FortiADC Kubernetes Controller 3.1.0

November 3, 2025
FortiADC Kubernetes Controller 3.1.0

TABLE OF CONTENTS

Change Log

Date	Change Description
January 2, 2026	Initial release

About FortiADC Kubernetes Controller

The **FortiADC Kubernetes Controller** integrates FortiADC's application delivery and security features with Kubernetes environments. It allows administrators to manage FortiADC virtual servers, real server pools, and GLB directly from Kubernetes by defining either standard *Ingress* resources or Fortinet's *VirtualServer*, *RemoteServer*, and *Host*.

Operating as an intermediary between the Kubernetes API and the FortiADC REST API, the controller continuously monitors cluster resources and synchronizes corresponding configurations on FortiADC. When services or pods are added, removed, or updated, the controller automatically reconciles these changes to maintain an accurate, up-to-date application delivery configuration.

Key capabilities include:

- **Automated configuration management** – Dynamically maps Kubernetes resources to FortiADC objects, eliminating manual updates.
- **Advanced application delivery** – Extends native Kubernetes load balancing with FortiADC features such as SSL offloading, persistence, and Layer 7/Layer 4 content routing.
- **Distributed global traffic** – Deploys FortiADC's Global Load Balance to publish services and direct traffic to suitable servers across different geographical locations.
- **Integrated application security** – Applies FortiADC's Web Application Firewall (WAF), antivirus, and DoS protection to Kubernetes workloads.
- **Operational visibility** – Provides unified traffic monitoring and analytics through FortiView and traffic logs.

This document describes the concepts, configuration, and deployment procedures for the FortiADC Kubernetes Controller. It includes information about supported Kubernetes resources, Helm-based installation, configuration parameters, examples of integrating FortiADC with Kubernetes services using *Ingress* and *VirtualServer* definitions, and use cases for FortiADC GLB using *RemoteServer* and *Host* definitions.

Prerequisite Knowledge

Kubernetes

Before you begin using FortiADC Kubernetes Controller, you will need to have prerequisite knowledge of the Kubernetes cluster, and Kubernetes Ingress, Service, Pod and Node. The terms and concepts discussed in this document is sourced directly from Kubernetes official documentation. For more information, please refer to the documents listed below:

- Kubernetes Concepts: https://kubernetes.io/docs/concepts/
- Kubernetes Ingress: https://kubernetes.io/docs/concepts/services-networking/ingress/
- Kubernetes Service: https://kubernetes.io/docs/concepts/services-networking/service/

Helm Charts

As Helm Charts are used in FortiADC Kubernetes Controller installation, you will also need to have understanding of how Helm Charts work. For more information, please refer to the documents listed below:

- Helm Charts values files: https://helm.sh/docs/chart_template_guide/values_files/
- Helm Charts Installation and upgrade from the Helm repository: https://helm.sh/docs/helm/helm_install/

Container Network Interface (CNI)

The resources below will help you understand where the Container Network Interface (CNI) fits into the Kubernetes architecture.

- Kubernetes network model: https://kubernetes.io/docs/concepts/cluster-administration/networking/
- Flannel: https://github.com/flannel-io/flannel
- Calico overlay networking: https://docs.tigera.io/calico/latest/networking/configuring/vxlan-ipip

  Calico supports two overlays types of encapsulation: VXLAN and IP in IP. Only VXLAN mode is supported in FortiADC Kubernetes Controller 3.1.
- Cert-Manager: https://cert-manager.io/docs/

  Starting with FortiADC Kubernetes Controller version 3.1, a webhook server is introduced. A cert-manager installation is required to generate the self-signed certificate used for the TLS connection between the Kubernetes API server and the webhook server.

Supported Environments and Versions

The FortiADC Kubernetes Controller has been verified to run in the Kubernetes clusters across the following environments:

Environment	Tools for building
Private cloud	kubeadm, minikube, microk8s
Public cloud	AWS EKS, Oracle OKE

Supported Release and Version

Product	Version								
FortiADC Kubernetes Controller	1.0.0	1.0.1	1.0.2	2.0.0	2.0.1	2.0.2	2.0.3	3.0.0	3.1.0
Kubernetes	1.19.8 - 1.23.x	1.19.8 - 1.24.x	1.19.8 - 1.27.x	1.19.8 - 1.27.x	1.19.8 - 1.28.x	1.19.8 - 1.30.x	1.19.8 - 1.32.x	1.19.8 - 1.33.x	1.19.8 - 1.35.x
FortiADC Version	5.4.5 - 8.x.x*								
*Some features from FortiADC Kubernetes Controller version 2.0.0 or later require FortiADC version 7.4.0 or later to support. *Calico VXLAN CNI feature support requires FortiADC Kubernetes Controller version 3.1.0 or later and requires FortiADC version 8.0.2 or later to support									

When using FortiADC Kubernetes Controller 2.0.x, the Ingress related objects on FortiADC (including virtual servers, content routing, real server pools, and real servers) will be fully managed by the Kubernetes Controller. This means that any virtual server, content routing, real server pool or real server object that is not deployed by FortiADC Kubernetes Controller will be removed automatically.

Kubernetes API Version Reference

The Kubernetes API allows you to query and manipulate the state of API objects in Kubernetes (for example: Pods, Nodes, Ingress, and Services).

For the API object, such as the Ingress object you defined in the YAML file, there will be an `apiVersion` field for Kubernetes to determine which API version to deploy the object. Different API versions may have different metadata and specific definitions for that object.

For example, for the Ingress API object, the API versions extensions/v1beta1/Ingress and networking.k8s.io/v1/Ingress would have different data structures for FortiADC Kubernetes Controller to parse. Before extensions/v1beta1/Ingress is entirely deprecated by networking.k8s.io/v1/Ingress in Kubernetes v1.22, you may find both API versions are supported by the Kubernetes API server in some cases, such as when upgrading Kubernetes from a lower version v1.16 to a higher version v1.19.

To ensure you use an API version of Kubernetes objects that FortiADC Kubernetes Controller supports, you can use the `kubectl` command to check the resource API version.

```
user@control-plane-node:~$ for kind in `kubectl api-resources | tail +2 | awk '{ print $1 }'`; do
kubectl explain $kind; done | grep -e "KIND:" -e "VERSION:"
```

The table below lists the API version of the required API object used for FortiADC Kubernetes Controller:

API Object	API Version
Node	v1
Pod	v1
PodTemplate	v1
ServiceAccount	v1
Service	v1
Deployment	apps/v1
ReplicaSet	apps/v1
DaemonSets	apps/v1
Endpoint	v1
EndpointSlice	discovery.k8s.io/v1
Event	v1
IngressClass	networking.k8s.io/v1
Ingress	networking.k8s.io/v1
ClusterRoleBinding	rbac.authorization.k8s.io/v1
ClusterRole	rbac.authorization.k8s.io/v1

API Object	API Version
RoleBinding	rbac.authorization.k8s.io/v1
Role	rbac.authorization.k8s.io/v1
*Starting with version 3.0.0, the FortiADC Kubernetes Controller uses the **EndpointSlice** resource (discovery.k8s.io/v1) instead of the legacy **Endpoint** (v1) API, aligning with Kubernetes' updated service discovery framework.	

Deploying FortiADC Kubernetes Controller

The FortiADC Kubernetes Controller runs as a pod within a Kubernetes cluster and provides dynamic synchronization between Kubernetes resources and FortiADC configuration objects. Once deployed, it continuously monitors the cluster for changes to relevant resources and automatically updates FortiADC to reflect the current application state.

Deployment Scenarios

The controller supports multiple traffic-management models to suit your architectural requirements:

- **Server Load Balancing (Ingress & VirtualServer)**: Use the familiar Kubernetes Ingress framework or Fortinet's VirtualServer Custom Resource to deliver advanced Layer 4 & Layer 7 proxying, WAF security, and antivirus scanning.
- **Global Load Balancing (GLB)**: Utilize RemoteServer and Host Custom Resources to distribute traffic across geographically dispersed servers via a DNS-based solution .

Core Deployment Components

A successful deployment integrates several key components within your cluster:

- **Controller Pod**: Manages communication between the Kubernetes API Server and FortiADC.
- **Custom Resource Definitions (CRDs)**: Extensions such as *VirtualServer*, *RemoteServer*, and *Host* that allow for granular FortiADC configuration. These are installed automatically via Helm and are required for controller operation.
- **Validating and Conversion Webhooks**: The FortiADC Kubernetes Controller utilizes webhooks to ensure the integrity and compatibility of resources within the cluster. These are automatically deployed during the Helm installation process.
  - **Validating Webhooks**: These act as a gatekeeper. When you run kubectl apply, the validating webhook checks the YAML for configuration errors or missing required fields. If the manifest is invalid, the webhook rejects the request before it reaches the FortiADC.
  - **Conversion Webhooks**: These handle the seamless upgrade and migration of Fortinet-defined Custom Resources (CRs). As new versions of the controller are released with updated CRD schemas, the conversion webhook ensures that older resources are automatically translated to the latest supported version without manual intervention.
- **cert-manager.io**: Automates the management of TLS certificates required for secure communication with the webhook server.

High-Level Deployment Workflow

To get the controller operational, follow these phases in order:

1. **Phase 1: Architecture Review**: Understand how Kubernetes objects map to FortiADC and select your preferred resource model.

   See: Architecture and Concepts on page 12
2. **Phase 2: Controller Installation**: Deploy **cert-manager** (mandatory for webhook TLS) followed by the Helm Chart installation.

   See: Controller Installation on page 49
3. **Phase 3: Network Integration (Optional)**: If using **ClusterIP** services, establish a VXLAN overlay tunnel for your specific CNI plugin.

   See: Kubernetes CNI Plugin on page 58
4. **Phase 4: Resource Configuration**: Define annotations and manifests to begin traffic synchronization.

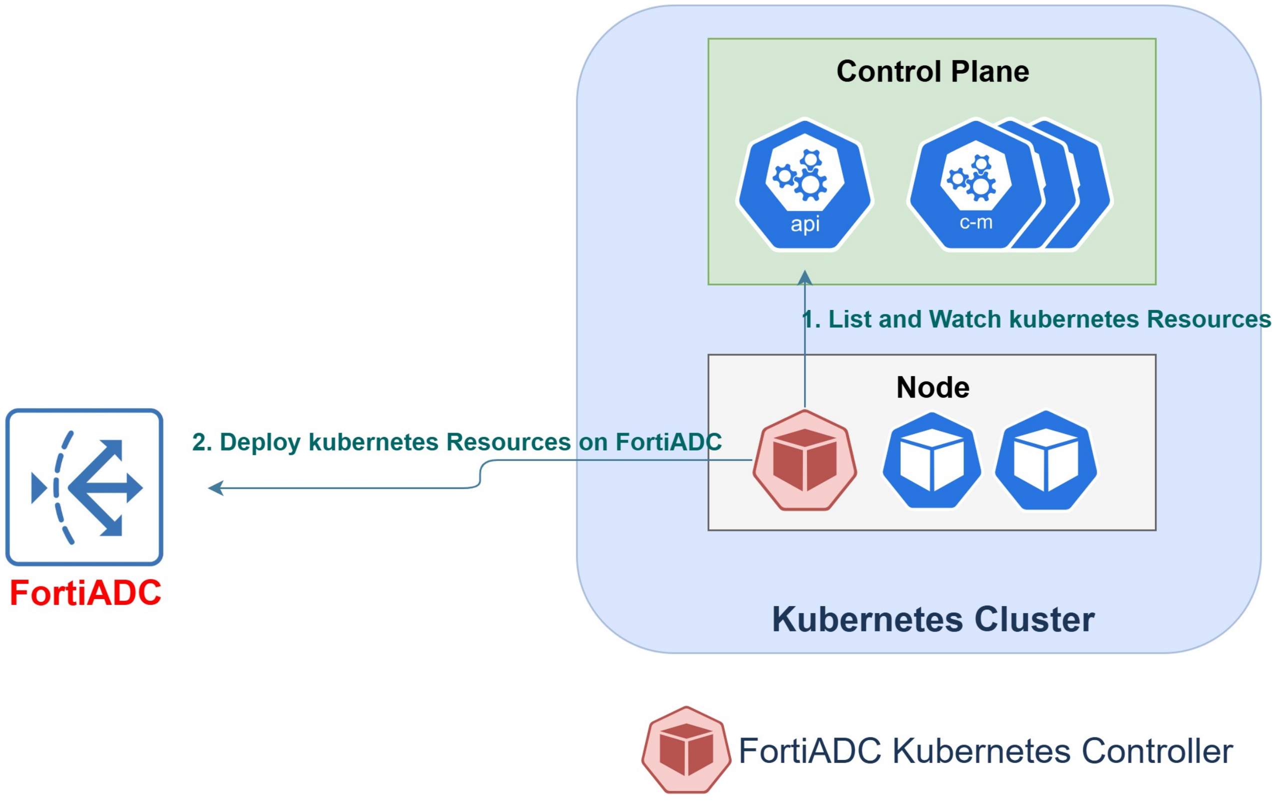
   See: Configuration Parameters on page 71

Architecture and Concepts

The FortiADC Kubernetes Controller operates as a containerized application within a Kubernetes cluster. It observes Kubernetes resources such as *Ingress*, *Service*, *EndpointSlice*, *Node*, *VirtualServer*, *RemoteServer*, and *Host*, and translates them into FortiADC configuration objects by using the FortiADC REST API. This enables continuous synchronization between cluster state and FortiADC's application delivery configuration.

Architecture Overview

The controller interacts with both the Kubernetes API Server and a FortiADC instance. It uses a Kubernetes service account to authenticate to the cluster and relies on a Kubernetes Secret to store FortiADC credentials securely.



Control Plane
api
c-m
1. List and Watch kubernetes Resources
Node
2. Deploy kubernetes Resources on FortiADC
FortiADC
Kubernetes Cluster
FortiADC Kubernetes Controller

FortiADC Kubernetes Controller is designed for a **single control plane**.

A single Kubernetes cluster or FortiADC Kubernetes Controller instance may manage **one or more FortiADC appliances**.

However, each FortiADC appliance must be managed by **one and only one** Kubernetes cluster or controller. Managing the same FortiADC appliance with multiple controllers or clusters is **not supported** and can lead to control-plane conflicts and unpredictable configuration behavior.

High-level workflow

1. **Watch and detect** – Monitors the Kubernetes API for Add, Update, and Delete events on supported resources.
2. **Translate** – Converts each resource specification into FortiADC objects such as virtual servers, content-routing rules, and real-server pools.
3. **Apply** – Pushes the translated configuration to FortiADC through the REST API.
4. **Reconcile** – Continuously compares Kubernetes resource definitions with FortiADC's active configuration and updates FortiADC whenever differences are detected.

This architecture allows FortiADC to function as a dynamic, state-aware load balancer that automatically reflects Kubernetes application topology changes.

Key Components

Component	Description
Controller Pod	Runs the FortiADC Kubernetes Controller inside the cluster and manages communication with both the Kubernetes API Server and FortiADC.
FortiADC Instance	The external FortiADC device that receives REST API calls from the controller and enforces the resulting configuration.
Service Account and RBAC	Provides the controller with permissions to read and watch Kubernetes objects such as Ingress, Service, EndpointSlice, Node, VirtualServer, RemoteServer, and Host.
Secret	Stores FortiADC login credentials securely within the cluster.
Helm Chart	Simplifies installation, upgrade, and removal of the controller and its supporting Kubernetes resources.
Custom Resource Definition (CRD)	Extends Kubernetes with Fortinet's VirtualServer, RemoteServer, and Host resource type for defining advanced traffic-management features.

Kubernetes Resource Models

The FortiADC Kubernetes Controller supports two primary methods for managing local cluster traffic, alongside a specialized model for global traffic distribution. Administrators can use these models individually or in combination depending on application complexity and operational requirements.

Server Load Balancing: The Dual-Resource Model

For traffic entering a single cluster, the controller allows you to choose between native Kubernetes objects or advanced Fortinet-defined resources:

1. **Standard Ingress (Native)**

   Leverages the native Kubernetes Ingress framework to expose HTTP and HTTPS services. This model provides broad compatibility and is ideal for standard application publishing workflows.

- **Integrated Security**: Extends native functionality by applying FortiADC's Web Application Firewall (WAF), antivirus scanning, and Denial-of-Service (DoS) protection.
- **Operational Visibility**: Enables enterprise-grade health checks, traffic logging, and FortiView analytics.

2. **VirtualServer (Custom Resource)**

   Uses the Fortinet-defined **VirtualServer CRD** to configure FortiADC-specific parameters directly within Kubernetes. When acting as a server load balancer, the controller translates these specifications into virtual servers, content routing rules, and real server pools.

   - **Advanced Routing**: Supports version `v1alpha2`, which provides L4 (TCP/UDP) and L7 (HTTP/HTTPS) content routing options.
   - **Granular Control**: Supports persistence, GSLB integration, advanced Layer 7 capabilities (such as WAF profiles, SSL settings), and Layer 4 security IPS features.

Global Load Balancing (GLB)

For traffic management across geographically dispersed clusters, the controller manages specialized GLB objects such as Servers, Data Centers, Hosts, and Virtual Server Pools:

- **RemoteServer**: Used to specify server information and monitor both FortiADC instances and third-party servers.
- **Host**: Supports a DNS-based solution for load-balancing traffic across different geographical locations.
- **GLB Priorities**: The DNS-based response is prioritized based on virtual server health, persistence, geographic or dynamic proximity, and weighted round robin.

Before deploying the FortiADC Kubernetes Controller, review the Kubernetes and Helm concepts described in Prerequisite Knowledge on page 6.

Resource Specifications

Review the following sections for the technical logic, naming conventions, and resource schemas required for deployment:

- Mapping Kubernetes Resources to FortiADC Objects on page 15: Detailed translation logic and the **Naming Rules** used for automated object creation.
- Kubernetes Ingress on page 17: Specifics on the five supported Ingress types and the required `IngressClass` configuration.
- Kubernetes Custom Resource on page 25: Field specifications and YAML examples for **VirtualServer**, **RemoteServer**, and **Host** definitions.

Mapping Kubernetes Resources to FortiADC Objects

The **FortiADC Kubernetes Controller** automatically translates Kubernetes and OpenShift resources into corresponding FortiADC configuration objects.

This mapping ensures that any changes to Kubernetes resources such as Ingress, VirtualServer, or Service are synchronized with FortiADC in real time.

The following tables show how each Kubernetes resource corresponds to a FortiADC object and describe the naming rules applied when the controller creates these objects.

Kubernetes Objects	FortiADC Objects
Ingress/ VirtualServer	Virtual server Content Routing
RemoteServer	Server Data Center
Host	Host Virtual Server Pool
Service	Real Server Pool Real Server
Node	Real Server
Endpoint/ EndpointSlice	Real Server Overlay Tunnel ARP/ Overlay Tunnel Remote/ Host
Secret	Local Certificate Local Certificate Group Client-SSL

Naming rule

For FortiADC objects created by FortiADC Kubernetes Controller, the name of the object is composed of the **namespace** and the name of the Kubernetes objects. The naming rule is shown below:

FortiADC Objects	Naming Rule
Virtual server Real Server Pool Local Certificate Local Certificate Group Client-SSL Host Virtual Server Pool	[namespace of Kubernetes objects]_[name of Kubernetes objects]

FortiADC Objects	Naming Rule
Server Data Center	
Content Routing	[namespace of Kubernetes objects]_[name of Kubernetes ingress]_[name of Kubernetes service] [namespace of Kubernetes objects]_[name of custom resource virtualserver]_[name of content routing defined in virtualserver]
Real Server	Name of the Kubernetes node or pod

Kubernetes Ingress

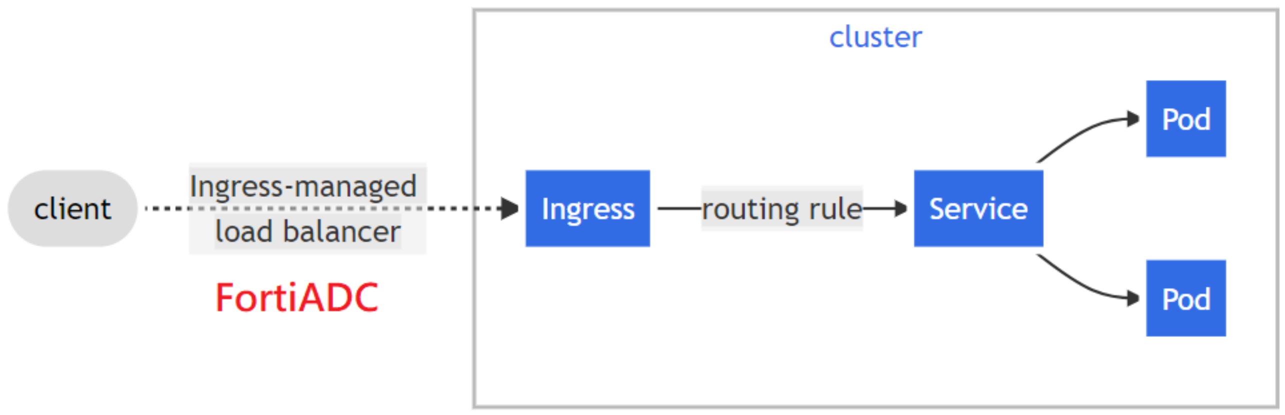
Kubernetes Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

An Ingress can be configured to give Kubernetes services externally reachable URLs, load balance traffic, terminate SSL/TLS, and offer name-based virtual hosting.

It is important to note that to satisfy an Ingress, you must have an Ingress controller, only creating an Ingress resource would have no effect.

FortiADC Kubernetes Controller is responsible for fulfilling the Ingress specified with the IngressClass "fadc-ingress-controller" with FortiADC.

Here is a simple example where an Ingress forwards traffic based on the routing rule to a Service:



In the above example, the Kubernetes service is an abstract method to expose an application running on a set of Pods as a network service. There are multiple types of Kubernetes services, among them, service with NodePort type exposes the service on each Kubernetes cluster Node's IP at a static port (the NodePort). You will be able to contact the Service from outside the cluster by requesting <NodeIP>:<NodePort>.

Services with the ClusterIP type only allows the Service to be reachable from within the cluster. In this case, you can use the Ingress to expose the service to the public internet.

Ingress class

Ingresses can be implemented by different controllers, often with different configurations. Each Ingress should specify an IngressClass name, which is a reference to an IngressClass resource that contains additional configuration including the name of the controller that should implement the class.

