtual server on the NetScaler. These profiles must include

`targetRef` section for binding.

A single frontend profile CRD can be efficiently applied to multiple Gateways.

Example 1 Frontend SSL Profile set on all Gateway listeners

The following example defines a new frontend SSL profile with custom configuration parameters, and applies it to all listeners or content switching virtual servers of the “my-gateway” resource.

```
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"
```

Backend Profiles

Backend Profiles are used to configure settings that affect the connection between the Gateway(NetScaler) and the backend Kubernetes application. For example, service groups or servers.

Backend Profiles must not contain a `targetRef`. Instead, they are attached directly to a specific HTTPRoute rule using the standard `ExtensionRef` filter type in the `HTTPRoute`, allowing granular control per route.

Example 1: Backend preconfigured SSL Profile for HTTPRoute

The following example defines a profile that uses an existing NetScaler profile named "manual_secure_backend" and is intended for use by `HTTPRoutes` (no `targetRef` is present).

```
apiVersion: citrix.com/v1

kind: sslprofile

metadata:

  name: simple-ssl-profile-backend

spec:

  # Define the existing profile name

  preconfigured: "manual_secure_backend"
```

Example 2: Apply a Backend Profile to HTTPRoute

The following HTTPRoute references the backend `sslprofile` created in the previous example through an `ExtensionRef` filter.

```
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
```

Types of NetScaler profiles

The following NetScaler profile types are supported:

- HTTP profile
- TCP profile

- SSL profile

SSL Profile

An SSL profile is a collection of settings for SSL entities. It offers ease of configuration and flexibility. Instead of configuring the settings on each application, you can configure them in a profile and bind the profile to all the application that the settings apply to.

The SSL profile CRD schema reference is as follows:

FIELD	TYPE	DESCRIPTION	REQUIRED FOR FRONTEND PROFILES	REQUIRED FOR BACKEND PROFILES
preconfigured	string	Mutually exclusive with config. The name of a preconfigured or built-in SSL profile existing on the NetScaler appliance to be used.	No	No
config	object	Mutually exclusive with preconfigured. Contains all the parameters related to the NetScaler NITRO API for a fully custom profile. Customers can specify any valid NITRO SSL profile parameters here. For information	No	No

FIELD	TYPE	DESCRIPTION	REQUIRED FOR FRONTEND PROFILES	REQUIRED FOR BACKEND PROFILES
		about key-value pairs supported in the SSL profile, see SSL profile .		
targetRef	array of objects	A list of target resources (typically a Gateway) where this profile is applied. This field is required for Frontend Profiles and must be omitted for Backend Profiles.	Yes	No
gatewayClassName	string	Name of the <code>GatewayClass</code> that this SSL profile must be applied to. Used for scoping the profile to a specific controller instance in multitenant environments.	No	No
sslciphers	array of strings	List of ciphers to be bound to the SSL profile. The priority is determined by	No	No

FIELD	TYPE	DESCRIPTION	REQUIRED FOR FRONTEND PROFILES	REQUIRED FOR BACKEND PROFILES
		the order in the list and can include a cipher suite, a predefined cipher group, or a user-created cipher group.		

IMPORTANT

- If you use a pre-configured SSL profile, you must bind ciphers manually through NetScaler, and `spec.policies.sslciphers` are ignored.
- Built-in cipher groups are supported in Tier-1 and Tier-2 NetScaler, but user-defined cipher groups are supported only in a Tier-1 NetScaler.

Example 1: Preconfigured Frontend SSL Profile

```

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"
```

Example 2: Custom Frontend SSL Profile

```
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"
```

Example 3: Preconfigured Backend SSL Profile and the HTTPRoute referring the profile

```
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
```

Example 4: Custom Backend SSL Profile and HTTPRoute referring the profile

```
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 profile

A TCP profile is a collection of TCP settings used to optimize transport layer behavior. This profile allows you to apply advanced settings, such as TCP Fast Open or HyStart to improve connection performance and efficiency.

TCP Profile CRD schema reference

FIELD	TYPE	DESCRIPTION	REQUIRED FOR FRONTEND PROFILES	REQUIRED FOR BACKEND PROFILES
preconfigured	string	Mutually exclusive with config. The name of a preconfigured or built-in TCP profile existing on the NetScaler appliance to use.	No	No

FIELD	TYPE	DESCRIPTION	REQUIRED FOR FRONTEND PROFILES	REQUIRED FOR BACKEND PROFILES
config	object	Mutually exclusive with preconfigured. Contains all the parameters related to the NetScaler NITRO API for a fully custom profile. Customers can specify any valid NITRO TCP profile parameters. For information about all the possible key-value pairs for a TCP profile, see TCP profile .	No	No
targetRef	array of objects	A list of target resources (typically a Gateway) where this profile is applied. This field is required for Frontend profiles and must be omitted for Backend profiles.	Yes	No
gatewayClassName	string	Name of the <code>GatewayClass</code>	No	No

FIELD	TYPE	DESCRIPTION	REQUIRED FOR FRONTEND PROFILES	REQUIRED FOR BACKEND PROFILES
		that this TCP profile must be applied to. Used for scoping the profile to a specific controller instance in multitenant environments.		

Example 1: Preconfigured Frontend TCP Profile

```
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"
```

Example 2: Custom Frontend TCP Profile

```
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
```

Example 3: Preconfigured Backend TCP profile and HTTPRoute referring the profile

```
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
```

Example 4: Custom Backend TCP profile and HTTPRoute referring the profile

```
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 Profile

An HTTP profile is a collection of HTTP settings used to configure various application layer behaviors, such as enabling WebSocket, customizing HTTP timeouts, or setting strict validation rules.

HTTP profile CRD schema reference

FIELD	TYPE	DESCRIPTION	REQUIRED FOR FRONTEND PROFILES	REQUIRED FOR BACKEND PROFILES
preconfigured	string	Mutually exclusive with config. The name of a preconfigured or built-in HTTP profile existing on the NetScaler appliance to use.	No	No
config	object	Mutually exclusive with preconfigured. Contains all the parameters related to the NetScaler NITRO API for a fully custom profile. Customers can specify any valid NITRO TCP profile parameters. For information about all the possible key-value pairs for	No	No

FIELD	TYPE	DESCRIPTION	REQUIRED FOR FRONTEND PROFILES	REQUIRED FOR BACKEND PROFILES
		the HTTP profile see, HTTP profile .		
targetRef	array of objects	A list of target resources (typically a Gateway) where this profile is applied. This field is required for Frontend profiles and must be omitted for Backend profiles.	Yes	No
gatewayClassName	string	Name of the GatewayClass that this HTTP profile must be applied to. Used for scoping the profile to a specific controller instance in multitenant environments.	No	No

Example 1: Preconfigured Frontend HTTP profile

```
apiVersion: citrix.com/v1
```

```
kind: httpprofile
```

```
metadata:

  name: frontend-http-preconfigured

spec:

  targetRef:

    - name: my-gateway-https

    kind: Gateway

  preconfigured: "frontend1"
```

Example 2: Custom Frontend HTTP profile

```
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
```

```
dropinvalidreqs: ENABLED
```

Example 3: Preconfigured Backend HTTP profile and HTTPRoute referring the profile

```
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
```

Example 4: Custom Backend HTTP profile and HTTPRoute referring the profile

```
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
```

Controller scoping through `spec.gatewayClassName`

The `spec.gatewayClassName` field controls which NetScaler Gateway Controller instances process the profile CRD. This control is crucial in multitenant or multi-controller environments.

SCENARIO	SPEC . GATEWAYCLASSNAME	CONTROLLER BEHAVIOR
Scoped	PRESENT (for example, <code>netScaler-prod</code>)	Only the Controller instance managing that <code>specific GatewayClass</code> processes the Profile, ensuring isolation and targeted configuration.
Global	ABSENT	All NetScaler Gateway Controller instances in the cluster are free to accept and configure the profile on their respective NetScaler appliances.

If a Profile specifies a `gatewayClassName` that is not currently deployed in the cluster, the profile CRD is skipped until the corresponding `GatewayClass` becomes available.

Traffic security

The following topics are available under Traffic security:

- [Using pre-configured certificates](#)
- [Certificate key bundle in NetScaler by using the NetScaler Kubernetes Gateway Controller](#)
- [Use "appProtocol" to define backend application communication protocol](#)

Using pre-configured certificates with NetScaler Kubernetes Gateway Controller

This topic explains how to use certificates or certificate bundles that are already configured on your NetScaler appliance with your Kubernetes Gateway. This feature simplifies certificate management by allowing you to reference existing certificates instead of managing them as Kubernetes Secrets.

Benefits

Some of the benefits of using pre-configured certificates with NetScaler Kubernetes Gateway Controller are as follows:

- **Simplified Management:** Use certificates already configured on your NetScaler appliance.
- **No Secret Creation:** Avoid the overhead of creating and maintaining Kubernetes Secrets for certificates.
- **Centralized Control:** Manage certificates directly on NetScaler while configuring them in Kubernetes.

How to reference pre-configured certificates

When defining your Gateway, you can reference pre-configured certificates or certificate bundles in the `certificateRefs` section of your listener configuration.

Certificate types

You can reference the following two types of pre-configured certificates or certificate-bundles:

1. **Domain Certificate:** Server certificates for HTTPS connections.

2. **CA Certificate:** Certificate Authority certificates for client authentication. You can specify multiple CA certificates.

