Oracle® Linux Cloud Native Environment

Container Orchestration

F22533-06
August 2020

Oracle Legal Notices

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

Table of Contents

Preface

This book describes how to use Kubernetes, which is an implementation of the open-source, containerized application management platform from the upstream Kubernetes release. Oracle provides additional tools, testing and support to deliver this technology with confidence. Kubernetes integrates with container products to handle more complex deployments where clustering may be used to improve the scalability, performance and availability of containerized applications. Detail is provided on the advanced features of Kubernetes and how it can be installed, configured and used as a component of Oracle Linux Cloud Native Environment.

This document describes functionality and usage available in the most current release of the product.

Document generated on: 2020-08-07 (revision: 733)

Audience

This document is intended for administrators who need to use Kubernetes in an Oracle Linux Cloud Native Environment. It is assumed that readers are familiar with web and virtualization technologies and have a general understanding of the Linux operating system.

Related Documents

The documentation for this product is available at:

https://docs.oracle.com/en/operating-systems/olcne/

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

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.

Chapter 1 Introduction to Kubernetes

Kubernetes is an open-source system for automating the deployment, scaling and management of containerized applications. Primarily, Kubernetes provides the tools to easily create a cluster of systems across which containerized applications can be deployed and scaled as required.

The Kubernetes project is maintained at:

https://kubernetes.io/

Kubernetes is fully tested on Oracle Linux 7 and includes additional tools developed at Oracle to ease configuration and deployment of a Kubernetes cluster.

For more information on Kubernetes releases, hardware and software requirements, new and notable features, and known issues, see *Release Notes*.

1.1 Kubernetes Components

You are likely to encounter the following common components when you start working with Kubernetes on Oracle Linux. The descriptions provided are brief, and largely intended to help provide a glossary of terms and an overview of the architecture of a typical Kubernetes environment. Upstream documentation can be found at:

https://kubernetes.io/docs/concepts/

1.1.1 Nodes

Kubernetes Node architecture is described in detail at:

https://kubernetes.io/docs/concepts/architecture/nodes/

1.1.1.1 Master Node

The master node is responsible for cluster management and for providing the API that is used to configure and manage resources within the Kubernetes cluster. Kubernetes master node components can be run within Kubernetes itself, as a set of containers within a dedicated pod. These components can be replicated to provide high availability (HA) Master Node functionality.

The following components are required for a master node:

- **API Server** (`kube-apiserver`): The Kubernetes REST API is exposed by the API Server. This component processes and validates operations and then updates information in the Cluster State Store to trigger operations on the worker nodes. The API is also the gateway to the cluster.
- **Cluster State Store** (`etcd`): Configuration data relating to the cluster state is stored in the Cluster State Store, which can roll out changes to the coordinating components like the Controller Manager and the Scheduler. It is essential to have a backup plan in place for the data stored in this component of your cluster.
- **Cluster Controller Manager** (`kube-controller-manager`): This manager is used to perform many of the cluster-level functions, as well as application management, based on input from the Cluster State Store and the API Server.
- **Scheduler** (`kube-scheduler`): The Scheduler handles automatically determining where containers should be run by monitoring availability of resources, quality of service and affinity and anti-affinity specifications.

The master node is also usually configured as a worker node within the cluster. Therefore, the master node also runs the standard node services: the kubelet service, the container runtime and the kube proxy service. Note that it is possible to taint a node to prevent workloads from running on an inappropriate node. The `kubeadm` utility automatically taints the master node so that no other workloads or containers can run on this node. This helps to ensure that the master node is never placed under any unnecessary load and that backup and restore of the master node for the cluster is simplified.

If the master node becomes unavailable for a period, cluster functionality is suspended, but the worker nodes continue to run container applications without interruption.

For single node clusters, when the master node is offline, the API is unavailable, so the environment is unable to respond to node failures and there is no way to perform new operations like creating new resources or editing or moving existing resources.

A high availability cluster with multiple master nodes ensures that more requests for master node functionality can be handled, and with the assistance of master replica nodes, uptime is significantly improved.

1.1.1.2 Master Replica Nodes

Master replica nodes are responsible for duplicating the functionality and data contained on master nodes within a Kubernetes cluster configured for high availability. To benefit from increased uptime and resilience, you can host master replica nodes in different zones, and configure them to load balance for your Kubernetes cluster.

Replica nodes are designed to mirror the master node configuration and the current cluster state in real time so that if the master nodes become unavailable the Kubernetes cluster can fail over to the replica nodes automatically whenever they are needed. In the event that a master node fails, the API continues to be available, the cluster can respond automatically to other node failures and you can still perform regular operations for creating and editing existing resources within the cluster.