For an Ingress that would be implemented by FortiADC Kubernetes Controller, please specify the `ingressClassName` to `fadc-ingress-controller` in the ingress specification. Upon installation, FortiADC Kubernetes Controller is set as the default Ingress controller in the Helm chart value.yaml.

```
controller:
  ingressClassResource:
    name: "fadc-ingress-controller"
    enabled: true
    default: true
    controllerValue: "fortinet.com/fadc-ingress-controller"
```

Ingress types

FortiADC supports all 5 types of Ingress:

1. Default backend
2. Minimal-Ingress
3. Name-based virtual hosting
4. Hostname wildcards
5. TLS

For details on each Ingress type, see https://kubernetes.io/docs/concepts/services-networking/ingress/.

For an example Ingress file, see https://github.com/fortinet/fortiadc-kubernetes-controller/tree/main/ingress_examples.

Ensure the service used in the Ingress is already deployed with **NodePort** type. Kubernetes does not verify whether the service defined in the Ingress exists or not when deploying an Ingress. So, if you remove the service used in the deployed Ingress, you will not be warned or blocked from this action.

Default backend

An Ingress with no rules sends all traffic to a single default backend. If none of the hosts or paths match the HTTP request in the Ingress objects, the traffic is routed to your default backend. Therefore, if Rules are not specified, `defaultBackend` must be specified in the Ingress.

Note: FortiADC Kubernetes Controller only supports `Service` backend. A `Resource` backend is not supported.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the Default backend Ingress YAML example, follow this link: https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/ingress_examples/default_backend.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: default-backend
  annotations: {
    "fortiadc-ip" : "10.0.100.133",
    "fortiadc-login" : "fad-login",
    "fortiadc-ctrl-log" : "enable",
    "fortiadc-vdom" : "root",
    "virtual-server-ip" : "172.23.133.125",
    "virtual-server-interface" : "port1",
    "virtual-server-port" : "443",
    "virtual-server-addr-type" : "ipv4",
    "virtual-server-waf-profile" : "High-Level-Security",
    "virtual-server-av-profile" : "Antivirus-Profile",
    "virtual-server-dos-profile" : "",
    "virtual-server-captcha-profile" : "LB_CAPTCHA_PROFILE_DEFAULT",
    "virtual-server-nat-src-pool" : "",
    "virtual-server-traffic-group" : "default",
    "virtual-server-fortiview" : "enable",
    "virtual-server-traffic-log" : "enable",
    "virtual-server-wccp" : "enable",
    "load-balance-method" : "LB_METHOD_LEAST_CONNECTION",
    "load-balance-profile" : "LB_PROF_HTTPS"
  }
spec:
  ingressClassName: fadc-ingress-controller
  defaultBackend:
    service:
      name: default-http-backend
      port:
        number: 80
```

You can add defaultBackend in any particular Ingress. You can check the example here:

https://raw.githubusercontent.com/fortinet/fortiadc-kubernetes-controller/main/ingress_examples/ingress-with-default-backend.yaml

Minimal-Ingress

A minimal-Ingress has at least one rule defined in the Ingress with no specified default backend. The following is an example of a minimal-Ingress.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the Minimal Ingress YAML example, follow this link:

https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/ingress_examples/minimal-ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations: {
    "fortiadc-ip" : "10.0.100.133",
    "fortiadc-login" : "fad-login",
    "fortiadc-ctrl-log" : "enable",
    "fortiadc-vdom" : "root",
    "virtual-server-ip" : "172.23.133.8",
    "virtual-server-interface" : "port1",
    "virtual-server-port" : "443",
    "virtual-server-addr-type" : "ipv4",
    "virtual-server-waf-profile" : "High-Level-Security",
    "virtual-server-av-profile" : "Antivirus-Profile",
    "virtual-server-dos-profile" : "",
    "virtual-server-captcha-profile" : "LB_CAPTCHA_PROFILE_DEFAULT",
    "virtual-server-nat-src-pool" : "",
    "virtual-server-traffic-group" : "default",
    "virtual-server-fortiview" : "enable",
    "virtual-server-traffic-log" : "enable",
    "virtual-server-wccp" : "enable",
    "load-balance-method" : "LB_METHOD_LEAST_CONNECTION",
    "load-balance-profile" : "LB_PROF_HTTPS",
    "virtual-server-fortigslb-publicip-type" : "ipv4",
    "virtual-server-fortigslb-publicip" : "192.0.2.1",
    "virtual-server-fortigslb-1clickgslb" : "enable",
    "virtual-server-fortigslb-hostname" : "www",
    "virtual-server-fortigslb-domainname" : "example.com."
  }
spec:
  ingressClassName: fadc-ingress-controller
  rules:
  - http:
      paths:
      - path: /info
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 1241
```

Name-based virtual hosting

Name-based virtual hosts support the routing of HTTP/HTTPS traffic to multiple host names at the same IP address.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the Name-based virtual hosting Ingress YAML example, follow this link:

https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/ingress_examples/name-virtual-host-ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: name-virtual-host-ingress
  annotations: {
    "fortiadc-ip" : "10.0.100.133",
    "fortiadc-login" : "fad-login",
    "fortiadc-vdom" : "root",
    "virtual-server-ip" : "2001:db8::68",
    "virtual-server-interface" : "port1",
    "virtual-server-addr-type" : "ipv6"

  }
spec:
  ingressClassName: fadc-ingress-controller
  rules:
  - host: foo.bar.com
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: service3
            port:
              number: 1245
  - host: bar.foo.com
    http:
      paths:
      - pathType: Prefix
        path: "/"
        backend:
          service:
            name: service2
            port:
              number: 1242
```

Hostname wildcards

The hostname can be a precise match or a wildcard. Precise matches require the HTTP host header to match the host field (e.g., `foo.bar.com`). Wildcard matches require the HTTP host header to be equal to the suffix of the wildcard rule (e.g., `*.foo.com`).

Refer to the examples below to determine whether an HTTP host header is a wildcard match or not.

Host	Host header	Does it match?
`*.foo.com`	`bar.foo.com`	Yes, it matches based on shared suffix.
`*.foo.com`	`baz.bar.foo.com`	No, it does not match. Wildcard only covers a single DNS label.
`*.foo.com`	`foo.com`	No, it does not match. Wildcard only covers a single DNS label.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the Hostname wildcards Ingress YAML example, follow this link: https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/ingress_examples/ingress-wildcard-host.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: ingress-wildcard-host
  annotations: {
    "virtual-server-ip" : "172.23.133.10",
    "virtual-server-interface" : "port1",
    "fortiadc-ip" : "10.0.100.133",
    "fortiadc-login" : "fad-login",
    "fortiadc-vdom" : "root"
  }

spec:
  ingressClassName: fadc-ingress-controller
  rules:
  - host: "foo.bar.com"
    http:
      paths:
      - pathType: Prefix
        path: "/info"
        backend:
          service:
            name: service1
            port:
              number: 1241
  - host: "*.foo.com"
    http:
      paths:
      - pathType: Prefix
        path: "/hello"
        backend:
          service:
            name: service2
            port:
              number: 80
```

TLS

You can secure an Ingress by specifying a Secret that contains a TLS private key and certificate. The Ingress resource only supports a single TLS port, 443, and assumes TLS termination at the Ingress point (traffic to the Service and its Pods is in plain text). If the TLS configuration section in an Ingress specifies different hosts, they are multiplexed on the same port according to the hostname specified through the SNI TLS extension (provided the Ingress controller supports

SNI). The TLS secret must contain keys named `tls.crt` and `tls.key` that contain the certificate and private key to use for TLS.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the TLS key YAML example, follow this link: https://kubernetes.io/docs/concepts/services-networking/ingress/#tls

For example:

```
apiVersion: v1
kind: Secret
metadata:
  name: testsecret-tls
  namespace: default
data:
  tls.crt: base64 encoded cert
  tls.key: base64 encoded key
type: kubernetes.io/tls
```

You can create the TLS secret by using the kubectl command, for example:

```
kubectl create secret tls testsecret-tls --cert=/etc/ssl/tls.crt --key=/etc/ssl/tls.key
```

Referencing this secret in an Ingress tells the Ingress controller to secure the channel from the client to the load balancer using TLS. You need to ensure the TLS secret you created came from a certificate that contains a Common Name (CN), also known as a Fully Qualified Domain Name (FQDN) for `https-example.foo.com`.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the TLS Ingress YAML example, follow this link: https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/ingress_examples/tls-example-ingress.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: tls-example-ingress
  annotations: {
    "virtual-server-ip" : "172.23.133.11",
    "virtual-server-interface" : "port1",
    "fortiadc-ip" : "10.0.100.133",
    "fortiadc-login" : "fad-login",
    "fortiadc-vdom" : "root"
  }
```

```
spec:
  ingressClassName: fadc-ingress-controller
  tls:
  - hosts:
      - https-example.foo.com
    secretName: testsecret-tls
  rules:
  - host: https-example.foo.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: service2
            port:
              number: 80
```

Kubernetes Custom Resource

Kubernetes Custom Resources extend the Kubernetes API by enabling users to define new resource types.

- VirtualServer Custom Resource Definition (CRD) was developed to fully leverage FortiADC virtual server capabilities in Kubernetes and improve usability. The VirtualServer resource functions similarly to Ingress; however, it allows most virtual server parameters to be specified directly in the resource `spec`, with only a few FortiADC login-related settings maintained in `annotations`. For details, see VirtualServer Custom Resource Definition on page 25.
- RemoteServer and Host Custom Resource Definition (CRD) were developed to better integrate FortiADC GLB features in Kubernetes. The RemoteServer resource functions are used to define and monitor virtual servers. The Host resource function defines an FQDN to distribute traffic to a suitable virtual server. For details, see GLB Custom Resource Definition on page 40.

These custom resources allows most parameters to be specified directly in the resource `spec`, with only a few FortiADC login-related settings maintained in annotations.

As a result, all Ingress features discussed previously, such as hostname wildcards and TLS, can also be configured within the VirtualServer resource.

VirtualServer CRD with version v1alpha2 extends FortiADC features, offers not only HTTP/HTTPS load balancing, but also TCP/UDP proxying. A Layer 4 TCP/UDP VirtualServer resource can be defined to be a proxy or load balancing server for Layer 4 servers in Kubernetes.

Due to networking limitations imposed by the Kubernetes CNI, when defining a Layer 4 (TCP/UDP) VirtualServer, the packet forwarding method is fixed to **Full NAT**.

FortiADC Kubernetes Controller version 3.1 still serves the v1alpha1 VirtualServer CRD; however, v1alpha1 is scheduled for future deprecation.

Please use v1alpha2 for new deployments. For any existing v1alpha1 resources stored in the Kubernetes API server, the FortiADC Kubernetes Controller will automatically convert them to v1alpha2.

VirtualServer Custom Resource Definition

The `spec` section in the VirtualServer custom resource defines the desired state of the virtual server, including listener settings, backend services, SSL configuration, and routing rules. It provides a more granular and FortiADC-aligned configuration than the traditional Ingress resource.

virtualServer Field Details

Field	Type	Description	Default
`addressType`	string	Specifies the IP version for the virtual server (`ipv4` or `ipv6`).	`ipv4`

Field	Type	Description	Default
address	string	The VirtualServer IP on FortiADC where client traffic is received. This parameter is required.	
port	int	The listening port of the virtual server, typically 443 for HTTPS.	443
interface	string	The FortiADC network interface for the client to access the virtual server (e.g., port1).	port1
loadBalanceProfile	string	Allowed values for the load balancing profile: • LB_PROF_HTTPS • LB_PROF_HTTP • LB_PROF_TCP • LB_PROF_UDP	LB_PROF_HTTPS
loadBalanceMethod	string	Load balancing method, such as LB_METHOD_ROUND_ROBIN.	LB_METHOD_ROUND_ROBIN
persistence	string	The persistence rule, such as LB_PERSIS_HASH_SRC_ADDR.	
wafProfile	string	The name of the Web Application Firewall (WAF) profile to apply.	
captchaProfile	string	CAPTCHA profile to enable human verification security features.	
avProfile	string	Antivirus profile to scan traffic for threats.	
ipsProfile	string	Intrusion Prevention System (IPS) profile name.	
trafficGroup	string	The traffic group that this virtual server belongs to for traffic forwarding.	default
fortiview	string	Enables/disables FortiView, FortiADC's traffic visualization and analytics (enable or disable).	disable
trafficLog	string	Enables/disables logging of traffic handled by this virtual server (enable or disable).	disable
wccp	string	Enables/disables WCCP (Web Cache Communication Protocol) (enable or disable).	disable
fortigslbPublicIpType	string	Type of public IP used for FortiGSLB (usually ipv4).	ipv4

Field	Type	Description	Default
`fortigslbPublicAddress`	string	The public IP address used for global server load balancing (GSLB).	
`fortigslbOneClick`	string	Enables/disables FortiADC's one-click GSLB configuration feature (`enable` or `disable`).	`disable`
`fortigslbHostName`	string	Hostname registered for GSLB. **Note**: In YAML, always enclose `@` and `*` in quotes (for example, `fortigslbHostName: "@"`). If not quoted, these characters may be interpreted as special YAML tokens and lead to parsing errors.	
`fortigslbDomainName`	string	Domain name associated with the GSLB hostname.	
`realServerPool`	object	Direct backend service definition. Use instead of `contentRoutings`.	
`contentRoutings`	list	Defines Layer 4 and Layer 7 routing rules. Each rule can route traffic based on host and path or source address, and includes: • `name`: routing rule name • type: Type of content routing. (`l4-content-routing` or `l7-content-routing`) • `host`: FQDN for this route • `path`: URL path • `pathType`: match type (`Prefix` or `Exact`) • `realServerPool`: backend service, port, and namespace • `sourceAddressType`: Source address type to match. • `sourceAddress`: Source address to match in CIDR form.	
`natSourcePoolList`	list	List of SNAT (source NAT) pools. **Each SNAT pool must already exist on FortiADC before creating the VirtualServer.** If the specified pool does not exist, the VirtualServer creation will fail.	
`tls`	list	List of TLS certificate configurations. Each entry includes: • `hosts`: list of hostnames this certificate applies to • `secretName`: name of the Kubernetes TLS	

Field	Type	Description	Default
		secret	
`vdom`	string	Specifies the VDOM (Virtual Domain) on FortiADC where the virtual server is deployed.	

realServerPool Field Details

This field defines the Kubernetes backend service and is available at top-level or inside `contentRoutings` fields. You should either define a `realServerPool` or `contentRoutings` list, but not both.

Field	Type	Description
`service`	string	Name of the Kubernetes service that this virtual server routes traffic to. Must match the format of a valid DNS label (e.g., `my-service`). This is the target backend.
`servicePort`	integer or string	Port of the target service. Can be a numeric port (e.g., 80) or a named port (e.g., "http"), depending on how the Kubernetes service is defined.
`serviceNamespace`	string	Namespace of the target service. Useful for cross-namespace routing.

contentRoutings Field Details

This field defines Layer 7 (HTTP/S) and Layer 4(TCP/UDP) routing rules, allowing traffic to be forwarded based on hostname and URL path to specific http/https Kubernetes services or to be filtered by the source address to the TCP/UDP Kubernetes services.

Field	Type	Description	Default
`name`	string	A unique name for the routing rule.	
`type`	string	Type of content routing. `l4-content-routing` or `l7-content-routing`	`l7-content-routing`
`host`	string	The hostname (FQDN) to match for this rule (e.g., `example.com`, `*.foo.com`). Only available when type is `l7-content-routing`	
`path`	string	The URL path to match (e.g., `/`, `/info`, `/hello`). Only available when type is `l7-content-routing`.	
`pathType`	string	The type of path matching: • `Prefix` - matches based on URL prefix • `Exact` - matches the exact path	

Field	Type	Description	Default
		Only available when type is `l7-content-routing`.	
`sourceAddressType`	string	Specifies the IP version for the Source address to match (ipv4 or ipv6). Only available when type is `l4-content-routing`.	`ipv4`
`sourceAddress`	string	Source address to match in CIDR form. (e.g., 0.0.0.0/0, ::/0) Only available when type is `l4-content-routing`.	`0.0.0.0/0`
`realServerPool.service`	string	The name of the Kubernetes Service to route traffic to.	
`realServerPool.servicePort`	int	The target port of the Kubernetes Service.	
`realServerPool.serviceNamespace`	string	The Kubernetes namespace where the service resides.	

tls Field Details

The `tls` field specifies the TLS termination configuration for the VirtualServer. It enables the association of multiple TLS certificates stored as Kubernetes secrets with specific hostnames through **Server Name Indication (SNI)**. This allows secure management of multiple domains.

Field	Type	Description
`hosts`	list of strings	A list of hostnames that this TLS certificate applies to (e.g., `a.foo.com`, `*.example.com`). The host must be defined in **contentRoutings**.
`secretName`	string	The name of the Kubernetes TLS secret that contains the certificate and private key. This secret must be present in the same namespace as the VirtualServer Custom Resource.

For more details on configuration parameters with virtual-server and load-balance prefixes, see the FortiADC Administration Guide on Configuring Virtual Servers.

VirtualServer Custom Resource Examples

The following examples demonstrate typical VirtualServer configurations:

Layer 7 HTTP/HTTPS VirtualServer:

- defaultbackend-virtualserver
- simple-fanout-virtualserver
- wildcard-host-virtualserver

- tls-san-virtualserver

Layer 4 TCP/UDP VirtualServer:

- tcp-echo-virtualserver on page 36
- tcp-postgres-ssl-virtualserver on page 37
- udp-echo-virtualserver on page 39

Layer 7 HTTP/HTTPS VirtualServer

defaultbackend-virtualserver

The defaultbackend virtual server defines an HTTPS entry point at `192.168.1.103:443`, which uses FortiADC to perform Layer 7 load balancing and route traffic to the `default-http-backend` service inside the Kubernetes cluster.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the **defaultbackend-virtualserver** YAML example, follow this link: https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/customResource/virtualserver/virtualserver_defaultbackend.yaml

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha2
kind: VirtualServer
metadata:
  name: defaultbackend-virtualserver
  annotations: {
    "fortiadc-ip" : "172.23.133.110",
    "fortiadc-login" : "fad-login",
    "fortiadc-ctrl-log" : "enable",
    "fortiadc-admin-port": "443"
   }
spec:
  addressType: ipv4
  address: 192.168.1.103
  port: 443
  interface: port1
  loadBalanceProfile: LB_PROF_HTTPS
  loadBalanceMethod: LB_METHOD_ROUND_ROBIN
  contentRoutings:
    - name: default_route
      realServerPool:
        service: default-http-backend
        servicePort: 80
        serviceNamespace: default
  vdom: root
```

simple-fanout-virtualserver

Simple-fanout virtualserver defines an advanced VirtualServer resource for FortiADC on Kubernetes. It listens on `192.168.1.101:443` via interface port3 and includes multiple routing rules based on URL paths under the same hostname (`app.example.com`), routing to two different backend services (`service1` and `service2`).

In addition to basic load balancing, it enables several FortiADC security and traffic management features:

- **WAF**, **CAPTCHA**, and **Antivirus** profiles for enhanced security
- **FortiGSLB** integration with one-click configuration, public IP mapping, and domain binding
- **Traffic logging** and **FortiView** visibility enabled
- A **NAT source pool** for outbound traffic handling
- All configurations are applied under the `root` VDOM

This setup demonstrates how the VirtualServer CRD can unify advanced FortiADC features with Kubernetes-native routing logic.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the **simple-fanout-virtualserver** YAML example, follow this link: https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/customResource/virtualserver/virtualserver_simple_fanout.yaml

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha2
kind: VirtualServer
metadata:
  name: simple-fanout-virtualserver
  annotations: {
    "fortiadc-ip" : "172.23.133.110",
    "fortiadc-login" : "fad-login",
    "fortiadc-ctrl-log" : "enable",
    "fortiadc-admin-port": "443"
   }
  labels:
    fadcr: "true"
spec:
  addressType: ipv4
  address: 192.168.1.101
  port: 443
  interface: port3
  loadBalanceProfile: LB_PROF_HTTPS
  loadBalanceMethod: LB_METHOD_ROUND_ROBIN
  wafProfile: High-Level-Security
  captchaProfile: LB_CAPTCHA_PROFILE_DEFAULT
  avProfile: Antivirus-Profile
  trafficGroup: default
  fortiview: enable
  trafficLog: enable
  wccp: disable
  fortigslbPublicIpType: ipv4
  fortigslbPublicAddress: 203.0.113.1
  fortigslbOneClick: enable
  fortigslbHostName: samplehost
  fortigslbDomainName: example.com.
  contentRoutings:
    - name: route1
      host: app.example.com
      path: /info
      pathType: Prefix
      realServerPool:
        service: service1
        servicePort: 1241
        serviceNamespace: default
    - name: route2
      host: app.example.com
      path: /hello
      pathType: Prefix
      realServerPool:
        service: service2
        servicePort: 1242
        serviceNamespace: default
  natSourcePoolList:
    - name: nat-pool-1
  vdom: root
```

wildcard-host-virtualserver

Wildcard-host-virtualserver defines a VirtualServer on FortiADC using a wildcard hostname setup. The server listens on `192.168.1.102:443` via `port1`, and supports routing based on specific hostnames, including both a **fully qualified domain** (`foo.bar.com`) and a **wildcard domain** (`*.foo.com`).

It demonstrates the content routing with general host and wildcard-host in FortiADC features:

Content Routing:

- `/info` for `foo.bar.com` → routes to `service1`
- `/hello` for any subdomain of `foo.com` → routes to `service2`

This example highlights how wildcard hostnames can be seamlessly applied in FortiADC virtual servers managed through Kubernetes CRDs.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the **wildcard-host-virtualserver** YAML example, follow this link: https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/customResource/virtualserver/virtualserver_wildcard_host.yaml

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha2
kind: VirtualServer
```

```
metadata:
  name: wildcard-host-virtualserver
  annotations: {
    "fortiadc-ip" : "172.23.133.110",
    "fortiadc-login" : "fad-login",
    "fortiadc-ctrl-log" : "enable",
    "fortiadc-admin-port": "443"
   }
  labels:
    fadcr: "true"
spec:
  addressType: ipv4
  address: 192.168.1.102
  port: 443
  interface: port1
  loadBalanceProfile: LB_PROF_HTTPS
  loadBalanceMethod: LB_METHOD_ROUND_ROBIN
  wafProfile: High-Level-Security
  captchaProfile: LB_CAPTCHA_PROFILE_DEFAULT
  avProfile: Antivirus-Profile
  trafficGroup: default
  fortiview: enable
  trafficLog: enable
  wccp: disable
  fortigslbPublicIpType: ipv4
  fortigslbPublicAddress: 203.0.113.1
  fortigslbOneClick: enable
  fortigslbHostName: samplehost
  fortigslbDomainName: example.com.
  contentRoutings:
    - name: route1
      host: "foo.bar.com"
      path: /info
      pathType: Prefix
      realServerPool:
        service: service1
        servicePort: 1241
        serviceNamespace: default
    - name: route2
      host: "*.foo.com"
      path: /hello
      pathType: Prefix
      realServerPool:
        service: service2
        servicePort: 1242
        serviceNamespace: default
  vdom: root
```

tls-san-virtualserver

Tls-san-virtualserver defines a VirtualServer that implements advanced TLS termination with **multiple hostnames and TLS secrets**, leveraging SNI/SAN functionality. The virtual server listens on `192.168.1.100:443` via `port1`, and applies different TLS certificates based on the requested hostname.