Basic configuration

Here's a simple example of referencing a pre-configured certificate:

NetScaler setup

```
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
```

Or use a pre-configured certificate bundle:

```
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
```

The description of the configuration fields are as follows:

- **group** : Specify the certificate source:
 - "netscaler.certkey" for pre-configured NetScaler certificates
 - "netscaler.certkeybundle" for pre-configured NetScaler certificate bundles
- **kind** : Specify the certificate type:
 - "ca" - Use for Certificate Authority certificates (for client authentication). You can specify multiple CA certificates
 - "sni" - Use for server certificates that should be bound as SNI.

NOTE:

kind is optional for server certificates. If only one server certificate is present without kind, it will be bound as default. If multiple server certificates are present, they will automatically be bound as SNI certificates

- **name** : The name of the certificate as configured on your NetScaler appliance

Common Scenarios

Scenario 1: Single certificate

Use a single pre-configured certificate for your HTTPS listener:

```
listeners:

- name: https

  protocol: HTTPS

  port: 443

  tls:

    mode: Terminate

    certificateRefs:

      - group: "netscaler.certkey"

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

Scenario 2: Multiple domains certificates

When multiple server certificates are provided, you must create an SSL profile with `snienable` set to `ENABLED`:

```
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"
```

Required SSL profile

```
apiVersion: citrix.com/v1

kind: sslprofile

metadata:

  name: frontend-ssl-custom

spec:

  targetRef:

    - name: my-gateway-https

      sectionName: https

  config:

    sslprofiletype: "frontend"

    snienable: "ENABLED"
```

Scenario 3: Client authentication with CA certificate

Add CA certificates for client authentication:

```
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"
```

Scenario 4: Using certificate bundles

If you have certificate bundles configured on NetScaler, use the `netscaler.certkeybundle` group:

```
listeners:

- name: https

  protocol: HTTPS

  port: 443

  tls:

    mode: Terminate

    certificateRefs:

      - group: "netscaler.certkeybundle"
```

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

Scenario 5: Combining pre-configured and Kubernetes certificates

You can combine pre-configured certificates with regular Kubernetes secrets:

```
listeners:

- name: https

  protocol: HTTPS

  port: 443

  tls:

    mode: Terminate

    certificateRefs:

      - kind: Secret

        name: "k8s-tls-secret"

      - group: "netscaler.certkey"

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

Prerequisites

Before using pre-configured certificates or certificate-bundle in your Gateway:

1. Ensure that the certificates are already configured on your NetScaler appliance.
2. Note the exact certificate names as they appear in NetScaler and refer it in the Gateway CRD.
3. If certificates are created on NetScaler after deploying the Gateway CRD instance, delete and re-apply the Gateway resource YAML to ensure proper binding

Troubleshooting

Some of the common issues and their resolutions are as follows:

- Issue: Certificate Not Found

Resolution: Ensure that the certificate name exactly matches the name configured on NetScaler.

- Issue: Wrong Certificate Served

Resolution: When using multiple certificates (SNI), ensure the following:

- An SSL profile with snienable set to ENABLED is created and bound to the SSL virtual server.
- You can create an SSL profile using the SSLProfile CRD - Example SSLProfile configuration that references your Gateway:

```
apiVersion: citrix.com/v1

kind: sslprofile

metadata:

  name: frontend-ssl-custom

spec:

  targetRef:

    - name: my-gateway-https

      sectionName: https

  config:

    sslprofiletype: "frontend"

    snienable: "ENABLED"
```

Summary

Referencing pre-configured certificates in your Kubernetes Gateway is straightforward and can be summarized as follows:

1. Use group: "netscaler.certkey" for regular certificates

2. Use group: "netscaler.certkeybundle" for certificate bundles
3. Specify kind: "ca" for CA certificates. For server certificates, kind is optional
4. Provide the certificate name as configured on NetScaler

Certificate key bundle in NetScaler® by using the NetScaler Kubernetes Gateway Controller

NetScaler Kubernetes Gateway Controller now supports certificate bundle (`certkeybundle`) functionality, which is supported on NetScaler starting from release 14.1 build 21.x. With this functionality, the issue with the certificate chain and the additional handling that is required when two certificates share an intermediate CA are resolved. For more information on certificate key bundle support in NetScaler, see [Support for SSL certificate key bundle](#).

NetScaler Kubernetes Gateway Controller creates a certificate bundle for certificates provided in Gateway Resource.

To enable this feature, a new environment variable `CERT_BUNDLE` is added in the NetScaler Kubernetes Gateway Controller, which can be set by using the `certBundle` argument from the Helm Charts. For more information on setting the `certBundle`, see Helm Charts of [NetScaler Kubernetes Gateway Controller](#). NetScaler Kubernetes Gateway Controller adds the certificate bundle to the content switching virtual server of type SSL.

Certificate Bundle creation in NetScaler does not work in the following scenarios:

- A secure back end (service group of type SSL) is configured.
- Self-signed certificates are present.

Points to note

1. Certificate bundle creation in NetScaler fails if the following order is not met:
 - Server certificate (SC) must be placed at the top of the bundle file.
 - IC are intermediate certificates. IC is issued by IC. The certificates must be placed in a sequence, and all the intermediate certificates must be present in the bundle.
 - Certificates must be of PEM format only.
 - Server certificate key (SCK) can be placed anywhere in the bundle.
 - A maximum of 9 intermediate certificates are supported

2. If you upgrade the Controllers with the `CERT_BUNDLE` feature, the previous certificate key bindings get removed and a new `certkeybundle` gets created. The new `certkeybundle` gets bound to the context switching virtual server of type SSL.

Use "appProtocol" to define backend application communication protocol

The NetScaler Kubernetes Gateway Controller uses the `appProtocol` field on a Kubernetes service port to explicitly set the protocol for the resulting NetScaler service group, that controls the communication between the NetScaler and the backend applications.

