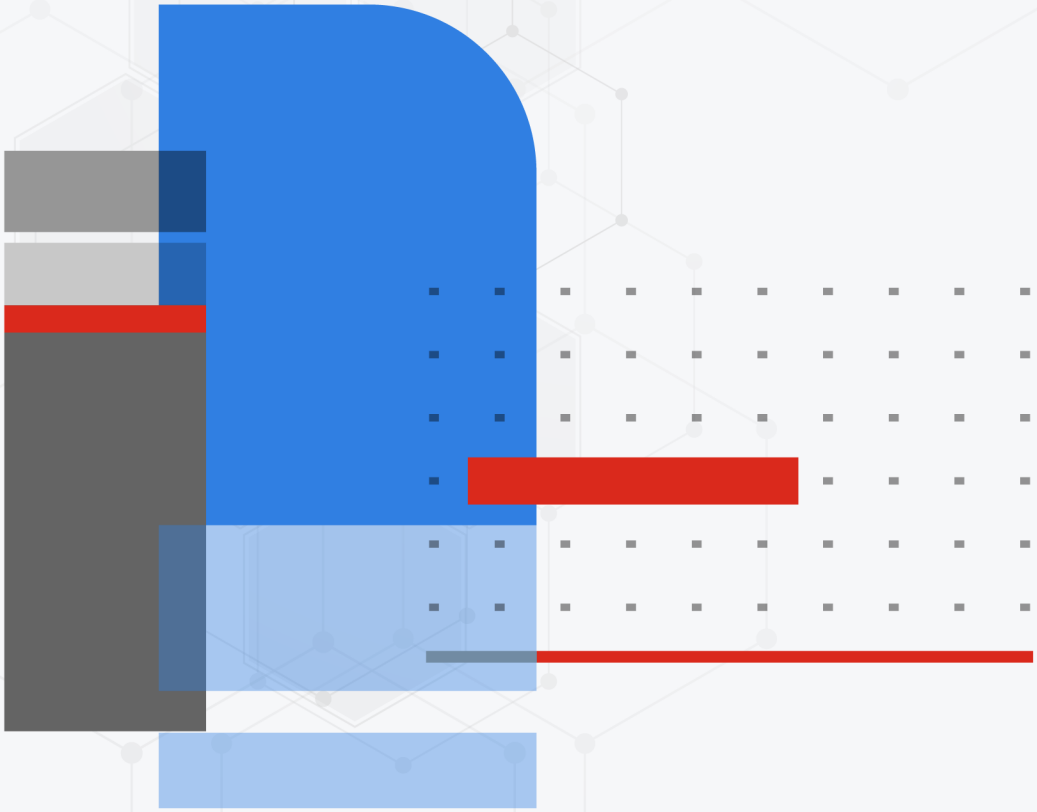
FortiADC Kubernetes Controller 3.0.0



FORTINET DOCUMENT LIBRARY
https://docs.fortinet.com

FORTINET VIDEO LIBRARY
https://video.fortinet.com

FORTINET BLOG
https://blog.fortinet.com

CUSTOMER SERVICE & SUPPORT
https://support.fortinet.com

FORTINET TRAINING & CERTIFICATION PROGRAM
https://www.fortinet.com/training-certification

FORTINET TRAINING INSTITUTE
https://training.fortinet.com

FORTIGUARD LABS
https://www.fortiguard.com

END USER LICENSE AGREEMENT
https://www.fortinet.com/doc/legal/EULA.pdf

FEEDBACK
Email: techdoc@fortinet.com

November 3, 2025
FortiADC Kubernetes Controller 3.0.0

TABLE OF CONTENTS

Change Log

Date	Change Description
November 3, 2025	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 and real server pools directly from Kubernetes by defining either standard *Ingress* resources or Fortinet's *VirtualServer* custom resources.

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 content routing.
- **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, installation using Helm, configuration parameters, and examples for integrating FortiADC with Kubernetes services through *Ingress* and *VirtualServer* definitions.

