Oracle Legal Notices

Copyright © 2015, 2017, Oracle and/or its affiliates. All rights reserved.

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

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

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

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

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

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

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

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

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.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Abstract

Document generated on: 2017-04-04 (revision: 919)
Table of Contents

Preface ........................................................................................................................................ v
1 Introduction to Oracle OpenStack for Oracle Linux ................................................................. 1
  1.1 Introduction to OpenStack ........................................................................................................ 1
  1.2 Introduction to Oracle VM Server ............................................................................................ 4
2 Getting Started .......................................................................................................................... 7
  2.1 Getting Started with Deployment ............................................................................................ 7
  2.2 Using Groups to Deploy Services ............................................................................................ 9
  2.3 Using the kollacli Command .................................................................................................. 10
3 Preparing the Basic Environment ............................................................................................... 13
  3.1 System Requirements .............................................................................................................. 13
  3.2 OpenStack Data Storage ......................................................................................................... 15
  3.3 Host Network Requirements ................................................................................................... 16
  3.4 Preparing Oracle Linux Nodes ................................................................................................ 18
  3.5 Preparing Oracle VM Server Compute Nodes ......................................................................... 26
  3.6 Preparing a Master Node ........................................................................................................ 32
  3.7 Setting up the Docker Registry ............................................................................................... 33
  3.7.1 Setting up the Docker Registry Server ............................................................................... 33
  3.7.2 Importing Images into the Docker Registry .................................................................... 37
4 Setting up a Deployment ............................................................................................................. 39
  4.1 Setting up Target Nodes ........................................................................................................... 39
  4.2 Setting up Services .................................................................................................................. 41
  4.3 Example Setup for an Initial Deployment ............................................................................... 44
  4.4 Setting Properties for Groups or Hosts ................................................................................... 48
      4.4.1 Dealing With Variable Network Interface Names ................................................................. 50
  4.5 Setting up Glance .................................................................................................................... 51
  4.6 Setting up Nova ....................................................................................................................... 52
      4.6.1 Automatic Hypervisor Configuration ............................................................................... 52
      4.6.2 Preparing a Compute Node ............................................................................................. 53
      4.6.3 Setting the iSCSI Initiator Name ......................................................................................... 53
      4.6.4 Enabling iSCSI Multipath ................................................................................................. 54
  4.7 Setting up Neutron ................................................................................................................... 54
      4.7.1 Configuring VLAN Networks ............................................................................................ 54
  4.8 Setting up Cinder ..................................................................................................................... 55
      4.8.1 Creating the cinder-volumes Volume Group ....................................................................... 55
      4.8.2 Configuring Alternative Volume Drivers for Cinder ......................................................... 56
      4.8.3 Configuring the Cinder NFS Backup Driver ...................................................................... 57
  4.9 Setting up Swift ........................................................................................................................ 57
  4.10 Setting up Ceilometer ............................................................................................................ 59
  4.11 Setting up Murano .................................................................................................................. 60
      4.11.1 Creating Oracle Linux Murano-enabled Images ................................................................. 60
5 Deploying Services ..................................................................................................................... 65
  5.1 Performing a Deployment ........................................................................................................ 65
  5.2 Making Configuration Changes After Deployment ................................................................. 67
  5.3 Updating to a New Release of Oracle OpenStack for Oracle Linux .................................. 67
  5.4 Removing Services .................................................................................................................. 68
6 Using OpenStack CLIs ............................................................................................................... 69
  6.1 Installing openstack-kolla-utils ............................................................................................. 69
  6.2 Using docker-ostk .................................................................................................................. 70
  6.3 Accessing the File System with docker-ostk ......................................................................... 71
7 Troubleshooting Deployment ...................................................................................................... 73
  7.1 Debugging a Deployment ........................................................................................................ 73
7.2 "Command Failed" Error When Starting the Ansible Container 73
7.3 Collecting the Log Files 74
7.4 Removing the Docker Images From a Node 75
A kollacli Command-Line Reference 77
Preface

This document is part of the documentation library for Oracle OpenStack for Oracle Linux Release 2.1, which is available at:

http://www.oracle.com/technetwork/server-storage/openstack/linux/documentation/

The documentation library consists of the following items:

**Oracle OpenStack for Oracle Linux Release Notes**

The release notes provide a summary of the new features, changes, fixed bugs and known issues in Oracle OpenStack for Oracle Linux. It contains last-minute information, which may not be included in the main body of documentation, and information on Oracle OpenStack for Oracle Linux support. Read this document before you install your environment.

**Oracle OpenStack for Oracle Linux Installation and Deployment Guide**

This document guides you through different options for installing and deploying Oracle OpenStack for Oracle Linux. It is intended for system administrators, and assumes that you are familiar with the Oracle Linux operating system, virtualization in general, and web technologies.

Further Information

You can get the latest information on Oracle OpenStack for Oracle Linux at:


OpenStack has a rich set of features and services. To find out more about existing and new OpenStack features, see the OpenStack documentation available at:

http://docs.openstack.org/kilo/

Conventions

The following text conventions are used in this document:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>boldface</strong></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>

Command Syntax

Command syntax appears in monospace font. The dollar character ($) and number sign (#) are command prompts. You do not enter them as part of the command. Commands that any user, including the root user, can run are shown with the $ prompt:

```
$ command
```
Commands that must be run as the root user, or by a user with superuser privileges obtained through another utility such as sudo, are shown with the # prompt:

```
# command
```

The following command syntax conventions are used in this guide:

<table>
<thead>
<tr>
<th>Convention</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>backslash \</td>
<td>A backslash is the Oracle Linux command continuation character. It is used in command examples that are too long to fit on a single line. Enter the command as displayed (with a backslash) or enter it on a single line without a backslash:</td>
</tr>
<tr>
<td></td>
<td>dd if=/dev/rdsk/c0t1d0s6 of=/dev/rst0 bs=10b \</td>
</tr>
<tr>
<td></td>
<td>count=10000</td>
</tr>
<tr>
<td>braces {  }</td>
<td>Braces indicate required items:</td>
</tr>
<tr>
<td></td>
<td>.DEFINE {macro1}</td>
</tr>
<tr>
<td>brackets [ ]</td>
<td>Brackets indicate optional items:</td>
</tr>
<tr>
<td></td>
<td>cvtcrt termname [outfile]</td>
</tr>
<tr>
<td>ellipses ...</td>
<td>Ellipses indicate an arbitrary number of similar items:</td>
</tr>
<tr>
<td></td>
<td>CHKVAL fieldname value1 value2 ... valueN</td>
</tr>
<tr>
<td>italics</td>
<td>Italic type indicates a variable. Substitute a value for the variable:</td>
</tr>
<tr>
<td></td>
<td>library_name</td>
</tr>
<tr>
<td>vertical line</td>
<td>A vertical line indicates a choice within braces or brackets:</td>
</tr>
<tr>
<td></td>
<td>FILE filesize [K</td>
</tr>
</tbody>
</table>
Chapter 1 Introduction to Oracle OpenStack for Oracle Linux

Table of Contents

1.1 Introduction to OpenStack ................................................................. 1
1.2 Introduction to Oracle VM Server ...................................................... 4

This chapter introduces you to Oracle OpenStack for Oracle Linux, to the OpenStack components and terminology.

1.1 Introduction to OpenStack

OpenStack is a cloud computing software platform that controls large pools of compute, storage, and networking resources in a data center. With OpenStack, you can manage different types of hypervisors, network devices and services, storage components, and more, using a single API that creates a unified data center fabric. OpenStack is, therefore, a pluggable framework that allows vendors to write plug-ins to implement a solution using their own technology, and which allows users to integrate their technology of choice.

OpenStack Services

OpenStack is built as a set of distributed services. These services communicate with each other, and are responsible for the various functions expected from virtualization/cloud management software. The following are some of the key OpenStack services:

- **Nova**: A compute service responsible for creating virtual machine instances and managing their life cycle, as well as managing the hypervisor of choice. The hypervisors are pluggable to Nova, while the Nova API remains the same, regardless of the underlying hypervisor.

- **Neutron**: A network service responsible for creating network connectivity and network services. It is capable of connecting with vendor network hardware via plug-ins. Neutron comes with a set of default services implemented by common tools. Network vendors can create plug-ins to replace any one of the services with their own implementation, adding value to their users.

- **Cinder**: A block storage service responsible for creating and managing external storage, including block devices and NFS. It is capable of connecting to vendor storage hardware through plug-ins. Cinder has several generic plug-ins, which can connect to NFS and iSCSI, for example. Vendors add value by creating dedicated plug-ins for their storage devices.

- **Swift**: An object and Binary Large Object (BLOB) storage service responsible for managing object-based storage.

- **Keystone**: An identity management system responsible for user and service authentication. Keystone is capable of integrating with third-party directory services such as LDAP.

- **Glance**: An image service responsible for managing images uploaded by users. Glance is not a storage service, but it is responsible for saving image attributes, making a virtual catalog of the images.

- **Heat**: An orchestration service responsible for managing the life cycle of the OpenStack infrastructure (such as servers, floating IP addresses, volumes, security groups, and so on) and applications. Uses Heat Orchestration Templates (HOT) to describe the infrastructure for an application and provides an API for Amazon's AWS template format.

- **Horizon**: A dashboard that creates a GUI for users to control the OpenStack deployment. This is an extensible framework to which vendors can add features. Horizon uses the same APIs exposed to users.
OpenStack Nodes

- **Murano**: An application catalog service for publishing cloud-ready applications from a catalog. An agent is installed into an instance's operating system, which enables deployment of the applications directly into the guest. Murano also includes a plug-in to the Horizon dashboard.

- **Ceilometer**: A telemetry service that collects, normalizes and transforms data produced by OpenStack services for various telemetry use cases, such as customer billing, resource tracking, metering, and alarming.

More details are available in the *OpenStack Cloud Administrator Guide* at:

http://docs.openstack.org/admin-guide-cloud/common/get_started_openstack_services.html

OpenStack has many more services that are responsible for various features and capabilities, and the full list can be found on the OpenStack website at:

http://www.openstack.org/

The list provided here is limited to those needed to get started with Oracle OpenStack for Oracle Linux.

### OpenStack Nodes

There are a number of node types used in OpenStack. Nodes are a physical host computer, with an operating system installed, with Oracle Linux using KVM (Kernel-based Virtual Machine), or Oracle VM Server. The main node types we discuss in this guide are:

- A **controller node** is a system running Oracle Linux, and is where most of the OpenStack services are installed. The term controller node is used to describe nodes that do not run virtual machine instances. The controller nodes may have all the non-compute services or only some of them. A controller node may also include the Oracle OpenStack for Oracle Linux CLI (kollacli), which is used to perform the deployment of OpenStack services to other nodes.

- A **compute node** is a system running Oracle Linux using KVM, or Oracle VM Server. A compute node runs the bare minimum of services to manage virtual machine instances.

- A **database node** is a system running Oracle Linux, and the services required to manage databases for images and instances.

- A **network node** is a system running Oracle Linux, and runs the neutron network worker daemon. The neutron worker daemon provides services such as providing an IP address to a booting Nova instance.

- A **storage node** is a system running Oracle Linux and the services required to manage storage for images and instances.

Some storage is not directly managed by the OpenStack services, but is instead managed by the storage appliance. On the storage node, Cinder communicates with the storage appliance’s API, and it is the storage appliance that performs the storage management. For example, when using the Oracle ZFS Storage Appliance, the Cinder driver on the storage node communicates with the Oracle ZFS Storage Appliance NFS driver, and it is the ZFS driver which performs the storage management.

- A **master node** is a system running Oracle Linux and kollacli, used to deploy the OpenStack services to the nodes. A master node is not an OpenStack node, although kollacli may be installed on a controller node.

More detailed information on the node types is available in the *OpenStack Operations Guide* at:

http://docs.openstack.org/openstack-ops/content/example_architecture.html#node_types
OpenStack Instances

OpenStack Instances

OpenStack virtual machines are called *instances*, mostly because they are instances of an image that is created upon request and that is configured when launched. The main difference between OpenStack and traditional virtualization technology is the way state is stored. With traditional virtualization technology, the state of the virtual machine is persistent.

OpenStack can support both *persistent* and *ephemeral* models. In the ephemeral model, an instance is launched from an image in the Image service, the image is copied to the run area, and when the copy is completed, the instance starts running. The size and connectivity of the instance are defined at the time of launching the instance. When an instance is terminated, the original image remains intact, but the state of the terminated instance is not retained. This ephemeral model is useful for scaling out quickly and maintaining agility for users.

In the persistent model, the instance is launched from a persistent volume on a compute node, or from a block storage volume, and not from the Image service. A volume can be any kind of persistent storage, including a file, a block device, an LVM partition, or any other form of persistent storage. In this case, when the instance is terminated, any session changes are retained and are present the next time an instance is launched. In the persistent model, the size and connectivity of the instance are also defined at the time the instance launches. In some sense, the persistent model in OpenStack is similar to the traditional approach to virtualization.

OpenStack Storage

The storage used in OpenStack can be either ephemeral or persistent. Ephemeral storage is deleted when an instance is terminated, while persistent storage remains intact. Persistent storage in OpenStack is referred to as a *volume*, regardless of the technology and device it is backed by. Persistent storage can either be used to launch an instance, or it can be connected to an instance as a secondary storage device to retain state. An example of this is a database launched as an ephemeral instance, with a volume connected to it to save the data. When the instance is terminated, the volume retains the data and can be connected to another instance as needed.

The OpenStack Cinder service is responsible for managing the volumes, and it offers a framework for vendors to create drivers. If a storage vendor wants to support OpenStack deployment and allow users to create volumes on the device, the vendor must create a Cinder driver that allows users to use the standard calls to control the storage device.

OpenStack also supports object storage using the Swift service.

OpenStack Networking

The OpenStack networking service, Neutron, offers a complete software-defined networking (SDN) solution, along with various network services. The network services Neutron can support include routing, firewall, DNS, DHCP, load balancing, VPN, and more.

Neutron, like Cinder, offers a framework for vendors to write plug-ins for various services. For example, a network vendor might want to offer a custom load balancer instead of the default load balancer provided by Neutron. The plug-in framework offers a powerful tool to build sophisticated network topologies using standard APIs.

Network Isolation: Tenant Networks

Tenant networks are the basis for Neutron’s SDN capability. Neutron has full control of layer-2 isolation. This automatic management of layer-2 isolation is completely hidden from the user, providing a convenient abstraction layer required by SDN.
To perform the layer-2 separation, Neutron supports three layer-2 isolation mechanisms: VLANs, VxLANs, and GRE (Generic Routing Encapsulation) tunnels. You must define which mechanism should be used and set up the physical topology as required. Neutron is responsible for allocating the resources as needed. For example, you would configure the VLAN switch, allocate the VLAN range, and configure the VLAN in Neutron. When you define a new network, Neutron automatically allocates a VLAN and takes care of the isolation. You do not have to manage VLANs, and do not need to be aware of which VLAN was assigned to the network.

**Complete Software-Defined Network Solution**

OpenStack, using Neutron, presents a complete SDN solution. You can define isolated networks with any address space, and connect between those networks using virtual routers. You can define firewall rules without the need to touch or change any element of the physical network topology. Furthermore, there is a complete abstraction between the physical topology and the virtual networks, so that multiple virtual networks can share the same physical resources, without any security or address space concerns.

**User Isolation: Multi-tenancy**

Allowing multiple users to share the same physical environment while ensuring complete separation between them is a key feature of OpenStack. OpenStack is designed so that multiple tenants can share physical resources in a way that is transparent to the users. OpenStack offers ways to share virtual resources between tenants, but maintains complete separation where needed.

**1.2 Introduction to Oracle VM Server**

Oracle VM Server is a component of the Oracle VM product. Oracle VM Server is based on the Xen hypervisor. Oracle VM Server can be managed using Oracle VM Manager, or it can be managed as a standalone product with OpenStack.

Xen is a bare-metal (type 1) hypervisor. The hypervisor is controlled from a privileged environment (which is also itself a virtual machine) called Domain0, or Dom0. Dom0 controls the physical devices on the system, and connects them to the other virtual machines. Dom0 is also responsible for launching virtual machines called domU(s) using tools run from the user space. When launching a virtual machine, Xen connects domU to storage and network resources.

**Figure 1.1 Oracle VM Server, the Xen Hypervisor and Virtual Machines**

Since dom0 is itself a virtual machine, any memory assigned to it cannot be used for domUs. Sufficient dom0 memory is important for performance and correct operation, so it is important to adhere to the minimum memory requirements. See **Section 3.1, “System Requirements”** for information on the memory requirements for using Oracle VM Server as a compute node.
The libvirt driver is used to connect Oracle VM Server to OpenStack. The libvirt driver fully supports Xen. On an Oracle Linux based compute node, the libvirt driver runs in a Docker container. On an Oracle VM Server based compute node, the native libvirt driver is used (libvirt is not run in a container). All other OpenStack compute node services are run in Docker containers on dom0.

The version of Oracle VM Server supported is Oracle VM Server Release 3.4, which uses Oracle’s Unbreakable Enterprise Kernel Release 4 (also known as UEK R4), an upstream Linux kernel hardened and shipped by Oracle. As a result, Oracle VM Server uses the latest versions of hardware drivers and Open vSwitch.
Chapter 2 Getting Started

Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Getting Started with Deployment</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Using Groups to Deploy Services</td>
<td>9</td>
</tr>
<tr>
<td>2.3 Using the kollacli Command</td>
<td>10</td>
</tr>
</tbody>
</table>

This chapter introduces you to the key concepts to get started with deploying Oracle OpenStack for Oracle Linux.

2.1 Getting Started with Deployment

The following is an overview of steps needed to deploy OpenStack services using Oracle OpenStack for Oracle Linux.

1. **Set up the basic OpenStack environment.**
   - Set up the *target nodes*.
     
     Target nodes are the hosts that run OpenStack services. The services a target node runs determines the type of node it is, either controller, compute, database, network, and storage. You install and prepare Oracle Linux or Oracle VM Server on each target node.
   - Set up the *master node*.
     
     The master node is used to deploy OpenStack services to target nodes. Typically, you use a controller node as a master node. But you can use a separate host if you prefer. You install the OpenStack Kolla command line interface (*kollacli*), and set up the *kolla* user. The *kolla* user is the Ansible SSH user that is used to deploy OpenStack services. You also add users to the *kolla* group. Membership of this group enables users to set up, configure and deploy OpenStack services using the *kollacli* command.
   - Set up a *Docker registry*.
     
