Table of Contents

Preface ................................................................................................................................. v
1 About Oracle Container Services for use with Kubernetes ........................................... 1
   1.1 Release Information for Oracle Container Services for use with Kubernetes .......... 1
      1.1.1 New and Notable Features .............................................................................. 1
      1.1.2 Technical Preview ......................................................................................... 3
      1.1.3 Known Issues ............................................................................................... 4
   1.2 Kubernetes Components .......................................................................................... 4
      1.2.1 Nodes ........................................................................................................... 4
      1.2.2 Pods ............................................................................................................ 6
      1.2.3 ReplicaSet, Deployment, StatefulSet Controllers ........................................... 6
      1.2.4 Services ...................................................................................................... 7
      1.2.5 Volumes ...................................................................................................... 8
      1.2.6 Namespaces ............................................................................................... 8
2 Installing Oracle Container Services for use with Kubernetes ...................................... 9
   2.1 Overview .............................................................................................................. 9
   2.2 Requirements ....................................................................................................... 10
      2.2.1 Yum or ULN Channel Subscription ................................................................. 10
      2.2.2 Setting up UEK R5 ....................................................................................... 10
      2.2.3 Resource Requirements .............................................................................. 11
      2.2.4 Docker Engine Requirements ..................................................................... 11
      2.2.5 Oracle Container Registry Requirements ................................................... 12
      2.2.6 Network Time Service Requirements ........................................................ 14
      2.2.7 Firewall and iptables Requirements ............................................................ 14
      2.2.8 Network Requirements .............................................................................. 15
      2.2.9 SELinux Requirements ............................................................................... 15
      2.2.10 Requirements to Use Oracle Container Services for use with Kubernetes on Oracle Cloud Infrastructure ................................................................. 16
   2.3 Setting Up the Master Node .................................................................................. 17
   2.4 Setting Up a Worker Node .................................................................................... 19
   2.5 Upgrading 1.1.9 to 1.1.12 ................................................................................... 21
      2.5.1 Upgrading the Master Node from 1.1.9 to 1.1.12 ........................................ 22
      2.5.2 Upgrading Worker Nodes from 1.1.9 to 1.1.12 .......................................... 35
   2.6 Updating to Errata Releases .............................................................................. 38
      2.6.1 Updating the Master Node .......................................................................... 39
      2.6.2 Updating Worker Nodes ............................................................................. 42
3 Installing High Availability Oracle Container Services for use with Kubernetes ........... 45
   3.1 Overview .............................................................................................................. 45
   3.2 Requirements ....................................................................................................... 46
      3.2.1 Yum or ULN Channel Subscription ................................................................. 46
      3.2.2 Requirement for Upgrading the Unbreakable Enterprise Kernel .................... 46
      3.2.3 Resource Requirements .............................................................................. 47
      3.2.4 Docker Engine Requirements ..................................................................... 47
      3.2.5 Oracle Container Registry Requirements ................................................... 48
      3.2.6 Network Time Service Requirements ........................................................ 50
      3.2.7 Firewall and iptables Requirements ............................................................ 50
      3.2.8 Network Requirements .............................................................................. 51
      3.2.9 SELinux Requirements ............................................................................... 52
      3.2.10 Requirements to Use Oracle Container Services for use with Kubernetes on Oracle Cloud Infrastructure ................................................................. 52
   3.3 Setting Up the Master Cluster ............................................................................. 54
   3.4 Setting Up a Worker Node .................................................................................... 57
3.5 Upgrading .................................................................................................................. 58
  3.5.1 Updating the High Availability cluster ................................................................. 58
4 Kubernetes Administration and Configuration .......................................................... 65
  4.1 Kubernetes and iptables Rules .................................................................................. 65
  4.2 Using Kubernetes With a Proxy Server .................................................................... 65
  4.3 Cluster Backup and Restore ...................................................................................... 66
    4.3.1 Single Master Cluster ......................................................................................... 66
    4.3.2 High Availability Cluster ................................................................................... 69
  4.4 Kubernetes Dashboard .............................................................................................. 72
  4.5 Removing Worker Nodes from the Cluster ............................................................... 73
    4.5.1 Single Master Cluster ......................................................................................... 73
    4.5.2 High Availability Cluster ................................................................................... 74
5 Getting Started with Kubernetes .................................................................................. 75
  5.1 kubectl Basics .......................................................................................................... 75
  5.2 Pod Configuration Using a YAML Deployment ....................................................... 78
  5.3 Using Persistent Storage ........................................................................................... 82
    5.3.1 Persistent Storage Concepts .............................................................................. 83
    5.3.2 Configuring NFS ............................................................................................... 84
    5.3.3 Configuring iSCSI ............................................................................................. 86
6 For More Information About Kubernetes .................................................................... 89
A Developer Preview Releases ...................................................................................... 91
Preface

The Oracle Container Services for use with Kubernetes User’s Guide describes how to use Oracle Container Services for use with Kubernetes, which is an implementation of the open-source, containerized application management platform known as Kubernetes. Oracle provides additional tools, testing and support to deliver this technology with confidence. Kubernetes integrates with container products like Docker 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 on Oracle Linux 7.

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

Document generated on: 2019-04-16 (revision: 7383)

Audience

This document is intended for administrators who need to install, configure and use Kubernetes on Oracle Linux 7. 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:


Conventions

The following text conventions are used in this document:

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

Documentation Accessibility

For information about Oracle’s commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

For information on documentation accessibility features specific to this document, please refer to the Oracle Linux 7 Accessibility User’s Guide at: https://docs.oracle.com/cd/E52668_01/E92218/html/index.html.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
Access to Oracle Support

Chapter 1 About Oracle Container Services for use with Kubernetes

Table of Contents

1.1 Release Information for Oracle Container Services for use with Kubernetes .......................... 1
   1.1.1 New and Notable Features ......................................................................................... 1
   1.1.2 Technical Preview ...................................................................................................... 3
   1.1.3 Known Issues ............................................................................................................. 4
1.2 Kubernetes Components ........................................................................................................ 4
   1.2.1 Nodes ......................................................................................................................... 4
   1.2.2 Pods .......................................................................................................................... 6
   1.2.3 ReplicaSet, Deployment, StatefulSet Controllers ...................................................... 6
   1.2.4 Services ..................................................................................................................... 7
   1.2.5 Volumes ..................................................................................................................... 8
   1.2.6 Namespaces .............................................................................................................. 8

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/.

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

1.1 Release Information for Oracle Container Services for use with Kubernetes

Oracle Container Services for use with Kubernetes version 1.1.12 is based on Kubernetes version 1.12.5, as released upstream. A full change log and links to source and binaries are provided at https://github.com/kubernetes/kubernetes/blob/master/CHANGELOG-1.12.md. This section contains details about notable features and known issues for Kubernetes releases on Oracle Linux.

Warning

Oracle does not support Kubernetes on systems where the ol7_preview, ol7_developer or ol7_developer_EPEL yum repositories or ULN channels are enabled, or where software from these repositories or channels is currently installed on the systems where Kubernetes runs. Even if you follow the instructions in this document, you may render your platform unsupported if these repositories or channels are enabled or software from these channels or repositories is installed on your system.

1.1.1 New and Notable Features

Oracle Container Services for use with Kubernetes 1.1.12

Features in this release of Oracle Container Services for use with Kubernetes include:

• Upstream Kubernetes 1.12 software packaged for Oracle Linux
• Improvements and updates to setup and configuration utilities
New and Notable Features

- Support for high availability multi-master clusters
- Updates for the Kubernetes Dashboard software
- Improvements to cluster backup and restore tools
- Integration testing for use with Oracle Cloud Infrastructure
- A new cluster DNS service

Oracle has provided and tested a new setup and configuration utility that takes advantage of the
kubeadm cluster configuration utility to create high availability clusters with three master nodes. For more
information, see Chapter 3, Installing High Availability Oracle Container Services for use with Kubernetes.

Oracle has provided support for CoreDNS to function as the cluster DNS service. CoreDNS is installed by
default on all new clusters, and support for KubeDNS is deprecated. Note that CoreDNS support requires
Unbreakable Enterprise Kernel Release 5 (UEK R5). Although Oracle makes KubeDNS and support for
Unbreakable Enterprise Kernel Release 4 (UEK R4) available for users upgrading from earlier versions,
this configuration is deprecated and future upgrades will automatically replace KubeDNS with CoreDNS
and will require that the host platform is running UEK R5.

Important

To facilitate upgrade from Oracle Container Services for use with Kubernetes 1.1.9,
Oracle makes packages available for the 1.10 and 1.11 releases of Kubernetes.
These packages are not supported outside of the context of the upgrade process
described in Section 2.5, “Upgrading 1.1.9 to 1.1.12”.

Unsupported developer preview builds are no longer released in the ol7_preview repository. You can
read more in Appendix A, Developer Preview Releases.

Oracle Container Services for use with Kubernetes 1.1.9

This release is the first supported release of Oracle Container Services for use with Kubernetes. The
release is supported with the appropriate Oracle Linux support level defined in the Oracle Linux 7 License

This release of Oracle Container Services for use with Kubernetes is only made available for Oracle Linux
7 and is designed to integrate with Oracle Container Runtime for Docker only. For more information about

This release of Oracle Container Services for use with Kubernetes includes:

- Upstream Kubernetes 1.9 software packaged for Oracle Linux
- Setup and configuration utilities
- Kubernetes Dashboard software
- Cluster backup and restore tools
- Integration testing for use with Oracle Cloud Infrastructure

Oracle has provided and tested a setup and configuration script that takes advantage of the kubeadm
cluster configuration utility. This setup script eases the configuration and setup processes on Oracle Linux
and provides additional support for backup and recovery.

Setting up and configuring Kubernetes on Oracle Linux should be limited to the parameters of the provided
scripts and utilities described within this document.
1.1.2 Technical Preview

The following items are highlighted as technical preview features within the current release:

- **Flexvolume driver for Oracle Cloud Infrastructure.** The `oci-flexvolume-driver` package enables you to add block storage volumes hosted on Oracle Cloud Infrastructure to your Kubernetes cluster.

- **IPVS switching.** This functionality can automate load balancing and firewall management in your Kubernetes cluster through the use of unique virtual IP addresses and kernel-level proxying.

- **API server functions available as technical preview.** The API server includes many functions that cater to the full range of capabilities available in Kubernetes. These are described in the upstream documentation available at:

  https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.12/

Not all features described for the API are fully supported by Oracle. The following items are available as technical preview only:

**Workloads**
- CronJob v1beta1 batch
- Job v1 batch
- ReplicationController v1 core

**Discovery and Load Balancing**
- Ingress v1beta1 extensions

**Metadata**
- ControllerRevision v1 apps
- CustomResourceDefinition v1beta1 apiextensions
- LimitRange v1 core
- HorizontalPodAutoscaler v1 autoscaling
- InitializerConfiguration v1alpha1 admissionregistration
- PodDisruptionBudget v1beta1 policy
- PriorityClass v1beta1 scheduling
- PodPreset v1alpha1 settings
- PodSecurityPolicy v1beta1 extensions

**Cluster**
- APIService v1 apiregistration.k8s.io
- Binding v1 core
- CertificateSigningRequest v1beta1 certificates
1.1.3 Known Issues

- **Overlay networking issue on Oracle Cloud Infrastructure compute instances using VM 2.x shapes.** When setting up a Kubernetes cluster that uses overlay networking on compute nodes within Oracle Cloud Infrastructure, where the compute nodes use a VM 2.x shape, issues can result in the vxlan configuration for the cluster. This issue is commonly caused when the tx offload feature is enabled in the `bnxt_en` driver module. Nodes that are affected by the issue display errors similar to the following in the `dmesg` output:

```
[  610.495450] bnxt_en 0000:00:03.0 ens3: hwrm req_type 0xa1 seq id 0x67
  error 0xf
[  610.498246] bnxt_en 0000:00:03.0 ens3: hwrm_tunnel_dst_port_alloc failed.
  rc:15
```

You can resolve this issue by disabling the tx offload feature using the `ethtool` command. For example:

```
# ethtool --offload $(ip -o -4 route show to default | awk '{print $5}') tx off
```

Not all nodes that use this shape seem to be affected.

1.2 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/](https://kubernetes.io/docs/concepts/).

1.2.1 Nodes

Kubernetes Node architecture is described in detail at:

[https://kubernetes.io/docs/concepts/architecture/nodes/](https://kubernetes.io/docs/concepts/architecture/nodes/)

1.2.1.1 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.
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 (the Docker engine, in this case) 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.

### 1.2.1.2 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.

You can use the *kubeadm-ha-setup* utility to create a multi-master cluster where all master nodes are replicas of each other.

### 1.2.1.3 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:

- The 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.

- A Container Runtime: an environment where containers can be run. In this release, only Docker is supported. Therefore, the runtime here is equivalent to the Docker Engine.

- The Kube Proxy Service: a service that programs iptables 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.

1.2.2 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.

See https://kubernetes.io/docs/concepts/workloads/pods/pod/ for more information.

1.2.3 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.

See https://kubernetes.io/docs/concepts/workloads/controllers/replicaset/ for more information on ReplicaSets.

You can use a Deployment to manage pods and ReplicaSets. Deployments 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 ReplicaSets.

See https://kubernetes.io/docs/concepts/workloads/controllers/deployment/ for more information on Deployments.
You can use StatefulSets 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.

See https://kubernetes.io/docs/concepts/workloads/controllers/statefulset/ for more information on StatefulSets.

1.2.4 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 that are managed using iptables. 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.

See https://kubernetes.io/docs/concepts/services-networking/service/ for more information.
1.2.5 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 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*. *PersistentVolumes* are similar to *Volumes* 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 https://kubernetes.io/docs/concepts/storage/persistent-volumes/. Examples of using *PersistentVolumes* are also provided in Section 5.2, “Pod Configuration Using a YAML Deployment” and Section 5.3, “Using Persistent Storage”.

1.2.6 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.

See https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/ for more information on namespaces.
Chapter 2 Installing Oracle Container Services for use with Kubernetes

Table of Contents

2.1 Overview .................................................................................................................................................. 9
2.2 Requirements ............................................................................................................................................ 10
  2.2.1 Yum or ULN Channel Subscription .................................................................................................... 10
  2.2.2 Setting up UEK R5 .............................................................................................................................. 10
  2.2.3 Resource Requirements ....................................................................................................................... 11
  2.2.4 Docker Engine Requirements ............................................................................................................. 11
  2.2.5 Oracle Container Registry Requirements .......................................................................................... 12
  2.2.6 Network Time Service Requirements ................................................................................................. 14
  2.2.7 Firewall and iptables Requirements .................................................................................................. 14
  2.2.8 Network Requirements ...................................................................................................................... 15
  2.2.9 SELinux Requirements ....................................................................................................................... 15
  2.2.10 Requirements to Use Oracle Container Services for use with Kubernetes on Oracle
        Cloud Infrastructure ............................................................................................................................. 16
2.3 Setting Up the Master Node ..................................................................................................................... 17
2.4 Setting Up a Worker Node ....................................................................................................................... 19
2.5 Upgrading 1.1.9 to 1.1.12 ......................................................................................................................... 21
  2.5.1 Upgrading the Master Node from 1.1.9 to 1.1.12 ........................................................................... 22
  2.5.2 Upgrading Worker Nodes from 1.1.9 to 1.1.12 ............................................................................. 35
2.6 Updating to Errata Releases ..................................................................................................................... 38
  2.6.1 Updating the Master Node .................................................................................................................. 39
  2.6.2 Updating Worker Nodes ..................................................................................................................... 42

This chapter describes the steps required to install Kubernetes on an Oracle Linux 7 host, and to build a
Kubernetes cluster.

2.1 Overview

Kubernetes can be deployed in a variety of ways depending on requirements and on the tools that
you have at hand. The kubeadm package provides the kubeadm utility, a tool designed to make the
deployment of a Kubernetes cluster simple. Many users may find that using this tool directly, along with the
upstream documentation, provides the maximum configuration flexibility.

Oracle provides the kubeadm-setup.sh script in the kubeadm package to help new users install and
configure a base deployment of Kubernetes with the greater ease, regardless of whether it is hosted on
bare metal, on a virtual machine, or out on the cloud. The script handles checking that basic package
requirements are in place, setting proxy and firewall requirements, configuring networking, and initializing
a cluster configuration for the Kubernetes environment. The script uses the kubeadm utility, but handles
many additional configuration steps that can help new users get running with minimal effort.

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 backing up and restoring the master node for the cluster is simplified.

The instructions provided here, assume that you are new to Kubernetes and are using the provided
kubeadm-setup.sh script to deploy your cluster. This script is developed and tested at Oracle and
deployment using this script is fully supported. Alternate configurations and deployment mechanisms are untested by Oracle.

2.2 Requirements

Kubernetes is a clustered environment that generally functions with more than one node in the cluster. It is possible to run a single node cluster, but this defeats the point of having a cluster in the first place. Therefore, your environment should consist of two or more systems where Kubernetes is installed.

The following sections describe various other requirements that must be met to install and configure Kubernetes on an Oracle Linux 7 system.

Note

Oracle Container Services for use with Kubernetes 1.12 requires that you configure the system to use the Unbreakable Enterprise Kernel Release 5 (UEK R5) or later and boot the system with this kernel.

If you are still using the Unbreakable Enterprise Kernel Release 4 (UEK R4) and have a pre-existing cluster based on Oracle Container Services for use with Kubernetes 1.1.9, this is the last supported release available for your environment. It is strongly recommended that you upgrade your system to use the Unbreakable Enterprise Kernel Release 5 (UEK R5) and boot the system with this kernel.

2.2.1 Yum or ULN Channel Subscription

To install all of the required packages to use Kubernetes, you must ensure that you are subscribed to the correct yum repositories or Unbreakable Linux Network (ULN) channels.

If your systems are registered with ULN, enable the `ol7_x86_64_addons` channel.

If you use the Oracle Linux yum server, enable the `ol7_addons` repository on each system in your deployment. You can do this easily using `yum-config-manager`:

```
# yum-config-manager --enable ol7_addons
```

For more information on the `ol7_x86_64_addons` channel, please see the ULN Channels section of the Oracle Linux Unbreakable Linux Network User's Guide, available at:

https://docs.oracle.com/cd/E52668_01/E39381/html/ol_ol7chan_uln.html

Important

Oracle does not support Kubernetes on systems where the `ol7_preview`, `ol7_developer` or `ol7_developer_EPEL` repositories are enabled, or where software from these repositories is currently installed on the systems where Kubernetes runs. Even if you follow the instructions in this document, you may render your platform unsupported if these repositories or channels are enabled or software from these channels or repositories is installed on your system.

2.2.2 Setting up UEK R5

Oracle Linux Container Services for use with Kubernetes 1.1.12 and later versions require that you configure the system to use the Unbreakable Enterprise Kernel Release 5 (UEK R5) and boot the system with this kernel. If you are using either UEK R4 or the Red Hat Compatible Kernel (RHCK), you must configure Yum to allow you to install UEK R5.
Resource Requirements

1. If your system is registered with the Unbreakable Linux Network (ULN), disable access to the `ol7_x86_64_UEKR4` channel and enable access to the `ol7_x86_64_UEKR5` channel.

   If you use the Oracle Linux yum server, disable the `ol7_UEKR4` repository and enable the `ol7_UEKR5` repository. You can do this easily using `yum-config-manager`, if you have the `yum-utils` package installed:

   ```
   # yum-config-manager --disable ol7_UEKR4
   # yum-config-manager --enable ol7_UEKR5
   ```

2. Run the following command to upgrade the system to UEK R5:

   ```
   # yum update
   ```

   For information on how to make UEK R5 the default boot kernel, see the Oracle Linux Administrator’s Guide for Release 7 at https://docs.oracle.com/cd/E52668_01/E54669/html/ol7-grub2_bootloader.html.

3. Reboot the system, selecting the UEK R5 kernel if this is not the default boot kernel.

   ```
   # systemctl reboot
   ```

2.2.3 Resource Requirements

Each node in your cluster requires at least 2 GB of RAM and 2 or more CPUs to facilitate the use of `kubeadm` and any further applications that are provisioned using `kubectl`.

Also ensure that each node has a unique hostname, MAC address and product UUID as Kubernetes uses this information to identify and track each node in the cluster. You can verify the product UUID on each host with:

```
# dmidecode -s system-uuid
``` 

A storage volume with at least 5 GB free space must be mounted at `/var/lib/kubelet` on each node. For the underlying Docker engine an additional volume with at least 5 GB free space must be mounted on each node at `/var/lib/docker`.

2.2.4 Docker Engine Requirements

Kubernetes is used to manage containers running on a containerization platform deployed on several systems. On Oracle Linux, Kubernetes is currently only supported when used in conjunction with the Docker containerization platform. Therefore, each system in the deployment must have the Docker engine installed and ready to run. Support of Oracle Container Services for use with Kubernetes is limited to usage with the latest Oracle Container Runtime for Docker version available in the `ol7_addons` repository on the Oracle Linux yum server and in the `ol7_x86_64_addons` channel on ULN.

Please note that if you enable the `ol7_preview` repository, you may install a preview version of Oracle Container Runtime for Docker and your installation can no longer be supported by Oracle. If you have already installed a version of Docker from the `ol7_preview` repository, you should disable the repository and uninstall this version before proceeding with the installation.

Install, the Docker engine on all nodes in the cluster:

```
# yum install docker-engine
``` 

Enable the Docker service in `systemd` so that it starts on subsequent reboots and you should start the service before running the `kubeadm-setup.sh` script.

```
# systemctl enable docker
# systemctl start docker
```
See the Oracle Linux Docker User’s Guide for more information on installing and running the Docker engine. This is available at:

https://docs.oracle.com/cd/E52668_01/E87205/html/index.html

2.2.5 Oracle Container Registry Requirements

The images that are deployed by the kubeadm-setup.sh script are hosted on the Oracle Container Registry. For the script to be able to install the required components, you must perform the following steps:


2. Use the web interface to navigate to the Container Services business area and accept the Oracle Standard Terms and Restrictions for the Oracle software images that you intend to deploy. You are able to accept a global agreement that applies to all of the existing repositories within this business area. If newer repositories are added to this business area in the future, you may need to accept these terms again before performing upgrades.

3. Ensure that each of the systems that are used as nodes within the cluster are able to access https://container-registry.oracle.com and use the docker login command to authenticate against the Oracle Container Registry using the same credentials that you used to log into the web interface:

```
# docker login container-registry.oracle.com
```

The command prompts you for your user name and password.

Detailed information about the Oracle Container Registry is available in the Oracle Linux Docker User's Guide at:

https://docs.oracle.com/cd/E52668_01/E87205/html/oracle-registry-server.html

2.2.5.1 Using an Oracle Container Registry Mirror

It is also possible to use any of the Oracle Container Registry mirror servers to obtain the correct images to set up Oracle Container Services for use with Kubernetes. The Oracle Container Registry mirror servers are located within the same data centers used for Oracle Cloud Infrastructure. More information about the Oracle Container Registry mirror servers is available in the Oracle Linux Docker User's Guide at:

https://docs.oracle.com/cd/E52668_01/E87205/html/oracle-registry-server.html

Steps to use an alternate Oracle Container Registry mirror server follow:


2. On each node, use the docker login command to authenticate against the Oracle Container Registry mirror server using the same credentials that you used to log into the web interface:

```
# docker login container-registry-ash.oracle.com
```

The command prompts you for your user name and password.

3. After you have logged in, set the environment variable to use the correct registry mirror when you deploy Kubernetes:

```
# export KUBE_REPO_PREFIX=container-registry-ash.oracle.com/kubernetes
# echo 'export KUBE_REPO_PREFIX=container-registry-ash.oracle.com/kubernetes' > ~/.bashrc
```
If you are using Oracle Container Services for use with Kubernetes on Oracle Cloud Infrastructure, the `kubeadm-setup.sh` script automatically detects the most appropriate mirror server to use and sets this environment variable for you so that you do not have to perform this step. If you manually set the `KUBE_REPO_PREFIX` environment variable on the command line, the `kubeadm-setup.sh` honors the variable and does not attempt to detect which mirror server you should be using.

### 2.2.5.2 Setting Up an Optional Local Registry

If the systems that you are using for your Kubernetes cluster nodes do not have direct access to the Internet and are unable to connect directly to the Oracle Container Registry, you can set up a local Docker registry to perform this function. The `kubeadm-setup.sh` script provides an option to change the registry that you use to obtain these images. Instructions to set up a local Docker registry are provided in the Oracle Linux Docker User's Guide at:

https://docs.oracle.com/cd/E52668_01/E87205/html/prepare-registry-server.html

When you have set up a local Docker registry, you must pull the images required to run Oracle Container Services for use with Kubernetes, tag these images and then push them to your local registry. The images must be tagged identically to the way that they are tagged in the Oracle Container Registry. The `kubeadm-setup.sh` matches version numbers during the setup process and cannot successfully complete many operations if it cannot find particular versions of images. To assist with this process, Oracle Container Services for use with Kubernetes provides the `kubeadm-registry.sh` script in the `kubeadm` package.

To use the `kubeadm-registry.sh` script to automatically pull images from the Oracle Container Registry, tag them appropriately and push them to your local registry:

1. If you are using the Oracle Container Registry to obtain images, log in following the instructions in Section 2.2.5, "Oracle Container Registry Requirements". If you are using one of the Oracle Container Registry mirrors, see Section 2.2.5.1, “Using an Oracle Container Registry Mirror” for more information.

