Oracle Cloud Native Environment

Oracle Cloud Infrastructure Cloud Controller Manager Module for Release 1.8

F87565-01
January 2024

Oracle Cloud Native Environment Oracle Cloud Infrastructure Cloud Controller Manager Module for Release 1.8,

F87565-01

Contents

Preface

This document contains information about setting up and using the Oracle Cloud Infrastructure Cloud Controller Manager module to provide persistent storage and application load balancers in Oracle Cloud Native Environment. The Oracle Cloud Infrastructure Cloud Controller Manager module is used for Oracle Cloud Infrastructure instances.

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/.

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 the Oracle Cloud Infrastructure Cloud Controller Manager Module

The Oracle Cloud Infrastructure Cloud Controller Manager module is used to set up dynamically provisioned persistent storage using the Oracle Cloud Infrastructure Block Volume and File System storage services, and for creating load balancers for Kubernetes applications.

The Oracle Cloud Infrastructure Cloud Controller Manager module uses the Kubernetes Cloud Controller Manager (`oci-cloud-controller-manager`) to provision storage and application load balancers.

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

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

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

This document shows you how to install and use the Oracle Cloud Infrastructure Cloud Controller Manager module.

2

Installing the Oracle Cloud Infrastructure Cloud Controller Manager Module

This chapter discusses how to install the Oracle Cloud Infrastructure Cloud Controller Manager module on 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.

Gather Oracle Cloud Infrastructure Identifiers

Before you set up the Oracle Cloud Infrastructure Cloud Controller Manager module, you need to gather information about the 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.

You might need more information related to the Oracle Cloud Infrastructure networking or other components.

If you're using the Oracle Cloud Infrastructure Cloud Controller Manager module to provide load balancers for Kubernetes pods, you must also gather:

- 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.

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 sets access to the LoadBalancer service to verify that `kube-proxy` is healthy on the nodes. The LoadBalancer service decides

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 Kubernetes documentation.

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

Deploying the Module

The Oracle Cloud Infrastructure Cloud Controller Manager module is used to provision both Oracle Cloud Infrastructure storage and application load balancers. 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 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 Oracle Cloud Infrastructure Cloud Controller Manager module and associate it with the Kubernetes module named `mycluster` using the `--oci-ccm-kubernetes-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-kubernetes-module mycluster \
   --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-file /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-kubernetes-module` option sets the name of the Kubernetes module.

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 the tenancy.

The `--oci-compartment` option sets the OCID for the 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-file` path option sets the location of the private key for the Oracle Cloud Infrastructure API signing key. This must be on the operator node.

The `--oci-vcn` option sets the OCID for the VCN on which to create load balancers. You don't need to include this option if you're not using a load balancer.

The `--oci-lb-subnet1` option sets the OCID for the VCN subnet on which to create load balancers. You don't need to include this option if you aren't using a load balancer.

To set up high availability for a load balancer, 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 \
```

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 Oracle Cloud Infrastructure Cloud Controller Manager module. For example:

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

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

Verifying the 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
myoci                     oci-ccm       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.

If you have included the options to use application load balancers, on a control plane node, verify the `oci-bv` StorageClass for the Oracle Cloud Infrastructure provisioner is created using the `kubectl get sc` command:

```
kubectl get sc
```

The output looks similar to:

```
NAME     PROVISIONER                     RECLAIMPOLICY
VOLUMEBINDINGMODE      ...
oci-bv   blockvolume.csi.oraclecloud.com   Delete
WaitForFirstConsumer   ...
```

You can get more details about the StorageClass using the `kubectl describe sc` command. For example:

```
kubectl describe sc oci-bv
```

The output looks similar to:

```
Name:                  oci-bv
IsDefaultClass:        No
```

```
Annotations:           meta.helm.sh/release-name=myoci,meta.helm.sh/release-
namespace=default
Provisioner:           blockvolume.csi.oraclecloud.com
Parameters:            <none>
AllowVolumeExpansion:  <unset>
MountOptions:          <none>
ReclaimPolicy:         Delete
VolumeBindingMode:     WaitForFirstConsumer
Events:                <none>
```

3
Using Oracle Cloud Infrastructure Storage

The Oracle Cloud Infrastructure Cloud Controller Manager module implements a Container Storage Interface (CSI) plugin for Kubernetes clusters that provides you with the following storage services for creating persistent data storage:

Oracle Cloud Infrastructure Block Volume service
The Oracle Cloud Infrastructure Block Volume service provides dynamically provisioned and managed block storage volumes. The block storage volumes can be created in sizes ranging from 50 GB to 32 TB.
After you attach a volume to a Kubernetes application, the volume can be used in a similar way to a regular hard drive. The volume can also be disconnected and attached to another instance without the loss of data.
Block Volume volumes are automatically replicated to help protect against data loss.
To read about example use cases, and for more information about the Oracle Cloud Infrastructure Block Volume service, see the Oracle Cloud Infrastructure documentation.

Oracle Cloud Infrastructure File Storage service
The Oracle Cloud Infrastructure File Storage service provides a scalable and durable enterprise-grade network file system. The File Storage service uses the Network File System version 3.0 (NFSv3) protocol. The service uses the Network Lock Manager (NLM) protocol for file locking functionality.
The File Storage service transparently manages storage provisioning as data storage needs increase from the order of bytes to exabytes.
Large compute clusters of thousands of instances can use the File Storage service for high-performance shared storage.
Oracle Cloud Infrastructure File Storage uses 5-way replicated storage, in different fault domains, to provide redundancy for resilient data protection.
To read about example use cases, and for more information about the Oracle Cloud Infrastructure File Storage service, see the Oracle Cloud Infrastructure documentation

The following sections provide examples to illustrate how to use each of the storage services to set up persistent storage for Kubernetes applications in Oracle Cloud Native Environment on Oracle Cloud Infrastructure instances.

Creating Block Storage

This section contains a basic test to verify you can create Oracle Cloud Infrastructure block storage to provide persistent storage to applications running on Kubernetes.

To create a test application to use Oracle Cloud Infrastructure block storage:

1. Create a Kubernetes PersistentVolumeClaim file. On a control plane node, create a file named `pvc.yaml`. Copy the following into the file.

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: myoci-pvc
spec:
```

```
 accessModes:
  - ReadWriteOnce
 storageClassName: oci-bv
 resources:
  requests:
    storage: 50Gi
```

Note that the `accessModes` setting for Oracle Cloud Infrastructure storage must be `ReadWriteOnce`. The minimum Oracle Cloud Infrastructure block size is 50Gi.

2. Create the Kubernetes PersistentVolumeClaim.

```
kubectl apply -f pvc.yaml
```

3. You can see the PersistentVolumeClaim is created using the `kubectl get pvc` command:

```
kubectl get pvc
```

The output looks similar to:

```
NAME        STATUS    VOLUME   CAPACITY   ACCESS MODES
STORAGECLASS   AGE
myoci-pvc   Pending                                      oci-
bv         15s
```

The `STATUS` is `Pending` and means the claim is waiting for an application to claim it.

You can get more details about the PersistentVolumeClaim using the `kubectl describe pvc` command. For example:

```
kubectl describe pvc myoci-pvc
```

The output looks similar to:

```
Name:          myoci-pvc
Namespace:     default
StorageClass:  oci-bv
Status:        Pending
Volume:
Labels:        <none>
Annotations:   <none>
Finalizers:    [kubernetes.io/pvc-protection]
Capacity:
Access Modes:
VolumeMode:    Filesystem
Used By:       <none>
Events:
  Type    Reason                Age
From                        ...
  ----    ------                ----
----
```

```
  Normal  WaitForFirstConsumer  2m18s (x26 over 8m29s)  persistentvolume-
controller  ...
```

4. Create a Kubernetes application that uses the PersistentVolumeClaim. Create a file named `nginx.yaml` and copy the following into the file.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    run: mynginx
  name: mynginx
spec:
  replicas: 1
  selector:
    matchLabels:
      run: mynginx
  template:
    metadata:
      labels:
        run: mynginx
    spec:
      containers:
      - image: container-registry.oracle.com/olcne/nginx:1.17.7
        name: mynginx
        ports:
        - containerPort: 80
        volumeMounts:
        - name: nginx-pvc
          mountPath: /usr/share/nginx/html
      volumes:
      - name: nginx-pvc
        persistentVolumeClaim:
          claimName: myoci-pvc
```

5. Start the application:

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

6. You can see the application is running using the `kubectl get deployment` command:

```
kubectl get deployment
```

   The output looks similar to:

```
NAME      READY   UP-TO-DATE   AVAILABLE   AGE
mynginx   1/1     1            1           63s
```

7. You can see the application is using the PersistentVolumeClaim to provide persistent storage on Oracle Cloud Infrastructure using the `kubectl describe deployment` command:

```
kubectl describe deployment mynginx
```

The output looks similar to:

```
...
Pod Template:
  Labels:  run=mynginx
  Containers:
   mynginx:
    Image:        container-registry.oracle.com/olcne/nginx:1.17.7
    Port:         80/TCP
    Host Port:    0/TCP
    Environment:  <none>
    Mounts:
      /usr/share/nginx/html from nginx-pvc (rw)
  Volumes:
   nginx-pvc:
    Type:       PersistentVolumeClaim (a reference to a
PersistentVolumeClaim in the same namespace)
    ClaimName:  myoci-pvc
    ReadOnly:   false
...
```

Note the `ClaimName` is `myoci-pvc`, which is the name of the PersistentVolumeClaim created earlier.

You can see the PersistentVolumeClaim is now bound to this application using the `kubectl get pvc` command:

```
kubectl get pvc
```

The output looks similar to:

```
NAME        STATUS   VOLUME             CAPACITY   ACCESS MODES
STORAGECLASS   AGE
myoci-pvc   Bound    csi-84175067-...   50Gi       RWO            oci-
bv         1m
```

> **Tip:**
>
> If you sign in to Oracle Cloud Infrastructure, you can see a block volume created with the name listed in the `VOLUME` column. The block volume is attached to the compute instance on which the Kubernetes application is running.

8. You can delete the test application using:

```
kubectl delete deployment mynginx
```

9. You can delete the PersistentVolumeClaim using:

```
kubectl delete pvc myoci-pvc
```

The storage is deleted.

> **Tip:**
>
> If you sign in to Oracle Cloud Infrastructure, you can see the block volume is terminated.

Creating File Storage

This section contains a basic example to verify you can use an Oracle Cloud Infrastructure File Storage file system to provide persistent file storage to applications running on Kubernetes. The example involves:

1. Creating a File Storage file system in Oracle Cloud Infrastructure.
2. Using YAML files to create a PersistentVolume and a PersistentVolumeClaim for the File Storage file system.
3. Using a YAML file to create a pod with a mount to the volume to write a test file in the File Storage file system.

Setting up a File System

Create a File Storage file system in a Virtual Cloud Network (VCN) in Oracle Cloud Infrastructure.

> **Tip:**
>
> For a step-by-step guide on provisioning a File System in Oracle Cloud Infrastructure, see the Oracle Luna Lab Provision Persistent Volumes Using File Storage Service on Oracle Cloud Native Environment.

To create a file system:

1. Configure a VCN for the File Storage file system. The precise configuration of the VCN varies. For example scenarios, see the Oracle Cloud Infrastructure documentation.
2. Create a File Storage file system. For more information, see the Oracle Cloud Infrastructure documentation.
3. Create a File Storage mount target to enable network access to the file system. For more information, see the Oracle Cloud Infrastructure documentation.
4. In Oracle Cloud Infrastructure, find and make a note of the following file system attributes. These are needed when you create YAML files or mount the file system in this example:
   - The OCID of the file system.
   - The export path of the file system. This example assumes you're setting the export path to `/my-fss-export`.
   - The mount commands. These commands are provided on the Mount Commands page of the File System. You use these commands later in the example when mounting the exported file system.
   - The IP address of the mount target.

Using a File System

To create a test application to use an Oracle Cloud Infrastructure File Storage file system:

1. Create a PersistentVolume file. On a control plane node, create a file called `fss-pv.yaml` with the following contents:

   ```
   apiVersion: v1
   kind: PersistentVolume
   metadata:
     name: fss-pv
   spec:
     capacity:
       storage: 50Gi
     volumeMode: Filesystem
     accessModes:
       - ReadWriteMany
     persistentVolumeReclaimPolicy: Retain
     csi:
       driver: fss.csi.oraclecloud.com
       volumeHandle:
   <filesystem_OCID>:<mount_target_IP>:<mount_target_export_path>
   ```

   The `accessModes` for the file system must be `ReadWriteMany`.

   The `storage` option is a Kubernetes requirement and must be included. The File Storage service ignores this value and creates a new file system with a default size, regardless of the value you specify for `storage`.

   The `volumeHandle` consists of a colon separated list of File Storage file system attributes, noted in an earlier step. For example:

   ```
   ocid1.filesystem.oc1.iad.aaaa...:10.0.0.200:/my-fss-export
   ```

2. Create the PersistentVolume.

   ```
   kubectl apply -f fss-pv.yaml
   ```

3. Get a list of the PersistentVolumes.

   ```
   kubectl get pv
   ```

   The output looks similar to:

   ```
   NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
   CLAIM   STORAGECLASS   REASON   AGE
   fss-pv   50Gi       RWX            Retain
   Available                                   47s
   ```

4. Create a PersistentVolumeClaim file. Create a file called `fss-pvc.yaml`. Copy the following into the file:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: fss-pvc
spec:
  accessModes:
    - ReadWriteMany
  storageClassName: ""
  resources:
    requests:
      storage: 5Gi
  volumeName: fss-pv
```

   The `accessModes` for the file system must be `ReadWriteMany`.

