Oracle Cloud Native Environment

Calico Module for Release 1.8

F87571-02
March 2026

Oracle Cloud Native Environment Calico Module for Release 1.8,

F87571-02

Contents

Preface

> **Warning**
>
> Oracle Cloud Native Environment 1.8 is now in Sustaining Support. See Oracle Open Source Support Policies for more information. New security patches and bug fixes are no longer provided for this software.
>
> Migrate applications and data to Oracle Cloud Native Environment Release 2.<latest> as soon as possible.

This document contains information about setting up Calico as the Kubernetes CNI in Oracle Cloud Native Environment. It describes the Calico module provided with Oracle Cloud Native Environment to set up Calico, and the native Kubernetes CNI option when you install the Kubernetes module. It provides examples on how to test that Calico is installed and working.

Documentation License

Conventions

The following text conventions are used in this document:

Convention	Meaning
boldface	Boldface type indicates graphical user interface elements associated with an action, or terms defined in text or the glossary.
italic	Italic type indicates book titles, emphasis, or placeholder variables for which you supply particular values.
`monospace`	Monospace type indicates commands within a paragraph, URLs, code in examples, text that appears on the screen, or text that you enter.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support for Accessibility

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1

Introduction to Calico

> **Warning**
>
> Oracle Cloud Native Environment 1.8 is now in Sustaining Support. See Oracle Open Source Support Policies for more information. New security patches and bug fixes are no longer provided for this software.
>
> Migrate applications and data to Oracle Cloud Native Environment Release 2.<latest> as soon as possible.

Calico is an L3/L4 networking solution to secure containers, Kubernetes clusters, and virtual machines. It offers scalable pod networking using overlay or non-overlay modes, optionally using Border Gateway Protocol (BGP), and includes advanced IP address management.

The Calico Kubernetes Container Network Interface (CNI) plugin enforces and extends the Kubernetes Network Policy API to set ingress and egress policies. Rules can be set using Boolean-like logic to create network policies using any combination of:

- Namespaces.
- Label selectors.
- Network protocols (TCP, UDP, SCTP).
- Network ports.
- Network CIDRs.

For information on the Kubernetes Network Policy API, see the upstream Kubernetes documentation.

The Tigera Calico operator is deployed into Oracle Cloud Native Environment. You can install the operator as the native Kubernetes CNI when you install the Kubernetes module. Or you install the operator using the Calico module. Both installation options are shown in this document.

The Tigera Calico operator is installed with a default VXLAN configuration (default-allow) to enable network traffic between pods. This means that if you don't create any network policies, all pods can communicate with each other.

More information on Calico is available in the upstream Calico documentation.

2 Installing Calico

> **Warning**
>
> Oracle Cloud Native Environment 1.8 is now in Sustaining Support. See Oracle Open Source Support Policies for more information. New security patches and bug fixes are no longer provided for this software.
>
> Migrate applications and data to Oracle Cloud Native Environment Release 2.<latest> as soon as possible.

This chapter discusses how to install the Calico module in Oracle Cloud Native Environment. This chapter also shows you how to install the Kubernetes Calico CNI when you create the Kubernetes module.

Prerequisites

This section contains the prerequisite information you need to set up the Tigera Calico operator.

Disabling firewalld Service

Disable the `firewalld` service on each Kubernetes node:

```
sudo systemctl stop firewalld.service
sudo systemctl disable firewalld.service
```

> **Important**
>
> As disabling the `firewalld` service removes the network protection provided by this service, you must implement Calico network policies to secure the Kubernetes cluster. For information on how to secure the cluster using Calico, see the upstream Calico documentation.

Updating Proxy Configuration

If you're using a proxy server in the environment, edit the CRI-O proxy configuration file and add the Kubernetes service IP (the default is `10.96.0.1`) to the `NO_PROXY` variable. For example, on each Kubernetes node, edit the `/etc/systemd/system/crio.service.d/proxy.conf` file:

```
[Service]
Environment="HTTP_PROXY=http://proxy.example.com:3128"
Environment="HTTPS_PROXY=https://proxy.example.com:3128"
Environment="NO_PROXY=mydomain.example.com,10.96.0.1"
```

Reload the configuration file and restart the `crio` service:

```
sudo systemctl daemon-reload
sudo systemctl restart crio.service
```

> **Note**
>
> You don't need to perform this step if you're using the `olcnectl provision` command to perform a quick installation. This is set up for you automatically when using that installation method and you provide any proxy information.