     OpenStack services are deployed to target nodes as Docker containers. The Docker registry stores the Oracle OpenStack for Oracle Linux Docker images. The Docker daemon running on each target node downloads the Docker images from the registry and uses them to create the Docker containers. The master node tells each target node which images they should download. You configure a Docker registry server and then load the Oracle OpenStack for Oracle Linux Docker images.

   Chapter 3, *Preparing the Basic Environment* has details of what you need to do.

2. **Set up the deployment.**
   - Add the target nodes to the deployment.
     
     You add nodes to the Ansible inventory to identify the target nodes for deployment.
   - Set up the *kolla* user on the target nodes.
     
     You copy the SSH public key for the *kolla* user to the target nodes, and confirm that the *kolla* user can log in to each node. This ensures the *kolla* user is able to deploy OpenStack services to the target nodes.
• Assign the target nodes to groups.

Groups are used to associate the target nodes with OpenStack services. The default groups are control, compute, database, network, and storage. You assign the target nodes to groups so that all nodes in the same group run the same services. For more information, see Section 2.2, “Using Groups to Deploy Services”.

• Enable the OpenStack services you want to deploy.

All OpenStack services are enabled by default, apart from the Swift object storage and the Ceilometer telemetry services. However, you have full control over the services that are deployed and can enable or disable services by setting Kolla properties. For example, the enable_horizon property controls whether or not you want to deploy the OpenStack web-based interface.

• Configure the passwords for all services.

Typically, when you deploy an OpenStack service, a database is created for the service and the credentials for the service admin user are registered with the Keystone service. You must set passwords for the database and the service admin users. You only do this once.

• Configure the deployment by setting properties.

You set Kolla properties to configure how the OpenStack services operate once they are deployed. For example, the cinder_backup_driver property enables you to specify the storage driver used to back up Cinder block storage volumes.

• Prepare to deploy OpenStack services.

Once you have prepared the basic environment, you might need to perform additional configuration, depending on the OpenStack services you intend to deploy.

For example, by default virtual machine instances use temporary storage that only exists for the lifetime of the instance. Once the instance is deleted so is the storage. If you want to set up persistent storage for virtual machines, you can configure the block storage devices for use with the Cinder block storage service, or you can configure the rings that provide the location and organization of your storage for the Swift object storage service.

Chapter 4, Setting up a Deployment has details of what you need to do.

3. Perform the deployment.

• Initial deployment

Initially, you want to deploy all the enabled OpenStack services to all hosts. When you deploy, the configuration files required for each service are assembled using a template and the configured Kolla properties. The kolla user logs in to each target node, copies the service configuration files and starts the Docker containers for the OpenStack services that the target node is configured to run.

• Update or expand the deployment

After your initial deployment, you might want to make configuration changes, to add or remove services, to move services to different target nodes, or to add or remove target nodes. You can do all this by redeploying services to specific groups or hosts, or by redeploying selected services. You are able to do this because the OpenStack data is stored separately from the Docker containers that run the OpenStack services.
2.2 Using Groups to Deploy Services

Oracle OpenStack for Oracle Linux uses groups to associate the target nodes with OpenStack services. Target nodes in the same group run the same OpenStack services. The default groups are:

- **control**: Contains the control-related services, such as glance, keystone, ndbcluster, nova, rabbitmq.
- **compute**: Contains the compute services, such as nova-compute.
- **database**: Contains the database-related services.
- **network**: Contains the network-related services, such as haproxy, neutron.
- **storage**: Contains the storage services, such as cinder and swift.

A node can belong to more than one group and can run multiple OpenStack services.

The minimum supported deployment of OpenStack contains at least three nodes (as shown in Figure 2.1):

- Two controller nodes, each node belongs to the control, database, network and storage groups.
- One or more nodes belonging to the compute group.

**Note**

Single-node deployments (sometimes referred to as *all-in-one* deployments) are **not** supported.

**Figure 2.1 Minimum Supported Deployment**

As your scaling and performance requirements change, you can increase the number of nodes and move groups on to separate nodes to spread the workload, as shown in Figure 2.2:

**Figure 2.2 Expanding Deployment**

As your deployment expands, note the following "rules" for deployment:
Using the kollacli Command

- The nodes in the compute group must not be assigned to the control group.
- The control group must contain at least two nodes.
- The number of nodes in the database group must always be a multiple of two.
- Each group must contain at least two nodes to enable high availability.

There is no limit on the number of nodes in a deployment. Figure 2.3 shows a fully-expanded deployment using the default groups. There are at least two nodes for each group to maintain high availability, and the number of database nodes is a multiple of two.

Figure 2.3 Fully-Expanded Deployment

You are not restricted to using the default groups. You can change the services a group runs, or configure your own groups. If you configure your own groups, be sure to remember the rules of deployment listed above.

2.3 Using the kollacli Command

You run the kollacli command on the master node and use it to configure and deploy OpenStack services to the target nodes. To run kollacli commands, you must be a member of the kolla group.

The kollacli command has a set of subcommands, which are organized by the objects that they manage.

To configure the layout of your OpenStack deployment, you perform actions on groups, hosts and services, as follows:

- The kollacli group commands manage the associations between target nodes and the OpenStack services they run. Target nodes in the same group run the same services.
  
  Example command: kollacli group addhost adds a host to a group.

- The kollacli host commands manage the target nodes in a deployment.
  
  Example command: kollacli host add adds a host to the list of target nodes.

- The kollacli service commands manage the OpenStack services.
  
  Example command: kollacli service addgroup adds a service to a group.

To configure your OpenStack deployment, you configure values for passwords and properties, as follows:

- The kollacli password commands manage the passwords for the OpenStack components.
  
  Example command: kollacli password set sets a value for a password.
Bash Command Completion

- The `kollacli property` commands manage the configuration settings for OpenStack services.
  
  Example command: `kollacli property set` sets a value for a configuration property.

Once you have configured your deployment, you deploy OpenStack services with the `kollacli deploy` command.

Help on how to use the `kollacli` command is available, as follows:

- To list all `kollacli` commands, use the `kollacli help` command.

- To list the related commands for an object, use the `kollacli help object` for example `kollacli help host`.

- To get help for a specific command, use the `kollacli help subcommand` command, where `subcommand` is the name of the command, for example `kollacli host list` or `kollacli service listgroups`.

For a complete syntax reference for `kollacli` commands, see Appendix A, `kollacli Command-Line Reference`.

Bash Command Completion

You can enable Bash command completion for the `kollacli` command, as follows:

1. Install the `bash-completion` package, if it is not already installed:

   ```bash
   # yum install bash-completion
   ```

2. Use the `kollacli complete` command to generate the command completion function.

   To display the function so you can copy and paste it into a file:

   ```bash
   $ kollacli complete
   ```

   To output the function to a file:

   ```bash
   $ kollacli complete > /etc/bash_completion.d/kollacli
   ```

   You need `root` privileges to write to the `/etc/bash_completion.d` directory.

3. Source the file to enable command completion:

   ```bash
   $ source /etc/bash_completion.d/kollacli
   ```

Using the kollacli Shell

The `kollacli` shell enables you to enter several commands without having to type the `kollacli` command each time. You start the shell with the `kollacli` command. When you are in the `kollacli` shell, the prompt changes to `(kollacli)`. From the shell prompt you can enter `kollacli` commands in their short form, for example:

```bash
$ kollacli
(kollacli) host list
(kollacli) group listhosts
```

In addition to the `help` command, the `kollacli` shell also supports the `-h` and `--help` options for obtaining help with `kollacli` commands.
To exit the kollacli shell and return to the operating system prompt, type exit, quit, or q.

Formatting Command Output

When you use kollacli commands, such as the kollacli property list command, to show what you have configured, these commands have a --format option which enable you to format the output to suit your needs, as shown in the following table.

<table>
<thead>
<tr>
<th>Option</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>csv</td>
<td>Comma-separated values.</td>
</tr>
<tr>
<td></td>
<td>Use the --quote option with this format to control the use of quotes in the output:</td>
</tr>
<tr>
<td></td>
<td>• all: Quote all values.</td>
</tr>
<tr>
<td></td>
<td>• minimal: Optimized minimal quoting of values.</td>
</tr>
<tr>
<td></td>
<td>• none: No quoting of any values.</td>
</tr>
<tr>
<td></td>
<td>• nonnumeric: Quote only non-numeric values.</td>
</tr>
<tr>
<td></td>
<td>The default is nonnumeric.</td>
</tr>
<tr>
<td>html</td>
<td>HTML table markup.</td>
</tr>
<tr>
<td>json</td>
<td>JavaScript Object Notation.</td>
</tr>
<tr>
<td>table</td>
<td>Simple ASCII display table.</td>
</tr>
<tr>
<td></td>
<td>This is the default output format.</td>
</tr>
<tr>
<td></td>
<td>Use the --max-width option with this format to set the maximum display width for each column. The value for this option must be an integer. The default is 0 (no maximum width).</td>
</tr>
<tr>
<td>value</td>
<td>Space separated values with no headers.</td>
</tr>
<tr>
<td></td>
<td>This format may be useful to pipe output to an operating system command.</td>
</tr>
<tr>
<td>yaml</td>
<td>YAML format.</td>
</tr>
</tbody>
</table>

You can also use the --column option to select the columns that are included in the output. Use multiple --column options to select each column you want. The names of the columns change depending on the kollacli command used and the names are case sensitive. By default, all columns are included.
Chapter 3 Preparing the Basic Environment

Table of Contents

3.1 System Requirements ................................................................. 13
3.2 OpenStack Data Storage ............................................................. 15
3.3 Host Network Requirements ....................................................... 16
3.4 Preparing Oracle Linux Nodes .................................................... 18
3.5 Preparing Oracle VM Server Compute Nodes .................................. 26
3.6 Preparing a Master Node ............................................................. 32
3.7 Setting up the Docker Registry ................................................... 33
  3.7.1 Setting up the Docker Registry Server ...................................... 33
  3.7.2 Importing Images into the Docker Registry ................................. 37

The systems in an OpenStack deployment are known as nodes. The nodes in a deployment are either Oracle Linux nodes or Oracle VM Server compute nodes. To prepare nodes for deployment, you set up the system and configure the Docker Engine, as described in Section 3.4, “Preparing Oracle Linux Nodes” and Section 3.5, “Preparing Oracle VM Server Compute Nodes”.

You deploy OpenStack services from a master node using the kollacli command. Section 3.6, “Preparing a Master Node” describes how to set up a master node.

OpenStack services are deployed to each node as Docker containers. The containers are created from Docker images, which are stored in a Docker registry. Section 3.7, “Setting up the Docker Registry” describes how to set up a Docker registry server and to import the Oracle OpenStack for Oracle Linux images.

3.1 System Requirements

Oracle OpenStack for Oracle Linux is supported on Oracle Linux (for all node types) and Oracle VM Server (as compute nodes only). Information on the supported hardware is available in the Hardware Certification List for Oracle Linux and Oracle VM at:

http://linux.oracle.com/hardware-certifications

The storage hardware you use should be included in the hardware list. Oracle is working with its partners to make sure customers have a choice of storage. For specific storage plug-ins, contact Oracle or the plug-in vendor.

You can download Oracle Linux and Oracle VM Server from the Oracle Software Delivery Cloud at:

https://edelivery.oracle.com/linux

The following table lists the minimum system requirements for each OpenStack node type:

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Minimum System Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Controller</td>
<td>• 1 CPU</td>
</tr>
<tr>
<td></td>
<td>• 8 GB RAM</td>
</tr>
<tr>
<td></td>
<td>• 2 NICs</td>
</tr>
<tr>
<td></td>
<td>• Oracle Linux Release 7 Update 2 and later</td>
</tr>
</tbody>
</table>
## Node Type

### Minimum System Requirements

- **Compute** (Oracle Linux)
  - 1 CPU
  - 16 GB RAM
  - 2 NICs (4 NICs recommended for best performance)
  - Oracle Linux Release 7 Update 2 and later
  - Unbreakable Enterprise Kernel Release 4
  - 64 GB btrfs file system mounted on `/var/lib/docker`

- **Compute** (Oracle VM Server)
  - 1 CPU
  - 16 GB RAM
  - 2 NICs (4 NICs recommended for best performance)
  - Oracle VM Server Release 3.4, on 64-bit x86 platforms (x86_64)
  - Unbreakable Enterprise Kernel Release 4
  - 64 GB btrfs file system mounted on `/var/lib/docker`

- **Database**
  - 1 CPU
  - 8 GB RAM
  - 2 NICs
  - Oracle Linux Release 7 Update 2 and later
  - Unbreakable Enterprise Kernel Release 4
  - 64 GB btrfs file system mounted on `/var/lib/docker`

- **Network**
  - 1 CPU
  - 8 GB RAM
  - 3 NICs (4 NICs recommended for best performance)
  - Oracle Linux Release 7 Update 2 and later
  - Unbreakable Enterprise Kernel Release 4
  - 64 GB btrfs file system mounted on `/var/lib/docker`

- **Storage**
  - 1 CPU
  - 8 GB RAM
OpenStack Data Storage

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Minimum System Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• 2 NICs (3 NICs recommended for best performance)</td>
</tr>
<tr>
<td></td>
<td>• Oracle Linux Release 7 Update 2 and later</td>
</tr>
<tr>
<td></td>
<td>• Unbreakable Enterprise Kernel Release 4</td>
</tr>
<tr>
<td></td>
<td>• 64 GB btrfs file system mounted on /var/lib/docker</td>
</tr>
</tbody>
</table>

In addition to the OpenStack nodes, Oracle OpenStack for Oracle Linux requires a node to host a Docker registry and a node (known as a master node) from which you deploy OpenStack services using the kollacli command. Typically these are hosted on a controller node, but you can host these on separate nodes if you prefer. The following are the minimum requirements for separate nodes:

Table 3.2 Additional Node Minimum System Requirements

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Minimum System Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master</td>
<td>• 1 CPU</td>
</tr>
<tr>
<td></td>
<td>• 2 GB RAM</td>
</tr>
<tr>
<td></td>
<td>• 1 NIC</td>
</tr>
<tr>
<td></td>
<td>• Oracle Linux Release 7 Update 2 and later</td>
</tr>
<tr>
<td></td>
<td>• Unbreakable Enterprise Kernel Release 4</td>
</tr>
<tr>
<td></td>
<td>• 64 GB btrfs file system mounted on /var/lib/docker</td>
</tr>
<tr>
<td>Registry</td>
<td>• 1 CPU</td>
</tr>
<tr>
<td></td>
<td>• 2 GB RAM</td>
</tr>
<tr>
<td></td>
<td>• 1 NIC</td>
</tr>
<tr>
<td></td>
<td>• Oracle Linux Release 7 Update 2 and later</td>
</tr>
<tr>
<td></td>
<td>• Unbreakable Enterprise Kernel Release 4</td>
</tr>
<tr>
<td></td>
<td>• 15 GB btrfs file system mounted on /var/lib/registry</td>
</tr>
</tbody>
</table>

3.2 OpenStack Data Storage

Many OpenStack services store their data in the /var/lib/kolla directory on the target nodes. If a service needs to store data, the /var/lib/kolla directory is created automatically when the service is deployed, and one or more subdirectories are created in /var/lib/kolla. The subdirectories are used as data volumes for the service’s Docker containers. For example when you deploy the Nova service, the /var/lib/kolla/var/lib/nova/instances directory is created on compute nodes and this is used for ephemeral storage for instances (virtual machines).

The following services currently store data outside the /var/lib/kolla directory:

• Glance: the /var/lib/glance directory on controller nodes.
• Nova: the /etc/iscsi and /etc/multipath directories on compute nodes.
You must ensure that your target nodes have sufficient disk space to store OpenStack data, especially if you use ephemeral storage for instances.

You should create regular backups of your OpenStack data.

When you remove OpenStack services from nodes, the data is not removed unless you explicitly request to remove it.

### 3.3 Host Network Requirements

In an OpenStack environment, the network traffic can be divided into the following networks:

**Administration network**

This network is not part of the OpenStack infrastructure. It provides Internet access for all nodes and is used for administration to install software packages and security updates from Oracle Unbreakable Linux Network or Oracle Yum Server, and to provide access to the Docker registry and other services such as NTP and DNS.

**Public network**

This network provides the external access to OpenStack services. For instances (virtual machines), this network provides the route out to the external network and the IP addresses to enable inbound connections to the instances. This network can also provide the public API endpoints to connect to OpenStack services.

**Management/API network**

This network is a private internal network used for communication between all the OpenStack services and for high availability. All nodes in the deployment must have an IPv4 address on this network.

**Virtual machine network**

This network is a private internal network that is used for communication between the OpenStack instances (virtual machines), and also between the instances and the network nodes to provide inbound and outbound connections to the public network.

**Storage network**

This network is a private internal that is used for the communication between compute nodes and storage nodes to access the storage.

Once you have decided how you want to implement these networks, you configure the network interfaces for the networks and other network settings by setting properties with the `kollacli property set` command, as described in Section 4.2, "Setting up Services".

Figure 3.1 shows the default network configuration and the properties that specify the network interfaces on the target nodes.
In the default configuration, the `neutron_external_interface` property specifies the name of the network interface on the public network. To make it easier to get started with deploying OpenStack services, the management/API, virtual machine, and storage networks are consolidated into a single network. The `network_interface` property specifies the name of the network interface on this network. This configuration requires the minimum number of network interfaces.

While consolidating the management/API, virtual machine, and storage networks into a single network makes it easier to deploy OpenStack services, for performance and security reasons it is best to separate and isolate these networks. Figure 3.2 shows a fully-expanded network configuration and the properties that specify the network interfaces on the target nodes.

With this configuration, the `neutron_external_interface` property specifies the name of the network interface on the public network, and there are separate properties for specifying the names of network interfaces on the management/API (`api_interface`), virtual machine (`tunnel_interface`), and storage networks (`storage_interface`).
Preparing Oracle Linux Nodes

storage \(\text{(storage\_interface)}\) networks. This configuration requires the maximum number of network interfaces.

If you are starting with a small OpenStack deployment, the network configuration is flexible so you do not have to separate the management/API, virtual machine, and storage networks in one go. By default, the properties for these networks are mapped to the network\_interface property. So you could start with the default consolidated network, and then separate the networks as your deployment expands.

You specify the network interface names on the target nodes using a single property for each network. This works best where the hardware on the target nodes is identical. When this is not the case, you can handle the variations by setting properties that are specific to groups or hosts, see Section 4.4, “Setting Properties for Groups or Hosts”.

3.4 Preparing Oracle Linux Nodes

This section describes how to prepare an Oracle Linux node for OpenStack. Section 3.5, “Preparing Oracle VM Server Compute Nodes” describes how to prepare an Oracle VM Server node.