1.1.1.3 Worker Nodes

Worker nodes within the Kubernetes cluster are used to run containerized applications and handle networking to ensure that traffic between applications across the cluster and from outside of the cluster can be properly facilitated. The worker nodes perform any actions triggered via the Kubernetes API, which runs on the master node.

All nodes within a Kubernetes cluster must run the following services:

- **Kubelet Service**: The agent that allows each worker node to communicate with the API Server running on the master node. This agent is also responsible for setting up pod requirements, such as mounting volumes, starting containers and reporting status.
- **Container Runtime**: An environment where containers can be run. In this release, the container runtimes are either runC or Kata Containers. For more information about the container runtimes, see *Container Runtimes*.
- **Kube Proxy Service**: A service that programs rules to handle port forwarding and IP redirects to ensure that network traffic from outside the pod network can be transparently proxied to the pods in a service.

In all cases, these services are run from `systemd` as inter-dependent daemons.

1.1.2 Pods

Kubernetes introduces the concept of "pods", which are groupings of one or more containers and their shared storage, and any specific options on how these should be run together. Pods are used for tightly

coupled applications that would typically run on the same logical host and which may require access to the same system resources. Typically, containers in a pod share the same network and memory space and can access shared volumes for storage. These shared resources allow the containers in a pod to communicate internally in a seamless way as if they were installed on a single logical host.

You can easily create or destroy pods as a set of containers. This makes it possible to do rolling updates to an application by controlling the scaling of the deployment. It also allows you to scale up or down easily by creating or removing replica pods. For more information on pods, see the upstream documentation at:

https://kubernetes.io/docs/concepts/workloads/pods/pod/

1.1.3 ReplicaSet, Deployment, StatefulSet Controllers

Kubernetes provides a variety of controllers that you can use to define how pods are set up and deployed within the Kubernetes cluster. These controllers can be used to group pods together according to their runtime needs and define pod replication and pod start up ordering.

You can define a set of pods that should be replicated with a *ReplicaSet*. This allows you to define the exact configuration for each of the pods in the group and which resources they should have access to. Using ReplicaSets not only caters to the easy scaling and rescheduling of an application, but also allows you to perform rolling or multi-track updates to an application. For more information on ReplicaSets, see the upstream documentation at:

https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/

You can use a *Deployment* to manage pods and *ReplicaSet*s. *Deployment*s are useful when you need to roll out changes to ReplicaSets. By using a *Deployment* to manage a *ReplicaSet*, you can easily rollback to an earlier *Deployment* revision. A *Deployment* allows you to create a newer revision of a *ReplicaSet* and then migrate existing pods from a previous *ReplicaSet* into the new revision. The *Deployment* can then manage the cleanup of older unused *ReplicaSet*s. For more information on Deployments, see the upstream documentation at:

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

You can use *StatefulSet*s to create pods that guarantee start up order and unique identifiers, which are then used to ensure that the pod maintains its identity across the lifecycle of the *StatefulSet*. This feature makes it possible to run stateful applications within Kubernetes, as typical persistent components such as storage and networking are guaranteed. Furthermore, when you create pods they are always created in the same order and allocated identifiers that are applied to host names and the internal cluster DNS. Those identifiers ensure there are stable and predictable network identities for pods in the environment. For more information on StatefulSets, see the upstream documentation at:

https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/

1.1.4 Services

You can use services to expose access to one or more mutually interchangeable pods. Since pods can be replicated for rolling updates and for scalability, clients accessing an application must be directed to a pod running the correct application. Pods may also need access to applications outside of Kubernetes. In either case, you can define a service to make access to these facilities transparent, even if the actual backend changes.

Typically, services consist of port and IP mappings. How services function in network space is defined by the service type when it is created.

The default service type is the `ClusterIP`, and you can use this to expose the service on the internal IP of the cluster. This option makes the service only reachable from within the cluster. Therefore, you should

use this option to expose services for applications that need to be able to access each other from within the cluster.

Frequently, clients outside of the Kubernetes cluster may need access to services within the cluster. You can achieve this by creating a `NodePort` service type. This service type enables you to take advantage of the *Kube Proxy* service that runs on every worker node and reroute traffic to a `ClusterIP`, which is created automatically along with the `NodePort` service. The service is exposed on each node IP at a static port, called the `NodePort`. The Kube Proxy routes traffic destined to the `NodePort` into the cluster to be serviced by a pod running inside the cluster. This means that if a `NodePort` service is running in the cluster, it can be accessed via any node in the cluster, regardless of where the pod is running.