TLS Configuration:

- TLS is enabled with **SNI-based certificate selection**.
- Two TLS secrets:
  - `tls-foo` for `a.foo.com`, `b.foo.com`, and `c.foo.com`
  - `testsecret-tls` for `https-example.foo.com`

Routing Rules:

- Traffic is routed based on host + path combinations:
  - `/` on `a.foo.com` → `default-http-backend`
  - `/env` on `b.foo.com` → `service1`
  - `/info` on `c.foo.com` → `service1`
  - `/hello` on `https-example.foo.com` → `service2`

This setup is ideal for scenarios where a single VirtualServer handles multiple domains securely, each with its own certificate and routing logic. It demonstrates how the CRD supports enterprise-grade TLS configurations in a Kubernetes-native way.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the **tls-san-virtualserver** YAML example, follow this link:
https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/customResource/virtualserver/virtualserver_tls_san.yaml

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha2
kind: VirtualServer
metadata:
  name: tls-san-virtualserver
  annotations: {
    "fortiadc-ip" : "172.23.133.110",
    "fortiadc-login" : "fad-login",
    "fortiadc-ctrl-log" : "enable",
    "fortiadc-admin-port": "443"
   }
spec:
  addressType: ipv4
  address: 192.168.1.100
  port: 443
  interface: port1
  loadBalanceProfile: LB_PROF_HTTPS
  loadBalanceMethod: LB_METHOD_ROUND_ROBIN
  wafProfile: High-Level-Security
  captchaProfile: LB_CAPTCHA_PROFILE_DEFAULT
  avProfile: Antivirus-Profile
  trafficGroup: default
  fortiview: enable
  trafficLog: enable
```

```
wccp: disable
fortigslbPublicIpType: ipv4
fortigslbPublicAddress: 203.0.113.1
fortigslbOneClick: enable
fortigslbHostName: samplehost
fortigslbDomainName: example.com.
contentRoutings:
  - name: default-route
    host: a.foo.com
    path: /
    pathType: Prefix
    realServerPool:
      service: default-http-backend
      servicePort: 80
      serviceNamespace: default
  - name: route1
    host: b.foo.com
    path: /env
    pathType: Exact
    realServerPool:
      service: service1
      servicePort: 1241
      serviceNamespace: default
  - name: route2
    host: c.foo.com
    path: /info
    pathType: Prefix
    realServerPool:
      service: service1
      servicePort: 1241
      serviceNamespace: default
  - name: route3
    host: https-example.foo.com
    path: /hello
    pathType: Prefix
    realServerPool:
      service: service2
      servicePort: 1242
      serviceNamespace: default
natSourcePoolList:
  - name: nat-pool-1
tls:
  - hosts:
      - a.foo.com
      - b.foo.com
      - c.foo.com
    secretName: tls-foo
  - hosts:
      - https-example.foo.com
    secretName: testsecret-tls
vdom: root
```

Layer 4 TCP/UDP VirtualServer

tcp-echo-virtualserver

This VirtualServer defines a Layer 4 TCP load balancer. It listens on IP 192.168.1.106, port 12345, via interface port3, and uses LB_PROF_TCP with round-robin load balancing. A SNAT pool named socat is configured to fulfill Kubernetes CNI requirements for Full NAT in L4 traffic. The routing rule route1 forwards traffic from source subnet 192.168.1.0/24 to a backend service tcp-service on port 12345. FortiADC connection details are provided via annotations, and the configuration applies to the root VDOM.

The tcp echo server is created based on alpine/socat docker image. See the following page for more deployment details: https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/service_examples/tcp_echo_service.yaml

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the **tcp-echo-virtualserver** YAML example, follow this link: https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/customResource/virtualserver/virtualserver_tcp_echo_server.yaml

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha2

kind: VirtualServer

metadata:

  name: tcp-echo-virtualserver

  annotations: {

    "fortiadc-ip" : "172.23.133.110",

    "fortiadc-login" : "fad-login",

    "fortiadc-ctrl-log" : "enable",

    "fortiadc-admin-port": "443"

   }

spec:

  addressType: ipv4

  address: 192.168.1.106

  port: 12345

  interface: port3
  loadBalanceProfile: LB_PROF_TCP

  loadBalanceMethod: LB_METHOD_ROUND_ROBIN

  natSourcePoolList:

    - name: socat

  contentRoutings:

    - name: route1

      type: l4-content-routing

      sourceAddressType: ipv4

      sourceAddress: 192.168.1.0/24

      realServerPool:

        service: tcp-echo-service

        servicePort: 12345

        serviceNamespace: default

  vdom: root
```

After deployed the tcp echo server, you can use nc tool to verify the deployment and test connection. For example:

```
~> echo "hello tcp" | nc -v 192.168.1.106 12345

Connection to 192.168.1.106 12345 port [tcp/*] succeeded!

hello tcp
```

tcp-postgres-ssl-virtualserver

This VirtualServer defines a Layer 4 TCP load balancer for PostgreSQL traffic. It listens on IP `192.168.1.108` and port `5432` via interface `port3`, with the `LB_PROF_TCP` profile and round-robin load balancing. A SNAT pool named socat is configured to enable full NAT, as required for TCP services in Kubernetes. The VirtualServer includes a content routing rule named route1, which matches traffic from the `192.168.1.0/24` subnet and forwards it to a backend service named `postgresql-service` on port `5432` in the `default` namespace. The configuration is applied under the `root` VDOM, and FortiADC connection details are provided via annotations for integration and control.

You can move to the deployment section for more details on how to use virtualserver with postgresql with ssl enabled in kubernetes.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the **tcp-postgres-ssl-virtualserver** YAML example, follow this link: https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/customResource/virtualserver/virtualserver_postgres_ssl.yaml

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha2

kind: VirtualServer

metadata:

  name: tcp-postgres-ssl-virtualserver

  annotations: {

    "fortiadc-ip" : "172.23.133.110",

    "fortiadc-login" : "fad-login",

    "fortiadc-ctrl-log" : "enable",

    "fortiadc-admin-port": "443"

   }

spec:

  addressType: ipv4

  address: 192.168.1.108

  port: 5432

  interface: port3

  loadBalanceProfile: LB_PROF_TCP

  loadBalanceMethod: LB_METHOD_ROUND_ROBIN

  natSourcePoolList:

    - name: socat

  contentRoutings:

    - name: route1

      type: l4-content-routing
      sourceAddressType: ipv4

      sourceAddress: 192.168.1.0/24

      realServerPool:

        service: postgres-ssl-service

        servicePort: 5432

        serviceNamespace: default

  vdom: root
```

udp-echo-virtualserver

This VirtualServer defines a Layer 4 UDP load balancer. It listens on IP `192.168.1.107`, port `9999`, via interface `port3`, and uses the `LB_PROF_UDP` profile with round-robin load balancing. Traffic is forwarded to a backend Kubernetes service named `udp-echo-service` on port `9999` in the `default` namespace. A SNAT pool named `socat` is specified to ensure Full NAT is applied, as required for UDP traffic in Kubernetes environments. The configuration is deployed under the `root` VDOM, and FortiADC connection details are provided via annotations for management access.

The udp echo server is created based on alpine/socat docker image. See the following page for more deployment details: https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/service_examples/udp_echo_service.yaml

---

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the **udp-echo-virtualserver** YAML example, follow this link: https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/customResource/virtualserver/virtualserver_udp_echo_server.yaml

---

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha2
kind: VirtualServer
metadata:
  name: udp-echo-virtualserver
  annotations: {
    "fortiadc-ip" : "172.23.133.110",
    "fortiadc-login" : "fad-login",
    "fortiadc-ctrl-log" : "enable",
    "fortiadc-admin-port": "443"
   }
spec:
  addressType: ipv4
  address: 192.168.1.107
  port: 9999
  interface: port3
  loadBalanceProfile: LB_PROF_UDP
  loadBalanceMethod: LB_METHOD_ROUND_ROBIN
  realServerPool:
    service: udp-echo-service
    servicePort: 9999
    serviceNamespace: default
  natSourcePoolList:
    - name: socat
  vdom: root
```

After deployed the udp echo server, you can use nc tool to verify the deployment and test connection. For example:

```
echo -n "hello-udp" | nc -u -w1 192.168.1.107 9999

hello-udp
```

GLB Custom Resource Definition

The `spec` section in the RemoteServer custom resource defines the connection information of the server, locations, health monitor, and state of virtual servers.

The `spec` section in the Host custom resource defines FQDN, multiple groups of virtual servers, and global load balancing method.

RemoteServer Field Details

Field	Type	Description	Default
`serverType`	string	Specifies the type of server. (`FortiADC-SLB` or `Generic-Host`)	FortiADC-SLB
`dataCenter`	object	Specifies the data center where the server is located. This parameter is required. • `name` • `location` • `description` For details, see dataCenter Field Details on page 41.	
`vdom`	string	Specifies the VDOM (Virtual Domain) on FortiADC where the server is deployed.	
FortiADC-SLB fields only			
`addressType`	string	Specifies the IP version of the server (`ipv4` or `ipv6`).	`ipv4`
`ip`	string	The IPv4 address of the server.	`0.0.0.0`
`ip6`	string	The IPv6 address of the server.	`::`

Field	Type	Description	Default
`port`	int	Specifies the port which the server listens on.	`5858`
`authtType`	string	Specifies the authentication method. • `none` • `TCP_MD5SIG` • `auth_verify`	`none`
`authKey`	string	Specifies an authentication key to be authenticated by peers.	
`cert`	string	Specifies a certificate to connect to the server.	
`caGroup`	string	Specifies an CA group to verify the peer certificate.	
`intermediateCAGroup`	string	Specifies an Intermediate CA group to verify the peer certificate.	
`autosSync`	string	Enables server to automatically synchronize virtual servers (`enable` or `disable`).	`disable`
Generic-Host fields only			
`healthCheckList`	string	Reference to the Health Check, which is used to monitor virtual servers.	
`virtualServers`	list	Specifies virtual servers managed by this server. This parameter is required. • `name` • `addressType` • `ip` • `ip6` For details, see virtualServers Field Details on page 42.	

dataCenter Field Details

This field defines Data Center information.

Field	Type	Description	Default
`name`	string	A unique name for the data center.	
`location`	string	Locations referenced to Geo IP.	`zz`
`description`	string	Description.	

virtualServers Field Details

This field defines Virtual Server information. The health check of each virtual server is inherited by `RemoteServer`.

Field	Type	Description	Default
`name`	string	A unique name for the virtual server.	
`addressType`	string	IP version	`ipv4`
`ip`	string	IPv4 address	`0.0.0.0`
`ip6`	string	IPv6 address	::

host Field Details

Field	Type	Description	Default
`hostname`	string	Specifies the hostname for FQDN. Symbol @ and wildcard name (starts with *) are allowed. This parameter is required. **Note**: In YAML, always quote "@" and "*" (e.g., fortigslbHostName: "@"), otherwise they may be parsed as special tokens and cause errors.	
`domain`	string	Specifies the domain name for FQDN. This parameter is required.	
`dnsPolicy`	list	Reference to Global DNS policys for FQDN. • `name`: Global DNS Policy name	
`singleRecord`	string	Enable to reply to only one single record (`enable` or `disable`).	`disable`
`persistence`	string	Enable the persistence table to reply to the same records when the same client resource queries to the same host (`enable` or `disable`).	`disable`
`lbMethod`	string	Specifies the selection method for virtual server pools. • `weight` • `topology` • `global-availability`	`weight`
`feedbackIPv4`	string	Specify an IPv4 address to reply due to no available virtual server.	`0.0.0.0`
`feedbackIPv6`	string	Specify an IPv6 address to reply due to no available virtual server.	`::`

Field	Type	Description	Default
`fortiview`	string	Enables FortiView, FortiADC's traffic visualization and analytics (`enable` or `disable`).	`disable`
`virtualServerPools`	list	Define virtual server pool includes: This parameter is required. • `name` • `topology` • `isp` • `weight` • `virtualServerPool` For details, see virtualServerPools Field Details on page 43.	
`vdom`	string	Specifies the VDOM (Virtual Domain) on FortiADC where the host is deployed.	

virtualServerPools Field Details

This field defines virtual server pools. The Global Load Balancing method used to select a virtual server pool is defined in `host`.

Field	Type	Description	Default
`name`	string	A unique name for the Virtual Server Pool.	
`topology`	string	Reference to a Location List for topology method.	`zz`
`isp`	string	Reference to an ISP Address for topology method.	
`weight`	int	Weight value for weight method.	`1`
`virtualServerPool`	object	Reference to a Virtual Server Pool. For details, see virtualServerPool Field Details on page 43.	

virtualServerPool Field Details

This field defines the details of each virtual server pool, including multiple virtual servers and priority rules. The FQDN response contains the available virtual servers listed in order of priority.

Field	Type	Description	Default
`preferred`	string	Specifies the first priority of selection method for virtual servers. • `None`	`None`

Field	Type	Description	Default
		• GEO • GEO-ISP • RTT • Least-Connections • Connection-Limit • Bytes-Per-Second • Server-Performance	
alternate	string	Specifies the second priority of selection method for virtual servers. • None • GEO • GEO-ISP • RTT • Least-Connections • Connection-Limit • Bytes-Per-Second • Server-Performance	None
preferredCPUMemoryRatio	int	Specifies a weight value for Server-Performance preferred method (0 to 10).	5
alternateCPUMemoryRatio	int	Specifies a weight value for Server-Performance alternate method (0 to 10).	5
checkServerStatus	string	Enable to not reply to the server if the server is unresponsive (enable or disable).	enable
checkVirtualServerExist	string	Enable to not reply to the virtual server if the virtual server is unresponsive (enable or disable).	enable
lbMethod	string	Specifies the selection method for virtual servers (wrr).	wrr
virtualServers	list	Define a list of virtual servers. This parameter is required. • server • serverMember • ttl • weight • backup For details, see virtualServers Field Details on page 45.	

virtualServers Field Details

This field specifies a virtual server which is already defined in `RemoteServer`.

Field	Type	Description	Default
`server`	string	Reference to a server.	
`serverMember`	string	Reference to a member on the server.	
`ttl`	int	Time to Live (TTL) of packets (`-1` to `2147483647`). `-1` means it will use the zone level TTL.	`-1`
`weight`	int	Weight value (`1` to `255`).	`1`
`backup`	string	Enable to designate the member as a backup (`enable` or `disable`).	`disable`

GLB Custom Resource Examples

The following examples demonstrate typical GLB configurations:

- **RemoteServer**:
  - generic-host-server on page 45
  - fortiadc-slb-server on page 46
- **Host**:
  - simple-host on page 47

RemoteServer

generic-host-server

The RemoteServer CRD defines two types of Server and Data Center objects in FortiADC.

The first server type, Generic-Host, represents a third-party ADC or virtual server along with its data center location. FortiADC provides a health check feature that polls service status based on the application protocol. Generic-Host servers can reference this health check to monitor virtual server availability.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the **generic-rs1** YAML example, follow this link:
https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/customResource/glb/remoteserver_host.yaml

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha1
kind: RemoteServer
metadata:
  name: generic-rs1
  annotations: {
    "fortiadc-ip" : "172.23.133.110",
    "fortiadc-login" : "fad-login",
    "fortiadc-ctrl-log" : "enable",
    "fortiadc-admin-port": "443"
   }
spec:
  serverType: Generic-Host
  dataCenter:
    name: US
    location: US
  healthCheckList: LB_HLTHCK_ICMP
  virtualServers:
    - name: v1
      addressType: ipv4
      ip: 20.20.20.1
    - name: v2
      addressType: ipv4
      ip: 20.20.20.2
    - name: v3
      addressType: ipv6
      ip6: 2001::2
  vdom: root
```

fortiadc-slb-server

The FortiADC-SLB server defines a FortiADC instance along with its data center location. This server type automatically discovers virtual servers on the remote FortiADC by connecting to `127.0.0.1:5858`. Instead of actively performing health checks, it synchronizes virtual server availability from the remote FortiADC.

When `autoSync` is enabled, discovered virtual servers become visible configuration objects on FortiADC and can be referenced by a virtual server pool.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the **fortiadc-rs1** YAML example, follow this link:

https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/customResource/glb/remoteserver_slb.yaml

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha1
kind: RemoteServer
metadata:
  name: fortiadc-rs1
  annotations: {
    "fortiadc-ip" : "172.23.133.110",
    "fortiadc-login" : "fad-login",
    "fortiadc-ctrl-log" : "enable",
    "fortiadc-admin-port": "443"
   }
spec:
  serverType: FortiADC-SLB
  dataCenter:
    name: Canada
    location: CA
  addressType: ipv4
  ip: 127.0.0.1
  port: 5858
  autoSync: enable
  vdom: root
```

Host

simple-host

The Host CRD defines Host and Virtual Server Pool objects in FortiADC.

A Host represents an FQDN, such as `www.host1.com`, and can contain multiple virtual server pools. FortiADC selects a pool based on the configured Global Load Balancing method.

The virtual server pool `generic-vp/fortiadc-vp` represents a set of virtual servers defined by the Generic-Host server `generic-rs1/fortiadc-rs1` in the previous example. After a pool is selected, FortiADC returns an ordered list of available virtual servers as resource records based on the preferred or alternate method.

This example demonstrates how a host and its virtual server pools can be managed through Kubernetes CRDs.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the **simple-host** YAML example, follow this link:https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/customResource/glb/host.yaml

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha1
kind: Host
metadata:
  name: simple-host
  annotations: {
    "fortiadc-ip" : "172.23.133.110",
    "fortiadc-login" : "fad-login",
    "fortiadc-ctrl-log" : "enable",
    "fortiadc-admin-port": "443"
   }
spec:
  hostname: www
  domain: host1.com.
  lbMethod: weight
  feedbackIPv4: 0.0.0.1
  fortiview: enable
  virtualServerPools:
    - name: generic-vp
      weight: 2
      virtualServerPool:
        preferred: GEO
        virtualServers:
          - server: default_generic-rs1
            serverMember: v1
          - server: default_generic-rs1
            serverMember: v2
            weight: 5
    - name: fortiadc-vp
      weight: 3
      virtualServerPool:
        preferred: RTT
        virtualServers:
          - server: default_fortiadc-rs1
            serverMember: v1
            weight: 3
          - server: default_fortiadc-rs1
            serverMember: v2
            weight: 5
  vdom: root
```

Controller Installation

The FortiADC Kubernetes Controller is deployed using Helm 3. Helm simplifies the installation process by packaging all required Kubernetes objects (such as permissions, services, and webhooks) into a single, manageable command.

Installation Roadmap

Follow these sections in sequence to complete your deployment:

1. Preparing for the Validating and Conversion Webhook Servers on page 51: Deploy the mandatory certificate management infrastructure required for webhook security.
2. Installing the Controller Using Helm Chart on page 52: Add the Fortinet repository and execute the installation or upgrade commands.
3. Verifying the Installation on page 56: Perform health checks on the pods, logs, and CRD registration to confirm success.

Installation Prerequisites

Before deploying the controller, verify that your environment meets these critical requirements:

- **Helm Version**: Only **Helm 3 (version 3.6.3 or later)** is supported.
- **TLS Management**: Version 3.1.0 introduces **Validating and Conversion Webhook Servers** that require **cert-manager** to handle mandatory TLS certificates.

You must install cert-manager before deploying or upgrading the controller. Failure to do so will cause the installation to fail.

Core Components Deployed

When you install the controller, the following resources are automatically created and configured in your cluster to manage traffic synchronization:

Kubernetes object	Description
Deployment	By configuring the replica and pod template in the Kubernetes deployment, the deployment ensures FortiADC Kubernetes Controller provides a non-terminated service.
Service Account	The service account is used in FortiADC Kubernetes Controller.
Cluster Role	A cluster role defines the permission on the Kubernetes cluster-scoped Ingress-related and FortiADC custom resources related objects.

Kubernetes object	Description
Cluster Role Binding	The cluster role is bound to the service account used for FortiADC Kubernetes Controller, allowing FortiADC Kubernetes Controller to access and operate the Kubernetes cluster-scoped Ingress-related and FortiADC custom resources related objects.
Ingress Class	The IngressClass "fadc-ingress-controller" is created for FortiADC Kubernetes Controller to identify the Ingress resource. If the Ingress is defined with the IngressClass "fadc-ingress-controller", FortiADC Kubernetes Controller will manage this Ingress resource.
CustomResourceDefinition	The CustomResourceDefinition “VirtualServer” defines advanced HTTP routing in Kubernetes and offer the TCP/UDP proxy for Kubernetes TCP/UDP server in Kubernetes. It enhances the standard `Ingress` by supporting: • Multiple upstreams • Precise path routing and delegation • Traffic splitting and advanced actions • Native integration with FortiADC virtualserver features (WAF, Fortiview, etc.) • Support Layer4 TCP/UDP proxy and content routing A webhook conversion is also defined here, which transforms VirtualServer resources from `v1alpha1` to `v1alpha2`. The CustomResourceDefinition “Host” and “RemoteServer” define global load balancing that was driven from the health of Kubernetes Services, running geographically in dispersed multiple Kubernetes clusters, was required.
cert-manager.io/v1	The **cert-manager** automates the management of TLS certificates for the Webhook service. Upon deployment, it establishes a **self-signed ClusterIssuer** (named `selfsigned-cluster-issuer`) and a **Certificate resource**. This infrastructure ensures all communication between the Kubernetes API server and the webhook server is encrypted via secure HTTPS.
ValidatingWebhookConfiguration	The **ValidatingWebhookConfiguration** registers webhooks to enforce technical validation logic during the creation or update of Kubernetes **Ingress** and **VirtualServer** resources. This prevents invalid configurations from being pushed to the **FortiADC**.
MutatingWebhookConfiguration	The **MutatingWebhookConfiguration** registers webhooks that automatically modify VirtualServer resources during creation or update. This is used to inject default values, ensure consistency, and provide warnings for non-optimal configurations.
webhook-service	The **webhook-service** exposes the webhook server to the Kubernetes control plane. It facilitates the conversion, mutation, and validation of Custom Resources, ensuring they are compatible with the FortiADC API.

Preparing for the Validating and Conversion Webhook Servers

Starting with version 3.1.0, the FortiADC Kubernetes Controller introduces **Validating and Conversion Webhook Servers.** This component acts as a security gatekeeper, ensuring that any Ingress or Custom Resource configuration is technically valid before it is accepted by the cluster and pushed to the FortiADC appliance.

Why cert-manager is Required

The Kubernetes API server requires an encrypted HTTPS connection to communicate with the webhook server. To facilitate this mandatory secure TLS connection, you must deploy cert-manager to automate the issuance and management of the required certificates.

Upon deployment, the controller uses cert-manager to establish a self-signed ClusterIssuer and a Certificate resource, ensuring all internal communication remains encrypted.

You must complete the cert-manager installation before installing or upgrading to FortiADC Kubernetes Controller 3.1.0. If cert-manager is not running, the controller pod will fail to initialize.

Installation Steps

Compatibility has been verified with cert-manager v1.19.1. Follow these steps to deploy it using Helm:

1. **Add the Jetstack Helm repository**:

```
helm repo add jetstack https://charts.jetstack.io
helm repo update
```

2. **Install cert-manager**: The following command creates the `cert-manager` namespace and installs the necessary Custom Resource Definitions (CRDs):

```
helm install --debug cert-manager jetstack/cert-manager \
       --namespace cert-manager \
       --create-namespace \
       --version v1.19.1 \
       --set crds.enabled=true
```

3. **Verify the installation**: Ensure the cert-manager pods are running before proceeding to the controller deployment:

```
kubectl get pods -n cert-manager
```

For more information, see the cert-manager documentation: https://cert-manager.io/docs/installation/.

Installing the Controller Using Helm Chart

After preparing the cluster with cert-manager, you can deploy the FortiADC Kubernetes Controller using Helm. This process creates the controller pod and all necessary supporting resources within your specified namespace.

Repository Setup

Starting with version 3.0.0, the Helm chart repository was renamed. If you are upgrading from an older version (2.x), you must remove the legacy repository before adding the new one.