You can download the installation ISO for the latest version of Oracle Linux Release 7 from the Oracle Software Delivery Cloud at:

https://edelivery.oracle.com/linux

You prepare an Oracle Linux node for OpenStack by enabling the required repositories and installing the Oracle OpenStack for Oracle Linux preinstallation package. When you install the preinstallation package, it installs all the other required packages on the node. The packages can be installed using either the Oracle Unbreakable Linux Network (ULN) or the Oracle Linux Yum Server. If you are using ULN, the following procedure assumes that you register the system with ULN during installation.

For more information about ULN and registering systems, see:

http://docs.oracle.com/cd/E52668_01/E39381/html/index.html

For more information on using the Oracle Linux Yum Server, see

http://docs.oracle.com/cd/E52668_01/E54669/html/ol7-yum.html

Oracle OpenStack for Oracle Linux Release 2.1 uses a version of Docker, which requires that you configure a system to use the Unbreakable Enterprise Kernel Release 4 (UEK R4) and boot the system with this kernel.

Oracle OpenStack for Oracle Linux requires a btrfs file system mounted on \(\text{/var/lib/docker}\) with at least 64GB available. The following provides instructions for setting up a btrfs file system using one or more available devices. The device could be a disk partition, an LVM volume, a loopback device, a multipath device, or a LUN.

To prepare an Oracle Linux node:

1. Install Oracle Linux using the instructions in the Oracle Linux Installation Guide for Release 7 at:


   Select **Minimal install** as the base environment for all node types.

   As part of the install, you should create a btrfs file system mounted at \(\text{/var/lib/docker}\). This file system requires a minimum of 64GB of disk space and is used to host a local copy of the OpenStack Docker images. If you prefer, you can create the file system after installation, as described in the following steps.
Preparation of Oracle Linux Nodes

If the node will also host the Docker registry, you should create an additional btrfs file system mounted at `/var/lib/registry`. This file system requires a minimum of 15GB of disk space. If you prefer, you can create the file system after installation, as described in Section 3.7, “Setting up the Docker Registry”.

2. Disable SELinux.

In order to use the btrfs storage engine with Docker, you must either disable SELinux or set the SELinux mode to Permissive.

To check the current SELinux mode, use the `getenforce` command. If the output of this command shows Enabled, you must disable SELinux as follows:

a. Edit `/etc/selinux/config` and set the value of the SELINUX directive to disabled.

```
# systemctl stop firewalld
# systemctl disable firewalld
```

3. Stop and disable the `firewalld` service.

If you require a system firewall, you can use `iptables` instead of `firewalld`.

```
# systemctl status firewalld
# systemctl is-enabled firewalld
```

4. Create a btrfs file system mounted on `/var/lib/docker`.

You create a btrfs file system with the utilities available in the `btrfs-progs` package, which should be installed by default.

```
# mkfs.btrfs [-L label] block_device ...
```

where `-L label` is an optional label that can be used to mount the file system.

For example:

- To create a file system in a partition `/dev/sdb1`:

  ```
  # mkfs.btrfs -L var-lib-docker /dev/sdb1
  ```

  The partition must already exist. Use a utility such as `fdisk` (MBR partitions) or `gdisk` (GPT partitions) to create one if needed.

- To create a file system across two disk devices, `/dev/sdc` and `/dev/sdd`:

  ```
  # mkfs.btrfs -L var-lib-docker /dev/sd[cd]
  ```
Preparing Oracle Linux Nodes

The default configuration is to stripe the file system data (raid0) and to mirror the file system metadata (raid1) across the devices. Use the -d (data) and -m (metadata) options to specify the required RAID configuration. For raid10, you must specify an even number of devices and there must be at least four devices.

- To create a file system in a logical volume named docker in the ol volume group:

```bash
# mkfs.btrfs -L var-lib-docker /dev/ol/docker
```

The logical volume must already exist. Use Logical Volume Manager (LVM) to create one if needed.

More information on using mkfs.btrfs is available in the Oracle Linux Administrator's Guide for Release 7 at:

http://docs.oracle.com/cd/E52668_01/E54669/html/ol7-create-btrfs.html

b. Obtain the UUID of the device containing the btrfs file system.

Use the blkid command to display the UUID of the device and make a note of this value, for example:

```bash
# blkid /dev/sdb1
/dev/sdb1: LABEL="var-lib-docker" UUID="460ed4d2-255f-4c1b-bb2a-588783ad72b1" \\
UUID_SUB="3b4562d6-b248-4c89-96c5-53d38b9b8b77" TYPE="btrfs"
```

If the btrfs file system is created across multiple devices, you can specify any of the devices to obtain the UUID. Alternatively you can use the btrfs filesystem show command to see the UUID. For a logical volume, specify the path to the logical volume as the device for example /dev/ol/docker. Ignore any UUID_SUB value displayed.

c. Edit the /etc/fstab file and add an entry to ensure the file system is mounted when the system boots.

```bash
UUID=UUID_value /var/lib/docker  btrfs  defaults  1 2
```

Replace UUID_value with the UUID that you found in the previous step. If you created a label for the btrfs file system, you can also use the label instead of the UUID, for example:

```bash
LABEL=label /var/lib/docker  btrfs  defaults  1 2
```

d. Create the /var/lib/docker directory.

```bash
# mkdir /var/lib/docker
```

e. Mount all the file systems listed in /etc/fstab.

```bash
# mount -a
```

f. Verify that the file system is mounted.

```bash
# df
Filesystem  1K-blocks  Used  Available  Use% Mounted on
... /dev/sdb1   ...  ...  ...  1%  /var/lib/docker
```

5. (Optional) If you use a proxy server for Internet access, configure Yum with the proxy server settings.

Edit the /etc/yum.conf file and specify the proxy setting, for example:
Preparing Oracle Linux Nodes

proxy=http://proxysvr.example.com:3128

If the proxy server requires authentication, additionally specify the `proxy_username` and `proxy_password` settings, for example:

proxy=http://proxysvr.example.com:3128
    proxy_username=USERNAME
    proxy_password=PASSWORD

If you use the yum plug-in (`yum-rhn-plugin`) to access the ULN, specify the `enableProxy` and `httpProxy` settings in the `/etc/sysconfig/rhn/up2date` file, for example:

enableProxy=1
    httpProxy=http://proxysvr.example.com:3128

If the proxy server requires authentication, additionally specify the `enableProxyAuth`, `proxyUser`, and `proxyPassword` settings, as follows:

enableProxy=1
    httpProxy=http://proxysvr.example.com:3128
    enableProxyAuth=1
    proxyUser=USERNAME
    proxyPassword=PASSWORD

6. Make sure the system is up-to-date:

   # yum update

7. Enable the required ULN channels or Yum repositories.

   **To enable the required ULN channels:**

   a. Log in to [http://linux.oracle.com](http://linux.oracle.com) with your ULN user name and password.

   b. On the **Systems** tab, click the link named for the system in the list of registered machines.

   c. On the System Details page, click **Manage Subscriptions**.

   d. On the System Summary page, use the left and right arrows to move channels to and from the list of subscribed channels.

   Subscribe the system to the following channels:

   - `ol7_x86_64_UEK4` - Unbreakable Enterprise Kernel Release 4 for Oracle Linux 7 (x86_64)
   - `ol7_x86_64_addons` - Oracle Linux 7 Addons (x86_64)
   - `ol7_x86_64_openstack21` - Oracle OpenStack 2.1 (x86_64)
   - `ol7_x86_64_latest` - Oracle Linux 7 Latest (x86_64)
   - *(Optional)* `ol7_x86_64_UEK4_OFED` - OFED supporting tool packages for Unbreakable Enterprise Kernel Release 4 on Oracle Linux 7 (x86_64)

   Subscribe to this channel if you are using the OFED (OpenFabrics Enterprise Distribution) packages provided by Oracle. UEK R4 requires a different set of OFED packages to UEK R3.

   **Unsubscribe** the system from the following channels:
Preparing Oracle Linux Nodes

- **ol7_x86_64_UEKR3** - Unbreakable Enterprise Kernel Release 3 for Oracle Linux 7 (x86_64) - Latest
- **ol7_x86_64_UEKR3_OFED20** - OFED supporting tool packages for Unbreakable Enterprise Kernel Release 3 on Oracle Linux 7 (x86_64)

e. Click **Save Subscriptions**.

To enable the required Yum repositories:

a. Check that you have the latest Oracle Linux Yum Server repository file.

Check that the `/etc/yum.repos.d/public-yum-ol7.repo` file contains an `[ol7_UEKR4]` section. If it does not, you do not have the most up-to-date version of the repository file.

To download the latest copy of the repository file:

```
```


Enable the following repositories by setting `enabled=1` in the following sections:

- `[ol7_UEKR4]`
- `[ol7_addons]`
- `[ol7_openstack21]`
- `[ol7_latest]`
- **(Optional)** `[ol7_UEKR4_OFED]`

Subscribe to this repository only if you have InfiniBand-capable devices and you are using the OFED (OpenFabrics Enterprise Distribution) packages provided by Oracle. UEK R4 requires a different set of OFED packages to UEK R3.

Disable the following repositories by setting `enabled=0` in the following sections:

- `[ol7_UEKR3]`
- `[ol7_UEKR3_OFED20]`

8. Use the `yum` command to check the repository configuration.

Clean all yum cached files from all enabled repositories.

```
# yum clean all
```

List the configured repositories for the system.

```
# yum repolist
```

9. **(Optional)** Remove the Open vSwitch kernel module package.

To check if the `kmod-openvswitch-uek` package is installed:

```
# yum list installed kmod-openvswitch-uek
```
Preparing Oracle Linux Nodes

If the `kmod-openvswitch-uek` package is installed, remove it:

```
# yum -y remove kmod-openvswitch-uek
```

You must remove the UEK R3 Open vSwitch kernel module package in order to resolve the package dependencies for UEK R4. UEK R4 includes the Open vSwitch kernel module.

10. **(Optional)** Remove any existing OFED packages.

Only perform this step if you have InfiniBand-capable devices and you are using the OFED packages provided by Oracle. UEK R4 requires a different set of OFED packages to UEK R3.

For instructions on how to remove the OFED packages, see the release notes for your UEK R4 release, available at [http://docs.oracle.com/cd/E52668_01/index.html](http://docs.oracle.com/cd/E52668_01/index.html).

11. Install the Oracle OpenStack for Oracle Linux preinstallation package.

If you are preparing an Oracle Linux node for a new OpenStack deployment:

```
# yum install openstack-kolla-preinstall
```

If you are updating an Oracle Linux node for a new release of Oracle OpenStack for Oracle Linux:

```
# yum update openstack-kolla-preinstall
```

This ensures the system has the required packages for OpenStack Kolla deployments.

12. **(Optional)** Install the OFED packages for UEK R4 and enable the RDMA service.

Only perform this step if you have InfiniBand-capable devices and you are using the OFED packages provided by Oracle.

For instructions on how to install the OFED packages and enable the RDMA service, see the release notes for your UEK R4 release, available at [http://docs.oracle.com/cd/E52668_01/index.html](http://docs.oracle.com/cd/E52668_01/index.html).

13. Reboot the system.

```
# systemctl reboot
```

14. Check the system has booted with the UEK R4 kernel.

```
# uname -r
4.1.12-32.2.1.el7uek.x86_64
```

If the output of this command begins with `4.1.12`, the system has booted with the UEK R4 kernel.

If the system has not booted with the UEK R4 kernel, you must edit your grub configuration to boot with this kernel and reboot, as follows:

a. Display the menu entries that are defined in the GRUB 2 configuration file.

On UEFI-based systems, the configuration file is `/boot/efi/EFI/redhat/grub.cfg`.  
On BIOS-based systems, the configuration file is `/boot/grub2/grub.cfg`.

```
# grep '^menuentry' /boot/grub2/grub.cfg
```

...  
```
menuentry 'Oracle Linux Server 7.2, with Unbreakable Enterprise Kernel 4.1.12-32.2.1.e ... {'
menuentry 'Oracle Linux Server (3.8.13-98.7.1.el7uek.x86_64 with Unbreakable Enterpris ... {'
...
Preparing Oracle Linux Nodes

In this example, the configuration file is for a BIOS-based system. GRUB 2 counts the menu entries in the configuration file starting at 0 for the first entry. In this example, menu entry 0 is for a UEK R4 kernel (4.1.12), and menu entry 1 is for a UEK R3 kernel (3.8.13).

b. Make the UEK R4 the default boot kernel.

In the following example, menu entry 0 is set as the default boot kernel for a BIOS-based system.

```bash
# grub2-set-default 0
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

In the following example, menu entry 0 is set as the default boot kernel for a UEFI-based system.

```bash
# grub2-set-default 0
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

c. Reboot the system and confirm that UEK R4 is the boot kernel.

15. If you are using a web proxy, configure the docker service to use the proxy.

a. Create the drop-in file `/etc/systemd/system/docker.service.d/http-proxy.conf` with the following content:

```bash
[Service]
Environment="HTTP_PROXY=proxy_URL:port"
Environment="HTTPS_PROXY=proxy_URL:port"
```

Replace `proxy_URL` and `port` with the appropriate URLs and port numbers for your web proxy.

If the host also runs the Docker registry that stores the Oracle OpenStack for Oracle Linux images, you should specify that local connections do not need to be proxied by setting the `NO_PROXY` environment variable:

```bash
[Service]
Environment="HTTP_PROXY=proxy_URL:port" "NO_PROXY=localhost,127.0.0.1"
Environment="HTTPS_PROXY=proxy_URL:port" "NO_PROXY=localhost,127.0.0.1"
```

b. Reload systemd manager configuration.

```bash
# systemctl daemon-reload
```

c. Restart the docker service.

```bash
# systemctl restart docker.service
```

d. Check that the docker service is running.

```bash
# systemctl status docker.service
```

Check the Drop-In: line and ensure that all the required systemd drop-in files are listed.

Check that any environment variables you have configured, such as web proxy settings, are loaded:

```bash
# systemctl show docker --property Environment
```
Environment=HTTP_PROXY=http://proxy.example.com:80

If you have installed the `mlocate` package, it is recommended that you add `/var/lib/docker` to the `PRUNEPATHS` entry in `/etc/updatedb.conf` to prevent `updatedb` from indexing directories below `/var/lib/docker`.

16. Synchronize the time.

Time synchronization is essential to avoid errors with OpenStack operations. Before deploying OpenStack, you should ensure that the time is synchronized on all nodes using the Network Time Protocol (NTP).

It is best to configure the controller nodes to synchronize the time from more accurate (lower stratum) NTP servers and to configure the other nodes to synchronize the time from the controller nodes.

Further information on network time configuration can be found in the *Oracle Linux Administration Guide for Release 7* at:

http://docs.oracle.com/cd/E52668_01/E54669/html/ol7-nettime.html

The following configuration assumes that the firewall rules for your internal networks enable you to access public or local NTP servers. Perform the following steps on all Oracle Linux nodes.

Time synchronization for Oracle VM Server compute nodes is described in *Section 3.5, “Preparing Oracle VM Server Compute Nodes”.*

a. Install the `chrony` package.

```
# yum install chrony
```

b. Edit the `/etc/chrony.conf` file to configure the `chronyd` service.

On the *controller nodes*, configure the `chrony` service to synchronize time from a pool of NTP servers and set the `allow` directive to enable the controller nodes to act as NTP servers for the other OpenStack nodes, for example:

```
server NTP_server_1
server NTP_server_2
server NTP_server_3
allow 10.0.0/24
```

The NTP servers can be public NTP servers or your organization may have its own local NTP servers. In the above example, the `allow` directive specifies a subnet from which the controller nodes accept NTP requests. Alternatively, you can specify the other OpenStack nodes individually with multiple `allow` directives.

On all other nodes, configure the `chrony` service to synchronize time from the controller nodes, for example:

```
server control1.example.com iburst
server control2.example.com iburst
```

c. Start the `chrony` service and configure it to start following a system reboot.

```
# systemctl start chrony
# systemctl enable chrony
```

d. Verify that `chrony` is accessing the correct time sources.
On the **controller nodes**, the Name/IP address column in the command output should list the configured pool of NTP servers. On all other nodes, it should list the controller nodes.

e. Ensure that the time is synchronized on all nodes.

Use the `chronyc -a tracking` command to check the offset (the Last offset row):

```
# chronyc -a tracking
200 OK
Reference ID    : 10.0.0.11 (control1.example.com)
Stratum         : 3
Ref time (UTC)  : Fri Mar 4 16:19:50 2016
System time     : 0.000000007 seconds slow of NTP time
Last offset     : -0.000088924 seconds
RMS offset      : 2.834978580 seconds
Frequency       : 3.692 ppm slow
Residual freq   : -0.006 ppm
Skew            : 0.222 ppm
Root delay      : 0.047369 seconds
Root dispersion : 0.004273 seconds
Update interval : 2.1 seconds
Leap status     : Normal
```

To force a node to synchronize its time:

```
# chronyc -a 'burst 4/4'
200 OK
200 OK
```

### 3.5 Preparing Oracle VM Server Compute Nodes

This section describes how to prepare an Oracle VM Server compute node for OpenStack. **Section 3.4, “Preparing Oracle Linux Nodes”** describes how to prepare an Oracle Linux node.

You can download the installation ISO for Oracle VM Server Release 3.4, from the Oracle Software Delivery Cloud at:
https://edelivery.oracle.com/linux

**To prepare an Oracle VM Server compute node:**

1. Install Oracle VM Server for x86 using the instructions in the **Oracle VM Installation and Upgrade Guide for Release 3.4** at:

2. When the operating system is installed, select **Alt+F2** to enter the login screen. Log in as the **root** user.

3. Increase the dom0 maximum memory allocation to 16GB.
   Any RAM not in use by dom0 is available for use by domUs (virtual machines).
Preparing Oracle VM Server Compute Nodes

a. Edit the `/etc/default/grub` file on the Oracle VM Server and change the `dom0_mem` parameter to **16384M**:

```bash
GRUB_CMDLINE_XEN="dom0_mem=max:16384M allowsuperpage"
```

b. Regenerate the `grub.cfg` file.

For example, in a BIOS installation, enter:

```bash
# grub2-mkconfig -o /boot/grub2/grub.cfg
```

In a UEFI installation, enter:

```bash
# grub2-mkconfig -o /boot/efi/EFI/redhat/grub.cfg
```

c. Restart the Oracle VM Server for the changes to take effect:

```bash
# reboot
```

4. Disable and stop the Oracle VM Agent service:

```bash
# chkconfig ovs-agent off
# service ovs-agent stop
```

5. Download the latest Oracle Linux Release 6 Yum Server repository file:

```bash
```

6. Edit the `/etc/yum.repos.d/public-yum-ol6.repo` file. Enable the following repositories by setting `enabled=1` in the following sections:

- [ol6_addons]
- [ol6_UEKR4]
- [ol6_openstack21]

All other repositories should be disabled by setting `enabled=0` in each section.

This repository contains the required packages for Oracle Linux Release 6, upon which Oracle VM Server is based. This repository also contains the preinstallation packages to set up Oracle OpenStack for Oracle Linux on Oracle VM Server. This repository does not contain updates for Oracle VM Server.

7. If you access the Yum repositories using a proxy server, update the `/etc/yum.conf` file with the proxy server address. For more information on using the Oracle Linux Yum Server, see:

   [https://docs.oracle.com/cd/E37670_01/E41138/html/ol_yum.html](https://docs.oracle.com/cd/E37670_01/E41138/html/ol_yum.html)

8. Use the `yum` command to check the repository configuration.

Clean all yum cached files from all enabled repositories.