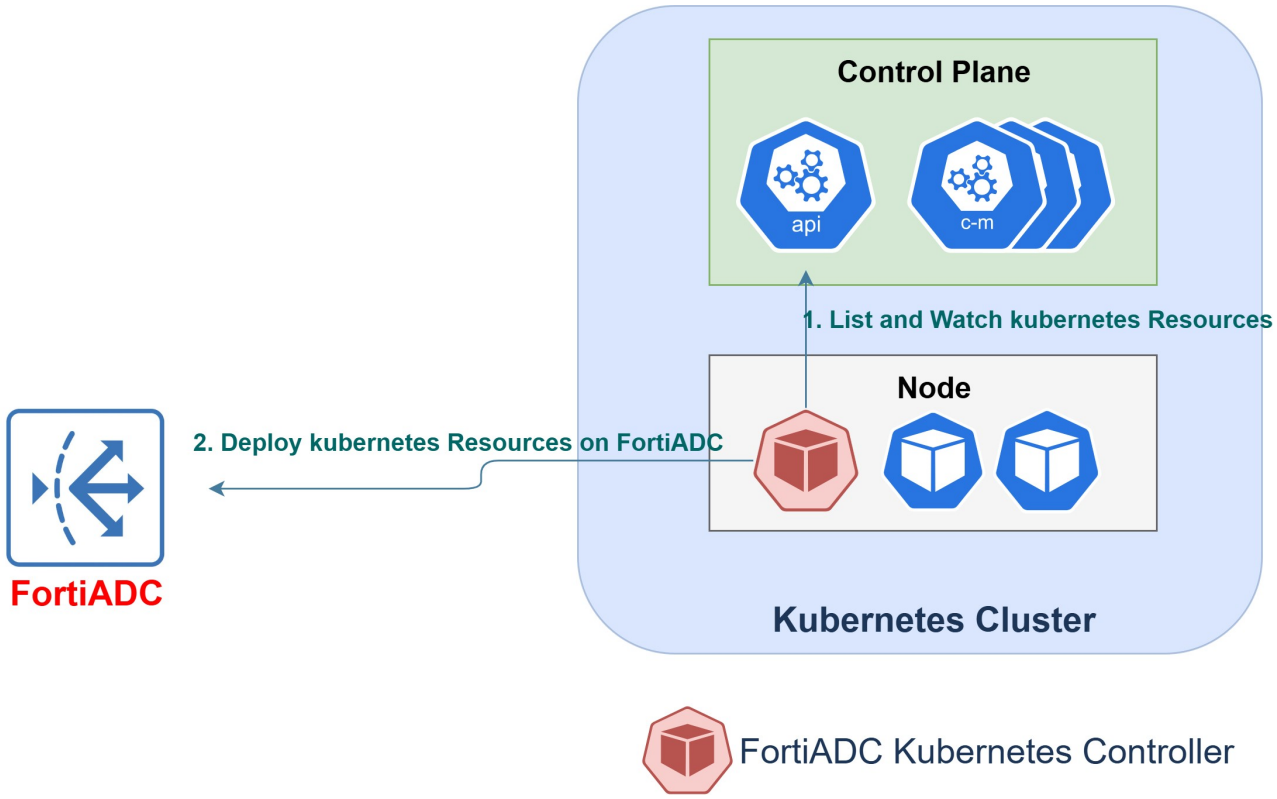
Architecture and Concepts

The FortiADC Kubernetes Controller operates as a containerized application within a Kubernetes cluster. It observes Kubernetes resources such as *Ingress*, *Service*, *EndpointSlice*, *Node*, and *VirtualServer* 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.



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, and VirtualServer.
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 resource type for defining advanced traffic-management parameters.

Resource Mapping

The following table summarizes how Kubernetes resources correspond to FortiADC objects.

Kubernetes Object	FortiADC Object
Ingress/ VirtualServer	Virtual Server / Content Routing / Scripting
Service	Real Server Pool / Real Server
Node	Real Server
Endpoint / EndpointSlice	Real Server/ Overlay Tunnel ARP/ Remote Host
Secret	Local Certificate / Certificate Group / Client-SSL

Ingress and VirtualServer Models

The FortiADC Kubernetes Controller supports two configuration models:

- **Ingress**

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

- **VirtualServer (Custom Resource)**

  Uses the Fortinet-defined *VirtualServer* CRD to configure FortiADC-specific parameters directly within Kubernetes. This model supports advanced Layer 7 capabilities such as WAF profiles, SSL settings, persistence, and GSLB integration.

Administrators can use either model, or a combination of both, depending on application complexity and operational requirements.

Before deploying the FortiADC Kubernetes Controller, review the Kubernetes and Helm concepts described in Prerequisite Knowledge on page 9.

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

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.

The controller supports both standard **Kubernetes Ingress** resources and Fortinet-defined **VirtualServer** custom resources. This dual-resource model allows administrators to either use the familiar Kubernetes Ingress framework or define FortiADC-specific configurations for advanced Layer 7 traffic management and security.

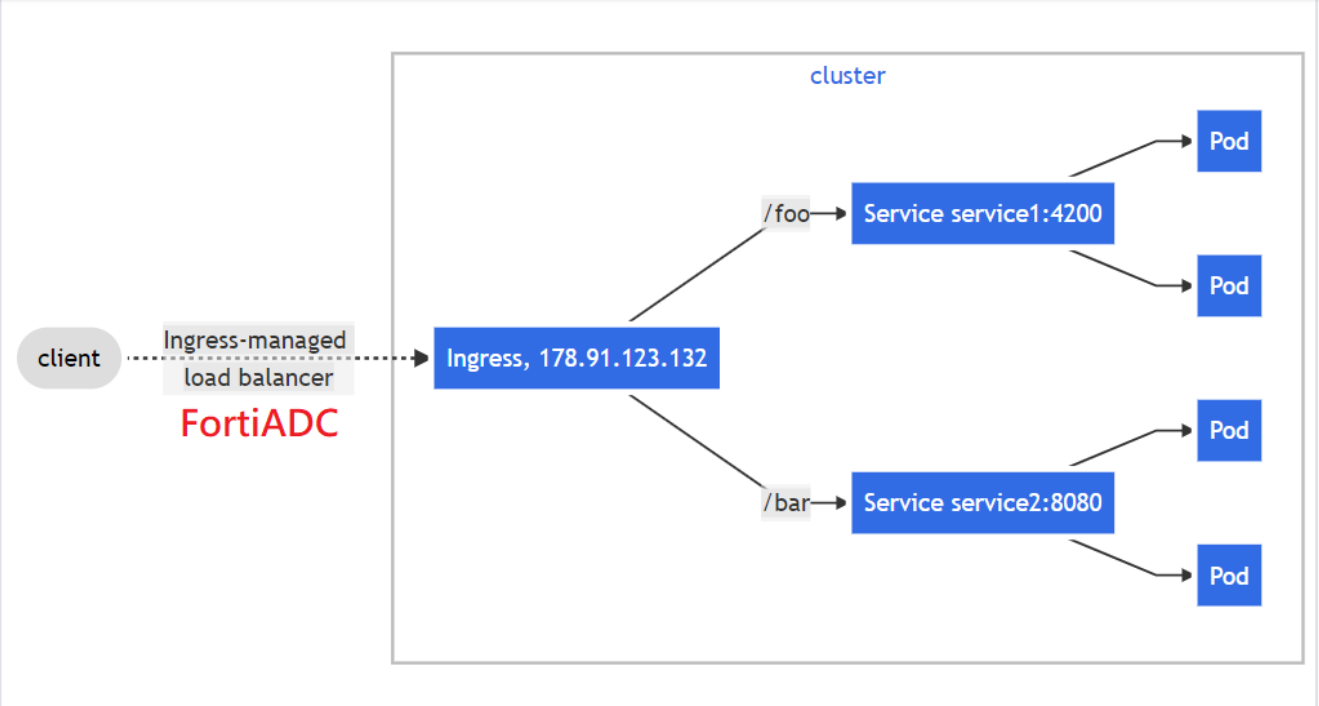
When the controller detects an event in Kubernetes—such as the creation, update, or deletion of an *Ingress* or *VirtualServer*—it translates the corresponding specifications into FortiADC objects, including **virtual servers**, **content routing rules**, and **real server pools**. This process ensures that load-balancing, SSL, and security policies remain consistent between Kubernetes and FortiADC.

FortiADC as an Ingress-Managed Load Balancer

FortiADC serves as the Ingress-managed load balancer, providing not only Layer 4–7 traffic distribution but also integrated security and visibility features, including:

- **Web Application Firewall (WAF)**
- **Antivirus scanning**
- **Denial-of-Service (DoS) protection**
- **Health checks, traffic logging, and FortiView analytics**

These capabilities extend Kubernetes' native ingress functionality by delivering enterprise-grade application delivery and protection directly within the cluster workflow.



Supported Environments

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
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
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.								