For example, setting `appProtocol: ssl` on the service port ensures the corresponding service group on the NetScaler is configured with the SSL protocol. If not explicitly set, the default service group protocol is HTTP.

```
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 Gateway

NetScaler® AI Gateway enables enterprises to govern AI traffic flowing between applications and large language model (LLM) backends. NetScaler Kubernetes Gateway Controller extends AI Gateway capabilities to Kubernetes environments, allowing you to configure and manage AI Gateway policies using Kubernetes-native resources without manual CLI or GUI configuration on the NetScaler instance.

Using NetScaler Kubernetes Gateway Controller, you can automate AI Gateway configuration through CRDs and annotation-based policies along with your existing application workloads.

Overview

When to use AI Gateway

- Your applications run on Kubernetes and call Azure OpenAI LLM.
- You need a single governed entry point for all AI traffic, with token-aware routing, rate limiting, prompt inspection, and spillover.
- You want an AI Gateway policy defined as Kubernetes manifests and managed through GitOps workflows.
- External applications or legacy systems outside the cluster need governed access to LLM backends running inside Kubernetes.

Deployment options

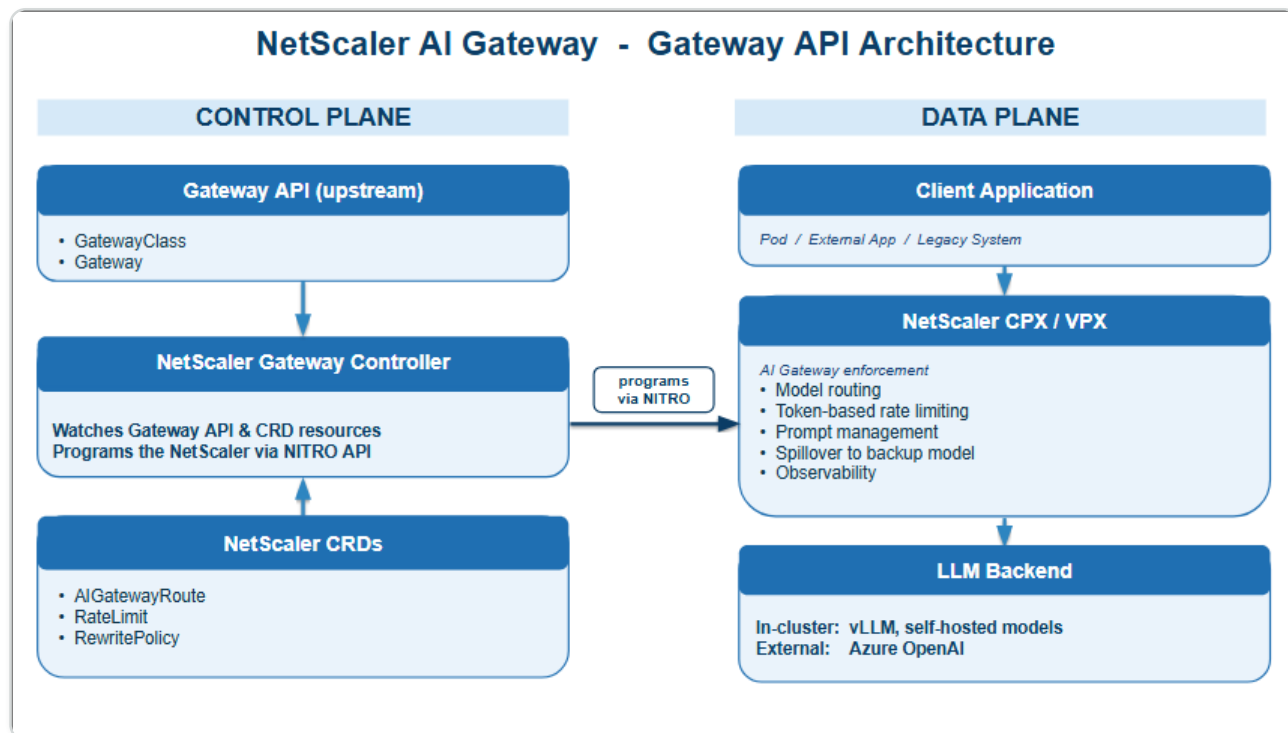
The Gateway Controller supports two NetScaler form factors depending on where your traffic originates:

DEPLOYMENT	FORM FACTOR	USE WHEN
In-cluster	NetScaler CPX	Client applications run inside the Kubernetes cluster. Handles both east-west traffic to self-hosted LLM backends and egress to external providers

DEPLOYMENT	FORM FACTOR	USE WHEN
External	NetScaler VPX	Client applications run outside the cluster — external apps, legacy systems, or other clusters reaching LLM backends

Architecture

The Gateway Controller sits in the control plane. It watches Kubernetes resources and programs NetScaler — it is not in the traffic path. Inference requests flow directly from the client application to NetScaler and on to the LLM backend.



Key Kubernetes resources

RESOURCE	DEFINED BY	PURPOSE
GatewayClass	Gateway API (upstream)	Binds the Gateway to the NetScaler controller (CPX or VPX)
Gateway	Gateway API (upstream)	Defines the listener with AI Gateway front end profile

RESOURCE	DEFINED BY	PURPOSE
AIGatewayRoute	NetScaler CRD	Model routing rules — matches by model name, routes to backend service
RateLimit	NetScaler CRD	Token-based quota — per-user, per-org, per-model, or per-gateway
RewritePolicy	NetScaler CRD	Prompt modification and URL rewrite for spillover

GatewayClass and Gateway are standard Gateway API resources. AIGatewayRoute , RateLimit , and RewritePolicy are NetScaler-specific CRDs installed with the Gateway Controller and are not portable to other Gateway API implementations.

NOTE:

AI traffic sent to external providers (Azure OpenAI) and any spillover to a backup provider is billed by that provider per request. Use token quotas (`aigatewayprofile`) and rate limits ([Configure rate limits and streams](#)) to control spend.

Related topics

- [Configure model routing](#)
- [Configure rate limits and streams](#)
- [Configure prompt management](#)
- [Configure spillover to an alternate model](#)
- [Observability](#)

Reference

For AI Gateway configuration at NetScaler level, see [AI capabilities](#).

Deploy NetScaler Kubernetes Gateway Controller with AI capabilities

From NetScaler Kubernetes Gateway Controller 2.2.0 release onwards, the NetScaler AI Gateway feature is supported. For more information on NetScaler AI Gateway, see [NetScaler AI Gateway](#).

You can enable NetScaler® AI Gateway capabilities using either of the following deployment options, depending on where your traffic originates:

- **NetScaler Kubernetes Gateway Controller with NetScaler VPX/MPX/BLX** — NetScaler Kubernetes Gateway Controller runs in the cluster and configures an external NetScaler VPX/MPX/BLX.
- **NetScaler CPX with Gateway Controller** — NetScaler CPX and NetScaler Kubernetes Gateway Controller run together inside the cluster.

Prerequisites

- A Kubernetes cluster with `kubect1` configured.
- Helm 3.x or later.
- Gateway API CRDs installed from the [official standard channel](#).

Deploy NetScaler Kubernetes Gateway Controller with NetScaler VPX/MPX/BLX

Use this option when NetScaler Kubernetes Gateway Controller manages an external NetScaler VPX/MPX/BLX.

Install NetScaler Kubernetes Gateway Controller

To install the chart with the release name, `my-release`, use the following command:

```
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>
```

NOTE:

By default, the chart installs the recommended [RBAC](#) roles and role bindings.

The command deploys NetScaler Kubernetes Gateway Controller on Kubernetes cluster with the default configuration. The [configuration](#) section lists the mandatory and optional parameters that you can configure during installation.

Create the GatewayClass

After the NetScaler Kubernetes Gateway Controller is installed, create a `GatewayClass` that binds to the controller. The `controllerName` must match the `gatewayController.gatewayControllerName` value used during installation.

```
kubectl apply -f - <<EOF

apiVersion: gateway.networking.k8s.io/v1

kind: GatewayClass

metadata:

  name: aigwclass

spec:

  controllerName: citrix.com/nsgw-controller

EOF
```

NOTE:

The `gatewayClassName` field in every AI Gateway resource (`Gateway` , `aigatewayprofile` , `aigatewayroute` , `ratelimit` , and `rewritepolicy`) must match the `metadata.name` of the `GatewayClass` created here (`aigwclass`).

Deploy NetScaler CPX with Gateway Controller

Use this option to run NetScaler CPX and the Gateway Controller together inside the cluster.

Install NetScaler CPX with Gateway Controller

To install the chart with the release name `my-release` :

```
helm install my-release netscaler/netscaler-cpx-with-gateway-controller --  
set  
license.accept=yes,gatewayController.gatewayControllerName=<controllerName  
>
```

NOTE:

By default the chart installs the recommended [RBAC](#) roles and role bindings.

The command deploys NetScaler CPX with NetScaler Kubernetes Gateway Controller as a sidecar on the Kubernetes cluster with the default configuration. The [configuration](#) section lists the mandatory and optional parameters that you can configure during installation.

Create the GatewayClass

After NetScaler CPX with Gateway Controller is installed, create a `GatewayClass` that binds to the controller. The `controllerName` must match the `gatewayController.gatewayControllerName` value used during installation.

```
kubect1 apply -f - <<EOF  
  
apiVersion: gateway.networking.k8s.io/v1  
  
kind: GatewayClass  
  
metadata:  
  
  name: aigwclass  
  
spec:  
  
  controllerName: citrix.com/nscpxgw-controller  
  
EOF
```

NOTE:

The `gatewayClassName` field in every AI Gateway resource (`Gateway` , `aigatewayprofile` , `aigatewayroute` , `ratelimit` , and `rewritepolicy`) must match the `metadata.name` of the `GatewayClass` created here (`aigwclass`).

Verify the deployment

Confirm that the controller is running and the `GatewayClass` is accepted before you apply AI Gateway resources.

```
kubectl get pods  
  
kubectl get gatewayclass aigwclass
```

The controller pod must show `STATUS Running` (for CPX, the pod shows `READY 2/2`), and the `GatewayClass` must show `ACCEPTED True` .

Reference

- [NetScaler Kubernetes Gateway Controller Helm chart](#)
- [NetScaler CPX with Gateway Controller Helm chart](#)

Configure model routing

NetScaler® AI Gateway routes incoming inference requests to the correct Large Language Model (LLM) backend based on the model name in the request. This feature provides a single Gateway endpoint for all models. Clients never need to know the backend service addresses or topology.

Model upgrades, backend changes, and new model additions don't require application code changes.

Prerequisites

- NetScaler Kubernetes Gateway Controller deployed with desired configurations.
- `GatewayClass` created and bound to the NetScaler instance.
- AI inference services deployed as external services in Kubernetes.

Configuration

AI Gateway profiles

An AI Gateway profile defines the settings that the NetScaler applies when it processes AI traffic between your applications and LLM backends. A profile can be of type `frontend` (applied to client-facing virtual servers) or `backend` (applied to the service groups that connect to the LLM backends). You can refer to the [aigatewayprofile CRD documentation](#) for more details.

NOTE:

Once created, modifying an AI Gateway profile, or changing the AI Gateway profile attached to a Gateway or AIGatewayRoute, causes the associated NetScaler entities to be recreated. As a result, traffic impacted.

Frontend AI Gateway profile

The front-end profile defines the AI Gateway endpoint type.

Front-end AIGateway profile for Gateway

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1

kind: aigatewayprofile

metadata:

  name: csaigwyfp

spec:

  gatewayClassName: "aigwclass"

  config:

    endpointtype: "azureopenai"

    profiletype: "frontend"
```

```
EOF
```

Front-end AIGateway profile for AIGatewayRoute

```
kubectl apply -f - <<EOF

apiVersion: citrix.com/v1

kind: aigatewayprofile

metadata:

  name: lbaigwyfp

spec:

  gatewayClassName: "aigwclass"

  config:

    endpointtype: "azureopenai"

    profiletype: "frontend"

EOF
```

Backend AI Gateway profile

The backend profile defines the per-model quota and authentication configuration.

Create a Kubernetes Secret containing the Authentication token/API key for the AI Gateway endpoint. Replace `<auth-token>` with the authentication token for AI Gateway endpoint.

```
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
```

Kubernetes Gateway with AI Gateway profile

The Kubernetes Gateway defines the entry point for the AI incoming traffic. To add the frontend AI Gateway profile to the Gateway, add the name of the frontend profile under the `infrastructure` reference. This frontend profile is applied to the content switching (CS) virtual server that fronts the AI traffic in NetScaler.

```
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

```

Refer to [Gateway Documentation](#) for the fields of the Gateway Object.