```
helm repo remove fortiadc-ingress
helm repo add fortiadc-kubernetes-controller \
https://fortinet.github.io/fortiadc-kubernetes-controller/
helm repo update
```

Customization via values.yaml

The installation is governed by a `values.yaml` file that provides the default configurations. You can override these values during installation to customize node tolerations or define whether security parameters (such as WAF or Antivirus profiles) are optional or mandatory.

Below is an excerpt of the default configuration variables:

```
# Default values for fadc-k8s-ctrl.
# This is a YAML-formatted file.
# Declare variables to be passed into your templates.
# FortiADC Kubernetes Controller image from Dockerhub.com
image:
  repository: fortinet/fortiadc-ingress
  pullPolicy: IfNotPresent
  tag: "3.1.0"

nameOverride: ""
fullnameOverride: ""

serviceAccount:
  create: true
  annotations: {}
  name: "fortiadc-ingress"

podAnnotations: {}

podSecurityContext: {}

securityContext: {}
nodeSelector: {}

tolerations:
  - effect: "NoExecute"
```

```
    key: "node.kubernetes.io/not-ready"
    operator: "Exists"
    tolerationSeconds: 30
  - effect: "NoExecute"
    key: "node.kubernetes.io/unreachable"
    operator: "Exists"
    tolerationSeconds: 30

affinity: {}

# Define Ingress Class for FortiADC Kubernetes Controller
controller:
  ingressClassResource:
    name: "fadc-ingress-controller"
    enabled: true
    default: true
    controllerValue: "fortinet.com/fadc-ingress-controller"
# You can decide parameters defined in annotation of Ingress to be optional or mandatory.
# FortiADC Kubernetes Controller will check the parameter if it marks mandatory.
parameters:
  virtualServerNatSrcPool : "optional"
  virtualServerWafProfile : "optional"
  virtualServerAvProfile : "optional"
  virtualServerDosProfile : "optional"
  virtualServerCaptchaProfile : "optional"
  virtualServerPersistence : "optional"
  virtualServerFortiGSLB : "optional"
  openshiftRouteSupport: "no"
  enableStaticRouteSupport: "no"
webhook:
  useCertManager: true
  service:
    name: fad-webhook
    port: 443
    targetPort: 8443
  tlsSecretName: webhook-tls
  validatingWebhookName: validator.fadk8sctrl.fortinet.com
  mutatingWebhookName: mutator.fadk8sctrl.fortinet.com
  rules:
    validating:
      - name: validate-vs.fadk8sctrl.fortinet.com
        group: fadk8sctrl.fortinet.com
        version: v1alpha2
        resources:
          - virtualservers
        scope: "Namespaced"
        path: /validate-vs
      - name: validate-ingress.fadk8sctrl.fortinet.com
        group: networking.k8s.io
        version: v1
        resources:
          - ingresses
        scope: "Namespaced"
        path: /validate-ingress

    mutating:
      - name: mutate-vs.fadk8sctrl.fortinet.com
        group: fadk8sctrl.fortinet.com
        version: v1alpha2
        resources:
          - virtualservers
        scope: "Namespaced"
        path: /mutate-vs
```

To get the verbose output, add `--debug` option for all the Helm commands.

Installation

You can specify a particular Kubernetes namespace in which FortiADC Kubernetes Controller will be deployed.

By default, if no Kubernetes namespace is specified, the default namespace would be `default`. The `RELEASE_NAME` is the name you give to this chart installation:

```
helm install  [RELEASE_NAME] --namespace [Kubernetes NameSpace] \
fortiadc-kubernetes-controller/fadc-k8s-ctrl
```

Standard Installation: The following example installs the chart with the release name `first-release` in the namespace `fortiadc-ingress`:

```
user@control-plane-node ~> helm install first-release --namespace fortiadc-ingress \
fortiadc-kubernetes-controller/fadc-k8s-ctrl
```

Overriding Values: You can use `--set` flags to override values in the `values.yaml` file. For example, to set the virtualServerWafProfile parameter as mandatory:

```
user@control-plane-node ~> helm install --debug first-release \
--set parameters.virtualServerWafProfile="mandatory" \
--namespace fortiadc-ingress fortiadc-kubernetes-controller/fadc-k8s-ctrl
```

Moreover, you can create a new namespace and deploy FortiADC Kubernetes Controller within the namespace at the same time:

```
helm install first-release --namespace fortiadc-ingress \
--create-namespace --wait fortiadc-kubernetes-controller/fadc-k8s-ctrl
```

Upgrading the Chart

Use the upgrade command to move to a newer version. The `--install` option ensures the release is installed if it does not already exist.

You can specify the namespace with the `--namespace` option. Use `--install` option to install the release with RELEASE_NAME if it does not exist.

Note: The `--reset-values` option will remove all the user-supplied values. For example, if you had specified the virtualServerWafProfile parameter to be mandatory in a previous upgrade or install, the value will be reset to optional. The `--reset-values` option ensures all the values are directly from the updated repository.

```
helm repo update
helm upgrade --reset-values --debug -n [Kubernetes NameSpace] [RELEASE_NAME] \
fortiadc-kubernetes-controller/fadc-k8s-ctrl --install
```

Customizing during Upgrade: You can also change fields using the `--set` command during an upgrade:

To see which values you can change, refer to https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/charts/fadc-k8s-ctrl-3.1.0/values.yaml.

In the example below, you can override the value for the virtualServerWafProfile parameter to make it mandatory:

```
helm upgrade --debug -n [Kubernetes NameSpace] \
--set parameters.virtualServerWafProfile="mandatory" \
[RELEASE_NAME] fortiadc-kubernetes-controller/fadc-k8s-ctrl
```

Using the `--debug` option, allows you to verify your settings under the **USER-SUPPLIED VALUES** section of the Helm output.

```
Release "first-release" has been upgraded. Happy Helming!
NAME: first-release
LAST DEPLOYED: Mon Apr 18 09:07:46 2022
NAMESPACE: fortiadc-ingress
STATUS: deployed
REVISION: 2
TEST SUITE: None
USER-SUPPLIED VALUES:
parameters:
  virtualServerWafProfile: mandatory
```

Uninstalling the Chart

To uninstall the Helm Chart:

```
helm uninstall [RELEASE_NAME]
```

To uninstall the FortiADC Kubernetes Controller in the specified Kubernetes namespace:

```
helm uninstall [RELEASE_NAME] --namespace [Kubernetes NameSpace]
```

Verifying the Installation

After the deployment is complete, use the following commands to confirm that the FortiADC Kubernetes Controller is installed correctly and operational.

Check Helm Release Status

The helm history command provides the installation status and revision information for your specific release:

```
helm history -n [Kubernetes NameSpace] [RELEASE_NAME]
```

Example Output:

```
user@control-plane-node ~> helm history -n fortiadc-ingress first-release
REVISION        UPDATED                         STATUS          CHART                   APP
VERSION     DESCRIPTION
1               Tue Feb  8 05:37:33 2022        superseded      fadc-k8s-ctrl-0.1.0     1.0.0
      Install complete
```

Verify Deployment and Pod Health

Use the `kubectl` command to ensure the deployment is available and the controller pod is running without restarts:

```
kubectl get -n [namespace] deployments
```

```
kubectl get -n [namespace] pods
```

Example Output:

```
user@control-plane-node ~> kubectl get -n fortiadc-ingress deployments
NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
first-release-fadc-k8s-ctrl   1/1     1            1           8s
```

```
user@control-plane-node ~> kubectl get -n fortiadc-ingress pods
NAME                                          READY   STATUS    RESTARTS   AGE
first-release-fadc-k8s-ctrl-6447856856-h5skx   1/1     Running   0          8s
```

Review Controller Logs

Examine the logs to confirm the controller has initialized successfully and is ready to manage traffic:

```
kubectl logs -n [namespace] -f [pod name]
```

Expected Log Entry:

```
user@control-plane-node ~> kubectl logs -n fortiadc-ingress -f \
first-release-fadc-k8s-ctrl-6447856856-h5skx
```

```
Starting fortiadc ingress controller
time=="2021-10-13T06:27:56Z"level=info msg="Starting FortiADC Kubernetes Controller"
```

Confirm Custom Resource Definition (CRD) Registration

The installation automatically installs the *VirtualServer*, *RemoteServer*, and *Host* CRDs. These extend the Kubernetes API to support advanced FortiADC configurations. Verify they are applied using the following command:

```
kubectl get crds
```

Expected Results:

```
NAME                                    CREATED AT
virtualservers.fadk8sctrl.fortinet.com   2025-11-19T02:21:52Z
hosts.fadk8sctrl.fortinet.com            2025-11-19T02:21:52Z
remoteservers.fadk8sctrl.fortinet.com    2025-11-19T02:21:52Z
```

Kubernetes CNI Plugin

The Kubernetes cluster network model is implemented by Container Network Interface (CNI) plugins. Depending on your performance requirements and service types, you may choose to integrate FortiADC directly into this network.

Networking Connectivity Options

FortiADC supports two ways to reach your Kubernetes workloads. Your choice determines whether you need to configure an overlay tunnel.

Connectivity Method	Description	Tunnel Required?
NodePort	FortiADC sends traffic to the IP address of the Kubernetes Nodes. The cluster then handles internal routing to the pods.	**No**
ClusterIP	FortiADC sends traffic directly to the Pod IPs. This provides better performance and visibility.	**Yes**

Why an Overlay Tunnel is Required

While FortiADC Kubernetes Controller version 1.0 only supported **NodePort**, version 2.0 and later added support for **ClusterIP**.

Because ClusterIP services exist only within the cluster's private virtual network, an **overlay tunnel (VXLAN)** is required to bridge the FortiADC appliance to the Kubernetes pods. This tunnel allows FortiADC to participate in the cluster's overlay network and communicate directly with **EndpointSlices** (the collection of Pods implementing your service).

Supported CNI Plugins

If you choose to use **ClusterIP** services, you must establish a VXLAN tunnel based on your cluster's CNI plugin. FortiADC supports the following:

- **Flannel**: Supported starting from FortiADC version 7.4.0.
- **Calico**: Supported starting from FortiADC version 8.0.2.

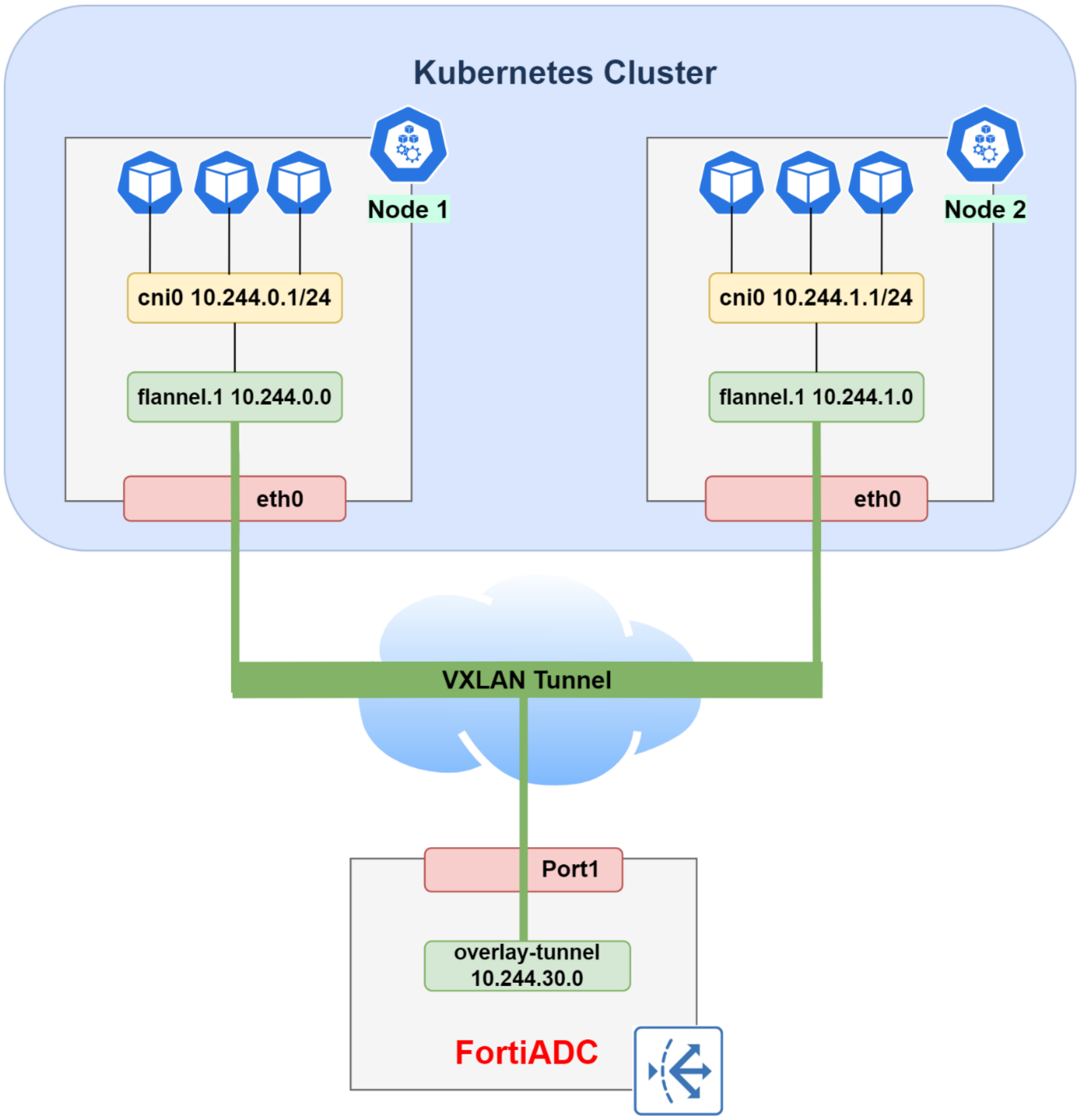
Configuration Guides (ClusterIP Only)

If you are using **NodePort**, you may skip this section. If you are using **ClusterIP**, follow the guide corresponding to your CNI plugin:

- Flannel VXLAN CNI on page 59: Steps to configure the VXLAN tunnel for clusters using Flannel.
- Calico VXLAN CNI on page 63: Steps to configure the VXLAN tunnel for clusters using Calico.

Flannel VXLAN CNI

To support **ClusterIP** services, FortiADC must participate in the Flannel overlay network. This is achieved by establishing a VXLAN tunnel and registering the FortiADC appliance as a "fake node" within the Kubernetes cluster. This registration allows the Kubernetes network to route traffic to the FortiADC even though it is not a physical server in the cluster.



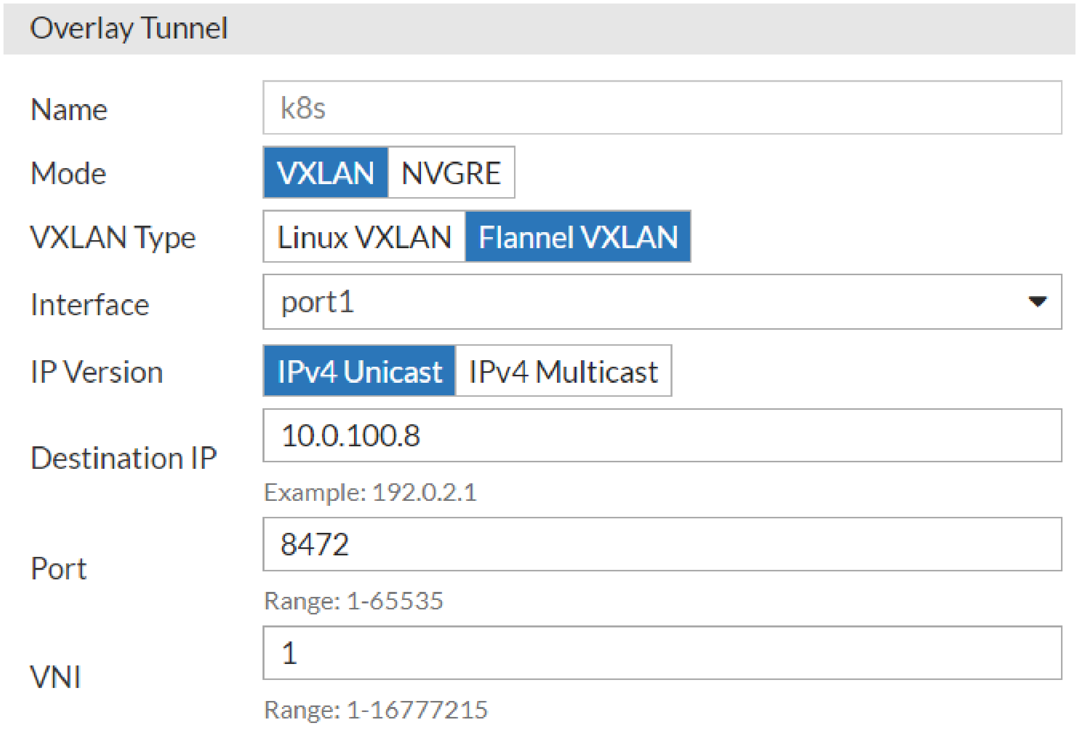
To configure the Flannel VXLAN CNI:

1. **Check Flannel Details**: Use the `kubectl` command to check the flannel VXLAN public IP and the VNI of the control plane node.

When using VXLAN backend, flannel uses UDP port **8472** for sending encapsulated packets.

```
kubectl describe node [NODE NAME] | grep flannel
```

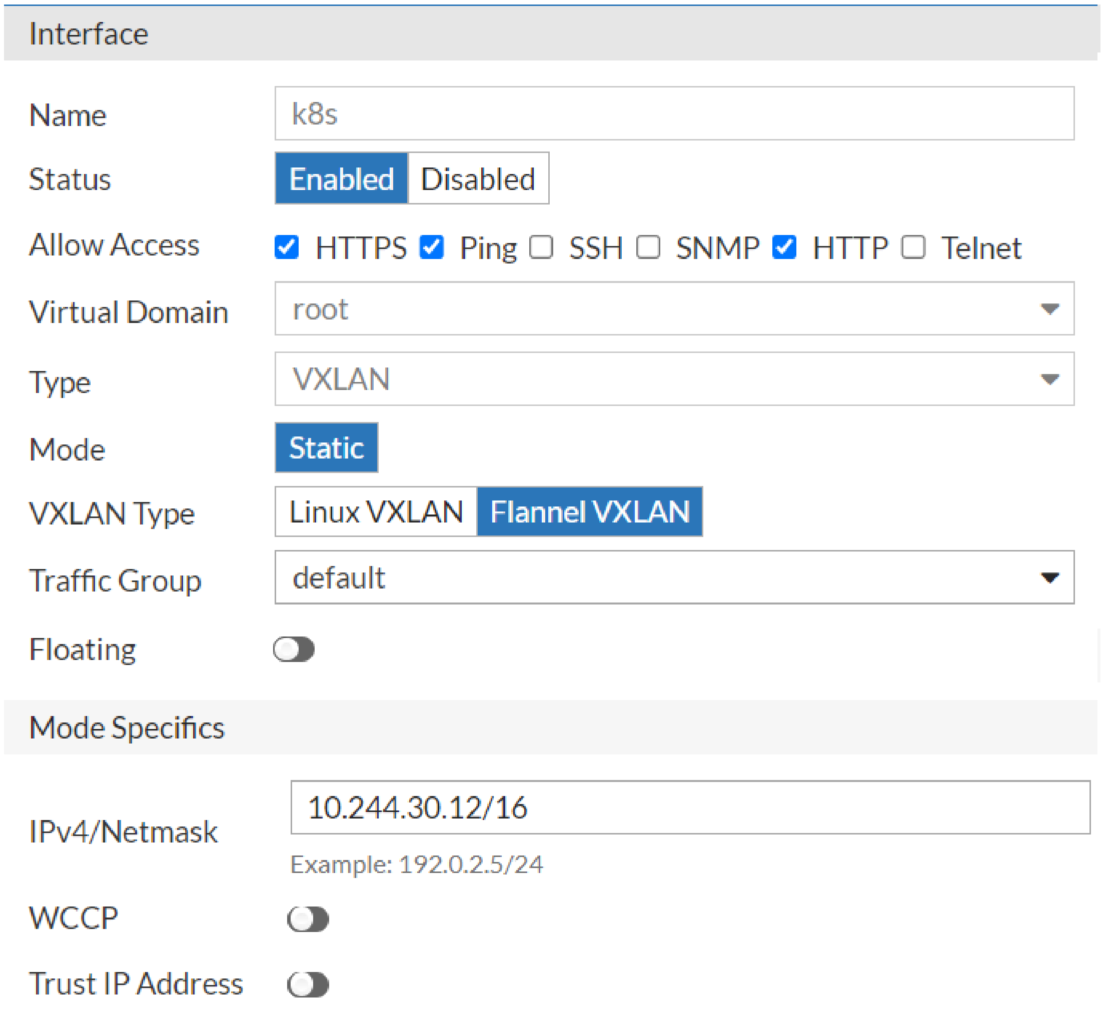
2. **Create Overlay Tunnel**: To integrate the FortiADC overlay tunnel with the flannel VXLAN, an overlay tunnel with the VXLAN type must be created on FortiADC.
   a. In FortiADC, go to **Network > Interface** and click the **Overlay Tunnel** tab.
   b. Create a new Overlay Tunnel configuration using the flannel VXLAN public IP obtained from the previous step as the **Destination IP**.
   c. Select **VXLAN** as the **Mode** and **Flannel VXLAN** as the **VXLAN Type**.



Overlay Tunnel

Name	k8s
Mode	[✓ VXLAN] NVGRE
VXLAN Type	Linux VXLAN [✓ Flannel VXLAN]
Interface	port1
IP Version	[✓ IPv4 Unicast] IPv4 Multicast
Destination IP	10.0.100.8 Example: 192.0.2.1
Port	8472 Range: 1-65535
VNI	1 Range: 1-16777215

3. **Configure Interface**: After the overlay tunnel is configured, you need to configure its interface to connect with your Kubernetes cluster.
   a. Go to **Network > Interface**. In the Interface page, locate the Interface configuration that share the same name as the overlay tunnel you just created.
   b. In the **Interface** configuration, specify the network interface IP and the netmask as the one used in your Kubernetes cluster CIDR netmask size.
   You can set the interface IP in any subset of the pod network if and only if the network subset is not used by another node. For example, we claim the subset `10.244.30.0/24` for FortiADC and set `10.244.30.12/16` as the interface IP/netmask.
   Allow **ping/http/https** traffic to go through the interface.



Interface	
Name	k8s
Status	[✓ Enabled] Disabled
Allow Access	✓ HTTPS ✓ Ping ☐ SSH ☐ SNMP ✓ HTTP ☐ Telnet
Virtual Domain	root
Type	VXLAN
Mode	[✓ Static]
VXLAN Type	Linux VXLAN [✓ Flannel VXLAN]
Traffic Group	default
Floating	off
Mode Specifics	
IPv4/Netmask	10.244.30.12/16 (Example: 192.0.2.5/24)
WCCP	off
Trust IP Address	off

4. **Retrieve MAC Address**: Go to the FortiADC CLI and use the `get system interface` command to get the VXLAN interface MAC address (`mac-addr`).