Building on top of these service types, the `LoadBalancer` service type makes it possible for you to expose the service externally by using a cloud provider's load balancer. This allows an external load balancer to handle redirecting traffic to pods directly in the cluster via the Kube Proxy. A `NodePort` service and a `ClusterIP` service are automatically created when you set up the `LoadBalancer` service.

Important

As you add services for different pods, you must ensure that your network is properly configured to allow traffic to flow for each service declaration. If you create a `NodePort` or `LoadBalancer` service, any of the ports exposed must also be accessible through any firewalls that are in place.

If you are using Oracle Cloud Infrastructure, you must add ingress rules to the security lists for the Virtual Cloud Network (VCN) for your compute instances connections. Each rule should allow access to the port that you have exposed for a service.

Equally, if you are running `firewalld` on any of your nodes, you must ensure that you add rules to allow traffic for the external facing ports of the services that you create.

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

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

1.1.5 Volumes

In Kubernetes, a *volume* is storage that persists across the containers within a pod for the lifespan of the pod itself. When a container within the pod is restarted, the data in the Kubernetes volume is preserved. Furthermore, Kubernetes volumes can be shared across containers within the pod, providing a file store that different containers can access locally.

Kubernetes supports a variety of volume types that define how the data is stored and how persistent it is, which are described in detail in the upstream documentation at:

https://kubernetes.io/docs/concepts/storage/volumes/

Kubernetes volumes typically have a lifetime that matches the lifetime of the pod, and data in a volume persists for as long as the pod using that volume exists. Containers can be restarted within the pod, but the data remains persistent. If the pod is destroyed, the data is usually destroyed with it.

In some cases, you may require even more persistence to ensure the lifecycle of the volume is decoupled from the lifecycle of the pod. Kubernetes introduces the concepts of the *PersistentVolume* and the *PersistentVolumeClaim*. *PersistentVolume*s are similar to *Volume*s except that they exist independently of a pod. They define how to access a storage resource type, such as NFS or iSCSI. You can configure

a *PersistentVolumeClaim* to make use of the resources available in a *PersistentVolume*, and the *PersistentVolumeClaim* will specify the quota and access modes that should be applied to the resource for a consumer. A pod you have created can then make use of the *PersistentVolumeClaim* to gain access to these resources with the appropriate access modes and size restrictions applied.

For more information about *PersistentVolumes*, see the upstream documentation at:

https://kubernetes.io/docs/concepts/storage/persistent-volumes/

1.1.6 Namespaces

Kubernetes implements and maintains strong separation of resources through the use of namespaces. Namespaces effectively run as virtual clusters backed by the same physical cluster and are intended for use in environments where Kubernetes resources must be shared across use cases.

Kubernetes takes advantage of namespaces to separate cluster management and specific Kubernetes controls from any other user-specific configuration. Therefore, all of the pods and services specific to the Kubernetes system are found within the `kube-system` namespace. A `default` namespace is also created to run all other deployments for which no namespace has been set.

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

https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/

1.2 About CRI-O

When you deploy Kubernetes worker nodes, CRI-O is also deployed. CRI-O is an implementation of the Kubernetes Container Runtime Interface (CRI) to enable using Open Container Initiative (OCI) compatible runtimes. It is a lightweight alternative to using Docker as the runtime for Kubernetes. CRI-O allows Kubernetes to use any OCI-compliant runtime as the container runtime for running pods.

CRI-O delegates containers to run on appropriate nodes, based on the configuration set in pod files. *Privileged* pods can be run using the runC runtime engine (`runc`), and *unprivileged* pods can be run using the Kata Containers runtime engine (`kata-runtime`). Defining whether containers are trusted or untrusted is set in the Kubernetes pod or deployment file.

For information on how to set the container runtime, see *Container Runtimes*.

Chapter 2 Using Kubernetes

This chapter describes how to get started using Kubernetes to deploy, maintain and scale your containerized applications.

Kubernetes is deployed using the Oracle Linux Cloud Native Environment Platform Command-Line Interface. For information on using the Platform CLI to deploy Kubernetes, see *Getting Started*.

2.1 About Runtime Engines

`runc` is the default runtime engine when you create containers. You can also use the `kata-runtime` runtime engine to create Kata containers. For information on Kata containers and how to create them, see *Container Runtimes*.

2.2 Setting up the kubectl Command

The `kubectl` utility is a command line tool that interfaces with the Kubernetes API server to run commands against the Kubernetes cluster. The `kubectl` command is typically run on the *master* node of the cluster, although you can also use the on Oracle Linux Cloud Native Environment *operator* node. The `kubectl` utility effectively grants full administrative rights to the cluster and all of the nodes in the cluster.

