Oracle Cloud Native Environment

Application Load Balancers for Release 1.5

F55665-04
July 2023

Oracle Cloud Native Environment Application Load Balancers for Release 1.5,

F55665-04

Contents

Preface

Important:

The software described in this documentation is either in Extended Support or Sustaining Support. See Oracle Open Source Support Policies for more information.

We recommend that you upgrade the software described by this documentation as soon as possible.

This document contains information about setting up network load balancers for Kubernetes applications in Oracle Cloud Native Environment. It describes the modules provided with Oracle Cloud Native Environment to set up application load balancers. These load balancers can be used with Kubernetes LoadBalancer services to externalize applications outside of the Kubernetes cluster.

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/.

For information about the accessibility of the Oracle Help Center, see the Oracle Accessibility Conformance Report at https://www.oracle.com/corporate/accessibility/templates/t2-11535.html.

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 Application Load Balancers

Important:

The software described in this documentation is either in Extended Support or Sustaining Support. See Oracle Open Source Support Policies for more information.

We recommend that you upgrade the software described by this documentation as soon as possible.

Network load balancers provide a method of externally exposing Kubernetes applications. A Kubernetes LoadBalancer service is used to create a network load balancer that provides and exposes an external IP address that can be used to connect to an application from outside the cluster.

More information on Kubernetes services, including the LoadBalancer service, is available in the upstream documentation at:

https://kubernetes.io/docs/concepts/services-networking/service/

Oracle Cloud Native Environment provides two methods to create a LoadBalancer service: using the Oracle Cloud Infrastructure load balancer, or using MetalLB.

The Oracle Cloud Infrastructure load balancer provides network load balancers for Kubernetes applications running on Oracle Cloud Infrastructure.

MetalLB is a network load balancer for Kubernetes applications running on bare metal hosts. MetalLB allows you to use Kubernetes LoadBalancer services, which traditionally make use of a cloud provider network load balancer, in a bare metal environment.

Introduction to the Oracle Cloud Infrastructure Cloud Controller Manager Module

The Oracle Cloud Infrastructure Flexible Network Load Balancing service (Oracle Cloud Infrastructure load balancer) provides automated traffic distribution from one entry point to multiple backend servers in a Virtual Cloud Network (VCN). It operates at the connection level and load balances incoming client connections to healthy backend servers based on Layer 3/Layer 4 (IP protocol) data.

For more information on the Oracle Cloud Infrastructure load balancer, see the Oracle Cloud Infrastructure documentation.

The Oracle Cloud Infrastructure Cloud Controller Manager module is used to create and manage Oracle Cloud Infrastructure load balancers for Kubernetes applications. The Oracle Cloud Infrastructure Cloud Controller Manager module is deployed by the Helm module into a Kubernetes cluster.

The Oracle Cloud Infrastructure Cloud Controller Manager module uses the Kubernetes Cloud Controller Manager (`oci-cloud-controller-manager`) to provide and manage Oracle Cloud Infrastructure load balancers. The Kubernetes Cloud Controller Manager ServiceController is responsible for creating load balancers when a Kubernetes LoadBalancer service is created.

The Platform API Server communicates with the Oracle Cloud Infrastructure API to provision and manage Oracle Cloud Infrastructure load balancers.

For more information on the Kubernetes Cloud Controller Manager, see the upstream documentation at:

https://github.com/oracle/oci-cloud-controller-manager

Introduction to the MetalLB Module

MetalLB is a network load balancer for Kubernetes applications running on bare metal hosts. MetalLB allows you to use Kubernetes LoadBalancer services, which traditionally make use of a cloud provider network load balancer, in a bare metal environment.

MetalLB has two features that enable the network load balancer: address allocation, and external announcement.

Address allocation provides IP addresses to Kubernetes applications from the pool of IP addresses you provide in the ConfigMap file.

External announcement makes the network beyond the Kubernetes cluster aware that the IP is available in the cluster. This is provided using either Address Resolution Protocol (ARP) and Neighbor Discover Protocol (NDP) in Layer 2 mode, or Border Gateway Protocol (BGP) in BGP mode.

For more information on MetalLB, see the upstream documentation at:

https://metallb.universe.tf/concepts/

The MetalLB module is used to set up network load balancers for Kubernetes applications using MetalLB. The MetalLB module is deployed by the Helm module into a Kubernetes cluster.