```
FortiADC-VM # get system interface k8s
type                          : vxlan
mode                          : static
vdom                          : root
redundant-master              :
ip                            : 10.244.30.12/16
allowaccess                   : https ping http
mtu                           : 1450
speed                         : auto
status                        : up
retrieve_physical_hwaddr      : disable
mac-addr                      : e4:74:eb:7c:c7:xx
flow-sniffer                  : disable
wccp                          : disable
trust-ip                      : disable
floating                      : disable
recv-seg-offload-override     : disable
```

```
send-seg-offload-override     : disable
vxlan-type                    : flannel_vxlan
```

5. **Deploy "Fake Node"**: Deploy FortiADC as the fake node installed with the flannel CNI plugin in your Kubernetes cluster using the following manifest:

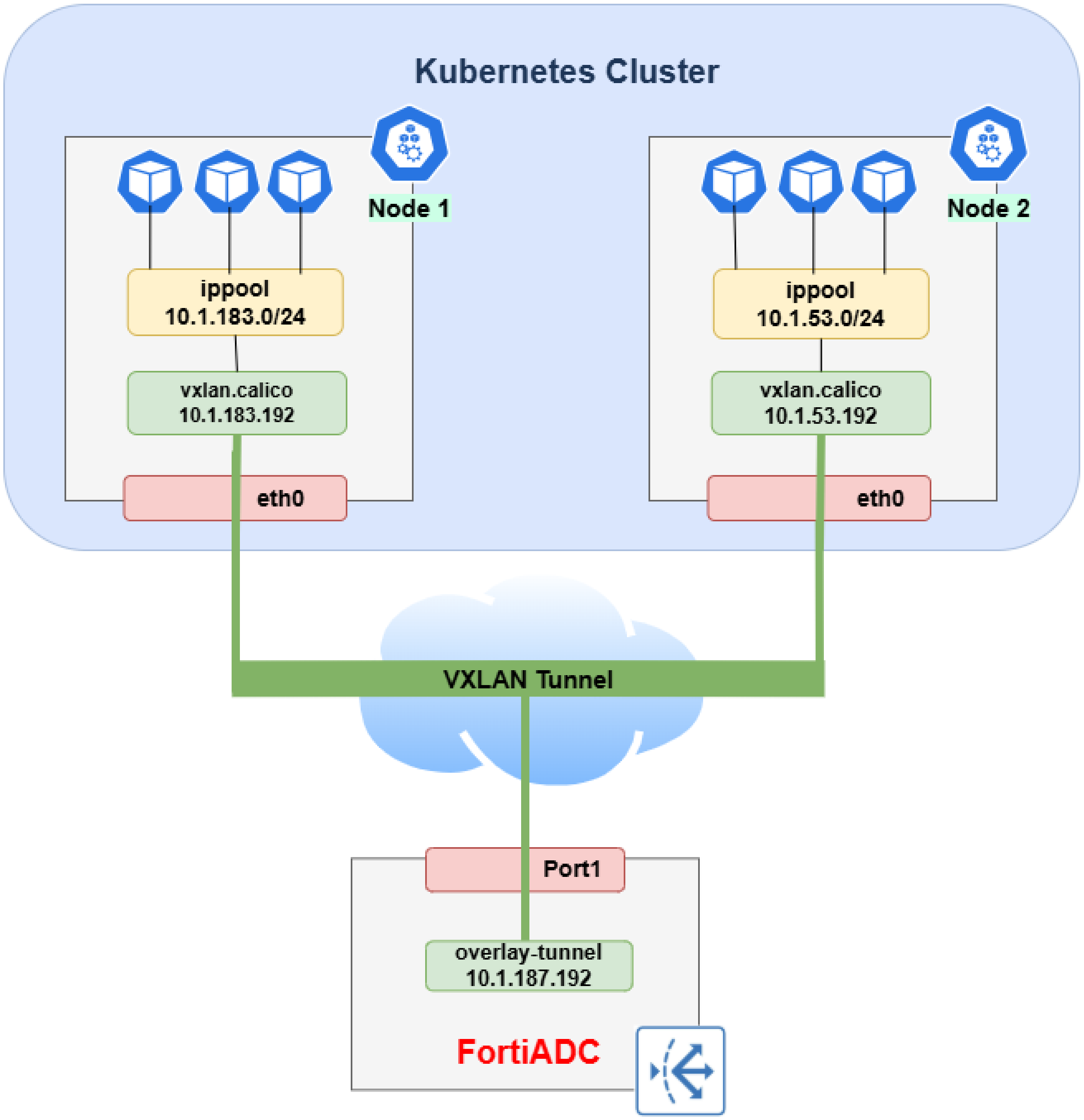
```
apiVersion: v1
kind: Node
metadata:
  name: fad
  labels:
    topology.kubernetes.io/zone: "fake-node"
  annotations:
    #Replace public-ip with outgoing interface IP of overlay tunnel
    flannel.alpha.coreos.com/public-ip: "10.0.100.133"
    #Replace VtepMAC with your VXLAN interface MAC
    flannel.alpha.coreos.com/backend-data: '{"VNI":1,"VtepMAC":"e4:74:eb:7c:c7:xx"}'
    flannel.alpha.coreos.com/backend-type: "vxlan"
    flannel.alpha.coreos.com/kube-subnet-manager: "true"
spec:
       podCIDR: "10.244.30.0/24"
```

Using the label `topology.kubernetes.io/zone` allows the fake node FortiADC to be distinguished from the zone where the control plane nodes and other worker nodes are located.

Since FortiADC is deployed as a fake node, the status of the fake node will always be shown as **"NotReady"**. This will not affect the connection between FortiADC and servers in the Kubernetes clusters.

Calico VXLAN CNI

To support **ClusterIP** services with **Calico**, FortiADC must be integrated into the **Calico VXLAN** overlay. This requires configuring Calico to use VXLAN mode and manually creating the network resources (IPAM blocks and handles) that Calico normally manages automatically for standard nodes.



Environmental Verification

Before proceeding, verify that your environment is configured to use **VXLAN mode**, as FortiADC Kubernetes Controller 3.1 only supports Calico CNI in this mode.

Calico deployments vary depending on the installation method and version. The controller supports deployments in either the `kube-system` or `calico-system` namespace, provided all resources are deployed consistently:

Installation Method	Common Namespace	Management Structure
Manifest-based	`kube-system`	Managed via **ConfigMaps** and **DaemonSets**.
Operator-based	`calico-system`	Managed via **Custom Resource Definitions (CRDs)**.

Configuration Workflow

Establishing the Calico VXLAN tunnel requires a three-phase "handshake" to ensure the FortiADC and the Kubernetes cluster can communicate:

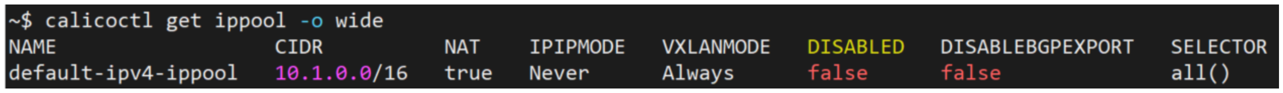
1. Phase 1: Calico Cluster Configuration: Modify the global CNI settings to ensure the cluster is running in VXLAN mode.
2. Phase 2: Deploy FortiADC as a "Fake Node": Manually register the FortiADC appliance and its required IPAM resources within the Kubernetes API.
3. Phase 3: FortiADC VXLAN Overlay Integration: Configure the FortiADC overlay tunnel and interface to match the cluster parameters.

Phase 1: Calico Cluster Configuration

Depending on your installation type, modify your cluster settings to ensure VXLAN is the primary encapsulation method.

For Manifest Installation:

- **IPPool**: VXLAN mode needs to be set as `Always` and `IPIPMODE` must be set as `Never`.

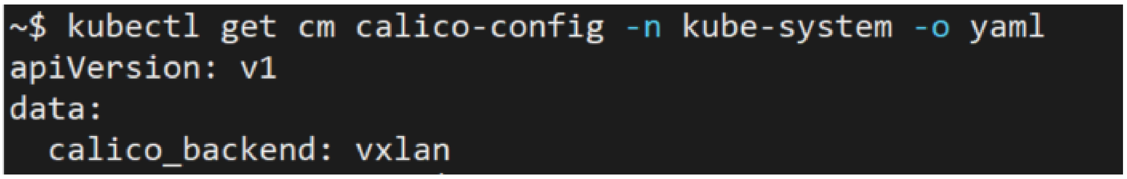


```
~$ calicoctl get ippool -o wide
NAME                 CIDR          NAT    IPIPMODE   VXLANMODE   DISABLED   DISABLEBGPEXPORT   SELECTOR
default-ipv4-ippool  10.1.0.0/16   true   Never      Always      false      false              all()
```

  To modify the ippool config:

```
~$  kubectl edit ippool default-ipv4-ippool
```

- **ConfigMap**: The `calico_backend` must be set to `vxlan` in ConfigMap `calico-config`.



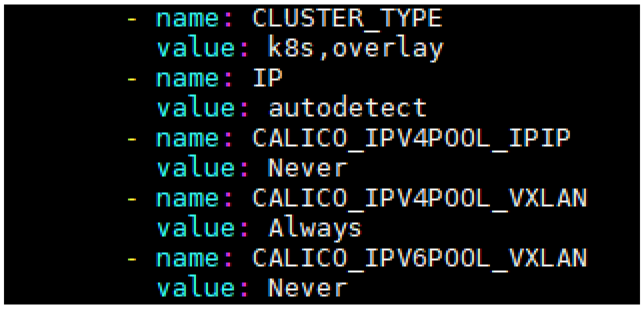
```
~$ kubectl get cm calico-config -n kube-system -o yaml
apiVersion: v1
data:
  calico_backend: vxlan
```

  To modify the `calico-config`:

```
~$  kubectl edit configmap calico-config -n kube-system
```

- **DaemonSet**: Ensure configurations are set correctly in `calico-node`.

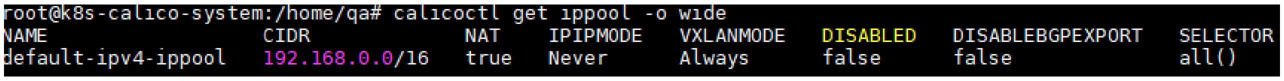
```
~$  kubectl edit daemonset calico-node -n kube-system
```



```
        - name: CLUSTER_TYPE
          value: k8s,overlay
        - name: IP
          value: autodetect
        - name: CALICO_IPV4POOL_IPIP
          value: Never
        - name: CALICO_IPV4POOL_VXLAN
          value: Always
        - name: CALICO_IPV6POOL_VXLAN
          value: Never
```

For Tigera Operator Installation:

- IPPool: VXLAN mode needs to be set as `Always` in ippool, and `IPIPMODE` must be set as `Never`.



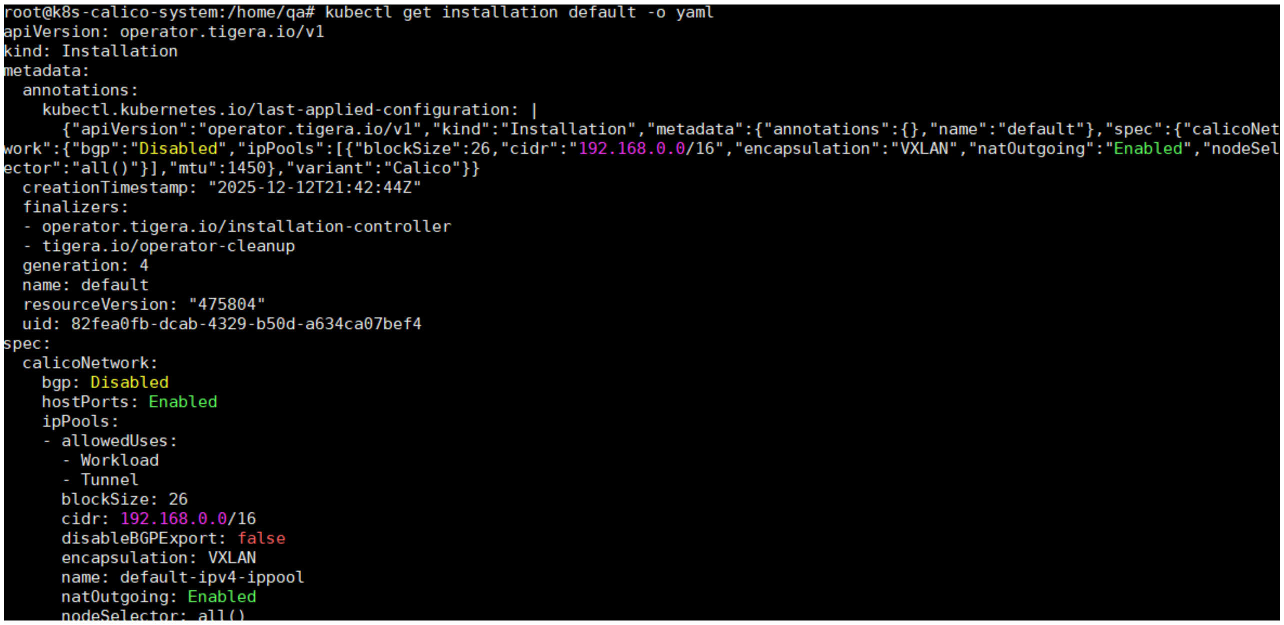
```
root@k8s-calico-system:/home/qa# calicoctl get ippool -o wide
NAME                 CIDR             NAT    IPIPMODE   VXLANMODE   DISABLED   DISABLEBGPEXPORT   SELECTOR
default-ipv4-ippool  192.168.0.0/16   true   Never      Always      false      false              all()
```

  To modify the ippool config:

```
~$  kubectl edit ippool default-ipv4-ippool
```

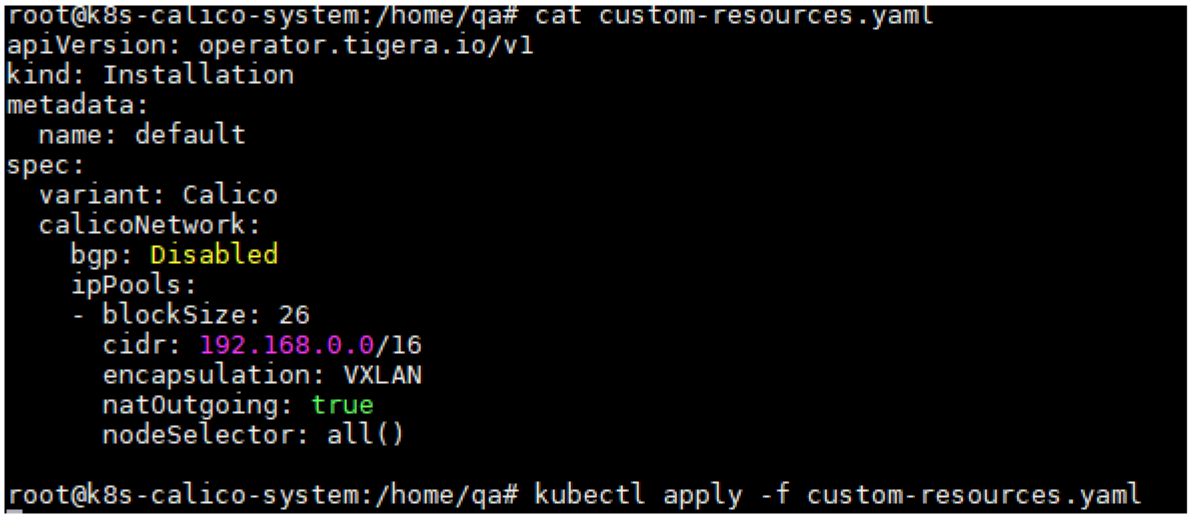
- **Installation Resource**: BGP needs to be disabled and encapsulation should be VXLAN in installation custom resource.

```
~$  kubectl get installation default -n calico-system -o yaml
```



```
root@k8s-calico-system:/home/qa# kubectl get installation default -o yaml
apiVersion: operator.tigera.io/v1
kind: Installation
metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"operator.tigera.io/v1","kind":"Installation","metadata":{"annotations":{},"name":"default"},"spec":{"calicoNet
work":{"bgp":"Disabled","ipPools":[{"blockSize":26,"cidr":"192.168.0.0/16","encapsulation":"VXLAN","natOutgoing":"Enabled","nodeSel
ector":"all()"}],"mtu":1450},"variant":"Calico"}}
  creationTimestamp: "2025-12-12T21:42:44Z"
  finalizers:
  - operator.tigera.io/installation-controller
  - tigera.io/operator-cleanup
  generation: 4
  name: default
  resourceVersion: "475804"
  uid: 82fea0fb-dcab-4329-b50d-a634ca07bef4
spec:
  calicoNetwork:
    bgp: Disabled
    hostPorts: Enabled
    ipPools:
    - allowedUses:
      - Workload
      - Tunnel
      blockSize: 26
      cidr: 192.168.0.0/16
      disableBGPExport: false
      encapsulation: VXLAN
      name: default-ipv4-ippool
      natOutgoing: Enabled
      nodeSelector: all()
```

  Example to install CustomResource in VXLAN mode for operator mode:



```
root@k8s-calico-system:/home/qa# cat custom-resources.yaml
apiVersion: operator.tigera.io/v1
kind: Installation
metadata:
  name: default
spec:
  variant: Calico
  calicoNetwork:
    bgp: Disabled
    ipPools:
    - blockSize: 26
      cidr: 192.168.0.0/16
      encapsulation: VXLAN
      natOutgoing: true
      nodeSelector: all()

root@k8s-calico-system:/home/qa# kubectl apply -f custom-resources.yaml
```

Phase 2: Deploy FortiADC as a "Fake Node"

The fake node configuration includes the `ipamhandle`, `ipamblock`, and `BlockAffinity` configurations. These are the network configurations Calico requires for a new node; the node IP address and subnet must be the same in all of these configurations.

By default, Calico uses `10.1.0.0/16` as the cluster-wide Pod CIDR and allocates a `/26` subnet to each node. You can set the `IPv4VXLANTunnelAddr` in any subset of the pod network if and only if another node does not use that network subset. For example, we claim the subset **10.1.187.192/26** for FortiADC.

Note: The `ipamhandle`, `ipamblock`, `BlockAffinity`, and node are connected by the `IPv4VXLANTunnelAddr`.

1. **Node Resource**

   The Node resource defines the FortiADC appliance's identity within the cluster. The `IPv4Address` annotation refers to the outgoing interface IP of your overlay tunnel, while the `IPv4VXLANTunnelAddr` represents the internal VXLAN interface IP.

```
apiVersion: v1
kind: Node
metadata:
  name: fadc-fake-node
  labels:
    kubernetes.io/hostname: fadc-fake-node
    kubernetes.io/os: linux
  annotations:
    #Replace public-ip with outgoing interface IP of overlay tunnel
    projectcalico.org/IPv4Address: 172.23.133.171/24
    #Replace IPv4VXLANTunnelAddr with your VXLAN interface IP
    projectcalico.org/IPv4VXLANTunnelAddr: 10.1.187.192
spec: {}
status:
    addresses:
    - address: 172.23.133.171
      type: InternalIP
    conditions:
```

```
  - type: Ready
    status: "True"
```

2. **BlockAffinity**

   The BlockAffinity resource creates a relationship between the fake node and its assigned CIDR block. The name and CIDR used here must match the node name and the `IPv4VXLANTunnelAddr` defined previously.

```
apiVersion: crd.projectcalico.org/v1
kind: BlockAffinity
metadata:
    #Combination of node name and IPv4VXLANTunnelAddr
    name: fadc-fake-node-10-1-187-192-26
spec:
    #Same IP as IPv4VXLANTunnelAddr, default Calico PodCIDR block is 26.
    cidr: 10.1.187.192/26
    #Node name
    node: fadc-fake-node
    deleted: "false"
    state: confirmed
```

3. **IPAMBlock**

   The IPAMBlock manages the actual IP allocations within the assigned subnet. It connects the `BlockAffinity`, the `IPAMHandle`, and the `Node` together to ensure Calico's IP Address Management tracks the fake node correctly.