```bash
# yum clean all
```

List the configured repositories for the system.

```bash
# yum repolist
```

9. Make sure the system is up-to-date:
# yum update

10. The `iptables` service is not running by default. If you enable the `iptables` service, you need to make sure all the required ports are open. You can find a list of the required ports by issuing the following command from the master node (after setting up all the required `kollacli` deployment options).

```
# kollacli property list | grep port
```

For more information on using `iptables`, see the Oracle Linux Administrator's Guide for Release 6:

https://docs.oracle.com/cd/E37670_01/E41138/html/ol_firewall_sec.html

11. Create a partition for the Docker images. This partition must be at least 64 GB in size. The partition can be created in a number of ways, depending on your hardware and storage configuration.

For simplicity, this example uses an Oracle VM Server installation on a single disk, accepting all the partitioning defaults in the Oracle VM Server installer. Your installation and configuration may vary.

**Example 3.1 In a UEFI-based installation:**

a. Start the `parted` utility:

```
# parted
GNU Parted 2.1
Using /dev/sda
Welcome to GNU Parted! Type 'help' to view a list of commands.
```

b. List the existing partitions:

```
(parted) print
...
Number Start   End     Size     File system     Name  Flags
1      ...     ...     ...      fat16                 boot
2      ...     ...     ...      ext4
3      ...     ...     ...      ext4
4      ...     ...     ...      linux-swap(v1)
5      ...     ...     ...
```

c. Remove the last partition:

```
(parted) rm 5
Warning: WARNING: the kernel failed to re-read the partition table on /dev/sda (Device or resource busy). As a result, it may not reflect all of your changes until after reboot.
```

d. Create a new btrfs partition of at least 64 GB for the Docker images, for example:

```
(parted) mkpart btrfs 100GB 164GB
Warning: WARNING: the kernel failed to re-read the partition table on /dev/sda (Device or resource busy). As a result, it may not reflect all of your changes until after reboot.
```

e. Create a partition with the remaining space for the Oracle VM Server local storage, for example:

```
(parted) mkpart extended 164GB 1200GB
Warning: WARNING: the kernel failed to re-read the partition table on /dev/sda (Device or resource busy). As a result, it may not reflect all of your changes until after reboot.
```

f. The results should look something like this:
Preparing Oracle VM Server Compute Nodes

```
(parted) print
...
Number Start  End  Size  File system  Name       Flags
1 .... .... .... fat16  boot
2 .... .... .... ext4
3 .... .... .... ext4
4 .... .... .... linux-swap(v1)
5 100GB 160GB 64GB  btrfs
6 160GB .... .... extended
```

g. Reboot the Oracle VM Server:

```
# reboot
```

**Example 3.2 In a BIOS-based installation:**

a. List the partitions using:

```
# fdisk -l
...
Device Boot Start  End  Blocks  Id  System
/dev/sda1  * .... .... .... 83  Linux
/dev/sda2 .... .... .... 83  Linux
/dev/sda3 .... .... .... 82  Linux swap / Solaris
/dev/sda4 .... .... .... 83  Linux
...
```

As shown in this output, there are four primary partitions. A BIOS-based installation is restricted to four primary partitions. To create another partition for Oracle OpenStack for Oracle Linux, delete the last partition, and create a new extended partition, which can include a new logical partition for the Docker images.

b. Start the `fdisk` utility.

```
# fdisk /dev/sda
WARNING: DOS-compatible mode is deprecated. It's strongly recommended to switch off the mode (command 'c') and change display units to sectors (command 'u').
```

c. Remove the last partition created by the Oracle VM Server installer, this case it is the fourth partition:

```
Command (m for help): d
Partition number (1-4): 4
```

d. Write the changes to the partition table and exit:

```
Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.
WARNING: Re-reading the partition table failed with error 16: Device or resource busy.
The kernel still uses the old table. The new table will be used at the next reboot or after you run partprobe(8) or kpartx(8)
Syncing disks.
```

e. Reboot the Oracle VM Server:

```
# reboot
```
f. Create an extended partition for the Oracle VM Server local storage:

```bash
# fdisk /dev/sda
WARNING: DOS-compatible mode is deprecated. It's strongly recommended to switch off the mode (command 'c') and change display units to sectors (command 'u').

Command (m for help): n
Command action
  e  extended
  p  primary partition (1-4)

Selected partition 4
First cylinder (......., default ...):
Using default value ...
Last cylinder, +cylinders or +size(K,M,G) (......., default ...):
Using default value ...
```

```
g. Create a logical partition for the Docker images, with a size of at least 64 GB:

Command (m for help): n
First cylinder (......., default ...):
Using default value ...
Last cylinder, +cylinders or +size(K,M,G) (......., default ...): +64G
```

h. The results should look something like this:

```
Command (m for help): p
```

```

<table>
<thead>
<tr>
<th>Device</th>
<th>Boot</th>
<th>Start</th>
<th>End</th>
<th>Blocks</th>
<th>Id</th>
<th>System</th>
</tr>
</thead>
<tbody>
<tr>
<td>/dev/sda1</td>
<td>*</td>
<td>1</td>
<td>...</td>
<td>...</td>
<td>83</td>
<td>Linux</td>
</tr>
<tr>
<td>/dev/sda2</td>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>83</td>
<td>Linux</td>
</tr>
<tr>
<td>/dev/sda3</td>
<td></td>
<td></td>
<td>...</td>
<td></td>
<td>82</td>
<td>Linux swap / Solaris</td>
</tr>
<tr>
<td>/dev/sda4</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>5</td>
<td>Extended</td>
</tr>
<tr>
<td>/dev/sda5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>83</td>
<td>Linux</td>
</tr>
</tbody>
</table>
```

```
i. Write the changes to the partition table and exit:

Command (m for help): w
The partition table has been altered!
Calling ioctl() to re-read partition table.

WARNING: Re-reading the partition table failed with error 16: Device or resource busy.
The kernel still uses the old table. The new table will be used at the next reboot or after you run partprobe(8) or kpartx(8)
Syncing disks.
```

j. Reboot the Oracle VM Server:

```
# reboot
```

12. Create a btrfs file system on the new partition, mounted on /var/lib/docker.

A simple example follows. This example formats an existing partition on /dev/sda5, and mounts it on /var/lib/docker using the /etc/fstab file.

a. Create the btrfs file system on an existing device:

```
# mkfs.btrfs -f -L docker /dev/sda5
```
b. Use the `blkid` command to display the UUID of the device and make a note of this value, for example:

```
# blkid /dev/sda5
/dev/sda5: LABEL="docker" UUID="bb6912ae-e6c4-4ffd-bcdc-37c8c19eb7db"
UUID_SUB="599202a1-fb2d-4352-916f-aee4d034d3bd" TYPE="btrfs"
```

c. Edit the `/etc/fstab` file and add an entry to ensure the file system is mounted when the system boots:

```
UUID=bb6912ae-e6c4-4ffd-bcdc-37c8c19eb7db /var/lib/docker btrfs defaults 1 2
```

d. Create the `/var/lib/docker` directory:

```
# mkdir /var/lib/docker
```

e. Mount all the file systems listed in `/etc/fstab`:

```
# mount -a
```

f. You can verify the partition is mounted using:

```
# df
```

```
Filesystem     1K-blocks    Used Available Use% Mounted on
...            ...        ...        ...       1%   /var/lib/docker
```

More information on using `mkfs.btrfs` is available in the Oracle Linux Administrator’s Guide for Release 6 at:

https://docs.oracle.com/cd/E37670_01/E41138/html/ol_create_btrfs.html

13. Install the Oracle OpenStack for Oracle Linux preinstallation package from the Oracle Yum Server:

```
# yum install openstack-kolla-preinstall
```

14. If you are using a web proxy, configure the `docker` service to use the proxy.

a. Add the following content to the `/etc/sysconfig/docker` file:

```
HTTP_PROXY=proxy_URL:port
http_proxy=$HTTP_PROXY
HTTPS_PROXY=$HTTP_PROXY
https_proxy=$HTTP_PROXY
```

Replace `proxy_URL` and `port` with the appropriate URLs and port numbers for your web proxy.

You may also want to specify that local connections do not need to be proxied by setting the `NO_PROXY` environment variable:

```
NO_PROXY=localhost,127.0.0.1
no_proxy=localhost,127.0.0.1
```

Include a line to export these environment variables:

```
export HTTP_PROXY HTTPS_PROXY http_proxy https_proxy NO_PROXY no_proxy
```

b. Restart the `docker` service:

```
# service docker restart
```
Preparing a Master Node

15. Synchronize the host time with the controller nodes to avoid errors during OpenStack operations. Time synchronization is discussed in more detail in Section 3.4, “Preparing Oracle Linux Nodes”.

a. To synchronize the host with the controller nodes, edit the `/etc/ntp.conf` file to configure the `ntpd` service. Add the controller nodes to the file, for example:

```
server control1.example.com
server control2.example.com
```

b. Create the drift file, which is used to store the system's clock drift value.

```
# touch /var/lib/ntp/drift
```

Start the `ntpd` service and configure it to start following a system reboot.

```
# service ntpd start
# chkconfig ntpd on
```

c. You can check the `ntpd` service is configured and synchronizing correctly using the `ntpq` and `ntpstat` commands, for example:

```
# ntpq -p
remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*control1       10.169.64.1      3 u   58   64  377    0.132   33.490  22.604
control2       .INIT.          16 u    -  128    0    0.000    0.000   0.000
```

```
# ntpstat
synchronised to NTP server (10.196.134.1) at stratum 4
time correct to within 120 ms
polling server every 64 s
```

3.6 Preparing a Master Node

A master node is a host from which you deploy Oracle OpenStack for Oracle Linux to the target nodes using the `kollacli deploy` command.

Typically, you use a controller node as a master node. You should prepare the node as described in Section 3.4, “Preparing Oracle Linux Nodes” before performing the following steps.

If you prefer, you can use a separate host. See Section 3.1, “System Requirements”, for the minimum requirements for a separate host, and you should prepare the node as described in Section 3.4, “Preparing Oracle Linux Nodes” before performing the following steps.

You should only configure a single node as a master node. In order to recover from a failure, you should ensure that you have backups of the `/etc/kolla` and `/usr/share/kolla` directories.

To prepare a master node:

1. Install the OpenStack Kolla CLI (`kollacli`).

   If you are preparing a master node for a new OpenStack deployment:

   ```
   # yum install openstack-kollacli
   ```

   If you are updating a master node for a new release of Oracle OpenStack for Oracle Linux:

   ```
   # yum update openstack-kollacli
   ```

   When you install the `openstack-kollacli` package, the following operations are performed on the node:
• Create the **kolla** user.

  The **kolla** user is the Ansible SSH user and is used to deploy OpenStack services to nodes. The user’s home directory is set to `/usr/share/kolla`.

• Create SSH keys for the **kolla** user.

  The SSH keys are created in the **kolla** user’s home directory. The public key is copied to `/etc/kolla/kollacli/id_rsa.pub` on the node.

• Create the `/etc/sudoers.d/kolla` sudoers configuration file.

  The settings in this file enable the **kolla** user to run commands, such as `ansible` and `ansible-playbook`, as `root` without prompting for a password.

• Create the **kolla** group.

  The **kolla** group is for the users that run the **kollacli** command.

2. Add a user to the **kolla** group.

   To add an existing user to the **kolla** group:

   ```bash
   # usermod -aG kolla username
   ```

   The user must log out and in again for the group setting to take effect.

   **Important**

   For security reasons, always run **kollacli** commands as this user. Never use **root** or the **kolla** user.

### 3.7 Setting up the Docker Registry

A Docker registry is a store of Docker images. A Docker image is a read-only template, which is used to create a Docker container. A Docker registry is required to store the Oracle OpenStack for Oracle Linux Docker images, which are used to deploy the OpenStack containers.

The following instructions assume that you set up the Docker registry on a controller node and that Docker is already configured and running, as described in Section 3.4, “Preparing Oracle Linux Nodes”. If you prefer, you can use a separate host. See Section 3.1, “System Requirements”, for the minimum requirements for a separate host.

#### 3.7.1 Setting up the Docker Registry Server

Before you set up the Docker Registry server, note the following:

- The registry server is a Docker container application. The host must have an Internet connection in order to download the registry image from the public Docker Hub Registry.

- The registry server runs on port 5000 by default. If you set up the registry on a controller node, you must select a different free port on the host (for example port 5443) because port 5000 is used by the OpenStack Keystone service. The other nodes in your OpenStack deployment must be able to access the registry on this port.

- The registry host requires a valid Secure Sockets Layer (SSL) certificate and private key, similar to using SSL for a web server.
If the host already has an SSL certificate, you can use that. However, if the SSL certificate was issued by an intermediate Certificate Authority (CA), you must combine the host's SSL certificate with the intermediate CA's certificate to create a certificate bundle so that Docker can verify the host's SSL certificate. For example:

```bash
# cat controll.example.com.crt intermediate-ca.pem > domain.crt
```

If the host does not already have an SSL certificate, the following instructions provide details for creating a self-signed certificate for testing purposes.

- The registry server requires a btrfs file system mounted on `/var/lib/registry` with at least 15GB available. The following instructions provide details for setting up a btrfs file system using one or more available devices. The device could be a disk partition, an LVM volume, a loopback device, a multipath device, or a LUN.

**To set up a Docker registry server:**

1. Create a btrfs file system for the registry.

   You create a btrfs file system with the utilities available in the `btrfs-progs` package, which should be installed by default.

   Create a btrfs file system on one or more block devices:

   ```bash
   # mkfs.btrfs [-L label] block_device ...
   ```

   where `-L label` is an optional label that can be used to mount the file system.

   For example:

   - To create a file system in a partition `/dev/sdc1`:
     ```bash
     # mkfs.btrfs -L var-lib-registry /dev/sdc1
     ```
     The partition must already exist. Use a utility such as `fdisk` (MBR partitions) or `gdisk` (GPT partitions) to create one if needed.

   - To create a file system across two disk devices, `/dev/sdd` and `/dev/sde`:
     ```bash
     # mkfs.btrfs -L var-lib-registry /dev/sd[de]
     ```
     The default configuration is to stripe the file system data (`raid0`) and to mirror the file system metadata (`raid1`) across the devices. Use the `-d` (data) and `-m` (metadata) options to specify the required RAID configuration. For `raid10`, you must specify an even number of devices and there must be at least four devices.

   - To create a file system in a logical volume named `docker-registry` in the `ol` volume group:
     ```bash
     # mkfs.btrfs -L var-lib-registry /dev/ol/docker-registry
     ```
     The logical volume must already exist. Use Logical Volume Manager (LVM) to create one if needed.

   More information on using `mkfs.btrfs` is available in the *Oracle Linux Administrator’s Guide for Release 7* at:


2. Mount the btrfs file system on `/var/lib/registry`. ```
a. Obtain the UUID of the device containing the btrfs file system.

Use the `blkid` command to display the UUID of the device and make a note of this value, for example:

```
# blkid /dev/sdc1
/dev/sdc1: LABEL="var-lib-registry" UUID="50041443-b7c7-4675-95a3-bf3a30b96c17" \
    UUID_SUB="09de3cb1-2f9b-4bd8-8881-87e591841c75" TYPE="btrfs"
```

If the btrfs file system is created across multiple devices, you can specify any of the devices to obtain the UUID. Alternatively you can use the `btrfs filesystem show` command to see the UUID. For a logical volume, specify the path to the logical volume as the device for example `/dev/ol/docker-registry`. Ignore any UUID_SUB value displayed.

b. Edit the `/etc/fstab` file and add an entry to ensure the file system is mounted when the system boots.

```
UUID=UUID_value /var/lib/registry  btrfs  defaults  1 2
```

Replace `UUID_value` with the UUID that you found in the previous step. If you created a label for the btrfs file system, you can also use the label instead of the UUID, for example:

```
LABEL=label /var/lib/registry  btrfs  defaults  1 2
```

c. Create the `/var/lib/registry` directory.

```
# mkdir /var/lib/registry
```

d. Mount all the file systems listed in `/etc/fstab`.

```
# mount -a
```

e. Verify that the file system is mounted.

```
# df
Filesystem   1K-blocks  Used Available  Use% Mounted on
...           ...      ...       ...    ...   /
/dev/sdc1    ...      ...       ...   ... %  /var/lib/registry
```

3. Add the host's SSL certificate and private key to Docker.

a. Create the `/var/lib/registry/conf.d` directory.

```
# mkdir -p /var/lib/registry/conf.d
```

b. Copy the host's SSL certificate and private key to the `/var/lib/registry/conf.d` directory.