Oracle Cloud Native Environment deploys MetalLB onto the control plane nodes using a ConfigMap file you set up beforehand.

2

Using the Oracle Cloud Infrastructure Load Balancer

Important:

The software described in this documentation is either in Extended Support or Sustaining Support. See Oracle Open Source Support Policies for more information.

We recommend that you upgrade the software described by this documentation as soon as possible.

This chapter discusses how to install and use the Oracle Cloud Infrastructure Cloud Controller Manager module to set up a load balancer for Kubernetes applications in Oracle Cloud Native Environment on Oracle Cloud Infrastructure instances.

Prerequisites

This section contains the prerequisite information you need to set up the Oracle Cloud Infrastructure Cloud Controller Manager module.

Setting up the Health Check Endpoint Network Ports

When using a Kubernetes LoadBalancer service with the `ServiceInternalTrafficPolicy` set to `Cluster` (the default), a health check endpoint is expected to be available on TCP port 10256. `kube-proxy` creates a listener on this port, which enables access to the LoadBalancer service to verify that `kube-proxy` is healthy on the nodes. The LoadBalancer service determines which nodes can have traffic routed to them using this policy. To allow traffic on this port, you must open TCP port 10256 on all Kubernetes nodes. On each Kubernetes node, run:

```
sudo firewall-cmd --zone=public --add-port=10256/tcp
sudo firewall-cmd --zone=public --add-port=10256/tcp --permanent
sudo systemctl restart firewalld.service
```

For more information on the `ServiceInternalTrafficPolicy`, see the upstream documentation at:

https://kubernetes.io/docs/concepts/services-networking/service-traffic-policy/

Make sure traffic is allowed for TCP port 10256 in the network security list.

Gather Oracle Cloud Infrastructure Identifiers

Gather information about your Oracle Cloud Infrastructure environment. The most common information you need is:

- The identifier for the region.
- The OCID for the tenancy.
- The OCID for the compartment.
- The OCID for the user.
- The public key fingerprint for the API signing key pair.
- The private key file for the API signing key pair. The private key must be copied to the primary control plane node. This is the first control plane node listed in the `--master-nodes` option when you create the Kubernetes module.
- The OCID for the Virtual Cloud Network (VCN).
- The OCIDs for two subnets in the VCN for high availability if required.
- The quota to use for the load balancers.
- The shape to use for the load balancers.

For information on finding each of these identifiers or components, see the Oracle Cloud Infrastructure documentation.

Deploying the Oracle Cloud Infrastructure Cloud Controller Manager Module

If you have already installed the Oracle Cloud Infrastructure Cloud Controller Manager module to make use of Oracle Cloud Infrastructure storage, you do not need to create another module to provision application load balancers. The Oracle Cloud Infrastructure Cloud Controller Manager module is used to provision both Oracle Cloud Infrastructure storage and load balancers.

You can deploy all the modules required to set up a Oracle Cloud Infrastructure load balancer for a Kubernetes cluster using a single `olcnectl module create` command. This method might be useful if you want to deploy the Oracle Cloud Infrastructure Cloud Controller Manager 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 Oracle Cloud Infrastructure Cloud Controller Manager module.

This section guides you through installing each component required to deploy the Oracle Cloud Infrastructure Cloud Controller Manager 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 Oracle Cloud Infrastructure Cloud Controller Manager module:

1. If you do not already have an environment set up, create one into which the modules can be deployed. For information on setting up an environment, see Getting Started. The name of the environment in this example is `myenvironment`.
2. If you do not already have a Kubernetes module set up or deployed, set one up.

   For information on adding a Kubernetes module to an environment, see Container Orchestration. The name of the Kubernetes module in this example is `mycluster`.

3. If you do not already have a Helm module created and installed, create one. The Helm module in this example is named `myhelm` and is associated with the Kubernetes module named `mycluster` using the `--helm-kubernetes-module` option.

   ```
   olcnectl module create \
   --environment-name myenvironment \
   --module helm \
   --name myhelm \
   --helm-kubernetes-module mycluster
   ```

4. If you are deploying a new Helm module, use the `olcnectl module validate` command to validate the Helm module can be deployed to the nodes. For example:

   ```
   olcnectl module validate \
   --environment-name myenvironment \
   --name myhelm
   ```

5. If you are deploying a new Helm module, use the `olcnectl module install` command to install the Helm module. For example:

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

   The Helm software packages are installed on the control plane nodes, and the Helm module is deployed into the Kubernetes cluster.

