Oracle Cloud Native Environment

NGINX Ingress Controller Module for Release 1.9

F93862-01
May 2024

Oracle Cloud Native Environment NGINX Ingress Controller Module for Release 1.9,

F93862-01

Contents

Preface

This document contains information about setting up and using the NGINX Ingress Controller module in Oracle Cloud Native Environment. It describes how to prepare a virtual machine image and use it to create a virtual machine instance using NGINX Ingress Controller module and deploy it into a Kubernetes cluster.

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 the NGINX Ingress Controller Module

The NGINX Ingress Controller module is used to install the NGINX Ingress Controller.

The default namespace for NGINX Ingress Controller module is `ingress-nginx`.

The NGINX Ingress Controller is an implementation of the Kubernetes Ingress Resource to access Services within a cluster. For information on the Kubernetes Ingress Resource, see the upstream documentation.

The NGINX Ingress Controller provides the options to use various Custom Resource Definitions (CRDs), such as Ingress, VirtualServer, VirtualServerRoute and TransportServer.

For more information on the NGINX Ingress Controller, see the upstream documentation.

2

Installing the NGINX Ingress Controller Module

This chapter discusses how to install the NGINX Ingress Controller module on Oracle Cloud Native Environment.

Prerequisites

This section contains the prerequisites for installing the NGINX Ingress Controller module.

Load Balancer

The NGINX Ingress Controller needs a load balancer. You must provide a load balancer service in the environment. You don't need to create a load balancer (one is created for you when you install the NGINX Ingress Controller module), but you must install a module that provides one.

You can use MetalLB for bare metal environments, or the Oracle Cloud Infrastructure load balancer when using Oracle Cloud Infrastructure.

For information on installing MetalLB, see MetalLB Module.

For information on installing the Oracle Cloud Infrastructure Cloud Controller Manager module, see Oracle Cloud Infrastructure Cloud Controller Manager Module.

Deploying the NGINX Ingress Controller Module

You can deploy all the modules required to set up the NGINX Ingress Controller module using a single `olcnectl module create` command. This method might be useful to deploy the NGINX Ingress Controller module at the same time as deploying a Kubernetes cluster.

If you have an existing deployment of the Kubernetes module, you can specify that instance when deploying the NGINX Ingress Controller module.

This section guides you through installing each component required to deploy the NGINX Ingress Controller module.

For the full list of the Platform CLI command options available when creating modules, see the `olcnectl module create` command in Platform Command-Line Interface.

To deploy the NGINX Ingress Controller module:

1. If you don't already have an environment set up, create one into which the modules can be deployed. For information on setting up an environment, see Installation. The name of the environment in this example is `myenvironment`.
2. If you don't already have a Kubernetes module set up or deployed, set one up. For information on adding a Kubernetes module to an environment, see Kubernetes Module. The name of the Kubernetes module in this example is `mycluster`.

3. Create an NGINX Ingress Controller module and associate it with the Kubernetes module named `mycluster` using the `--ingress-nginx-kubernetes-module` option. In this example, the NGINX Ingress Controller module is named `myingress-nginx`.

```
olcnectl module create \
--environment-name myenvironment \
--module ingress-nginx \
--name myingress-nginx \
--ingress-nginx-kubernetes-module mycluster
```

The `--module` option sets the module type to create, which is `ingress-nginx`. You define the name of the NGINX Ingress Controller module using the `--name` option, which in this case is `myingress-nginx`.

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

If you're using the Oracle Cloud Infrastructure Cloud Controller Manager module to provide a load balancer, include the `--ingress-controller-service-annotations` option to configure a load balancer to use with the NGINX Ingress Controller. The load balancer is created when you install the module.

```
--ingress-controller-service-annotations {annotation,...}
```

For example, some options to provision an Oracle Cloud Infrastructure load balancer might include the following, in a comma separated list:

```
service.beta.kubernetes.io/oci-load-balancer-shape: flexible
service.beta.kubernetes.io/oci-load-balancer-shape-flex-max: "100"
service.beta.kubernetes.io/oci-load-balancer-shape-flex-min: "10"
service.beta.kubernetes.io/oci-load-balancer-internal: "true"
```

In some Oracle Cloud Infrastructure tenancies, you might also need to include the `oci-load-balancer-subnet1` annotation to identify the network subnet, for example:

```
service.beta.kubernetes.io/oci-load-balancer-subnet1:
"ocid1.subnet.oc1..unique_ID"
```

For the full list of annotations you can include, see the upstream documentation.