Kubernetes Module

To install the Calico module, the Kubernetes module must be created and installed with no CNI set. When you create the Kubernetes module, set the `--pod-network none` option as part of the `olcnectl module create` command. For example:

```
olcnectl module create \
--environment-name myenvironment \
--module kubernetes \
--name mycluster \
--pod-network none \
...
```

> **Important**
>
> To deploy the Tigera Calico operator as the native Kubernetes CNI, this isn't required. You instead set this option to `--pod-network calico`.

Creating a Calico Configuration File

You can optionally install the Calico module with a configuration file. A Calico configuration file is used to configure any specific modifications of the Tigera Calico operator. This YAML file contains the `spec` part of an `operator.tigera.io/v1/Installation`. This file must be available on the operator node.

> **Note**
>
> You can't use a Calico configuration file if you deploy the Tigera Calico operator as a native Kubernetes CNI when you create the Kubernetes module with the `--pod-network calico` option.

For information on the Calico configuration file, see the upstream Calico documentation.

An example Calico configuration file is:

```
installation:
  cni:
```

```
    type: Calico
  calicoNetwork:
    bgp: Disabled
    ipPools:
    - cidr: 198.51.100.0/24
      encapsulation: VXLAN
  registry: container-registry.oracle.com
  imagePath: olcne
```

Deploying the Calico CNI

The easiest way to install Calico is to set the Kubernetes CNI to Calico when you create the Kubernetes module. This installs the Tigera Calico operator into the Kubernetes cluster with the default configuration. You don't need to install the Calico module with this method.

Before you set Calico as the Kubernetes CNI, perform the prerequisites to disable the `firewalld` service and update any proxy configuration for CRI-O, as discussed in Prerequisites.

To set Calico as the Kubernetes CNI, create a Kubernetes module using the `--pod-network calico` option of the `olcnectl module create --module kubernetes` command. This option sets Calico as the Kubernetes CNI for pods in the Kubernetes cluster. The name of the Kubernetes module in this example is `mycluster`.

```
olcnectl module create \
--environment-name myenvironment \
--module kubernetes \
--name mycluster \
--pod-network calico \
...
```

For more information on creating a Kubernetes module, see Kubernetes Module.

Deploying the Calico Module

The Calico module lets you use a configuration file to configure the Tigera Calico operator. If you don't use a configuration file, the installation is the same as when you use the native Calico CNI installation option when you create the Kubernetes module. The benefit of using the Calico module is that you can use a configuration file. If you don't want to change the operator configuration, you might want to install Calico using the Calico CNI method as less steps are required.

To use this method, you must create the Kubernetes module using the `--pod-network none` option. This option sets no Kubernetes CNI for pods in the cluster. You then install the Calico module to set the CNI.

For the syntax to use to create a Calico module, see the `calico` option of the `olcnectl module create` command in Platform Command-Line Interface.

To deploy the Calico module:

1. Create and install a Kubernetes module using the `--pod-network none` option of the `olcnectl module create --module kubernetes` command. This option sets no

Kubernetes CNI for pods in the cluster. The name of the Kubernetes module in this example is `mycluster`. For example:

```
olcnectl module create \
--environment-name myenvironment \
--module kubernetes \
--name mycluster \
--pod-network none \
...
```

> **Important**
>
> When you install the Kubernetes module with the `--pod-network none` option, all `kube-system` pods are marked as `pending` until you install the Calico module. When the Calico module is installed, these `kube-system` pods are marked as `running`.

2. Create a Calico module and associate it with the Kubernetes module named `mycluster` using the `--calico-kubernetes-module` option. In this example, the Calico module is named `mycalico`.

   ```
   olcnectl module create \
   --environment-name myenvironment \
   --module calico \
   --name mycalico \
   --calico-kubernetes-module mycluster
   ```

   The `--module` option sets the module type to create, which is `calico`. You define the name of the Calico module using the `--name` option, which in this case is `mycalico`.

   The `--calico-kubernetes-module` option sets the name of the Kubernetes module.

   An optional `--calico-installation-config` sets the location for a Calico configuration file. This file must be available on the operator node under the provided path. For information on creating this configuration file, see Prerequisites.

   If you don't include all the required options when adding the module, you're prompted to provide them.