6. Create an Oracle Cloud Infrastructure Cloud Controller Manager module and associate it with the Helm module named `myhelm` using the `--oci-ccm-helm-module` option. In this example, the Oracle Cloud Infrastructure Cloud Controller Manager module is named `myoci`.

   ```
   olcnectl module create \
   --environment-name myenvironment \
   --module oci-ccm \
   --name myoci \
   --oci-ccm-helm-module myhelm \
   --oci-region us-ashburn-1 \
   --oci-tenancy ocid1.tenancy.oc1..unique_ID \
   --oci-compartment ocid1.compartment.oc1..unique_ID \
   --oci-user ocid1.user.oc1..unique_ID \
   --oci-fingerprint b5:52:... \
   --oci-private-key /home/opc/.oci/oci_api_key.pem \
   --oci-vcn ocid1.vcn.oc1..unique_ID \
   --oci-lb-subnet1 ocid1.subnet.oc1..unique_ID
   ```

   The `--module` option sets the module type to create, which is `oci-ccm`. You define the name of the Oracle Cloud Infrastructure Cloud Controller Manager module using the `--name` option, which in this case is `myoci`.

   The `--oci-ccm-helm-module` option sets the name of the Helm module. If there is an existing Helm module with the same name, the Platform API Server uses that instance of Helm.

The `--oci-region` option sets the Oracle Cloud Infrastructure region to use. The region in this example is `us-ashburn-1`.

The `--oci-tenancy` option sets the OCID for your tenancy.

The `--oci-compartment` option sets the OCID for your compartment.

The `--oci-user` option sets the OCID for the user.

The `--oci-fingerprint` option sets the fingerprint for the public key for the Oracle Cloud Infrastructure API signing key.

The `--oci-private-key` option sets the location of the private key for the Oracle Cloud Infrastructure API signing key. The private key must be available on the primary control plane node.

The `--oci-vcn` option sets the OCID for the VCN on which to create load balancers.

The `--oci-lb-subnet1` option sets the OCID for the VCN subnet on which to create load balancers.

If you want to set up high availability for the load balancer, you should provide a second subnet on a different availability domain using the `--oci-lb-subnet2` option. For example:

```
--oci-lb-subnet2 ocid1.subnet.oc1..unique_ID \
```

Tip:

If you have an existing Oracle Cloud Infrastructure Cloud Controller Manager module used for Oracle Cloud Infrastructure storage, you can update it to include this networking information using the `olcnectl module update` command. This configures the module to provision load balancers. For example:

```
olcnectl module update \
--environment-name myenvironment \
--name myoci \
--oci-vcn ocid1.vcn.oc1..unique_ID \
--oci-lb-subnet1 ocid1.subnet.oc1..unique_ID \
--oci-lb-subnet2 ocid1.subnet.oc1..unique_ID
```

If you do not include all the required options when adding the modules, you are prompted to provide them.

7. Use the `olcnectl module validate` command to validate the Oracle Cloud Infrastructure Cloud Controller Manager module can be deployed to the nodes. For example:

```
olcnectl module validate \
--environment-name myenvironment \
--name myoci
```

8. Use the `olcnectl module install` command to install the Oracle Cloud Infrastructure Cloud Controller Manager module. For example:

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

   The Oracle Cloud Infrastructure Cloud Controller Manager module is deployed into the Kubernetes cluster.

Verifying the Oracle Cloud Infrastructure Cloud Controller Manager Module Deployment

You can verify the Oracle Cloud Infrastructure Cloud Controller Manager module is deployed using the `olcnectl module instances` command on the operator node. For example:

```
olcnectl module instances \
--environment-name myenvironment
INSTANCE                  MODULE        STATE
mycluster                 kubernetes    installed
myhelm                    helm          installed
myoci                     oci-ccm       installed
control1.example.com      node          installed
...
```

Note the entry for `oci-ccm` 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 Oracle Cloud Infrastructure Cloud Controller Manager module named `myoci` in `myenvironment`:

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

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

Creating an Application Using an Oracle Cloud Infrastructure Load Balancer

This section contains a basic test to verify you can create a Kubernetes application that uses an Oracle Cloud Infrastructure load balancer to provide external IP addresses.