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

4. Use the `olcnectl module install` command to install the NGINX Ingress Controller module. For example:

```
olcnectl module install \
--environment-name myenvironment \
--name myingress-nginx
```

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 NGINX Ingress Controller module is deployed into the Kubernetes cluster.

Verifying the NGINX Ingress Controller Module Deployment

You can verify the NGINX Ingress Controller 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
mycluster                      kubernetes             installed
myingress-nginx                ingress-nginx          installed
...
```

Note the entry for `ingress-nginx` 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 NGINX Ingress Controller module named `myingress-nginx` in `myenvironment`:

```
olcnectl module report \
--environment-name myenvironment \
--name myingress-nginx \
--children
```

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

On a control plane node, verify the NGINX Ingress Controller deployment is running in the `ingress-nginx` namespace:

```
kubectl get deployments --namespace ingress-nginx
```

The output looks similar to:

```
NAME                         READY   UP-TO-DATE   AVAILABLE   AGE
myingress-nginx-controller   1/1     1            1           10m
```

You can also show the settings in the ConfigMap, for example:

```
kubectl describe configmaps --namespace ingress-nginx myingress-nginx-
controller
```

The output looks similar to:

```
Name:         myingress-nginx-controller
Namespace:    ingress-nginx
Labels:       app.kubernetes.io/component=controller
              app.kubernetes.io/instance=myingress-nginx
              app.kubernetes.io/managed-by=Helm
              app.kubernetes.io/name=ingress-nginx
              app.kubernetes.io/part-of=ingress-nginx
              app.kubernetes.io/version=1.9.6
              helm.sh/chart=ingress-nginx-4.9.1
Annotations:  meta.helm.sh/release-name: myingress-nginx
              meta.helm.sh/release-namespace: ingress-nginx

Data
====
allow-snippet-annotations:
----
false

BinaryData
====

Events:
  Type    Reason  Age   From                      Message
  ----    ------  ----  ----                      -------
  Normal  CREATE  11m   nginx-ingress-controller  ConfigMap ingress-
nginx/myingress-nginx-controller
```

Verify the NGINX Ingress Controller service is running in the `ingress-nginx` namespace:

```
kubectl get service --namespace ingress-nginx
```

The output looks similar to:

```
NAME                                 TYPE           CLUSTER-IP
EXTERNAL-IP      PORT(S)                      AGE
myingress-nginx-controller           LoadBalancer   10.104.157.191
203.0.113.11     80:30154/TCP,443:30394/TCP   12m
myingress-nginx-controller-admission   ClusterIP      10.105.18.193
<none>           443/TCP                      12m
```

You can also show the settings in the service, for example:

```
kubectl describe service --namespace ingress-nginx myingress-nginx-
controller
```

3
Using the NGINX Ingress Controller

This chapter includes two examples to test ingress rules with the NGINX Ingress Controller.

You can create ingress rules, using one, or more Kubernetes Ingress, VirtualServer, or VirtualServerRoute Custom Resource Definitions (CRDs). These definitions set up the ingress configuration.

Example 3-1 Single Application Routing Test

This example creates an ingress rule that directs incoming traffic to an NGINX application.

1. Create a Deployment and associated Service in a CRD YAML file:

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: container-registry.oracle.com/olcne/nginx:1.17.7
        ports:
        - containerPort: 80
---
kind: Service
apiVersion: v1
metadata:
  name: nginx-service
spec:
  selector:
    app: nginx
  ports:
  - name: http
    port: 80
    targetPort: 80
```

2. Create the Deployment and Service using:

```
kubectl apply -f filename.yaml
```

3. Create an Ingress CRD in a YAML file for the NGINX service.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
spec:
  ingressClassName: nginx
  rules:
  - http:
      paths:
        - path: /
          pathType: Prefix
          backend:
            service:
              name: nginx-service
              port:
                number: 80
```

4. You can then create the Ingress using:

```
kubectl apply -f filename.yaml
```

5. You can see the Ingress is created using:

```
kubectl get ingress
```

The output looks similar to:

```
NAME              CLASS   HOSTS   ADDRESS           PORTS   AGE
example-ingress   nginx   *       203.0.113.11      80      7m16s
```

The ADDRESS shown is also the address for the load balancer. You can confirm this using:

```
kubectl get service --namespace ingress-nginx
```

The output looks similar to:

```
NAME                                      TYPE           CLUSTER-
IP       EXTERNAL-IP       PORT(S)                      AGE
myingress-nginx-controller                LoadBalancer
10.106.167.143   203.0.113.11      80:31036/TCP,443:32343/TCP   7d1h
myingress-nginx-controller-admission   ClusterIP
10.105.9.191     <none>            443/TCP                      7d1h
```

6. You can show the ingress rules using:

```
kubectl describe ingress example-ingress
```

The output looks similar to:

```
Name:             example-ingress
Labels:           <none>
Namespace:        default
Address:          203.0.113.11
Ingress Class:    nginx
Default backend:  <default>
Rules:
  Host        Path  Backends
  ----        ----  --------
  *
              /   nginx-service:80
(10.244.1.4:80,10.244.3.4:80,10.244.4.3:80 + 1 more...)
Annotations:  <none>
Events:
  Type    Reason  Age                From                      Message
  ----    ------  ----               ----                      -------
  Normal  Sync    14m (x2 over 15m)  nginx-ingress-controller  Scheduled
for sync
```

This shows that all traffic is to be routed to the `nginx-service`.

7. Use the `curl` command to create a request to the NGINX application. Use the IP address of the load balancer and the Ingress.

```
curl http://203.0.113.11
```

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. You can delete the Ingress, Deployment, and Service using:

```
kubectl delete ingress example-ingress
kubectl delete service nginx-service
kubectl delete deployment nginx-deployment
```

Example 3-2 Two Applications and URL Redirection Test

This test application creates two NGINX Deployments with customized HTML pages and an Service for each Deployment. An ingress rule is then created to redirect traffic to these services depending on the path entered in the URL.

1. Create a YAML file that contains the HTML responses to be provided by NGINX instead of the default web server page. A ConfigMap is provided for each Service.

```
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap-nginx-a
data:
  index.html: |
    <html>
      <head>
        <title>NGINX Application A</title>
      </head>
      <body>
        <h1>This is from path /a to service nginx-a.</h1>
      </body>
    </html>
---
apiVersion: v1
kind: ConfigMap
metadata:
  name: configmap-nginx-b
data:
  index.html: |
    <html>
      <head>
        <title>NGINX Application B</title>
      </head>
      <body>
        <h1>This is from path /b to service nginx-b.</h1>
      </body>
    </html>
```

2. You can then create the ConfigMaps using:

```
kubectl apply -f filename.yaml
```

3. Create two NGINX Deployments and associated Services in a CRD YAML file. Volumes are created on each Deployment to use the ConfigMaps. Each Deployment provides a different HTML page to show which application is returning data.

```
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-a
  labels:
    app: nginx-a
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx-a
  template:
    metadata:
      labels:
        app: nginx-a
    spec:
      containers:
      - name: nginx-a
        image: container-registry.oracle.com/olcne/nginx:1.17.7
        ports:
        - containerPort: 80
        volumeMounts:
        - name: volume-a
          mountPath: /usr/share/nginx/html
      volumes:
      - name: volume-a
        configMap:
          name: configmap-nginx-a

---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment-b
  labels:
    app: nginx-b
spec:
  replicas: 2
  selector:
    matchLabels:
      app: nginx-b
  template:
    metadata:
      labels:
        app: nginx-b
    spec:
```