This section discusses setting up the `kubectl` command to access a Kubernetes cluster from either a master node or an operator node.

2.2.1 Setting up the kubectl Command on a Master Node

To use the `kubectl` command as a regular user, perform the following steps on the master node.

To setup `kubectl` on a master node:

1. Create the `.kube` subdirectory in your home directory:

   ```
   $ mkdir -p $HOME/.kube
   ```

2. Create a copy of the Kubernetes `admin.conf` file in the `.kube` directory:

   ```
   $ sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
   ```

3. Change the ownership of the file to match your regular user profile:

   ```
   $ sudo chown $(id -u):$(id -g) $HOME/.kube/config
   ```

4. Export the path to the file for the `KUBECONFIG` environment variable:

   ```
   $ export KUBECONFIG=$HOME/.kube/config
   ```

   To permanently set this environment variable, add it to your `.bashrc` file.

   ```
   $ echo 'export KUBECONFIG=$HOME/.kube/config' >> $HOME/.bashrc
   ```

5. Verify that you can use the `kubectl` command.

   Kubernetes runs many of its services to manage the cluster configuration as containers running as Kubernetes pods, which can be viewed by running the following command on a master node:

   ```
   $ kubectl get pods -n kube-system
   NAME                                READY   STATUS    RESTARTS   AGE
   coredns-5bc65d7f4b-qzfcc            1/1     Running   0          23h
   ```

```
coredns-5bc65d7f4b-z64f2                1/1     Running   0          23h
etcd-master1.example.com                1/1     Running   0          23h
kube-apiserver-master1.example.com      1/1     Running   0          23h
kube-controller-master1.example.com     1/1     Running   0          23h
kube-flannel-ds-2sjbx                   1/1     Running   0          23h
kube-flannel-ds-njg9r                   1/1     Running   0          23h
kube-proxy-m2rt2                        1/1     Running   0          23h
kube-proxy-tbkxd                        1/1     Running   0          23h
kube-scheduler-master1.example.com      1/1     Running   0          23h
kubernetes-dashboard-7646bf6898-d6x2m   1/1     Running   0          23h
```

2.2.2 Setting up the kubectl Command on the Operator Node

Oracle Linux Cloud Native Environment allows you to create multiple environments from the operator node. With this in mind, it is recommended that you use the `kubectl` command on a **master** node in the Kubernetes cluster. If you use the `kubectl` command from the operator node, and you have multiple environments deployed, you may inadvertently manage an unexpected Kubernetes cluster. If you do want to set up the `kubectl` command to run it the operator node, you need to configure it.

The `kubectl` command is not set up by default to connect to Kubernetes from the operator node. To set up the `kubectl` command on the operator node, create a local copy of the Kubernetes configuration file, and use that to connect to the cluster.

To use the `kubectl` command as a regular user, perform the following steps on the operator node.

To setup `kubectl` on an operator node:

1. Get the Kubernetes configuration and copy it to a local file on the operator node. Use the `olcnectl module property get` command to get the Kubernetes configuration from the `kubecfg` property of the `kubernetes` module. For example:

   ```
   $ olcnectl --api-server 127.0.0.1:8091 module property get \
     --environment-name myenvironment --name mycluster \
     --property kubecfg | base64 -d > kubeconfig.yaml
   ```

2. You can use the `kubeconfig.yaml` file directly when running `kubectl` commands using the `--kubeconfig` option. For example:

   ```
   $ kubectl get pods -n kube-system --kubeconfig kubeconfig.yaml
   ```

3. You can also save the Kubernetes configuration so you do not need to use the `--kubeconfig` option. Create the `.kube` subdirectory in your home directory:

   ```
   $ mkdir -p $HOME/.kube
   ```

4. Copy the Kubernetes `kubeconfig.yaml` file to the `.kube` directory:

   ```
   $ cp kubeconfig.yaml $HOME/.kube/config
   ```

5. Export the path to the file for the `KUBECONFIG` environment variable:

   ```
   $ export KUBECONFIG=$HOME/.kube/config
   ```

   To permanently set this environment variable, add it to your `.bashrc` file.