```
# cp certfile /var/lib/registry/conf.d/domain.crt
# cp keyfile /var/lib/registry/conf.d/domain.key
```

where `certfile` is the full path to the host's SSL certificate and `keyfile` is the full path to the host's private key. For example:

```
# cp /etc/pki/tls/certs/control1.example.com.crt \
/var/lib/registry/conf.d/domain.crt
# cp /etc/pki/tls/private/control1.example.com.key \
/var/lib/registry/conf.d/domain.key
```

If the host does not have an SSL certificate and private key, you can create a self-signed certificate for testing purposes, as follows:
```
cd /var/lib/registry/conf.d
openssl req -newkey rsa:4096 -nodes -sha256 -x509 -days 365 \
-keyout domain.key -out domain.crt
Generating a 4096 bit RSA private key
............................................................................................
............................................................................................
.....++
............................................................................................
............................................................................................
.........................++
............................................................................................
............................................................................................
writing new private key to 'domain.key'
-----
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [XX]: US
State or Province Name (full name) []: Massachusetts
Locality Name (eg, city) [Default City]: Boston
Organization Name (eg, company) [Default Company Ltd]:Example Com
Organizational Unit Name (eg, section) []: DevOps
Common Name (eg, your name or your server's hostname) []: control1.example.com
Email Address []: admin@example.com

mkdir -p /etc/docker/certs.d/control1.example.com:

cp /var/lib/registry/conf.d/domain.crt \
/etc/docker/certs.d/control1.example.com:/etc/docker/certs.d/control1.example.com:port/ca.crt

The Common Name must be the same as the fully qualified domain name (FQDN) of the host.
Replace port with the port number you selected for your Docker registry, for example 5443.

c. Change the file permissions on the private key:

```
chmod 600 /var/lib/registry/conf.d/domain.key
```

4. Create the Docker registry container.

In the following example, port is the port number you selected for your Docker registry server, for example 5443.

```
# docker run -d -p port:5000 --name registry --restart=always \
-v /var/lib/registry:/registry_data \
-e REGISTRY_STORAGE_FILESYSTEM_ROOTDIRECTORY=/registry_data \
-e REGISTRY_HTTP_TLS_KEY=/registry_data/conf.d/domain.key \
-e REGISTRY_HTTP_TLS_CERTIFICATE=/registry_data/conf.d/domain.crt \
registry:2.3
Unable to find image 'registry:2.3' locally
2.3: Pulling from library/registry
fdd5d7827f33: Pull complete
a3ed95caeb02: Pull complete
a79b4a92697e: Pull complete
6cb75c7cc30: Pull complete
4831699594bc: Pull complete
Digest: sha256:0c8c2a39d64da5eebf305f9b50e0d889d823d799414c5a1d42f9cdfab2462da19
Status: Downloaded newer image for registry:2.3
3c2a7fae6db4dd7e163cab530e7c40209792afeded771f2853a4c4244a147d7b
```
The registry image is pulled from the public Docker Hub Registry and the Docker registry container is started. The `--restart=always` option ensures that the registry container is started whenever Docker is started.

5. If the registry host uses a self-signed SSL certificate, distribute the SSL certificate to all master and target nodes in your deployment.

Perform the following steps on each node, where `registry_hostname` is the name of the registry host, and `port` is the port number you selected for your Docker registry server, for example 5443:

a. Create the `/etc/docker/certs.d/registry_hostname:port` directory.

```bash
# mkdir -p /etc/docker/certs.d/registry_hostname:port
```

b. Copy the SSL certificate from the registry host.

```bash
# scp root@registry_hostname:/etc/docker/certs.d/registry_hostname:port/ca.crt /
/etc/docker/certs.d/registry_hostname:port/ca.crt
```

For example:

```bash
# mkdir -p /etc/docker/certs.d/controll.example.com:5443
# scp root@controll.example.com:/etc/docker/certs.d/controll.example.com:5443/ca.crt /
/etc/docker/certs.d/controll.example.com:5443/ca.crt
```

c. Restart the `docker` service.

```bash
# systemctl restart docker.service
```

### 3.7.2 Importing Images into the Docker Registry

Once you have set up a Docker registry server, you import the Oracle OpenStack for Oracle Linux Docker images into the registry so that they can be used to deploy OpenStack containers.

**To import the Oracle OpenStack for Oracle Linux images into the Docker registry:**

1. Ensure the `unzip` and `bzip2` packages are installed on the host.

```bash
# yum install unzip bzip2
```


On Oracle Software Delivery Cloud, search by product for ‘openstack’ and then select **Oracle OpenStack for Oracle Linux**.

Download the zip file to a temporary location on the host, such as `/tmp`.

3. Change to the temporary directory and unzip the Oracle OpenStack for Oracle Linux Docker images zip file.

```bash
# cd /tmp
# unzip ol-openstack-images-version.zip
```

```
Archive:  ol-openstack-images-version.zip
    inflating: import_to_registry.sh
    inflating: ol-openstack-images-version.tar.bz2
    inflating: ol-openstack-images-version.sha256sum
```

The following files are unzipped to the directory:
Importing Images into the Docker Registry

- **import_to_registry.sh**: A script to load the images to the Docker registry.


- **ol-openstack-images-version.tar.bz2**: Contains the Oracle OpenStack for Oracle Linux Docker images files. Do not decompress this file.

4. Use the **import_to_registry.sh** script to import the Oracle OpenStack for Oracle Linux Docker images to the registry.

```
# ./import_to_registry.sh [-c] registry_hostname:port
```

where *registry_hostname* is the name of the host, and *port* is the port number you selected for your Docker registry, for example 5443.

For example:

```
# ./import_to_registry.sh control1.example.com:5443
Checking file integrity:
ol-openstack-images-2.1.1.tar.bz2: OK

Please wait, this will take quite some time to complete!
The push refers to a repository [control1.example.com:5443/oracle/ol-openstack-nova-base]
(len: 1)
a8e1024207cf: Pushed
...
63adc2d9636b: Pushed
2.1.1: digest: sha256:fd81fb1da57e6134a4eb130eab8289e72a69642b582f0cdf06ac55c913a size: 9782
```

It may take some time to import the images to the Docker registry.

5. Confirm the registry is working.

Run the following command on each node in your deployment:

```
# docker pull registry_hostname:port/oracle/ol-openstack-base:2.1.1
```

This command confirms that the node can download a Docker image from the registry and verifies that there are no issues with access to the registry port or with the registry host's SSL certificate.
4.1 Setting up Target Nodes

Before you deploy OpenStack services, you need to configure the target nodes and services.

1. **On the master node, log in as a user that is a member of the kolla group.**
   
   You add users to the **kolla** group as part of setting up a master node, see Section 3.6, “Preparing a Master Node”.
   
   All of the following steps use the **kollacli** command, and you must be a member of the **kolla** group to use this command.

2. **Add the target nodes to the deployment.**
   
   You add nodes to the deployment with the **kollacli host add** command.

   ```bash
   $ kollacli host add host
   ```

   where **host** is the fully qualified domain name or IP address of the host.

   Use the **kollacli host list** command to check that you have added all the hosts.

3. **Set up the kolla user on the target nodes.**
   
   You only need to perform this step once for each target node. Only repeat this step if you change the SSH keys for the **kolla** user.
The **kolla** user is the Ansible SSH user that is used to deploy OpenStack services. You need to copy the SSH public key for the **kolla** user to each host, and confirm that the **kolla** user can log in to the host using SSH keys.

You set up the **kolla** user with the **kollacli host setup** command.

```
$ kollacli host setup host
```

where *host* is the fully qualified domain name or IP address of the host.

You are prompted for a password value. By default, the password is for the **root** user on the target node.

If you want to use a different user to set up the **kolla** user, set the **KOLLA_CLI_SETUP_USER** environment variable before running the **kollacli host setup** command. The variable should contain the user name of an alternative user on the target host. The user must already exist on the target host and have sufficient privileges to be able to write to the */usr/share/kolla/.ssh/authorized_keys* file, which is owned by the **kolla** user and group.

If you want to set up multiple hosts with a single command, you can create a YAML file that contains the names of the hosts to set up and the passwords to use. If the file specifies a user name for a host (optional), this is used instead of **root** or the user name in the **KOLLA_CLI_SETUP_USER** environment variable. For example, you might have a file named **hosts.yml** in the */myhome* directory which contains:

```yaml
---
control1.example.com:
    password: password
control2.example.com:
    password: password
compute1.example.com:
    password: password
    username: user_name
compute2.example.com:
    password: password
    username: user_name
compute3.example.com:
    password: password
    username: user_name
compute4.example.com:
    password: password
    username: user_name
```

Caution

For security reasons, make sure you secure the file containing these credentials.

To set up the **kolla** user using a YAML file:

```
$ kollacli host setup --file /myhome/hosts.yml
```

When you have set up the **kolla** user, use the **kollacli host check** command to verify the user.

```
kollacli host check { host | all }
```

where *host* is the fully qualified domain name or IP address of the host, or **all** for all hosts.

4. **Assign target nodes to each group.**
Groups are used to associate the target nodes with OpenStack services. Target nodes in the same group run the same OpenStack services. The default groups are control, compute, database, network, and storage.

You assign target nodes to groups with the `kollacli group addhost` command:

```bash
$ kollacli group addhost group_name host
```

where `host` is the fully qualified domain name or IP address of the host.

For example, to add a node to the control group:

```
$ kollacli group addhost control control1.example.com
```

A node can belong to more than one group and can run multiple OpenStack services. When you assign nodes to groups, remember the following "rules":

- The nodes in the compute group must not be assigned to the control group.
- The control group must contain at least two nodes.
- The number of nodes in the database group must always be a multiple of two.
- Each group must contain at least two nodes to enable high availability.

For more information on using groups, see Section 2.2, “Using Groups to Deploy Services”.

Use the `kollacli group listhosts` and the `kollacli host list` commands to check that you have assigned the hosts correctly.

### 4.2 Setting up Services

Before you can deploy OpenStack services, you need to enable the services you want to deploy, and then configure the services by setting passwords and properties.

1. **On the master node, log in as a user that is a member of the kolla group.**

   You add users to the `kolla` group as part of setting up a master node, see Section 3.6, “Preparing a Master Node”.

   All of the following steps use the `kollacli` command, and you must be a member of the `kolla` group to use this command.

2. **Enable the OpenStack services you want to deploy.**

   All OpenStack services are enabled by default, apart from the Swift object storage and the Ceilometer telemetry services.

   You enable and disable services with the `kollacli property set` command:

   ```bash
   $ kollacli property set enable_service_name yes|no
   ```

   For example, to enable the Swift service and disable the Cinder service:

   ```bash
   $ kollacli property set enable_swift yes
   $ kollacli property set enable_cinder no
   ```
Many OpenStack services contain several components, and each component is deployed as a separate Docker container. When a service is enabled, all of the service components (known as subservices) are also enabled by default. If required, you can enable and disable subservices with the kollacli property set command, although normally you do not need to do this.

To see a list of the available services and their subservices, use the kollacli service list command. Use the kollacli property list command to see the properties for enabling and disabling services and subservices.

3. **Configure the passwords for all services.**

You only need to perform this step once.

Typically, when you deploy an OpenStack service, a database is created for the service and the credentials for the service admin user are registered with the Keystone service. You must set passwords for the database and the service admin user. Passwords for other OpenStack features might also be required.

You set passwords with the kollacli password set command:

```bash
$ kollacli password set password_name
```

For example, for the Cinder service set the following passwords:

```bash
$ kollacli password set cinder_database_password
$ kollacli password set cinder_keystone_password
```

You are prompted for a password, and to confirm the password. The password value is not displayed on screen. You might want to use a strong password generator to generate the passwords.

If you do not set passwords, the default password is used, which is "password". For security reasons, it is best to set passwords for all the password names listed in the following table, even if you do not deploy all the OpenStack services.

<table>
<thead>
<tr>
<th>Password Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ceilometer_database_password</td>
<td>Database password for the Ceilometer service.</td>
</tr>
<tr>
<td>ceilometer_keystone_password</td>
<td>Password for the Ceilometer service user, named ceilometer by default.</td>
</tr>
<tr>
<td>cinder_database_password</td>
<td>Database password for the Cinder service.</td>
</tr>
<tr>
<td>cinder_keystone_password</td>
<td>Password for the Cinder service user, named cinder by default.</td>
</tr>
<tr>
<td>database_password</td>
<td>Root password for the MySQL database.</td>
</tr>
<tr>
<td>glance_database_password</td>
<td>Database password for the Glance service.</td>
</tr>
<tr>
<td>glance_keystone_password</td>
<td>Password for the Glance service user, named glance by default.</td>
</tr>
<tr>
<td>heat_database_password</td>
<td>Database password for the Heat service.</td>
</tr>
<tr>
<td>heat_domain_admin_password</td>
<td>Password for the Heat ‘stack-domain-admin’ user.</td>
</tr>
<tr>
<td>heat_keystone_password</td>
<td>Password for the Heat service user, named heat by default.</td>
</tr>
<tr>
<td>horizon_database_password</td>
<td>Database password for the Horizon service.</td>
</tr>
<tr>
<td>keystone_admin_password</td>
<td>Password for the Keystone admin user.</td>
</tr>
<tr>
<td>keystone_database_password</td>
<td>Database password for the Keystone service.</td>
</tr>
<tr>
<td>murano_database_password</td>
<td>Database password for the Murano service.</td>
</tr>
</tbody>
</table>
### Setting up Services

<table>
<thead>
<tr>
<th>Password Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>murano_keystone_password</td>
<td>Password for the Murano service user, named <code>murano</code> by default.</td>
</tr>
<tr>
<td>neutron_database_password</td>
<td>Database password for the Neutron service.</td>
</tr>
<tr>
<td>neutron_keystone_password</td>
<td>Password for the Neutron service user, named <code>neutron</code> by default.</td>
</tr>
<tr>
<td>nova_database_password</td>
<td>Database password for the Nova service.</td>
</tr>
<tr>
<td>nova_keystone_password</td>
<td>Password for the Nova service user, named <code>nova</code> by default.</td>
</tr>
<tr>
<td>rabbitmq_password</td>
<td>Password for the RabbitMQ <code>guest</code> user.</td>
</tr>
<tr>
<td>swift_keystone_password</td>
<td>Password for the Swift service user, named <code>swift</code> by default.</td>
</tr>
</tbody>
</table>

4. **Configure the deployment by setting properties.**

You set properties for services with the `kollacli property set` command:

```bash
$ kollacli property set property_name property_value
```

Use the `kollacli property list` command to see a list of the available properties. Some properties are used in the configuration files for the OpenStack services when they are deployed. These properties usually contain the name of the service in the property name, for example names of the properties for the Glance image service start with "glance_".

For examples of setting properties for specific services, see Section 4.7.1, “Configuring VLAN Networks” and Section 4.10, “Setting up Ceilometer”.

When you set properties, they are global properties that can be applied to all hosts and OpenStack services in your deployment. However, to enable you to tune your deployment, you can also set properties that are specific to particular groups or individual hosts, see Section 4.4, “Setting Properties for Groups or Hosts”.

For the majority of the properties, the default property values are sufficient.

The only properties that you **must** set are the network settings for the deployment. Be sure to review the information in Section 3.3, “Host Network Requirements” before setting the following properties:

**docker_registry**

The host name or IP address, and the port number, of the server that runs your Docker registry, for example myregistry.example.com:5000.

**kolla_external_address**

The DNS name or IP address to be used in the public OpenStack API endpoints. By default, this property is set to the same value as `kolla_internal_address`.

For private cloud deployments, do not change the default setting (the `kolla_internal_address` is used).

For public cloud deployments, this is a DNS name that resolves to a public IP address. The OpenStack operator is responsible for mapping the public IP address to the `kolla_internal_address`, for example through a firewall or load balancer.

By default, this value is used to populate part of each OpenStack service’s `public_url` in Keystone. But you can configure the address for individual services if you prefer.
**kolla_internal_address**

The IP address on the internal management/API network used to access OpenStack control services.

This is an unused IP address on the internal management/API network and is used by Keepalived as the virtual IP (VIP) address. Keepalived maps the VIP address to the real IP address of the controller node that runs the current active HAProxy.

By default, this value is used to populate part of each OpenStack service's `private_url` and `admin_url` in Keystone. But you can configure the address for individual services if you prefer.

**network_interface**

The name of the network interface (for example, `em1`) on all nodes which is connected to the internal management/API network.

By default, this interface is also used for the virtual machine and storage network traffic. If you set up separate networks for this traffic, set the name of the network interface on each network by configuring the `tunnel_interface` (virtual machine network), and `storage_interface` (storage network) properties.

**neutron_external_interface**

The name of the network interface (for example, `em2`) on all network nodes which is connected to the external network where the neutron public network will be created. This interface should not have an IP address and should not be the same as the `network_interface`.

**virtual_router_id**

The virtual router ID used by Keepalived to manage the virtual IP address. The default ID is 51.

Only change this property if you have multiple OpenStack deployments on the same management/API network. Each deployment must have a unique virtual router ID.

If the nodes have differing network interface names, use group and host properties to set different interface names, see Section 4.4, "Setting Properties for Groups or Hosts".

### 4.3 Example Setup for an Initial Deployment

The following is an example of how to set up an initial OpenStack deployment which is enabled for high availability. In this example:

- The default groups are used: control, compute, database, network, and storage.
- The deployment contains six nodes, four compute nodes, and the other two nodes run all other services.
- High availability is enabled because there are at least two nodes for each group.
- You want to deploy all the OpenStack services that are enabled by default except the Murano service, and you want to deploy the Ceilometer service, which is not enabled by default.
- All nodes are connected to the management/API network using the `em1` network interface.
- The virtual IP address on the management/API network is `10.0.0.20`.
- All the network nodes are connected to the public network using the `em2` network interface.
1. On the master node, log in as a user that is a member of the \texttt{kolla} group.

2. Add the target nodes to the deployment.

   \begin{verbatim}
   $ kollacli host add control1.example.com
   $ kollacli host add control2.example.com
   $ kollacli host add compute1.example.com
   $ kollacli host add compute2.example.com
   $ kollacli host add compute3.example.com
   $ kollacli host add compute4.example.com
   \end{verbatim}

3. Check that the target nodes are added.

   \begin{verbatim}
   $ kollacli host list
   +----------------------+--------+
   | Host                 | Groups |
   +----------------------+--------+
   | compute1.example.com | []     |
   | compute2.example.com | []     |
   | compute3.example.com | []     |
   | compute4.example.com | []     |
   | control1.example.com | []     |
   | control2.example.com | []     |
   +----------------------+--------+
   \end{verbatim}

4. Set up the \texttt{kolla} user on the target nodes.

   \begin{verbatim}
   $ kollacli host setup control1.example.com
   root password for control1.example.com:
   Starting setup of host (control1.example.com).
   Host (control1.example.com) setup succeeded.
   