2. Run the `kubeadm-registry.sh` script with the required options:

   ```bash
   # kubeadm-registry.sh --to host.example.com:5000
   
   Substitute `host.example.com:5000` with the resolvable domain name and port by which your local Docker registry is available.
   
   You may optionally use the `--from` option to specify an alternate registry to pull the images from. You may also use the `--version` option to specify the version of Kubernetes images that you intend to host. For example:
   
   ```bash
   # kubeadm-registry.sh --to host.example.com:5000 --from \n   container-registry-ash.oracle.com/kubernetes --version 1.12.5
   ```

**Important**

If you are upgrading your environment and you intend to use a local registry, you must make sure that you have the most recent version of the images required to run Oracle Container Services for use with Kubernetes. You can use the `kubeadm-registry.sh` script to pull the correct images and to update your local registry before running the upgrade on the master node.

After your local Docker registry is installed and configured and the required images have been imported, you must set the environment variable that controls which registry server the `kubeadm-setup.sh` script uses. On each of the systems where you intend to run the `kubeadm-setup.sh` tool run the following commands:
2.2.6 Network Time Service Requirements

As a clustering environment, Kubernetes requires that system time is synchronized across each node within the cluster. Typically, this can be achieved by installing and configuring an NTP daemon on each node. You can do this in the following way:

1. Install the ntp package, if it is not already installed:

```bash
# yum install ntp
```

2. Edit the NTP configuration in `/etc/ntp.conf`. Your requirements may vary. If you are using DHCP to configure the networking for each node, it is possible to configure NTP servers automatically. If you have not got a locally configured NTP service that your systems can sync to, and your systems have Internet access, you can configure them to use the public `pool.ntp.org` service. See [https://www.ntppool.org/](https://www.ntppool.org/).

3. Ensure that NTP is enabled to restart at boot and that it is started before you proceed with your Kubernetes installation. For example:

```bash
# systemctl start ntpd
# systemctl enable ntpd
```

Note that systems running on Oracle Cloud Infrastructure are configured to use the `chronyd` time service by default, so there is no requirement to add or configure NTP if you are installing into an Oracle Cloud Infrastructure environment.


2.2.7 Firewall and iptables Requirements

Kubernetes uses `iptables` to handle many networking and port forwarding rules. Therefore, you must ensure that you do not have any rules set that may interfere with the functioning of Kubernetes. The `kubeadm-setup.sh` script requires an `iptables` rule to accept forwarding traffic. If this rule is not set, the script exits and notifies you that you may need to add this `iptables` rule. A standard Docker installation may create a firewall rule that prevents forwarding, therefore you may need to run:

```bash
# iptables -P FORWARD ACCEPT
```

The `kubeadm-setup.sh` script checks `iptables` rules and, where there is a match, instructions are provided on how to modify your `iptables` configuration to meet any requirements. See Section 4.1, “Kubernetes and iptables Rules” for more information.

If you have a requirement to run a firewall directly on the systems where Kubernetes is deployed, you must ensure that all ports required by Kubernetes are available. For instance, the TCP port 6443 must be accessible on the master node to allow other nodes to access the API Server. All nodes must be able to accept connections from the master node on the TCP port 10250 and traffic should be allowed on the UDP port 8472. All nodes must be able to receive traffic from all other nodes on every port on the network fabric that is used for the Kubernetes pods. The firewall must support masquerading.

Oracle Linux 7 installs and enables `firewalld`, by default. If you are running `firewalld`, the `kubeadm-setup.sh` script notifies you of any rules that you may need to add. In summary, run the following commands on all nodes:

```bash
# export KUBE_REPO_PREFIX="local-registry.example.com:5000/kubernetes"
# echo 'export KUBE_REPO_PREFIX="local-registry.example.com:5000/kubernetes"' > ~/.bashrc
```

Substitute `local-registry.example.com` with the IP address or resolvable domain name of the host on which your local Docker registry is configured.
Network Requirements

```bash
# firewall-cmd --add-masquerade --permanent
# firewall-cmd --add-port=10250/tcp --permanent
# firewall-cmd --add-port=8472/udp --permanent
```

Additionally, run the following command on the master node:

```bash
# firewall-cmd --add-port=6443/tcp --permanent
```

Use the `--permanent` option to make these firewall rules persistent across reboots.

Remember to restart the firewall for these rules to take effect:

```bash
# systemctl restart firewalld
```

### 2.2.8 Network Requirements

The `kubeadm-setup.sh` script requires that it is able to access the Oracle Container Registry and possibly other internet resources to be able to pull any container images that you required. Therefore, unless you intend to set up a local mirror for all of your container image requirements, the systems where you intend to install Kubernetes must either have direct internet access, or must be configured to use a proxy. See Section 4.2, “Using Kubernetes With a Proxy Server” for more information.

The `kubeadm-setup.sh` script checks whether the `br_netfilter` module is loaded and exits if it is not available. This module is required to enable transparent masquerading and to facilitate Virtual Extensible LAN (VxLAN) traffic for communication between Kubernetes pods across the cluster. If you need to check whether it is loaded, run:

```bash
# lsmod|grep br_netfilter
```

Kernel modules are usually loaded as they are needed, and it is unlikely that you would need to load this module manually. However, if necessary, you can load the module manually by running:

```bash
# modprobe br_netfilter
# echo "br_netfilter" > /etc/modules-load.d/br_netfilter.conf
```

Kubernetes requires that packets traversing a network bridge are processed by iptables for filtering and for port forwarding. To achieve this, tunable parameters in the kernel bridge module are automatically set when the kubeadm package is installed and a sysctl file is created at `/etc/sysctl.d/k8s.conf` that contains the following lines:

```bash
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
```

If you modify this file, or create anything similar yourself, you must run the following command to load the bridge tunable parameters:

```bash
# /sbin/sysctl -p /etc/sysctl.d/k8s.conf
```

The `kubeadm-setup.sh` script configures a flannel network as the network fabric that is used for communications between Kubernetes pods. This overlay network uses VxLANs to facilitate network connectivity: [https://github.com/coreos/flannel](https://github.com/coreos/flannel)

By default, the `kubeadm-setup.sh` script creates a network in the `10.244.0.0/16` range to host this network. The `kubeadm-setup.sh` script provides an option to set the network range to an alternate range, if required, during installation. Systems in the Kubernetes deployment must not have any network devices configured for this reserved IP range.

### 2.2.9 SELinux Requirements

The `kubeadm-setup.sh` script checks whether SELinux is set to enforcing mode. If enforcing mode is enabled, the script exits with an error requesting that you set SELinux to permissive mode. Setting SELinux
to permissive mode allows containers to access the host file system, which is required by pod networks. This requirement exists until SELinux support in the kubelet tool for Kubernetes is improved.

To disable SELinux temporarily, do the following:

```
# /usr/sbin/setenforce 0
```

To disable SELinux enforcing mode for subsequent reboots so that Kubernetes continues to run correctly, modify /etc/selinux/config and set the SELinux variable:

```
SELINUX=Permissive
```

### 2.2.10 Requirements to Use Oracle Container Services for use with Kubernetes on Oracle Cloud Infrastructure

Oracle Container Services for use with Kubernetes is engineered to work on Oracle Cloud Infrastructure. You can use all of the instructions that are provided in this document to install and configure Kubernetes across a group of compute instances. Additional information about configuration steps and usage of Oracle Cloud Infrastructure can be found at [https://docs.us-phoenix-1.oraclecloud.com/Content/home.htm](https://docs.us-phoenix-1.oraclecloud.com/Content/home.htm).

The most important requirement for Oracle Container Services for use with Kubernetes on Oracle Cloud Infrastructure is that your Virtual Cloud Network (VCN) allows the compute nodes that are used in your Kubernetes deployment to communicate through the required ports. By default, compute nodes are unable to access each other across the VCN until you have configured the Security List with the appropriate ingress rules.

Ingress rules should match the rules that are required in any firewall configuration, as described in Section 2.2.7, "Firewall and iptables Requirements". Typically, the configuration involves adding the following ingress rules to the default security list for your VCN:

1. **Allow 6443/TCP.**
   - **STATELESS:** Unchecked
   - **SOURCE CIDR:** `10.0.0.0/16`
   - **IP PROTOCOL:** TCP
   - **SOURCE PORT RANGE:** All
   - **DESTINATION PORT RANGE:** 6443

2. **Allow 10250/TCP.**
   - **STATELESS:** Unchecked
   - **SOURCE CIDR:** `10.0.0.0/16`
   - **IP PROTOCOL:** TCP
   - **SOURCE PORT RANGE:** All
   - **DESTINATION PORT RANGE:** 10250

3. **Allow 8472/UDP.**
   - **STATELESS:** Unchecked
• SOURCE CIDR: 10.0.0.0/16
• IP PROTOCOL: UDP
• SOURCE PORT RANGE: All
• DESTINATION PORT RANGE: 8472

Substitute 10.0.0.0/16 with the range used for the subnet that you created within the VCN for the compute nodes that will participate in the Kubernetes cluster. You may wish to limit the specific IP address range to the range that is used specifically by the cluster components, or you may expand this range, depending on your particular security requirements.

Important
The ingress rules that are described here are the core rules that you need to set up to allow the cluster to function. For each service that you define or intend to use, you might need to define additional rules in the Security List.

When creating compute instances to host Oracle Container Services for use with Kubernetes, all shape types are supported. The environment requires that you use Oracle Linux 7 Update 5 or later, with Unbreakable Enterprise Kernel Release 5 (UEK R5).

Note
A future version of Oracle Linux Container Services for use with Kubernetes will migrate existing single master clusters from KubeDNS to CoreDNS. CoreDNS requires an Oracle Linux 7 Update 5 image or later with the Unbreakable Enterprise Kernel Release 5 (UEK R5).

Existing Oracle Linux Container Services for use with Kubernetes 1.1.9 installations may already run on an Oracle Linux 7 Update 3 image, with Unbreakable Enterprise Kernel Release 4 (UEK R4), but you must upgrade your environment to permit future product upgrades.

2.3 Setting Up the Master Node

Before you begin, ensure you have satisfied the requirements in Section 2.2.5, “Oracle Container Registry Requirements”. Then on the host that you are configuring as the master node, install the kubeadm package and its dependencies:

```
# yum install kubeadm kubelet kubectl
```

As root, run kubeadm-setup.sh up to add the host as a master node:

```
# kubeadm-setup.sh up
Checking kubeadm and kubectl RPM ...
Starting to initialize master node ...
Checking if env is ready ...
Checking whether docker can pull busybox image ...
Checking access to container-registry.oracle.com/kubernetes...
Trying to pull repository container-registry.oracle.com/kube-proxy ...
v1.12.5: Pulling from container-registry.oracle.com/kube-proxy
Digest: sha256:9f57fd95dc9c59185913b316474d10aca262b5c89bba588f45c1b96ba6f8b
Status: Image is up to date for container-registry.oracle.com/kube-proxy:v1.12.5
Checking whether docker can run container ...
Checking firewalld settings ...
Checking iptables default rule ...
```
Setting Up the Master Node

Checking br_netfilter module ...
Checking sysctl variables ...
Enabling kubelet ...
Created symlink from /etc/systemd/system/multi-user.target.wants/kubelet.service to /etc/systemd/system/kubelet.service.
Check successful, ready to run 'up' command ...
Waiting for kubeadm to setup master cluster...
Please wait ...
\ - 80% completed
Waiting for the control plane to become ready ...
............
100% completed
clusterrole.rbac.authorization.k8s.io/flannel created
clusterrolebinding.rbac.authorization.k8s.io/flannel created
serviceaccount/flannel created
cfgmap/kube-flannel-cfg created
daemonset.extensions/kube-flannel-ds created
Installing kubernetes-dashboard ...
secret/kubernetes-dashboard-certs created
serviceaccount/kubernetes-dashboard created
role.rbac.authorization.k8s.io/kubernetes-dashboard-minimal created
rolebinding.rbac.authorization.k8s.io/kubernetes-dashboard-minimal created
deployment.apps/kubernetes-dashboard created
service/kubernetes-dashboard created
Enabling kubectl-proxy.service ...
Starting kubectl-proxy.service ...

[===> PLEASE DO THE FOLLOWING STEPS BELOW: <===]

Your Kubernetes master has initialized successfully!

To start using your cluster, you need to run the following as a regular user:

    mkdir -p $HOME/.kube
    sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
    sudo chown $(id -u):$(id -g) $HOME/.kube/config

You can now join any number of machines by running the following on each node as root:

    export KUBE_REPO_PREFIX=container-registry.oracle.com/kubernetes && kubeadm-setup.sh join 192.0.2.10:6443 \
        --token 8tipwo.tst0nvf7wcaqjcj0 --discovery-token-ca-cert-hash \n        sha256:f2a5b22b658683c3634459c8e7617c9d6c080c72dd149f3eb903445efe9d8346

If you do not specify a network range, the script uses the default network range of 10.244.0.0/16 to configure the internal network used for pod interaction within the cluster. To specify an alternative network range, run the script with the --pod-network-cidr option. For example, you would set the network to use the 10.100.0.0/16 range as follows:

    # kubeadm-setup.sh up --pod-network-cidr 10.100.0.0/16

The kubeadm-setup.sh script checks whether the host meets all of the requirements before it sets up the master node. If a requirement is not met, an error message is displayed, along with the recommended fix. You should fix the errors before running the script again.

The systemd service for the kubelet is automatically enabled on the host so that the master node always starts at system boot.

The output of the kubeadm-setup.sh script provides the command for adding worker nodes to the cluster. Take note of this command for later use. The token that is shown in the command is only valid for 24 hours. See Section 2.4, “Setting Up a Worker Node” for more details about tokens.
Preparing to Use Kubernetes as a Regular User

To use the Kubernetes cluster as a regular user, perform the following steps on the master node:

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

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

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

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

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

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

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

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

   **Note**
   You cannot use the `kubectl` command if the path to this file is not set for this environment variable. Remember to export the `KUBECONFIG` variable for each subsequent login so that the `kubectl` and `kubeadm` commands use the correct `admin.conf` file, otherwise you might find that these commands do not behave as expected after a reboot or a new login.

   For example, append the export line to your `.bashrc`:

   ```bash
   $ 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 Docker containers running as a Kubernetes pod, which can be viewed by running the following command on the master node:

   ```bash
   $ kubectl get pods -n kube-system
   ```

2.4 Setting Up a Worker Node

Repeat these steps on each host that you want to add to the cluster as a worker node.

Install the `kubeadm` package and its dependencies:

```bash
$ yum install kubeadm kubelet kubectl
```

As `root`, run the `kubeadm-setup.sh join` command to add the host as a worker node:

```bash
$ kubeadm-setup.sh join 192.0.2.10:6443 --token 8tipwo.tst0nvf7wcagjcj0
```

```bash
---discovery-token-ca-cert-hash 
sha256:f2a5b22b658683c3634459c8e7617c9d6c080c72dd149f3ab903445efed8346
```
Checking kubelet and kubectl RPM ...
Starting to initialize worker node ...
Checking if env is ready ...
Checking whether docker can pull busybox image ...
Checking access to container-registry.oracle.com/kubernetes ... v1.12.5: Pulling from container-registry.oracle.com/kubernetes/kube-proxy
Digest: sha256:9f57fd95dc9c591859930b2316474d10a1ca262b5c89ba588f45c1b96ba6f8b
Status: Image is up to date for container-registry.oracle.com/kubernetes/kube-proxy:v1.12.5
Checking whether docker can run container ...
Checking firewall settings ...
Checking iptables default rule ...
Checking br_netfilter module ...
Checking sysctl variables ...
Enabling kubelet ...
Created symlink from /etc/systemd/system/multi-user.target.wants/kubelet.service
to /etc/systemd/system/kubelet.service.
Check successful, ready to run 'join' command ...

[validation] WARNING: kubeadm doesn't fully support multiple API Servers yet
[preflight] running pre-flight checks
[discovery] Trying to connect to API Server "192.0.2.10:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://192.0.2.10:6443"
[discovery] Trying to connect to API Server "192.0.2.10:6443"
[discovery] Created cluster-info discovery client, requesting info from "https://192.0.2.10:6443"
[discovery] Requesting info from "https://192.0.2.10:6443" again
to validate TLS against the pinned public key
[discovery] Requesting info from "https://192.0.2.10:6443" again
to validate TLS against the pinned public key
[discovery] Cluster info signature and contents are valid
and TLS certificate validates against pinned roots, will use API Server "192.0.2.10:6443"
[discovery] Successfully established connection with API Server "192.0.2.10:6443"
[discovery] Cluster info signature and contents are valid
and TLS certificate validates against pinned roots, will use API Server "192.0.2.10:6443"
[discovery] Successfully established connection with API Server "192.0.2.10:6443"
[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.12" ConfigMap
in the kube-system namespace
[kubelet] Writing kubelet configuration to file "*/var/lib/kubelet/config.yaml"
[kubelet] Writing kubelet environment file with flags
to file "*/var/lib/kubelet/kubeadm-flags.env"
[preflight] Activating the kubelet service
[tlsbootstrap] Waiting for the kubelet to perform the TLS Bootstrap...
[patchnode] Uploading the CRI Socket information "*/var/run/dockershim.sock"
to the Node API object "worker1.example.com" as an annotation

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.

Replace the IP address and port, **192.0.2.10:6443**, with the IP address and port that is used by the API Server (the master node). Note that the default port is 6443.

Replace the **--token** value, **8tipwo.tst0nvf7wcaqjcj0**, with a valid token for the master node. If you do not have this information, run the following command on the **master node** to obtain this information:

```
# kubeadm token list

+---------+-------+------------+-----------+-----------------+-------------------+
<table>
<thead>
<tr>
<th>TOKEN</th>
<th>TTL</th>
<th>EXPIRES</th>
<th>USAGES</th>
<th>DESCRIPTION</th>
<th>EXTRA GROUPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>8tipwo.tst0nvf7wcaqjcj0</td>
<td>22h</td>
<td>2018-12-11</td>
<td></td>
<td></td>
<td>system:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>T03:32:44:08:00</td>
<td>authentication,</td>
<td>&lt;none&gt;</td>
<td>bootstrappers:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>signing</td>
<td></td>
<td>kubeadm:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>default-node-token</td>
</tr>
</tbody>
</table>
```

By default, tokens expire after 24 hours. If you are joining a node to the cluster after the current token has expired, you can create a new token by running the following command on the **master node**: 

```
# kubeadm token create --print-join-info
```
Upgrading 1.1.9 to 1.1.12

You can explicitly set the expiry period for a token when you create it by using the `--ttl` option. This option sets the expiration time of the token, relative to the current time. The value is generally set in seconds, but other units can be specified as well. For example, you can set the token expiry for 15m (or 15 minutes) from the current time; or, for 1h (1 hour) from the current time. A value of 0 means the token never expires, but this value is not recommended.

Replace the `--discovery-token-ca-cert-hash` value, `f2a5b22b658683c3634459c8e7617c9d6c080c72dd149f3eb903445efe9d8346`, with the correct SHA256 CA certificate hash that is used to sign the token certificate for the master node. If you do not have this information, run the following command chain on the master node to obtain it:

```bash
# openssl x509 -pubkey -in /etc/kubernetes/pki/ca.crt | openssl rsa -pubin -outform der 2>/dev/null | 
openssl dgst -sha256 -hex | sed 's/^.* //'
f2a5b22b658683c3634459c8e7617c9d6c080c72dd149f3eb903445efe9d8346
```

The `kubeadm-setup.sh` script checks whether the host meets all the requirements before it sets up a worker node. If a requirement is not met, an error message is displayed together with the recommended fix. You should fix the errors before running the script again.

The `kubelet systemd` service is automatically enabled on the host so that the worker node always starts at boot.

After the `kubeadm-setup.sh` join command completes, check that the worker node has joined the cluster on the master node:

```bash
$ kubectl get nodes
NAME                  STATUS    ROLES   AGE       VERSION
master.example.com    Ready     master  1h        v1.12.5+2.1.1.el7
worker1.example.com   Ready     <none>  1h        v1.12.5+2.1.1.el7
```

The output for this command displays a listing of all of the nodes in the cluster and their status.

2.5 Upgrading 1.1.9 to 1.1.12

The following instructions are specifically for a major package upgrade from Oracle Linux Container Services for use with Kubernetes 1.1.9 to version 1.1.12.

**Warning**

The upgrade process that is described here only applies for the stated upgrade path on existing hosts.

Oracle does not support upgrading existing clusters between smaller errata releases by using the `kubeadm-upgrade.sh` script. Instead, you must use the `kubeadm-setup.sh` script that is described in Section 2.6, “Updating to Errata Releases”.

These instructions work on hosts that are booting from UEK R4, but it is recommended that hosts currently running UEK R4 are upgraded to use UEK R5 to facilitate future upgrades, where KubeDNS is deprecated.

The upgrade process requires you to first upgrade the master node in your cluster, and then update each of the worker nodes. The upgrade of the master node is scripted so that the pre-requisite checks, validation, and reconfiguration are automated. It is a good practice to make a backup file for your cluster.
before upgrading. This process is described in Section 2.5.1, “Upgrading the Master Node from 1.1.9 to 1.1.12”.

After the master node is upgraded, you can upgrade each worker node, as described in Section 2.5.2, “Upgrading Worker Nodes from 1.1.9 to 1.1.12”.

When the upgrade of the cluster has completed, you must restart or redeploy any applications that were running in the cluster.

Important
Oracle does not support an upgrade from a preview release to a stable and supported release.

Oracle also does not support upgrading existing single master node clusters that are built with the kubeadm-setup.sh script to High Availability clusters. You must build and manage High Availability clusters by using the kubeadm-ha-setup utility.

2.5.1 Upgrading the Master Node from 1.1.9 to 1.1.12

You must upgrade the master node in your cluster before upgrading the worker nodes. Use the kubeadm-upgrade.sh upgrade command on the master node to create the necessary backup files and complete the necessary steps to prepare and upgrade the cluster.

Important
Before you perform any update operations, make a backup file of your cluster at its current version. After you update the kubeadm package, any backup files that you make are not backward compatible: if you revert to an earlier version of Oracle Container Services for use with Kubernetes, the restore operation might fail to successfully load your backup file. See Section 4.3, “Cluster Backup and Restore” for more information.

Do not use backups that are generated by kubeadm-setup to restore from a failed 1.1.9 to 1.1.12 upgrade. The kubeadm-upgrade tool provides its own separate backup and restore mechanism, as described later in this section.

Upgrade the master node to 1.1.12

1. Unlike errata upgrades, you do not need to manually update the kubeadm package, but you do need to install the kubeadm-upgrade package is required:

```
# yum install kubeadm-upgrade
```

2. If you are using the Oracle Container Registry to obtain images, log in.

Follow the instructions in Section 2.2.5, “Oracle Container Registry Requirements”. Note that if images are updated on the Oracle Container Registry, you may be required to accept the Oracle Standard Terms and Restrictions again before you are able to perform the upgrade. If you are using one of the Oracle Container Registry mirrors, see Section 2.2.5.1, “Using an Oracle Container Registry Mirror” for more information.

If you configured a local registry, you may need to set the KUBE_REPO_PREFIX environment variable to point to the appropriate registry. You might also need to update your local registry with the most current images for the version that you are upgrading to. See Section 2.2.5.2, “Setting Up an Optional Local Registry” for more information.
3. Ensure that you open any new firewall ports, as described in Section 2.2.7, “Firewall and iptables Requirements”.

4. Create a pre-upgrade backup file. Unlike the errata release upgrade procedure, the backup file is generated by using `kubeadm-upgrade.sh backup`. In the event that the upgrade does not complete successfully, the backup can revert back to the configuration of your cluster prior to upgrade.

```
# kubeadm-setup.sh stop
Stopping kubelet now ...  
Stopping containers now ...  

# kubeadm-upgrade.sh backup /backups
-- Running upgrade script---
Back ing up cluster 
Creating backup at directory /backups ... 
Using 3.1.11 
Checking if container-registry.oracle.com/kubernetes/etcd-amd64:3.1.11 is available 
dc9ed9408e82dbd9d925c4dd60206f9c60dce98c150cb32517284a6ef764f59d 
/var/run/kubeadm/backup/etcd-backup-1546953894.tar 
aa2dad1ba2c2ec486d30fe0a1b29566b257474429d79889472fd79128489ae0 
/var/run/kubeadm/backup/k8s-master-0-1546953894.tar 
Backup is successfully stored at /backups/master-backup-v1.9.11-0-1546953894.tar ... 
You can restart your cluster now by doing: 
# kubeadm-setup.sh restart 
Storing meta-data to backup file master-backup-v1.9.11-0-1546953894.tar .version-info 
Backup creation successful :) 

# kubeadm-setup.sh restart 
Restarting containers now ... 
Detected node is master ... 
Checking if env is ready ... 
Checking whether docker can pull busybox image ... 
Checking access to container-registry.oracle.com/kubernetes ... 
Trying to pull repository container-registry.oracle.com/kubernetes/pause ... 
3.1: Pulling from container-registry.oracle.com/kubernetes/pause 
Digest: sha256:802ef89b9eb7e874a76e1cfdf79ed990b63b0b84005cfa9f0293779ac0261b49 
Status: Image is up to date for container-registry.oracle.com/kubernetes/pause:3.1 
Checking firewalld settings ... 
Checking iptables default rule ... 
Checking br_netfilter module ... 
Checking sysctl variables ... 
Restarting kubelet ... 
Waiting for node to restart ... 
.........*............. 
Master node restarted. Complete synchronization between nodes may take a few minutes. 
```

5. Run the `kubeadm-upgrade.sh upgrade` command as `root` on the master node.

```
# kubeadm-upgrade.sh upgrade
-- Running upgrade script---
Number of cpu present in this system 2 
Total memory on this system: 7710MB 
Space available on the mount point /var/lib/docker: 44GB 
Space available on the mount point /var/lib/kubelet: 44GB 
kubeadm version 1.9 
kubectl version 1.9 
kubelet version 1.9 
o17_addons repo enabled 
[WARNING] This action will upgrade this node to latest version 
[WARNING] The cluster will be upgraded through intermediate versions which are unsupported 
[WARNING] You must take backup before upgrading the cluster as upgrade may fail 
Please select 1 (continue) or 2 (abort) : 
1) continue 
2) abort 
```
Upgrading the Master Node from 1.1.9 to 1.1.12

Upgrading master node
Checking access to container-registry.oracle.com/kubernetes for update
Trying to pull repository container-registry.oracle.com/kubernetes/kube-proxy-amd64
v1.10.5: Pulling from container-registry.oracle.com/kubernetes/kube-proxy-amd64
Digest: sha256:4739e1154b8a95786bc94d441cbf493083d1983e98087c8a8279e616582f1
Status: Image is up to date for container-registry.oracle.com/kubernetes/kube-proxy-amd64:v1.10.5
Checking access to container-registry.oracle.com/kubernetes for update
Trying to pull repository container-registry.oracle.com/kubernetes/kube-proxy-amd64
v1.11.3: Pulling from container-registry.oracle.com/kubernetes/kube-proxy-amd64
Digest: sha256:9eb8b56e15078cb499a3360f138cc16987cf5ae06593f77d0881af6badbe
Status: Image is up to date for container-registry.oracle.com/kubernetes/kube-proxy-amd64:v1.11.3
Checking access to container-registry.oracle.com/kubernetes for update
Trying to pull repository container-registry.oracle.com/kubernetes/kube-proxy ...
v1.12.5: Pulling from container-registry.oracle.com/kubernetes/kube-proxy
Digest: sha256:9eb8b56e15078cb499a3360f138cc16987cf5ae06593f77d0881af6badbe
Status: Image is up to date for container-registry.oracle.com/kubernetes/kube-proxy:v1.12.5

Loaded plugins: langpacks, ulninfo
Resolving Dependencies
---> Running transaction check
---> Package kubeadm.x86_64 0:1.9.11-2.1.1.el7 will be updated
---> Package kubeadm.x86_64 0:1.10.5-2.0.2.el7 will be an update
---> Package kubectl.x86_64 0:1.9.11-2.1.1.el7 will be updated
---> Package kubectl.x86_64 0:1.10.5-2.0.2.el7 will be an update
---> Package kubelet.x86_64 0:1.9.11-2.1.1.el7 will be updated
---> Package kubelet.x86_64 0:1.10.5-2.0.2.el7 will be an update
---> Package kubelet.x86_64 0:1.10.5-2.0.2.el7 will be an update
---> Finished Dependency Resolution

Dependencies Resolved

================================================================
Package   Arch          Version          Repository       Size
================================================================
Updating:
kubelet   x86_64     1.10.5-2.0.2.el7.o17_addons 17 M
kubectl   x86_64     1.10.5-2.0.2.el7.o17_addons 7.7 M
kubeadm   x86_64     1.10.5-2.0.2.el7.o17_addons 18 M

Transaction Summary
================================================================
Upgrade 3 Packages
Total download size: 43 M
Downloading packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.