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

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
Endpoint	v1
EndpointSlice	discovery.k8s.io/v1
Event	v1
IngressClass	networking.k8s.io/v1
Ingress	networking.k8s.io/v1

API Object	API Version
ClusterRoleBinding	rbac.authorization.k8s.io/v1
ClusterRole	rbac.authorization.k8s.io/v1
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.

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.

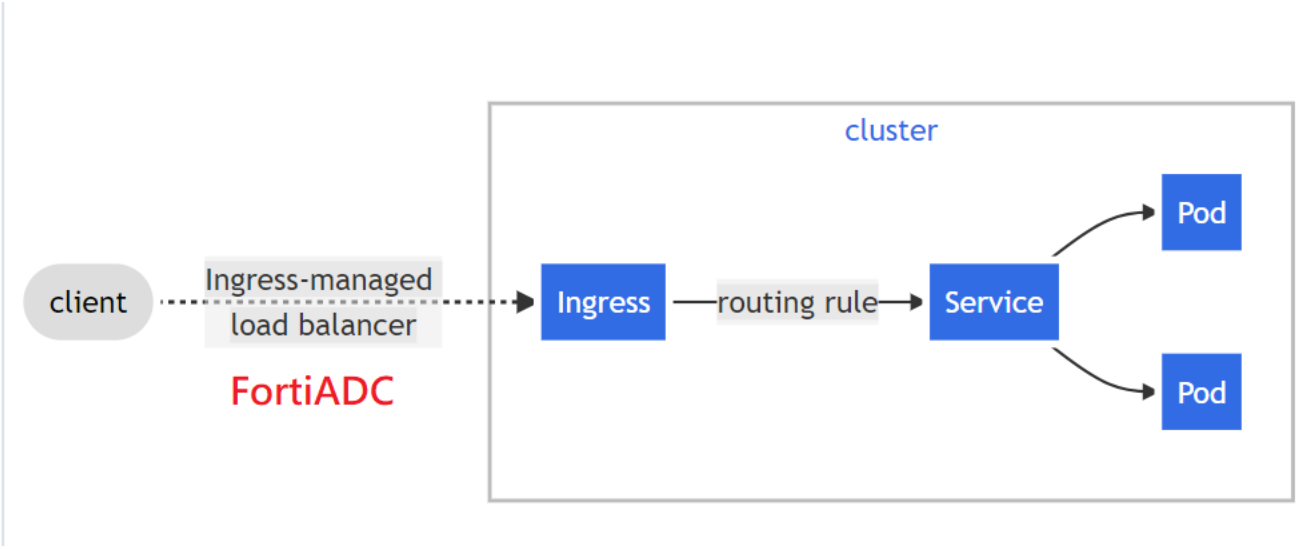
An Ingress can be configured to give Kubernetes services externally reachable URLs, load balance traffic, terminate SSL/TLS, and offer name-based virtual hosting.

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

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. To fully leverage FortiADC virtual server capabilities in Kubernetes and improve usability, a VirtualServer Custom Resource Definition (CRD) was developed. 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`.

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

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`
`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	The load balancing profile to be used (e.g., `LB_PROF_HTTPS`).	`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	

Field	Type	Description	Default
		security features.	
avProfile	string	Antivirus profile to scan traffic for threats.	
dosProfile	string	The dosProfile defines the configuration used to detect and mitigate denial-of-service (DoS) attacks.	
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
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.	
contentRoutings	list	Defines Layer 7 routing rules. Each rule can route traffic based on host and path, and includes: • name: routing rule name • host: FQDN for this route • path: URL path • pathType: match type (Prefix or Exact) • realServerPool: backend service, port, and namespace	
natSourcePoolList	list	List of SNAT (source NAT) pools.	

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

contentRoutings Field Details

This field defines Layer 7 (HTTP/S) routing rules, allowing traffic to be forwarded based on hostname and URL path to specific Kubernetes services.

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

Field	Type	Description
`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:

- defaultbackend-virtualserver
- simple-fanout-virtualserver
- wildcard-host-virtualserver
- tls-san-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_defaultbackend.yaml

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha1
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_simple_fanout.yaml

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha1
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_wildcard_host.yaml

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha1
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_tls_san.yaml

```
apiVersion: fadk8sctrl.fortinet.com/v1alpha1
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
```

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 Scripting
Service	Real Server Pool Real Server
Node	Real Server
Endpoint/ EndpointSlice	Real Server Overlay Tunnel ARP 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:

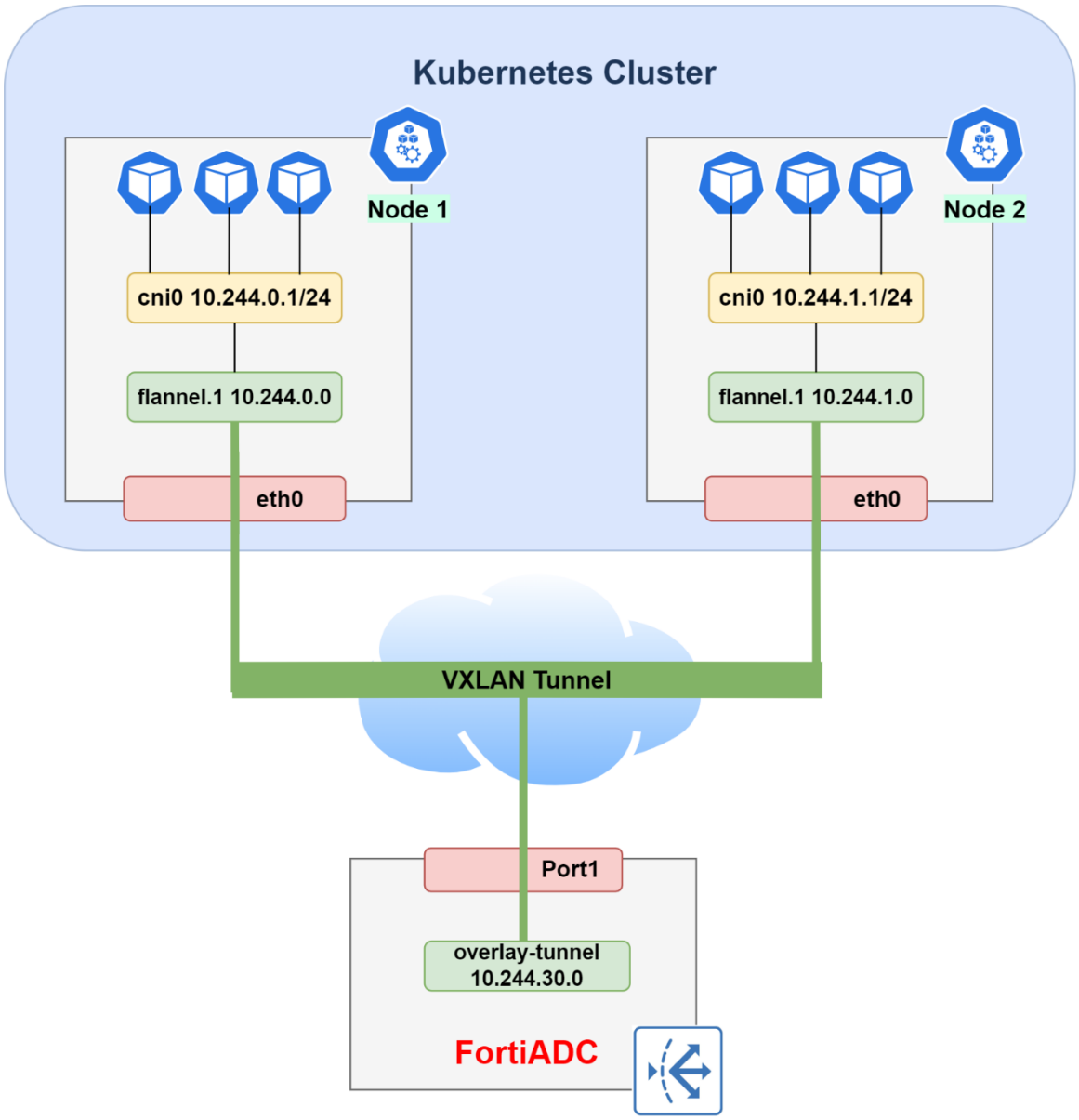
FortiADC Objects	Naming Rule
Virtual server Real Server Pool Scripting Local Certificate Local Certificate Group Client-SSL	[namespace of Kubernetes objects]_[name of Kubernetes objects]
Content Routing	[namespace of Kubernetes objects]_[name of Kubernetes ingress]_[name of