   $ kollacli host setup control2.example.com
   ...
   $ kollacli host setup compute1.example.com
   ...
   $ kollacli host setup compute2.example.com
   ...
   $ kollacli host setup compute3.example.com
   ...
   $ kollacli host setup compute4.example.com
   ...
   \end{verbatim}

5. Check that the \texttt{kolla} user is set up correctly on the target nodes.

   \begin{verbatim}
   $ kollacli host check all
   Host (compute4.example.com): success
   Host (compute3.example.com): success
   Host (compute2.example.com): success
   Host (compute1.example.com): success
   Host (control2.example.com): success
   Host (control1.example.com): success
   \end{verbatim}

6. Assign target nodes to each group.

   \begin{verbatim}
   $ kollacli group addhost control control1.example.com
   $ kollacli group addhost control control2.example.com
   
   $ kollacli group addhost database control1.example.com
   $ kollacli group addhost database control2.example.com
   
   $ kollacli group addhost network control1.example.com
   $ kollacli group addhost network control2.example.com
   
   $ kollacli group addhost storage control1.example.com
   $ kollacli group addhost storage control2.example.com
   \end{verbatim}
Example Setup for an Initial Deployment

7. Check that the target nodes are assigned correctly.

```bash
$ kollacli group listhosts
+----------+------------------------------------------------------------------------+
| Group    | Hosts                                                                  |
+----------+------------------------------------------------------------------------+
| compute  | ['compute1.example.com', 'compute2.example.com',                       |
|          | 'compute3.example.com', 'compute4.example.com']                        |
| control  | ['control1.example.com', 'control2.example.com']                       |
| database | ['control1.example.com', 'control2.example.com']                       |
| network  | ['control1.example.com', 'control2.example.com']                       |
| storage  | ['control1.example.com', 'control2.example.com']                       |
+----------+------------------------------------------------------------------------+

$ kollacli host list
+----------------------+-----------------------------------------------+
| Host                 | Groups                                        |
+----------------------+-----------------------------------------------+
| compute1.example.com | ['compute']                                   |
| compute2.example.com | ['compute']                                   |
| compute3.example.com | ['compute']                                   |
| compute4.example.com | ['compute']                                   |
| control1.example.com | ['control', 'storage', 'network', 'database'] |
| control2.example.com | ['control', 'storage', 'network', 'database'] |
+----------------------+-----------------------------------------------+
```

8. Enable the OpenStack services you want to deploy.

All OpenStack services are enabled by default, apart from the Swift object storage and the Ceilometer telemetry services. In this example, you want to disable the Murano service and enable the Ceilometer service.

```bash
$ kollacli property set enable_murano no
$ kollacli property set enable_ceilometer yes
```

9. Check that the correct services are enabled.

```bash
$ kollacli property list
+-----+------------------------------------------+------------------------------------------+
| OVR | Property Name                            | Property Value                           |
+-----+------------------------------------------+------------------------------------------+
|     | enable_ceilometer                        | yes                                      |
|     | enable_cinder                            | yes                                      |
|     | enable_glance                            | yes                                      |
|     | enable_haproxy                           | yes                                      |
|     | enable_heat                              | yes                                      |
|     | enable_horizon                           | yes                                      |
|     | enable_keystone                          | yes                                      |
|     | enable_memcached                         | yes                                      |
|     | enable_murano                            | no                                       |
|     | enable_mysqlcluster                      | yes                                      |
|     | enable_neutron                           | yes                                      |
|     | enable_memcached                         | yes                                      |
|     | enable_murano                            | no                                       |
|     | enable_neutron                           | yes                                      |
+-----+------------------------------------------+------------------------------------------+
```
Example Setup for an Initial Deployment

| --- | enable_nova | yes |
| --- | enable_rabbitmq | yes |
| --- | enable_swift | no |

10. Configure the passwords for all services.

$ kollacli password set database_password
$ kollacli password set ceilometer_database_password
$ kollacli password set ceilometer_keystone_password
$ kollacli password set cinder_database_password
$ kollacli password set cinder_keystone_password
$ kollacli password set glance_database_password
$ kollacli password set glance_keystone_password
$ kollacli password set heat_database_password
$ kollacli password set heat_keystone_password
$ kollacli password set heat_domain_admin_password
$ kollacli password set horizon_database_password
$ kollacli password set keystone_admin_password
$ kollacli password set keystone_database_password
$ kollacli password set murano_database_password
$ kollacli password set murano_keystone_password
$ kollacli password set neutron_database_password
$ kollacli password set neutron_keystone_password
$ kollacli password set nova_database_password
$ kollacli password set nova_keystone_password
$ kollacli password set rabbitmq_password
$ kollacli password set swift_keystone_password

11. Configure the deployment by setting properties.

The default property values are sufficient, so you only need to configure the network properties.

$ kollacli property set docker_registry registry.example.com:5000
$ kollacli property set kolla_internal_address 10.0.0.20
$ kollacli property set network_interface em1
$ kollacli property set neutron_external_interface em2

12. Check that the properties are set.

$ kollacli property list --max-width 40

<table>
<thead>
<tr>
<th>OVR</th>
<th>Property Name</th>
<th>Property Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>docker_registry</td>
<td>registry.example.com:5000</td>
</tr>
<tr>
<td></td>
<td>kolla_internal_address</td>
<td>10.0.0.20</td>
</tr>
<tr>
<td></td>
<td>network_interface</td>
<td>em1</td>
</tr>
<tr>
<td></td>
<td>neutron_external_interface</td>
<td>em2</td>
</tr>
</tbody>
</table>
4.4 Setting Properties for Groups or Hosts

Because OpenStack Kolla uses properties and templates to configure OpenStack services, deploying OpenStack Kolla containers works best where the deployment nodes are identical. However, where your deployment nodes are not identical, or you have custom configuration requirements, you can set properties that are specific to groups or hosts.

When you use the `kollacli property set` command without any options, the properties are global properties that apply to all hosts and OpenStack services in your deployment. However, to enable you to tune your deployment, you can use the `--groups` and `--hosts` options to set properties that are specific to particular groups or individual hosts.

The properties you set are used in the following order of precedence:

1. Host properties.
2. Group properties.
4. Default properties.

Group properties files are used in alphabetical order, for example properties set for the group named control have precedence over properties for the group named database.

**To set properties for hosts or groups**

1. Identify the name of the property that you want to set.

   Use the `kollacli property list` command to see a list of properties. By default, only the properties with values of less than 50 characters are shown. Use the `--all` option to list all properties, or set the `KOLLA_PROP_LIST_LENGTH` environmental variable with the maximum number of characters you want.

2. (Optional) Set the global property value.

   If the default value for the property is not what you require for the majority of the groups or hosts, you should set the global property for your deployment:

   ```sh
   $ kollacli property set property_name value
   ```

   Use the `kollacli property list` command with the `--long` option to list default and global properties.

3. Set the property for individual groups or hosts.

   To set the property for one or more groups:

   ```sh
   $ kollacli property set property_name value --groups group_list
   ```

   where `group_list` is a comma-separated list of group names. The names must match the names of the defined groups. Use the `kollacli group listservices` command to list the defined groups.

   To set the property for one or more hosts:

   ```sh
   $ kollacli property set property_name value --hosts host_list
   ```
where *host_list* is a comma-separated list of host names. The host names must match the names of the hosts added to the deployment. Use the *kollacli host list* command to list the hosts.

You cannot use the *--groups* and *--hosts* options together.

### To check host and group properties

- To check whether a property is overridden by a group or host property:

```
$ kollacli property list
```

<table>
<thead>
<tr>
<th>OVR</th>
<th>Property Name</th>
<th>Property Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>*--</td>
<td>openstack_release</td>
<td>2.1.1</td>
</tr>
<tr>
<td>---</td>
<td>openvswitch_db_tag</td>
<td>{{ openstack_release }}</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The *OVR* column in the output indicates whether a property has been set that overrides either a default property or a global property. The *OVR* column contains three characters:

- The first character is either "*" or ",", where "*" means the property is overridden by a global property, and "," means the property is not overridden by a global property.
- The second character is either "G" or ",", where "G" means the property is overridden by a group property, and "," means the property is not overridden by a group property.
- The third character is either "H" or ",", where "H" means the property is overridden by a host property, and "," means the property is not overridden by a host property.

- To check group properties:

```
$ kollacli property list --groups all|group_list
```

where *group_list* is a comma-separated list of group names.

- To check host properties:

```
$ kollacli property list --hosts all|host_list
```

where *host_list* is a comma-separated list of host names.

### To remove properties for groups or hosts

- To remove a group property:

```
$ kollacli property clear property_name --groups all|group_list
```

where *group_list* is a comma-separated list of group names.

- To remove a host property:

```
$ kollacli property clear property_name --hosts all|host_list
```

where *host_list* is a comma-separated list of host names.
4.4.1 Dealing With Variable Network Interface Names

The following is an example of how to use global, group, and host properties to deal with the case where you need to use different interface names for the `network_interface` property.

1. Check the current value for the `network_interface` property.

```bash
$ kollacli property list
```

```
+-----+------------------------------------------+------------------------------------------+
| OVR | Property Name                            | Property Value                           |
+-----+------------------------------------------+------------------------------------------+
| --- | network_interface                        | eth0                                     |
| --- |                                          |                                          |
+-----+------------------------------------------+------------------------------------------+
```

The first "-" character in the OVR column indicates that `eth0` is the default property value.

As the majority of your nodes are Oracle Linux nodes with a network interface named `em1`, it is best to set a global property with this value.

2. Set the global property value.

```bash
$ kollacli property set network_interface em1
```

3. Check that the global property value is applied.

```bash
$ kollacli property list
```

```
+-----+------------------------------------------+------------------------------------------+
| OVR | Property Name                            | Property Value                           |
+-----+------------------------------------------+------------------------------------------+
| --- | network_interface                        | em1                                      |
| --- |                                          |                                          |
+-----+------------------------------------------+------------------------------------------+
```

The "*" character in the OVR column indicates that `em1` is the global property value.

However, all your compute nodes are Oracle VM Server nodes with a network interface named `eth0`. As you are only using Oracle VM Server for your compute nodes, it is best to set a group property for the compute group.

4. Set the property value for the compute group.

```bash
$ kollacli property set network_interface eth0 --groups compute
```

5. Check that the group property value is applied.

```bash
$ kollacli property list
```

```
+-----+------------------------------------------+------------------------------------------+
| OVR | Property Name                            | Property Value                           |
+-----+------------------------------------------+------------------------------------------+
| G-  | network_interface                        | em1                                      |
| --- |                                          |                                          |
+-----+------------------------------------------+------------------------------------------+
```

The "*G" character in the OVR column indicates that `em1` is still the global property value, but the "G" character indicates that the property has been set for one or more groups.

To check which groups have the property value set:

```bash
$ kollacli property list --groups all
```
Setting up Glance

<table>
<thead>
<tr>
<th>OVR</th>
<th>Property Name</th>
<th>Property Value</th>
<th>Group</th>
</tr>
</thead>
<tbody>
<tr>
<td>*G-</td>
<td>network_interface</td>
<td>eth0</td>
<td>compute</td>
</tr>
</tbody>
</table>

However, the storage node storage1.example.com needs to be configured to use the network interface named em3, so you need to set a property specific to this host.

6. Set the property value for the storage1.example.com host.

```bash
$ kollacli property set network_interface em3 --hosts storage1.example.com
```

7. Check that the host property value is applied.

```bash
$ kollacli property list
```

The "*" character in the OVR column indicates that em1 is still the global property value, and the "H" character indicates that the property has been set for one or more hosts.

To check which hosts have the property value set:

```bash
$ kollacli property list --hosts all
```

When you deploy OpenStack with these settings, all hosts will use em1 as the setting for the `network_interface` property apart from hosts that belong the compute group, which use eth0, and the host storage1.example.com, which uses em3.

4.5 Setting up Glance

The Glance image service is the OpenStack service used to manage virtual machine images. By default, it runs on controller nodes.

The Glance service uses the local file system to store virtual machine images and is not, by default, configured for high availability (HA). To enable HA for Glance, you can use a shared file system, such as NFS, mounted on `/var/lib/glance` on each Glance node.

Alternatively, you can deploy Glance to a single node. This avoids any HA issues with file-based storage, though it also means that Glance is not HA-enabled. To deploy Glance to a single node, create a group which contains only Glance, and add one host to that group.

Using a Shared File System for Glance

The glance_api container mounts the `/var/lib/glance` directory on the host and uses it to store virtual machine images. To provide for high availability, you can mount a shared file system on `/var/lib/glance` on each Glance host.

You must ensure that the file system is mounted before the Glance containers are started because the Docker daemon does not wait for file systems to be mounted before it starts the containers. You can
configure *systemd* to check that a file system is mounted before starting the Docker service. For example, if you have added an entry to /etc/fstab to mount an NFS file system at boot time, perform the following steps:

1. Configure *systemd* to mount the NFS file system before starting the *docker* service.

   Create the drop-in file /etc/systemd/system/docker.service.d/nfs.conf with the following content:

   ```
   [Unit]
   After=remote-fs.target
   ```

2. Reload *systemd* manager configuration.

   ```
   # systemctl daemon-reload
   ```

3. Restart the *docker* service.

   ```
   # systemctl restart docker.service
   ```

### 4.6 Setting up Nova

The Nova compute service is responsible for managing the hypervisors and virtual machine instances. You might need to perform additional configuration before you deploy this service.

#### 4.6.1 Automatic Hypervisor Configuration

For the Nova compute service, the *virt_type* option in the [libvirt] section of the *nova.conf* configuration file sets the hypervisor that runs the instances in your deployment. KVM is the default hypervisor.

In an OpenStack deployment, it is possible to use a mixture of hypervisors as compute nodes. To simplify configuration, when you deploy the Nova compute service to a node, the hypervisor is detected and the *virt_type* option is configured automatically. The following table shows the *virt_type* option settings and the conditions when they are used.

<table>
<thead>
<tr>
<th>Setting</th>
<th>Conditions</th>
</tr>
</thead>
</table>
| xen     | Platform: Linux  
|         | Distribution: Oracle VM Server |
| kvm     | Platform: Linux  
|         | Distribution: Oracle Linux Server  
|         | Virtualization support: enabled |
| qemu    | Platform: Linux  
|         | Distribution: Oracle Linux Server  
|         | Virtualization support: disabled |
| hyperv  | Platform: Microsoft Windows |

To check whether virtualization support is enabled on the compute node, run the following command:

```bash
$ egrep '(vmx|svm)' /proc/cpuinfo
```
If virtualization support is enabled, the output should contain \texttt{vmx} (for an Intel CPU) or \texttt{svm} (for an AMD CPU).

### 4.6.2 Preparing a Compute Node

Before you deploy Nova compute services to a compute node, perform the following:

- **Ensure you have sufficient disk space, if you use ephemeral storage for instances (virtual machines).**

  The Nova data container uses the \texttt{/var/lib/kolla/var/lib/nova/instances} directory on the compute node for ephemeral storage for instances. You must ensure there is sufficient disk space in this location for the amount of virtual machine data you intend to store.

- **(Oracle Linux compute nodes only) Stop and disable the \texttt{libvirtd} service.**

  ```
  # systemctl stop libvirtd.service
  # systemctl disable libvirtd.service
  ```

  If the \texttt{libvirtd} service is running, this prevents the \texttt{nova_libvirt} container from starting when you deploy Nova services.

  Do not perform this step on Oracle VM Server compute nodes.

- **Stop iSCSI initiator processes.**

  You only need to perform this step if you are using Cinder block storage with a volume driver that uses the iSCSI protocol.

  By default, the Cinder block storage service uses local volumes managed by the Linux Logical Volume Manager (LVM). The Cinder LVM volume driver uses the iSCSI protocol to connect an instance to a volume and the \texttt{nova_iscsid} container handles the iSCSI session. If there are iSCSI initiator processes running on the compute node, this prevents the \texttt{nova_iscsid} container from starting when you deploy Nova services.

  First, unmount the file systems on any attached iSCSI disks and disconnect from all iSCSI targets. Then do \texttt{either} of the following:

  - **Uninstall the iscsi-initiator-utils package.**

    ```
    # yum remove iscsi-initiator-utils
    ```

  - **Disable iSCSI services.**

    On Oracle Linux compute nodes:

    ```
    # systemctl stop iscsid.socket iscsiucio.socket iscsid.service
    # systemctl disable iscsid.socket iscsiucio.socket iscsid.service
    ```

    On Oracle VM Server compute nodes:

    ```
    # service iscsid stop
    # chkconfig iscsid off
    ```

### 4.6.3 Setting the iSCSI Initiator Name

By default, the Cinder block storage service uses the iSCSI protocol to connect instances to volumes. The \texttt{nova_iscsid} container runs on compute nodes and handles the iSCSI session using an iSCSI initiator name that is generated when you deploy Nova compute services.
If you prefer, you can configure your own iSCSI initiator name. You set the iSCSI initiator name in the `/etc/kolla/nova-iscsid/initiatorname.iscsi` file on each compute node. If the `initiatorname.iscsi` file does not exist, create it. The file has one line, which contains the name of the initiator, in the format:

```
InitiatorName=iqn.yyyy-mm.naming-authority:unique_name
```

For example:

```
InitiatorName=iqn.1988-12.com.oracle:myiscsihost
```

### 4.6.4 Enabling iSCSI Multipath

The Nova compute service supports iSCSI multipath for failover purposes and increased performance. When multipath is enabled, the iSCSI initiator (the Compute node) is able to obtain a list of addresses from the storage node that the initiator can use as multiple paths to the iSCSI LUN (the Cinder volume).

To enable iSCSI multipath:

```
$ kollacli property set nova_iscsi_use_multipath true
```

### 4.7 Setting up Neutron

The Neutron network service enables you to create and attach interface devices managed by other OpenStack services to networks.

#### 4.7.1 Configuring VLAN Networks

With Neutron network services, the default tenant networking option is to use GRE/VxLANs. GRE/VxLANs are the preferred tenant networking option for enterprise deployments. If you prefer to use VLANs for your tenant networks, you need to use the `kollacli property set` command to set the following properties before you perform the deployment:

- `neutron_tenant_type`
  - The tenant network type. Valid options for this are `vlan`, `gre`, and `vxlan`. The default is `vxlan`. To use VLANs for tenant networks, set this to `vlan`.

- `neutron_vlan_physnet`
  - The name of the VLAN network. The default is `physnet1`, which is generally used to name flat networks. To avoid confusion with a flat network, you should change this to something other than the default, for example, `physnet2`.

- `neutron_vlan_range`
  - The range for VLAN IDs, in the format `start_range:end_range`. The default range is `1:1000`.

- `neutron_vlan_bridge`
  - The name for the VLAN network bridge. The default name is `br-vlan`.

- `neutron_vlan_interface`
  - The VLAN traffic network interface name. The network interface must be available on each compute and network node, and by default must have the same name.
If the nodes have differing network interface names, you can use group and host properties to set different interface names, see Section 4.4, “Setting Properties for Groups or Hosts”.

The interface must not have an IP address (because it is a bridged interface) and it must not be the same interface as either the network_interface or neutron_external_interface.

For example:

```bash
$ kollacli property set neutron_tenant_type vlan
$ kollacli property set neutron_vlan_physnet physnet2
$ kollacli property set neutron_vlan_range 1000:2000
$ kollacli property set neutron_vlan_interface em3
```

### 4.8 Setting up Cinder

The Cinder block storage service provides persistent block storage for OpenStack instances to use. The Cinder service is enabled by default.