```
      containers:
      - name: nginx-b
        image: container-registry.oracle.com/olcne/nginx:1.17.7
        ports:
        - containerPort: 80
        volumeMounts:
        - name: volume-b
          mountPath: /usr/share/nginx/html
      volumes:
      - name: volume-b
        configMap:
          name: configmap-nginx-b
---
kind: Service
apiVersion: v1
metadata:
  name: nginx-service-a
spec:
  selector:
    app: nginx-a
  ports:
  - name: http
    port: 80
    targetPort: 80

---
kind: Service
apiVersion: v1
metadata:
  name: nginx-service-b
spec:
  selector:
    app: nginx-b
  ports:
  - name: http
    port: 80
    targetPort: 80
```

4. Create the Deployments and Services using:

   ```
   kubectl apply -f filename.yaml
   ```

5. Create an Ingress CRD in a YAML file for the NGINX Services. This Ingress directs any traffic that contains the path `/a` to `nginx-service-a`, and traffic that contains the path `/b` to `nginx-service-b`. No other traffic is allowed.

   ```
   apiVersion: networking.k8s.io/v1
   kind: Ingress
   metadata:
     name: example-ingress
     annotations:
       nginx.ingress.kubernetes.io/rewrite-target: /
   spec:
     ingressClassName: nginx
     rules:
   ```

```
  - http:
      paths:
        - path: /a
          pathType: Prefix
          backend:
            service:
              name: nginx-service-a
              port:
                number: 80
        - path: /b
          pathType: Prefix
          backend:
            service:
              name: nginx-service-b
              port:
                number: 80
```

6. You can then create the Ingress using:

   ```
   kubectl apply -f filename.yaml
   ```

7. You can show the ingress rules using:

   ```
   kubectl describe ingress example-ingress
   ```

   The output looks similar to:

   ```
   Name:             example-ingress
   Labels:           <none>
   Namespace:        default
   Address:          203.0.113.11
   Ingress Class:    nginx
   Default backend:  <default>
   Rules:
     Host        Path  Backends
     ----        ----  --------
     *
                 /a   nginx-service-a:80 (10.244.2.8:80,10.244.4.5:80)
                 /b   nginx-service-b:80 (10.244.2.9:80,10.244.4.4:80)
   Annotations:  nginx.ingress.kubernetes.io/rewrite-target: /
   Events:       <none>
   ```

8. Use the `curl` command to create a request to the `nginx-a` application by including the `/a` path in the URL. Use the IP address of the load balancer and the Ingress.

   ```
   curl http://203.0.113.11/a
   ```

   The output looks similar to:

   ```
   <html>
     <head>
       <title>NGINX Application A</title>
     </head>
   ```

```
  <body>
    <h1>This is from path /a to service nginx-a.</h1>
  </body>
</html>
```

The `nginx-a` Deployment responds with the default HTML page to show it's coming through the `nginx-service-a` Service.

9. Use the `curl` command to create a request to the `nginx-b` application by including the `/b` path in the URL.

```
curl http://203.0.113.11/b
```

The output looks similar to:

```
<html>
  <head>
    <title>NGINX Application B</title>
  </head>
  <body>
    <h1>This is from path /b to service nginx-b.</h1>
  </body>
</html>
```

The `nginx-b` Deployment responds with the default HTML page to show it's coming through the `nginx-service-b` Service.

10. Use the `curl` command to create a request without adding any path to the URL.

```
curl http://203.0.113.11
```

The output looks similar to:

```
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

As no ingress rule is provided for this URL, no traffic is allowed.

11. You can delete the Ingress, Deployments, Services, and ConfigMaps using:

```
kubectl delete ingress example-ingress
kubectl delete service nginx-service-a
kubectl delete service nginx-service-b
kubectl delete deployment nginx-deployment-a
kubectl delete deployment nginx-deployment-b
kubectl delete configmaps configmap-nginx-a
kubectl delete configmaps configmap-nginx-b
```

4

Removing the NGINX Ingress Controller Module

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

Use the `olcnectl module uninstall` command to remove the NGINX Ingress Controller module. For example, to uninstall the NGINX Ingress Controller module named `mynginx-ingress` in the environment named `myenvironment`:

```
olcnectl module uninstall \
--environment-name myenvironment \
--name mynginx-ingress
```

The NGINX Ingress Controller module is removed from the environment.