3. Use the `olcnectl module install` command to install the Calico module. For example:

   ```
   olcnectl module install \
   --environment-name myenvironment \
   --name mycalico
   ```

   You can optionally use the `--log-level` option to set the level of logging displayed in the command output. By default, error messages are displayed. For example, you can set the logging level to show all messages when you include:

   ```
   --log-level debug
   ```

   The log messages are also saved as an operation log. You can view operation logs as commands are running, or when they've completed. For more information using operation logs, see Platform Command-Line Interface.

The Calico module is deployed into the Kubernetes cluster.

Verifying the Calico Deployment

This section contains information on how to verify the installation of Calico, with both the Kubernetes CNI installation option, or using the Calico module.

Verifying the Calico Module

If you installed Calico using the Calico module, you can verify the module is deployed using the `olcnectl module instances` command on the operator node. For example:

```
olcnectl module instances \
--environment-name myenvironment
```

The output looks similar to:

```
INSTANCE                  MODULE        STATE
mycalico                  calico        installed
mycluster                 kubernetes    installed
control1.example.com      node          installed
...
```

Note the entry for `calico` in the `MODULE` column is in the `installed` state.

In addition, use the `olcnectl module report` command to review information about the module. For example, use the following command to review the Calico module named `mycalico` in `myenvironment`:

```
olcnectl module report \
--environment-name myenvironment \
--name mycalico \
--children
```

For more information on the syntax for the `olcnectl module report` command, see Platform Command-Line Interface.

Verifying the Tigera Calico Operator

The Tigera Calico operator is deployed when you use the Kubernetes Calico CNI installation option and with the Calico module installation method. This section shows you some areas you can check to verify the Calico installation and learn about the configuration.

Tigera Calico Operator Status

You can get information about the Tigera Calico operator status using the `kubectl get tigerastatus` command:

```
kubectl get tigerastatus
```

The output shows the status of the operator components, and looks similar to:

```
NAME        AVAILABLE   PROGRESSING   DEGRADED   SINCE
apiserver   True        False         False      8m24s
calico      True        False         False      8m39s
```

IP Pools

You can get information about the default IP Pools that are set up using the `kubectl get ippools` command:

```
kubectl get ippools
```

The output looks similar to:

```
NAME                  CREATED AT
default-ipv4-ippool   ...
```

To get more information about the IP Pool, use:

```
kubectl describe ippools default-ipv4-ippool
```

The output looks similar to:

```
Name:         default-ipv4-ippool
Namespace:
Labels:       <none>
Annotations:  <none>
API Version:  projectcalico.org/v3
Kind:         IPPool
Metadata:
  Creation Timestamp:  ...
  Resource Version:    1112
  UID:                 fd04d1d2-b5c9-4feb-9385-2b423c4dd67f
Spec:
  Allowed Uses:
    Workload
    Tunnel
  Block Size:     26
  Cidr:           10.244.0.0/16
  Ipip Mode:      Never
  Nat Outgoing:   true
  Node Selector:  all()
  Vxlan Mode:     Always
Events:           <none>
```

Network Policies

To get information on the network policies that are set up, use:

```
kubectl get networkpolicies --all-namespaces
```

The output looks similar to:

```
NAMESPACE          NAME              POD-SELECTOR     AGE
calico-apiserver   allow-apiserver   apiserver=true   66m
```

To get more information about the network policies, use:

```
kubectl describe networkpolicies --all-namespaces
```

The output looks similar to:

```
Name:         allow-apiserver
Namespace:    calico-apiserver
Created on:   <date> 05:27:30 +0000 GMT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     apiserver=true
  Allowing ingress traffic:
    To Port: 5443/TCP
    From: <any> (traffic not restricted by source)
  Not affecting egress traffic
  Policy Types: Ingress
```

Installation Configuration

You can see the installation configuration for the Tigera Calico operator using:

```
kubectl get installation -o yaml
```

The output looks similar to:

```
apiVersion: v1
items:
- apiVersion: operator.tigera.io/v1
  kind: Installation
...
  spec:
    calicoNetwork:
      bgp: Disabled
      hostPorts: Enabled
      ipPools:
      - blockSize: 26
        cidr: 10.244.0.0/16
        disableBGPExport: false
        encapsulation: VXLAN
        natOutgoing: Enabled
        nodeSelector: all()
      linuxDataplane: Iptables
      multiInterfaceMode: None
      nodeAddressAutodetectionV4:
        firstFound: true
      cni:
        ipam:
```

```
      type: Calico
    type: Calico
  controlPlaneReplicas: 2
  flexVolumePath: /usr/libexec/kubernetes/kubelet-plugins/volume/exec/
  imagePath: olcne
  kubeletVolumePluginPath: /var/lib/kubelet
  nodeUpdateStrategy:
    rollingUpdate:
      maxUnavailable: 1
    type: RollingUpdate
  nonPrivileged: Disabled
  registry: container-registry.oracle.com/
  variant: Calico
...
```