FortiADC Objects	Naming Rule
	Kubernetes service]
Real Server	Name of the Kubernetes node

Kubernetes CNI Plugin

The Kubernetes cluster network model is implemented by the container runtime on each node. The most common container runtimes use Container Network Interface (CNI) plugins to manage their network and security capabilities. Many different CNI plugins exist from many different vendors. FortiADC version 7.4.0 supports the most widely used CNI plugin, flannel with VXLAN backend. By setting up the VXLAN tunnel, as shown in the figure below, FortiADC can communicate with Kubernetes EndpointSlices, which is a collection of endpoints that implement the actual service (usually these endpoints are Pods).

FortiADC Kubernetes Controller 2.0 or later supports ingress to expose services with the ClusterIP type. Since the ClusterIP type Service can only be accessed within the cluster, an overlay-tunnel is required to connect the FortiADC to the Kubernetes cluster network.



FortiADC Ingress Controller version 1.0 only supports services of type **NodePort**.
FortiADC Ingress Controller version 2.0 or later supports services of type **NodePort** and **ClusterIP**.

Installation

Install FortiADC Kubernetes Controller using Helm Charts.

Currently, only Helm 3 (version 3.6.3 or later) is supported.

Helm Charts ease the installation of FortiADC Kubernetes Controller in the Kubernetes cluster. By using the Helm 3 installation tool, most of the Kubernetes objects required for FortiADC Kubernetes Controller can be deployed in one simple command.

The Kubernetes objects required for FortiADC Kubernetes Controller are listed below:

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 objects.
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. 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.)

The Helm Chart is composed of a collection of files that describe the related set of Kubernetes files required by FortiADC Kubernetes Controller; one of which is the `values.yaml` file that provides the default configuration for deploying the Kubernetes objects listed above.

Below lists parts of the values in the `values.yaml` file.

```
# 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: "3.0.0"

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"
```

In some scenarios, you may want to override some of the values included in the `values.yaml`, such as for the toleration seconds or parameter properties. As the `values.yaml` file is packed in the Helm Chart package, you can override the values when installing or upgrading the Helm Chart (see Install the Helm Chart on page 38 and Upgrade the Helm Chart on page 39). For more details on the parameters, see Configuration Parameters on page 42.

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

Get Repo Information

To get the repository information:

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

Install the Helm Chart

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
```

In the example below, the Helm chart is installed with the release name "first-release" in the Kubernetes namespace "fortiadc-ingress":

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

If you want to override values in the Helm Chart, you can use `--set` flags in the command. In the example below, you can 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
```

Upgrade the Helm Chart

Use the following commands:

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

Starting with version 3.0.0, the Helm chart repository was renamed to fortiadc-kubernetes-controller. When upgrading from version 2.x to 3.0.0 or later, ensure you remove the old Helm repository and add the new repository before continuing with the upgrade.

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
```

You can also change the field of `values.yaml` with the `--set` command.

To see which values you can change, please refer to https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/charts/fadc-k8s-ctrl-3.0.0/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, you can check the Helm debug information “USER-SUPPLIED VALUES” to check if you have all the value set as you need.

```
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
```

Check the Installation

Check to see if the FortiADC Kubernetes Controller is installed correctly:

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

The helm history command shows the installation information:

```
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
```

You can also use the kubectl command to check the installation:

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

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

You will get the FortiADC Kubernetes Controller deployment and pod status like the following:

```
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
```

Check the FortiADC Kubernetes Controller log:

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

You can get the FortiADC Kubernetes Controller logs like the following:

```
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"
```

Uninstall the Helm 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]
```