   ```
   $ echo 'export KUBECONFIG=$HOME/.kube/config' >> $HOME/.bashrc
   ```

6. Verify that you can use the `kubectl` command.

   Kubernetes runs many of its services to manage the cluster configuration as containers running as Kubernetes pods, which can be viewed by running the following command on a master node:

```
$ kubectl get pods -n kube-system
NAME                                    READY   STATUS    RESTARTS   AGE
coredns-5bc65d7f4b-qzfcc                1/1     Running   0          23h
coredns-5bc65d7f4b-z64f2                1/1     Running   0          23h
etcd-master1.example.com                1/1     Running   0          23h
kube-apiserver-master1.example.com      1/1     Running   0          23h
kube-controller-master1.example.com     1/1     Running   0          23h
kube-flannel-ds-2sjbx                   1/1     Running   0          23h
kube-flannel-ds-njg9r                   1/1     Running   0          23h
kube-proxy-m2rt2                        1/1     Running   0          23h
kube-proxy-tbkxd                        1/1     Running   0          23h
kube-scheduler-master1.example.com      1/1     Running   0          23h
kubernetes-dashboard-7646bf6898-d6x2m   1/1     Running   0          23h
```

2.3 Using the kubectl Command

In this section, we describe basic usage of the `kubectl` tool to get you started creating and managing pods and services within your environment.

The `kubectl` utility is fully documented in the upstream documentation at:

https://kubernetes.io/docs/reference/kubectl/overview/

2.3.1 Getting Information about Nodes

To get a listing of all of the nodes in a cluster and the status of each node, use the `kubectl get` command. This command can be used to obtain listings of any kind of resource that Kubernetes supports. In this case, the `nodes` resource:

```
$ kubectl get nodes
NAME                 STATUS    ROLES   AGE      VERSION
master.example.com   Ready     master  1h       v1.17.x+x.x.x.el7
worker1.example.com  Ready     <none>  1h       v1.17.x+x.x.x.el7
worker2.example.com  Ready     <none>  1h       v1.17.x+x.x.x.el7
```

You can get more detailed information about any resource using the `kubectl describe` command. If you specify the name of the resource, the output is limited to information about that resource alone; otherwise, full details of all resources are also printed to screen:

```
$ kubectl describe nodes worker1.example.com
Name:               worker1.example.com1
Roles:              <none>
Labels:             beta.kubernetes.io/arch=amd64
                    beta.kubernetes.io/os=linux
                    kubernetes.io/arch=amd64
                    kubernetes.io/hostname=worker1.example.com
                    kubernetes.io/os=linux
Annotations:        flannel.alpha.coreos.com/backend-data: {"VtepMAC":"fe:78:5f:ea:7c:c0"}
                    flannel.alpha.coreos.com/backend-type: vxlan
                    flannel.alpha.coreos.com/kube-subnet-manager: true
                    flannel.alpha.coreos.com/public-ip: 192.0.2.11
                    kubeadm.alpha.kubernetes.io/cri-socket: /var/run/crio/crio.sock
                    node.alpha.kubernetes.io/ttl: 0
                    volumes.kubernetes.io/controller-managed-attach-detach: true
...
```

2.3.2 Running an Application in a Pod

To create a pod with a single running container, you can use the `kubectl create` command:

```
$ kubectl create deployment --image nginx hello-world
deployment.apps/hello-world created
```

Substitute *hello-world* with a name for your deployment. Your pods are named by using the deployment name as a prefix. Substitute *nginx* with a container image.

Tip

Deployment, pod and service names conform to a requirement to match a DNS-1123 label. These must consist of lower case alphanumeric characters or `-`, and must start and end with an alphanumeric character. The regular expression that is used to validate names is '`[a-z0-9]([-a-z0-9]*[a-z0-9])?`'. If you use a name for your deployment that does not validate, an error is returned.

There are many additional optional parameters that can be used when you run a new application within Kubernetes. For instance, at run time, you can specify how many replica pods should be started, or you might apply a label to the deployment to make it easier to identify pod components. To see a full list of options available to you, run `kubectl run --help`.

To check that your new application deployment has created one or more pods, use the `kubectl get pods` command:

```
$ kubectl get pods
NAME                           READY     STATUS    RESTARTS   AGE
hello-world-5f55779987-wd857   1/1       Running   0          1m
```