3
Using Calico

> **Warning**
>
> Oracle Cloud Native Environment 1.8 is now in Sustaining Support. See Oracle Open Source Support Policies for more information. New security patches and bug fixes are no longer provided for this software.
>
> Migrate applications and data to Oracle Cloud Native Environment Release 2.<latest> as soon as possible.

This section provides examples of using the Kubernetes and Calico network policies to define network policies for traffic in an NGINX web server application. The Kubernetes network policy example uses the network policy available in the Kubernetes Network Policy API. The Calico network policy example uses extensions available in the Calico policy, and provides more options for setting network policies.

Kubernetes Network Policy

This section shows a basic example of using the Kubernetes network policy. More examples of the Kubernetes network policy are available in the upstream Calico documentation.

The example defines a NetworkPolicy to disable all network traffic between pods in a namespace. NetworkPolicy resources are created to define rules to specify allowed network traffic between the application pods. If a NetworkPolicy in a namespace selects a pod, that pod rejects any connections that aren't allowed by the NetworkPolicy.

To create a test application to use a Kubernetes network policy:

1. Create a Kubernetes namespace for the test application:

   ```
   kubectl create namespace myapp
   ```

2. Create an NGINX pod with label of `app: nginx`. This is created in the `myapp` namespace. On a control plane node, create a file named `nginx.yaml` and copy the following into the file.

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: nginx
     namespace: myapp
     labels:
       app: nginx
   spec:
       containers:
       - name: nginx
         image: container-registry.oracle.com/olcne/nginx:1.17.7
   ```

```
      ports:
      - containerPort: 80
```

3. Start the NGINX pod:

   ```
   kubectl apply -f nginx.yaml
   ```

4. Create an Oracle Linux 9 pod with label of `app: nginx`. This is created in the `myapp` namespace. Create a file named `ol9.yaml` and copy the following into the file:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: ol9
     namespace: myapp
     labels:
       app: nginx
   spec:
     containers:
     - name: ol9
       command: ["/bin/sh", "-c", "trap : TERM INT; sleep infinity & wait"]
       image: container-registry.oracle.com/os/oraclelinux:9-slim
   ```

5. Start the pod using the `kubectl` command:

   ```
   kubectl apply -f ol9.yaml
   ```

6. You can see the `nginx-app` and `ol9` pods are running using the `kubectl get pods` command:

   ```
   kubectl get pods --namespace myapp -o wide
   ```

   The output looks similar to:

   ```
   NAME    READY   STATUS    RESTARTS   AGE   IP
   NODE                        ...
   nginx   1/1     Running   0          38s   10.244.140.68
   worker1.example.com         ...
   ol9     1/1     Running   0          22h   10.244.80.196
   worker2.example.com         ...
   ```

7. Mount the Oracle Linux 9 pod, `ol9`, and run `curl` to connect to the NGINX server pod, `nginx`, using the IP address assigned to the pod.

   ```
   kubectl exec -it ol9 --namespace myapp -- curl 10.244.140.68
   ```

   The output looks similar to:

   ```
   <!DOCTYPE html>
   <html>
   <head>
   <title>Welcome to nginx!</title>
   <style>
       body {
   ```

```
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

8. Create a NetworkPolicy to deny all traffic to all pods in the `myapp` namespace. Create a file named `deny-all.yaml` and copy the following into the file.

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: default-deny
  namespace: myapp
spec:
  podSelector:
    matchLabels: {}
```

9. Create the NetworkPolicy using the `kubectl` command:

```
kubectl apply -f deny-all.yaml
```

> **Tip**
>
> You can get a list of all NetworkPolicies in the `myapp` namespace using:
>
> ```
> kubectl get networkpolicies --namespace myapp
> ```
>
> In addition, adding the `-o yaml` option lists details about the NetworkPolicies.

10. Mount the Oracle Linux 9 pod, `ol9`, again and run `curl` to connect to the NGINX server pod.

```
kubectl exec -it ol9 --namespace myapp -- curl 10.244.140.68
```

This time no results are returned as the traffic between pods on that namespace isn't allowed. You can enter `Ctrl+C` to exit the command.