NOTES:

- Only frontend AI Gateway profile can be applied with a Gateway object.
- Create the TLS secret `web-ingress-secret` referenced by the Gateway listener before applying the Gateway. Replace the certificate and key paths with your own:

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

NetScaler AI Gateway route

Define routing rules that match requests by model name and forward each rule to the corresponding backend service. In the `AIGatewayRoute`, the frontend AI Gateway profile is applied to the load balancing (LB) virtual server, and the backend AI Gateway profile is applied to the service group that connects to the LLM backend. You can refer to the [aigatewayroute CRD documentation](#) for more details.

Complex routing with NetScaler PolicyExpression

Use a NetScaler `policyExpression` in `matches` to route on any part of the request. For example, this sample inspects the request body and routes to `gpt-model-svc` when the body contains `gpt`.

```

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
```

Simple routing in Kubernetes native way

Use the standard Gateway API `path` match for straightforward routing. For example, this sample routes all requests under the `/` path prefix to `gpt-model-svc`.

```
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

Reference

[Content switching-based model routing](#)

Configure rate limits and streams

Rate limiting enforces quotas at the Gateway and/or AIGatewayRoute, giving you precise control over LLM resource usage per user and per organization. You can limit by request count, by actual token consumption, or both.

Token-based rate limiting (`mode: TOKEN_RATE`) measures actual token consumption, which reflects real GPU resource usage — unlike request-count limits.

Requests exceeding the configured quota receive a `429 Too Many Requests` response before the request reaches the LLM backend.

IMPORTANT:

Rate limits are the primary control for capping LLM spend. Without a token-based limit (`mode: TOKEN_RATE`), a single user or organization can consume unlimited tokens against a paid provider, with no upper bound on cost. Configure a `TOKEN_RATE` limit per user and per organization to cap spend.

Per Gateway Vs Per Model rate limits

Rate limits can be applied globally at the Gateway level (across all model routes) or to a specific AI model through an `AIGatewayRoute` .

TYPE	HOW IT BINDS	SCOPE
Per-model	<code>filters[].extensionRef</code> in <code>AIGatewayRoute</code> rule	One model route only
Per-gateway	<code>targetRef</code> pointing to the <code>Gateway</code> resource	All routes on the Gateway

NOTE:

Both per-model and global rate limits can be simultaneously active. The per-model limit is evaluated first. The global limit applies across all models regardless of per-model quota state.

Configuration

Apply per-model rate limits

These `RateLimit` CRDs apply per AI model and are referenced from `AIGatewayRoute` rules via `filters[].extensionRef`.

The following examples enforce rate limits per organization and per user, keyed on the `X-org-id` and `X-user-id` request headers. These are only examples — you can key a rate limit on any attribute that the [ratelimit CRD](#) supports.

Per organization rate limits

A per-organization limit caps how much every organization can consume, identified by the `X-org-id` header. For example, this sample configuration sets two quotas per organization over a 60-second window: a request-count limit (`req_threshold: 20`) and a token limit (`mode: TOKEN_RATE, req_threshold: 4000`). Requests over either quota are blocked (`throttle_action: RESPOND`). For all parameters, see the [ratelimit CRD](#).

```
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
```

Per user rate limits

A per-user limit caps how much each individual user can consume, identified by the `X-user-id` header. For example, this sample configuration allows each user 20 requests and 4000 tokens per minute. For all parameters, see the [ratelimit CRD](#).

```
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
```

Streams for observability

Streams collect and export consumption metrics from the Gateway, so you can see how requests and tokens are being used against the configured limits. You add a `streams` section to the `ratelimit` spec, and the same configuration applies to both per-user and per-organization rate limits.

For example, this sample configuration defines two streams over a 10-minute interval (`interval: 10` , in minutes): one that tracks the top consumers by request count (`sort: REQUESTS`) and one by token count (`sort: TOKENS`). Each stream logs up to 20 entries (`logLimit: 20`) at a `logInterval` of 10 minutes. The `logInterval` must be greater than or equal to the stream `interval` .

For these streams to be useful, you must also apply the analytics configuration — see [Observability](#).

```
streams:

  - logInterval: 10

    logLimit: 20

    sort: "REQUESTS"

    interval: 10

  - logInterval: 10

    logLimit: 20

    sort: "TOKENS"

    interval: 10
```

Apply AIGatewayRoute with rate limit filters

The `AIGatewayRoute` binds the rate limits defined earlier to a model route through its `filters` section. For example, this sample references both the `orgaimodel` and `useraimodel` rate limits (defined above) as `ExtensionRef` filters, so both quotas apply to the matched model.

```
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

```

Each rule in the `AIGatewayRoute` references the `RateLimit` CRDs via `filters` of type `ExtensionRef` — binding the rate limit policy to that specific model route.

Apply Gateway-level rate limits

These `RateLimit` CRDs use `targetRef` to reference the Gateway directly — enforcing quotas across all model routes regardless of which model is matched.

Per org Gateway level rate limits

```

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
```

Per user Gateway level rate limits

```
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

Reference

[Configure rate limiting based on token consumption](#)

Configure prompt management

NetScaler® AI Gateway can intercept and modify incoming prompt content before it reaches the LLM backend. A `RewritePolicy` CRD locates the user message in the request body using a JSON XPath expression and appends or replaces content transparently to the client application.

Use cases

- Inject contextual metadata into every prompt.
- Append organization-specific instructions globally to all requests.
- Enforce system-level constraints on prompt content before model inference.

RewritePolicy CRD for prompt management

A `RewritePolicy` intercepts the request body, identifies the user message field via a JSON XPath expression, and appends or replaces the configured content before forwarding to the backend model. You can apply it globally at the Gateway level or to a specific AI model through an `AIGatewayRoute`.

Per Gateway Vs Per Model rewrite policies

A rewrite policy can be applied globally at the Gateway level (across all model routes) or to a specific AI model through an `AIGatewayRoute`.

TYPE	HOW IT BINDS	SCOPE
Per-model	<code>filters[].extensionRef</code> in <code>AIGatewayRoute</code> rule	One model route only
Per-gateway	<code>targetRef</code> pointing to the <code>Gateway</code> resource	All routes on the Gateway

NOTE:

Both per-model and global rewrite policies can be simultaneously active.

Configuration

Per-model rewrite policy

This `RewritePolicy` CRD applies to a specific AI model and is referenced from an `AIGatewayRoute` rule via `filters[].extensionRef`.

For example, this sample configuration appends the text `"I am trained till 10-28-2025"` after the first user message in the request body, located using a JSON XPath expression (`operation: insert_after`). For all parameters, see the [rewritepolicy CRD](#).

```
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
```

NetScaler AI Gateway route with the rewrite policy filter

The `AIGatewayRoute` binds the per-model rewrite policy to a model route through its `filters` section, referencing it as an `ExtensionRef` filter.

```
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

```

Rewrite policy at Kubernetes Gateway

This `RewritePolicy` CRD uses `targetRef` to reference the Gateway directly — applying to all model routes regardless of which model is matched.

```

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
```

Reference

[Prompt management](#)

Configure spillover to an alternate model

When a primary model's token quota is exhausted, NetScaler® AI Gateway automatically routes requests to a configured backup model, maintaining service continuity without any client-side changes.

WARNING:

Spillover forwards overflow traffic to the backup model you configure. If that backup is a paid external provider (Azure OpenAI, OpenAI, Anthropic, or Google Vertex AI), every request that exceeds the primary model's token quota is billed by that provider. Size the primary `tokenquota` and choose the spillover backend deliberately to avoid unexpected charges.

How it works

Each rule in an `AIGatewayRoute` lists a primary `backendRef` and a spillover `backendRef` marked with `is_spillover: true`. The spillover backend uses its own `backend-aigatewayprofile` for independent quota tracking.

A `RewritePolicy` rewrites the model name in the URL path so the backup backend receives the correct model identifier it expects.

Key configuration elements

ELEMENT	PURPOSE
<code>is_spillover: true</code> on <code>backendRef</code>	Marks the backend as spillover. This element is used only when the primary quota is exhausted.
<code>backend-aigatewayprofile</code> on the spillover backend	Separate backend profile giving the spillover backend its own independent quota tracking. If not specified, the <code>default-backend</code> profile under <code>spec.aigatewayprofiles</code> is used.
<code>spillover-frontend</code> under <code>spec.aigatewayprofiles</code>	Frontend profile always used for the spillover backend.
<code>RewritePolicy</code> through <code>ExtensionRef</code> filter	Rewrites the model name in the URL to match what the backup service expects.

Configuration

Spillover backend profile

Create a backend AI Gateway profile for the spillover model so it gets its own independent quota tracking. The route references this profile as `aigwybp-spill`.

Create a Kubernetes Secret containing the authentication token or API key for the spillover AI Gateway endpoint. Replace `<auth-token>` with the authentication token for the spillover endpoint.

```
kubect1 create secret generic aigw-auth-spill --from-  
literal=authToken=<auth-token>
```

```
kubect1 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
```

Spillover rewrite policy

```
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
```

This `RewritePolicy` replaces the model name in the URL path (third segment) with the backup model name. For example, a request originally targeting `/openai/deployments/gpt/chat/completions` is rewritten to `/openai/deployments/gpt2.5/chat/completions` before forwarding to the backup service.

AI Gateway routes with spillover service

The `AIGatewayRoute` defines primary and spillover backends per model rule. The spillover backend (`is_spillover: true`) attaches the spillover rewrite policy through `filters[].extensionRef`.