```
apiVersion: crd.projectcalico.org/v1
kind: IPAMBlock
metadata:
    #Follow IPv4VXLANTunnelAddr
    name: 10-1-187-192-26
spec:
    #Follow IPv4VXLANTunnelAddr
    cidr: 10.1.187.192/26
    #Host is the fake node name
    affinity: host:fadc-fake-node
    strictAffinity: false
    allocations:
        - null
        - 0
        - null
    unallocated: [2,3,4,5,6,7,8,9,10]
    attributes:
           #Connect to IPAMHandle
        - handle_id: vxlan-tunnel-addr-fadc-fake-node
```

4. **IPAMHandle**

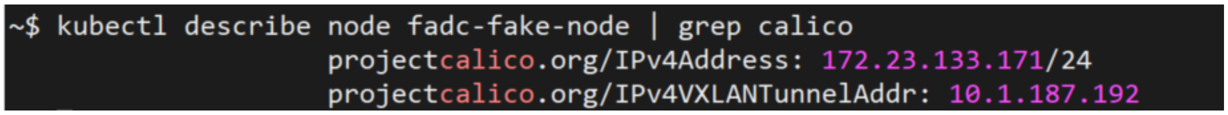
   The IPAMHandle serves as a reference pointer that allows the IPAMBlock to identify and track the VXLAN tunnel address allocation.

```
apiVersion: crd.projectcalico.org/v1
kind: IPAMHandle
metadata:
    name: vxlan-tunnel-addr-fadc-fake-node
spec:
    #The handleID is used by IPAMBlock
    handleID: vxlan-tunnel-addr-fadc-fake-node
    block:
        #Follow IPv4VXLANTunnelAddr
        10.1.187.192/26: 1
```

Phase 3: FortiADC VXLAN Overlay Integration

1. **Verify Cluster Network Details**:
   - Retrieve the Calico IPv4Address:

   ```
   kubectl describe node [NODE NAME] | grep calico
   ```

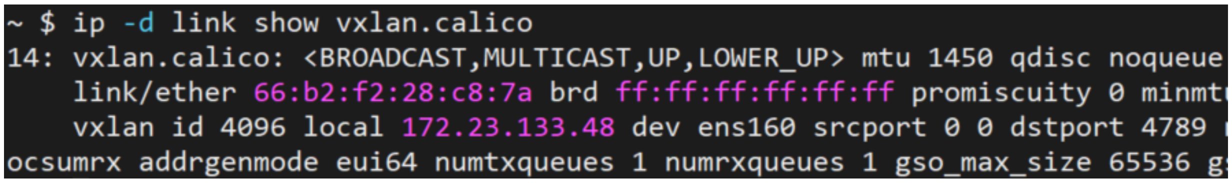
   

   ```
   ~$ kubectl describe node fadc-fake-node | grep calico
                       projectcalico.org/IPv4Address: 172.23.133.171/24
                       projectcalico.org/IPv4VXLANTunnelAddr: 10.1.187.192
   ```

   - Check the Calico VXLAN VNI and port of the control plane node.

     Calico creates a `vxlan.calico` interface on each node. Verify the VXLAN settings on a control plane node:

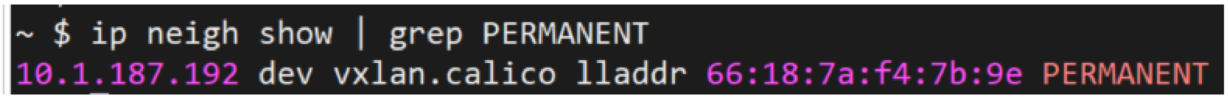
   ```
   ip -d link show vxlan.calico
   ```

   ```
   ~ $ ip -d link show vxlan.calico
   14: vxlan.calico: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue
       link/ether 66:b2:f2:28:c8:7a brd ff:ff:ff:ff:ff:ff promiscuity 0 minmt
       vxlan id 4096 local 172.23.133.48 dev ens160 srcport 0 0 dstport 4789
   ocsumrx addrgenmode eui64 numtxqueues 1 numrxqueues 1 gso_max_size 65536 g
   ```

2. **Retrieve the Assigned MAC Address**:
   - Locate the MAC address Calico has assigned for the FortiADC fake node by checking the ARP table on the control plane node:

   ```
   ip neigh show | grep PERMANENT
   ```

   ```
   ~ $ ip neigh show | grep PERMANENT
   10.1.187.192 dev vxlan.calico lladdr 66:18:7a:f4:7b:9e PERMANENT
   ```

3. **Configure the FortiADC Overlay Tunnel:**

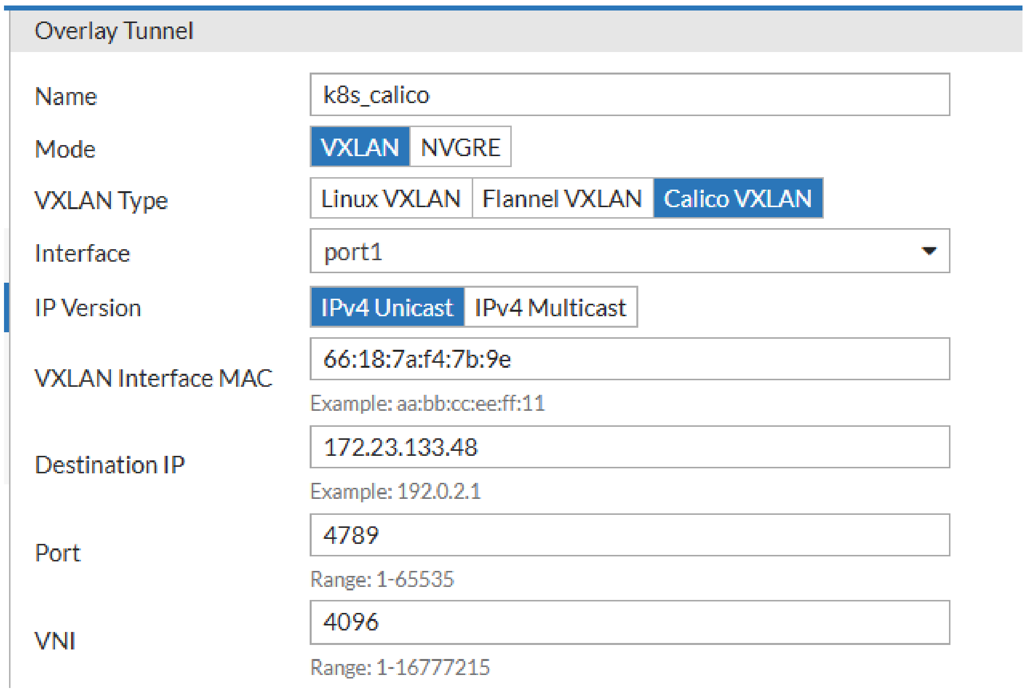
   To integrate the FortiADC overlay tunnel with the Calico VXLAN, an overlay tunnel with the VXLAN type must be created in FortiADC.

   a. In FortiADC, go to **Network > Interface** and click the **Overlay Tunnel** tab.

   b. Create a new Overlay Tunnel configuration using the Calico VXLAN IPv4Address obtained from the previous step as the **Destination IP**.

      Select **VXLAN** as the **Mode** and **CalicoVXLAN** as the **VXLAN Type**.

      Input the MAC address obtained in the previous step into the **VXLAN Interface MAC** field.


Overlay Tunnel

Name	k8s_calico
Mode	[✓ VXLAN] NVGRE
VXLAN Type	Linux VXLAN Flannel VXLAN [✓ Calico VXLAN]
Interface	port1
IP Version	[✓ IPv4 Unicast] IPv4 Multicast
VXLAN Interface MAC	66:18:7a:f4:7b:9e (Example: aa:bb:cc:ee:ff:11)
Destination IP	172.23.133.48 (Example: 192.0.2.1)
Port	4789 (Range: 1-65535)
VNI	4096 (Range: 1-16777215)

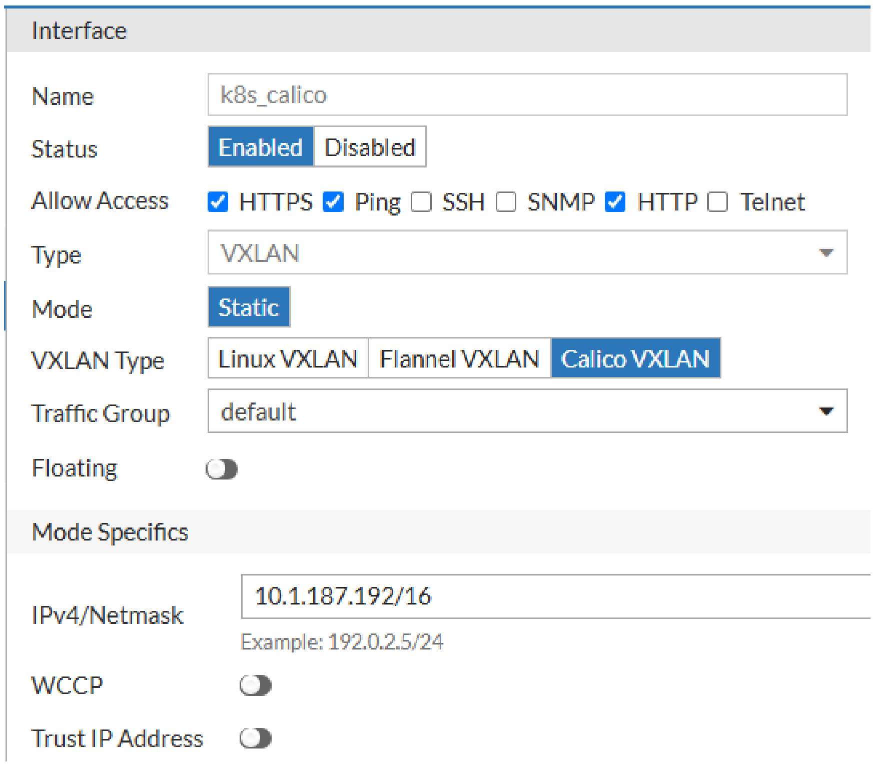
4. **Finalize the Interface Settings**:

   After the overlay tunnel is configured, you need to configure its interface to connect with your Kubernetes cluster.

   a. Go to **Network > Interface**. In the Interface page, locate the Interface configuration that share the same name as the overlay tunnel you just created.

   b. Specify the network interface IP and the netmask as the one used in your Kubernetes cluster CIDR netmask size. Also allow **ping/http/https** traffic to go through the interface.

      Set the interface IP to be the IPv4VXLANTunnelAddr just set in fake node. Since Calico uses `10.1.0.0/16` as the cluster-wide Pod CIDR by default, we set `10.1.187.192/16` as the interface IP/netmask.



Interface

Name	k8s_calico
Status	[✓ Enabled] Disabled
Allow Access	[✓ HTTPS] [✓ Ping] SSH SNMP [✓ HTTP] Telnet
Type	VXLAN
Mode	[✓ Static]
VXLAN Type	Linux VXLAN Flannel VXLAN [✓ Calico VXLAN]
Traffic Group	default
Floating	off

Mode Specifics

IPv4/Netmask	10.1.187.192/16 (Example: 192.0.2.5/24)
WCCP	off
Trust IP Address	off

Since FortiADC is deployed as a fake node, the status of the fake node will always be shown as “NotReady”. This will not affect the connection between FortiADC and servers in the Kubernetes clusters.

```
~$ kubectl get node fadc-fake-node
NAME              STATUS     ROLES    AGE    VERSION
fadc-fake-node    NotReady   <none>   7d3h
```

Configuration Parameters

The FortiADC Kubernetes Controller uses **Annotations** to define specific FortiADC behaviors that are not covered by standard Kubernetes resource fields. Annotations allow you to leverage enterprise-grade features like WAF profiles, persistence rules, and health checks directly from your Kubernetes manifests.

Authentication Mechanism

The FortiADC Kubernetes Controller manages **Ingress** and **Custom Resources** through REST API calls to the FortiADC appliance. To secure this communication, the controller requires valid authentication credentials.

The controller authenticates with the FortiADC via a two-part connection:

- **The Secret (The Key)**: A Kubernetes Secret that stores your FortiADC credentials.
- **The Annotation (The Pointer)**: The `fortiadc-login` annotation in your manifest that points the controller to that specific Secret.

Authentication Secret Setup

Before configuring annotations, create the secret containing your FortiADC username and password:

```
kubectl create secret generic fad-login -n [namespace]  \
--from-literal=username=admin --from-literal=password=[admin password]
```

In this example, the secret is named **fad-login**. This must be specified in the `fortiadc-login` annotation of your resource manifest to allow the controller to authenticate with the FortiADC.

The Kubernetes Secret must reside in the same Namespace as the resource (Ingress, Service, or CRD) that references it.

Annotation Categories

Configuration parameters are grouped by the resource type they modify. Select a category below to view the detailed parameter tables:

- Native Ingress Annotations on page 73 – Used with standard Kubernetes Ingress resources to determine how the controller deploys the Ingress, including Virtual Server IP assignments, interface selection, and security profiles (WAF/AV/DoS).
- Service Annotations on page 76 – Used within a Kubernetes Service manifest to automatically configure the **Real Server Pool** on the FortiADC for health check logic and SSL profiles.

- Custom Resource (CRD) Annotations on page 78 – Essential connectivity parameters for **VirtualServer**, **RemoteServer**, and **Host** resources. These allow the controller to identify and authenticate with the target FortiADC instance.

Native Ingress Annotations

Configuration parameters are required to be specified in the **Ingress** annotation to enable FortiADC Kubernetes Controller to determine how to deploy the Ingress resource.

Parameter	Description	Default
fortiadc-ip	The Ingress will be deployed on FortiADC with the given IP address or domain name. This parameter is required.	
fortiadc-admin-port	FortiADC HTTPS service port.	443
fortiadc-login	The Kubernetes secret name preserves the FortiADC authentication information. This parameter is required.	
fortiadc-vdom	Specify which VDOM to deploy the Ingress resource if VDOM is enabled on FortiADC.	root
fortiadc-ctrl-log	Enable/disable the FortiADC Kubernetes Controller log. Once enabled, FortiADC Kubernetes Controller will print the verbose log the next time the Ingress is updated.	enable
virtual-server-ip	The virtual server IP of the virtual server to be configured on FortiADC. This IP will be used as the address of the Ingress. This parameter is required.	
virtual-server-interface	The FortiADC network interface for the client to access the virtual server. This parameter is required.	
virtual-server-port	Default is 80. If TLS is specified in the Ingress, then the default is 443. Note: If the **fortiadc-ip** is the same as the **virtual-server-ip**, you should specify **virtual-server-port** to be other than 80/443 or change the system default reserved HTTP/HTTPS port on FortiADC. For more details, see the FortiADC Administration Guide on Management service ports.	80 for HTTP service. 443 for HTTPS service.
load-balance-method	Specify the predefined or user-defined method configuration name. For more details, see the FortiADC Administration Guide on load balancing methods.	LB_METHOD_ROUND_ROBIN
load-balance-profile	Default is LB_PROF_HTTP.	LB_PROF_HTTP LB_PROF_HTTPS

Parameter	Description	Default
	If TLS is specified in the Ingress, then the default is LB_ PROF_HTTPS.	
virtual-server-addr-type	IPv4 or IPv6.	ipv4
virtual-server-traffic-group	Specify the traffic group for the virtual server. For more details, see the FortiADC Administration Guide on traffic groups.	default
virtual-server-nat-src-pool	Specify the NAT source pool. For more details, see the FortiADC Administration Guide on NAT source pools.	
virtual-server-waf-profile	Specify the WAF profile name. For more details, see the FortiADC Administration Guide on WAF profiles.	
virtual-server-av-profile	Specify the AV profile name. For more details, see the FortiADC Administration Guide on AV profiles.	
virtual-server-dos-profile	Specify the DoS profile name. For more details, see the FortiADC Administration Guide on DoS profiles.	
virtual-server-captcha-profile	Specify the Captcha profile name. For more details, see the FortiADC Administration Guide on Captcha profiles. **Note**: This field is available if **WAF profile** or **DoS profile** is specified.	
virtual-server-fortiview	Enable/disable FortiView.	disable
virtual-server-traffic-log	Enable/disable the traffic log.	disable
virtual-server-wccp	Enable/disable WCCP. For more details, see the FortiADC Administration Guide on WCCP.	disable
virtual-server-persistence	Specify a predefined or user-defined persistence configuration name. For more details, see the FortiADC Administration Guide on persistence rules.	
virtual-server-fortigslb-publicip-type	Specify the public IP type for the virtual server as either IPv4 or IPv6.	ipv4
virtual-server-fortigslb-publicip	Specify the virtual server public IP address.	

Parameter	Description	Default
virtual-server-fortigslb-1clickgslb	Enable/disable the FortiGSLB One-click GSLB server.	disable
virtual-server-fortigslb-hostname	The **Host Name** option is available if **One-click GSLB Server** is enabled. Enter the hostname part of the FQDN. For example: `www`. **Note**: You can use `@` to denote the zone root. The value substitute for `@` is the preceding `$ORIGIN` directive.	
virtual-server-fortigslb-domainname	The **Domain Name** option is available if One-click GSLB Server is enabled. The domain name must end with a period. For example: `example.com`.	

For more details on configuring parameters with virtual-server prefix and load-balance prefix, please reference FortiADC Administration Guide on Configuring virtual servers.

Service Annotations

You can define the health check profile and SSL profile in the Kubernetes service annotation.

The health check profile and SSL profile will be automatically configured in the corresponding real server pool on FortiADC.

Parameter	Description	Default
health-check-ctrl	Enable/disable the health checking for the real server pool.	disable
health-check-relation	• AND – All of the selected health checks must pass for the server to be considered available. • OR – One of the selected health checks must pass for the server to be considered available.	
health-check-list	One or more health check configuration names. Concatenate the health check names with a space between each name. For example: "LB_HLTHCK_ICMP LB_HLTHCK_HTTP". For more details, see the FortiADC Administration Guide on health checks.	
real-server-ssl-profile	Specify the real server SSL profile name. Real server profiles determine settings for communication between FortiADC and the backend real servers. The default is NONE, which is applicable for non-SSL traffic. For more details, see the FortiADC Administration Guide on SSL profiles.	NONE
overlay_tunnel	Specify the overlay tunnel name. This is used for services with the ClusterIP type.	

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the service YAML example, follow this link:

https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/service_examples/default-http-backend.yaml

Here is an example service.yaml with health check parameters:

```
kind: Service
apiVersion: v1
metadata:
  labels:
```

```
  name: default-http-backend
  namespace: default
  annotations: {
    "health-check-ctrl" : "enable",
    "health-check-relation" : "OR",
    "health-check-list" : "LB_HLTHCK_ICMP",
    "real-server-ssl-profile" : "NONE"
  }
spec:
  type: NodePort
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: nginx
  sessionAffinity: None
```

Custom Resource (CRD) Annotations

Configuration parameters must be specified in the **VirtualServer**, **RemoteServer**, and **Host** annotations to allow the FortiADC Kubernetes Controller to identify the target FortiADC instance for deploying the VirtualServer, RemoteServer, and Host resource.

Parameter	Description	Default
fortiadc-ip	The VirtualServer/RemoteServer/Host will be deployed on FortiADC with the given IP address or domain name. This parameter is required.	
fortiadc-admin-port	FortiADC HTTPS service port.	443
fortiadc-login	The Kubernetes secret name preserves the FortiADC authentication information. This parameter is required.	
fortiadc-ctrl-log	Enable/disable the FortiADC Kubernetes Controller log. Once enabled, FortiADC Kubernetes Controller will print the verbose log the next time the VirtualServer/RemoteServer/Host is updated.	enable

Deployment Scenarios

This section provides configuration guides for integrating FortiADC with Kubernetes. These scenarios demonstrate how the FortiADC Kubernetes Controller translates cluster resources into application delivery configurations for L4/L7 load balancing and Global Load Balancing (GLB).

Implementation Prerequisites

Connectivity between FortiADC and your cluster must be established before deploying these scenarios. Verify your requirements based on the Kubernetes **Service Type** you intend to use:

- **NodePort Type**: If your services are exposed via `type: NodePort`, no additional network configuration is required. FortiADC will communicate with the nodes using their physical IP addresses.
- **ClusterIP Type (VXLAN)**: If you intend to route traffic directly to Pods using `type: ClusterIP`, you **must** first establish a VXLAN overlay tunnel. If this is not yet configured, please complete the steps for your specific CNI:
  - Flannel VXLAN CNI on page 59
  - Calico VXLAN CNI on page 63

How to Select a Scenario

The implementation guides are organized by the Kubernetes Resource Type you intend to manage. Use the logic below to identify the correct path for your deployment:

1. Standard Kubernetes Ingress & Services

If your objective is to use native Kubernetes objects for portability and standard workflows:

- **Ingress Deployment on page 80**: Best for standard web traffic requiring path-based routing (e.g., `/info`, `/hello`) and SSL termination.
- **Service Deployment and Lifecycle on page 85**: Explains how the controller handles the translation of **ClusterIP** and **NodePort** services and responds to service updates.
- **Handling Node Lifecycle Events on page 88**: Vital for maintaining high availability when cluster nodes are added, removed, or experience failure.

2. FortiADC Custom Resources (CRDs)

If your objective is to access advanced ADC features (WAF, IPS, Antivirus) or manage non-HTTP protocols:

- **Simple Fanout L7 VirtualServer on page 90**: Use this for advanced L7 routing that requires FortiADC security profiles (WAF/Captcha/AV) not supported by standard Ingress.
- **PostgreSQL L4 VirtualServer on page 95**: Use this to securely proxy database traffic (TCP) with SSL enabled.
- **Host-Based Global Load Balancing (GLB) on page 97**: Use this for multi-cluster or multi-region availability using DNS-based load balancing.

Ingress Deployment

The FortiADC Kubernetes Controller allows you to use the native Kubernetes Ingress resource to manage Layer 7 traffic. When an Ingress resource is deployed, the controller automatically provisions a Virtual Server on the FortiADC, configures Content Routing for path-based steering, and synchronizes real server pools with cluster endpoints.

Simple-Fanout Scenario

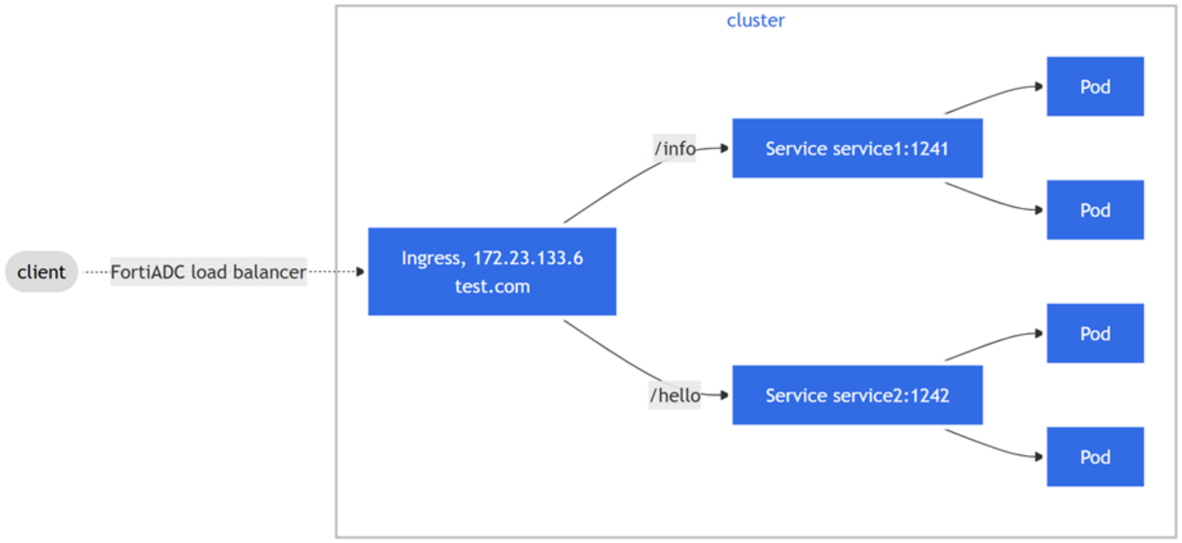
A "simple-fanout" deployment allows a single Load Balancer IP to route traffic to multiple backend services based on the URL path.

In this scenario, the client can access **service1** with the URL `https://test.com/info` and access **service2** with the URL `https://test.com/hello`.

Service1 defines a logical set of Pods with the label `run=sise`. Sise is a simple HTTP web server.

Service2 defines a logical set of Pods with the label `run=nginx-demo`. Nginx is also a simple HTTP web server. Services are deployed under the namespace `default`.

In this simple-fanout example, the Pods are exposed using the **NodePort** service type. You can also use the **ClusterIP** service type, or a combination of both, by defining `Service2` as `ClusterIP`. For more information, see Service Deployment and Lifecycle on page 85.



Deploy the Pods and expose the Services

```
Service1:
kubectl apply -f https://raw.githubusercontent.com/fortinet/fortiadc-kubernetes-
controller/main/service_examples/service1.yaml
```

```
Service2:
kubectl apply -f https://raw.githubusercontent.com/fortinet/fortiadc-kubernetes-
controller/main/service_examples/service2.yaml
```

Check the service1 and service2 you have deployed.