Configuration Parameters

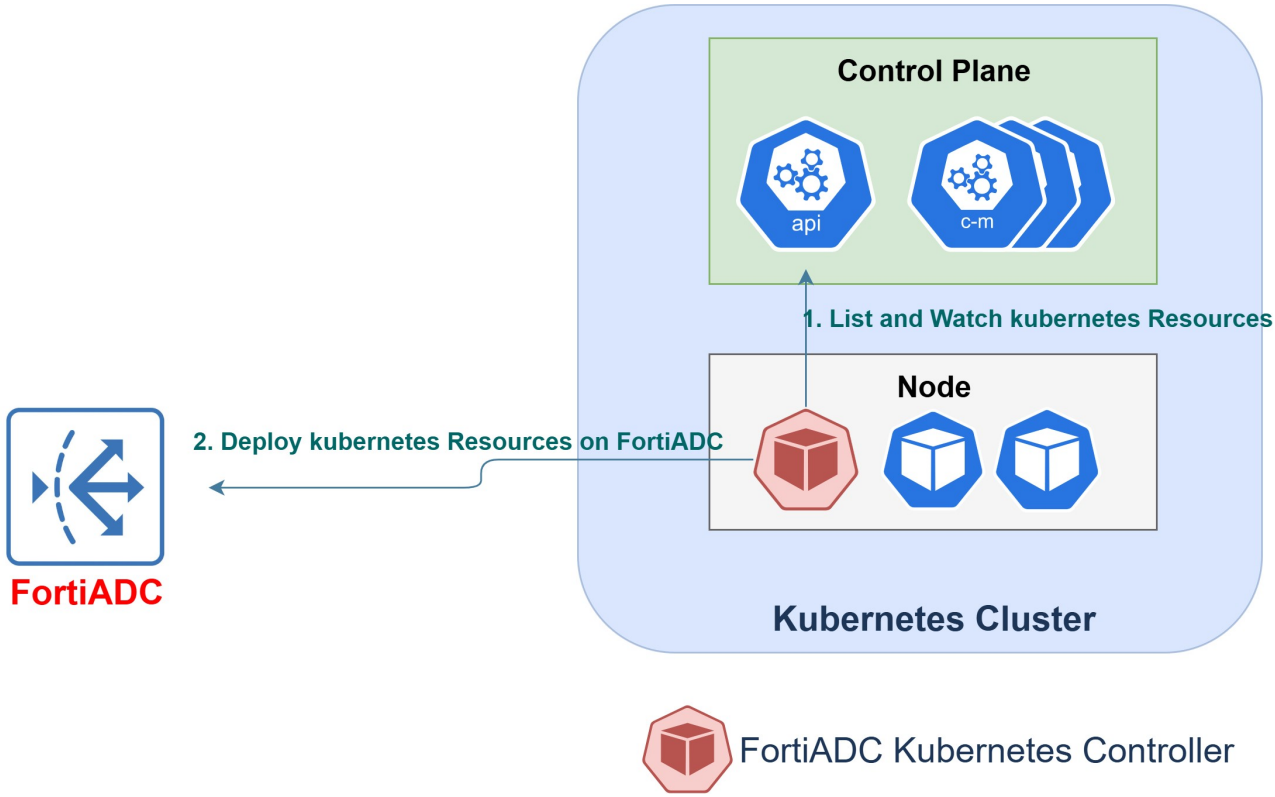


Figure 1 FortiADC Kubernetes Controller

FortiADC Authentication Secret

As shown in *Figure 1*, the FortiADC Kubernetes Controller satisfies an *Ingress* or *custom resource* through REST API calls to FortiADC. To enable this communication, the controller must have valid authentication credentials to access the FortiADC instance.

To store these credentials securely within the Kubernetes cluster, create a Kubernetes Secret that contains the FortiADC username and password.

For example:

```
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**. must be specified in the *Ingress* or *custom resource* annotation `fortiadc-login`, which allows the FortiADC Kubernetes Controller to authenticate to FortiADC using the provided credentials.

The namespace of the authentication secret must be the same as the *Ingress* or *custom resource* that references this authentication secret.

Annotation in Ingress

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. **Note**: This parameter is **required**.	
fortiadc-admin-port	FortiADC HTTPS service port.	443
fortiadc-login	The Kubernetes secret name preserves the FortiADC authentication information. **Note**: 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 theFortiADC 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. **Note**: This parameter is **required**.	
virtual-server-interface	The FortiADC network interface for the client to access the virtual server. **Note**: 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.	LB_METHOD_ROUND_ROBIN

Parameter	Description	Default
	For more details, see the FortiADC Administration Guide on load balancing methods.	
load-balance-profile	Default is LB_PROF_HTTP. If TLS is specified in the Ingress, then the default is LB_PROF_HTTPS.	LB_PROF_HTTP 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

Parameter	Description	Default
virtual-server-fortigslb-publicip	Specify the virtual server public IP address.	
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.

Annotation in Service

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.	NONE

Parameter	Description	Default
	For more details, see the FortiADC Administration Guide on SSL profiles.	
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
```

Annotation in VirtualServer

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

Parameter	Description	Default
fortiadc-ip	The VirtualServer will be deployed on FortiADC with the given IP address or domain name. This parameter is required.	

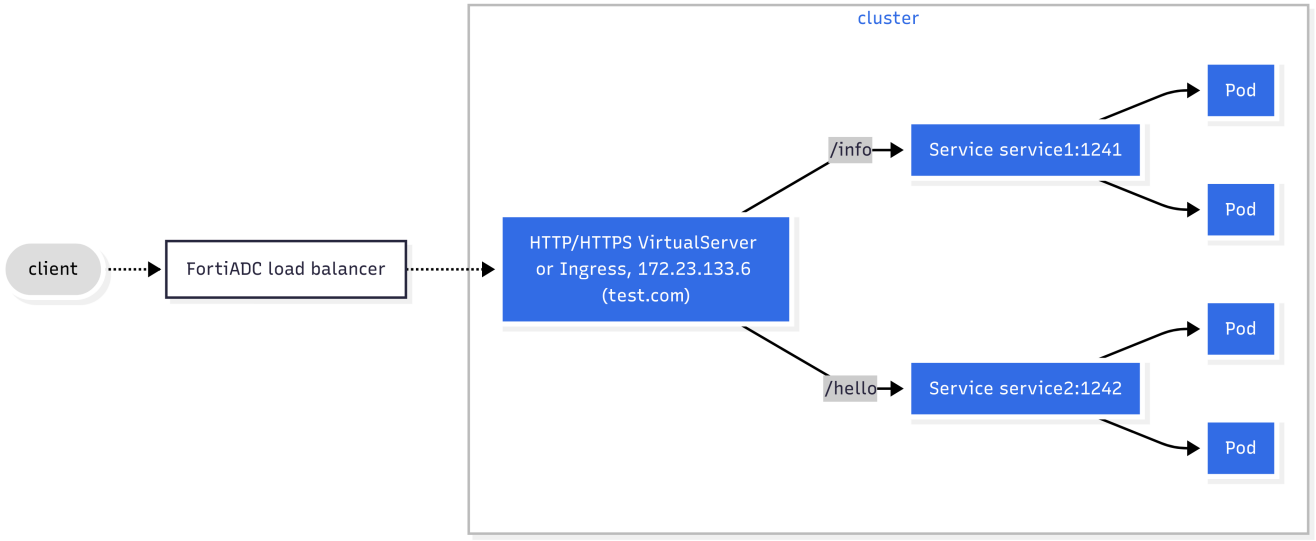
Parameter	Description	Default
fortiadc-admin-port	FortiADC HTTPS service port.	443
fortiadc-login	The Kubernetes secret name preserves the FortiADC authentication information. This parameter is required.	
fortiadc-ctrl-log	Enable/disable theFortiADC Kubernetes Controller log. Once enabled, FortiADC Kubernetes Controller will print the verbose log the next time the VirtualServer is updated.	enable

Deployment

The FortiADC Kubernetes Controller supports two deployment models for exposing applications in a Kubernetes cluster:

- **Ingress-based deployment** – Uses the standard Kubernetes Ingress resource to define external access rules for HTTP/HTTPS traffic.
- **Custom Resource-based deployment** – Uses the Fortinet-defined VirtualServer Custom Resource Definition (CRD) to configure advanced FortiADC-specific parameters directly within Kubernetes.