11. Create a NetworkPolicy to allow ingress traffic on port 80 for pods in the `myapp` namespace. This lets pods connect to the NGINX server. Create a file named `ingress.yaml` and copy the following into the file:

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: myapp-ingress
  namespace: myapp
spec:
  podSelector:
    matchLabels:
      app: nginx
  ingress:
  - from:
    - podSelector:
        matchLabels:
          app: nginx
    ports:
      - port: 80
```

12. Create the NetworkPolicy using the `kubectl` command:

```
kubectl apply -f ingress.yaml
```

13. Mount the Oracle Linux 9 pod, `ol9`, again and run `curl` to connect to the NGINX server pod.

```
kubectl exec -it ol9 --namespace myapp -- curl 10.244.140.68
```

    The NGINX page is now returned as ingress is allowed to the pods.

14. You can delete the resources created in this Calico test using:

```
kubectl delete pod --namespace myapp nginx
kubectl delete pod --namespace myapp ol9
kubectl delete networkpolicies --namespace myapp myapp-ingress
kubectl delete networkpolicies --namespace myapp default-deny
kubectl delete namespace myapp
```

Calico Network Policy

This section shows a basic example of using the Calico policy extensions for the Kubernetes Network Policy API. More examples of the Calico policy are available in the upstream documentation.

The example defines a GlobalNetworkPolicy to disable all network traffic, except for Kubernetes system and Calico pods. NetworkPolicy resources are created to define rules to specify allowed network traffic between the application pods. Any other pod traffic isn't allowed.

To create a test application to use a Calico network policy:

1. Create a Kubernetes namespace for the test application:

   ```
   kubectl create namespace myapp
   ```

2. Create an NGINX pod with label of `app: nginx`. This is created in the `myapp` namespace. On a control plane node, create a file named `nginx.yaml` and copy the following into the file.

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: nginx
     namespace: myapp
     labels:
       app: nginx
   spec:
       containers:
       - name: nginx
         image: container-registry.oracle.com/olcne/nginx:1.17.7
         ports:
         - containerPort: 80
   ```

3. Start the NGINX pod:

   ```
   kubectl apply -f nginx.yaml
   ```

4. Create an Oracle Linux 9 pod with label of `app: nginx`. This is created in the `myapp` namespace. Create a file named `ol9.yaml` and copy the following into the file:

   ```
   apiVersion: v1
   kind: Pod
   metadata:
     name: ol9
     namespace: myapp
     labels:
       app: nginx
   spec:
     containers:
     - name: ol9
       command: ["/bin/sh", "-c", "trap : TERM INT; sleep infinity & wait"]
       image: container-registry.oracle.com/os/oraclelinux:9-slim
   ```

5. Start the pod using the `kubectl` command:

   ```
   kubectl apply -f ol9.yaml
   ```

6. You can see the `nginx-app` and `ol9` pods are running using the `kubectl get pods` command:

   ```
   kubectl get pods --namespace myapp -o wide
   ```

The output looks similar to:

```
NAME    READY   STATUS    RESTARTS   AGE   IP
NODE                        ...
nginx   1/1     Running   0          38s   10.244.140.68
worker1.example.com         ...
ol9     1/1     Running   0          22h   10.244.80.196
worker2.example.com         ...
```

7. Mount the Oracle Linux 9 pod, `ol9`, and run `curl` to connect to the NGINX server pod, `nginx`, using the IP address assigned to the pod.

```
kubectl exec -it ol9 --namespace myapp -- curl 10.244.140.68
```

The output looks similar to:

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
        font-family: Tahoma, Verdana, Arial, sans-serif;
    }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

8. Create a GlobalNetworkPolicy to deny all traffic to all pods in the namespaces that aren't in the `kube-system`, `calico-system` or `calico-apiserver` namespaces. Create a file named `deny-all.yaml` and copy the following into the file.

```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: default-deny
spec:
  selector: projectcalico.org/namespace not in  {'kube-system', 'calico-
system', 'calico-apiserver'}
  types:
```

```
  - Ingress
  - Egress
```

9. Create the GlobalNetworkPolicy using the `kubectl` command:

   ```
   kubectl apply -f deny-all.yaml
   ```

   Tip

   You can get a list of all Calico GlobalNetworkPolicies in the `myapp` namespace using:

   ```
   kubectl get globalnetworkpolicies.crd.projectcalico.org --namespace
   myapp
   ```

   In addition, adding the `-o yaml` option lists details about the GlobalNetworkPolicies.