Total
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction

Updating: kubeadm-1.10.5-2.0.2.el7.x86_64 1/6
Updating: kubelet-1.10.5-2.0.2.el7.x86_64 2/6
Updating: kubectl-1.10.5-2.0.2.el7.x86_64 3/6
Cleanup: kubeadm-1.9.11-2.1.1.el7.x86_64 4/6
Cleanup: kubelet-1.9.11-2.1.1.el7.x86_64 5/6
Cleanup: kubectl-1.9.11-2.1.1.el7.x86_64 6/6
Verifying: kubeadm-1.10.5-2.0.2.el7.x86_64 1/6
Verifying: kubelet-1.10.5-2.0.2.el7.x86_64 2/6
Verifying: kubectl-1.10.5-2.0.2.el7.x86_64 3/6
Verifying: kubelet-1.9.11-2.1.1.el7.x86_64 4/6
Verifying: kubectl-1.9.11-2.1.1.el7.x86_64 5/6
Verifying: kubectl-1.9.11-2.1.1.el7.x86_64 6/6
Upgrading the Master Node from 1.1.9 to 1.1.12

Updated:
  kubeadm.x86_64 0:1.10.5-2.0.2.e17  kubectl.x86_64 0:1.10.5-2.0.2.e17
  kubelet.x86_64 0:1.10.5-2.0.2.e17

Complete!
Upgrading pre-requisite
Checking whether api-server is using image lower than 1.9
Upgrading pre-requisite done
Checking cluster health ...
  ...
  [preflight] Running pre-flight checks.
  [upgrade] Making sure the cluster is healthy:
  [upgrade/config] Making sure the configuration is correct:
  [upgrade/config] Reading configuration options from a file:
    /var/run/kubeadm/kubeadm-cfg
  [upgrade/version] You have chosen to change the cluster version to "v1.10.5"
  [upgrade/versions] Cluster version: v1.9.11+2.1.1.e17
  [upgrade/versions] kubeadm version: v1.10.5+2.0.2.e17
  [upgrade/prepull] Will prepull images for components [kube-apiserver kube-controller-manager kube-scheduler]
  [upgrade/apply] Upgrading your Static Pod-hosted control plane to version "v1.10.5"...
    Static pod: kube-apiserver-master.example.com hash: 3b6cc643053ae0164a687e53fbcf4eb7
    Static pod: kube-controller-manager-master.example.com hash: 78b0313a0deb52655c1f6668001a2c093
    Static pod: kube-scheduler-master.example.com hash: 8fa7d39f0a3246bb39baf371270221a
  [upgrade/etcd] Upgrading to TLS for etcd
    Static pod: etcd-master.example.com hash: 196164156fbbd2ef7daaf8c6a0ec6379
    [etcd] Wrote Static Pod manifest for a local etcd instance to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests139181353/etcd.yaml"
    [certificates] Generated etcd/ca certificate and key.
    [certificates] Generated etcd/server certificate and key.
    [certificates] etcd/server serving cert is signed for DNS names [localhost] and IPs [127.0.0.1]
    [certificates] Generated etcd/peer certificate and key.
    [certificates] etcd/peer serving cert is signed for DNS names [master.example.com] and IPs [19.0.2.10]
    [certificates] Generated etcd/healthcheck-client certificate and key.
    [upgrade/staticpods] Moved new manifest to "/etc/kubernetes/manifests/etcd.yaml" and backed up old manifest to "/etc/kubernetes/tmp/kubeadm-backup-manifests154060916/etcd.yaml"
    [upgrade/staticpods] Not waiting for pod-hash change for component "etcd"
    [util/etcd] Waiting for etcd to become available
    [util/etcd] Waiting 30s for initial delay
    [util/etcd] Attempting to see if all cluster endpoints are available 1/10
    [util/etcd] Waiting 15s until next retry
    [util/etcd] Attempting to see if all cluster endpoints are available 2/10
    [util/etcd] Waiting 15s until next retry
    [util/etcd] Attempting to see if all cluster endpoints are available 3/10
    [upgrade/staticpods] Writing new Static Pod manifests to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests139181353"
    [controlplane] Wrote Static Pod manifest for component kube-apiserver to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests139181353/kube-apiserver.yaml"
    [controlplane] Wrote Static Pod manifest for component kube-controller-manager to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests139181353/kube-controller-manager.yaml"
    [controlplane] Wrote Static Pod manifest for component kube-scheduler to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests139181353/kube-scheduler.yaml"
    [upgrade/staticpods] The etcd manifest will be restored if component "kube-apiserver" fails to upgrade
    [certificates] Using the existing etcd/ca certificate and key.
Upgrading the Master Node from 1.1.9 to 1.1.12

[upgrade/staticpods] Moved new manifest to "/etc/kubernetes/manifests/kube-apiserver.yaml" and backed up old manifest to "/etc/kubernetes/tmp/kube-apiserver-backup-manifests154060916/kube-apiserver.yaml"

[upgrade/staticpods] Waiting for the kubelet to restart the component
Static pod: kube-apiserver-master.example.com hash: 3b6cc643053ae016a4687e53fbcf4eb7
Static pod: kube-apiserver-master.example.com hash: f7c7c2a1693f48bc614611996c47cad
[apiclient] Found 1 Pods for label selector component=kube-apiserver
[upgrade/staticpods] Component "kube-apiserver" upgraded successfully!

[upgrade/staticpods] Moved new manifest to "/etc/kubernetes/manifests/kube-controller-manager.yaml" and backed up old manifest to "/etc/kubernetes/tmp/kube-apiserver-backup-manifests154060916/kube-controller-manager.yaml"

[upgrade/staticpods] Waiting for the kubelet to restart the component
Static pod: kube-controller-manager-master.example.com hash: 78b0313a30bbf65cf169686001a2c093
Static pod: kube-controller-manager-master.example.com hash: 3fffc1159801c3777e45ff96e75444
[apiclient] Found 1 Pods for label selector component=kube-controller-manager
[upgrade/staticpods] Component "kube-controller-manager" upgraded successfully!

[upgrade/staticpods] Moved new manifest to "/etc/kubernetes/manifests/kube-scheduler.yaml" and backed up old manifest to "/etc/kubernetes/tmp/kube-apiserver-backup-manifests154060916/kube-scheduler.yaml"

[upgrade/staticpods] Waiting for the kubelet to restart the component
Static pod: kube-scheduler-master.example.com hash: 8fa7d39f0a3246b8f39baf3712702214a
Static pod: kube-scheduler-master.example.com hash: c191e26d0f0a009812a2d61f0d7e5f
[apiclient] Found 1 Pods for label selector component=kube-scheduler
[upgrade/staticpods] Component "kube-scheduler" upgraded successfully!

[uploadconfig] Storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[bootstraptoken] Configured RBAC rules to allow Node Bootstrap tokens to post CSRs
[bootstraptoken] Configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstraptoken] Configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[addons] Applied essential addon: kube-dns
[addons] Applied essential addon: kube-proxy

[upgrade/Successful] SUCCESS! Your cluster was upgraded to "v1.10.5". Enjoy!

[upgrade/kubelet] Now that your control plane is upgraded, please proceed with upgrading your kubelets in turn.
Upgrading kubeadm to 1.11.3 version
Loaded plugins: langpacks, ulninfo
Resolving Dependencies
--> Running transaction check
--> Running transaction check
  Package kubeadm.x86_64 0:1.10.5-2.0.2.el7 will be updated
  Package kubeadm.x86_64 0:1.11.3-2.0.2.el7 will be an update
--> Finished Dependency Resolution

Transaction Summary
---------------------------------------------
Upgrade 1 Package

Total download size: 7.6 M
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.
Running transaction check
Running transaction test
Transaction test succeeded

Running transaction
  Updating : kubeadm-1.11.3-2.0.2.el7.x86_64 1/2
  Cleanup : kubeadm-1.10.5-2.0.2.el7.x86_64 2/2
  Verifying : kubeadm-1.11.3-2.0.2.el7.x86_64 1/2
  Verifying : kubeadm-1.10.5-2.0.2.el7.x86_64 2/2

Updated:
  kubeadm.x86_64 0:1.11.3-2.0.2.el7

Complete!

Upgrading pre-requisite
Checking whether api-server is using image lower than 1.9
Upgrading pre-requisite done
Checking cluster health ...

[preflight] Running pre-flight checks.
[upgrade] Making sure the cluster is healthy:
  [upgrade/config] Making sure the configuration is correct:
  [upgrade/config] Reading configuration options from a file: /var/run/kubeadm/kubeadm-cfg
  [upgrade/apply] Respecting the --cri-socket flag that is set with higher priority than the config file.
  [upgrade/version] You have chosen to change the cluster version to "v1.11.3"
  [upgrade/versions] Cluster version: v1.10.5+2.0.2.el7
  [upgrade/versions] kubeadm version: v1.11.3+2.0.2.el7
  [upgrade/version] Found 1 potential version compatibility errors but skipping since the --force flag is set:
    - There are kubelets in this cluster that are too old that have these versions [v1.9.11+2.1.1.el7]
  [upgrade/prepull] Will prepull images for components [kube-apiserver kube-controller-manager kube-scheduler etcd]
  [upgrade/apply] Upgrading your Static Pod-hosted control plane to version "v1.11.3" ...
  Static pod: kube-apiserver-master.example.com hash: f7c7c2a1693f48bc61466119961c47cad
  Static pod: kube-apiserver-master.example.com hash:
  3fffc1595801c3777e45f9507544
  Static pod: kubeadm-upgraded-manifests842182537/etcd.yaml
  [certificates] Using the existing etcd/ca certificate and key.
  [certificates] Using the existing etcd/server certificate and key.
  [certificates] Using the existing etcd/peer certificate and key.
  [certificates] Using the existing etcd/healthcheck-client certificate and key.
  [upgrade/staticpods] Moved new manifest to "/etc/kubernetes/manifests/etcd.yaml" and backed up old manifest to
  [util/etcd] Waiting 0s for initial delay
  [util/etcd] Waiting for the kubelet to restart the component
  Static pod: etcd-master.example.com hash: 6ecccc01b0c9daa0705a1396ef38e5
  Static pod: etcd-master.example.com hash: 6ecccc01b0c9daa0705a1396ef38e5
  Static pod: etcd-master.example.com hash: 6ecccc01b0c9daa0705a1396ef38e5
  Static pod: etcd-master.example.com hash: 560672e3081cf0ff6a30a1f943240eb
  [api/client] Found 1 Pods for label selector component=etcd
  [upgrade/staticpods] Component "etcd" upgraded successfully!
  [upgrade/etcd] Waiting for etcd to become available
  [util/etcd] Waiting 0s for initial delay
  [util/etcd] Attempting to see if all cluster endpoints are available 1/10
  [upgrade/staticpods] Writing new Static Pod manifest to
  [util/etcd] Attempting to see if all cluster endpoints are available 2/10
  [util/etcd] Attempting to see if all cluster endpoints are available 3/10
  [util/etcd] Attempting to see if all cluster endpoints are available 4/10
  [util/etcd] Attempting to see if all cluster endpoints are available 5/10
  [util/etcd] Attempting to see if all cluster endpoints are available 6/10
  [util/etcd] Attempting to see if all cluster endpoints are available 7/10
  [util/etcd] Attempting to see if all cluster endpoints are available 8/10
  [util/etcd] Attempting to see if all cluster endpoints are available 9/10
  [util/etcd] Attempting to see if all cluster endpoints are available 10/10
Upgrading the Master Node from 1.1.9 to 1.1.12

[certificates] Using the existing apiserver-etcd-client certificate and key.

[upgrade/staticpods] Moved new manifest to "/etc/kubernetes/manifests/kube-apiserver.yaml" and backed up old manifest to "/etc/kubernetes/tmp/kubeadm-backup-manifests-2019-01-09-07-25-48/kube-apiserver.yaml"

[upgrade/staticpods] Waiting for the kubelet to restart the component Static pod: kube-apiserver-master.example.com hash: f7c7c2a1693f48bc6146119961c47cad
Static pod: kube-apiserver-master.example.com hash: f7c7c2a1693f48bc6146119961c47cad
Static pod: kube-apiserver-master.example.com hash: f7c7c2a1693f48bc6146119961c47cad
Static pod: kube-apiserver-master.example.com hash: 9eefcb38114108702fad91f927799c04

[apiclient] Found 1 Pods for label selector component=kube-apiserver

[upgrade/staticpods] Component "kube-apiserver" upgraded successfully!


[upgrade/staticpods] Waiting for the kubelet to restart the component Static pod: kube-controller-manager-master.example.com hash: 3fffc11595801c3777e45ff96e75444
Static pod: kube-controller-manager-master.example.com hash: 32bf07233f37a7b5c48793a087d3f98a

[apiclient] Found 1 Pods for label selector component=kube-controller-manager

[upgrade/staticpods] Component "kube-controller-manager" upgraded successfully!

[upgrade/staticpods] Moved new manifest to "/etc/kubernetes/manifests/kube-scheduler.yaml" and backed up old manifest to "/etc/kubernetes/tmp/kubeadm-backup-manifests-2019-01-09-07-25-48/kube-scheduler.yaml"

[apiclient] Found 1 Pods for label selector component=kube-scheduler

[upgrade/staticpods] Component "kube-scheduler" upgraded successfully!

[uploadconfig] storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace

[kubelet] Creating a ConfigMap "kubelet-config-1.11" in namespace kube-system with the configuration for the kubelets in the cluster

[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.11" ConfigMap in the kube-system namespace

[kubelet] Writing kubelet configuration file to "/var/lib/kubelet/config.yaml"
[kubelet] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"

[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node API object "master.example.com" as an annotation

[bootstraptoken] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials

[bootstraptoken] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token

[bootstraptoken] configured RBAC rules to allow certificate rotation for all node client certificates in the cluster

[addons] Applied essential addon: kube-dns
[addons] Applied essential addon: kube-proxy

[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.11.3". Enjoy!

[upgrade/kubelet] Now that your control plane is upgraded, please proceed with upgrading your kubelets if you haven't already done so.

Upgrading kubelet and kubectl now ...

Checking kubelet and kubectl RPM ...

[INFO] yum install -y kubelet-1.11.3-2.0.2.el7.x86_64
Loaded plugins: langpacks, ulninfo
Resolving Dependencies
--> Running transaction check
---> Package kubelet.x86_64 0:1.10.5-2.0.2.el7 will be updated
---> Package kubectl.x86_64 0:1.11.3-2.0.2.el7 will be an update
---> Finished Dependency Resolution

Dependencies Resolved

===================================================================================
Upgrading the Master Node from 1.1.9 to 1.1.12

<table>
<thead>
<tr>
<th>Package</th>
<th>Arch</th>
<th>Version</th>
<th>Repository</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>kubelet</td>
<td>x86_64</td>
<td>1.11.3-2.0.2.el7</td>
<td>ol7_addons</td>
<td>18 M</td>
</tr>
</tbody>
</table>

Transaction Summary

Upgrade 1 Package

Total download size: 18 M

Loading packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.
kubelet-1.11.3-2.0.2.el7.x86_64.rpm

Running transaction check
Running transaction test
Transaction test succeeded

Running transaction
  Updating: kubelet-1.11.3-2.0.2.el7.x86_64 1/2
  Cleanup: kubelet-1.10.5-2.0.2.el7.x86_64 2/2
  Verifying: kubelet-1.11.3-2.0.2.el7.x86_64 1/2
  Verifying: kubelet-1.10.5-2.0.2.el7.x86_64 2/2

Updated:
kubelet.x86_64 0:1.11.3-2.0.2.el7

Complete!

Loaded plugins: langpacks, ulninfo
Resolving Dependencies

Transaction Summary

Upgrade 1 Package

Total download size: 7.6 M

Loading packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.
kubelet-1.11.3-2.0.2.el7.x86_64.rpm

Running transaction check
Running transaction test
Transaction test succeeded

Running transaction
  Updating: kubelet-1.11.3-2.0.2.el7.x86_64 1/2
  Cleanup: kubelet-1.10.5-2.0.2.el7.x86_64 2/2
  Verifying: kubelet-1.11.3-2.0.2.el7.x86_64 1/2
  Verifying: kubelet-1.10.5-2.0.2.el7.x86_64 2/2

Updated:
kubelet.x86_64 0:1.11.3-2.0.2.el7

Complete!

Upgrading kubelet and kubectl to 1.11.3 version

Loaded plugins: langpacks, ulninfo
Package kubelet-1.11.3-2.0.2.el7.x86_64 already installed and latest version
Package kubectl-1.11.3-2.0.2.el7.x86_64 already installed and latest version
Nothing to do
Upgrading kubeadm to 1.12.5-2.11.el7 version
Loaded plugins: langpacks, ulninfo
Resolving Dependencies
--> Running transaction check
--> Package kubeadm.x86_64 0:1.11.3-2.0.2.el7 will be updated
--> Package kubeadm.x86_64 0:1.12.5-2.11.el7 will be an update
--> Finished Dependency Resolution

Dependencies Resolved

Package Arch Version Repository Size
-----------------------------------------------
kubeadm x86_64 1.12.5-2.1.1.el7 o17_addons 7.3 M

Transaction Summary
-----------------------------------------------
Upgrade 1 Package
Total download size: 7.3 M
Downloading packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.
kubeadm-1.12.5-2.1.1.el7.x86_64.rpm | 7.3 MB 00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
   Updating : kubeadm-1.12.5-2.1.1.el7.x86_64 1/2
   Cleanup  : kubeadm-1.11.3-2.0.2.el7.x86_64 2/2
   Verifying: kubeadm-1.12.5-2.1.1.el7.x86_64 1/2
   Verifying: kubeadm-1.11.3-2.0.2.el7.x86_64 2/2

Updated:
kubeadm.x86_64 0:1.12.5-2.1.1.el7

Complete!
Upgrading pre-requisite
Checking whether api-server is using image lower than 1.9
Upgrading pre-requisite done
Checking cluster health ...

[preflight] Running pre-flight checks.
[upgrade] Making sure the cluster is healthy:
[upgrade/config] Making sure the configuration is correct:
[upgrade/config] Reading configuration options from a file: /var/run/kubeadm/kubeadm-cfg
[upgrade/apply] Respecting the --cri-socket flag that is set with higher priority than the config file.
[upgrade/version] You have chosen to change the cluster version to "v1.12.5"
[upgrade/versions] Cluster version: v1.11.3+2.0.2.el7
[upgrade/versions] kubeadm version: v1.12.5+2.1.1.el7
[upgrade/versions] Found 1 potential version compatibility errors
but skipping since the --force flag is set:
   - There are kubelets in this cluster that are too old that have these versions [v1.9.11+2.1.1.el7]
[upgrade/prepull] Will prepull images for components [kube-apiserver kube-controller-manager kube-scheduler etcd]
[upgrade/prepull] Prepulling image for component etcd.
[upgrade/prepull] Prepulling image for component kube-apiserver.
[upgrade/prepull] Prepulling image for component kube-scheduler.
[apiclient] Found 0 Pods for label selector k8s-app=upgrade-prepull-kube-scheduler
[apiclient] Found 0 Pods for label selector k8s-app=upgrade-prepull-kube-scheduler
[apiclient] Found 1 Pods for label selector k8s-app=upgrade-prepull-kube-controller-manager
[apiclient] Found 1 Pods for label selector k8s-app=upgrade-prepull-kube-apiserver
Upgrading the Master Node from 1.1.9 to 1.1.12

[apiclient] Found 1 Pods for label selector k8s-app=upgrade-prepull-kube-scheduler
[apiclient] Found 0 Pods for label selector k8s-app=upgrade-prepull-etcd
[apiclient] Found 1 Pods for label selector k8s-app=upgrade-prepull-kube-controller-manager
[apiclient] Found 1 Pods for label selector k8s-app=upgrade-prepull-kube-apiserver
[apiclient] Found 1 Pods for label selector k8s-app=upgrade-prepull-kube-scheduler
[apiclient] Found 1 Pods for label selector k8s-app=upgrade-prepull-etcd

[upgrade/prepull] Prepulled image for component kube-apiserver.
[upgrade/prepull] Prepulled image for component etcd.

[upgrade/apply] Upgrading your Static Pod-hosted control plane to version "v1.12.5"...
Static pod: kube-apiserver-master.example.com hash: 0221796c26ebe3d6f237a7256d5a5fa36
Static pod: kube-scheduler-master.example.com hash: e0549b904166a0ec7afcdaf337ab1e0
Static pod: etcd-master.example.com hash: 8a68f8a24bf031e2027cc6d528ce6efe
[apiclient] Found 1 Pods for label selector k8s-app=upgrade-prepull-etcd
[apiclient] Found 1 Pods for label selector k8s-app=upgrade-prepull-kube-controller-manager
[apiclient] Found 1 Pods for label selector k8s-app=upgrade-prepull-kube-scheduler
[apiclient] Found 1 Pods for label selector k8s-app=upgrade-prepull-etcd

[upgrade/prepull] Successfully prepulled the images for all the control plane components
[upgrade/apply] Upgrading your Static Pod-hosted control plane to version "v1.12.5"...