The `backend-aigatewayprofile` on the spillover backend defines a separate backend profile so the spillover model gets its own independent quota tracking. If `backend-aigatewayprofile` is not specified, the spillover backend falls back to the `default-backend` profile defined under `spec.aigatewayprofiles`.

The frontend profile for the spillover backend is always taken from `spec.aigatewayprofiles.spillover-frontend`.

```
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
```

Reference

[Configure spillover route requests to a different language model](#)

Observability

NetScaler® AI Gateway exports per-request AI metrics to Splunk, including token counts, model identity, client identity, latency, and cost attribution once observability is configured. No application instrumentation is required.

NetScaler exports these metrics to Splunk through the Splunk HTTP Event Collector (HEC). You configure the HEC connection and analytics profiles as described in this page.

Prerequisites

- NetScaler Kubernetes Gateway Controller already deployed. See [Deploy Kubernetes Gateway Controller with AI capabilities](#).
- A reachable Splunk instance with the HTTP Event Collector (HEC) enabled, plus a HEC token and a target index.
- The Splunk HEC endpoint host and port (the HEC default port is `8088`), reachable from the cluster and NetScaler.
- `kubectl` and Helm 3.x, with permission to create secrets and ConfigMaps in the target namespace.

NOTE:

Create the Splunk token secret and the ConfigMap for NetScaler CPX custom metrics in the same namespace as the controller or CPX, and set `authToken.namespace` to that namespace.

Configuration overview

Observability is configured in two parts:

PART	WHERE	REQUIRED
Splunk server connection	<code>analyticsConfig</code> in Helm <code>values.yaml</code>	Mandatory

PART	WHERE	REQUIRED
Per-route analytics profile	<code>aigatewayroute.citrix.com/analytics-profile</code> annotation on <code>AIGatewayRoute</code>	Optional — overrides global config per route

Global versus route-level profiles

PROFILE TYPE	CONFIGURED IN	SCOPE
Time series metrics	<code>analyticsConfig.splunk.servers[].profiles.timeseries</code> in <code>values.yaml</code>	Global only — cannot be per-route
Stream insight	<code>analyticsConfig.splunk.servers[].profiles.streaminsight</code> in <code>values.yaml</code>	Global default
Per-route override	<code>aigatewayroute.citrix.com/analytics-profile</code> annotation	Per <code>AIGatewayRoute</code>

Create the Splunk token secret

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

The chart reads the token from the `splunk-token` key of this secret. Use your Splunk HEC token as the value. The `authToken` reference in `analyticsConfig` (below) points to this secret by `name` and `namespace`.

Configure the Splunk connection in Helm values (mandatory)

The `analyticsConfig` section in `values.yaml` of Helm chart creates the Splunk server connection and default analytics profiles:

```
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"
```

NOTE:

Setting `logAllJsonFields: "true"` enables all counters for transaction-type metrics, ensuring AI Gateway-specific details are included in the exported data. However, it exports all JSON fields on every transaction and can significantly increase Splunk ingest and licensing cost. Enable it when you need full AI metrics; otherwise leave it disabled.

After editing `values.yaml`, apply it to update the deployment:

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

Alternatively, add `-f values.yaml` to the initial `helm install` command.

Override the analytics profile per route

Use the `aigatewayroute.citrix.com/analytics-profile` annotation on an `AIGatewayRoute` to customize the collector and insight type for specific model routes. Add the annotation to the `AIGatewayRoute` metadata, as shown in this complete example:

```
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

```

IMPORTANT:

The server name in the annotation's `collectors` field (for example, `"my-splunk"`) must match the server `name` defined in `analyticsConfig` in `values.yaml`. The `analyticsConfig` section in `values.yaml` creates the server connection; the annotation references it by name.

Custom AI Gateway metrics

To export AI Gateway-specific metrics counters, create a custom metrics configuration file named `metrics.json`. Build this file using the AI Gateway counters listed in [AI Gateway observability](#). The way you add this file to NetScaler depends on the form factor (NetScaler CPX, the containerized NetScaler that runs in the cluster, or NetScaler VPX/MPX/BLX, the virtual appliance that runs outside the cluster).

NetScaler CPX

For NetScaler CPX (the containerized NetScaler), add the `metrics.json` file to the CPX container using a ConfigMap. The `metrics.json` file is available at <https://raw.githubusercontent.com/netScaler/netScaler-k8s-ingress-controller/refs/heads/master/example/metrics.json>.

Create a ConfigMap from the `metrics.json` file:

```

kubectl create configmap my-custom-metrics-configmap \

  --from-file=metrics.json -n <namespace>

```

Configure `extraVolumeMounts` and `extraVolumes` in `values.yaml` to mount the ConfigMap into the CPX container:

```
extraVolumeMounts:

  - name: custom-metrics-conf

    mountPath: /cpx/custom_metrics_conf


extraVolumes:

  - name: custom-metrics-conf

    configMap:

      name: "my-custom-metrics-configmap"

      defaultMode: 420
```

After editing `values.yaml`, apply it to update the deployment:

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

NetScaler VPX/MPX/BLX

For NetScaler VPX/MPX/BLX (the virtual appliance running outside the cluster), place the `metrics.json` file directly inside NetScaler VPX/MPX/BLX as described in [AI Gateway observability](#). The `metrics.json` file is available at <https://raw.githubusercontent.com/netScaler/netScaler-k8s-ingress-controller/refs/heads/master/example/metrics.json>.

Verify

After you apply the configuration, confirm that metrics are reaching Splunk:

- In Splunk, search the configured index or endpoint (for example, a source containing `netScaler/metric`).
- Within one `metricsexportfrequency` interval (60 seconds in the example), you should see per-request AI events that include token counts and model identity.

Reference

[AI Gateway observability](#)

NetScaler CRDs

The following topics are available under NetScaler CRDs:

- [AI Gateway profile](#)
- [AI Gateway route](#)
- [Rate limit](#)
- [Rewrite and responder policies](#)

AI Gateway profile

The `aigatewayprofile` custom resource definition (CRD) lets you define an AI Gateway profile on NetScaler® using Kubernetes-native configuration. An AI Gateway profile carries the settings that NetScaler applies when it processes AI traffic between your applications and large language model (LLM) backends.

You can use the `aigatewayprofile` CRD to:

- Apply a **frontend** profile to the client-facing load balancing virtual server that receives inference requests.
- Apply a **backend** profile to the service groups that connect to the upstream LLM providers, with a per-model token quota and a quota refresh frequency.
- Reference the authentication token (API key) for an AI endpoint from a Kubernetes Secret.
- Reuse a preconfigured, built-in AI Gateway profile that already exists on NetScaler.

A profile is attached to a `GatewayClass` and is consumed by an `aigatewayroute` through its `aigatewayprofiles` field.

NOTE:

The `preconfigured` and `config` fields are mutually exclusive. You must specify one of them.

Deploy the aigatewayprofile CRD

Download the [aigatewayprofile CRD](#) and deploy it using the following command:

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

NOTE:

Do not modify the CRD deployment YAML file.

aigatewayprofile CRD attributes

The following table lists the attributes available under `spec` for the `aigatewayprofile` CRD.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>gatewayClassName</code>	Name of the <code>GatewayClass</code> that this AI Gateway profile is applied to. The profile is applied only when this value matches a <code>GatewayClass</code> managed by the controller. This field is required.	String
<code>preconfigured</code>	Name of a preconfigured, built-in AI Gateway profile on NetScaler to use. Mutually exclusive with <code>config</code> .	String
<code>config</code>	Contains all the parameters related to NetScaler for a fully custom AI Gateway profile. Mutually exclusive with <code>preconfigured</code> .	Object

config attributes

When you define a custom profile, the following attributes are available under `spec.config`.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>profiletype</code>	Type of AI Gateway profile. This value is immutable after it is set.	<code>backend</code> , <code>frontend</code>

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>endpointtype</code>	Type of the AI endpoint. This value is immutable after it is set.	<code>azureopenai</code>
<code>tokenquota</code>	Token quota limit for backend profiles. Applicable only when <code>profiletype</code> is <code>backend</code> .	Integer (0–2147483647)
<code>quotarefreshfrequency</code>	Quota refresh rate, in minutes. Applicable only when <code>profiletype</code> is <code>backend</code> .	Integer (1–65535)
<code>authtoken</code>	Authentication token or API key for the AI Gateway endpoint, sourced from a Kubernetes Secret. This value is immutable after it is set. Applicable only when <code>profiletype</code> is <code>backend</code> .	Object

authtoken attributes

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>name</code>	Name of the Kubernetes Secret object that contains the authentication token. This field is required.	String matching <code>^a-z0-9?\$</code>
<code>namespace</code>	Namespace of the Kubernetes Secret object. Defaults to the namespace of the AI Gateway profile.	String matching <code>^a-z0-9?\$</code>

How to write the profile configuration

In the `aigatewayprofile` YAML definition, set the `kind` as `aigatewayprofile`. In the `spec` section, add the `gatewayClassName` and either a `config` block (for a custom profile) or a `preconfigured` value (to reuse a built-in profile).

Keep the following guidelines in mind:

- Specify exactly one of `config` or `preconfigured`.
- Set `tokenquota` only on profiles where `profiletype` is `backend`.
- The `profiletype`, `endpointtype`, and `authtoken` values cannot be changed after the profile is created.
- Store the API key in a Kubernetes Secret and reference it through `authtoken`.