```
kubectl get service

NAME                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service1              NodePort    10.111.143.250   <none>        1241:31320/TCP   10m

service2              NodePort    10.109.117.79    <none>        1242:32075/TCP   2m59s
```

Deploy the Ingress

Define the Simple-fanout Ingress resource.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

To copy the **simple-fanout** Ingress YAML example, follow this link:
https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/ingress_examples/simple-fanout-example.yaml

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: simple-fanout-example
  annotations: {
    "fortiadc-ip" : "10.0.100.133",
    "fortiadc-login" : "fad-login",
    "fortiadc-vdom" : "root",
    "fortiadc-ctrl-log" : "enable",
    "virtual-server-ip" : "172.23.133.6",
    "virtual-server-interface" : "port1",
    "virtual-server-port" : "443",
    "load-balance-method" : "LB_METHOD_LEAST_CONNECTION",
    "load-balance-profile" : "LB_PROF_HTTPS"
  }
spec:
  ingressClassName: fadc-ingress-controller
  rules:
  - host: test.com
    http:
      paths:
      - path: /info
        pathType: Prefix
        backend:
          service:
            name: service1
            port:
              number: 1241
      - path: /hello
        pathType: Prefix
        backend:
          service:
            name: service2
            port:
              number: 1242
```

Deploy it with the `kubectl` command:

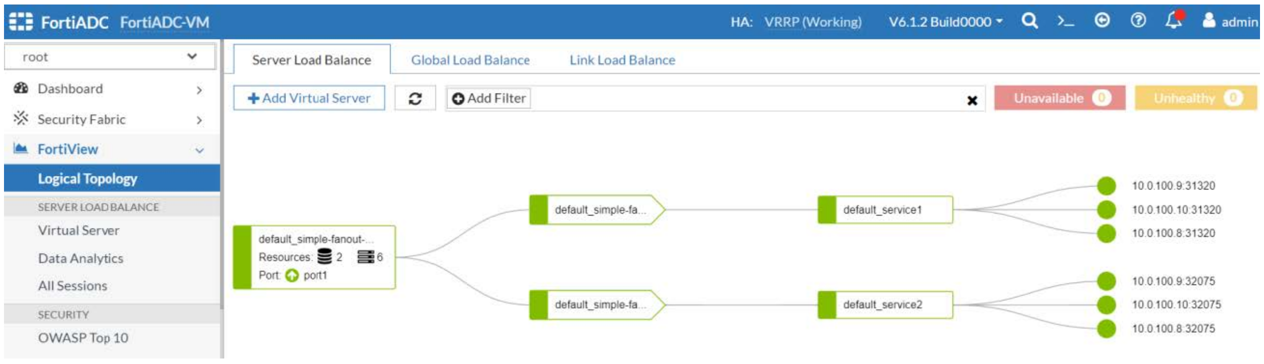
```
kubectl apply -f simple-fanout.yaml
ingress.networking.k8s.io/simple-fanout-example created
```

Get the information of the **simple-fanout-example Ingress** by using the `kubectl describe` command:

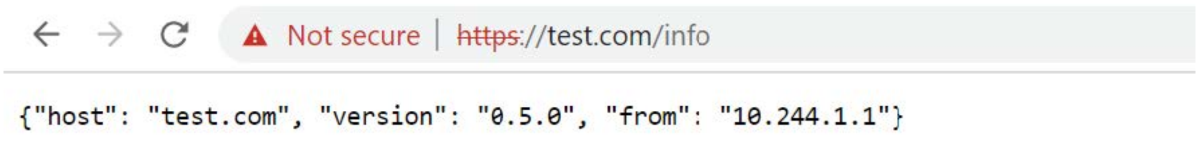
```
user@control-plane-node ~> kubectl describe ingress simple-fanout-example

Name:             simple-fanout-example


Namespace:        default

Address:          172.23.133.6


Default backend:  default-http-backend:80


Rules:
  Host        Path  Backends

  ----        ----  --------
  test.com

              /info   service1:1241 (10.244.1.16:9876)
              /hello  service2:1242 (10.244.12.26:80)


Annotations:  fortiadc-admin: admin
              fortiadc-ctrl-log: enable
              fortiadc-ip: 10.0.100.133
              fortiadc-vdom: root
              load-balance-method: LB_METHOD_LEAST_CONNECTION
              load-balance-profile: LB_PROF_HTTPS
              virtual-server-interface: port1
              virtual-server-ip: 172.23.133.6
              virtual-server-port: 443
Events:       <none>
```

FortiView

Check the deployed Ingress with FortiView.

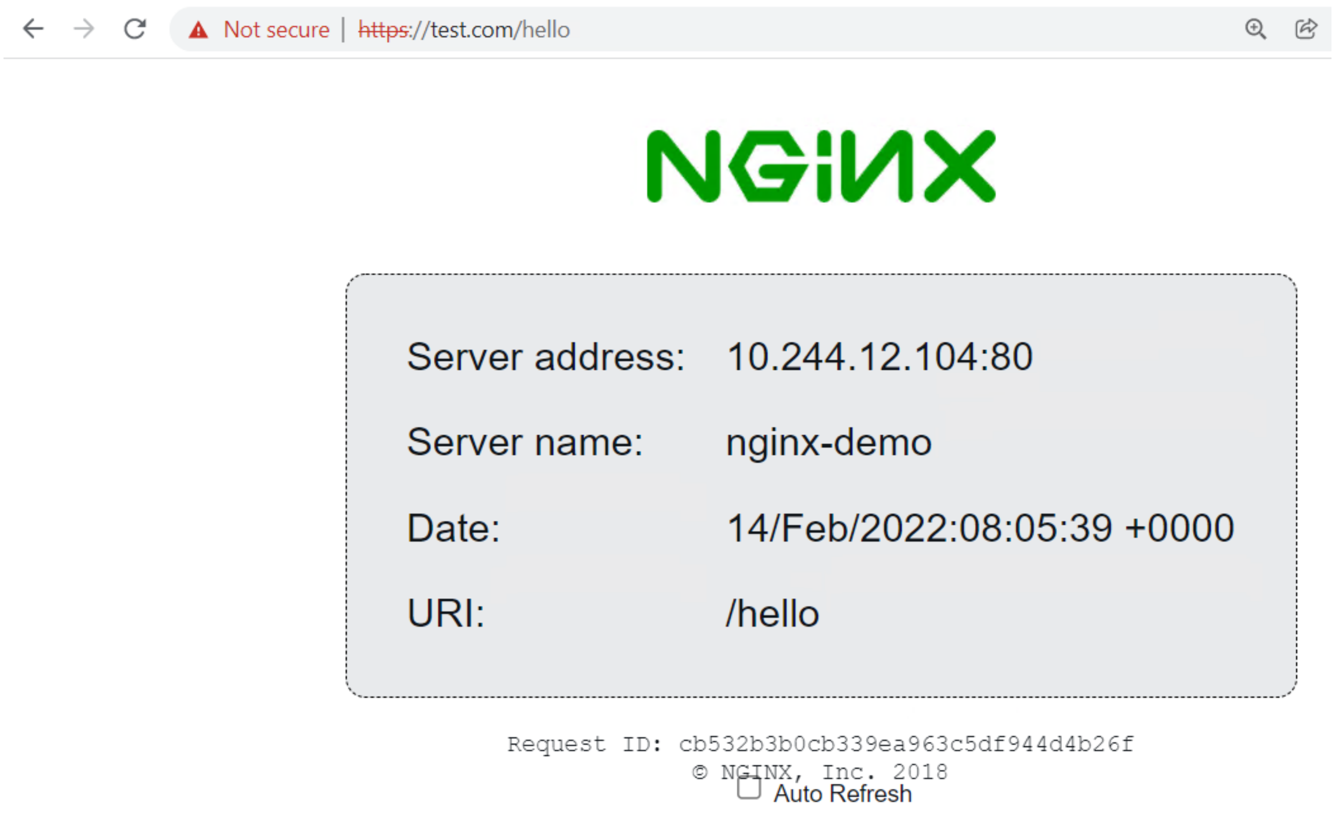


Try to access `https://test.com/info`.



```
{"host": "test.com", "version": "0.5.0", "from": "10.244.1.1"}
```

Try to access `https://test.com/hello`.



Server address: 10.244.12.104:80

Server name: nginx-demo

Date: 14/Feb/2022:08:05:39 +0000

URI: /hello

Request ID: cb532b3b0cb339ea963c5df944d4b26f
© NGINX, Inc. 2018
Auto Refresh

Update or delete the Ingress

To update an Ingress resource:

You can edit the `ingress.yaml.` and use `kubectl apply` or use the `kubectl edit` command.

```
kubectl edit ingress simple-fanout-example
```

To delete the Ingress resource:

```
kubectl delete ingress/simple-fanout-example
```

Service Deployment and Lifecycle

The FortiADC Kubernetes Controller continuously monitors the Kubernetes API for Service-related events. This section defines the operational constraints for service management and provides a template for deploying **ClusterIP** services via an overlay tunnel.

Operational Constraints

When configuring Kubernetes Services for use with FortiADC, the following technical limitations and behaviors apply:

- **Selective Monitoring**: The controller only tracks Service types and annotations for services explicitly referenced in a deployed `Ingress` or `VirtualServer` resource.
- **Single Port Limitation**: Currently, the controller does not support services with multiple exposed ports. Each service must map to a single logical port.
- **Dependency Handling**: If a Service referenced by an active Ingress/VirtualServer is deleted, Kubernetes does not issue a warning. The controller does not handle standalone Service delete events; the parent resource (Ingress/VS) must be updated or removed to clear the configuration on the FortiADC.
- **ClusterIP Support**:
  - **Controller v2.0**: Supports `Ingress` referencing ClusterIP services.
  - **Controller v3.0+**: Supports both `Ingress` and `VirtualServer` referencing ClusterIP services.

Deploying a ClusterIP Service (Overlay Mode)

Because ClusterIP addresses are only reachable within the cluster network, you must associate the service with a specific overlay tunnel defined on the FortiADC.

1. **Implementation Requirements**

   - **Networking**: A Flannel or Calico VXLAN tunnel must be pre-configured on the FortiADC.
   - **Annotations**: The `overlay_tunnel` annotation in the Service manifest must match the tunnel name created on the FortiADC.

2. **Manifest Example: NGINX via ClusterIP**

   The following YAML defines a deployment and a service. The service is annotated to use a VXLAN tunnel named `k8s` on the FortiADC. In this example, please ensure you have generated the required Kubernetes Secret and ConfigMap for your NGINX pods.

   Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

   Please follow this link to copy and modify the my-web service YAML example: https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/service_examples/my-web.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: my-web
  labels:
    run: my-web
  annotations: {
    "health-check-ctrl" : "enable",
    "health-check-relation" : "OR",
    "health-check-list" : "LB_HLTHCK_HTTPS",
    "real-server-ssl-profile" : "LB_RS_SSL_PROF_HIGH",
    "overlay_tunnel" : "k8s",
  }
spec:
  ports:
  - port: 443
    protocol: TCP
    name: https
    targetPort: https
  selector:
    app: my-web
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-web
spec:
  selector:
    matchLabels:
      app: my-web
  replicas: 2
  template:
    metadata:
      labels:
        app: my-web
    spec:
      volumes:
      - name: secret-volume
        secret:
          secretName: nginxsecret
      - name: configmap-volume
        configMap:
          name: nginxconfigmap
      containers:
      - name: nginxhttps
        image: nginx
        ports:
        - containerPort: 443
          name: https
        - containerPort: 80
          name: http
        volumeMounts:
        - mountPath: /etc/nginx/ssl
          name: secret-volume
        - mountPath: /etc/nginx/conf.d
          name: configmap-volume
```

Lifecycle Behavior: What Happens When...

Event	Controller Action
New Pod Added	The controller detects the new endpoint and adds the Pod IP/Port to the FortiADC Real Server Pool immediately.
Pod Terminated	The endpoint is removed from the FortiADC pool to prevent "blackholing" traffic.
Annotation Updated	The controller updates the FortiADC Real Server Pool settings (e.g., changing the health check method) to match the new annotation.

Handling Node Lifecycle Events

The FortiADC Kubernetes Controller maintains a synchronized map of the cluster's physical topology. When worker nodes are added, removed, or experience failure, the controller automatically updates the FortiADC configuration to ensure traffic is only directed to healthy, available infrastructure.

Automated Node Synchronization

The controller monitors the Kubernetes API for three specific node-level events. Its behavior depends on whether the application is exposed via **NodePort** or **ClusterIP**.

1. **Node Addition (Scale-Out)**

   When a new worker node joins the cluster:
   - **For NodePort Services**: The controller detects the new Node IP and automatically adds it as a new "Real Server" to the associated pools on the FortiADC.
   - **For ClusterIP Services**: While the node itself is new, the controller focuses on the **Pods** scheduled onto that node. As pods become `Ready`, their IPs are added to the FortiADC in the Real Server and Overlay Tunnel configurations.

2. **Node Deletion (Scale-In)**

   When a node is decommissioned or removed from the cluster:
   - The controller immediately identifies the node's removal and purges the corresponding Real Server entry from all FortiADC pools.
   - This prevents the "blackholing" of traffic that would occur if the ADC attempted to send requests to an IP that no longer exists.

3. **Node Health Status (`NotReady`)**

   The controller continuously monitors the `NodeCondition` of every worker node.
   - If a node enters a **NotReady** state (due to network partition, kubelet failure, or resource exhaustion), the controller treats this as a failure event.
   - **Traffic Suspension**: The controller will temporarily disable the Real Server associated with that node on the FortiADC until the node returns to a `Ready` status.

Comparison of Lifecycle Handling

Event	Impact on NodePort Services	Impact on ClusterIP Services
New Node Added	New Node IP added to FADC Pool.	No change until Pods are scheduled.
Node Deleted	Node IP removed from FADC Pool.	Pod IPs on that node removed from FADC.
Node `NotReady`	Node IP disabled on FADC.	FADC relies on individual Pod health checks.

Technical Considerations

- **Fake Nodes**: Note that the "Fake Node" created for **Calico** or **Flannel** integration will always show as `NotReady` in Kubernetes. The controller is specifically programmed to ignore the health status of this fake node to ensure the VXLAN tunnel remains active.
- **Draining Nodes**: If you use `kubectl drain`, the controller will see the Pods being evicted and update the FortiADC pools in real-time before the node actually shuts down.

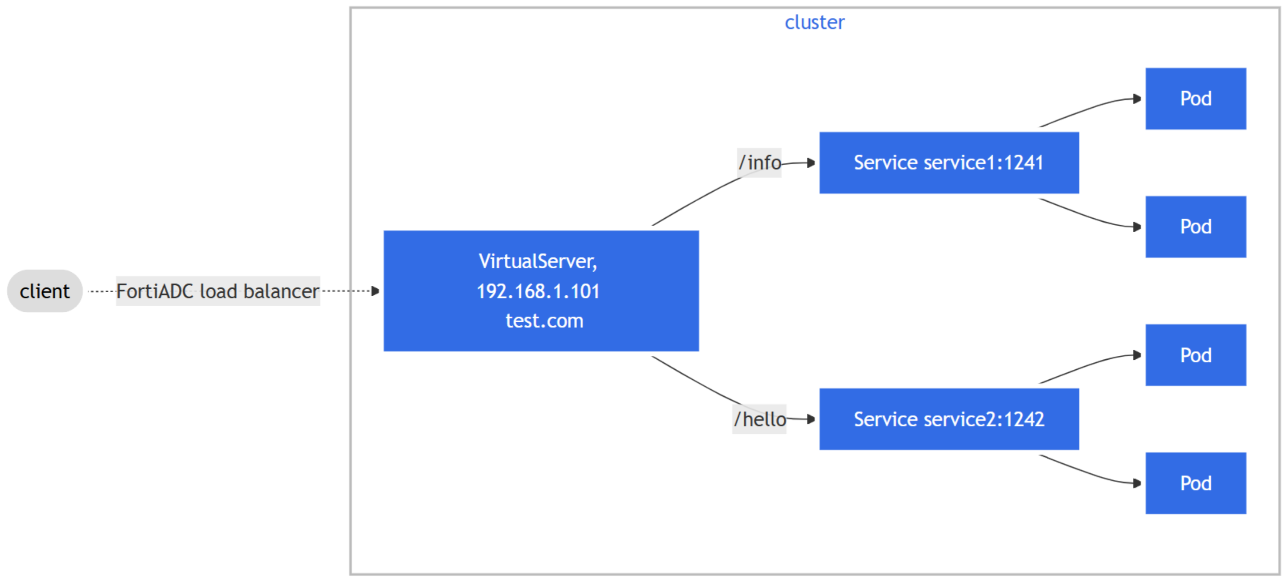
Simple Fanout L7 VirtualServer

While standard Kubernetes Ingress is suitable for basic traffic steering, the FortiADC **VirtualServer Custom Resource (CRD)** allows you to leverage the full suite of FortiADC Application Delivery features. This scenario demonstrates a "simple-fanout" configuration where a single hostname uses multiple routing paths to reach different services, while simultaneously applying enterprise-grade security profiles.

Scenario Overview

In this deployment, the VirtualServer CRD is used to orchestrate a Layer 7 environment with the following advanced configurations:

- **Content Routing**: Traffic is steered based on specific URL paths; requests for /info are sent to one service, while requests for /hello are directed to another.
- **Security Offloading**: Integrated **WAF (Web Application Firewall)**, **Antivirus**, and **Captcha** profiles are applied at the ADC level to protect the backend pods.
- **Global Availability**: One-click **GSLB** integration is enabled via the fortigslb specification fields.



Implementation Steps

1. **Define the VirtualServer Resource**

   The following manifest defines a VirtualServer that listens on port3 of the FortiADC. It maps test.com/info to service1 and test.com/hello to service2.

Due to PDF formatting limitations, the code example below would not retain indentations if copy and pasted directly into a YAML file. Without the proper indentations, the YAML will be invalid.

Please follow this link to copy and modify the **VirtualServer** YAML example: https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/customResource/virtualserver_simple_fanout.yaml

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha2
kind: VirtualServer
metadata:
  name: simple-fanout-virtualserver
  annotations: {
    "fortiadc-ip" : "172.23.133.110",
    "fortiadc-login" : "fad-login",
    "fortiadc-ctrl-log" : "enable",
    "fortiadc-admin-port": "443"
   }
  labels:
    fadcr: "true"
spec:
  addressType: ipv4
  address: 192.168.1.101
  port: 443
  interface: port3
  loadBalanceProfile: LB_PROF_HTTPS
  loadBalanceMethod: LB_METHOD_ROUND_ROBIN
  wafProfile: High-Level-Security
  captchaProfile: LB_CAPTCHA_PROFILE_DEFAULT
  avProfile: Antivirus-Profile
  trafficGroup: default
  fortiview: enable
  trafficLog: enable
  wccp: disable
  fortigslbPublicIpType: ipv4
  fortigslbPublicAddress: 203.0.113.1
  fortigslbOneClick: enable
  fortigslbHostName: samplehost
  fortigslbDomainName: example.com.
  contentRoutings:
    - name: route1
      host: test.com
      path: /info
      pathType: Prefix
      realServerPool:
        service: service1
        servicePort: 1241
        serviceNamespace: default
    - name: route2
      host: test.com
      path: /hello
      pathType: Prefix
      realServerPool:
        service: service2
        servicePort: 1242
        serviceNamespace: default
  natSourcePoolList:
    - name: nat-pool-1
  vdom: root
```

2. **Deploy and Verify**

   a. **Apply the configuration**:

   ```
   kubectl apply -f virtualserver_simple_fanout.yaml
   virtualserver.fadk8sctrl.fortinet.com/simple-fanout-virtualserver created
   ```

   b. **Describe the resource** to check the status and verify that the controller has parsed the parameters correctly. Get the information of the **simple-fanout-virtualserver** by using the `kubectl describe` command:

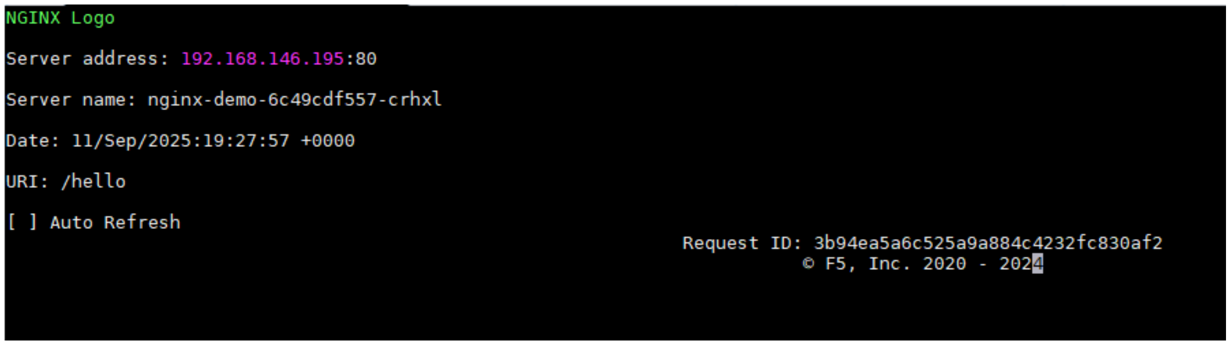
   ```
   # kubectl describe virtualserver simple-fanout-virtualserver
   Name:         simple-fanout-virtualserver
   Namespace:    default
   Labels:       fadcr=true
   Annotations:  fortiadc-admin-port: 443
                 fortiadc-ctrl-log: enable
                 fortiadc-ip: 172.23.133.110
                 fortiadc-login: fad-login
   API Version:  fadk8sctrl.fortinet.com/v1alpha2
   Kind:         VirtualServer
   Metadata:
     Creation Timestamp:  2025-09-11T19:06:24Z
     Generation:          1
     Resource Version:    30096049
     UID:                 687306af-c22e-4c9a-badf-06590f2927d5
   Spec:
     Address:          192.168.1.101
     Address Type:     ipv4
     Av Profile:       Antivirus-Profile
     Captcha Profile:  LB_CAPTCHA_PROFILE_DEFAULT
     Content Routings:
       Host:       test.com
       Name:       route1
       Path:       /info
       Path Type:  Prefix
       Real Server Pool:
         Service:            service1
         Service Namespace:  default
         Service Port:       1241
       Host:                 test.com
       Name:                 route2
       Path:                 /hello
       Path Type:            Prefix
   ```

```
     Real Server Pool:
       Service:               service2
       Service Namespace:     default
       Service Port:          1242
   Fortigslb Domain Name:     example.com.
   Fortigslb Host Name:       samplehost
   Fortigslb One Click:       enable
   Fortigslb Public Address:  203.0.113.1
   Fortigslb Public Ip Type:  ipv4
   Fortiview:                 enable
   Interface:                 port1
   Load Balance Method:       LB_METHOD_ROUND_ROBIN
   Load Balance Profile:      LB_PROF_HTTPS
   Port:                      443
   Traffic Group:             default
   Traffic Log:               enable
   Vdom:                      root
   Waf Profile:               High-Level-Securityaaa
   Wccp:                      disable
 Events:                      <none>
```

3. **Traffic Verification**

Once the controller has synchronized the resource with the FortiADC, verify the traffic flow:

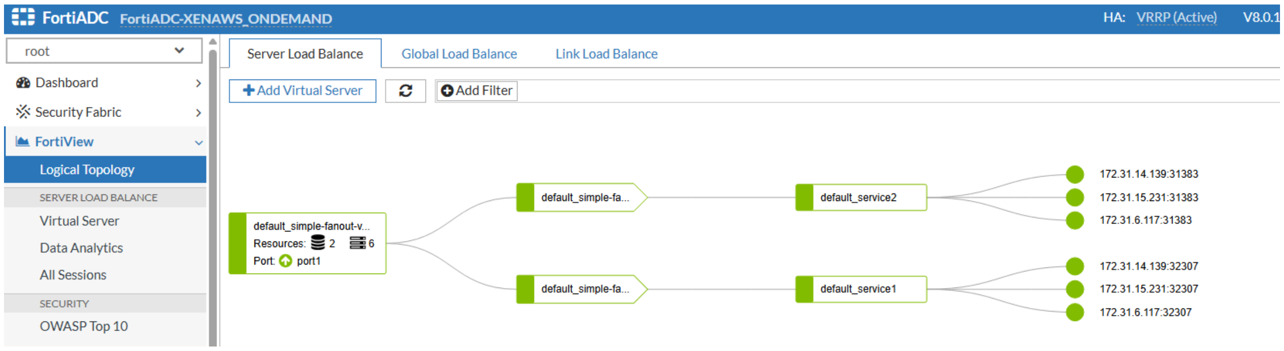
- **Test Path 1**: `curl -k https://test.com/info`
  - Expected Response: Host information from **service1**.
- **Test Path 2**: `curl -k https://test.com/hello`
  - Expected Response: Greetings from **service2**.



```
NGINX Logo

Server address: 192.168.146.195:80

Server name: nginx-demo-6c49cdf557-crhxl

Date: 11/Sep/2025:19:27:57 +0000

URI: /hello

[ ] Auto Refresh
                                                  Request ID: 3b94ea5a6c525a9a884c4232fc830af2
                                                            © F5, Inc. 2020 - 2024
```

FortiView Monitoring

You can monitor real-time performance, security events, and health status by logging into the FortiADC GUI and navigating to FortiView. This provides granular visibility into the traffic hitting your Kubernetes pods, including any blocked WAF or Antivirus violations.



4. Lifecycle Management

- **Update**: To modify security profiles or routing paths, you can edit the `virtualserver_simple_fanout.yaml.` and use `kubectl apply` or use the `kubectl edit` command.

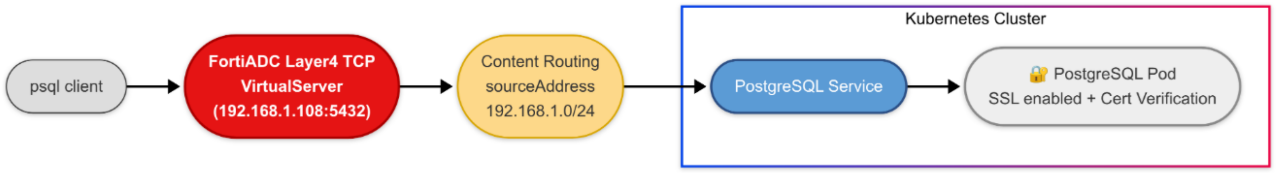
```
kubectl edit virtualserver simple-fanout-virtualserver
```

- **Delete**: To remove the configuration from both the cluster and the FortiADC hardware, run:

```
kubectl delete virtualserver/simple-fanout-virtualserver
```

PostgreSQL L4 VirtualServer

This scenario demonstrates how to securely expose a PostgreSQL database service. By using a Layer 4 TCP VirtualServer, FortiADC acts as a high-performance proxy, providing a single entry point for database traffic while offloading SSL processing and managing real-server persistence.


psql client
FortiADC Layer4 TCP VirtualServer (192.168.1.108:5432)
Content Routing sourceAddress 192.168.1.0/24
Kubernetes Cluster
PostgreSQL Service
PostgreSQL Pod SSL enabled + Cert Verification

Prerequisites

- **Cert-Manager**: This deployment relies on cert-manager for automated TLS certificate issuance. If it is not already present in your cluster, refer to Preparing for the Validating and Conversion Webhook Servers on page 51.
- **SNAT Pool**: Layer 4 VirtualServers on FortiADC use Full NAT by default. You must pre-configure a SNAT pool on the FortiADC hardware before deploying this CRD.

  Note: If using ClusterIP, the SNAT IP range should reside within the VXLAN interface subnet to ensure symmetric return routing.

Implementation Steps:

1. Deploy the PostgreSQL database on page 95
2. Deploy the TCP VirtualServer on page 95
3. Test the connection to the TCP VirtualServer on page 96

Deploy the PostgreSQL database

You can download the PostresSQL service example and change part of the definition based on your environment. Especially the overlay tunnel specified in service annotation if you want to define the service with clusterIP type.

```
curl -k \
https://raw.githubusercontent.com/fortinet/fortiadc-kubernetes-
controller/main/customResource/service_examples/postgresql_ssl_service.yaml \
-o postgresql_ssl_service.yaml
```

Then deploy the PostgresSQL with SSL enabled service with kubectl command

```
kubectl apply -f postgresql_ssl_service.yaml
```

Deploy the TCP VirtualServer

You can download the tcp-postgres-ssl-virtualserver example and modify the VirtualServer Annotation in virtualserver_ postgres_ssl.yaml to accommodate to your environment, ex: fortiadc-ip, fortiadc-admin-port, etc. Also, modify the

VirtualServer Spec, ex: address, contentRoutings.SourceAddress. Then, deploy the virtualserver with kubectl command.

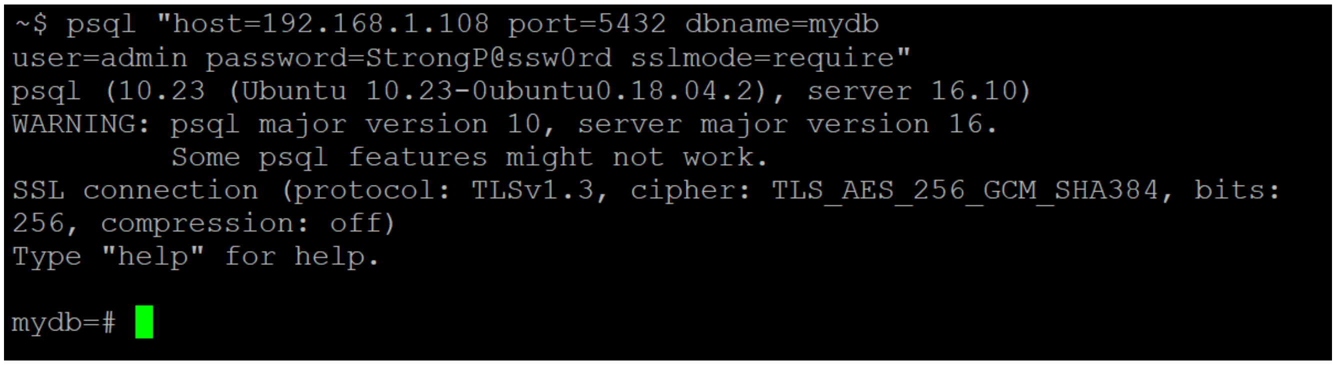
When configuring a Layer 4 VirtualServer, the default packet forwarding method is Full NAT. Therefore, the natSourcePoolList field must be specified in the custom resource. Before deploying the VirtualServer custom resource, you must pre-configure a SNAT pool on the FortiADC.

The SNAT IP address range should match the subnet of the VXLAN interface if service with clusterIP type is used, so that the return traffic is correctly routed.

```
curl -k \
https://github.com/fortinet/fortiadc-kubernetes-
controller/blob/main/customResource/virtualserver/virtualserver_postgres_ssl.yaml \
-o virtualserver_postgres_ssl.yaml

kubectl apply -f virtualserver_postgres_ssl.yaml
```

Test the connection to the TCP VirtualServer

Use psql to connect to the tcp virtualserver ip.



```
~$ psql "host=192.168.1.108 port=5432 dbname=mydb
user=admin password=StrongP@ssw0rd sslmode=require"
psql (10.23 (Ubuntu 10.23-0ubuntu0.18.04.2), server 16.10)
WARNING: psql major version 10, server major version 16.
         Some psql features might not work.
SSL connection (protocol: TLSv1.3, cipher: TLS_AES_256_GCM_SHA384, bits:
256, compression: off)
Type "help" for help.

mydb=#
```

Host-Based Global Load Balancing (GLB)

Global Load Balancing (GLB) allows you to distribute traffic across multiple application servers or FortiADC instances under a single Fully Qualified Domain Name (FQDN). This scenario demonstrates the **Simple Host** setup, where FortiADC uses DNS-based logic to return the most appropriate IP address to a client.

Setup the GLB settings

Before starting to deploy GLB-related CRDs, we need to make sure that the GLB function in FortiADC is operational.

- Enable the Global DNS Configuration
  - Go to **Global Load Balance > Zone Tools**. In the **General Settings** tab, enable **Global DNS Configuration**.
- Create two virtual servers
  - Go to **Server Load Balance > Virtual Server**. In the **Virtual Server** tab, create two new virtual servers named v1 and v2.

Deploy the Servers

You can download the server example and change part of the definition based on your environment. Especially the RemoteServer annotation if you want to define the FortiADC with fortiadc-ip, fortiadc-admin-port, etc.

```
curl -k https://raw.githubusercontent.com/fortinet/fortiadc-kubernetes-
controller/main/customResource/glb/remoteserver_slb.yaml -o remoteserver_slb.yaml
```

```
curl -k https://raw.githubusercontent.com/fortinet/fortiadc-kubernetes-
controller/main/customResource/glb/remoteserver_host.yaml -o remoteserver_host.yaml
```

Deploy it with the kubectl command:

```
kubectl apply -f remoteserver_slb.yaml
remoteserver.fadk8sctrl.fortinet.com/fortiadc-rs1 created
```

```
kubectl apply -f remoteserver_host.yaml
remoteserver.fadk8sctrl.fortinet.com/generic-rs1 created
```

Deploy the Host

You can download the host example and modify parts of the definition to match your environment, especially the Host annotations used to define the FortiADC settings such as `fortiadc-ip` and `fortiadc-admin-port`.

```
curl -k https://raw.githubusercontent.com/fortinet/fortiadc-kubernetes-
controller/main/customResource/glb/host.yaml -o host.yaml
```

Deploy it with the kubectl command:

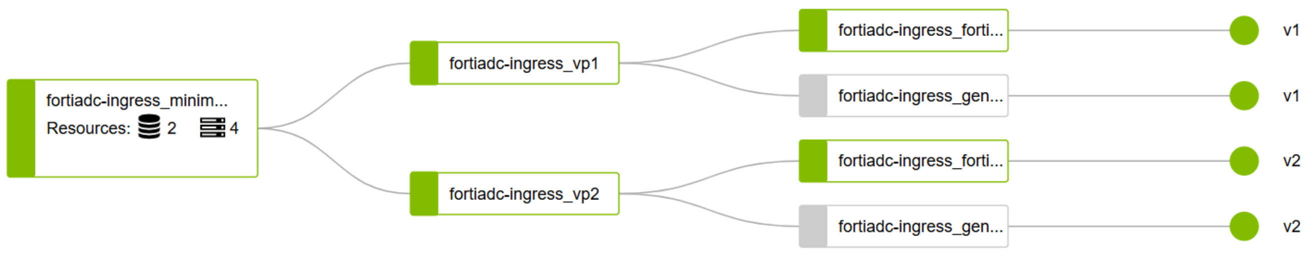
```
kubectl apply -f host.yaml

host.fadk8sctrl.fortinet.com/simple-host created
```

Get the information of the simple-host by using the `kubectl describe` command:

```
# kubectl describe host simple-host

Name:          simple-host

Namespace:     default

Labels:        fadcr=true

Annotations:   fortiadc-admin-port: 443

               fortiadc-ctrl-log: enable

               fortiadc-ip: 172.31.5.197

               fortiadc-login: fad-login

API Version:   fadk8sctrl.fortinet.com/v1alpha1

Kind:          Host

Metadata:

  Creation Timestamp:  2025-11-24T09:52:50Z

  Generation:          1

  Resource Version:    1797237

  UID:                 54fd070f-c353-4e7f-9a32-1797a289382b

Spec:

  Domain:        host1.com.

  feedbackIPv4:  0.0.0.1

Events:                       <none>
```

FortiView

Check the deployed Host and RemoteServer with FortiView.



To access the FQDN `www.host1.com` using the `dig` tool:

```
dig @192.168.1.108 www.host1.com +short
20.20.20.2
```

Update or delete the Host

To update a host resource:

You can edit the host.yaml. and use kubectl apply or use the kubectl edit command.

```
kubectl edit host simple-host
```

To delete the host resource:

```
kubectl delete host/simple-host
```

Debug

By default, the FortiADC Kubernetes Controller records the process of the Ingress and Fortinet-defined custom resources implementation in verbose mode. The debug log shows the check of annotation parameters and the process of calling the REST API to FortiADC when deploying, updating, or deleting the Ingress resources.

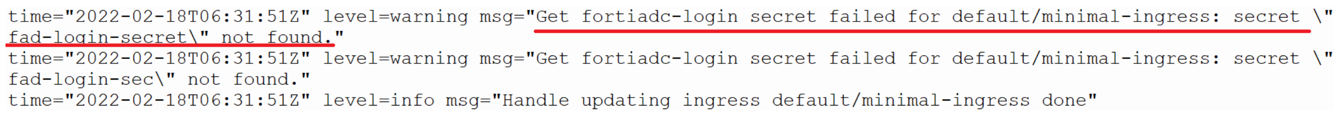
The verbose log can be enabled or disabled for any particular Ingress or Fortinet-defined custom resources. This is determined by the status of the fortiadc-ctrl-log configuration parameter.

For more information, see Configuration Parameters on page 71.

To see the log, you can use the `kubectl logs` command:

```
kubectl logs -n [namespace] -f [FortiADC Kubernetes Controller pod name]
```

The log shows which problem you encounter. For example, the below log shows that you do not have the correct FortiADC Authentication Secret in the Kubernetes cluster.



```
time="2022-02-18T06:31:51Z" level=warning msg="Get fortiadc-login secret failed for default/minimal-ingress: secret \"
fad-login-secret\" not found."
time="2022-02-18T06:31:51Z" level=warning msg="Get fortiadc-login secret failed for default/minimal-ingress: secret \"
fad-login-sec\" not found."
time="2022-02-18T06:31:51Z" level=info msg="Handle updating ingress default/minimal-ingress done"
```

Based on the error message, you can correct it and use the `kubectl apply` or `edit` command to reconfigure the resource.

Some troubleshooting steps may require restarting the FortiADC Kubernetes Controller. For example, the FortiADC Kubernetes Controller may not connect to FortiADC after changing the network firewall rule. To fix this type of environment issue, you can restart the FortiADC Kubernetes Controller by using the following command:

```
kubectl -n [namespace] rollout restart deployment/[ FortiADC Kubernetes Controller deployment
name]
```

FAQ

1) What should I do if I find a Kubernetes API object is supported by multiple API groups?

You may encounter this scenario when upgrading your Kubernetes cluster and the higher Kubernetes version has deprecated some of the API groups.

For example, extensions/v1beta1/Ingress is entirely deprecated by networking.k8s.io/v1/Ingress in Kubernetes v1.22. You may find extensions/v1beta1/Ingress and networking.k8s.io/v1/Ingress both exist in your system when upgrading the Kubernetes cluster from v1.16 to v1.20.

In this case, you have to disable the API group extensions/v1beta1 to ensure the system use the API group networking.k8s.io/v1 that is supported by FortiADC Kubernetes Controller. After disabling the API group, the Kubernetes API server needs to be restarted to apply the changes.

Follow the steps below:

1. Edit `/etc/kubernetes/manifests/kube-apiserver.yaml`
2. Under spec.containers.command, add `--runtime-config=extensions/v1beta1=false`
3. Restart the Kubernetes API Server using `systemctl restart kubelet.service`

For more information on enabling/disabling deprecated API groups, see https://kubernetes.io/docs/reference/using-api/#enabling-or-disabling.

2) Why does FortiADC Kubernetes Controller not spin up when failover occurs?

First, check if the NotReady Node is marked with taints, **node.kubernetes.io/unreachable:NoExecute** or **node.kubernetes.io/not-ready:NoExecute**.

If both taints are missing, check the following:

1. If the Kubernetes version is earlier than 1.19.9, upgrade the Kubernetes version to avoid this issue.
   For more information, see https://github.com/kubernetes/kubernetes/issues/97100.
2. If the percentage of NotReady nodes in the same zone is greater than the value of unhealthyZoneThreshold (default is 55%), then taints with the NoExecute effect may not apply to the NotReady nodes.
   For details, see https://kubernetes.io/docs/reference/command-line-tools-reference/kube-controller-manager/.

3) How do I resolve a `TLS Handshake error: bad certificate`?

This error typically occurs during a reinstallation or upgrade. Because **cert-manager** automatically issues the webhook certificate, the old certificate may persist in the cluster after the controller is uninstalled, leading to a mismatch.

The controller logs will show a `TLS handshake error` and a `remote error: tls: bad certificate`.

To resolve this, uninstall the FortiADC Kubernetes Controller and manually delete the leftover TLS secrets in the controller's namespace before reinstalling:

1. Run `helm uninstall` for the FortiADC Kubernetes Controller.
2. Delete the leftover TLS secrets (`webhook-tls` / `webhook-tls-ca`). The secret is saved in the namespace the same with FortiADC Kubernetes Controller
3. Then perform a fresh `helm install`.

4) How should I handle CRD deletion hangs or webhook timeouts?

Starting with v3.1, a Conversion Webhook Server checks all VirtualServer custom resource (CR) operations. If the controller is uninstalled while CRs still exist, Kubernetes may hang while waiting for a response from the offline webhook.

To avoid this, follow this specific deletion order:

1. Delete all VirtualServer CRs created by the controller.
2. Uninstall the controller via Helm:

   ```
   helm uninstall <release-name> -n <namespace>
   ```

3. Delete the VirtualServer CRDs once all VirtualServer CRs and the controller are fully removed:

   ```
   kubectl delete crd <crd-name>
   ```

If a CRD is already stuck, you can force the deletion by patching the resource to remove its finalizers:

```
kubectl patch crd virtualservers.fadk8sctrl.fortinet.com \
-p '{"metadata":{"finalizers":[]}}' --type=merge
```

5) How do I resolve kubeadm upgrade errors related to Fake Nodes?

When using FortiADC as a **Fake Node** for CNI integration, `kubeadm` may fail to parse the kubelet version of that logical node during a cluster upgrade, resulting in a `couldn't parse kubelet version` error.

You can bypass this preflight validation by **applying the --force flag** to your upgrade command:

```
sudo kubeadm upgrade apply <version> --force
```

6) How do I handle node draining issues?

During a node drain, Kubernetes may block the process if the FortiADC Kubernetes Controller is using local storage (**emptyDir**) for temporary caching. The error will state that it `cannot delete Pods with local storage`.

Since this local storage is only used for temporary cache, it is safe to delete. **Add the --delete-emptydir-data flag** to the drain command to proceed:

```
kubectl drain <node-name> --ignore-daemonsets --delete-emptydir-data
```

www.fortinet.com