Use `kubectl describe` to show a more detailed view of your pods, including which containers are running and what image they are based on, as well as which node is currently hosting the pod:

```
$ kubectl describe pods
Name:               hello-world-5f55779987-wd857
Namespace:          default
Priority:           0
PriorityClassName:  <none>
Node:               worker1.example.com/192.0.2.11
Start Time:         Fri, 16 Aug 2019 08:48:33 +0100
Labels:             app=hello-world
                    pod-template-hash=5f55779987
Annotations:        <none>
Status:             Running
IP:                 10.244.1.3
Controlled By:      ReplicaSet/hello-world-5f55779987
Containers:
  nginx:
    Container ID:   cri-o://417b4b59f7005eb4b1754a1627e01f957e931c0cf24f1780cd94fa9949be1d31
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:5d32f60db294b5deb55d078cd4feb410ad88e6fe7...
    Port:           <none>
    Host Port:      <none>
    State:          Running
      Started:      Mon, 10 Dec 2018 08:25:25 -0800
    Ready:          True
    Restart Count:  0
    Environment:    <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-s8wj4 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-s8wj4:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-s8wj4
```

```
    Optional:    false
QoS Class:       BestEffort
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute for 300s
                 node.kubernetes.io/unreachable:NoExecute for 300s
Events:
....
```

2.3.3 Scaling a Pod Deployment

To change the number of instances of the same pod that you are running, you can use the `kubectl scale deployment` command:

```
$ kubectl scale deployment --replicas=3 hello-world
deployment.apps/hello-world scaled
```

You can check that the number of pod instances has been scaled appropriately:

```
$ kubectl get pods
NAME                           READY     STATUS    RESTARTS   AGE
hello-world-5f55779987-tswmg   1/1       Running   0          18s
hello-world-5f55779987-v8w5h   1/1       Running   0          26m
hello-world-5f55779987-wd857   1/1       Running   0          18s
```

2.3.4 Exposing a Service Object for an Application

Typically, while many applications may only need to communicate internally within a pod, or even across pods, you may need to expose your application externally so that clients outside of the Kubernetes cluster can interface with the application. You can do this by creating a service definition for the deployment.

To expose a deployment using a service object, you must define the service type that should be used. If you are not using a cloud-based load balancing service, you can set the service type to `NodePort`. The `NodePort` service exposes the application running within the cluster on a dedicated port on the public IP address on all of the nodes within the cluster. Use the `kubectl expose deployment` to create a new service:

```
$ kubectl expose deployment hello-world --port 80 --type=LoadBalancer
service/hello-world exposed
```

Use `kubectl get services` to list the different services that the cluster is running, and to obtain the port information required to access the service:

```
$ kubectl get services
NAME          TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
hello-world   LoadBalancer   10.102.42.160   <pending>     80:31847/TCP   3s
kubernetes    ClusterIP      10.96.0.1       <none>        443/TCP        5h13m
```

In this example output, you can see that traffic to port 80 inside the cluster is mapped to the `NodePort` 31847. The external IP that can be used to access the service is listed as `<pending>`, meaning that if you connect to the external IP address for any of the nodes within the cluster on the port 31847, you are able access the service.

For the sake of the example in this guide, you can open a web browser to point at any of the nodes in the cluster, such as `http://worker1.example.com:31847/`, and it should display the NGINX demonstration application.

2.3.5 Deleting a Service or Deployment

Objects can be deleted easily within Kubernetes so that your environment can be cleaned. Use the `kubectl delete` command to remove an object.

To delete a service, specify the services object and the name of the service that you want to remove:

```
$ kubectl delete services hello-world
service "hello-world" deleted
```

To delete an entire deployment, and all of the pod replicas running for that deployment, specify the deployment object and the name that you used to create the deployment:

```
$ kubectl delete deployment hello-world
deployment.extensions "hello-world" deleted
```

2.3.6 Working With Namespaces

Namespaces can be used to further separate resource usage and to provide limited environments for particular use cases. By default, Kubernetes configures a namespace for Kubernetes system components and a standard namespace to be used for all other deployments for which no namespace is defined.

To view existing namespaces, use the `kubectl get namespaces` and `kubectl describe namespaces` commands.

The `kubectl` command only displays resources in the default namespace, unless you set the namespace specifically for a request. Therefore, if you need to view the pods specific to the Kubernetes system, you would use the `--namespace` option to set the namespace to `kube-system` for the request. For example, in a cluster with a single master node:

```
$ kubectl get pods --namespace=kube-system
NAME                                    READY   STATUS    RESTARTS   AGE
coredns-5bc65d7f4b-qzfcc                1/1     Running   0          23h
coredns-5bc65d7f4b-z64f2                1/1     Running   0          23h
etcd-master1.example.com                1/1     Running   0          23h
kube-apiserver-master1.example.com      1/1     Running   0          23h
kube-controller-master1.example.com     1/1     Running   0          23h
kube-flannel-ds-2sjbx                   1/1     Running   0          23h
kube-flannel-ds-njg9r                   1/1     Running   0          23h
kube-proxy-m2rt2                        1/1     Running   0          23h
kube-proxy-tbkxd                        1/1     Running   0          23h
kube-scheduler-master1.example.com      1/1     Running   0          23h
kubernetes-dashboard-7646bf6898-d6x2m   1/1     Running   0          23h
```

2.4 Using Deployment Files

To simplify the creation of pods and their related requirements, you can create a deployment file that define all of the elements that comprise the deployment. This deployment defines which images should be used to generate the containers within the pod, along with any runtime requirements, as well as Kubernetes networking and storage requirements in the form of services that should be configured and volumes that may need to be mounted.