Sample profile configurations

Frontend profile

The following configuration creates a client-facing AI Gateway profile.

```
apiVersion: citrix.com/v1

kind: aigatewayprofile

metadata:

  name: frontend-profile

  namespace: default

spec:

  gatewayClassName: aigwclass

  config:

    profiletype: frontend

    endpointtype: azureopenai
```

Backend profile with a token quota

The following configuration creates a backend AI Gateway profile that enforces a token quota that is refreshed every 60 minutes.

```
apiVersion: citrix.com/v1

kind: aigatewayprofile

metadata:

  name: backend-profile

  namespace: default

spec:

  gatewayClassName: aigwclass

  config:

    profiletype: backend

    endpointtype: azureopenai

    tokenquota: 100000

    quotarefreshfrequency: 60

    authtoken:

      name: azure-openai-key

      namespace: default
```

Preconfigured profile

The following configuration reuses a built-in AI Gateway profile that already exists on NetScaler.

```
apiVersion: citrix.com/v1

kind: aigatewayprofile

metadata:

  name: builtin-profile
```

```
namespace: default

spec:

  gatewayClassName: aigwclass

  preconfigured: default-ai-frontend
```

Related articles

- [AI Gateway overview](#)
- [Configure model routing](#)
- [AI Gateway route](#)
- [AI capabilities on NetScaler](#)

AI Gateway route

The `aigatewayroute` custom resource definition (CRD) combines multiple AI service backends and attaches them to `Gateway` resources managed by NetScaler® Kubernetes Gateway Controller. It provides a way to define a *unified* AI API for a `Gateway`, so that downstream clients can use a single schema to interact with multiple AI backends across providers.

NetScaler Kubernetes Gateway Controller provides the following capabilities for an

`aigatewayroute`:

- Header-based and policy-based routing to AI backends through the `HTTPRoute` of the Gateway API. The generated `HTTPRoute` has the same name as the `aigatewayroute`.
- Priority-based model spillover to backup backends, which enables automatic failover when the primary AI backends are unavailable or overloaded.
- Separate front end and back end AI Gateway profiles for multi-provider deployments, which allows independent configuration of client-facing and upstream AI service policies.
- Support for `ratelimit` and `rewritepolicy` CRDs for fine-grained traffic control and request or response transformation.
- Support for `sslprofile`, `tcpprofile`, and `httpprofile` CRDs for protocol-level tuning of AI backend connections.
- Built-in TCP, HTTP, and streaming analytics for observability of AI traffic patterns, token usage, and backend performance.

All of these resources are created in the same namespace as the `aigatewayroute`. You can customize the default behavior by attaching NetScaler policy CRDs to the generated `HTTPRoute` through `ExtensionRef` filters.

NOTE:

In the matching conditions, the `x-ai-eg-model` header is available for routing decisions based on the model name. The model name is extracted from the request content before routing.

Deploy the aigatewayroute CRD

Download the [aigatewayroute CRD](#) and deploy it using the following command:

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

NOTE:

Do not modify the CRD deployment YAML file.

aigatewayroute CRD attributes

The following table lists the top-level attributes available under `spec` for the `aigatewayroute` CRD.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>rules</code>	List of routing rules that this route matches the traffic to. This field is required.	Array of rules (maximum 128)
<code>aigatewayprofiles</code>	Default AI Gateway profiles to apply to the generated load balancing virtual servers and service groups.	Object
<code>hostnames</code>	Set of hostnames matched against the HTTP <code>Host</code> header to select this route.	Array of RFC 1123 hostnames (maximum 16)

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
	Wildcard prefixes (<code>*.</code>) are interpreted as a suffix match. IP addresses are not allowed.	
<code>parentRefs</code>	References to the resources (usually <code>Gateway</code> objects) that this route is attached to.	Array (maximum 32)

aigatewayprofiles attributes

Each of the following profile references specifies a `name` (required) and an optional `namespace` that defaults to the `aigatewayroute` namespace.

ATTRIBUTE	DESCRIPTION
<code>frontend</code>	AI Gateway profile applied to the primary load balancing virtual server.
<code>spillover-frontend</code>	AI Gateway profile applied to the spillover load balancing virtual server.
<code>default-backend</code>	AI Gateway profile applied to all backend service groups unless overridden by a per-backendRef <code>backend-aigatewayprofile</code> .

parentRefs attributes

The following table lists `parentRefs` attributes.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>name</code>	Name of the referent. This field is required.	String
<code>group</code>	Group of the referent. When unspecified, <code>gateway.networking.k8s.io</code> is inferred.	String
<code>kind</code>	Kind of the referent.	Default: <code>Gateway</code>

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>namespace</code>	Namespace of the referent. Defaults to the local namespace of the route.	String
<code>port</code>	Network port that this route targets.	Integer (1–65535)
<code>sectionName</code>	Name of a section (listener name for a <code>Gateway</code>) within the target resource.	String

rules attributes

Each entry in `rules` supports the following attributes.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>backendRefs</code>	List of AI service backends that this rule routes traffic to.	Array (maximum 128)
<code>matches</code>	List of conditions that determine when this rule applies. Supports path-based, header-based, and policy expression matching.	Array (maximum 128)
<code>filters</code>	NetScaler policy CRDs to attach to this rule through <code>ExtensionRef</code> .	Array (maximum 16)
<code>timeouts</code>	Timeouts that can be configured for an HTTP request.	Object

backendRefs attributes

The following table lists `backendRefs` attributes.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>name</code>	Name of the backend <code>Service</code> resource. This field is required.	String
<code>group</code>	API group of the referent. When empty, the core API group is inferred.	String
<code>kind</code>	Kubernetes resource kind of the referent.	Default: <code>Service</code>
<code>namespace</code>	Namespace of the backend resource. Defaults to the local namespace of the route.	String
<code>port</code>	Destination port number to use for this backend.	Integer (1–65535)
<code>weight</code>	Proportion of traffic to send to this backend relative to other backends in the same rule.	Integer (default 1)
<code>is_spillover</code>	Indicates whether this backend is a spillover backend that receives traffic only when the primary backends are unavailable or overloaded. Only one backendRef may set this to <code>true</code> .	Boolean (default <code>false</code>)
<code>backend-aigatewayprofile</code>	Backend AI Gateway profile to apply to this backend's service group. Overrides the spec-level <code>default-backend</code> profile.	Object (<code>name</code> , <code>namespace</code>)
<code>filters</code>	Per-backend NetScaler policy CRDs to attach through <code>ExtensionRef</code> .	Array (maximum 16)

matches attributes

The following table lists matches attributes.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>path</code>	HTTP request path matcher. Defaults to a <code>PathPrefix</code> match on <code>/</code> .	<code>type</code> : <code>Exact</code> , <code>PathPrefix</code> , <code>RegularExpression</code> ; <code>value</code> : string
<code>headers</code>	HTTP request header matchers. Use AND for multiple values.	<code>name</code> , <code>value</code> , <code>type</code> : <code>Exact</code> , <code>RegularExpression</code>
<code>policyExpression</code>	NetScaler policy expression for advanced matching, evaluated in addition to header matches.	String

filters attributes

The following table lists filters attributes.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>type</code>	Type of filter. Currently only <code>ExtensionRef</code> is supported.	<code>ExtensionRef</code>
<code>extensionRef</code>	Reference to a NetScaler policy CRD (<code>group</code> , <code>kind</code> , <code>name</code> — all required). Supported kinds include <code>ratelimit</code> , <code>rewritepolicy</code> , <code>aigatewayprofile</code> , <code>sslprofile</code> , <code>tcpprofile</code> , and <code>httpprofile</code> .	Object

timeouts attributes

The following table lists timeouts attributes.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>request</code>	Maximum duration for the gateway to respond to an HTTP request.	Duration, for example <code>30s</code>
<code>backendRequest</code>	Timeout for an individual request from the gateway to a backend. Cannot be longer than <code>request</code> .	Duration, for example <code>20s</code>

NOTE:

For streaming responses (such as chat completions with `stream=true`), set longer timeouts because the response can take time to complete.

How to write the route configuration

In the `aigatewayroute` YAML definition, set the `kind` as `aigatewayroute`. In the `spec` section, add the `parentRefs` that attach the route to a `Gateway`, the `aigatewayprofiles` that select the front end and back end profiles, and one or more `rules` that match and route traffic.

Keep the following guidelines in mind:

- The `rules` field is mandatory.
- Backends with lower priority values are preferred. Set `is_spillover: true` in maximum one backend in a rule to designate the failover backend.
- Use the `x-ai-eg-model` header or `policyExpression` in `matches` to route by model name.
- Attach policy CRDs (`ratelimit`, `rewritepolicy`, and so on) at the rule level or per-backend through `ExtensionRef` filters.

Sample route configurations

Model routing to multiple backends

The following configuration routes requests to different backends based on the model name carried in the `x-ai-eg-model` header.

```
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
```

Priority-based model spillover

The following configuration sends traffic to a primary backend and spills over to a backup backend when the primary is unavailable or overloaded.

```
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
```

Attach a rate limit policy to a rule

The following configuration attaches a `ratelimit` CRD to a routing rule through an `ExtensionRef` filter.

```
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
```

Related articles

- [AI Gateway overview](#)
- [Configure model routing](#)
- [Configure spillover to an alternate model](#)
- [AI Gateway profile](#)
- [Rate limit](#)
- [Rewrite and responder policies](#)

Rate limit

The `ratelimit` custom resource definition (CRD) lets you configure rate limiting and throttling policies on NetScaler® using Kubernetes-native configuration. You can use the `ratelimit` CRD to protect your services and AI backends from overload by limiting the rate of requests, connections, or tokens.