Static pod: kube-apiserver-master.example.com hash: 0221796c26ebe3d6f237a7256d5a5fa36
Static pod: kube-scheduler-master.example.com hash: e0549b904166a0ec7afcdaf337ab1e0
Static pod: etcd-master.example.com hash: 8a68f8a24bf031e2027cc6d528ce6efe
Static pod: etcd-master.example.com hash: 8a68f8a24bf031e2027cc6d528ce6efe
Static pod: etcd-master.example.com hash: 7a68f8a24bf031e2027cc6d528ce6efe
Static pod: etcd-master.example.com hash: 7a68f8a24bf031e2027cc6d528ce6efe
Static pod: etcd-master.example.com hash: 7a68f8a24bf031e2027cc6d528ce6efe
[apiclient] Found 1 Pods for label selector component=etcd
[apiclient] Found 1 Pods for label selector component=kube-apiserver

[util/etcd] Waiting 0s for initial delay
[util/etcd] Attempting to see if all cluster endpoints are available 1/10
[util/etcd] Writing new Static Pod manifests to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests665746710"
[controlplane] wrote Static Pod manifest for component kube-apiserver
to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests665746710/kube-apiserver.yaml"
[controlplane] wrote Static Pod manifest for component kube-controller-manager
to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests665746710/kube-controller-manager.yaml"
[controlplane] wrote Static Pod manifest for component kube-scheduler
to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests665746710/kube-scheduler.yaml"
[controlplane] wrote Static Pod manifest for component k8s-control-plane
to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests665746710/k8s-control-plane.yaml"
[controlplane] wrote Static Pod manifest for component kube-apiserver
to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests665746710/kube-apiserver.yaml"

[util/etcd] Waiting 0s for initial delay
[util/etcd] Attempting to see if all cluster endpoints are available 1/10
[util/etcd] Writing new Static Pod manifests to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests665746710"
[controlplane] wrote Static Pod manifest for component kube-apiserver
to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests665746710/kube-apiserver.yaml"
[controlplane] wrote Static Pod manifest for component kube-controller-manager
to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests665746710/kube-controller-manager.yaml"
[controlplane] wrote Static Pod manifest for component kube-scheduler
to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests665746710/kube-scheduler.yaml"
[controlplane] wrote Static Pod manifest for component k8s-control-plane
to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests665746710/k8s-control-plane.yaml"
[controlplane] wrote Static Pod manifest for component kube-apiserver
to "/etc/kubernetes/tmp/kubeadm-upgraded-manifests665746710/kube-apiserver.yaml"

[apiclient] Found 1 Pods for label selector component=etcd
[apiclient] Found 1 Pods for label selector component=kube-apiserver

[upgrade/staticpods] Waiting for the kubelet to restart the component
[upgrade/staticpods] This might take a minute or longer depending on the component/version gap (timeout 5m0s)
Static pod: kube-apiserver-master.example.com hash: 7c19bbee52e8a857c9e75551339951b7
Static pod: kube-api-server.example.com hash: 7c19bbee52e8a857c9e75551339951b7
Static pod: kube-api-server.example.com hash: 7c19bbee52e8a857c9e75551339951b7
Static pod: kube-api-server.example.com hash: 7c19bbee52e8a857c9e75551339951b7
Static pod: kube-api-server.example.com hash: 5c6ceef93d0a8c04d3316e6a4b6a7
[apiclient] Found 1 Pods for label selector component=etcd
[apiclient] Found 1 Pods for label selector component=kube-apiserver

[apiclient] Found 0 Pods for label selector k8s-app=upgrade-prepull-kube-scheduler
[apiclient] Found 0 Pods for label selector k8s-app=upgrade-prepull-etcd
[apiclient] Found 0 Pods for label selector k8s-app=upgrade-prepull-kube-controller-manager
[apiclient] Found 0 Pods for label selector k8s-app=upgrade-prepull-kube-apiserver
[apiclient] Found 0 Pods for label selector k8s-app=upgrade-prepull-kube-scheduler
[apiclient] Found 0 Pods for label selector k8s-app=upgrade-prepull-etcd
[apiclient] Found 0 Pods for label selector k8s-app=upgrade-prepull-kube-scheduler
[apiclient] Found 0 Pods for label selector k8s-app=upgrade-prepull-etcd

[upgrade/prepull] Prepulled image for component kube-apiserver.
Upgrading the Master Node from 1.1.9 to 1.1.12

[upgrade/staticpods] Moved new manifest to "/etc/kubernetes/manifests/kube-scheduler.yaml" and backed up old manifest to "/etc/kubernetes/tmp/kubeadm-backup-manifests-2019-01-09-07-34-07/kube-scheduler.yaml"
[upgrade/staticpods] Waiting for the kubelet to restart the component
[upgrade/staticpods] This might take a minute or longer depending on the component/version gap (timeout 5m0s)
Static pod: kube-scheduler-master.example.com hash: e0549b9041665ae07cf4acdf337able0
Static pod: kube-scheduler-master.example.com hash: 68edd49aba200239d15bc7eb4e99bee7
[apiclient] Found 1 Pods for label selector component=kube-scheduler
[upgrade/staticpods] Component "kube-scheduler" upgraded successfully!

[uploadconfig] storing the configuration used in ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubelet-config-1.12" in namespace kube-system
with the configuration for the kubelets in the cluster
[kubelet] Downloading configuration for the kubelet from the "kubelet-config-1.12" ConfigMap in the kube-system namespace
[kubelet] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"

[patchnode] Uploading the CRI Socket information "/var/run/docker.sock" to the Node API object "master.example.com" as an annotation
[bootstraptoken] configured RBAC rules to allow Node Bootstrap tokens to post CSRs in order for nodes to get long term certificate credentials
[bootstraptoken] configured RBAC rules to allow the csrapprover controller automatically approve CSRs from a Node Bootstrap Token
[bootstraptoken] configured RBAC rules to allow certificate rotation for all node client certificates in the cluster
[addons] Applied essential addon: kube-dns
[addons] Applied essential addon: kube-proxy

[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.12.5". Enjoy!

[upgrade/kubelet] Now that your control plane is upgraded, please proceed with upgrading your kubelets if you haven’t already done so. Upgrading kubelet and kubectl now ...
Checking kubelet and kubectl RPM ...
[INFO] yum install -y kubelet-1.12.5-2.1.1.el7.x86_64
Loaded plugins: langpacks, ulninfo
Resolving Dependencies

--- Running transaction check
--- Package kubelet.x86_64 0:1.11.3-2.0.2.el7 will be updated
--- Package kubelet.x86_64 0:1.12.5-2.1.1.el7 will be an update
--- Finished Dependency Resolution

Dependencies Resolved

<table>
<thead>
<tr>
<th>Package</th>
<th>Arch</th>
<th>Version</th>
<th>Repository</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>kubelet</td>
<td>x86_64</td>
<td>1.12.5-2.1.1.el7</td>
<td>ol7_addons</td>
<td>19 M</td>
</tr>
</tbody>
</table>

Transaction Summary

Upgrade 1 Package

Total download size: 19 M
Downloaded packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.
kubelet-1.12.5-2.1.1.el7.x86_64.rpm
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction

Updating : kubelet-1.12.5-2.1.1.el7.x86_64 1/2
Cleanup : kubelet-1.11.3-2.0.2.el7.x86_64 2/2
Verifying : kubelet-1.12.5-2.1.1.el7.x86_64 1/2
Upgrading the Master Node from 1.1.9 to 1.1.12

Verifying : kubelet-1.11.3-2.0.2.el7.x86_64 2/2

Updated:
  kubelet.x86_64 0:1.12.5-2.1.1.el7

Complete!
[INFO] yum install -y kubectl-1.12.5-2.1.1.el7.x86_64
Loaded plugins: langpacks, ulninfo
Resolving Dependencies
--> Running transaction check
---> Package kubectl.x86_64 0:1.11.3-2.0.2.el7 will be updated
---> Package kubectl.x86_64 0:1.12.5-2.1.1.el7 will be an update
---> Finished Dependency Resolution

Dependencies Resolved

Package Arch Version Repository Size
===================================================================================
Updating:
  kubectl x86_64 1.12.5-2.1.1.el7 ol7_addons 7.7 M

Transaction Summary
===================================================================================
Upgrade  1 Package

Total download size: 7.7 M
Downloading packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.
kubectl-1.12.5-2.1.1.el7.x86_64.rpm | 7.7 MB 00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Updating : kubectl-1.12.5-2.1.1.el7.x86_64 1/2
  Cleanup : kubectl-1.11.3-2.0.2.el7.x86_64 2/2
  Verifying : kubectl-1.12.5-2.1.1.el7.x86_64 1/2
  Verifying : kubectl-1.11.3-2.0.2.el7.x86_64 2/2

Updated:
  kubectl.x86_64 0:1.12.5-2.1.1.el7

Complete!
Unable to connect to the server: net/http: TLS handshake timeout
[INSTALLING DASHBOARD NOW]

Installing kubernetes-dashboard ...

Kubernetes version: v1.12.5 and dashboard yaml file:
/usr/local/share/kubeadm/kubernetes-dashboard-self-certs.yaml
Error from server (AlreadyExists): error when creating
  "/tmp/dashboard.yaml": secrets "kubernetes-dashboard-certs" already exists
Error from server (AlreadyExists): error when creating
  "/tmp/dashboard.yaml": serviceaccounts "kubernetes-dashboard" already exists
Error from server (AlreadyExists): error when creating
  "/tmp/dashboard.yaml": roles.rbac.authorization.k8s.io
"kubernetes-dashboard-minimal" already exists
Error from server (AlreadyExists): error when creating
  "/tmp/dashboard.yaml": rolebindings.rbac.authorization.k8s.io
"kubernetes-dashboard-minimal" already exists
Error from server (AlreadyExists): error when creating
  "/tmp/dashboard.yaml": deployments.apps "kubernetes-dashboard" already exists
Error from server (AlreadyExists): error when creating
  "/tmp/dashboard.yaml": services "kubernetes-dashboard" already exists
Restarting kubectl-proxy.service ...
[INFO] Upgrading master node done successfully
Upgrading the Master Node from 1.1.9 to 1.1.12

6. Because the flannel service that Oracle Linux Container Services for use with Kubernetes 1.1.12 depends on is not upgraded automatically by the specialized upgrade script, ensure you upgrade separately, for example:

```bash
# kubeadm-setup.sh upgrade --flannel
```

7. If the master node upgrade fails, roll back as follows:

```bash
# kubeadm-upgrade.sh restore /backups/master-backup-v1.9.11-0-1546953894.tar
```

8. If the script completes successfully, create a fresh backup on your new Oracle Linux Container Services for use with Kubernetes 1.1.12 master node by using `kubeadm-setup.sh backup`.

See Section 4.3, “Cluster Backup and Restore”.

You can read the full upgrade log in `/var/log/kubeadm-upgrade`. After completing the master node upgrade, you can upgrade the packages for Oracle Container Services for use with Kubernetes on each worker node.
2.5.2 Upgrading Worker Nodes from 1.1.9 to 1.1.12

Only upgrade worker nodes after the master node has completed the upgrade process, as described in Section 2.5.1, “Upgrading the Master Node from 1.1.9 to 1.1.12”.

Important

You must perform several manual steps to complete the upgrade of a worker node. These steps involve draining the node prior to upgrade to prevent the cluster from scheduling or starting any pods on the node while it is being upgraded. The drain process deletes any running pods from the node. If there is local storage configured, the drain process errors out so that you have the opportunity to determine whether or not you need to back up local data.

When the upgrade is complete, you can uncordon the worker node so that pods are able to resume on this node.

To upgrade a worker node, perform the following steps:

1. Drain the worker node by running the following command from the master node:

   ```
   $ kubectl drain worker1.example.com --ignore-daemonsets
   ```

   where `worker1.example.com` is the hostname of the worker node that you wish to upgrade.

   If local storage is configured for the node, the drain process may generate an error. The following example output shows a node, using local storage, that fails to drain:

   ```
   node/worker1.example.com cordoned
   error: unable to drain node "worker1.example.com", aborting command...
   There are pending nodes to be drained:
   worker1.example.com
   error: pods with local storage (use --delete-local-data to override):
   carts-74f4558cb8-c8p8x,
   carts-db-7fcd99bc79-c5pkx, orders-787bf5b89f-nt92j, orders-db-775655b675-rhlp7,
   shipping-5bd69f4cc-twvtf, user-db-5f9d89bbbb-7t8sk
   ```

   In the case where a node fails to drain, determine whether to follow any procedure to back up local data and restore it later or whether you can proceed and delete the local data directly. After any backup files have been made, you can rerun the command with the `--delete-local-data` switch to force the removal of the data and drain the node. For example, on the master node, run:

   ```
   $ kubectl drain worker1.example.com --ignore-daemonsets --delete-local-data
   ```

   node/worker1.example.com cordoned already cordoned
   WARNING: Ignoring DaemonSet-managed pods: kube-flannel-ds-xrszk, kube-proxy-79p9x;
   Deleting pods with local storage:
   carts-74f4558cb8-g2fdw, orders-db-775655b675-gfggs,
   user-db-5f9d89bbbb-k78sk
   pod "user-db-5f9d89bbbb-k78sk" evicted
   pod "rabbitmq-96d887875-lxm5f" evicted
   pod "orders-db-775655b675-gfggs" evicted
   pod "catalogue-676d4b9f7-c-1v9bf" evicted
   pod "payment-75f5b467f-srhrq" evicted
   pod "carts-74f4558cb8-g2fdw" evicted
   node "kubernetes-worker1" drained
   ```

2. Check that the worker node is unable to accept any further scheduling by running the following command on the master node:

   ```
   $ kubectl get nodes
   ```

   Note that a node that has been drained should have its status set to `SchedulingDisabled`. 
3. If you are using the Oracle Container Registry to obtain images, log in.

Follow the instructions in Section 2.2.5, “Oracle Container Registry Requirements”. Note that if images are updated on the Oracle Container Registry, you may be required to accept the Oracle Standard Terms and Restrictions again before you are able to perform the upgrade. If you are using one of the Oracle Container Registry mirrors, see Section 2.2.5.1, “Using an Oracle Container Registry Mirror” for more information. If you have configured a local registry, you may need to set the KUBE_REPO_PREFIX environment variable to point to the appropriate registry. You may also need to update your local registry with the most current images for the version that you are upgrading to. See Section 2.2.5.2, “Setting Up an Optional Local Registry” for more information.

4. Run the `kubeadm-upgrade.sh upgrade` command as root on the worker node:

```
# kubeadm-upgrade.sh upgrade
-- Running upgrade script--
Number of cpu present in this system 2
Total memory on this system: 7710MB
Space available on the mount point /var/lib/docker: 44GB
Space available on the mount point /var/lib/kubelet: 44GB
kubeadm version 1.9
kubectl version 1.9
kubelet version 1.9
ol7_addons repo enabled
[WARNING] This action will upgrade this node to latest version
[WARNING] The cluster will be upgraded through intermediate versions which are unsupported
[WARNING] You must take backup before upgrading the cluster as upgrade may fail
Please select 1 (continue) or 2 (abort) :
1) continue
2) abort
#? 1
Upgrading worker node
Updating kubeadm package
Checking access to container-registry.oracle.com/kubernetes for update
Trying to pull repository container-registry.oracle.com/kubernetes/kube-proxy ...  
v1.12.5: Pulling from container-registry.oracle.com/kubernetes/kube-proxy
Digest: sha256:9eba681b56e15078cb499a3360f138cc16987cf5aea06593f77d0881af6badbe
Status: Image is up to date for container-registry.oracle.com/kubernetes/kube-proxy:v1.12.5
Upgrading kubeadm to latest version
Resolving Dependencies
--> Running transaction check
---+ Package kubeadm.x86_64 0:1.9.11-2.1.1.el7 will be updated
---+ Package kubeadm.x86_64 0:1.12.5-2.1.1.el7 will be an update
--> Finished Dependency Resolution

Transaction Summary
===============================================================================
Package  Arch   Version          Repository      Size
===============================================================================
Updating:
  kubeadm  x86_64  1.12.5-2.1.1.el7  ol7_addons  7.3 M

Total download size: 7.3 M
```

Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Upgrading kubeadm forcefully from version earlier that 1.11
Upgrading Worker Nodes from 1.1.9 to 1.1.12

Upgrading : kubeadm-1.12.5-2.1.1.el7.x86_64 1/2
Cleanup : kubeadm-1.9.11-2.1.1.el7.x86_64 2/2
Verifying : kubeadm-1.12.5-2.1.1.el7.x86_64 1/2
Verifying : kubeadm-1.9.11-2.1.1.el7.x86_64 2/2

Updated:
  kubeadm.x86_64 0:1.12.5-2.1.1.el7

Complete!
Upgrading kubelet and kubectl now ...
Checking kubelet and kubectl RPM ...
[INFO] yum install -y kubelet=1.12.5-2.1.1.el7.x86_64
Loaded plugins: langpacks, ulninfo
Resolving Dependencies
---> Running transaction check
----> Package kubelet.x86_64 0:1.9.11-2.1.1.el7 will be updated
----> Package kubelet.x86_64 0:1.12.5-2.1.1.el7 will be an update
---> Finished Dependency Resolution

Dependencies Resolved

==========================================================================================
Package    Arch           Version                       Repository             Size
==========================================================================================
Updating:
  kubelet    x86_64        1.12.5-2.1.1.el7              ol7_addons              19 M

Transaction Summary
==========================================================================================
Upgrade  1 Package
Total download size: 19 M
Downloading packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.
kubelet-1.12.5-2.1.1.el7.x86_64.rpm                               |  19 MB  00:00:01
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Updating : kubelet-1.12.5-2.1.1.el7.x86_64                      1/2
  Cleanup : kubelet-1.9.11-2.1.1.el7.x86_64                      2/2
  Verifying : kubelet-1.12.5-2.1.1.el7.x86_64                    1/2
  Verifying : kubelet-1.9.11-2.1.1.el7.x86_64                    2/2

Updated:
  kubelet.x86_64 0:1.12.5-2.1.1.el7

Complete!
[INFO] yum install -y kubectl=1.12.5-2.1.1.el7.x86_64
Loaded plugins: langpacks, ulninfo
Resolving Dependencies
---> Running transaction check
----> Package kubectl.x86_64 0:1.9.11-2.1.1.el7 will be updated
----> Package kubectl.x86_64 0:1.12.5-2.1.1.el7 will be an update
---> Finished Dependency Resolution

Dependencies Resolved

==========================================================================================
Package    Arch             Version                       Repository           Size
==========================================================================================
Updating:
  kubectl       x86_64       1.12.5-2.1.1.el7              ol7_addons         7.7 M

Transaction Summary
==========================================================================================
Upgrade  1 Package
Updating to Errata Releases

Total download size: 7.7 M
Downloading packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.
kubectl-1.12.5-2.1.1.el7.x86_64.rpm | 7.7 MB 00:00:00
Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
  Updating  : kubectl-1.12.5-2.1.1.el7.x86_64 1/2
  Cleanup   : kubectl-1.9.11-2.1.1.el7.x86_64 2/2
  Verifying : kubectl-1.12.5-2.1.1.el7.x86_64 1/2
  Verifying : kubectl-1.9.11-2.1.1.el7.x86_64 2/2

Updated:
  kubectl.x86_64 0:1.12.5-2.1.1.el7

Complete!
[kubelet] Downloading configuration for the kubelet from
the "kubelet-config-1.12" ConfigMap in the kube-system namespace
[kubelet] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[upgrade] The configuration for this node was successfully updated!
[upgrade] Now you should go ahead and upgrade the kubelet package
using your package manager.
[WORKER NODE UPGRADED SUCCESSFULLY]

Note that you are warned that the upgrade affects the node's availability temporarily. You must confirm
that you wish to continue to complete the upgrade.

The kubelet service and all running containers are restarted automatically after upgrade.

5. Uncordon the worker node so that it is able to schedule new nodes, as required. On the master node, run:

$ kubectl uncordon worker1.example.com
node/worker1.example.com uncordoned

where worker1.example.com is the hostname of the worker node that you have just upgraded.

6. When you have finished the upgrade process, check that the nodes are all running the expected
version as follows:

$ kubectl get nodes
NAME       STATUS  ROLES       AGE   VERSION
master.example.com   Ready   master  1h   v1.12.5+2.1.1.el7
worker1.example.com  Ready    <none>  1h   v1.12.5+2.1.1.el7
worker2.example.com  Ready    <none>  1h   v1.12.5+2.1.1.el7

2.6 Updating to Errata Releases

Updates for Oracle Container Services for use with Kubernetes are released on the Oracle Linux yum
server and on ULN.

Warning

The update process that is described here only applies for updates to errata releases that provide minor updates and security patches for existing installations.

Oracle does not support upgrading existing clusters that are created by using Oracle Container Services for use with Kubernetes 1.1.9 to 1.1.12 with the kubeadm-setup.sh script. You must use the kubeadm-upgrade.sh script, as described in Section 2.5, "Upgrading 1.1.9 to 1.1.12".
These instructions work on hosts that are booting from UEK R4, but it is recommended that hosts currently running UEK R4 are upgraded to use UEK R5 to facilitate future upgrades, where KubeDNS is deprecated.

The update process requires that you first update the master node in your cluster, and then update each of the worker nodes. Update of the master node is scripted so that the pre-requisite checks, validation, and reconfiguration are automated. It is good practice to make a backup file for your cluster before update. See Section 2.6.1, “Updating the Master Node”.

After the master node is updated, you can update each worker node, as described in Section 2.6.2, “Updating Worker Nodes”.

### Important

Oracle does not support any upgrade from a preview release to a stable and supported release.

Oracle also does not support upgrading existing single master node clusters built with the `kubeadm-setup.sh` script to High Availability clusters. You must build and manage High Availability clusters by using the `kubeadm-ha-setup` utility.

## 2.6.1 Updating the Master Node

You must update the master node in your cluster before you update worker nodes. The `kubeadm-setup.sh` upgrade command is used on the master node to complete the necessary steps to prepare and update the cluster. The following steps describe how to update the master node.

### Important

Before you perform any update operations, make a backup file for your cluster at its current version. After you update the `kubeadm` package, any backup files that you make are not backward compatible, and if you revert to an earlier version of Oracle Container Services for use with Kubernetes, the restore operation may fail to successfully load your backup file. See Section 4.3, “Cluster Backup and Restore” for more information.

### Steps to update the master node

1. On the master node, update the `kubeadm` package first:

   ```bash
   # yum update kubeadm
   ```

2. If you are using the Oracle Container Registry to obtain images, log in.

   Follow the instructions in Section 2.2.5, “Oracle Container Registry Requirements”. Note that if images are updated on the Oracle Container Registry, you may be required to accept the Oracle Standard Terms and Restrictions again before you are able to perform the update. If you are using one of the Oracle Container Registry mirrors, see Section 2.2.5.1, “Using an Oracle Container Registry Mirror” for more information. If you have configured a local registry, you may need to set the `KUBE_REPO_PREFIX` environment variable to point to the appropriate registry. You may also need to update your local registry with the most current images for the version that you are upgrading to. See Section 2.2.5.2, “Setting Up an Optional Local Registry” for more information.

3. Ensure that you open any new firewall ports in Section 2.2.7, “Firewall and iptables Requirements”.

4. Create a pre-update backup file. In the event that the update does not complete successfully, the backup can revert back to the configuration of your cluster prior to update.
Updating the Master Node

# kubeadm-setup.sh stop
Stopping kubelet now ...
Stopping containers now ...

# kubeadm-setup.sh backup /backups
Creating backup at directory /backup ...
Using 3.2.24
Checking if container-registry.oracle.com/kubernetes/etcd:3.2.24 is available

d05a0ef2bea8d0e131f1cb5391d8878a5537f86384887ae31694689bc6d57f5
/var/run/kubeadm/backup/etcd-backup-1543581013.tar
9aa26d014d2c2f7a7343b0d4b54e666b7ee5685c08df7047555eb7e0e6f
/var/run/kubeadm/backup/k8s-master-0-1543581013.tar
Backup is successfully stored at /backup/master-backup-v1.12.5-2-1543581013.tar ...
You can restart your cluster now by doing:

# kubeadm-setup.sh restart

# kubeadm-setup.sh restart
Restarting containers now ...
Checking if env is ready ...
Checking whether docker can pull busybox image ...
Checking access to container-registry.oracle.com/kubernetes ...
Trying to pull repository container-registry.oracle.com/kubernetes/pause ...
3.1: Pulling from container-registry.oracle.com/kubernetes/pause
Digest: sha256:802ef89b9eb7e874e6cfc79ed900b63b0b84a05ca09c8293378ac0261e49
Status: Image is up to date for container-registry.oracle.com/kubernetes/pause:3.1
Checking firewalld settings ...
Checking iptables default rule ...
Checking br_netfilter module ...
Checking sysctl variables ...
Restarting kubelet ...
Waiting for node to restart ...

5. Run the `kubeadm-setup.sh upgrade` command as `root` on the master node. The script prompts you to continue with the update and warns you to make a backup file before you continue. Enter 1 to continue.

# kubeadm-setup.sh upgrade
Checking whether api-server is using image lower than 1.12
[WARNING] Please make sure that you have performed backup of the cluster before upgrading
Please select 1 (continue) or 2 (abort) :
1) continue
2) abort

[upgrade] Making sure the cluster is healthy:
[upgrade/config] Reading configuration options from a file: /var/run/kubeadm/kubeadm-cfg
[upgrade/version] You have chosen to change the cluster version to "v1.12.5-2"
[upgrade/versions] Cluster version: v1.12.5+2.1.1.el7
[upgrade/versions] kubeversion: v1.12.5+2.1.1.el7
[upgrade/prepull] Will prepull images for components [kube-apiserver kube-controller-manager kube-scheduler]
[upgrade/apply] Upgrading your Static Pod-hosted control plane to version "v1.12.5-2"
[upgrade/staticpods] Writing new Static Pod manifests to
"/etc/kubernetes/tmp/kubeadm-upgraded-manifests120255399"
The upgrade command performs a health check on the cluster, validates the existing configuration, and then pulls the necessary images that are required to update the cluster. All of the controlplane components for the cluster are updated and certificates and tokens are configured to ensure that all cluster components on all nodes are able to continue to function after update.

After these components have been updated, the kubelet and kubectl packages are updated automatically.

6. If you are prompted by the following message, it is an indication that you need to update the flannel component manually:

[INFO] Flannel is not upgraded yet. Run 'kubeadm-setup.sh upgrade --flannel' to upgrade flannel
Re-run the `kubeadm-setup.sh upgrade` with the `--flannel` flag to ensure that you have fully upgraded your cluster:

```
# kubeadm-setup.sh upgrade --flannel
```

After you have completed the master node upgrade, you can upgrade the packages for Oracle Container Services for use with Kubernetes on each worker node.

### 2.6.2 Updating Worker Nodes

Only update worker nodes after the master node has completed the update process, as described in Section 2.6.1, “Updating the Master Node”.

---

**Important**

You must perform several manual steps to complete the update of a worker node. These steps involve draining the node prior to update to prevent the cluster from scheduling or starting any pods on the node while it is being updated. The drain process deletes any running pods from the node. If there is local storage configured, the drain process errors out so that you have the opportunity to determine whether or not you need to back up local data.

When the update is complete you can uncordon the worker node so that pods are able to resume on this node.

To update a worker node, perform the following steps:

1. Drain the worker node by running the following command from the master node:

   ```
   $ kubectl drain worker1.example.com --ignore-daemonsets
   
   where `worker1.example.com` is the hostname of the worker node that you wish to update.
   