To create a test application to use an Oracle Cloud Infrastructure load balancer:

1. Create a Kubernetes application that uses a LoadBalancer service. The deployment in this example creates an NGINX application with a replica count of 2, and an associated LoadBalancer service.

On a control plane node, create a file named `nginx-oci-lb.yaml` and copy the following into the 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
  annotations:
    service.beta.kubernetes.io/oci-load-balancer-security-list-
management-mode: "None"
    service.beta.kubernetes.io/oci-load-balancer-internal: "true"
    service.beta.kubernetes.io/oci-load-balancer-shape: "flexible"
    service.beta.kubernetes.io/oci-load-balancer-shape-flex-min:
"10"
    service.beta.kubernetes.io/oci-load-balancer-shape-flex-max:
"10"
spec:
  selector:
    app: nginx
  type: LoadBalancer
  ports:
  - name: http
    port: 80
    targetPort: 80
```

The `annotations` section contains the information required to provision an Oracle Cloud Infrastructure load balancer. This is where you set the load balancer shape.

For example, to use a 10Mbps shape instead of the flexible shape as shown in the example above, you might use:

```
 annotations:
    service.beta.kubernetes.io/oci-load-balancer-security-list-management-
mode: "None"
    service.beta.kubernetes.io/oci-load-balancer-internal: "true"
    service.beta.kubernetes.io/oci-load-balancer-shape: "10Mbps"
```

In some Oracle Cloud Infrastructure tenancies, you may 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 at:

https://github.com/oracle/oci-cloud-controller-manager/blob/master/docs/load-balancer-annotations.md

2. Start the NGINX deployment and LoadBalancer service:

```
kubectl apply -f nginx-oci-lb.yaml
deployment.apps/nginx-deployment created
service/nginx-service created
```

3. You can see the `nginx-deployment` application is running using the `kubectl get deployment` command:

```
kubectl get deployments.apps
NAME               READY   UP-TO-DATE   AVAILABLE   AGE
nginx-deployment   2/2     2            2           31s
```

4. You can see the `nginx-deployment` service is running using the `kubectl get svc` command:

```
kubectl get svc nginx-service
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP
PORT(S)        AGE
nginx-service   LoadBalancer   10.99.107.243   203.0.113.10
80:31288/TCP   10m
```

Oracle Cloud Infrastructure may take a few minutes to assign an IP address. Until this completes, the EXTERNAL-IP column shows the `pending` state for the `nginx-service`. When the IP address is assigned, this field changes to show the IP address.

Tip:

You can see the load balancer is created in the Oracle Cloud Infrastructure UI under **Networking** > **Load Balancers**.

You can see the EXTERNAL-IP for the `nginx-service` LoadBalancer has an IP address of `203.0.113.10`. This IP address is provided by Oracle Cloud Infrastructure and is the external IP address that you can use to connect to the application.

5. Use `curl` to connect to the NGINX application's IP address and add the port for the application (`203.0.113.10:80` in this example) to show the NGINX default page.

```
curl 203.0.113.10:80
<!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>
```

6. You can delete the `nginx-service` LoadBalancer service using:

```
kubectl delete svc nginx-service
service "nginx-service" deleted
```

Tip:

You can see the load balancer is removed in the Oracle Cloud Infrastructure UI under **Networking** > **Load Balancers**.

7. You can delete the `nginx-deployment` application using:

```
kubectl delete deployments.apps nginx-deployment
deployment.apps "nginx-deployment" deleted
```

Removing the Oracle Cloud Infrastructure Cloud Controller Manager Module

You can remove a deployment of the Oracle Cloud Infrastructure Cloud Controller Manager module and leave the Kubernetes cluster in place. To do this, you remove the Oracle Cloud Infrastructure Cloud Controller Manager module from the environment.

Use the `olcnectl module uninstall` command to remove the Oracle Cloud Infrastructure Cloud Controller Manager module. For example, to uninstall the Oracle Cloud Infrastructure Cloud Controller Manager module named `myoci` in the environment named `myenvironment`:

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

The Oracle Cloud Infrastructure Cloud Controller Manager module is removed from the environment.

3

Using the MetalLB Load Balancer

Important:

The software described in this documentation is either in Extended Support or Sustaining Support. See Oracle Open Source Support Policies for more information.

We recommend that you upgrade the software described by this documentation as soon as possible.

This chapter discusses how to install and use the MetalLB module to set up a network load balancer for Kubernetes applications using MetalLB in Oracle Cloud Native Environment on bare metal hosts.

Prerequisites

This section contains the prerequisite information you need to set up the MetalLB module.

Setting up the Health Check Endpoint Network Ports

When using a Kubernetes LoadBalancer service with the `ServiceInternalTrafficPolicy` set to `Cluster` (the default), a health check endpoint is expected to be available on TCP port 10256. `kube-proxy` creates a listener on this port, which enables access to the LoadBalancer service to verify that `kube-proxy` is healthy on the nodes. The LoadBalancer service determines which nodes can have traffic routed to them using this policy. To allow traffic on this port, you must open TCP port 10256 on all Kubernetes nodes. On each Kubernetes node, run:

```
sudo firewall-cmd --zone=public --add-port=10256/tcp
sudo firewall-cmd --zone=public --add-port=10256/tcp --permanent
sudo systemctl restart firewalld.service
```

For more information on the `ServiceInternalTrafficPolicy`, see the upstream documentation at:

https://kubernetes.io/docs/concepts/services-networking/service-traffic-policy/

Make sure traffic is allowed for TCP port 10256 in the network security list.

Setting up the Network Ports

You must open the following ports on Kubernetes worker nodes. On each worker node, run:

```
sudo firewall-cmd --zone=public --add-port=7946/tcp --permanent
sudo firewall-cmd --zone=public --add-port=7946/udp --permanent
sudo systemctl restart firewalld.service
```

Creating a MetalLB Configuration File

You must provide a MetalLB configuration file on the **operator** node. The configuration file contains the required information to configure MetalLB. This file is where you list configuration information such as the IP address ranges to use when provisioning load balancer IPs to Kubernetes applications, and the protocol to use.

The configuration file is a snippet, or cut down version, of the upstream MetalLB ConfigMap file. The snippet file should only contain the options available to be set under the `config` section shown in the upstream ConfigMap files, that is, any combination of `address-pools`, `peers`, `bgp-communities`, `bfd-profiles`, and so on. For example:

```
peers:
- peer-address: 10.0.0.1
  peer-asn: 64501
  my-asn: 64500
address-pools:
- name: default
  protocol: bgp
  addresses:
  - 192.168.10.0/24
```

The Platform API Server uses the information contained in the configuration file when creating the MetalLB module.

Important:

Oracle Cloud Native Environment installs MetalLB Release 0.12.1. This release uses a ConfigMap to configure the MetalLB cluster. MetalLB Release 0.13 onwards uses a CustomResource to perform this configuration. You should use the upstream examples for MetalLB Release 0.12.1 to create a snippet of a ConfigMap to configure the version of MetalLB installed with Oracle Cloud Native Environment.

For information on the options available to use in the configuration file, see the upstream documentation for the MetalLB ConfigMap file, at:

https://github.com/metallb/metallb/blob/v0.12.1/website/content/configuration/_index.md

Important:

Do not include a full ConfigMap file in the configuration file, only the options available under the `config` section.

The following example configuration file uses a MetalLB Layer 2 configuration and provides the IP address range from 192.168.1.240 to 192.168.1.250 to MetalLB to create load balancer IPs for Kubernetes applications. This example file is named `metallb-config.yaml` and contains:

```
address-pools:
- name: default
  protocol: layer2
  addresses:
  - 192.168.1.240-192.168.1.250
```

Deploying the MetalLB Module

You can deploy all the modules required to set up MetalLB for a Kubernetes cluster using a single `olcnectl module create` command. This method might be useful if you want to deploy the MetalLB 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 MetalLB module.

This section guides you through installing each component required to deploy the MetalLB 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 MetalLB module:

1. If you do not already have an environment set up, create one into which the modules can be deployed. For information on setting up an environment, see Getting Started. The name of the environment in this example is `myenvironment`.
2. If you do not already have a Kubernetes module set up or deployed, set one up. For information on adding a Kubernetes module to an environment, see Container Orchestration. The name of the Kubernetes module in this example is `mycluster`.
3. If you do not already have a Helm module created and installed, create one. The Helm module in this example is named `myhelm` and is associated with the Kubernetes module named `mycluster` using the `--helm-kubernetes-module` option.

   ```
   olcnectl module create \
   --environment-name myenvironment \
   --module helm \
   --name myhelm \
   --helm-kubernetes-module mycluster
   ```

4. If you are deploying a new Helm module, use the `olcnectl module validate` command to validate the Helm module can be deployed to the nodes. For example:

   ```
   olcnectl module validate \
   --environment-name myenvironment \
   --name myhelm
   ```

5. If you are deploying a new Helm module, use the `olcnectl module install` command to install the Helm module. For example:

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

The Helm software packages are installed on the control plane nodes, and the Helm module is deployed into the Kubernetes cluster.

6. Create a MetalLB module and associate it with the Helm module named `myhelm` using the `--metallb-helm-module` option. In this example, the MetalLB module is named `mymetallb`.

   ```
   olcnectl module create \
   --environment-name myenvironment \
   --module metallb \
   --name mymetallb \
   --metallb-helm-module myhelm \
   --metallb-config /home/opc/metallb-config.yaml
   ```

   The `--module` option sets the module type to create, which is `metallb`. You define the name of the MetalLB module using the `--name` option, which in this case is `mymetallb`.

   The `--metallb-helm-module` option sets the name of the Helm module. If there is an existing Helm module with the same name, the Platform API Server uses that instance of Helm.

   The `--metallb-config` option sets the location for the MetalLB configuration file. This file must be available on the operator node under the provided path. For information on creating this configuration file, see Creating a MetalLB Configuration File.

   If you do not include all the required options when adding the modules, you are prompted to provide them.

7. Use the `olcnectl module validate` command to validate the MetalLB module can be deployed to the nodes. For example:

   ```
   olcnectl module validate \
   --environment-name myenvironment \
   --name mymetallb
   ```

8. Use the `olcnectl module install` command to install the MetalLB module. For example:

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

   The MetalLB module is deployed into the Kubernetes cluster.

Verifying the MetalLB Module Deployment

You can verify the MetalLB module is deployed using the `olcnectl module instances` command on the operator node. For example:

```
olcnectl module instances \
--environment-name myenvironment
INSTANCE                  MODULE        STATE
mymetallb                 metallb       installed
mycluster                 kubernetes    installed
myhelm                    helm          installed
control1.example.com      node          installed
...
```

Note the entry for `metallb` 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 MetalLB module named `mymetallb` in `myenvironment`:

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

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

Creating an Application Using MetalLB

This section contains a basic test to verify you can create a Kubernetes application that uses MetalLB to provide external IP addresses.

To create a test application to use MetalLB:

1. Create a Kubernetes application that uses a LoadBalancer service. The deployment in this example creates an NGINX application with a replica count of 2, and an associated LoadBalancer service.

   On a control plane node, create a file named `nginx-metallb.yaml` and copy the following into the 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
     type: LoadBalancer
     ports:
   ```

```
    - name: http
      port: 80
      targetPort: 80
```

2. Start the NGINX deployment and LoadBalancer service:

   ```
   kubectl apply -f nginx-metallb.yaml
   deployment.apps/nginx-deployment created
   service/nginx-service created
   ```

3. You can see the `nginx-deployment` application is running using the `kubectl get deployment` command:

   ```
   kubectl get deployments.apps
   NAME               READY   UP-TO-DATE   AVAILABLE   AGE
   nginx-deployment   2/2     2            2           31s
   ```

4. You can see the `nginx-deployment` service is running using the `kubectl get svc` command:

   ```
   kubectl get svc
   NAME            TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)
   AGE
   kubernetes      ClusterIP      10.96.0.1       <none>          443/TCP
   25h
   nginx-service   LoadBalancer   10.99.253.99    192.168.1.240   80:31875/TCP
   70s
   ```

   You can see the EXTERNAL-IP for the `nginx-service` LoadBalancer has an IP address of `192.168.1.240`. This IP address is provided by MetalLB and is the external IP address that you can use to connect to the application.

5. Use `curl` to connect to the NGINX application's IP address and add the port for the application (`192.168.1.240:80` in this example) to show the NGINX default page.

   ```
   curl 192.168.1.240:80
   <!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>
```

6. You can delete the `nginx-service` LoadBalancer service using:

   ```
   kubectl delete svc nginx-service
   service "nginx-service" deleted
   ```

7. You can delete the `nginx-deployment` application using:

   ```
   kubectl delete deployments.apps nginx-deployment
   deployment.apps "nginx-deployment" deleted
   ```

Removing the MetalLB Module

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

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

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

The MetalLB module is removed from the environment.