You can use the `ratelimit` CRD to:

- Throttle traffic based on request rate, connection count, or token rate.
- Apply limits per client IP address, per API path, per HTTP method, or per API client identified by an HTTP header.
- Choose the action to take when a limit is exceeded, such as dropping, resetting, redirecting, or responding to requests.
- Collect traffic analytics with stream identifiers for observability of requests, connections, response times, bandwidth, and token usage.

The `ratelimit` CRD is referenced from an `aigatewayroute` through an `ExtensionRef` filter, or applied directly to services through `servicenames` or `targetRef`.

NOTE:

You must specify at least one of `ratelimits` or `streams` .

Deploy the ratelimit CRD

Download the [ratelimit CRD](#) and deploy it using the following command:

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

NOTE:

Do not modify the CRD deployment YAML file.

ratelimit CRD attributes

NOTE:

In NetScaler Kubernetes Gateway Controller 2.2.0 and later releases, attributes of `ratelimit` CRD schema are updated to be under the `ratelimits` spec. Also, new rate limit attributes are introduced. In previous releases, `ratelimit` CRD schema was defined directly under `spec` .

The following table lists the top-level attributes available under `spec` for the `ratelimit` CRD.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>ingressclass</code>	Ingress class. If not specified, all NetScaler Ingress Controllers in the cluster process the resource. Otherwise, only the controller with that ingress class processes it.	String

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>servicenames</code>	Name of the services to which the rate limit policies are applied.	Array of strings (maximum length 127)
<code>gatewayClassName</code>	Name of the <code>GatewayClass</code> that this profile is applied to.	String
<code>targetRef</code>	List of target resources where this profile is applied.	Array
<code>selector_keys</code>	Traffic match criteria to which the rate limit or throttling applies.	Object
<code>ratelimits</code>	List of rate limit configurations.	Array
<code>streams</code>	List of stream identifier configurations for traffic analytics and tracking.	Array

targetRef attributes

ATTRIBUTE	DESCRIPTION
<code>name</code>	Name of the target resource.
<code>namespace</code>	Namespace of the target resource.
<code>group</code>	Group of the target resource.
<code>kind</code>	Kind of the target resource.
<code>sectionName</code>	Specific section within the target resource.

selector_keys attributes

The `selector_keys.basic` object defines the traffic stream selection criteria. All keys are applied as an AND condition. If no keys are specified, the rate limit applies at the service level.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>path</code>	API resource path prefix match, for example <code>/api/v1/products</code> .	Array of strings
<code>method</code>	HTTP methods to match.	<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>	HTTP header that identifies the unique API client, for example <code>X-apikey</code> .	String
<code>per_client_ip</code>	When set, applies the throttling limit to each unique client IP address accessing the API resource.	Boolean

ratelimits attributes

Each entry in `ratelimits` supports the following attributes. The `req_threshold` attribute is required.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>req_threshold</code>	Maximum requests per timeslice to be allowed. This field is required.	Integer
<code>timeslice</code>	Timeslice in milliseconds, in multiples of 10. Defaults to 1000 milliseconds.	Integer
<code>limittype</code>	Limit type. Defaults to <code>SMOOTH</code> if not specified.	<code>BURSTY</code> , <code>SMOOTH</code>
<code>mode</code>	Metric to which the limit is applied.	<code>REQUEST_RATE</code> , <code>CONNECTION</code> , <code>TOKEN_RATE</code>
<code>alertsintimeslice</code>	Number of alerts to raise within a timeslice.	Integer

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>throttle_action</code>	Action to take when the limit is exceeded. <code>DROP</code> drops requests exceeding the limit, <code>RESET</code> resets the client connection, <code>REDIRECT</code> redirects to the specified URL, <code>RESPOND</code> responds with <code>429 Exceeded allowed rate of requests</code> .	<code>DROP</code> , <code>RESET</code> , <code>REDIRECT</code> , <code>RESPOND</code> , <code>NOOP</code>
<code>redirect_url</code>	Redirect URL used when <code>throttle_action</code> is <code>REDIRECT</code> .	String
<code>logpackets</code>	Adds an audit message action that specifies whether to log the message, and to which log.	Object (<code>logexpression</code> , <code>loglevel</code>)

The `logpackets` object supports the following attributes. Both are required when `logpackets` is used.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>logexpression</code>	Default-syntax expression that defines the format and content of the log message.	String (maximum length 7991)
<code>loglevel</code>	Audit log level that specifies the severity of the generated log message.	<code>EMERGENCY</code> , <code>ALERT</code> , <code>CRITICAL</code> , <code>ERROR</code> , <code>WARNING</code> , <code>NOTICE</code> , <code>INFORMATIONAL</code> , <code>DEBUG</code>

streams attributes

Each entry in `streams` supports the following attributes. The `sort` attribute is required.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>interval</code>	Number of minutes of data to use when calculating session statistics (number of requests, bandwidth, and response times).	Integer (1–10080)
<code>sampleCount</code>	Size of the sample from which to select a request for evaluation. To evaluate all requests, set the sample count to 1.	Integer (1–65535)
<code>sort</code>	Sort stored records by the specified statistics column, in descending order. This field is required.	<code>REQUESTS</code> (default), <code>CONNECTIONS</code> , <code>RESPTIME</code> , <code>BANDWIDTH</code> , <code>RESPTIME_BREACHES</code> , <code>TOKENS</code> , <code>NONE</code>
<code>snmpTrap</code>	Enable or disable SNMP trap for the stream identifier.	<code>ENABLED</code> , <code>DISABLED</code>
<code>appflowLog</code>	Enable or disable AppFlow logging for the stream identifier.	<code>ENABLED</code> , <code>DISABLED</code>
<code>trackAckOnlyPackets</code>	Track ACK-only packets. Applicable only when packet rate limiting is used.	<code>ENABLED</code> , <code>DISABLED</code>
<code>trackTransactions</code>	Track transactions exceeding the configured threshold. When set to <code>TOKENS</code> , the transaction threshold attributes do not apply.	<code>RESPTIME</code> , <code>TOKENS</code> , <code>NONE</code>
<code>maxTransactionThreshold</code>	Maximum per-transaction value of the tracked metric.	Integer (minimum 0)
<code>minTransactionThreshold</code>	Minimum per-transaction value of the tracked metric.	Integer (minimum 0)

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>acceptanceThreshold</code>	Non-breaching transactions to total transactions threshold, expressed as a percentage. A maximum of 6 decimal places is supported.	String (maximum length 10)
<code>breachThreshold</code>	Breaching transactions threshold calculated over the interval.	Integer (minimum 0)
<code>log</code>	Location where objects collected on the identifier are logged.	<code>SYSLOG</code> , <code>NONE</code>
<code>logInterval</code>	Time interval in minutes for logging the collected objects. Must be greater than or equal to the interval of the stream identifier.	Integer (1–10080)
<code>logLimit</code>	Maximum number of objects to be logged in the log interval.	Integer (1–1000)

How to write the rate limit configuration

In the `ratelimit` YAML definition, set the `kind` as `ratelimit`. In the `spec` section, select the traffic to throttle with `selector_keys`, and define the limits under `ratelimits`, the analytics under `streams`, or both.

Keep the following guidelines in mind:

- Specify at least one of `ratelimits` or `streams`.
- Use `mode: TOKEN_RATE` to enforce token-based rate limiting for AI backends.
- Use `selector_keys.basic.per_client_ip` or `header_name` to apply per-client limits.
- Attach the `ratelimit` CRD to an `aigatewayroute` through an `ExtensionRef` filter, or bind it to services through `servicenames` or `targetRef`.

Sample rate limit configurations

Token-based rate limiting

The following configuration limits traffic per client IP address to 100,000 tokens per minute and responds with a `429` status when the limit is exceeded.

```
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
```

Per-API-client request rate limiting with logging

The following configuration limits each API client (identified by the `X-apikey` header) to 500 requests per second and logs when the limit is breached.

```
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
```

Stream identifier for traffic analytics

The following configuration collects analytics on token usage without enforcing a limit.

```
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
```

Related articles

- [Configure rate limits and streams](#)
- [AI Gateway route](#)
- [Observability](#)
- [AI capabilities on NetScaler](#)

Rewrite and responder policies

The `rewritepolicy` custom resource definition (CRD) lets you configure rewrite and responder policies on NetScaler® using Kubernetes-native configuration. These policies let you rewrite information in requests and responses, and respond to requests based on the characteristics of the request.

You can use the `rewritepolicy` CRD for scenarios such as:

- Rewriting the URL, host name, or headers of a request or response.
- Inserting, deleting, or replacing HTTP headers.
- Redirecting requests to a different URL, responding with a custom payload, or dropping and resetting connections.
- Referencing datasets, patsets, and string maps in policy expressions.
- Making HTTP callouts to an external agent to retrieve information used in policy decisions.

The `rewritepolicy` CRD is referenced from an `aigatewayroute` through an `ExtensionRef` filter, or applied directly to services through `servicenames` or `targetRef`.

NOTE:

You must specify at least one of `rewrite-policies` or `responder-policies`.