Cinder requires some form of back-end storage. By default, Cinder uses volumes in a local volume group managed by the Linux Logical Volume Manager (LVM). The local volume group must be named cinder-volumes and has to be configured manually. You can also enable Cinder to use specialized storage appliances by configuring vendor-specific volume drivers.

Cinder also provides a backup service, which enables you to automatically back up volumes to an external storage. By default, the external storage is an NFS share.

#### 4.8.1 Creating the cinder-volumes Volume Group

By default, the Cinder block storage service uses local volumes managed by the Linux Logical Volume Manager (LVM). The Cinder service creates and manages the volumes in an LVM volume group called cinder-volumes on the storage node. You have to manually create this volume group.

Perform the following steps on each storage node:

1. Install the LVM tools.

   The LVM tools are usually installed by default. If they not installed, install them:
   ```bash
   # yum install lvm2
   ```

2. Use the `pvcreate` command to set up the devices that you want to use as physical volumes with LVM.

   If the devices contain any existing data, back up the data.
   ```bash
   # pvcreate [options] device ...
   ```

   For example, to set up `/dev/sdb` and `/dev/sdc` devices as physical volumes:
   ```bash
   # pvcreate --v /dev/sdb[bc]  
   Set up physical volume for "/dev/sdb" with 41943040 available sectors
   Zeroing start of device /dev/sdb
   Writing physical volume data to disk "/dev/sdb*
   Physical volume "/dev/sdb" successfully created
   ...
   ```

3. Use the `vgcreate` command to create the cinder-volumes volume group.

   ```bash
   # vgcreate [options] cinder-volumes physical_volume ...
   ```

   For example, to create the volume group from the physical volumes `/dev/sdb` and `/dev/sdc`:
4.8.2 Configuring Alternative Volume Drivers for Cinder

If you have dedicated storage appliances, you do not have to use the default Cinder LVM volume driver.

To use a different volume driver, you add the configuration settings for the driver to the `/etc/kolla/config/cinder.conf` file on the master node. If this file does not exist, create it.

The `volume_driver` configuration setting is used to specify the volume driver:

```bash
volume_driver = cinder.volume.drivers.driver_name
```

**Note**
Using an NFS driver for Cinder volumes is not supported.

For the **Oracle ZFS Storage Appliance iSCSI driver**, use:

```bash
volume_driver = cinder.volume.drivers.zfssa.zfssaiscsi.ZFSSAISCSIDriver
```

For more information about this driver, see [http://docs.openstack.org/juno/config-reference/content/zfssa-volume-driver.html](http://docs.openstack.org/juno/config-reference/content/zfssa-volume-driver.html).

For **Oracle Flash Storage Systems**, use:

```bash
volume_driver = cinder.volume.drivers.ofs.ofs.OracleFSSFibreChannelDriver
```

To download the Cinder volume driver for Oracle Flash Storage Systems, go to:

For more information about the available Cinder volume drivers and their configuration settings, see the [OpenStack Configuration Reference](http://docs.openstack.org/kilo/config-reference/content/section_volume-drivers.html):

```bash
[DEFAULT]
enabled_backends = zfsISCIdriver-1

[zfsISCIdriver-1]
volume_backend_name = zfsISCIdriver-1
volume_driver = cinder.volume.drivers.zfssa.zfssaiscsi.ZFSSAISCSIDriver
san_ip = 10.10.10.10
san_login = cinder
san_password = password
zfssa_pool = mypool
zfssa_project = myproject
zfssa_initiator_username = iqn.name
zfssa_initiator_group = default
zfssa_target_portal = 10.10.10.11:3260
```
4.8.3 Configuring the Cinder NFS Backup Driver

The Cinder service provides the ability to back up Cinder volumes to an external storage. By default, the backup driver is configured to use NFS. However, you need to configure the NFS share as the location for the backups by setting a Kolla property.

**On the master node** run the following command:

```
$ kollacli property set cinder_backup_share host_name:path
```

where `host_name` is the fully qualified DNS name or IP address of the host that exports the NFS share, and `path` is the full path to the share on that host.

You can configure separate shares for individual hosts or groups, see Section 4.4, “Setting Properties for Groups or Hosts” for details.

By default, Cinder supports NFS version 4.1 or higher. If the NFS host uses an earlier version of NFS, this can cause errors. To configure Cinder to downgrade the NFS version, you can add configuration settings to the `/etc/kolla/config/cinder.conf` file on the **master node**. If this file does not exist, create it.

If the NFS host supports NFS version 4, add the following to the configuration file:

```conf
[DEFAULT]
backup_mount_options="vers=4,minorversion=0"
```

Otherwise, add the following to the configuration file:

```conf
[DEFAULT]
backup_mount_options="vers=3"
```

4.9 Setting up Swift

Swift provides object-based persistent storage for virtual machines. Swift is disabled by default. In order to use object-based persistent storage, you need to first create Swift rings, then enable Swift in the deployment. The steps in this procedure are to be performed on a master node.

**To set up Swift:**

1. Create and rebalance the Swift ring manually.

   You can use the following example configuration scripts as the basis for your Swift ring configuration.

   The scripts create the following required files in the `/etc/kolla/config/swift` directory on the master node:

   - `account.builder`
   - `account.ring.gz`
   - `container.builder`
   - `container.ring.gz`
   - `object.builder`
   - `object.ring.gz`
Example 4.1 Swift multinode ring configuration script

The following script is an example of setting up the Swift ring for a multinode deployment. The example uses a three-node storage array.

The `REGISTRY` and `TAG` variables use the Kolla properties that contain the location and port of your OpenStack Docker registry and the OpenStack release number.

Set the IP address for each storage node using the `node_IP` option in the `STORAGE` array at the start of the script, for example, `STORAGE[1]=10.0.0.10`.

```bash
#!/usr/bin/env bash

REGISTRY=registry_hostname:registry_port
TAG=openstack_release

STORAGE[1]=node_IP
STORAGE[2]=node_IP
STORAGE[3]=node_IP

# Note: In this example, each storage node is placed in its own zone

# Object ring
docker run --rm -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \ 
$(REGISTRY)/oracle/ol-openstack-swift-base:${TAG} \ 
swift-ring-builder /etc/kolla/config/swift/object.builder create 10 3 1
for NODE in 1 2 3; do
echo "object.builder: Adding ${STORAGE[$NODE]} to zone z${NODE}"
edocker run --rm -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \ 
$(REGISTRY)/oracle/ol-openstack-swift-base:${TAG} swift-ring-builder \ 
/etc/kolla/config/swift/object.builder add z${NODE}-${STORAGE[$NODE]}:6000/sdb1 1
done

# Account ring
docker run --rm -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \ 
$(REGISTRY)/oracle/ol-openstack-swift-base:${TAG} \ 
swift-ring-builder /etc/kolla/config/swift/account.builder create 10 3 1
for NODE in 1 2 3; do
echo "account.builder: Adding ${STORAGE[$NODE]} to zone z${NODE}"
edocker run --rm -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \ 
$(REGISTRY)/oracle/ol-openstack-swift-base:${TAG} swift-ring-builder \ 
/etc/kolla/config/swift/account.builder add z${NODE}-${STORAGE[$NODE]}:6001/sdb1 1
done

# Container ring
docker run --rm -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \ 
$(REGISTRY)/oracle/ol-openstack-swift-base:${TAG} \ 
swift-ring-builder /etc/kolla/config/swift/container.builder create 10 3 1
for NODE in 1 2 3; do
echo "container.builder: Adding ${STORAGE[$NODE]} to zone z${NODE}"
edocker run --rm -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \ 
$(REGISTRY)/oracle/ol-openstack-swift-base:${TAG} swift-ring-builder \ 
/etc/kolla/config/swift/container.builder add z${NODE}-${STORAGE[$NODE]}:6002/sdb1 1
done
```
for ring in object account container; do
docker run --rm \
 -v /etc/kolla/config/swift/:/etc/kolla/config/swift/ \
 $(REGISTRY)/oracle/ol-openstack-swift-base:${TAG} swift-ring-builder \
 /etc/kolla/config/swift/${ring}.builder rebalance
done

2. Enable the Swift service:

$ kollacli property set enable_swift yes

For more information on configuring Swift, see the OpenStack Kolla documentation at:

http://docs.openstack.org/developer/kolla/swift-readme.html

4.10 Setting up Ceilometer

Ceilometer is a telemetry service that collects, normalizes and transforms data produced by OpenStack services. Ceilometer is disabled by default.

Ceilometer requires a database to operate. Oracle recommends that you use a separate database for Ceilometer. The choice of database system, and how it is configured, is up to you. A database must be created and available on the network. Oracle recommends using MySQL for this purpose. For information on installing MySQL see:


To set up Ceilometer:

1. If you do not have a dedicated database for Ceilometer, create the database. For example, using MySQL you could use the following MySQL commands to create the database and user:

```sql
mysql> create database ceilometer;
mysql> grant all privileges on ceilometer.* to 'ceilometer'@'%' identified by 'password';
```

   In this example, the database is named `ceilometer`, with a user named `ceilometer`. These are the default settings for the deployment property settings.

2. Use the `kollacli property set` command to set the Ceilometer database deployment properties. These properties are used to create the Ceilometer database connection string.

   ```
   ceilometer_database_backend
   The database type, either `mysql` or `mongodb`. The default is `mysql`.
   ```

   ```
   ceilometer_database_address
   The fully qualified host name or IP address of the database host.
   
   If you do not have a dedicated database for Ceilometer, the default MySQL Cluster database on the first controller node should be used. In this case, set this value to be the host name or IP address of the first controller node.
   
   The default is the value of the `kolla_internal_address` property. If you leave it as the default, it is highly likely that Ceilometer will fail to collect all data and will not produce correct or expected results.
   ```

   ```
   ceilometer_database_port
   The database port, usually 3306 for MySQL or 27017 for MongoDB. The default is 3306.
   ```
**ceilometer_database_name**

The name of the Ceilometer database. The default is `ceilometer`.

**ceilometer_database_user**

The name of the Ceilometer database user. The default is `ceilometer`.

In this example, MySQL is used as the Ceilometer database. A MySQL database has been created named `ceilometer`, with a database user named `ceilometer`, and MySQL is available on port 3306 on `ceilometerdbhost.example.com`. As the database is set up using the defaults for the Ceilometer database deployment property settings, all that is required is to set the host on which the database is running:

```
$ kollacli property set ceilometer_database_address ceilometerdbhost.example.com
```

3. Add the password for the Ceilometer database user:

```
$ kollacli password set ceilometer_database_password
```

You are prompted to enter and confirm the password. The password value is not displayed on screen. You might want to use a strong password generator to generate the passwords.

4. Enable the Ceilometer service:

```
$ kollacli property set enable_ceilometer yes
```

### 4.11 Setting up Murano

The Murano project provides a method of creating and deploying cloud-ready applications. In order to use Murano applications, you must:

- Deploy the Murano and Heat services (both enabled by default).
- Create or download Murano applications.
- Create images which include the Murano Agent.
- Create a Murano environment which includes one or more Murano-enabled images and applications, and deploy the environment.

This section shows you how to create an image, ready to use with Murano.

#### 4.11.1 Creating Oracle Linux Murano-enabled Images

In order to deploy Murano applications to virtual machine instances, the Murano Agent must be installed in the virtual machine. This section shows how to install the Murano Agent in Oracle Linux images. To create an image which includes the Murano Agent you can:

- Create an Oracle Linux image, and install the openstack-murano-agent package.
- Download an Oracle Linux image for OpenStack from the Oracle Software Delivery Cloud, mount the image, and install the openstack-murano-agent package.

When you have created an image with the Murano Agent installed, you then load it into Glance with an additional image property. The image can then be used in a Murano deployment.
Installing the Murano Agent in a virtual machine:

If you choose to create a virtual machine, you can install the openstack-murano-agent package from the Oracle Yum Server. Log into the virtual machine and perform these actions:

1. Set up the yum repository location for the Murano Agent.

   For Oracle Linux 7, the openstack-murano-agent package is in the [ol7_openstack21] repository.
   For Oracle Linux 6, the openstack-murano-agent package is in the [ol6_openstack21] repository.

   Make sure this repository is enabled in the /etc/yum.repos.d/*_repo file. For ease, the steps to download this repository file are included.

   For Oracle Linux 7:

   ```
   # curl -L -o /etc/yum.repos.d/public-yum-ol7.repo 
   ```

   Make sure the [ol7_openstack21] and [ol7_addons] repositories are enabled.

   For Oracle Linux 6:

   ```
   # curl -L -o /etc/yum.repos.d/public-yum-ol6.repo 
   ```

   Make sure the [ol6_openstack21] and [ol6_addons] repositories are enabled.

2. Install the openstack-murano-agent package.

   ```
   # yum install -y openstack-murano-agent
   ```

3. Install the cloud-init package.

   ```
   # yum install -y cloud-init
   ```

Installing the Murano Agent in an image:

If you choose to use the Oracle Linux virtual machine images from the Oracle Software Delivery Cloud, this section contains instructions on installing the Murano Agent into these images.

The following should be performed by a user with root privileges. The final steps to load the image into Glance should be performed on a master node, so it may be easier to do all the steps from the master node.

1. Oracle Linux virtual machine images for OpenStack are available on the Oracle Software Delivery Cloud, for both Oracle Linux 7 and Oracle Linux 6. Download the Oracle Linux virtual machine image(s) for OpenStack from the Oracle Software Delivery Cloud at:

   https://edelivery.oracle.com/linux

2. Install the qemu-img package:

   ```
   # yum install qemu-img
   ```

3. Load the qemu ndb module:

   ```
   # modprobe nbd max_part=63
   ```

4. Share the virtual machine's disk:
Creating Oracle Linux Murano-enabled Images

5. Mount the virtual machine's disk. You can see a list of the partitions using:

   ```bash
   # fdisk /dev/nbd0 -l
   ```

   The Oracle Linux 6.7 and 7.2 image have two partitions: the boot partition, and the root partition. The system files are stored on the second (root) partition. The root partition (`/dev/nbd0p2`) is the mount point to access the file system and install packages.

   ```bash
   # mount /dev/nbd0p2 /mnt
   ```

6. Set the networking to resolve the host names required to do the package install.

   On the Oracle Linux 7.2 image:

   ```bash
   # cat /etc/resolv.conf > /mnt/root/etc/resolv.conf
   ```

   On the Oracle Linux 6.7 image:

   ```bash
   # cat /etc/resolv.conf > /mnt/etc/resolv.conf
   ```

7. Set up the yum repository location for the Murano Agent on the appropriate partition.

   For Oracle Linux 7, the `openstack-murano-agent` package is in the `[ol7_openstack21]` repository. For the Oracle Linux 7.2 image, use:

   ```bash
   ```

   For Oracle Linux 6, the `openstack-murano-agent` package is in the `[ol6_openstack21]` repository. For the Oracle Linux 6.7 image use:

   ```bash
   ```

   Make sure the repository is enabled in the `yum.repos.d/*.repo` file using a text editor such as `vi`.

8. Install the `openstack-murano-agent` package.

   For the Oracle Linux 7.2 image, use:

   ```bash
   # chroot /mnt/root/ /bin/bash -c "/usr/bin/yum install -y openstack-murano-agent"
   ```

   For the Oracle Linux 6.7 image use:

   ```bash
   # chroot /mnt/ /bin/bash -c "/usr/bin/yum install -y openstack-murano-agent"
   ```

   If you need to access the yum server using a proxy, set the `http_proxy` environment variable as well, for example:

   ```bash
   # chroot /mnt/ /bin/bash -c "http_proxy=http://proxy.example.com:80 /usr/bin/yum install -y openstack-murano-agent"
   ```

9. Clean up any cached packages to reduce the image size.

   For the Oracle Linux 7.2 image, use:

   ```bash
   # chroot /mnt/root/ /bin/bash -c "/usr/bin/yum clean all"
   ```

   For the Oracle Linux 6.7 image use:
10. Remove the Yum repository files.