   If local storage is configured for the node, the drain process might generate an error. The following example output shows a node, using local storage, that fails to drain:
   
   ```node/work1.example.com.cordon
   error: unable to drain node "worker1.example.com", aborting command...
   
   There are pending nodes to be drained:
   
   `worker1.example.com`
   
   error: pods with local storage (use --delete-local-data to override): carts-74f4558cb8-c8p8x, carts-db-7fcddfb79-c5pxx, orders-787bf5b89f-nt9zj, orders-db-775655b675-rhlp7, shipping-5bd69fb4cc-twvtf, user-db-5f9d89bb8b-7t85k
   ```

   In the case where a node fails to drain, determine whether you should follow any procedure to back up local data and restore it later or whether you can proceed and delete the local data directly. After any backup files have been made, you can rerun the command with the `--delete-local-data` switch to force the removal of the data and drain the node, for example:

   ```
   $ kubectl drain worker1.example.com --ignore-daemonsets --delete-local-data
   
   node/work1.example.com.already.cordon
   WARNING: Ignoring DaemonSet-managed pods: kube-flannel-ds-xrszk, kube-proxy-7g9px;
   Deleting pods with local storage: carts-74f4558cb8-g2f4w, orders-db-775655b675-gfggs, user-db-5f9d89bb8b-k78sk
   pod "user-db-5f9d89bb8b-k78sk" evicted
   pod "rabbitmq-96d887875-lxm5f" evicted
   pod "orders-db-775655b675-gfggs" evicted
   pod "catalogue-676d4b9?c-1wfb" evicted
   ```
2. Check that the worker node is unable to accept any further scheduling by running the following command on the master node:

```
$ kubectl get nodes
```

A node that has been drained should have its status set to `SchedulingDisabled`.

3. Update the packages on the worker node by using a standard `yum update` command. To specifically update only those packages required for Oracle Container Services for use with Kubernetes, on the worker node, run the following command as `root`:

```
# yum update kubeadm
```

4. If you are using the Oracle Container Registry to obtain images, log in.

Follow the instructions in Section 2.2.5, “Oracle Container Registry Requirements”. Note that if images are updated on the Oracle Container Registry, you may be required to accept the Oracle Standard Terms and Restrictions again before you are able to perform the update. If you are using one of the Oracle Container Registry mirrors, see Section 2.2.5.1, “Using an Oracle Container Registry Mirror” for more information. If you have configured a local registry, you may need to set the `KUBE_REPO_PREFIX` environment variable to point to the appropriate registry. You may also need to update your local registry with the most current images for the version that you are upgrading to. See Section 2.2.5.2, “Setting Up an Optional Local Registry” for more information.

5. When the `yum update` process completes, run the `kubeadm-setup.sh upgrade` command as `root` on the worker node. You are warned that the update affects the node's availability temporarily. Confirm that you wish to continue to complete the update:

```
# kubeadm-setup.sh upgrade
[WARNING] Upgrade will affect this node's application(s) availability temporarily
Please select 1 (continue) or 2 (abort) : 1

Checking access to container-registry.oracle.com/kubernetes for update
v1.12.5-2: Pulling from kubernetes/kube-proxy-amd64
Digest: sha256:f525d06eeb7f21c55550b1da8cee4720e3eb9ffee8976db357f49eddd04c6d0
Status: Image is up to date for
container-registry.oracle.com/kubernetes/kube-proxy-amd64:v1.12.5-2
Restarting containers ....
[NOE UPGRADED SUCCESSFULLY]
```

The `kubelet` service and all of the running containers are restarted automatically after the update.

6. Uncordon the worker node so that it is able to schedule new nodes, as required, by running the following command on the master node:

```
$ kubectl uncordon worker1.example.com
```

where `worker1.example.com` is the hostname of the worker node that you have just updated.

7. After the update process has completed, run the following command on the master node to check that all of the nodes are running the expected version:

```
$ kubectl get nodes
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
<th>ROLES</th>
<th>AGE</th>
<th>VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node</td>
<td>State</td>
<td>Role</td>
<td>Age</td>
<td>Version</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------</td>
<td>-------</td>
<td>-----</td>
<td>------------------</td>
</tr>
<tr>
<td>master.example.com</td>
<td>Ready</td>
<td>master</td>
<td>1h</td>
<td>v1.12.5+2.1.1.el7</td>
</tr>
<tr>
<td>worker1.example.com</td>
<td>Ready</td>
<td>&lt;none&gt;</td>
<td>1h</td>
<td>v1.12.5+2.1.1.el7</td>
</tr>
<tr>
<td>worker2.example.com</td>
<td>Ready</td>
<td>&lt;none&gt;</td>
<td>1h</td>
<td>v1.12.5+2.1.1.el7</td>
</tr>
</tbody>
</table>
Chapter 3 Installing High Availability Oracle Container Services for use with Kubernetes

Table of Contents

3.1 Overview .................................................................................................................. 45
3.2 Requirements ............................................................................................................ 46
  3.2.1 Yum or ULN Channel Subscription ................................................................. 46
  3.2.2 Requirement for Upgrading the Unbreakable Enterprise Kernel ..................... 46
  3.2.3 Resource Requirements .................................................................................. 47
  3.2.4 Docker Engine Requirements ......................................................................... 47
  3.2.5 Oracle Container Registry Requirements ..................................................... 48
  3.2.6 Network Time Service Requirements ............................................................ 50
  3.2.7 Firewall and iptables Requirements ................................................................ 50
  3.2.8 Network Requirements ................................................................................... 51
  3.2.9 SELinux Requirements ................................................................................... 52
  3.2.10 Requirements to Use Oracle Container Services for use with Kubernetes on Oracle Cloud Infrastructure ............................................................... 52
3.3 Setting Up the Master Cluster .................................................................................... 54
3.4 Setting Up a Worker Node ......................................................................................... 57
3.5 Upgrading ................................................................................................................... 58
  3.5.1 Updating the High Availability cluster ............................................................. 58

This chapter describes the steps required to install Kubernetes clusters using master nodes configured for high availability on Oracle Linux 7 hosts.

3.1 Overview

Kubernetes can be deployed with more than one replica of the required master node, and automated failover to those replicas, for the purpose of providing a more scalable and resilient service.

The kubeadm package provides the kubeadm utility, a tool designed to make the deployment of a Kubernetes cluster simple. Many users may find that using this tool directly, along with the upstream documentation, provides the maximum configuration flexibility.

Oracle provides the kubeadm-ha-setup tool in an additional kubeadm-ha-setup package to help new users install and configure a high availability deployment of Kubernetes with greater ease, regardless of whether it is hosted on bare metal, on a virtual machine, or out on the cloud. The tool handles checking that basic package requirements are in place, setting proxy and firewall requirements, configuring networking, and initializing a high availability master cluster configuration for the Kubernetes environment. The tool uses the kubeadm utility, but handles many additional configuration steps that can help new users get running with minimal effort.

The instructions provided here assume that you are new to Kubernetes and are using the provided kubeadm-ha-setup tool to deploy your cluster. This tool is developed and tested at Oracle and deployment using this tool is fully supported. Alternate configurations and deployment mechanisms are untested by Oracle.

Important

High availability clusters have resilience for one master node failure. If more than one master node fails then you will need to restore you master cluster from a backup file to avoid data loss.
3.2 Requirements

Kubernetes configured for high availability requires three nodes in the master cluster and at least one worker node.

Creating three master nodes ensures replication of configuration data between them through the distributed key store, etcd, so that your high availability cluster is resilient to a single node failing without any loss of data or uptime.

Placing each master node in a different Kubernetes zone can safeguard cluster availability in the event of a zone failure within the master cluster.

The following sections describe various requirements that must be met to install and configure Kubernetes clusters with high availability on Oracle Linux 7 hosts.

3.2.1 Yum or ULN Channel Subscription

To install all of the required packages to use Kubernetes, you must ensure that you are subscribed to the correct yum repositories or Unbreakable Linux Network (ULN) channels.

If your systems are registered with ULN, enable the ol7_x86_64_addons channel.

If you use the Oracle Linux yum server, enable the ol7_addons repository on each system in your deployment. You can do this easily using yum-config-manager:

```
# yum-config-manager --enable ol7_addons
```

For more information on the ol7_x86_64_addons channel, please see the ULN Channels section of the Oracle Linux Unbreakable Linux Network User’s Guide, available at:

https://docs.oracle.com/cd/E52668_01/E39381/html/ol_ol7chan_uln.html

Important
Oracle does not support Kubernetes on systems where the ol7_preview, ol7_developer or ol7_developer_EPEL repositories are enabled, or where software from these repositories is currently installed on the systems where Kubernetes runs. Even if you follow the instructions in this document, you may render your platform unsupported if these repositories or channels are enabled or software from these channels or repositories is installed on your system.

3.2.2 Requirement for Upgrading the Unbreakable Enterprise Kernel

Oracle Linux Container Services for use with Kubernetes 1.1.12 and later versions require that you configure the system to use the Unbreakable Enterprise Kernel Release 5 (UEK R5) and boot the system with this kernel. If you are using either UEK R4 or the Red Hat Compatible Kernel (RHCK), you must configure Yum to allow you to install UEK R5.

1. If your system is registered with the Unbreakable Linux Network (ULN), disable access to the ol7_x86_64_UEKR4 channel and enable access to the ol7_x86_64_UEKR5 channel.

If you use the Oracle Linux yum server, disable the ol7_UEKR4 repository and enable the ol7_UEKR5 repository. You can do this easily using yum-config-manager, if you have the yum-utils package installed:
2. Run the following command to upgrade the system to UEK R5:

```
# yum update
```

For information on how to make UEK R5 the default boot kernel, see the Oracle Linux Administrator's Guide for Release 7 at https://docs.oracle.com/cd/E52668_01/E54669/html/ol7-grub2_bootloader.html.

3. Reboot the system, selecting the UEK R5 kernel if this is not the default boot kernel.

```
# systemctl reboot
```

### 3.2.3 Resource Requirements

Each node in your cluster requires at least 2 GB of RAM and 2 or more CPUs to facilitate the use of `kubeadm` and any further applications that are provisioned using `kubectl`.

Also ensure that each node has a unique hostname, MAC address and product UUID as Kubernetes uses this information to identify and track each node in the cluster. You can verify the product UUID on each host with:

```
# dmidecode -s system-uuid
```

A storage volume with at least 5 GB free space must be mounted at `/var/lib/kubelet` on each node. For the underlying Docker engine an additional volume with at least 5 GB free space must be mounted on each node at `/var/lib/docker`.

### 3.2.4 Docker Engine Requirements

Kubernetes is used to manage containers running on a containerization platform deployed on several systems. On Oracle Linux, Kubernetes is currently only supported when used in conjunction with the Docker containerization platform. Therefore, each system in the deployment must have the Docker engine installed and ready to run. Support of Oracle Container Services for use with Kubernetes is limited to usage with the latest Oracle Container Runtime for Docker version available in the `ol7_addons` repository on the Oracle Linux yum server and in the `ol7_x86_64_addons` channel on ULN.

Please note that if you enable the `ol7_preview` repository, you may install a preview version of Oracle Container Runtime for Docker and your installation can no longer be supported by Oracle. If you have already installed a version of Docker from the `ol7_preview` repository, you should disable the repository and uninstall this version before proceeding with the installation.

Install, the Docker engine on all nodes in the cluster:

```
# yum install docker-engine
```

Enable the Docker service in `systemd` so that it starts on subsequent reboots and you should start the service before running the `kubeadm-setup.sh` tool.

```
# systemctl enable docker
# systemctl start docker
```

See the Oracle Linux Docker User's Guide for more information on installing and running the Docker engine. This is available at:

https://docs.oracle.com/cd/E52668_01/E87205/html/index.html
3.2.5 Oracle Container Registry Requirements

The images that are deployed by the kubeadm-ha-setu tool are hosted on the Oracle Container Registry. For the tool to be able to install the required components, you must perform the following steps:


2. Use the web interface to navigate to the Container Services business area and accept the Oracle Standard Terms and Restrictions for the Oracle software images that you intend to deploy. You are able to accept a global agreement that applies to all of the existing repositories within this business area. If newer repositories are added to this business area in the future, you may need to accept these terms again before performing upgrades.

3. Ensure that each of the systems that are used as nodes within the cluster are able to access https://container-registry.oracle.com and use the docker login command to authenticate against the Oracle Container Registry using the same credentials that you used to log into the web interface:

```bash
# docker login container-registry.oracle.com
```

The command prompts you for your user name and password.

Detailed information about the Oracle Container Registry is available in the Oracle Linux Docker User’s Guide at:

https://docs.oracle.com/cd/E52668_01/E87205/html/oracle-registry-server.html

3.2.5.1 Using an Oracle Container Registry Mirror

It is also possible to use any of the Oracle Container Registry mirror servers to obtain the correct images to set up Oracle Container Services for use with Kubernetes. The Oracle Container Registry mirror servers are located within the same data centers used for Oracle Cloud Infrastructure. More information about the Oracle Container Registry mirror servers is available in the Oracle Linux Docker User’s Guide at:

https://docs.oracle.com/cd/E52668_01/E87205/html/oracle-registry-server.html

Steps to use an alternate Oracle Container Registry mirror server follow:


2. On each node, use the docker login command to authenticate against the Oracle Container Registry mirror server using the same credentials that you used to log into the web interface:

```bash
# docker login container-registry-ash.oracle.com
```

The command prompts you for your user name and password.

3. After you have logged in, set the environment variable to use the correct registry mirror when you deploy Kubernetes:

```bash
# export KUBE_REPO_PREFIX=container-registry-ash.oracle.com/kubernetes
# echo 'export KUBE_REPO_PREFIX=container-registry-ash.oracle.com/kubernetes' > ~/.bashrc
```

If you are using Oracle Container Services for use with Kubernetes on Oracle Cloud Infrastructure, the kubeadm-ha-setu tool automatically detects the most appropriate mirror server to use and sets this environment variable for you so that you do not have to perform this step. If you manually set the
**Oracle Container Registry Requirements**

**3.2.5.2 Setting Up an Optional Local Registry**

If the systems that you are using for your Kubernetes cluster nodes do not have direct access to the Internet and are unable to connect directly to the Oracle Container Registry, you can set up a local Docker registry to perform this function. The *kubeadm-ha-setup* tool provides an option to change the registry that you use to obtain these images. Instructions to set up a local Docker registry are provided in the *Oracle Linux Docker User's Guide* at:

https://docs.oracle.com/cd/E52668_01/E87205/html/prepare-registry-server.html

When you have set up a local Docker registry, you must pull the images required to run Oracle Container Services for use with Kubernetes, tag these images and then push them to your local registry. The images must be tagged identically to the way that they are tagged in the Oracle Container Registry. The *kubeadm-ha-setup* matches version numbers during the setup process and cannot successfully complete many operations if it cannot find particular versions of images. To assist with this process, Oracle Container Services for use with Kubernetes provides the *kubeadm-registry.sh* tool in the *kubeadm* package.

To use the *kubeadm-registry.sh* tool to automatically pull images from the Oracle Container Registry, tag them appropriately, and push them to your local registry:

1. If you are using the Oracle Container Registry to obtain images, log in following the instructions in Section 2.2.5, “Oracle Container Registry Requirements”. If you are using one of the Oracle Container Registry mirrors, see Section 3.2.5.1, “Using an Oracle Container Registry Mirror” for more information.

2. Run the *kubeadm-registry.sh* tool with the required options:

   ```bash
   # kubeadm-registry.sh --to host.example.com:5000
   ```

   Substitute *host.example.com:5000* with the resolvable domain name and port by which your local Docker registry is available.

   You may optionally use the *--from* option to specify an alternate registry to pull the images from. You may also use the *--version* option to specify the version of Kubernetes images that you intend to host. For example:

   ```bash
   # kubeadm-registry.sh --to host.example.com:5000 --from \n   container-registry-ash.oracle.com/kubernetes --version 1.12.0
   ```

**Important**

If you are upgrading your environment and you intend to use a local registry, you must make sure that you have the most recent version of the images required to run Oracle Container Services for use with Kubernetes. You can use the *kubeadm-registry.sh* tool to pull the correct images and to update your local registry before running the upgrade on the master node.

After your local Docker registry is installed and configured and the required images have been imported, you must set the environment variable that controls which registry server the *kubeadm-ha-setup* tool uses. On each of the systems where you intend to run the *kubeadm-ha-setup* tool run the following commands:

```bash
# export KUBE_REPO_PREFIX="local-registry.example.com:5000/kubernetes"
# echo 'export KUBE_REPO_PREFIX="local-registry.example.com:5000/kubernetes"' > ~/.bashrc
```
Substitute local-registry.example.com with the IP address or resolvable domain name of the host on which your local Docker registry is configured.

### 3.2.6 Network Time Service Requirements

As a clustering environment, Kubernetes requires that system time is synchronized across each node within the cluster. Typically, this can be achieved by installing and configuring an NTP daemon on each node. You can do this in the following way:

1. Install the ntp package, if it is not already installed:

   ```bash
   # yum install ntp
   ```

2. Edit the NTP configuration in `/etc/ntp.conf`. Your requirements may vary. If you are using DHCP to configure the networking for each node, it is possible to configure NTP servers automatically. If you have not got a locally configured NTP service that your systems can sync to, and your systems have Internet access, you can configure them to use the public pool.ntp.org service. See [https://www.ntppool.org/](https://www.ntppool.org/).

3. Ensure that NTP is enabled to restart at boot and that it is started before you proceed with your Kubernetes installation. For example:

   ```bash
   # systemctl start ntpd
   # systemctl enable ntpd
   ```

   Note that systems running on Oracle Cloud Infrastructure are configured to use the `chronyd` time service by default, so there is no requirement to add or configure NTP if you are installing into an Oracle Cloud Infrastructure environment.

   For information on configuring a Network Time Service, see the [Oracle Linux Administrator's Guide for Release 7](https://docs.oracle.com/cd/E52668_01/E54669/html/ol7-nettime.html).

### 3.2.7 Firewall and iptables Requirements

Kubernetes uses `iptables` to handle many networking and port forwarding rules. Therefore, you must ensure that you do not have any rules set that may interfere with the functioning of Kubernetes. The `kubeadm-ha-setup` tool requires an iptables rule to accept forwarding traffic. If this rule is not set, the tool exits and notifies you that you may need to add this `iptables` rule. A standard Docker installation may create a firewall rule that prevents forwarding, therefore you may need to run:

```bash
# iptables -P FORWARD ACCEPT
```

The `kubeadm-ha-setup` tool checks `iptables` rules and, where there is a match, instructions are provided on how to modify your `iptables` configuration to meet any requirements. See Section 4.1, “Kubernetes and `iptables` Rules” for more information.

If you have a requirement to run a firewall directly on the systems where Kubernetes is deployed, you must ensure that all ports required by Kubernetes are available. For instance, the TCP port 6443 must be accessible on the master node to allow other nodes to access the API Server. All nodes must be able to accept connections from the master node on the TCP ports 10250-10252 and 10255, and traffic should be allowed on the UDP port 8472. All nodes must be able to receive traffic from all other nodes on every port on the network fabric that is used for the Kubernetes pods. The firewall must support masquerading.

Oracle Linux 7 installs and enables `firewalld`, by default. If you are running `firewalld`, the `kubeadm-ha-setup` tool notifies you of any rules that you may need to add. In summary, run the following commands on all nodes:
### Network Requirements

```bash
# firewall-cmd --add-masquerade --permanent
# firewall-cmd --add-port=2379-2380/tcp --permanent
# firewall-cmd --add-port=10250/tcp --permanent
# firewall-cmd --add-port=10251/tcp --permanent
# firewall-cmd --add-port=10252/tcp --permanent
# firewall-cmd --add-port=10255/tcp --permanent
# firewall-cmd --add-port=8472/udp --permanent
```

Additionally, run the following command on each node in the master cluster to enable API access:

```bash
# firewall-cmd --add-port=6443/tcp --permanent
```

The `--permanent` option ensures these firewall rules persistent across reboots. Remember to restart the firewall for these rules to take effect:

```bash
# systemctl restart firewalld
```

#### 3.2.8 Network Requirements

The `kubeadm-ha-setup` tool requires that it is able to access the Oracle Container Registry and possibly other internet resources to be able to pull any container images that you required. Therefore, unless you intend to set up a local mirror for all of your container image requirements, the systems where you intend to install Kubernetes must either have direct internet access, or must be configured to use a proxy. See Section 4.2, “Using Kubernetes With a Proxy Server” for more information.

The `kubeadm-ha-setup` tool checks whether the `br_netfilter` module is loaded and exits if it is not available. This module is required to enable transparent masquerading and to facilitate Virtual Extensible LAN (VxLAN) traffic for communication between Kubernetes pods across the cluster. If you need to check whether it is loaded, run:

```bash
# lsmod|grep br_netfilter
```

Kernel modules are usually loaded as they are needed, and it is unlikely that you would need to load this module manually. However, if necessary, you can load the module manually by running:

```bash
# modprobe br_netfilter
# echo "br_netfilter" > /etc/modules-load.d/br_netfilter.conf
```

Kubernetes requires that packets traversing a network bridge are processed by `iptables` for filtering and for port forwarding. To achieve this, tunable parameters in the kernel bridge module are automatically set when the `kubeadm` package is installed and a `sysctl` file is created at `/etc/sysctl.d/k8s.conf` that contains the following lines:

```ini
net.bridge.bridge-nf-call-ip6tables = 1
net.bridge.bridge-nf-call-iptables = 1
```

If you modify this file, or create anything similar yourself, you must run the following command to load the bridge tunable parameters:

```bash
# /sbin/sysctl -p /etc/sysctl.d/k8s.conf
```

The `kubeadm-ha-setup` tool configures a flannel network as the network fabric that is used for communications between Kubernetes pods. This overlay network uses VxLANs to facilitate network connectivity: [https://github.com/coreos/flannel](https://github.com/coreos/flannel)

By default, the `kubeadm-ha-setup` tool creates a network in the `10.244.0.0/16` range to host this network. The `kubeadm-ha-setup` tool provides an option to set the network range to an alternate range, if required, during installation. Systems in the Kubernetes deployment must not have any network devices configured for this reserved IP range.
3.2.9 SELinux Requirements

The `kubeadm-ha-setup` tool checks whether SELinux is set to enforcing mode. If enforcing mode is enabled, the tool exits with an error requesting that you set SELinux to permissive mode. Setting SELinux to permissive mode allows containers to access the host file system, which is required by pod networks. This is a requirement until SELinux support is improved in the `kubelet` tool for Kubernetes.

To disable SELinux temporarily, do the following:

```
# /usr/sbin/setenforce 0
```

To disable SELinux enforcing mode for subsequent reboots so that Kubernetes continues to run correctly, modify `/etc/selinux/config` and set the `SELinux` variable:

```
SELINUX=Permissive
```

3.2.10 Requirements to Use Oracle Container Services for use with Kubernetes on Oracle Cloud Infrastructure

Oracle Container Services for use with Kubernetes is engineered to work on Oracle Cloud Infrastructure. All of the instructions provided in this document can be used to install and configure Kubernetes across a group of compute instances. If you require additional information on configuration steps and usage of Oracle Cloud Infrastructure, please see:

[https://docs.us-phoenix-1.oraclecloud.com/Content/home.htm](https://docs.us-phoenix-1.oraclecloud.com/Content/home.htm)

The most important requirement for Oracle Container Services for use with Kubernetes on Oracle Cloud Infrastructure is that your Virtual Cloud Network (VCN) allows the compute nodes used in your Kubernetes deployment to communicate on the required ports. By default, compute nodes are unable to access each other across the Virtual Cloud Network until you have configured the Security List with the appropriate ingress rules.

Ingress rules should match the rules required in any firewall configuration, as described in Section 3.2.7, “Firewall and iptables Requirements”. Typically this involves adding the following ingress rules to the default security list for your VCN:

1. **Allow 2379-2380/TCP.**
   - **STATELESS**: Unchecked
   - **SOURCE CIDR**: `10.0.0.0/16`
   - **IP PROTOCOL**: TCP
   - **SOURCE PORT RANGE**: All
   - **DESTINATION PORT RANGE**: 2379-2380

2. **Allow 6443/TCP.**
   - **STATELESS**: Unchecked
   - **SOURCE CIDR**: `10.0.0.0/16`
   - **IP PROTOCOL**: TCP
   - **SOURCE PORT RANGE**: All
• **DESTINATION PORT RANGE**: 6443

3. **Allow 10250-10252/TCP**.
   - **STATELESS**: Unchecked
   - **SOURCE CIDR**: 10.0.0.0/16
   - **IP PROTOCOL**: TCP
   - **SOURCE PORT RANGE**: All
   - **DESTINATION PORT RANGE**: 10250-10252

4. **Allow 10255/TCP**.
   - **STATELESS**: Unchecked
   - **SOURCE CIDR**: 10.0.0.0/16
   - **IP PROTOCOL**: TCP
   - **SOURCE PORT RANGE**: All
   - **DESTINATION PORT RANGE**: 10255

5. **Allow 8472/UDP**.
   - **STATELESS**: Unchecked
   - **SOURCE CIDR**: 10.0.0.0/16
   - **IP PROTOCOL**: UDP
   - **SOURCE PORT RANGE**: All
   - **DESTINATION PORT RANGE**: 8472

Substitute **10.0.0.0/16** with the range used for the subnet that you created within the VCN for the compute nodes that will participate in the Kubernetes cluster. You may wish to limit this to the specific IP address range used specifically by the cluster components, or you may set this wider depending on your own security requirements.

**Important**

The ingress rules described here are the core rules that you need to set up to allow the cluster to function. For each service that you define or that you intend to use, you may need to define additional rules in the security list.

When creating compute instances to host Oracle Container Services for use with Kubernetes, all shape types are supported. The environment requires that for high availability clusters you use an Oracle Linux 7 Update 5 image or later with the Unbreakable Enterprise Kernel Release 5 (UEK R5).

If you intend to configure load balancers for your master cluster, while using Oracle Cloud Infrastructure, as described in **Configure Load Balancing**, see:

https://docs.cloud.oracle.com/iaas/Content/Balance/Concepts/balanceoverview.htm
3.3 Setting Up the Master Cluster

Before you begin, ensure you have satisfied the requirements in Section 3.2.5, “Oracle Container Registry Requirements”. Then on all the hosts that you are configuring as master nodes, install the `kubeadm` and `kubeadm-ha-setup` packages and their dependencies:

```bash
# yum install kubeadm kubelet kubectl kubeadm-ha-setup
```

Define the nodes in your high availability master cluster before proceeding further. To generate a template configuration file, copy the one provided on any node in the master cluster at `/usr/local/share/kubeadm/kubeadm-ha/ha.yaml`:

```bash
# cp /usr/local/share/kubeadm/kubeadm-ha/ha.yaml ~/ha.yaml
```

The first step is to specify the server IP addresses for each node used in the master cluster. There must be three nodes defined in this cluster, and they must each have unique hostnames:

```yaml
clusters:
  - name: master
    vip: 192.0.2.13
    nodes:
      - 192.0.2.10
      - 192.0.2.11
      - 192.0.2.12
```

Your cluster's `vip` address is the IP address of the server running the `keepalive` service for your cluster. This service is included by default with Oracle Linux 7, and you can find out more information about this service at https://docs.oracle.com/cd/E52668_01/E54669/html/ol7-about-keepalive.html.

All master nodes in your cluster must have shell access with password-less key-based authentication for the other master nodes whenever you use `kubeadm-ha-setup`. You can configure SSH keys for this, by following the instructions in the Oracle Linux 7 Administrator’s Guide at: https://docs.oracle.com/cd/E52668_01/E54669/html/ol7-s11-openssh.html.

You must define the SSH private key in the `private_key` variable, and the remote user in the `user` variable:

```yaml
private_key: /root/.ssh/id_rsa
user: root
```

You can optionally define a `pod_cidr` for your pod network. This is set by default to a reserved local IP range:

```yaml
pod_cidr: 10.244.0.0/16
```

Set the `image` variable to point at the Oracle Container Registry or an Oracle Container Registry mirror so that you are able to fetch the container images for the current release. See Section 3.2.5.1, “Using an Oracle Container Registry Mirror” for more information on using a mirror:

```yaml
image: container-registry.oracle.com/kubernetes
k8sversion: v1.12.5
```

Run `kubeadm-ha-setup up` instead of `kubeadm-setup.sh up` on just one Kubernetes node in the master cluster to apply these settings and automatically provision the other master nodes.

As `root`, run `kubeadm-ha-setup up` to add the host as a master node:

```bash
# kubeadm-ha-setup up ~/ha.yaml
Cleaning up ...
Reading configuration file /usr/local/share/kubeadm/kubeadm-ha/ha.yaml ...
CreateSSH /root/.ssh/id_rsa root
```
Checking 192.0.2.10
status 0

Checking 192.0.2.11
status 0

Checking 192.0.2.12
status 0

Configuring keepalived for HA ...
success

Setting up first master ... (maximum wait time 185 seconds)

[init] using Kubernetes version: v1.12.5
[preflight] running pre-flight checks
[preflight/images] Pulling images required for setting up a Kubernetes cluster
[preflight/images] This might take a minute or two, depending on the speed of your internet connection
[preflight/images] You can also perform this action beforehand using 'kubeadm config images pull'
[kubelet] Writing kubelet environment file with flags to file "/var/lib/kubelet/kubeadm-flags.env"
[kubelet] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[preflight] Activating the kubelet service
[certificates] Generated front-proxy-ca certificate and key.
[certificates] Generated etcd/ca certificate and key.
[certificates] Generated etcd/server certificate and key.
[certificates] Generated etcd/peer certificate and key.
[certificates] etcd/server serving cert is signed for DNS names 
[kubernetes.default.svc kubernetes.default.svc.cluster.local] and 
IPs [192.0.2.10 192.0.2.11 192.0.2.12 192.0.2.10]
[certificates] Generated etcd/peer serving cert is signed for DNS names 
[kubernetes.default.svc kubernetes.default.svc.cluster.local] and 
IPs [192.0.2.10 192.0.2.11 192.0.2.12 192.0.2.10]
[certificates] Generated etcd/healthcheck-client certificate and key.
[certificates] Generated ca certificate and key.
[certificates] Generated apiserver certificate and key.
[certificates] Generated apiserver-client certificate and key.
[certificates] valid certificates and keys now exist in "/etc/kubernetes/pki"
[certificates] Generated sa key and public key.
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/admin.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/kubelet.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/controller-manager.conf"
[kubeconfig] Wrote KubeConfig file to disk: "/etc/kubernetes/scheduler.conf"
[controlplane] wrote Static Pod manifest for component kube-apiserver to "/etc/kubernetes/manifests/kube-apiserver.yaml"
[controlplane] wrote Static Pod manifest for component kube-controller-manager to "/etc/kubernetes/manifests/kube-controller-manager.yaml"
[etcd] Wrote Static Pod manifest for a local etcd instance to "/etc/kubernetes/manifests/etcd.yaml"
[init] waiting for the kubelet to boot up the control plane as Static Pods from directory "/etc/kubernetes/manifests"
[apiclient] All control plane components are healthy after 27.004111 seconds
[uploadconfig] storing the configuration used in
ConfigMap "kubeadm-config" in the "kube-system" Namespace
[kubelet] Creating a ConfigMap "kubeadm-config-1.12" in namespace kube-system with the configuration for the kubelets in the cluster
[markmaster] Marking the node master1.example.com as master
[markmaster] Marking the node master1.example.com as master by adding the label "node-role.kubernetes.io/master="
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to the Node API object "master1.example.com" as an annotation
[bootstraptoken] using token: ixxbh9.zrtxo7jwo1u2zssp
[bootstraptoken] configured RBAC rules to allow Node Bootstrap tokens
Configure Load Balancing

To support a load balancer as part of your high availability master cluster configuration, set its IP address as the `loadbalancer` value in your `~/ha.yaml` file:

```
loadbalancer: 192.0.2.15
```

The `loadbalancer` value will be applied as part of the setup process with the following command:

```
# kubeadm-ha-setup up ~/ha.yaml --lb
```

This configuration step is optional, but if it is included ensure port 6443 is open for all of your master nodes. See Section 3.2.7, “Firewall and iptables Requirements”.

Preparing to Use Kubernetes as a Regular User

To use the Kubernetes cluster as a regular user, perform the following steps on each of the nodes in the master cluster:
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
```

You cannot use the `kubectl` command if the path to this file is not set for this environment variable. Remember to export the `KUBECONFIG` variable for each subsequent login so that the `kubectl` and `kubeadm` commands use the correct `admin.conf` file, otherwise you might find that these commands do not behave as expected after a reboot or a new login. For instance, append the export line to your `.bashrc`:

```
$ 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 Docker containers running as a Kubernetes pod. These can be viewed by running the following command on the master node:

```
$ kubectl get pods -n kube-system
```

### 3.4 Setting Up a Worker Node

Repeat these steps on each host that you want to add to the cluster as a worker node.

Install the `kubeadm` package and its dependencies:

```
# yum install kubeadm kubelet kubectl kubeadm-ha-setup
```

As `root`, run the `kubeadm-ha-setup join` command to add the host as a worker node:
# Upgrading

Trying to pull image kube-proxy v1.12.5 from container-registry.oracle.com/kubernetes

Cleaning up...

This node has joined the cluster:
* Certificate signing request was sent to apiserver and a response was received.
* The Kubelet was informed of the new secure connection details.

Run 'kubectl get nodes' on the master to see this node join the cluster.

To verify that the worker has been successfully added to the high availability cluster, run `kubectl get nodes` on any node in the master cluster:

```
$ kubectl get nodes
NAME                  STATUS   ROLES    AGE     VERSION
master1.example.com   Ready    master   10m     v1.12.5+2.1.1.el7
master2.example.com   Ready    master   9m56s   v1.12.5+2.1.1.el7
master3.example.com   Ready    master   9m16s   v1.12.5+2.1.1.el7
worker1.example.com   Ready    <none>   2m26s   v1.12.5+2.1.1.el7
```

## 3.5 Upgrading

Oracle Container Services for use with Kubernetes 1.1.12 is the first release to support high availability clusters.

### Important

Oracle does not support upgrading existing single master node clusters built with the `kubeadm-setup.sh` script to high availability clusters. You must build and manage high availability clusters using the `kubeadm-ha-setup` utility.

Similarly, upgrading master nodes in a high availability cluster with the `kubeadm-setup.sh` script is not supported. All maintenance and management operations within high availability clusters must be performed with the `kubeadm-ha-setup` utility.

### 3.5.1 Updating the High Availability cluster

### Important

The `kubeadm-ha-setup update` command is only supported for errata release updates on existing High Availability clusters.
A `kubeadm-ha-setup upgrade` command for larger upgrades will be provided in a future release. Major release upgrades are not supported at this time.

Errata Release Update Steps

1. Create a backup for your High Availability cluster before proceeding by following the instructions in Section 4.3, “Cluster Backup and Restore”.

2. On each master node in the cluster, update the `kubeadm-ha-setup` package:

   ```
   # yum update kubeadm-ha-setup
   ```

3. On the master node from which you intend to run the cluster update from, update the required prerequisite packages:

   ```
   # yum update kubeadm kubelet kubectl
   ```

4. If you are using the Oracle Container Registry to obtain images, log in.

   Follow the instructions in Section 3.2.5, “Oracle Container Registry Requirements”. Note that if images are updated on the Oracle Container Registry, you may be required to accept the Oracle Standard Terms and Restrictions again before you are able to perform the update. If you are using one of the Oracle Container Registry mirrors, see Section 3.2.5.1, “Using an Oracle Container Registry Mirror” for more information. If you have configured a local registry, you may need to set the `KUBE_REPO_PREFIX` environment variable to point to the appropriate registry. You may also need to update your local registry with the most current images for the version that you are upgrading to. See Section 3.2.5.2, “Setting Up an Optional Local Registry” for more information.

5. Verify that the currently reported node versions match those of the previous package:

   ```
   # kubectl get nodes
   NAME                  STATUS   ROLES    AGE     VERSION
   master1.example.com   Ready    master   4m8s    v1.12.5+2.1.1.el7
   master2.example.com   Ready    master   2m25s   v1.12.5+2.1.1.el7
   master3.example.com   Ready    master   2m12s   v1.12.5+2.1.1.el7
   worker1.example.com   Ready    <none>   25s     v1.12.5+2.1.1.el7
   ```

6. Start the scripted update process by using the `kubeadm-ha-setup` tool:

   ```
   # kubeadm-ha-setup update
   [WARNING] This action will update this cluster to the latest version(1.12.7).
   [WARNING] You must take a backup before updating the cluster, as the update may fail.
   [PROMPT] Do you want to continue updating your cluster? Please type Yes/y to confirm or No/n to abort (Case insensitive):
   Y
   Kubernetes Cluster Version: v1.12.5
   Kubeadm version:1.12.7-1.1.2, Kubelet version 1.12.5-2.1.1
   Kubeadm version: 1.12.5-2.1.1 Kubelet version: 1.12.7-1.1.2
   Reading configuration file /usr/local/share/kubeadm/run/kubeadm/ha.yaml ...
   Checking repo access
   [preflight] Running pre-flight checks.
   [upgrade] Making sure the cluster is healthy:
   [upgrade/config] Making sure the configuration is correct:
   [upgrade/config] Reading configuration from the cluster...
   [upgrade/config] FYI: You can look at this config file with 'kubectl -n kube-system get cm kubeadm-config -oyaml'
   [upgrade/apply] Respecting the --cri-socket flag that is set with higher priority than the config file.
   [upgrade/version] You have chosen to change the cluster version to "v1.12.7"
   [upgrade/versions] Cluster version: v1.12.5+2.1.1.el7
   [upgrade/versions] Kubeadm version: v1.12.7+1.1.2.el7
   [upgrade/prepull] Will prepull images for components
Updating the High Availability cluster

[upgrade/prepull] Prepulling image for component etcd.
[upgrade/prepull] Prepulling image for component kube-apiserver.
[upgrade/prepull] Prepulling image for component kube-scheduler.
[apiclient] Found 0 Pods for label selector k8s-app=upgrade-prepull-etcd
[apiclient] Found 1 Pods for label selector k8s-app=upgrade-prepull-kube-controller-manager
[apiclient] Found 0 Pods for label selector k8s-app=upgrade-prepull-kube-scheduler
[apiclient] Found 3 Pods for label selector k8s-app=upgrade-prepull-kube-apiserver
[apiclient] Found 3 Pods for label selector k8s-app=upgrade-prepull-kube-controller-manager
[apiclient] Found 3 Pods for label selector k8s-app=upgrade-prepull-kube-scheduler
[upgrade/prepull] Prepulled image for component kube-apiserver.
[upgrade/prepull] Prepulled image for component kube-scheduler.
[upgrade/prepull] Successfully prepulled the images for all the control plane components
[upgrade/apply] Upgrading your Static Pod-hosted control plane to version "v1.12.7"...

Static pod: kube-apiserver-master1.example.com hash: f9004e982edd18c6303596943ce5493
Static pod: kube-controller-manager-master1.example.com hash: 959010be574fca0a237ca3f029f03ea2
Static pod: kube-scheduler-master1.example.com hash: 22961405d099b7721c7598daa73d6

[upgrade/staticpods] Writing new Static Pod manifests to
"/etc/kubernetes/tmp/kubeadm-upgraded-manifests867609756"
[controlplane] wrote Static Pod manifest for component kube-apiserver to
"/etc/kubernetes/tmp/kubeadm-upgraded-manifests867609756/kube-apiserver.yaml"
[controlplane] wrote Static Pod manifest for component kube-controller-manager to
"/etc/kubernetes/tmp/kubeadm-upgraded-manifests867609756/kube-controller-manager.yaml"
[controlplane] wrote Static Pod manifest for component kube-scheduler to
"/etc/kubernetes/tmp/kubeadm-upgraded-manifests867609756/kube-scheduler.yaml"

[upgrade/staticpods] Moved new manifest to
"/etc/kubernetes/manifests/kube-apiserver.yaml" and backed up old manifest to
"/etc/kubernetes/tmp/kubeadm-backup-manifests-2019-04-08-14-28-11/kube-apiserver.yaml"

[upgrade/staticpods] Waiting for the kubelet to restart the component
[upgrade/staticpods] This might take a minute or longer depending on
the component/version gap (timeout 5m0s
Static pod: kube-apiserver-master1.example.com hash: f9004e982edd18c6303596943ce5493
Static pod: kube-apiserver-master1.example.com hash: f9004e982edd18c6303596943ce5493
Static pod: kube-apiserver-master1.example.com hash: f9004e982edd18c6303596943ce5493
Static pod: kube-apiserver-master1.example.com hash: f9004e982edd18c6303596943ce5493
[apiclient] Found 3 Pods for label selector component=kube-apiserver
[upgrade/staticpods] Component "kube-apiserver" upgraded successfully!
[upgrade/staticpods] Moved new manifest to
"/etc/kubernetes/manifests/kube-controller-manager.yaml" and backed up old manifest to
"/etc/kubernetes/tmp/kubeadm-backup-manifests-2019-04-08-14-28-11/kube-controller-manager.yaml"

[upgrade/staticpods] Waiting for the kubelet to restart the component
[upgrade/staticpods] This might take a minute or longer depending on
the component/version gap (timeout 5m0s
Static pod: kube-apiserver-master1.example.com hash: a692b972629a4c2a9e2cdd8301035
Static pod: kube-apiserver-master1.example.com hash: a692b972629a4c2a9e2cdd8301035
Static pod: kube-apiserver-master1.example.com hash: a692b972629a4c2a9e2cdd8301035
Static pod: kube-apiserver-master1.example.com hash: a692b972629a4c2a9e2cdd8301035
[apiclient] Found 3 Pods for label selector component=kube-controller-manager
[upgrade/staticpods] Component "kube-controller-manager" upgraded successfully!
[upgrade/staticpods] Moved new manifest to
"/etc/kubernetes/manifests/kube-scheduler.yaml" and backed up old manifest to
"/etc/kubernetes/tmp/kubeadm-backup-manifests-2019-04-08-14-28-11/kube-scheduler.yaml"

[upgrade/staticpods] Waiting for the kubelet to restart the component
[upgrade/staticpods] This might take a minute or longer depending on
the component/version gap (timeout 5m0s
Static pod: kube-scheduler-master1.example.com hash: 959010be574fca0a237ca3f029f03ea2
Static pod: kube-scheduler-master1.example.com hash: 959010be574fca0a237ca3f029f03ea2
Static pod: kube-scheduler-master1.example.com hash: 959010be574fca0a237ca3f029f03ea2
Static pod: kube-scheduler-master1.example.com hash: 959010be574fca0a237ca3f029f03ea2
[apiclient] Found 3 Pods for label selector component=kube-scheduler

[upgrade/staticpods] Component "kube-scheduler" upgraded successfully!
[uploadconfig] storing the configuration used in
ConfigMap "kubelet-config-1.12" in namespace kube-system
with the configuration for the kuberets in the cluster
[kubelet] Creating a ConfigMap "kubelet-config-1.12" in namespace kube-system
[kubelet] Downloading configuration for the kubelet from
the "kubelet-config-1.12" ConfigMap in the kube-system namespace
[kubelet] Writing kubelet configuration to file "/var/lib/kubelet/config.yaml"
[patchnode] Uploading the CRI Socket information "/var/run/dockershim.sock" to
the Node API object "master1.example.com" as an annotation
[bootstraptoken] configured RBAC rules to allow Node Bootstrap tokens to
post CSRs in order for nodes to get long term certificate credentials
[bootstraptoken] configured RBAC rules to allow the csrapprover controller
automatically approve CSRs from a Node Bootstrap Token
[bootstraptoken] configured RBAC rules to allow certificate rotation for
all node client certificates in the cluster
[addons] Applied essential addon: CoreDNS
[addons] Applied essential addon: kube-proxy
[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.12.7". Enjoy!

[upgrade/kubelet] Now that your control plane is upgraded, please proceed with
upgrading your kubelets if you haven't already done so.

Loaded plugins: langpacks, ulninfo
Resolving Dependencies
---> Running transaction check
---> Package kubelet.x86_64 0:1.12.5-2.1.1.el7 will be updated
---> Package kubelet.x86_64 0:1.12.7-1.1.2.el7 will be an update
---> Processing Dependency: conntrack for package: kubelet-1.12.7-1.1.2.el7.x86_64
---> Running transaction check
---> Package conntrack-tools.x86_64 0:1.4.4-4.el7 will be installed
---> Processing Dependency: libnetfilter_cttimeout.so.1(LIBNETFILTER_CTTIMEOUT_1.1)(64bit)
for package: conntrack-tools-1.4.4-4.el7.x86_64
---> Processing Dependency: libnetfilter_cttimeout.so.1(LIBNETFILTER_CTTIMEOUT_1.0)(64bit)
for package: conntrack-tools-1.4.4-4.el7.x86_64
---> Processing Dependency: libnetfilter_chelper.so.0(LIBNETFILTER_CTHELPER_1.0)(64bit)
for package: conntrack-tools-1.4.4-4.el7.x86_64
---> Processing Dependency: libnetfilter_cttimeout.so.1()(64bit)
for package: conntrack-tools-1.4.4-4.el7.x86_64
---> Processing Dependency: libnetfilter_chelper.so.0()()(64bit)
for package: conntrack-tools-1.4.4-4.el7.x86_64
---> Processing Dependency: libnetfilter_queue.so.0.1()(64bit)
for package: conntrack-tools-1.4.4-4.el7.x86_64
---> Running transaction check
---> Package libnetfilter_chelper.x86_64 0:1.0.0-9.el7 will be installed
---> Package libnetfilter_cttimeout.x86_64 0:1.0.0-6.el7 will be installed
---> Package libnetfilter_queue.x86_64 0:1.0.2-2.el7_2 will be installed
---> Finished Dependency Resolution

Dependencies Resolved

================================================================================
Package Arch        Version                Repository     Size
================================================================================
Updating:
kubelet  x86_64      1.12.7-1.1.2.el7     ol7_addons     19 M
Installing for dependencies:
conntrack-tools x86_64      1.4.4-4.el7     ol7_latest     186 k
libnetfilter_chelper x86_64      1.0.0-9.el7     ol7_latest     17 k
libnetfilter_cttimeout x86_64      1.0.0-6.el7     ol7_latest     17 k
libnetfilter_queue x86_64      1.0.2-2.el7_2     ol7_latest     22 k

Transaction Summary
================================================================================
Install     ( 4 Dependent packages)
Upgrade  1 Package
Updating the High Availability cluster

Total download size: 19 M
Downloading packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.

Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Installing: libnetfilter_cthelper-1.0.0-9.el7.x86_64 1/6
Installing: libnetfilter_cttimeout-1.0.0-6.el7.x86_64 2/6
Installing: libnetfilter_queue-1.0.2-2.el7_2.x86_64 3/6
Installing: conntrack-tools-1.4.4-4.el7.x86_64 4/6
Updating: kubelet-1.12.7-1.1.2.el7.x86_64 5/6
Cleanup: kubelet-1.12.5-2.1.1.el7.x86_64 6/6
Verifying: libnetfilter_queue-1.0.2-2.el7_2.x86_64 1/6
Verifying: libnetfilter_cttimeout-1.0.0-6.el7.x86_64 2/6
Verifying: kubelet-1.12.7-1.1.2.el7.x86_64 3/6
Verifying: libnetfilter_cthelper-1.0.0-9.el7.x86_64 4/6
Verifying: conntrack-tools-1.4.4-4.el7.x86_64 5/6
Verifying: kubelet-1.12.5-2.1.1.el7.x86_64 6/6

Dependency Installed:
conntrack-tools.x86_64 0:1.4.4-4.el7
libnetfilter_cthelper.x86_64 0:1.0.0-9.el7
libnetfilter_cttimeout.x86_64 0:1.0.0-6.el7
libnetfilter_queue.x86_64 0:1.0.2-2.el7_2

Updated:
kubelet.x86_64 0:1.12.7-1.1.2.el7

Complete!

Resolved Dependencies

Transaction Summary

Upgrade 1 Package

Total download size: 7.7 M
Downloading packages:
Delta RPMs disabled because /usr/bin/applydeltarpm not installed.

Running transaction check
Running transaction test
Transaction test succeeded
Running transaction
Updating: kubectl-1.12.7-1.1.2.el7.x86_64 1/2
Cleanup: kubectl-1.12.5-2.1.1.el7.x86_64 2/2
Verifying: kubectl-1.12.7-1.1.2.el7.x86_64 1/2
Verifying: kubectl-1.12.5-2.1.1.el7.x86_64 2/2

Updated:
kubectl.x86_64 0:1.12.7-1.1.2.el7

Complete!
Updating the High Availability cluster

Waiting for the cluster to become healthy
Updating remote master nodes
CreateSSH /root/.ssh/id_rsa root
Updating the master node: master2.example.com
Successfully updated the master node: master2.example.com
Updating the master node: master3.example.com
Successfully updated the master node: master3.example.com
The cluster has been updated successfully
Please update the worker nodes in your cluster and do the following:
1. On Master: kubectl drain worker1.example.com --ignore-daemonsets
2. On Worker1: yum install -y \\
kubeadm-1.12.7-1.1.2.el7 kubelet-1.12.7-1.1.2.el7 \\
kubectl-1.12.7-1.1.2.el7 kubeadm-ha-setup-0.0.2-1.0.21.el7
3. On Worker1: systemctl restart kubelet
4. On Master: kubectl uncordon worker1.example.com
5. Verify the update on master node: kubectl get nodes

Optionally, you can override the default container registry choice during the errata release update by specifying the --registry option:

```bash
# kubeadm-ha-setup update --registry container-registry-ash.oracle.com
```

7. Verify that your master nodes have been updated correctly before proceeding to update the worker nodes:

```bash
# kubectl get nodes
NAME                  STATUS   ROLES    AGE   VERSION
master1.example.com   Ready    master   17m   v1.12.7+1.1.2.el7
master2.example.com   Ready    master   15m   v1.12.7+1.1.2.el7
master3.example.com   Ready    master   15m   v1.12.7+1.1.2.el7
worker1.example.com   Ready    <none>   13m   v1.12.5+2.1.1.el7
```

8. Use the kubectl tool to drain each of your worker nodes from the cluster:

```bash
# kubectl drain worker1.example.com --ignore-daemonsets
node/worker1.example.com cordoned
```

Check that the worker nodes are unable to accept any further scheduling or new pods:

```bash
# kubectl get nodes
```

Note that a node that has been drained should have its status set to **SchedulingDisabled**.

9. On each of the worker nodes, upgrade the required packages to the latest versions and restart the kubelet service:

```bash
# yum update kubeadm kubelet kubectl kubeadm-ha-setup
# systemctl restart kubelet
```

10. Now that the upgrades are complete for each worker node, uncordon them using the kubectl tool from the master cluster:

```bash
# kubectl uncordon worker1.example.com
node/worker1.example.com uncordoned
```

Check that the worker nodes are now able to accept new schedules and pods:

```bash
# kubectl get nodes
NAME                  STATUS   ROLES    AGE   VERSION
master1.example.com   Ready    master   17m   v1.12.7+1.1.2.el7
master2.example.com   Ready    master   15m   v1.12.7+1.1.2.el7
master3.example.com   Ready    master   15m   v1.12.7+1.1.2.el7
worker1.example.com   Ready    <none>   13m   v1.12.7+1.1.2.el7
```
Recover from Errata Release Update Failures

If the update fails to complete successfully, you will need to do a full cluster restore from backup. Note that the cluster will not be responsive to new commands until the restore process is complete.