The following example demonstrates a **simple-fanout** deployment using both resource types.



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 Exposing Kubernetes ClusterIP type services on page 52.

For a detailed walkthrough of deploying the same application using the VirtualServer CRD–including validation and monitoring steps–see Deploying the VirtualServer with Custom Resource Definition (CRD) on page 57.

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:

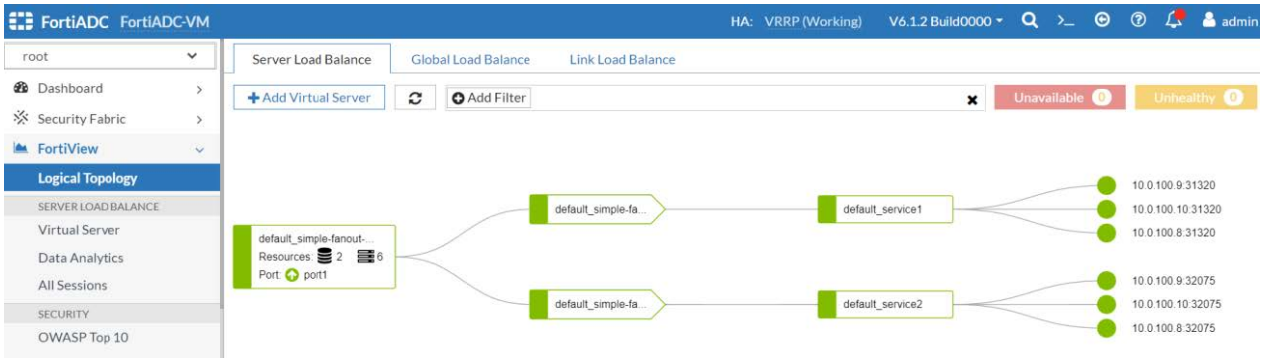
```
kubectl apply -f simple-fanout.yaml
ingress.networking.k8s.io/simple-fanout-example created
```

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

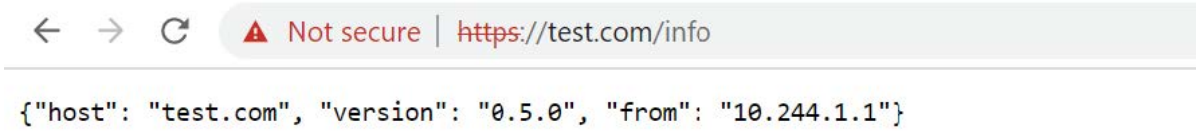
```
user@control-plane-node ~> kubectl describe ingress simple-fanout-example

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.

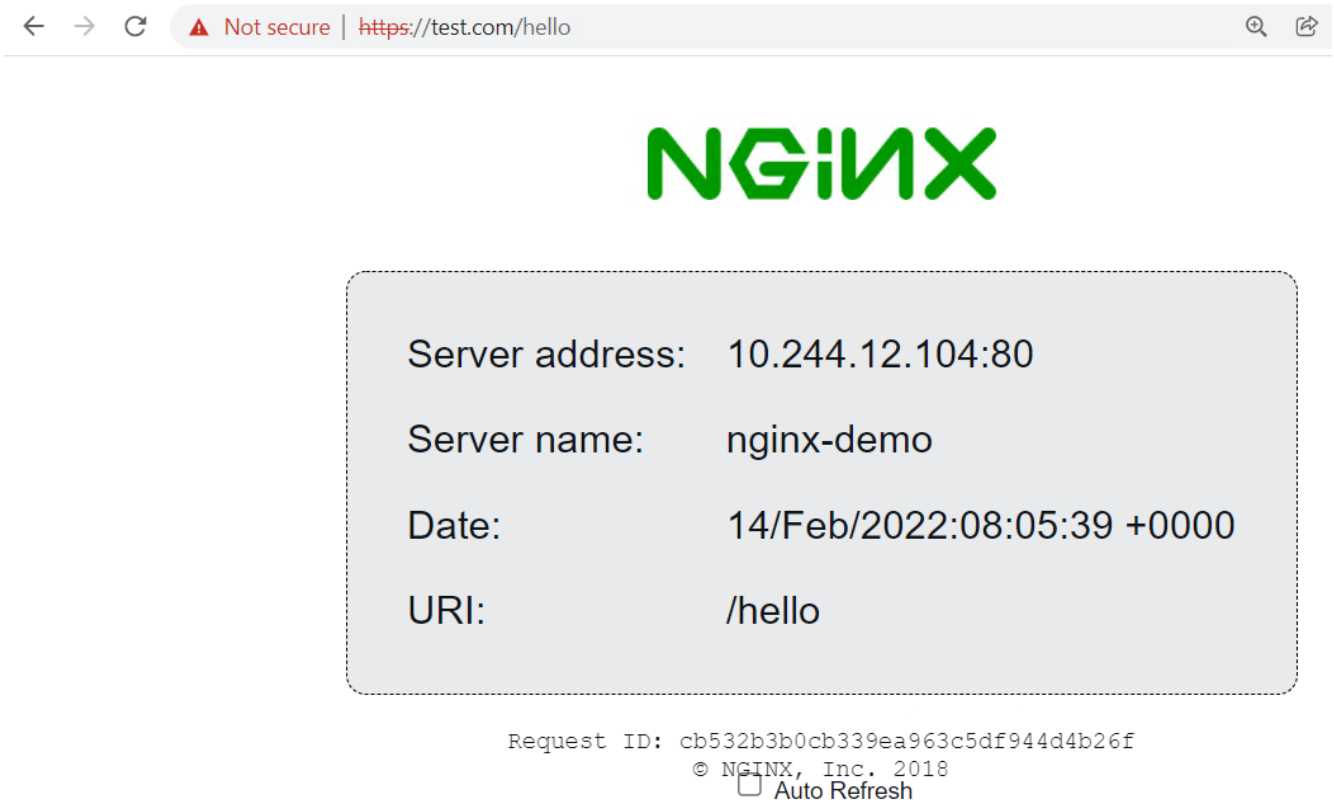


Try to access `https://test.com/info`.




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

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



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
```

Add, update or delete Service and Node

Service

FortiADC Kubernetes Controller only monitors the **service type** and **annotations** defined in services used in the deployed Ingress resource.