Deployments are described in detail at:

https://kubernetes.io/docs/concepts/workloads/controllers/deployment/

Chapter 3 Accessing the Dashboard

The Kubernetes Dashboard container is created as part of the `kube-system` namespace. This provides an intuitive graphical user interface to Kubernetes that can be accessed using a standard web browser.

The Kubernetes Dashboard is described in the upstream Kubernetes documentation at:

https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/

This chapter shows you how to start and connect to the Dashboard.

3.1 Starting the Dashboard

To start the Dashboard, you can run a proxy service that allows traffic on the node where it is running to reach the internal pod where the Dashboard application is running. This is achieved by running the `kubectl proxy` service:

```
$ kubectl proxy
Starting to serve on 127.0.0.1:8001
```

The Dashboard is available on the node where the proxy is running for as long as the proxy runs. To exit the proxy, use **`Ctrl+C`**.

You can run this as a `systemd` service and enable it so that it is always available after subsequent reboots:

```
$ sudo systemctl enable --now kubectl-proxy
```

This `systemd` service requires that the `/etc/kubernetes/admin.conf` is present to run. If you want to change the port that is used for the proxy service, or you want to add other proxy configuration parameters, you can configure this by editing the `systemd` drop-in file at `/etc/systemd/system/kubectl-proxy.service.d/10-kubectl-proxy.conf`. You can get more information about the configuration options available for the `kubectl proxy` service by running:

```
$ kubectl proxy --help
```

3.2 Connecting to the Dashboard

To access the Dashboard, open a web browser on the node where the `kubectl proxy` service is running and navigate to:

http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/

To log in, you must authenticate using a token. For more information on authentication tokens, see the upstream documentation at:

https://github.com/kubernetes/dashboard/blob/master/docs/user/access-control/README.md

If you have not set up specific tokens for this purpose, you can use a token allocated to a service account, such as the `namespace-controller`. Run the following command to obtain the token value for the `namespace-controller`:

```
$ kubectl -n kube-system describe $(kubectl -n kube-system \
   get secret -n kube-system -o name | grep namespace) | grep token:
token:      eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJrdWJlcm5ldGVzL3NlcnZpY2VhY2Nvd\
            W50Iiwia3ViZXJuZXRlcy5pby9zZXJ2aWNlYWNjb3VudC9uYW1lc3BhY2UiOiJrdWJlLXN5c3RlbSI\
            sImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VjcmV0Lm5hbWUiOiJuYW1lc3BhY2UtY29ud\
```

```
            HJvbGxlci10b2tlbi1zeHB3ayIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFjY291bnQvc2VydmljZS1h\
            Y2NvdW50Lm5hbWUiOiJuYW1lc3BhY2UtY29udHJvbGxlciIsImt1YmVybmV0ZXMuaW8vc2VydmljZWFj\
            Y291bnQvc2VydmljZS1hY2NvdW50LnVpZCI6IjM4OTk1MWIyLWJlNDYtMTFlNy04ZGY2LTA4MDAyNzY\
            wOTVkNyIsInN1YiI6InN5c3RlbTpzZXJ2aWNlYWNjb3VudDprdWJlLXN5c3RlbTpuYW1lc3BhY2UtY2\
            9udHJvbGxlciJ9.aL-9sRGic_b7XW2eOsDfxn9QCCobBSU41J1hMbT5D-Z86iahl1mQnV60zEKOg-45\
            5pLO4aW_RSETxxCp8zwaNkbwoUF1rbi17FMR_zfhj9sfNKzHYO1tjYf0lN452k7_oCkJ7HR2mzCHmw-\
            AygILeO0NlIgjxH_2423Dfe8In9_nRLB_PzKvlEV5Lpmzg4IowEFhawRGib3R1o74mgIb3SPeMLEAAA
```

Copy and paste the entire value of the token into the token field on the log in page to authenticate.