**Recovery Steps**

1. Check which of the required packages were updated on each node:

   ```
   # yum list installed kubeadm kubelet kubectl
   ```

2. Downgrade each of the individual packages that has already been updated to the previous errata version. For example, to downgrade the `kubeadm` package:

   ```
   # yum downgrade kubeadm
   ```

   **Note**
   Do not downgrade the `kubeadm-ha-setup` package on your master nodes, as the latest version is always designed to support errata release update recovery.

3. Follow the restore steps in Section 4.3, “Cluster Backup and Restore”, but add the `--force` flag to override any version checks:

   ```
   # kubeadm-ha-setup restore /backups/master-backup-v1.12.5-2-154442719.tar --force
   ```

4. When recovery is complete, you may re-attempt the High Availability cluster update.
Chapter 4 Kubernetes Administration and Configuration

Table of Contents

4.1 Kubernetes and iptables Rules ................................................................. 65
4.2 Using Kubernetes With a Proxy Server ..................................................... 65
4.3 Cluster Backup and Restore ....................................................................... 66
  4.3.1 Single Master Cluster ....................................................................... 66
  4.3.2 High Availability Cluster .................................................................. 69
4.4 Kubernetes Dashboard ............................................................................... 72
4.5 Removing Worker Nodes from the Cluster .................................................. 73
  4.5.1 Single Master Cluster ....................................................................... 73
  4.5.2 High Availability Cluster .................................................................. 74

This chapter describes how to configure and administer your Kubernetes deployment.

4.1 Kubernetes and iptables Rules

Kubernetes uses iptables to handle many networking and port forwarding rules. Be careful of using services that may create conflicting iptables rules. You can check the rules by running `iptables-save`, which dumps the rule set to STDOUT.

If you intend to expose application services externally, by either using the NodePort or LoadBalancing service types, traffic forwarding must be enabled in your iptables rule set. If you find that you are unable to access a service from outside of the network used by the pod where your application is running, check that your iptables rule set does not contain a rule similar to the following:

```plaintext
:FORWARD DROP [0:0]
```

If you have a rule to drop all forwarding traffic, you may need to run:

```plaintext
# iptables -P FORWARD ACCEPT
```

If you are running iptables as a service instead of firewalld, you can save current iptables configuration so that it is persistent across reboots. To do this, run:

```plaintext
# iptables-save > /etc/sysconfig/iptables
```

Note that you must have the iptables-services package installed for this to work. Oracle recommends using the default firewalld service as this provides a more consistent experience and allows you to make changes to the firewall configuration without flushing existing rules and reloading the firewall.

Nodes running applications that need to communicate directly between pods and that are IP aware, may require additional custom iptables configuration to bypass the default firewalld masquerading rules. For example, setting these two iptables rules on the nodes running a server application on IP address 192.0.2.15 and a client application on IP address 192.0.2.16 enables direct communication between them:

```plaintext
# iptables -t nat -I POST_public_allow -s 192.0.2.15/32 -d 192.0.2.16/32 -j RETURN
# iptables -t nat -I POST_public_allow -s 192.0.2.16/32 -d 192.0.2.15/32 -j RETURN
```

4.2 Using Kubernetes With a Proxy Server

In environments where a proxy server is configured to access the internet services, such as the Docker Hub or the Oracle Container Registry, you may need to perform several configuration steps to get Kubernetes to install and to run correctly.
1. Ensure that the Docker engine startup configuration on each node in the cluster is configured to use the proxy server. For instance, create a systemd service drop-in file at `/etc/systemd/system/docker.service.d/http-proxy.conf` with the following contents:

```
[Service]
Environment="HTTP_PROXY=http://proxy.example.com:80/"
Environment="HTTPS_PROXY=https://proxy.example.com:443/"
```

Replace `http://proxy.example.com:80/` with the URL for your HTTP proxy service. If you have an HTTPS proxy and you have specified this as well, replace `https://proxy.example.com:443/` with the URL and port for this service. If you have made a change to your Docker systemd service configuration, run the following commands:

```
# systemctl daemon-reload; systemctl restart docker
```

2. You may need to set the `http_proxy` or `https_proxy` environment variables to be able to run other commands on any of the nodes in your cluster. For example:

```
# export http_proxy="http://proxy.example.com:80/"
# export https_proxy="https://proxy.example.com:443/"
```

3. Disable the proxy configuration for the local host and any node IPs in the cluster:

```
# export no_proxy="127.0.0.1, 192.0.2.10, 192.0.2.11, 192.0.2.12"
```

These steps should be sufficient to enable the deployment to function normally. Use of a transparent proxy that does not require configuration on the host and which ignores internal network requests, can reduce the complexity of the configuration and may help to avoid unexpected behavior.

### 4.3 Cluster Backup and Restore

#### 4.3.1 Single Master Cluster

The `kubeadm-setup.sh` script enables cluster backup and restore functionality so that you can easily protect your Kubernetes deployment from a failure of the master node in the cluster. Cluster status and configuration data is stored in the Cluster State Store, also referred to as `etcd`.

For the backup and restore processes to work properly, there are some basic requirements:

- The hostname and IP address of the master node being restored, must match the hostname and IP address of the master node that was backed up. The usual use case for restore is after system failure, so the restore process expects a matching system for the master node with a fresh installation of the Docker engine and the Kubernetes packages.

- The master node must be tainted so that is unable to run any workloads or containers other than those that the master node requires. This is the default configuration if you used the `kubeadm-setup.sh` script to setup your environment. The backup process does not back up any containers running on the master node other than the containers specific to managing the Kubernetes cluster.

- The `backup` command must be run on the master node.

- Any Docker engine configuration applied to the master node prior to the backup process must be manually replicated on the node on which you intend to run the restore operation. You may need to manually configure your Docker storage driver and proxy settings before running a restore operation.

- The `backup` command checks for minimum disk space of 100 MB at the specified backup location. If the space is not available, the `backup` command exits with an error.
• A restore can only function correctly using the backup file for a Kubernetes cluster running the same version of Kubernetes. You cannot restore a backup file for a Kubernetes 1.7.4 cluster, using the Kubernetes 1.8.4 tools.

The **backup** command requires that you stop the cluster during the backup process. Running container configurations on the worker nodes are unaffected during the backup process. The following steps describe how to create a backup file for the master node.

**Back up the cluster configuration and state**

1. **Stop the cluster.**

To back up the cluster configuration and state, the cluster must be stopped so that no changes can occur in state or configuration during the backup process. While the cluster is stopped, the worker nodes continue to run independently of the cluster, allowing the containers hosted on each of these nodes to continue to function. To stop the cluster, on the master node, run:

```
# kubeadm-setup.sh stop
Stopping kubelet now ...
Stopping containers now ...
```

2. **Run** `kubeadm-setup.sh backup` **and specify the directory where the backup file should be stored.**

```
# kubeadm-setup.sh backup /backups
Using container-registry.oracle.com/etcd:3.2.24
Checking if container-registry.oracle.com/etcd:3.2.24 is available
376e8be3701cc4e37f4e3e3f04430105696de18f3eb6f6af74c354ca61cd0e02a0
/var/run/kubeadm/backup/etcd-backup-1544442719.tar
Using container-registry.oracle.com/pause:3.1
/var/run/kubeadm/backup/pause-backup-1544442719.tar
Backup is successfully stored at /backups/master-backup-v1.12.5-2-1544442719.tar ...
You can restart your cluster now by doing:
# kubeadm-setup.sh restart
```

Substitute `/backups` with the path to a directory where you wish to store the backed up data for your cluster.

Each run of the `backup` command creates as a tar file that is timestamped so that you can easily restore the most recent backup file. The backup file also contains a sha256 checksum that is used to verify the validity of the backup file during restore. The `backup` command instructs you to restart the cluster when you have finished backing up.

3. **Restart the cluster.**

```
# kubeadm-setup.sh restart
Restarting containers now ...
Detected node is master ...
Checking if env is ready ...
Checking whether docker can pull busybox image ...
Checking access to container-registry.oracle.com ...
Trying to pull repository container-registry.oracle.com/pause ...
3.1: Pulling from container-registry.oracle.com/pause
Digest: sha256:802ef89b9eb7e874a76e1cfd79ed990b63b0b84a05cfa09f0293379ac0261b49
Status: Image is up to date for container-registry.oracle.com/pause:3.1
Checking firewalld settings ...
Checking iptables default rule ...
Checking br_netfilter module ...
Checking systemd variables ...
Restarting kubelet ...
Waiting for node to restart ...
....
Master node restarted. Complete synchronization between nodes may take a few minutes.
```
Checks, similar to those performed during cluster setup, are performed when the cluster is restarted, to ensure that no environment changes may have occurred that could prevent the cluster from functioning correctly. Once the cluster has started, it can take a few minutes for the nodes within the cluster to report status and for the cluster to settle back to normal operation.

A restore operation is typically performed on a freshly installed host, but can be run on an existing setup, as long as any pre-existing setup configuration is removed. The restore process assumes that the Docker engine is configured in the same way as the original master node. The Docker engine must be configured to use the same storage driver and if proxy configuration is required, you must set this up manually before restoring, as described in the following steps.

**Restore the cluster configuration and state**

1. On the master host, ensure that the latest Docker and Kubernetes versions are installed and that the master node IP address and hostname match the IP address and hostname used before failure. The `kubeadm` package pulls in all of the required dependencies, including the correct version of the Docker engine.

```bash
# yum install kubeadm kubectl kubelet
```

2. Run the `kubeadm-setup.sh restore` command.

```bash
# kubeadm-setup.sh restore /backups/master-backup-v1.12.5-2-1544442719.tar
Checking sha256sum of the backup files ... /var/run/kubeadm/backup/etcd-backup-1544442719.tar: OK
/var/run/kubeadm/backup/k8s-master-0-1544442719.tar: OK
Restoring backup from /backups/master-backup-v1.12.5-2-1544442719.tar ...
Using 3.2.24
etcd cluster is healthy ...
Cleaning up etcd container ...
27148ae6765a546bf45d527d627e5344130fb453c4a532aa2f47c54946f2e665
27148ae6765a546bf45d527d627e5344130fb453c4a532aa2f47c54946f2e665
Restore successful ...
You can restart your cluster now by doing:
# kubeadm-setup.sh restart
```

Substitute `/backups/master-backup-v1.12.5-2-1544442719.tar` with the full path to the backup file that you wish to restore.

3. Restart the cluster.

```bash
# kubeadm-setup.sh restart
Restarting containers now ...
Detected node is master ...
Checking if env is ready ...
Checking whether docker can pull busybox image ...
Checking access to container-registry.oracle.com ...
Trying to pull repository container-registry.oracle.com/pause ...
3.1: Pulling from container-registry.oracle.com/pause
Digest: sha256:802ef89b9eb7e874a76e1cfd79ed990b63b0b84a05ca09f0293379ac0261b49
Status: Image is up to date for container-registry.oracle.com/pause:3.1
Checking firewall settings ...
Checking iptables default rule ...
Checking br_netfilter module ...
Checking sysctl variables ...
Enabling kubelet ...
Created symlink from /etc/systemd/system/multi-user.target.wants/kubelet.service to /etc/systemd/system/kubelet.service.
Restarting kubelet ...
Waiting for node to restart ...
........+++ Restarting pod kube-flannel-ds-glwgx
```
4. Copy the Kubernetes `admin.conf` file to your home directory:

```bash
$ sudo cp /etc/kubernetes/admin.conf $HOME/
```

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

```bash
$ sudo chown $(id -u):$(id -g) $HOME/admin.conf
```

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

```bash
$ export KUBECONFIG=$HOME/admin.conf
```

You cannot use the `kubectl` command if the path to this file is not set for this environment variable. Remember to export the `KUBECONFIG` variable for each subsequent login so that the `kubectl` and `kubeadm` commands use the correct `admin.conf` file, otherwise you might find that these commands do not behave as expected after a reboot or a new login. For instance, append the export line to your `.bashrc`:

```bash
$ echo 'export KUBECONFIG=$HOME/admin.conf' >> $HOME/.bashrc
```

5. Check that your cluster has been properly restored. Use `kubectl` to check on the status of the nodes within the cluster and to check any existing configuration. For example:

```bash
$ kubectl get nodes
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
<th>ROLES</th>
<th>AGE</th>
<th>VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>master.example.com</td>
<td>Ready</td>
<td>master</td>
<td>1h</td>
<td>v1.12.5+2.1.1.el7</td>
</tr>
<tr>
<td>worker1.example.com</td>
<td>Ready</td>
<td>&lt;none&gt;</td>
<td>1h</td>
<td>v1.12.5+2.1.1.el7</td>
</tr>
<tr>
<td>worker2.example.com</td>
<td>Ready</td>
<td>&lt;none&gt;</td>
<td>1h</td>
<td>v1.12.5+2.1.1.el7</td>
</tr>
</tbody>
</table>

```bash
$ kubectl get pods
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>nginx-deployment-4234284026-g8g95</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>10m</td>
</tr>
<tr>
<td>nginx-deployment-4234284026-k1h8w</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>10m</td>
</tr>
<tr>
<td>nginx-deployment-4234284026-sbkqr</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>10m</td>
</tr>
</tbody>
</table>

### 4.3.2 High Availability Cluster

The `kubeadm-ha-setup` tool enables cluster backup and restore functionality so that you can easily protect your Kubernetes deployment from a failure of the master node in the cluster. Cluster status, configuration data and snapshots are stored in the Cluster State Store, also referred to as `etcd`.

For the backup and restore processes to work properly, there are some basic requirements:

- The hostname and IP address of the master node being restored, must match the hostname and IP address of the master node that was backed up. The usual use case for restore is after system failure, so the restore process expects a matching system for each master node with a fresh installation of the Docker engine and the Kubernetes packages.

- A restore can only function correctly using a backup of a Kubernetes high availability cluster running the same version of Kubernetes. The Docker engine versions must also match.

- There must be a dedicated share storage directory that is accessible to all nodes in the master cluster during the backup and restore phases.

- All nodes in the master cluster must have root access using password-less key-based authentication for all other nodes in the master cluster whenever `kubeadm-ha-setup` is used.
A full restore is only required if a period of downtime included more than one node in the master cluster. Note that a full restore disrupts master node availability throughout the duration of the restore process.

**Back up the cluster configuration and state**

1. Run `kubeadm-ha-setup backup` and specify the directory where the backup file should be stored.

```
# kubeadm-ha-setup backup /backups
```

Substitute `/backups` with the path to the network share directory where you wish to store the backup data for your master cluster.

Each run of the `backup` command creates as a tar file that is timestamped so that you can easily restore the most recent backup file. The backup file also contains a sha256 checksum that is used to verify the validity of the backup file during restore. The `backup` command instructs you to restart the cluster when you have finished backing up.

A restore operation is typically performed on a freshly installed host, but can be run on an existing setup, as long as any pre-existing setup configuration is removed.

The restore process assumes that the IP address configuration for each node in the master cluster matches the configuration in the backed up data. If you are restoring on one or more freshly installed hosts, make sure that the IP addressing matches the address assigned to the host or hosts that you are replacing.

The restore process assumes that the Docker engine is configured in the same way as the original master node. The Docker engine must be configured to use the same storage driver and if proxy configuration is required, you must set this up manually before restoring, as described in the following steps.

```
# yum install kubeadm kubectl kubelet kubeadm-ha-setup
```

2. Run the `kubeadm-ha-setup restore` command.

```
# kubeadm-ha-setup restore /backups/fulldir-1544115826/fullbackup-1544115827.tar
```

**Note**

A full restore of the high availability master cluster disrupts service availability for the duration of the restore operation

**Restore the cluster configuration and state**

1. On the master host, ensure that the latest Docker and Kubernetes versions are installed and that the master node IP address and hostname match the IP address and hostname used before failure. The `kubeadm` package pulls in all of the required dependencies, including the correct version of the Docker engine.

```
# yum install kubeadm kubectl kubelet kubeadm-ha-setup
```

2. Run the `kubeadm-ha-setup restore` command.
High Availability Cluster

Reading configuration file /usr/local/share/kubeadm/run/kubeadm/ha.yaml ...
CreateSSH /root/.ssh/id_rsa root
Restore /share/fulldir-1544115826/fullbackup-1544115827.tar
with binary /usr/bin/kubeadm-ha-setup
Checking etcd clusters health (this will take a few mins) ...
Cleaning up node 10.147.25.195
Cleaning up node 10.147.25.196
Cleaning up node 10.147.25.197
file to be restored from:  /share/fulldir-1544115826/fullbackup-10.147.25.195-1544115826.tar
Configuring keepalived for HA ...
success
file to be restored from:  /share/fulldir-1544115826/fullbackup-10.147.25.196-1544115826.tar
file to be restored from:  /share/fulldir-1544115826/fullbackup-10.147.25.197-1544115827.tar
[INFO]  /usr/local/share/kubeadm/kubeadm-ha/etcd-extract.sh/10.147.25.196:22 retrying ...
file to be restored from:  /share/fulldir-1544115826/fullbackup-10.147.25.197-1544115827.tar
[INFO]  /usr/bin/kubeadm-ha-setup etcd
fullrestore 10.147.25.197 10.147.25.197:22 retrying ...
[COMPLETED] Restore completed, cluster(s) may take a few minutes to get backup!

Substitute /backups/fulldir-1544115826/fullbackup-1544115827.tar with the full path to the backup file that you wish to restore. Note that the backup directory and file must be accessible to all master nodes in the cluster during the restore process.

If the script detects that all three master nodes are currently healthy, you need to confirm you wish to proceed:

[WARNING] All nodes are healthy !!! This will perform a FULL CLUSTER RESTORE
pressing [y] will restore cluster to the state stored
in /share/fulldir-1544115826/fullbackup-1544115827.tar
Alternatively if the script detects that more than one master node is unavailable then it prompts you before proceeding with a full cluster restore.

3. Copy the Kubernetes admin.conf file to your home directory:

```bash
$ sudo cp /etc/kubernetes/admin.conf $HOME/
```

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

```bash
$ sudo chown $(id -u):$(id -g) $HOME/admin.conf
```

Export the path to the file for the KUBECONFIG environment variable:

```bash
$ export KUBECONFIG=$HOME/admin.conf
```

You cannot use the kubectl command if the path to this file is not set for this environment variable. Remember to export the KUBECONFIG variable for each subsequent login so that the kubectl and kubeadm commands use the correct admin.conf file, otherwise you might find that these commands do not behave as expected after a reboot or a new login. For instance, append the export line to your .bashrc:

```bash
$ echo 'export KUBECONFIG=$HOME/admin.conf' >> $HOME/.bashrc
```

4. Check that you cluster has been properly restored. Use kubectl to check on the status of the nodes within the cluster and to check any existing configuration. For example:

```bash
$ kubectl get nodes
NAME                  STATUS    ROLES   AGE      VERSION
master1.example.com   Ready     master  1h       v1.12.5+2.1.1.el7
master2.example.com   Ready     master  1h       v1.12.5+2.1.1.el7
master3.example.com   Ready     master  1h       v1.12.5+2.1.1.el7
worker2.example.com   Ready     <none>  1h       v1.12.5+2.1.1.el7
```
4.4 Kubernetes Dashboard

When the `kubeadm-setup.sh` script or `kubeadm-ha-setup` utility is used to install master nodes in the Kubernetes cluster, 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 Kubernetes documentation at https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/.

To access 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:

```
# systemctl start kubectl-proxy
# systemctl enable 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
```

To access the Dashboard, open a web browser on the node where the proxy is running and navigate to `http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login`. To login, you must authenticate using a token. See https://github.com/kubernetes/dashboard/wiki/Access-control for more information. 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 \n  get secret -n kube-system -o name | grep namespace) | grep token:
token: eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJpc3MiOiJzdWIiLCJhdWQiOiJvbmV0d29tbW9uIiwiaWF0IjoxfQ.eyJ0eXBlIjoiRGF0ZSBXaXRoIiwiZXhwIjoxODg1ODgyMDI0fQ.5Z8J5ZJx
```

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

If you need to access the Dashboard remotely, Oracle recommends using SSH tunneling to do port forwarding from your localhost to the proxy node, as described in the following sections.
SSH Tunneling

The easiest option is to use SSH tunneling to forward a port on your local system to the port configured for the proxy service running on the node that you wish 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 \texttt{192.0.2.10} with the IP address of the host where you are running \texttt{kubectl proxy}. Once the SSH connection is established, you can open a browser on your localhost and navigate to \url{http://localhost:8001/api/v1/namespaces/kube-system/services/https:kubernetes-dashboard:/proxy/#!/login} to access the Dashboard hosted in the remote Kubernetes cluster. Use the same token information to authenticate as if you were connecting to the Dashboard locally.

4.5 Removing Worker Nodes from the Cluster

4.5.1 Single Master Cluster

At any point, you can remove a worker node from the cluster. Use the \texttt{kubeadm-setup.sh down} command to completely remove all of the Kubernetes components installed and running on the system. Since this operation is destructive, the script warns you when you attempt to do this on a worker node and requires confirmation to continue with the action. The script also reminds you that you need to remove the node from the cluster configuration:

```bash
# kubeadm-setup.sh down

[WARNING] This action will RESET this node !!!!!
  # kubectl delete node worker1.example.com
  Please select 1 (continue) or 2 (abort) :
  1) continue
  2) abort
  #? 1
```

The cluster must be updated so that it no longer looks for a node that you have decommissioned. Remove the node from the cluster using the \texttt{kubectl delete node} command:

```bash
$ kubectl delete node worker1.example.com
node "test2.example.com" deleted
```

Substitute \texttt{worker1.example.com} with the name of the worker node that you wish to remove from the cluster.

If you run the \texttt{kubeadm-setup.sh down} command on the master node, the only way to recover the cluster is to restore from a backup file. Doing this effectively destroys the entire cluster. The script warns you that this is a destructive action and that you are performing it on the master node. You must confirm the action before you are able to continue:

```bash
# kubeadm-setup.sh down

[WARNING] This action will RESET this node !!!!!
```
Since this is a master node, all of the clusters information will be lost !!!!
Please select 1 (continue) or 2 (abort) :
1) continue
2) abort
#? 1
[preflight] Running pre-flight checks
[reset] Stopping the kubelet service
[reset]Unmounting mounted directories in "/var/lib/kubelet"
[reset] Removing kubernetes-managed containers
[reset] Deleting contents of stateful directories: [/var/lib/kubelet /etc/cni/net.d \
 /var/lib/docker/ /var/lib/etcd]
[reset] Deleting contents of config directories: [/etc/kubernetes/manifests /etc/kubernetes/pki]
[reset] Deleting files: [/etc/kubernetes/admin.conf /etc/kubernetes/kubelet.conf \
 /etc/kubernetes/controller-manager.conf /etc/kubernetes/scheduler.conf]
deleting flannel.1 ip link ...
deleting cni0 ip link ...
removing /var/lib/cni directory ...
removing /var/lib/etcd directory ...
removing /etc/kubernetes directory ...

4.5.2 High Availability Cluster

Removing a worker node from a high availability cluster follows a similar process, but uses the kubeadm-
ha-setup down command instead. Since this operation is destructive, the utility warns you when you
attempt to do this on a worker node and requires confirmation to continue with the action. The script also
reminds you that you need to remove the node from the cluster configuration:

# kubeadm-ha-setup down
[WARNING] This operation will clean up all kubernetes installations on this node
press [y] to continue ...
y
[INFO] Removing interface flannel.1

The cluster must be updated so that it no longer looks for a node that you have decommissioned. Remove
the node from the cluster using the kubectl delete node command on any of your master nodes:

$ kubectl delete node worker1.example.com
node "worker1.example.com" deleted

Substitute worker1.example.com with the name of the node that you wish to remove from the cluster.

If you run the kubeadm-ha-setup command on any of your master nodes, the only way to recover the
cluster is to restore from a backup file.
Chapter 5 Getting Started with Kubernetes

Table of Contents

5.1 kubectl Basics ........................................................................................................... 75
5.2 Pod Configuration Using a YAML Deployment .......................................................... 78
5.3 Using Persistent Storage ......................................................................................... 82
  5.3.1 Persistent Storage Concepts ............................................................................... 83
  5.3.2 Configuring NFS .............................................................................................. 84
  5.3.3 Configuring iSCSI ............................................................................................ 86

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

5.1 kubectl Basics

The kubectl utility is a command line tool that interfaces with the API Server to run commands against the cluster. The tool is typically run on the master node of the cluster. It effectively grants full administrative rights to the cluster and all of the nodes in the cluster.

The kubectl utility is documented fully at:

https://kubernetes.io/docs/user-guide/kubectl-overview/

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

Get Information About the Nodes in a Cluster

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

<table>
<thead>
<tr>
<th>NAME</th>
<th>STATUS</th>
<th>ROLES</th>
<th>AGE</th>
<th>VERSION</th>
</tr>
</thead>
<tbody>
<tr>
<td>master.example.com</td>
<td>Ready</td>
<td>master</td>
<td>1h</td>
<td>v1.12.5+2.1.1.el7</td>
</tr>
<tr>
<td>worker1.example.com</td>
<td>Ready</td>
<td>&lt;none&gt;</td>
<td>1h</td>
<td>v1.12.5+2.1.1.el7</td>
</tr>
<tr>
<td>worker2.example.com</td>
<td>Ready</td>
<td>&lt;none&gt;</td>
<td>1h</td>
<td>v1.12.5+2.1.1.el7</td>
</tr>
</tbody>
</table>

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.com
Roles: <none>
Labels: beta.kubernetes.io/arch=amd64
        beta.kubernetes.io/os=linux
        kubernetes.io/hostname=worker1.example.com
Annotations: flannel.alpha.coreos.com/backend-data: {“VtepMAC”:“f2:24:33:ab:be:82”}
            flannel.alpha.coreos.com/backend-type: vxlan
            flannel.alpha.coreos.com/kube-subnet-manager: true
            flannel.alpha.coreos.com/public-ip: 10.147.25.196
            kubeadm.alpha.kubernetes.io/cri-socket: /var/run/dockershim.sock
            node.alpha.kubernetes.io/ttl: 0
Run an Application in a Pod

To create a pod with a single running Docker 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 Docker image that can be pulled by the Docker engine.

**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]*'. 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 -h`.

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:         Mon, 10 Dec 2018 08:25:17 -0800
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:   docker://417b4b59f7005eb4b175a1627e0f957e931c0cf24f1780cd94fa9949be1d31
    Image:          nginx
    Image ID:       docker-pullable://nginx@sha256:5d32f60db294b5deb55d078c4f2410ad88e6fe77500c87d3970eca97f
    Port:           <none>
    Host Port:      <none>
    State:          Running
    Started:        Mon, 10 Dec 2018 08:25:25 -0800
    Ready:          True
    Restart Count:  0
    Environment:    <none>
```
Scale 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:

```bash
$ 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:

```bash
$ kubectl get pods
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello-world-5f55779987-tswmg</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>18s</td>
</tr>
<tr>
<td>hello-world-5f55779987-v8w5h</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>26m</td>
</tr>
<tr>
<td>hello-world-5f55779987-wd857</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>18s</td>
</tr>
</tbody>
</table>

Expose a Service Object for Your 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:

```bash
$ 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:

```bash
$ kubectl get services
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>TYPE</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>hello-world</td>
<td>LoadBalancer</td>
<td>10.102.42.160</td>
<td>&lt;pending&gt;</td>
<td>80:31847/TCP</td>
<td>3s</td>
</tr>
<tr>
<td>kubernetes</td>
<td>ClusterIP</td>
<td>10.96.0.1</td>
<td>&lt;none&gt;</td>
<td>443/TCP</td>
<td>5h13m</td>
</tr>
</tbody>
</table>

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.