FortiADC Kubernetes Controller does not support services with multiple ports exposed.

If you delete a service that is used in the deployed Ingress resource, Kubernetes would not give you any warning, and FortiADC Kubernetes Controller would not handle any delete events on the service.

Node

If you add or delete a worker node, FortiADC Kubernetes Controller will check the deployed Ingress resources and handle the add/delete event. For updating a node, FortiADC Kubernetes Controller only monitors the node's IP and node condition type NotReady status.

Exposing Kubernetes ClusterIP type services

FortiADC Kubernetes Controller 2.0 supports ingress to expose services with the ClusterIP type. Since the ClusterIP type Service can only be accessed within the cluster, an overlay-tunnel is required to connect the FortiADC to the Kubernetes cluster network.

Follow the basic steps below to expose ClusterIP type services:

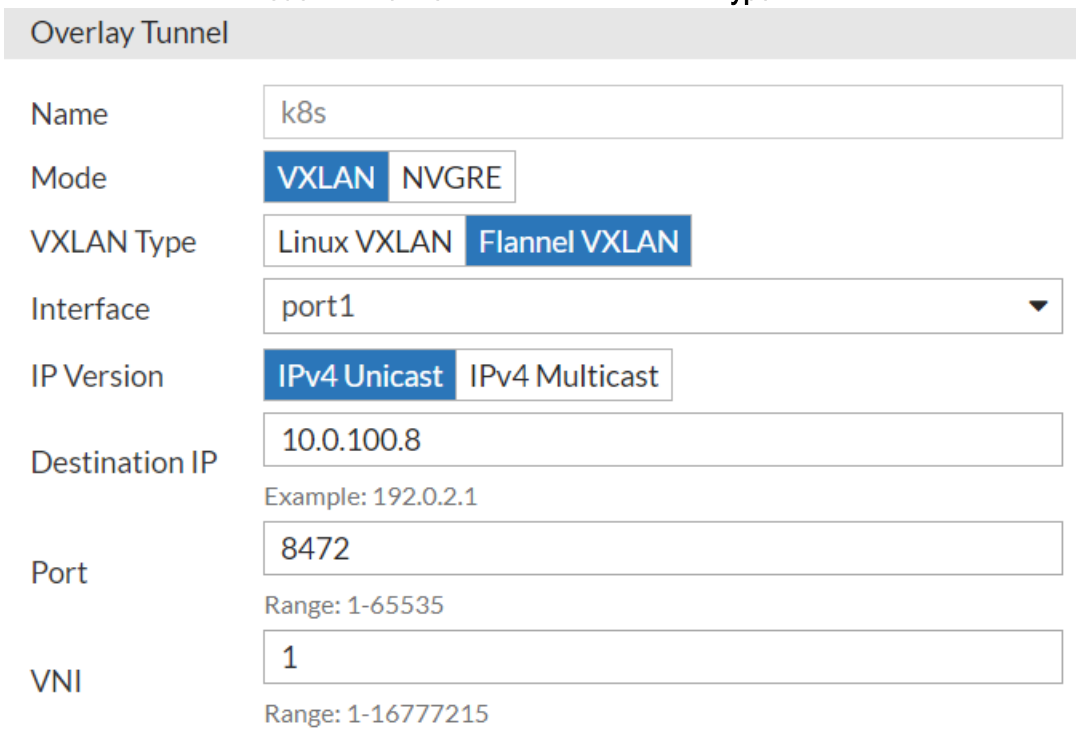
1. Deploy FortiADC as a fake node in the Kubernetes cluster
2. Deploy the Pods and expose the ClusterIP type Service

Deploy FortiADC as a fake node in the Kubernetes cluster

1. 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.

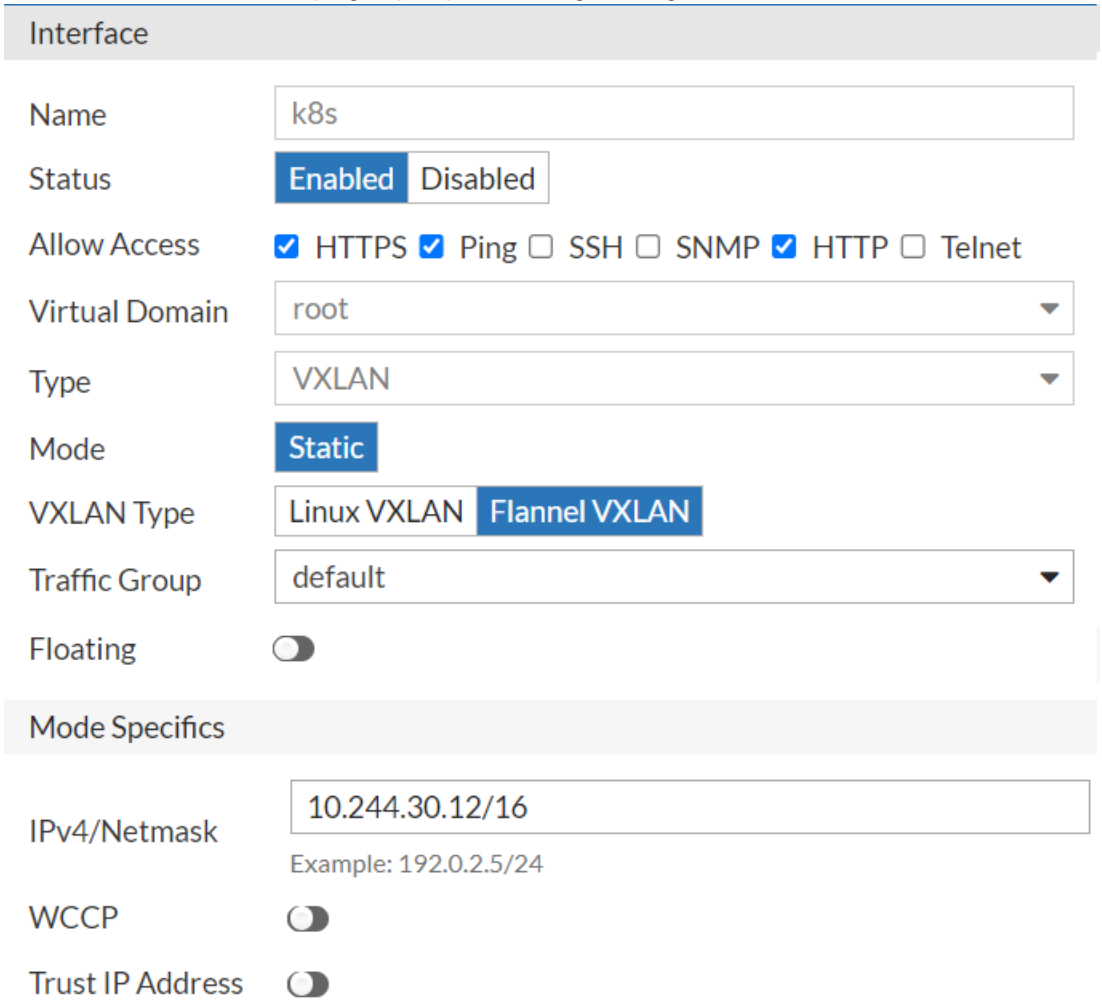
```
kubectl describe node [NODE NAME] | grep flannel
```