   The `storage` option is a Kubernetes requirement and must be included. The File Storage service ignores this value and creates a new file system with a default size, regardless of the value you specify for `storage`.

   The `volumeName` is set to the name attribute in the PersistentVolume file `fss-pv.yaml`.

5. Create the PersistentVolumeClaim.

```
kubectl apply -f fss-pvc.yaml
```

6. Get a list of PersistentVolumeClaims.

```
kubectl get pvc
```

   The output looks similar to:

```
NAME      STATUS   VOLUME   CAPACITY   ACCESS MODES   STORAGECLASS   AGE
fss-pvc   Bound    fss-pv   50Gi       RWX                           33s
```

7. Create a pod manifest file. Create a file called `fss-pod.yaml`. Copy the following into the file:

```
apiVersion: v1
kind: Pod
metadata:
  name: app
spec:
  containers:
  - name: app
    image: container-registry.oracle.com/os/oraclelinux:9-slim
    command: ["/bin/sh"]
    args: ["-c", "while true; do echo $(date -u) >> /data/out.txt; sleep
5; done"]
    volumeMounts:
    - name: persistent-storage
      mountPath: /data
```

```
  volumes:
  - name: persistent-storage
    persistentVolumeClaim:
      claimName: fss-pvc
```

The `image` option specifies the registry location to an Oracle Linux container image.

The `mountPath` option specifies the directory mounted to the persistent storage.

The `command` option specifies a command to write to file `out.txt` in the `/data` directory.

8. Create the pod.

```
kubectl apply -f fss-pod.yaml
```

9. Get a list of the pods.

```
kubectl get pods
```

The output looks similar to:

```
NAME   READY   STATUS    RESTARTS   AGE
app    1/1     Running   0          21s
```

10. Open a shell to the container within the pod:

```
kubectl exec -i -t app --container app -- /bin/bash
```

11. From the container's shell, confirm the container is writing to the `/data/out.txt` file by using the `tail` command:

```
tail -f /data/out.txt
```

The date and time output of the `date -u` command is listed to the terminal and written to the `/data/out.txt` file on the container. Exit the `tail` program by using `CTRL+C`.

12. Exit the container shell using the `exit` command:

```
exit
```

13. Verify that the `/data` directory within the container is mounted to the file system export path. This step uses the mount commands noted when you created the file system.

    a. On the control plane node, install the NFS client.

```
sudo dnf install nfs-utils
```

b. Create and mount a local directory to the file system's export path.

```
sudo mkdir -p /mnt/my-fss-export
sudo mount 10.0.0.200:/my-fss-export /mnt/my-fss-export
```

Replace `10.0.0.200` with the IP address of the mount target.

c. Confirm the `output.txt` file the container is writing to is in the file system's export path. You can use the `tail` command for this:

```
sudo tail -f /mnt/my-fss-export/out.txt
```

The date and time being written to the file by the pod is listed. Exit the `tail` program by using `CTRL+C`.

14. Delete the pod. If you don't see the command prompt after the pod is deleted you can use the `CTRL+C` key combination to get it back:

```
kubectl delete pod app
```

15. Delete the PersistentVolumeClaim:

```
kubectl delete pvc fss-pvc
```

16. Confirm that the status of the PersistentVolume is `Released`:

```
kubectl get pv
```

The output looks similar to:

```
NAME     CAPACITY   ACCESS MODES   RECLAIM POLICY   STATUS
CLAIM             STORAGECLASS   REASON   AGE
fss-pv   50Gi       RWX            Retain           Released   default/
fss-pvc                           57m
```

17. Delete the PersistentVolume:

```
kubectl delete pv fss-pv
```

18. Confirm the file is still present:

```
ls -l  /mnt/my-fss-export/out.txt
```

19. Remove the mount:

```
sudo umount /mnt/my-fss-export
```

20. Confirm the `out.txt` file is no longer available and the mount is removed:

```
ls -l /mnt/my-fss-export
```

21. Confirm the `/mnt/my-fss-export` is no longer mounted:

```
mount | grep my-fss-export
```

4

Using the Oracle Cloud Infrastructure Load Balancer

This chapter discusses how to 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.

Introduction to Oracle Cloud Infrastructure Load Balancers

The Oracle Cloud Infrastructure Flexible Network Load Balancing service (Oracle Cloud Infrastructure load balancer) provides automated traffic distribution from one entry point to many 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 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.

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, 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 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 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
```

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

```
kubectl get deployments.apps
```

The output looks similar to:

```
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
```

The output looks similar to:

```
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 might 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 Oracle Cloud Infrastructure 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
```

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>
```

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

```
kubectl delete svc nginx-service
```

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

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

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

5

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.