Delete 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
```

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
```

Work 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
```

<table>
<thead>
<tr>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
<th>RESTARTS</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>coredns-6c77847dcf-77grm</td>
<td>1/1</td>
<td>Running</td>
<td>2</td>
<td>5h26m</td>
</tr>
<tr>
<td>coredns-6c77847dcf-vtk8k</td>
<td>1/1</td>
<td>Running</td>
<td>2</td>
<td>5h26m</td>
</tr>
<tr>
<td>etcd-master.example.com</td>
<td>1/1</td>
<td>Running</td>
<td>3</td>
<td>5h25m</td>
</tr>
<tr>
<td>kube-apiserver-master.example.com</td>
<td>1/1</td>
<td>Running</td>
<td>4</td>
<td>5h25m</td>
</tr>
<tr>
<td>kube-controller-manager-master.example.com</td>
<td>1/1</td>
<td>Running</td>
<td>4</td>
<td>5h25m</td>
</tr>
<tr>
<td>kube-flannel-ds-4c285</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>115m</td>
</tr>
<tr>
<td>kube-flannel-ds-ds66r</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>115m</td>
</tr>
<tr>
<td>kube-proxy-5lsw</td>
<td>1/1</td>
<td>Running</td>
<td>0</td>
<td>117m</td>
</tr>
<tr>
<td>kube-proxy-tv2mj</td>
<td>1/1</td>
<td>Running</td>
<td>3</td>
<td>5h26m</td>
</tr>
<tr>
<td>kube-scheduler-master.example.com</td>
<td>1/1</td>
<td>Running</td>
<td>3</td>
<td>5h25m</td>
</tr>
<tr>
<td>kubernetes-dashboard-64458f66b6-q8dzh</td>
<td>1/1</td>
<td>Running</td>
<td>4</td>
<td>5h26m</td>
</tr>
</tbody>
</table>

5.2 Pod Configuration Using a YAML Deployment

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/.

Kubernetes deployment files can be easily shared and Kubernetes is also capable of creating a deployment based on a remotely hosted file, allowing anyone to get a deployment running in minutes. You can create a deployment by running the following command:

```bash
$ kubectl create -f https://example.com/deployment.yaml
```

In the following example, you will create two YAML deployment files. The first is used to create a deployment that runs MySQL Server with a persistent volume for its data store. You will also configure the services that allow other pods in the cluster to consume this resource.

The second deployment will run a phpMyAdmin container in a separate pod that will access the MySQL Server directly. That deployment will also create a `NodePort` service so that the phpMyAdmin interface can be accessed from outside of the Kubernetes cluster.

The following example illustrates how you can use YAML deployment files to define the scope and resources that you need to run a complete application.

---

**Important**

The examples provided here are provided for demonstration purposes only. They are not intended for production use and do not represent a preferred method of deployment or configuration.

---

### MySQL Server Deployment

To create the MySQL Server Deployment, create a single text file `mysql-db.yaml` in an editor. The description here provides a breakdown of each of the objects as they are defined in the text file. All of these definitions can appear in the same file.

One problem when running databases within containers is that containers are not persistent. This means that data hosted in the database must be stored outside of the container itself. Kubernetes handles setting up these persistent data stores in the form of Persistent Volumes. There are a wide variety of Persistent Volume types. In a production environment, some kind of shared file system that is accessible to all nodes in the cluster would be the most appropriate implementation choice, however for this simple example you will use the `hostPath` type. The `hostPath` type allows you to use a local disk on the node where the container is running.

In the Persistent Volume specification, we can define the size of the storage that should be dedicated for this purpose and the access modes that should be supported. For the `hostPath` type, the path where the data should be stored is also defined. In this case, we use the path `/tmp/data` for demonstration purposes. These parameters should be changed according to your own requirements.

The definition in the YAML file for the Persistent Volume object should appear similarly to the following:

```yaml
apiVersion: v1
kind: PersistentVolume
metadata:
  name: mysql-pv-volume
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 5Gi
```

---
A Persistent Volume object is an entity within Kubernetes that stands on its own as a resource. For a pod to use this resource, it must request access and abide by the rules applied to its claim for access. This is defined in the form of a Persistent Volume Claim. Pods effectively mount Persistent Volume Claims as their storage.

The definition in the YAML file for the Persistent Volume Claim object should appear similarly to the following:

```yaml
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: mysql-pv-claim
spec:
  storageClassName: manual
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
```

It is important to define a service for the deployment. This specifies the TCP ports used by the application that we intend to run in our pod. In this case, the MySQL server listens on port 3306. Most importantly, the name of the service can be used by other deployments to access this service within the cluster, regardless of the node where it is running. This service does not specify a service type as it uses the default `ClusterIP` type so that it is only accessible to other components running in the cluster internal network. In this way, the MySQL server is isolated to requests from containers running in pods within the Kubernetes cluster.

The Service definition in the YAML file might look as follows:

```yaml
---
apiVersion: v1
kind: Service
metadata:
  name: mysql-service
spec:
  selector:
    app: mysql
  ports:
  - port: 3306
    clusterIP: None
```

A MySQL Server instance can be easily created as a Docker container running in a pod, using the `mysql/mysql-server:latest` Docker image. In the pod definition, specify the volume information to attach the Persistent Volume Claim that was defined previously for this purpose. Also, specify the container parameters, including the image that should be used, the container ports that are used, volume mount points and any environment variables required to run the container. In this case, we mount the Persistent Volume Claim onto `/var/lib/mysql` in each running container instance and we specify the `MYSQL_ROOT_PASSWORD` value as an environment variable, as required by the image.

```yaml
---
apiVersion: v1
kind: Pod
```
phpMyAdmin Deployment

metadata:
  - name: mysql
  - app: mysql
spec:
  volumes:
    - name: mysql-pv-storage
      persistentVolumeClaim:
        claimName: mysql-pv-claim
      containers:
        - image: mysql/mysql-server:latest
          name: mysql
          ports:
            - containerPort: 3306
              name: mysql
          volumeMounts:
            - mountPath: /var/lib/mysql
              name: mysql-pv-storage
          env:
            - name: MYSQL_ROOT_PASSWORD
              value: "password"

Replace the password value specified for the MYSQL_ROOT_PASSWORD environment variable with a better alternative, suited to your security requirements.

When you have created your YAML deployment file, save it and then run:

$ kubectl create -f mysql-db.yaml
persistentvolume/mysql-pv-volume created
persistentvolumeclaim/mysql-pv-claim created
service/mysql-service created
pod/mysql created

All of the resources and components defined in the file are created and loaded in Kubernetes. You can use the kubectl command to view details of each component as you require.

phpMyAdmin Deployment

To demonstrate how deployments can interconnect and consume services provided by one another, it is possible to set up a phpMyAdmin Docker instance that connects to the backend MySQL server that you deployed in the first part of this example.

The phpMyAdmin deployment uses a standard Docker image to create a container running in a pod, and also defines a NodePort service that allows the web interface to be accessed from any node in the cluster.

Create a new file called phpmyadmin.yaml and open it in an editor to add the two component definitions described in the following text.

First, create the Service definition. This service defines the port that is used in the container and the targetPort that this is mapped to within the internal Kubernetes cluster network. Also specify the Service type and set it to NodePort, to make the service accessible from outside of the cluster network via any of the cluster nodes and the port forwarding service that the NodePort service type provides.

The declaration should look similar to the following:

apiVersion: v1
kind: Service
metadata:
  labels:
    name: phpmyadmin
Finally, define the pod where the phpMyAdmin container is loaded. Here, you can specify the Docker image that should be used for this container and the port that the container uses. You can also specify the environment variables required to run this image. Notably, the Docker image requires you to set the environment variable `PMA_HOST`, which should provide the IP address or resolvable domain name for the MySQL server. Since we cannot guess which IP address should be used here, we can rely on Kubernetes to take care of this, by providing the `mysql-service` name as the value here. Kubernetes automatically links the two pods using this service definition.

The Pod definition should look similar to the following:

```yaml
apiVersion: v1
class: Pod
metadata:
  name: phpmyadmin
spec:
  containers:
    - name: phpmyadmin
      image: phpmyadmin/phpmyadmin
      env:
        - name: PMA_HOST
          value: mysql-service
      ports:
        - containerPort: 80
          name: phpmyadmin
```

Save the file and then run the `kubectl create` command to load the YAML file into a deployment.

`kubectl create -f phpmyadmin.yaml`

`service/phpmyadmin created`

`pod/phpmyadmin created`

To check that this is working as expected, you need to determine what port is being used for the port forwarding provided by the `NodePort` service:

`kubectl get services phpmyadmin`

<table>
<thead>
<tr>
<th>NAME</th>
<th>CLUSTER-IP</th>
<th>EXTERNAL-IP</th>
<th>PORT(S)</th>
<th>AGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>phpmyadmin</td>
<td>10.110.16.56</td>
<td>&lt;nodes&gt;</td>
<td>80:31485/TCP</td>
<td>1d</td>
</tr>
</tbody>
</table>

In this example output, port 80 on the cluster network is being mapped to port 30582 on each of the cluster nodes. Open a browser to point to any of the cluster nodes on the specified port mapping. For example: `http://master.example.com:31485/`. You should be presented with the phpMyAdmin login page and you should be able to log into phpMyAdmin as root with the password that you specified as the `MYSQL_ROOT_PASSWORD` environment variable when you deployed the MySQL server.

### 5.3 Using Persistent Storage

The concept of using persistent storage for a database deployment was introduced in the previous section, Section 5.2, "Pod Configuration Using a YAML Deployment". Persistent storage is essential when working
with stateful applications like databases, as it is important that you are able to retain data beyond the lifecycle of the container, or even of the pod, itself.

Persistent storage, in Kubernetes, is handled in the form of PersistentVolume objects and are bound to pods using PersistentVolumeClaims. PersistentVolumes can be hosted locally or can be hosted on networked storage devices or services.

While it is convenient to use the hostPath persistent volume type to store data on the local disk in a demonstration or small-scale deployment, a typical Kubernetes environment involves multiple hosts and usually includes some type of networked storage. Using networked storage helps to ensure resilience and allows you to take full advantage of a clustered environment. In the case where the node where a pod is running fails, a new pod can be started on an alternate node and storage access can be resumed. This is particularly important for database environments where replica setup has been properly configured.

In this section, we continue to explore the Kubernetes components that are used to configure persistent storage, with the focus on using networked storage to host data.

5.3.1 Persistent Storage Concepts

Persistent storage is provided in Kubernetes using the PersistentVolume subsystem. To configure persistent storage, you should be familiar with the following terms:

- **PersistentVolume.** A PersistentVolume defines the type of storage that is being used and the method used to connect to it. This is the real disk or networked storage service that is used to store data.

- **PersistentVolumeClaim.** A PersistentVolumeClaim defines the parameters that a consumer, like a pod, uses to bind the PersistentVolume. The claim may specify quota and access modes that should be applied to the resource for a consumer. A pod can use a PersistentVolumeClaim to gain access to the volume and mount it.

- **StorageClass.** A StorageClass is an object that specifies a volume plugin, known as a provisioner that allows users to define PersistentVolumeClaims without needing to preconfigure the storage for a PersistentVolume. This can be used to provide access to similar volume types as a pooled resource that can be dynamically provisioned for the lifecycle of a PersistentVolumeClaim.

PersistentVolumes can be provisioned either statically or dynamically.

Static PersistentVolumes are manually created and contain the details required to access real storage and can be consumed directly by any pod that has an associated PersistentVolumeClaim.

Dynamic PersistentVolumes can be automatically generated if a PersistentVolumeClaim does not match an existing static PersistentVolume and an existing StorageClass is requested in the claim. A StorageClass can be defined to host a pool of storage that can be accessed dynamically. Creating a StorageClass is an optional step that is only required if you intend to use dynamic provisioning.

The process to provision persistent storage is as follows:

1. Create a PersistentVolume or StorageClass.
2. Create PersistentVolumeClaims.
3. Configure a pod to use the PersistentVolumeClaim.

The examples, here, assume that you have configured storage manually and that you are using static provisioning. In each case, a PersistentVolume is configured, the PersistentVolumeClaim is created, and finally a pod is created to use the PersistentVolumeClaim.
5.3.2 Configuring NFS

In this example, it is assumed that an NFS appliance is already configured to allow access to all of the nodes in the cluster. Note that if your NFS appliance is hosted on Oracle Cloud Infrastructure, you must create ingress rules in the security list for the Virtual Cloud Network (VCN) subnet that you are using to host your Kubernetes nodes. The rules must be set to allow traffic on ports 2049 and 20049 for NFS Access and NFS Mount.

Each worker node within the cluster must also have the `nfs-utils` package installed:

```
# yum install nfs-utils
```

The following steps describe a deployment using YAML files for each object:

1. Create a PhysicalVolume object in a YAML file. For example, on the master node, create a file `pv-nfs.yml` and open it in an editor to include the following content:

   ```yaml
   apiVersion: v1
   kind: PersistentVolume
   metadata:
     name: nfs
   spec:
     capacity:
       storage: 1Gi
     accessModes:
     - ReadWriteMany
     nfs:
       server: 192.0.2.100
       path: /nfsshare
   ```

   Replace `1Gi` with the size of the storage available. Replace `192.0.2.100` with the IP address of the NFS appliance in your environment. Replace `/nfsshare` with the exported share name on your NFS appliance.

2. Create the PersistentVolume using the YAML file you have just created, by running the following command on the master node:

   ```
   $ kubectl create -f pv-nfs.yml
   persistentvolume/nfs created
   ```

3. Create a PhysicalVolumeClaim object in a YAML file. For example, on the master node, create a file `pvc-nfs.yml` and open it in an editor to include the following content:

   ```yaml
   apiVersion: v1
   kind: PersistentVolumeClaim
   metadata:
     name: nfs
   spec:
     accessModes:
     - ReadWriteMany
     resources:
       requests:
       storage: 1Gi
   ```

   Note that you can change the `accessModes` by changing the `ReadWriteMany` value, as required. You can also change the quota available in this claim, by changing the value of the `storage` option from `1Gi` to some other value.

4. Create the PersistentVolumeClaim using the YAML file you have just created, by running the following command on the master node:

   ```
   $ kubectl create -f pvc-nfs.yml
   ```
Configuring NFS

5. Check that the PersistentVolume and PersistentVolumeClaim have been created properly and that the PersistentVolumeClaim is bound to the correct volume:

```
$ kubectl get pv,pvc
NAME      CAPACITY   ACCESSMODES   RECLAIMPOLICY   STATUS     CLAIM         STORAGECLASS   REASON    AGE
pv/nfs    1Gi        RWX           Retain         Bound      default/nfs                            7m
NAME        STATUS   VOLUME    CAPACITY   ACCESSMODES   STORAGECLASS   AGE
pvc/nfs     Bound    nfs       1Gi        RWX                          2m
```

6. At this point, you can set up pods that can use the PersistentVolumeClaim to bind to the PersistentVolume and use the resources that are available there. In the example steps that follow, a ReplicationController is used to set up two replica pods running web servers that use the PersistentVolumeClaim to mount the PersistentVolume onto a mountpath containing shared resources.

   a. Create a ReplicationController object in a YAML file. For example, on the master node, create a file `rc-nfs.yml` and open it in an editor to include the following content:

   ```yaml
   apiVersion: v1
   kind: ReplicationController
   metadata:
     name: rc-nfs-test
   spec:
     replicas: 2
     selector:
       app: nginx
     template:
       metadata:
         labels:
           app: nginx
       spec:
         containers:
           - name: nginx
             image: nginx
             ports:
               - name: nginx
                 containerPort: 80
             volumeMounts:
               - name: nfs
                 mountPath: "/usr/share/nginx/html"
         volumes:
           - name: nfs
             persistentVolumeClaim:
               claimName: nfs
   
   b. Create the ReplicationController using the YAML file you have just created, by running the following command on the master node:

   ```
   $ kubectl create -f rc-nfs.yml
   replicationcontroller/rc-nfs-test created
   ```

   c. Check that the pods have been created:

   ```
   $ kubectl get pods
   NAME                 READY STATUS    RESTARTS AGE
   rc-nfs-test-c5440   1/1   Running   0   54s
   rc-nfs-test-8997k   1/1   Running   0   54s
   ```

   d. On the NFS appliance, create an index file in the `/nfsshare` export, to test that the web server pods have access to this resource. For example:

   ```
   $ echo "This file is available on NFS" > /nfsshare/index.html
   ```
e. You can either create a service to expose the web server ports so that you are able to check the output of the web server, or you can simply view the contents in the /usr/share/nginx/html folder on each pod, since the NFS share should be mounted onto this directory in each instance. For example, on the master node:

```
$ kubectl exec rc-nfs-test-c5440 cat /usr/share/nginx/html/index.html
This file is available on NFS
$ kubectl exec rc-nfs-test-8997k cat /usr/share/nginx/html/index.html
This file is available on NFS
```

You can experiment further by shutting down a node where a pod is running. A new pod is spawned on a running node and instantly has access to the data on the NFS share. In this way, you can demonstrate data persistence and resilience during node failure.

### 5.3.3 Configuring iSCSI

In this example, it is assumed that an iSCSI service is already configured to expose a block device, as an iSCSI LUN, to all of the nodes in the cluster. Note that if your iSCSI server is hosted on Oracle Cloud Infrastructure, you must create ingress rules in the security list for the Virtual Cloud Network (VCN) subnet that you are using to host your Kubernetes nodes. The rules must be set to allow traffic on ports 860 and 3260.

Each worker node within the cluster must also have the `iscsi-initiator-utils` package installed:

```
# yum install iscsi-initiator-utils
```

You must manually edit the `/etc/iscsi/initiatorname.iscsi` file on all nodes of cluster to add the initiator name (`iqn`) of the device. Restart the `iscsid` service once you have edited this file.

For more information on configuring iSCSI on Oracle Linux 7, see the *Oracle Linux Administrator's Guide for Release 7* at:


The following steps describe a deployment using YAML files for each object:

1. Create a `PersistentVolume` object in a YAML file. For example, on the master node, create a file `pv-iscsi.yml` and open it in an editor to include the following content:

```yaml
apiVersion: v1
type: PersistentVolume
metadata:
  name: iscsi-pv
spec:
  capacity:
    storage: 12Gi
  accessModes:
    - ReadWriteOnce
  iscsi:
    targetPortal: 192.0.2.100:3260
    iqn: iqn.2017-10.local.example.server:disk1
    lun: 0
    fsType: 'ext4'
    readOnly: false
```

Replace `12Gi` with the size of the storage available. Replace `192.0.2.100:3260` with the IP address and port number of the iSCSI target in your environment. Replace `iqn.2017-10.local.example.server:disk1` with the `iqn` for the device that you wish to use via iSCSI.
2. Create the PersistentVolume using the YAML file you have just created, by running the following command on the master node:

```
$ kubectl create -f pv-iscsi.yml
persistentvolume/iscsi-pv created
```

3. Create a PhysicalVolumeClaim object in a YAML file. For example, on the master node, create a file `pvc-iscsi.yml` and open it in an editor to include the following content:

```yaml
kind: PersistentVolumeClaim
apiVersion: v1
metadata:
  name: iscsi-pvc
spec:
  accessModes:
  - ReadWriteOnce
  resources:
    requests:
      storage: 12Gi
```

Note that you can change the `accessModes` by changing the `ReadWriteOnce` value, as required. Supported modes for iSCSI include `ReadWriteOnce` and `ReadOnlyMany`. You can also change the quota available in this claim, by changing the value of the `storage` option from `12Gi` to some other value.

Note that with iSCSI, support for both read and write operations limit you to hosting all of your pods on a single node. The scheduler automatically ensures that pods with the same PersistentVolumeClaim run on the same worker node.

4. Create the PersistentVolumeClaim using the YAML file you have just created, by running the following command on the master node:

```
$ kubectl create -f pvc-iscsi.yml
persistentvolumeclaim/iscsi-pvc created
```

5. Check that the PersistentVolume and PersistentVolumeClaim have been created properly and that the PersistentVolumeClaim is bound to the correct volume:

```
$ kubectl get pv,pvc
NAME         CAPACITY ACCESSMODES  RECLAIMPOLICY STATUS  CLAIM          STORAGECLASS REASON  AGE
pv/iscsi-pv  12Gi     RWX          Retain        Bound   default/iscsi-pvc                   25s
NAME            STATUS    VOLUME     CAPACITY   ACCESSMODES   STORAGECLASS   AGE
pvc/iscsi-pvc   Bound     iscsi-pv   12Gi       RWX                          21s
```

6. At this point you can set up pods that can use the PersistentVolumeClaim to bind to the PersistentVolume and use the resources available there. In the following example, a ReplicationController is used to set up two replica pods running web servers that use the PersistentVolumeClaim to mount the PersistentVolume onto a mountpath containing shared resources.

a. Create a ReplicationController object in a YAML file. For example, on the master node, create a file `rc-iscsi.yml` and open it in an editor to include the following content:

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: rc-iscsi-test
spec:
  replicas: 2
  selector:
    app: nginx
  template:
```
metadata:
  labels:
    app: nginx
spec:
  containers:
  - name: nginx
    image: nginx
    ports:
      - name: nginx
        containerPort: 80
    volumeMounts:
      - name: iscsi
        mountPath: "/usr/share/nginx/html"
  volumes:
  - name: iscsi
    persistentVolumeClaim:
      claimName: iscsi-pvc

b. Create the ReplicationController using the YAML file you have just created, by running the following command on the master node:

```bash
$ kubectl create -f rc-iscsi.yml
replicationcontroller "rc-iscsi-test" created
```

c. Check that the pods have been created:

```bash
$ kubectl get pods
NAME                  READY     STATUS    RESTARTS   AGE
rc-iscsi-test-05kdr   1/1       Running   0          9m
rc-iscsi-test-wv4p5   1/1       Running   0          9m
```

d. On any host where the iSCSI LUN can be mounted, mount the LUN and create an index file, to test that the web server pods have access to this resource. For example:

```bash
# mount /dev/disk/by-path/ip-192.0.2.100:3260-iscsi-qn.2017-10.local.example.server:disk1-lun-0 /mnt
$ echo "This file is available on iSCSI" > /mnt/index.html
```

e. You can either create a service to expose the web server ports so that you are able to check the output of the web server, or you can simply view the contents in the `/usr/share/nginx/html` folder on each pod, since the NFS share should be mounted onto this directory in each instance. For example, on the master node:

```bash
$ kubectl exec rc-nfs-test-c5440 cat /usr/share/nginx/html/index.html
This file is available on iSCSI
$ kubectl exec rc-nfs-test-8997k cat /usr/share/nginx/html/index.html
This file is available on iSCSI
```
Chapter 6 For More Information About Kubernetes

For more information about Kubernetes, see https://kubernetes.io/ and https://github.com/kubernetes/kubernetes.
Warning

Oracle does not support Kubernetes on systems where the `ol7_preview`, `ol7_developer` or `ol7_developer_EPEL` yum repositories or ULN channels are enabled, or where software from these repositories, or channels, is currently installed on the systems where Kubernetes runs. If you follow the instructions in this section, you may render your platform unsupported if these repositories or channels are enabled or software from these channels or repositories is installed on your system.

Oracle makes interim releases of Oracle Container Services for use with Kubernetes available as technical previews. These releases are not supported by Oracle and are not intended for production use.

Developer preview releases can be obtained by enabling the `ol7_developer` repository on the Oracle Linux yum server:

```
# yum-config-manager --enable ol7_developer
```

The packages from this repository are intended for use with the images provided in the Container Services (Developer) Repositories on the Oracle Container Registry. You must login and accept the terms and conditions to use these.

If you install developer preview packages, make sure that you are pulling the correct images for the release by setting the `KUBE_REPO_PREFIX` environment variable:

```
# export KUBE_REPO_PREFIX=container-registry.oracle.com/kubernetes_developer
```

The installation procedures described in this guide should continue to apply for each developer preview release. You can find older preview releases in `ol7_preview`, but the use of that channel for Oracle Container Services for use with Kubernetes is now deprecated.

Important

Oracle does not support any upgrade from a developer preview release to a stable and supported release. Oracle does not support upgrade from a supported release to a newer developer preview.