Deploy the rewritepolicy CRD

Download the [rewritepolicy CRD](#) and deploy it using the following command:

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

NOTE:

Do not modify the CRD deployment YAML file.

rewritepolicy CRD attributes

The following table lists the top-level attributes available under `spec` for the `rewritepolicy` CRD.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>ingressclass</code>	Ingress class. If not specified, all NetScaler Ingress Controllers in the cluster process the resource. Otherwise, only the controller with that ingress class processes it.	String
<code>gatewayClassName</code>	Name of the <code>GatewayClass</code> that this profile is applied to.	String
<code>targetRef</code>	List of target resources where this profile is applied (<code>name</code> , <code>namespace</code> , <code>group</code> , <code>kind</code> , <code>sectionName</code>).	Array
<code>rewrite-policies</code>	List of rewrite policies.	Array
<code>responder-policies</code>	List of responder policies.	Array
<code>dataset</code>	List of datasets referenced by policy expressions.	Array
<code>patset</code>	List of pattern sets referenced by policy expressions.	Array

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>stringmap</code>	List of string maps referenced by policy expressions.	Array
<code>httpcallout_policy</code>	List of HTTP callout configurations.	Array

Rewrite policy

Each entry in `rewrite-policies` supports the following attributes. The `rewrite-policy` object is required.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>servicenames</code>	Name of the services to bind to the rewrite policy.	Array of strings (maximum length 127)
<code>goto-priority-expression</code>	Expression or value specifying the next policy to evaluate if the current policy evaluates to <code>TRUE</code> . <code>NEXT</code> evaluates the policy with the next higher priority number, <code>END</code> ends policy evaluation. Defaults to <code>END</code> .	String (maximum length 1499)
<code>logpackets</code>	Adds an audit message action (<code>logexpression</code> , <code>loglevel</code>).	Object
<code>rewrite-policy</code>	The rewrite policy definition. This object is required.	Object

The `rewrite-policy` object supports the following attributes. The `rewrite-criteria` , `operation` , `target` , and `direction` attributes are required.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>rewrite-criteria</code>	Expression against which traffic is evaluated. Required.	String (maximum length 1299)

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>operation</code>	Type of user-defined rewrite action. Required.	<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> , and other supported operations
<code>target</code>	Default-syntax expression that specifies which part of the request or response to rewrite. Required.	String (maximum length 1229)
<code>direction</code>	Bind point to which to bind the policy. Required.	<code>REQUEST</code> , <code>RESPONSE</code>
<code>default-action</code>	Action to perform if the result of policy evaluation is undefined (UNDEF).	<code>NOREWRITE</code> , <code>RESET</code> , <code>DROP</code>
<code>modify-expression</code>	Default-syntax expression that specifies the content to insert, or that replaces the specified string.	String (maximum length 7991)
<code>multiple-occurrence-modify</code>	Search facility used to match multiple strings in the request or response.	String (maximum length 171)
<code>additional-multiple-occurrence-modify</code>	Additional criteria to refine the search results. Starts with the <code>extend(m,n)</code> operation.	String (maximum length 1299)
<code>comment</code>	Any comments to preserve information about this rewrite policy.	String (maximum length 255)

Responder policy

Each entry in `responder-policies` supports the following attributes. The `responder-policy` object is required.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>servicenames</code>	Name of the services to bind to the responder policy.	Array of strings (maximum length 127)
<code>goto-priority-expression</code>	Expression or value specifying the next policy to evaluate if the current policy evaluates to <code>TRUE</code> . Defaults to <code>END</code> .	String (maximum length 1499)
<code>logpackets</code>	Adds an audit message action (<code>logexpression</code> , <code>loglevel</code>).	Object
<code>responder-policy</code>	The responder policy definition. This object is required.	Object

The `responder-policy` object supports the following attributes. The `respond-criteria` attribute is required, and exactly one of `redirect` , `respondwith` , `noop` , `reset` , or `drop` must be specified.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>respond-criteria</code>	Default-syntax expression that the policy uses to determine whether to respond to the request. Required.	String (maximum length 1299)
<code>redirect</code>	Redirect the request when it matches the policy (<code>url</code> required, <code>redirect-status-code</code> , <code>redirect-reason</code>).	Object

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>respondwith</code>	Respond to the request when it matches the policy (<code>http-payload-string</code> required).	Object
<code>noop</code>	Send the request to the protected server instead of responding to it.	Object
<code>reset</code>	Reset the client connection by closing it when the request matches the policy.	Object
<code>drop</code>	Drop the request without sending a response when it matches the policy.	Object
<code>default-action</code>	Action to perform if the result of policy evaluation is undefined (UNDEF).	<code>NOOP</code> , <code>RESET</code> , <code>DROP</code>
<code>comment</code>	Any comments to preserve information about this responder policy.	String (maximum length 255)

Dataset

Each entry in `dataset` supports the following attributes. The `name` , `type` , and `values` attributes are required.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>name</code>	Name of the dataset.	String (maximum length 32)
<code>type</code>	Type of value to bind to the dataset.	<code>ipv4</code> , <code>number</code> , <code>ipv6</code> , <code>ulong</code> , <code>double</code> , <code>mac</code>
<code>values</code>	Values of the specified type associated with this dataset.	Array of strings

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>comment</code>	Any comments to preserve information about this dataset.	String (maximum length 255)

Patset

Each entry in `patset` supports the following attributes. The `name` and `values` attributes are required.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>name</code>	Name of the patset.	String (maximum length 32)
<code>values</code>	Strings of characters that constitute the patterns associated with this patset.	Array of strings
<code>comment</code>	Any comments to preserve information about this patset.	String (maximum length 255)

String map

Each entry in `stringmap` supports the following attributes. The `name` and `values` attributes are required.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>name</code>	Name of the string map.	String (maximum length 32)
<code>values</code>	List of key-value pairs to bind to this string map. Each pair has a <code>key</code> and a <code>value</code> (each with a maximum length of 2047 characters).	Array of objects
<code>comment</code>	Any comments to preserve information about this string map.	String (maximum length 255)

HTTP callout

Each entry in `httpcallout_policy` supports the following attributes. The `name`, `server_ip`, and `server_port` attributes are required.

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>name</code>	HTTP callout name.	String (maximum length 32)
<code>server_ip</code>	IP address of the callout agent server to which the callout is sent.	String
<code>server_port</code>	Port of the callout agent server.	Integer (1–65535)
<code>http_method</code>	Method used in the HTTP request. Defaults to <code>GET</code> .	<code>GET</code> , <code>POST</code>
<code>host_expr</code>	String expression to configure the <code>Host</code> header. Mutually exclusive with the full HTTP request expression.	String (maximum length 255)
<code>url_stem_expr</code>	String expression for generating the URL stem.	String (maximum length 8191)
<code>headers</code>	One or more headers to insert into the HTTP request (<code>name</code> , <code>expr</code>). Maximum of eight headers.	Array of objects
<code>parameters</code>	One or more query parameters to insert into the request (<code>name</code> , <code>expr</code>).	Array of objects
<code>body_expr</code>	Advanced string expression for generating the body of the request.	String
<code>full_req_expr</code>	Exact HTTP request, as an expression, that NetScaler sends to the callout agent.	String

ATTRIBUTE	DESCRIPTION	SUPPORTED VALUES
<code>scheme</code>	Scheme for the callout server. Defaults to <code>HTTP</code> .	<code>HTTP</code> , <code>HTTPS</code>
<code>cache_for_secs</code>	Duration, in seconds, for which the callout response is cached.	Integer (1–31536000)
<code>return_type</code>	Type of data that the callout agent returns. <code>TEXT</code> treats the value as text, <code>NUM</code> as a number, <code>BOOL</code> as a Boolean.	<code>TEXT</code> , <code>NUM</code> , <code>BOOL</code>
<code>result_expr</code>	Expression that extracts the callout results from the response. Must be a response-based expression (begins with <code>HTTP.RES</code>).	String (maximum length 8191)
<code>comment</code>	Any comments to preserve information about this HTTP callout.	String (maximum length 255)

How to write the policy configuration

In the `rewritepolicy` YAML definition, set the `kind` as `rewritepolicy`. In the `spec` section, add one or more of the following sections: `rewrite-policies` , `responder-policies` , `dataset` , `patset` , `stringmap` , and `httpcallout_policy` .

Keep the following guidelines in mind:

- Specify at least one of `rewrite-policies` or `responder-policies` .
- For a rewrite policy, the `rewrite-criteria` , `operation` , `target` , and `direction` attributes are mandatory.
- For a responder policy, the `respond-criteria` attribute is mandatory, and you must specify exactly one action out of `redirect` , `respondwith` , `noop` , `reset` , or `drop` .
- Attach the `rewritepolicy` CRD to an `aigatewayroute` through an `ExtensionRef` filter, or bind it to services through `servicenames` or `targetRef` .

Sample policy configurations

Rewrite policy to add a response header

The following configuration inserts a custom header into the HTTP response.

```
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"
```

Responder policy to redirect a request

The following configuration redirects matching requests to another URL.

```
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"
```

Responder policy with a custom response

The following configuration responds directly to matching requests with a custom payload.

```
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"
```

Rewrite policy with a dataset

The following configuration defines a dataset and uses it in a rewrite policy.

```
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"
```

Related articles

- [Configure prompt management](#)
- [Configure spillover to an alternate model](#)
- [AI Gateway route](#)
- [Rewrite and Responder policies on NetScaler](#)

fix

This is the content for the sample heading topic. It is hidden from the main table of contents but remains accessible via links.