3.3 Connecting to the Dashboard Remotely

If you need to access the Dashboard remotely, you can use SSH tunneling to do port forwarding from your localhost to the node running the `kubectl proxy` service. The easiest option is to use SSH tunneling to forward a port on your local system to the port configured for the `kubectl proxy` service on the node that you want to access. This method retains some security as the HTTP connection is encrypted by virtue of the SSH tunnel and authentication is handled by your SSH configuration. For example, on your local system run:

```
$ ssh -L 8001:127.0.0.1:8001 192.0.2.10
```

Substitute *`192.0.2.10`* with the IP address of the host where the `kubectl proxy` service is running. When the SSH connection is established, you can open a browser on your localhost and navigate to:

http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/

You should see the Dashboard log in screen for the remote Kubernetes cluster. Use the same token information to authenticate as if you were connecting to the Dashboard locally.

Chapter 4 Backing up and Restoring Kubernetes

This chapter discusses how to back up and restore the Kubernetes cluster in an Oracle Linux Cloud Native Environment.

4.1 Backing up Master Nodes

This section discusses backing up the Kubernetes master nodes in an Oracle Linux Cloud Native Environment.

Adopting a back up strategy to protect your Kubernetes cluster against master node failures is important, particularly for single node master clusters. High availability clusters with multiple master nodes also need a fallback plan if the resilience provided by the provided replication and failover functionality has been exceeded.

You do not need to bring down the cluster to perform a back up as part of your disaster recovery plan. On the operator node, use the `olcnectl module backup` command to back up all the key containers and manifests for all the master nodes in your cluster.

Important

Only the key containers required for the Kubernetes control plane are backed up. No application containers are backed up.

For example:

```
$ olcnectl module backup --environment-name myenvironment --name mycluster
```

The back up files are stored in the `/var/olcne/backups` directory on the operator node. The files are saved to a timestamped folder that follows the pattern:

`/var/olcne/backups/`*`environment-name`*`/kubernetes/`*`module-name/timestamp`*

You can interact with the directory and the files it contains just like any other:

```
$ sudo ls /var/olcne/backups/myenvironment/kubernetes/mycluster/20191007040013
master1.example.com.tar master2.example.com.tar master3.example.com.tar  etcd.tar
```

4.2 Restoring Master Nodes

This section discusses restoring Kubernetes nodes from back ups in an Oracle Linux Cloud Native Environment.

These restore steps are intended for use when one or more Kubernetes master clusters needs to be reconstructed as part of a planned disaster recovery scenario. Unless there is a total cluster failure you do not need to manually recover individual master nodes in a high availability cluster that is able to self-heal with replication and failover.

In order to restore a master node, you must have a pre-existing Oracle Linux Cloud Native Environment, and have deployed the Kubernetes module. You cannot restore to a non-existent environment.

To restore a master node:

1. Make sure the Platform Agent is running correctly on the replacement master nodes before proceeding:

   ```
   $ sudo systemctl status olcne-agent
   ```

2. On the operator node, use the `olcnectl module restore` command to restore the key containers and manifests for the master nodes in your cluster. For example:

```
$ olcnectl module restore --environment-name myenvironment --name mycluster
```

   The files from the latest timestamped folder from `/var/olcne/backups/environment-name/kubernetes/module-name/` are used to restore the cluster to its previous state.

   You may be prompted by the Platform CLI to perform additional set up steps on your master nodes to fulfil the prerequisite requirements. If that happens, follow the instructions and run the `olcnectl module restore` command again.

3. You can verify the restore operation was successful by using the `kubectl` command on a master node. For example:

```
$ kubectl get nodes
 NAME                 STATUS   ROLES    AGE     VERSION
master1.example.com   Ready    master   9m27s   v1.17.x+x.x.x.el7
worker1.example.com   Ready    <none>   8m53s   v1.17.x+x.x.x.el7

$ kubectl get pods -n kube-system
NAME                                    READY   STATUS    RESTARTS   AGE
coredns-5bc65d7f4b-qzfcc                1/1     Running   0          9m
coredns-5bc65d7f4b-z64f2                1/1     Running   0          9m
etcd-master1.example.com                1/1     Running   0          9m
kube-apiserver-master1.example.com      1/1     Running   0          9m
kube-controller-master1.example.com     1/1     Running   0          9m
kube-flannel-ds-2sjbx                   1/1     Running   0          9m
kube-flannel-ds-njg9r                   1/1     Running   0          9m
kube-proxy-m2rt2                        1/1     Running   0          9m
kube-proxy-tbkxd                        1/1     Running   0          9m
kube-scheduler-master1.example.com      1/1     Running   0          9m
kubernetes-dashboard-7646bf6898-d6x2m   1/1     Running   0          9m
```