2. To integrate the FortiADC overlay tunnel with the flannel VXLAN, an overlay tunnel with the VXLAN type must be created 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.
   Select **VXLAN** as the **Mode** and **Flannel VXLAN** as the **VXLAN Type**.



3. 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.



4. 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 FortiADC as the fake node installed with the flannel CNI plugin in your 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.

Please follow this link to copy and modify the FAD Node YAML example:

https://github.com/fortinet/fortiadc-kubernetes-controller/blob/main/node_examples/fad.yaml

```
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"
```

Note: 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.

Deploy the Pods and expose the ClusterIP type Service

1. Use the Kubernetes deployment to deploy the NGINX server in the pods. The NGINX server serves the simple web HTTP and HTTPS application.
   Note: You have to generate the corresponding Kubernetes Secret and Configmap for HTTPS applications.
2. Expose the web application running on these pods to a service with default type ClusterIP. You **must** specify the **overlay tunnel** in the service annotations.

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
```

Installing Kubernetes Custom Resource

As part of deploying the **FortiADC Kubernetes Controller** via Helm, the **VirtualServer Custom Resource Definition (CRD)** is automatically installed.

This CRD extends the Kubernetes API to support advanced FortiADC virtual server configurations beyond the capabilities of standard Ingress resources.

You can use the following command to check whether VirtualServer Custom Resource Definition is applied.

```
kubectl get crds
NAME                                    CREATED AT
virtualservers.fadk8sctrl.fortinet.com   2025-08-01T03:23:33Z
```

Deploying the VirtualServer with Custom Resource Definition (CRD)

After confirming the CRD installation, you can deploy a VirtualServer resource to test controller functionality.

The following example demonstrates the simple-fanout scenario, which defines multiple routing paths under a single hostname.

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/v1alpha1
kind: VirtualServer
metadata:
  name: simple-fanout-virtualserver
  annotations: {
    "fortiadc-ip" : "172.31.5.197",
    "fortiadc-login" : "fad-login",
    "fortiadc-ctrl-log" : "enable",
    "fortiadc-admin-port": "443"
   }
  labels:
    fadcr: "true"
spec:
  addressType: ipv4
```

```
  address: 172.31.12.174
  port: 8443
  interface: port2
  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
```

Deploy it with the `kubectl` command:

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

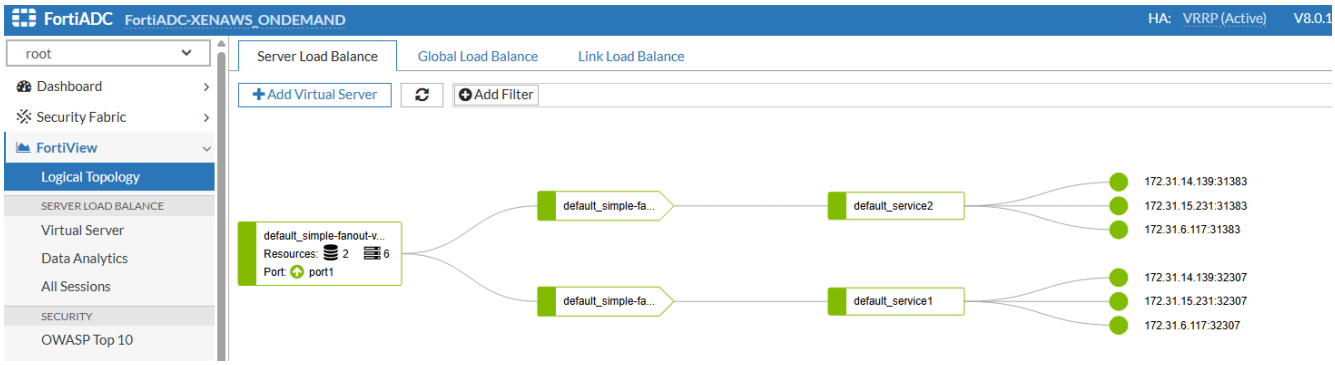
Get the information of the **simple-fanout-virtualserver** by using the `kubectl describe` command:

```
# 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: 172.31.5.197
```

```
              fortiadc-login: fad-login
API Version:  fadk8sctrl.fortinet.com/v1alpha1
Kind:         VirtualServer
Metadata:
  Creation Timestamp:  2025-09-11T19:06:24Z
  Generation:          1
  Resource Version:    30096049
  UID:                 687306af-c22e-4c9a-badf-06590f2927d5
Spec:
  Address:          172.31.12.174
  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>
```

FortiView

Check the deployed VirtualServer with FortiView.



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

```
{"host": "test.com", "version": "0.5.0", "from": "172.31.15.231"}
```

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

Update or delete the VirtualServer

To update a virtualserver resource:

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

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

To delete the virtualserver resource:

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

Debug

By default, the FortiADC Kubernetes Controller records the process of the Ingress 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. This is determined by the status of the fortiadc-ctrl-log configuration parameter. For more information, see Configuration Parameters on page 42.

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` command to reconfigure the Ingress.

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/.

www.fortinet.com