NetScaler® Kubernetes Gateway Controller

Contents

Release notes

September 27, 2025

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 1.2.0
- Release 1.1.0
- Release 1.0.0

Release 1.2.0

September 27, 2025

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.

Release 1.1.0

September 27, 2025

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

September 27, 2025

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.

NetScaler® Kubernetes Gateway Controller

September 27, 2025

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

September 27, 2025

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

Prerequisites

- You have installed Kubernetes cluster.
- You have installed Helm version 3.x or later. To install Helm, see here.
- You have Installed the Gateway API CRDs from Gateway API CRDs from the official standard channel.

- You have a system user account on NetScaler. NetScaler Kubernetes Gateway Controller uses this account to push configuration to NetScaler. For more information, see Create a system user account for NetScaler Ingress Controller in NetScaler.
- You have a Kubernetes secret to store NetScaler user account credentials. See How to use Kubernetes secrets for storing NetScaler credentials.

How to deploy NetScaler® Kubernetes Gateway Controller in Kubernetes cluster

1. Add the NetScaler Helm chart repository to your local registry by using the following command:

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

   If the NetScaler Helm chart repository is already added to your local registry, use the following command to update the repository:

```
helm repo update netscaler
```

2. Install NetScaler Kubernetes Gateway Controller by using the following command.

```
helm install gateway-controller netscaler/netscaler-kubernetes-gateway-controller --set gatewayController.gatewayControllerName=citrix.com/nsgc-controller,license.accept=yes,gatewayController.entityPrefix=gwy
```

3. Verify the installation.

```
kubectl get pods
```

4. (Optional) View the summary of 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 netscaler-kubernetes-gateway-controller
```

What's next

Deploy NetScaler Kubernetes Gateway Controller.

NetScaler® Kubernetes gateway controller CRDs

September 27, 2025

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
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 vitrual 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. Currently, the controller supports a single gateway reference.	Partially Supported
spec.parentRefs[].kind	Currently, 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

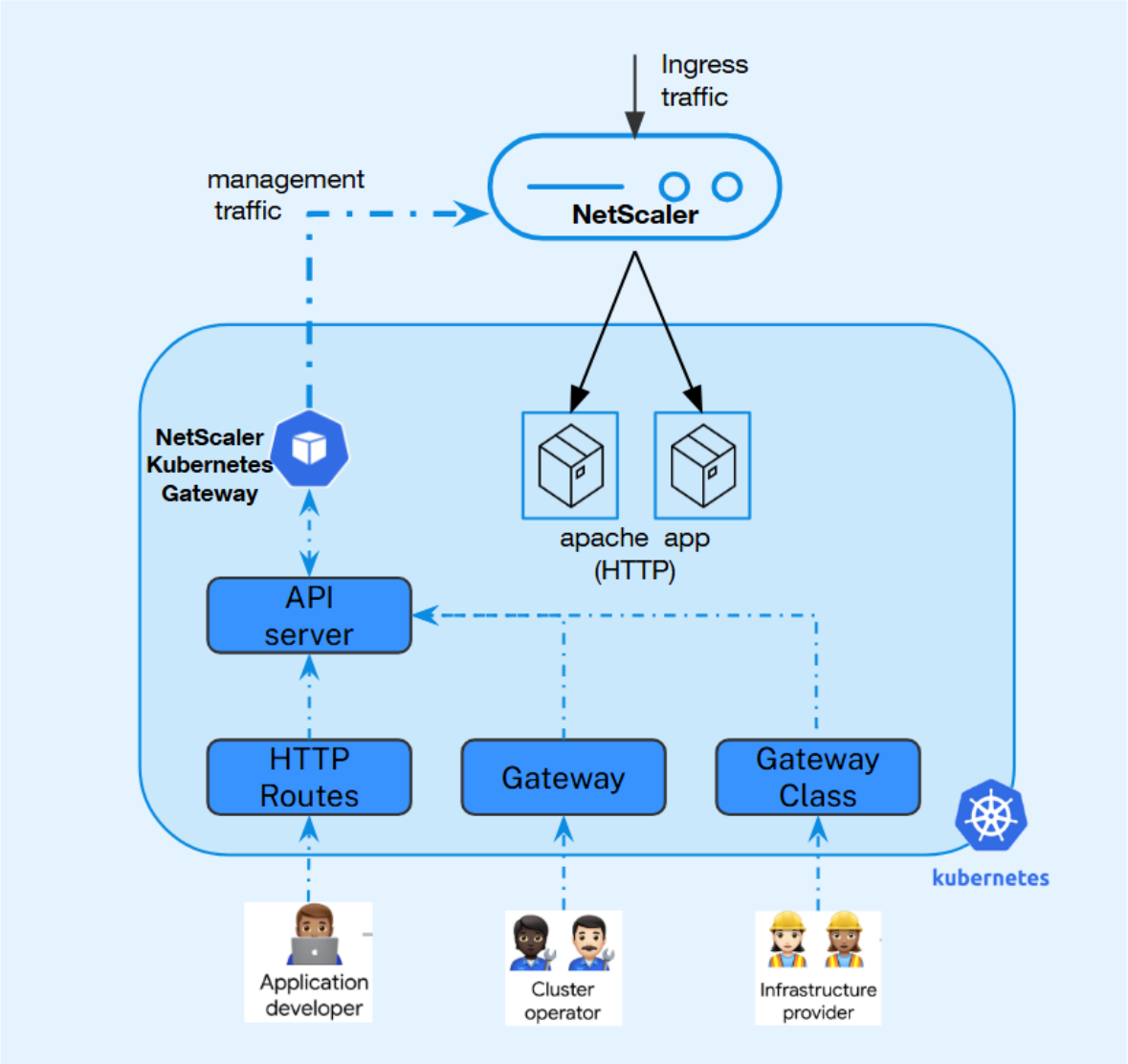
Points to note

- The controller considers only the 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.

Deploy NetScaler® Kubernetes Gateway Controller

September 27, 2025

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.

```yaml
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.

```
netscaler:
  adcCredentialSecret: "nslogin"
  nsIP: "4.4.4.4"
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
```

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/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>
```

Multiple listeners with HTTP routes

September 27, 2025

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
```



```
35            name: my-secret
36            group: ""
37        allowedRoutes:
38          namespaces:
39            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

September 27, 2025

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, appqoepolicy.

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 `appqoepolicy`.

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.


```yaml
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.

```yaml
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
```

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

September 27, 2025

NetScaler Kubernetes Gateway Controller now supports certificate bundle (`certkeybundle`) functionality, which is supported on 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 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 fails if the following order is not met:
    - Server certificate (SC) must be placed at the top of the bundle file.
    - IC[1-9] are intermediate certificates. IC[i] is issued by IC[i+1]. 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.