10. Mount the Oracle Linux 9 pod, `ol9`, again and run `curl` to connect to the NGINX server pod.

    ```
    kubectl exec -it ol9 --namespace myapp -- curl 10.244.140.68
    ```

    This time no results are returned as the traffic between pods on that namespace isn't allowed. You can enter `Ctrl+C` to exit the command.

11. Create a NetworkPolicy to allow ingress traffic on port 80 for pods in the `myapp` namespace. This lets pods connect to the NGINX server. Create a file named `ingress.yaml` and copy the following into the file:

    ```
    apiVersion: projectcalico.org/v3
    kind: NetworkPolicy
    metadata:
      name: allow-nginx-ingress
      namespace: myapp
    spec:
      selector: app == 'nginx'
      types:
      - Ingress
      ingress:
      - action: Allow
        protocol: TCP
        source:
          selector: app == 'nginx'
        destination:
          ports:
            - 80
    ```

12. Create the NetworkPolicy using the `kubectl` command:

    ```
    kubectl apply -f ingress.yaml
    ```

> **Tip**
>
> You can get a list of all Calico NetworkPolicies in the `myapp` namespace using:
>
> ```
> kubectl get networkpolicies.crd.projectcalico.org --namespace myapp
> ```
>
> In addition, adding the `-o yaml` option lists details about the NetworkPolicies.

13. Mount the Oracle Linux 9 pod, `ol9`, again and run `curl` to connect to the NGINX server pod.

    ```
    kubectl exec -it ol9 --namespace myapp -- curl 10.244.140.68
    ```

    No results are returned as egress traffic between pods on that namespace isn't allowed. You can enter `Ctrl+C` to exit the command.

14. Create a NetworkPolicy to allow egress traffic for pods in the `myapp` namespace. Create a file named `egress.yaml` and copy the following into the file:

    ```
    apiVersion: projectcalico.org/v3
    kind: NetworkPolicy
    metadata:
      name: allow-nginx-egress
      namespace: myapp
    spec:
      selector: app == 'nginx'
      types:
      - Egress
      egress:
      - action: Allow
    ```

15. Create the NetworkPolicy using the `kubectl` command:

    ```
    kubectl apply -f egress.yaml
    ```

16. Mount the Oracle Linux 9 pod, `ol9`, again and run `curl` to connect to the NGINX server pod.

    ```
    kubectl exec -it ol9 --namespace myapp -- curl 10.244.140.68
    ```

    The NGINX page is now returned as both ingress and egress is allowed for the pods.

17. You can delete the resources created in this Calico test using:

    ```
    kubectl delete pod --namespace myapp nginx
    kubectl delete pod --namespace myapp ol9
    kubectl delete networkpolicies.crd.projectcalico.org --namespace myapp
    default.allow-nginx-egress
    kubectl delete networkpolicies.crd.projectcalico.org --namespace myapp
    default.allow-nginx-ingress
    kubectl delete globalnetworkpolicies.crd.projectcalico.org --namespace
    ```

```
myapp default.default-deny
kubectl delete namespace myapp
```

4

Removing the Calico Module

> **Warning**
>
> Oracle Cloud Native Environment 1.8 is now in Sustaining Support. See Oracle Open Source Support Policies for more information. New security patches and bug fixes are no longer provided for this software.
>
> Migrate applications and data to Oracle Cloud Native Environment Release 2.<latest> as soon as possible.

You can remove a deployment of the Calico module and leave the Kubernetes cluster in place. To do this, you remove the Calico module from the environment.

> **Important**
>
> If you remove the Calico module, no Kubernetes CNI is set for the pods in the cluster.

Use the `olcnectl module uninstall` command to remove the Calico module. For example, to uninstall the Calico module named `mycalico` in the environment named `myenvironment`:

```
olcnectl module uninstall \
--environment-name myenvironment \
--name mycalico
```

The Calico module is removed from the environment.

As the networking policies to secure the cluster are removed when you uninstall the Calico module, it's highly recommended you enable the `firewalld` service on each Kubernetes node to protect the cluster. To enable the `firewalld` service, on each Kubernetes node, run:

```
sudo systemctl start firewalld.service
sudo systemctl enable firewalld.service
```