For the Oracle Linux 7.2 image, use:

```
# rm /mnt/root/etc/yum.repos.d/public-yum-ol7.repo
```

For the Oracle Linux 6.7 image use:

```
# rm /mnt/etc/yum.repos.d/public-yum-ol6.repo
```

11. Remove the networking changes:

For the Oracle Linux 7.2 image, use:

```
# cat > /mnt/root/etc/resolv.conf << __EOF__
EOF
```

For the Oracle Linux 6.7 image use:

```
# cat > /mnt/etc/resolv.conf << __EOF__
EOF
```

12. Unmount the image.

```
# umount /mnt/
# qemu-nbd -d /dev/nbd0
```

### Loading a Murano-enabled Image into Glance

1. After installing the Murano Agent, the image is ready to load into Glance. To let OpenStack know that this image can be used in a Murano deployment, make sure you include the `--property` parameter as shown.

```
--property murano_image_info='{"type": "linux", "title": "My Image Title"}''
```

For example:

```
$ glance image-create --name OL7-murano --is-public True --disk-format qcow2 \ 
--container-format bare \ 
--property murano_image_info='{"type": "linux", "title": "OL7-murano"}'' \ 
--file /data/OracleLinux-7.2-x86_64.qcow2
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property 'murano_image_info'</td>
<td>{&quot;type&quot;: &quot;linux&quot;, &quot;title&quot;: &quot;OL7-murano&quot;}</td>
</tr>
<tr>
<td>checksum</td>
<td>0c487c29f3dfb08034dc98f25a3690</td>
</tr>
<tr>
<td>container_format</td>
<td>bare</td>
</tr>
<tr>
<td>created_at</td>
<td>2016-03-17T05:15:28.000000</td>
</tr>
<tr>
<td>deleted</td>
<td>False</td>
</tr>
<tr>
<td>deleted_at</td>
<td>None</td>
</tr>
<tr>
<td>disk_format</td>
<td>qcow2</td>
</tr>
<tr>
<td>id</td>
<td>2610deca-7b4e-4cf3-9aaa-8b4c0562a75e</td>
</tr>
<tr>
<td>is_public</td>
<td>True</td>
</tr>
<tr>
<td>min_disk</td>
<td>0</td>
</tr>
<tr>
<td>min_ram</td>
<td>0</td>
</tr>
<tr>
<td>name</td>
<td>OL7-murano</td>
</tr>
<tr>
<td>owner</td>
<td>41375e2937f04d7fa81b33f2ce134b2d</td>
</tr>
<tr>
<td>protected</td>
<td>False</td>
</tr>
<tr>
<td>size</td>
<td>431392792</td>
</tr>
<tr>
<td>status</td>
<td>active</td>
</tr>
<tr>
<td>updated_at</td>
<td>2016-03-17T05:15:31.000000</td>
</tr>
</tbody>
</table>
2. You can create a virtual machine instance using this image, as well as using the image in a Murano deployment. For example, to deploy a virtual machine using this image, you could use:

```
$ nova boot --image OL7-murano --flavor m1.medium --nic net-id=4a922c1b-25a0-44ea-a476-7626eb87738a OL7-murano-vm
```

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS-DCF:diskConfig</td>
<td>MANUAL</td>
</tr>
<tr>
<td>OS-EXT-AZ:availability_zone</td>
<td>nova</td>
</tr>
<tr>
<td>OS-EXT-SRV-ATTR:host</td>
<td>-</td>
</tr>
<tr>
<td>OS-EXT-SRV-ATTR: hypervisor_hostname</td>
<td>-</td>
</tr>
<tr>
<td>OS-EXT-SRV-ATTR: instance_name</td>
<td>instance-0000001d</td>
</tr>
<tr>
<td>OS-EXT-STS:power_state</td>
<td>0</td>
</tr>
<tr>
<td>OS-EXT-STS:task_state</td>
<td>scheduling</td>
</tr>
<tr>
<td>OS-EXT-STS:vm_state</td>
<td>building</td>
</tr>
<tr>
<td>OS-SRV-USG:launched_at</td>
<td>-</td>
</tr>
<tr>
<td>OS-SRV-USG:terminated_at</td>
<td>-</td>
</tr>
<tr>
<td>accessIPv4</td>
<td></td>
</tr>
<tr>
<td>accessIPv6</td>
<td></td>
</tr>
<tr>
<td>adminPass</td>
<td>FNxRi3PH8AGd</td>
</tr>
<tr>
<td>config_drive</td>
<td></td>
</tr>
<tr>
<td>created</td>
<td>2016-03-15T02:58:11Z</td>
</tr>
<tr>
<td>flavor</td>
<td>m1.medium (3)</td>
</tr>
<tr>
<td>hostId</td>
<td></td>
</tr>
<tr>
<td>id</td>
<td>02749598-6da4-440b-8ea1-d58e986d2a04</td>
</tr>
<tr>
<td>image</td>
<td>OL7-murano (da9ddf7d-fe7c-4b02-9a90-09300fd29b23)</td>
</tr>
<tr>
<td>key_name</td>
<td>-</td>
</tr>
<tr>
<td>metadata</td>
<td>{}</td>
</tr>
<tr>
<td>name</td>
<td>OL7-murano-vm</td>
</tr>
<tr>
<td>os-extended-volumes:volumes_attached</td>
<td>[]</td>
</tr>
<tr>
<td>progress</td>
<td>0</td>
</tr>
<tr>
<td>security_groups</td>
<td>default</td>
</tr>
<tr>
<td>status</td>
<td>BUILD</td>
</tr>
<tr>
<td>tenant_id</td>
<td>41375e2937f04d7fa81b33f2ce134b2d</td>
</tr>
<tr>
<td>updated</td>
<td>2016-03-15T02:58:10Z</td>
</tr>
<tr>
<td>user_id</td>
<td>cb96169f57ed4f69888ad749ed1f2efcb</td>
</tr>
</tbody>
</table>
Once you have set up your deployment, you use the kollacli command to deploy services to target nodes.

### 5.1 Performing a Deployment

Before you deploy OpenStack services, ensure that you have set up your deployment, as described in Chapter 4, *Setting up a Deployment*.

1. **Ensure that the Docker Engine is running on the target nodes.**

   To check that the Docker Engine is running:

   ```
   $ systemctl status docker.service
   ● docker.service - Docker Application Container Engine
   Loaded: loaded (/usr/lib/systemd/system/docker.service; enabled; vendor preset: disabled)
   Drop-In: /etc/systemd/system/docker.service.d
     └─docker-sysconfig.conf, var-lib-docker-mount.conf
   Active: inactive (dead) since Tue 2016-03-29 13:20:53 BST; 2min 35s ago
   ...
   ``

   If the output of this command shows the status of the Docker service to be *inactive (dead)*, start the Docker Engine:

   ```
   # systemctl start docker.service
   ``

2. **On the master node, log in as a user that is a member of the kolla group.**

   You add users to the kolla group as part of setting up a master node, see Section 3.6, *Preparing a Master Node*.

   You deploy OpenStack services with the kollacli command, and you must be a member of the kolla group to use this command.

3. **(Optional) Check that the target nodes are configured correctly and ready for the deployment.**

   ```
   $ kollacli host check all --predeploy
   ``

   This command is only suitable for an initial deployment. Do not use this command if OpenStack services have already been deployed.

   If you are adding nodes to a deployment and they are currently not running any OpenStack services, you can check the nodes individually.

   ```
   $ kollacli host check host --predeploy
   ``

   where *host* is the fully qualified domain name or IP address of the node.
Performing a Deployment

The checks verify:

- The openstack-kolla-preinstall package version and the Docker Engine version.
- The required ports are available.
- The Docker service is running and that the `libvirtd` service is not running.
- The LVM volume group named `cinder-volumes` exists on storage nodes that use the LVM driver for Cinder volumes (the default).
- The number of controller nodes (there must be a minimum of two).

If the check indicates a problem (you do not see a `Success` message), fix the problem before performing the deployment.

4. **Deploy OpenStack services to the target nodes.**

Use the `kollacli deploy` command to deploy services.

To deploy all enabled OpenStack services to all hosts:

```
$ kollacli deploy
```

Use this command to use to perform an initial deployment.

You can control the deployment by specifying the groups, services or hosts that you want to deploy:

- To deploy all enabled OpenStack services in the control group to all hosts in the control group:
  ```
  $ kollacli deploy --groups control
  ```

- To deploy OpenStack services to selected hosts, the services that are deployed are controlled by the groups the individual hosts are assigned to:
  ```
  $ kollacli deploy --hosts compute1.example.com,compute2.example.com
  ```

- To deploy selected services to all hosts in the control group:
  ```
  $ kollacli deploy --services mysqlcluster,neutron,keystone --groups control
  ```

- To deploy selected services, the groups the services belong to control the hosts that run the services:
  ```
  $ kollacli deploy --services heat,murano
  ```

You can also deploy services serially (one host at a time). In a high availability environment, this is useful for updating existing services without interrupting them.

- To deploy all enabled services in the network and storage groups serially to all hosts in the network and storage groups:
  ```
  $ kollacli deploy --serial --groups network,storage
  ```

- Deploying all services serially to selected hosts, the services that are deployed are controlled by the groups that the individual hosts are assigned to:
  ```
  $ kollacli deploy --serial --hosts compute1.example.com,compute2.example.com
  ```

If the deployment fails, see *Chapter 7, Troubleshooting Deployment.*
5.2 Making Configuration Changes After Deployment

If you make configuration changes to services after deployment, the configuration changes are not automatically applied to the Docker containers when you redeploy the services. The configuration changes are only applied when the Docker containers are restarted. However, the `kollacli` command does not currently support restarting individual containers.

To deploy configuration changes:

1. Remove all services from the nodes that run the changed services.
   
   Use the `kollacli host destroy` command, see Section 5.4, “Removing Services”. Do not use the `--includedata` option remove the OpenStack data.

2. Redeploy all services to the nodes.
   
   Use `kollacli deploy` command, see Section 5.1, “Performing a Deployment”.

For compute nodes that are currently running instances, you must move the running instances to another node before performing these steps.

5.3 Updating to a New Release of Oracle OpenStack for Oracle Linux

Perform the following steps to update a deployment to a new release of Oracle OpenStack for Oracle Linux.

1. Shut down all running instances in the deployment.

2. **All nodes**. Ensure you have a backup of the Docker file system mounted on `/var/lib/docker`.

3. **Compute nodes**. If you store ephemeral virtual machine data on the compute nodes, ensure you have a backup of the data mounted on `/var/lib/kolla`.

4. **All nodes**. Update the Oracle OpenStack for Oracle Linux preinstallation package and ensure the Docker Engine is configured correctly for the new release.
   
   See Section 3.4, “Preparing Oracle Linux Nodes” for details.

5. **Master node only**. Update the Oracle OpenStack for Oracle Linux CLI (`kollacli`).
   
   See Section 3.6, “Preparing a Master Node”.

6. **Docker registry only**. Import the Docker images for the new release to the Docker registry.
   
   See Section 3.7.2, “Importing Images into the Docker Registry”.

7. **Master node only**. Update the `openstack_release` property.

   By default, the property values set for the previous release are retained. You need to update the `openstack_release` property so that the latest OpenStack Kolla Docker images are pulled from the registry when you update.

   ```bash
   $ kollacli property set openstack_release 2.1.1
   ```

8. **Master node only**. Run the `kollacli upgrade` command.

   This command updates the OpenStack containers on all nodes.
By default, the data for the various OpenStack services are preserved when you update.

9. All network and compute nodes. Reboot the node.

For networking to function after the update, you must reboot these node types.

```
# systemctl reboot
```

## 5.4 Removing Services

You may want to remove the services from a node, for example because you want to replace the node or because you want to change the services it runs.

You remove services with `kollacli host destroy` command. This command removes all the Docker containers that run services. By default, it does **not** remove the data containers (and therefore the data on the host).

When you remove services, bear in mind the deployment “rules”:

- The nodes in the compute group must not be assigned to the control group.
- The control group must contain at least two nodes.
- The number of nodes in the database group must always be a multiple of two.
- Each group must contain at least two nodes to enable high availability.

To remove all services from a node:

```
$ kollacli host destroy compute1.example.com
```

By default, the containers are killed (`docker kill`) before they are removed. Use the `--stop` option to perform a graceful shutdown (`docker stop`) of the containers before removing them. This might be slower.

To remove all services from all nodes:

```
$ kollacli host destroy all
```

To remove all services and the OpenStack data from a node:

```
$ kollacli host destroy --includedata control1.example.com
```

If you no longer want to include a node in a deployment, you must also remove it from the list of target nodes:

```
$ kollacli host remove [ host | all ]
```

where `host` is the fully qualified domain name or IP address of the host, or `all` for all hosts.
Chapter 6 Using OpenStack CLIs

Table of Contents

6.1 Installing openstack-kolla-utils ................................................................. 69
6.2 Using docker-ostk ....................................................................................... 70
6.3 Accessing the File System with docker-ostk ............................................ 71

In order to access the OpenStack command line tools, such as nova, glance, and so on, a host must be
installed with the python-openstack_project client packages. Installing these packages may cause
dependency problems with Docker and OpenStack client package versions. To make the installation of the
OpenStack client tools easier, Oracle provides a single package for you which includes all the OpenStack
CLI tools. The package name is openstack-kolla-utils. This package is provided with pre-tested package
dependencies, and enables you to submit OpenStack CLI commands via Docker to an OpenStack service
in a container. This single package includes all the OpenStack CLI packages, and a utility named docker-ostk, which wraps the sometimes lengthy Docker CLI commands needed to access the OpenStack CLI.

Oracle suggests you install the openstack-kolla-utils package on a master node.

If you do not want to use the docker-ostk utility to access the deployed containers, you can instead
install and use the OpenStack client packages. For information on installing the OpenStack clients, see:
http://docs.openstack.org/cli-reference/common/cli_install_openstack_command_line_clients.html

6.1 Installing openstack-kolla-utils

This section shows you how to install and set up the openstack-kolla-utils package. This is the package
that wraps the OpenStack CLI commands using a utility named docker-ostk.

To install openstack-kolla-utils:

1. Set up an Oracle Linux host as described in Section 3.6, “Preparing a Master Node”.

2. Install the openstack-kolla-utils package:

```
# yum install openstack-kolla-utils
```

The docker-ostk utility is now installed.

3. The user which runs the docker-ostk utility must be a member of the docker group. To add an
existing user to the docker group, enter:

```
# usermod -aG docker username
```

The user must log out and in again for the group setting to take effect.

Caution

Any member of the docker group has full (root) access to all containers in
the deployment. If you add an unprivileged user to the docker group, you
should consider the elevation in security granted with this access. Do not run the
docker-ostk utility as the root user.

4. Set the environment variables to use with docker-ostk. You can manually set the variables, or
download a configuration file from Horizon.

The variables used by docker-ostk are listed in the following table:
Using docker-ostk

<table>
<thead>
<tr>
<th>Environment Variable</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS_AUTH_URL</td>
<td>Authentication URL.</td>
</tr>
<tr>
<td>OS_TENANT_NAME</td>
<td>Keystone tenant name.</td>
</tr>
<tr>
<td>OS_USERNAME</td>
<td>Keystone user name.</td>
</tr>
<tr>
<td>OS_PASSWORD</td>
<td>Keystone password.</td>
</tr>
<tr>
<td>OS_PROJECT_NAME</td>
<td>Keystone project name.</td>
</tr>
<tr>
<td>OS_PROJECT_DOMAIN_ID</td>
<td>Keystone domain ID containing the project.</td>
</tr>
<tr>
<td>OS_USER_DOMAIN_ID</td>
<td>Keystone user's domain ID.</td>
</tr>
<tr>
<td>ENV_FILE</td>
<td>The location and name of a file which contains key/value pairs.</td>
</tr>
<tr>
<td></td>
<td>Used to pass additional environment variables to docker-ostk.</td>
</tr>
<tr>
<td></td>
<td>Each key/value pair should be on a new line.</td>
</tr>
<tr>
<td>OPENSTACK_UTILS_IMAGE</td>
<td>Docker registry image name. The default is oracle/ol-openstack-utils:latest.</td>
</tr>
<tr>
<td></td>
<td>You should not need to change this.</td>
</tr>
</tbody>
</table>

The list of variables in this table is not a complete list. For more information on OpenStack Command Line variables, and a full list of variables you can use, see the OpenStack Keystone Command Line Utility documentation at:

http://docs.openstack.org/developer/python-keystoneclient/man/keystone.html

To manually set the environment variables, for each variable, enter:

```
$ export variable_name=variable_value
```

For example:

```
$ export OS_AUTH_URL=http://10.0.10:5000/v2.0
$ export OS_TENANT_NAME=admin
$ export OS_PROJECT_NAME=admin
$ export OS_USERNAME=admin
$ export OS_PASSWORD=password
```

To download a file which contains all the required environment variables:

a. Log into Horizon.

b. Select Project, select Compute, and then select Access & Security. Select the API Access subtab.

c. Click Download OpenStack RC File and save this file to the host on which the openstack-kolla-utils package is installed.

d. Source the file to load the environment variables:

```
$ source admin-openrc.sh
```

You are prompted to enter the OpenStack admin password. Enter the password and the environment variables are set up and ready for use with docker-ostk.

6.2 Using docker-ostk

To use the docker-ostk utility, use the syntax:
Accessing the File System with docker-ostk

```
docker-ostk openstack_client openstack_client_argument [ -h | --help ]
```

`openstack_client`

An OpenStack CLI client, for example, `nova`.

`openstack_client_argument`

The argument for the OpenStack CLI client command, for example, `list`.

`-h, --help`

Display syntax help.

The `docker-ostk` utility wraps `docker run` commands, and includes the environment variables you set. So instead of needing to issue a command like:

```
$ docker run --rm -it --env ADMIN_USER=admin --env ADMIN_PASS=password ... openstack-utils nova list
```

You can use the more straight-forward command of:

```
$ docker-ostk nova list
```

For a list of OpenStack CLI commands and their arguments, see the OpenStack Command-Line Interface Reference:

[http://docs.openstack.org/cli-reference/content/](http://docs.openstack.org/cli-reference/content/)

The first time you issue a `docker-ostk` command, the Docker registry is checked for a local copy of the ol-openstack-utils image. If a local copy is not available, a copy is downloaded to your local copy of the registry.

If the Docker registry cannot be found, make sure the registry location is set using the `kollacli property set docker_registry` command.

### 6.3 Accessing the File System with docker-ostk

To allow access to files on a container, the current working directory of the host running `docker-ostk` is bind-mounted into the container as `/data`. When you issue commands that require access to the container's file system, you must prefix the local file name with `/data/`. For example, if you have an image named `cirros-0.3.4-x86_64-disk.img` in the local working directory of the master node, you can use the following command to create an image using Glance:

```
$ docker-ostk glance image-create --name cirros --is-public True --disk-format qcow2 --container-format bare --file /data/cirros-0.3.4-x86_64-disk.img
```
This chapter contains some troubleshooting tips to help you with deployment.

### 7.1 Debugging a Deployment

If you experience errors when deploying services, use the following information to help debug the issue.

1. **(Optional)** Remove the services that are currently deployed on the target node(s).

   It is cleaner to remove services before attempting a redeployment. See Section 5.4, "Removing Services".

2. Redeploy services with debug output.

   The `kollacli` command has the following options which can be used for debugging purposes.

   **\(-v, --verbose\)**

   This option controls the verbosity of the output. You repeat the \(v\), to increase the verbosity, so \(-vv\) gives you more than \(-v\), and \(-vvv\) gives you more than \(-vv\).

   **\(--debug\)**

   This option enables the display of traceback on errors.

   **\(--log-file\)**

   This option enables you capture output to a log file, appending content if the file already exists.

To deploy with increased verbosity:

```
$ kollacli deploy -vv
```

To deploy with increased verbosity, with traceback on errors, and logging output to a file:

```
$ kollacli deploy -vvv --debug --log-file log-out.txt
```

If you are unable to resolve the issue, you might need to gather the log files (see Section 7.3, "Collecting the Log Files") and contact Oracle Support.

### 7.2 "Command Failed" Error When Starting the Ansible Container

When you perform a deployment, you might experience a `Command failed` error when starting the Ansible container, for example:

```
TASK [common : Starting Ansible container]
```
7.3 Collecting the Log Files

The log_collector.py script enables you to collect the Docker log files from the containers in your deployment. The script is included in the openstack-kollacli package, which is installed on master nodes, and is located in /usr/share/kolla/kollacli/tools. You collect the log files from all nodes, or from selected nodes, as follows:

    log_collector.py [all | host1[,host2,host3...]] [-h|--help]

If you specify a host, the host must have been added to the deployment inventory using the kollacli host add command. If a host is not accessible, the script ignores the host and processes the next host in the list.

The script only collects logs from Docker containers whose image names start with "ol-openstack".

The script collects the logs files into a tar archive in /tmp and prints the name of the file to the screen. The archive contains a directory for each specified deployment node, and each directory contains the Docker log files for the containers deployed on that node.

The script also runs the kollacli dump command, which retrieves the configuration files and logs, and includes the output of this command in the archive.
7.4 Removing the Docker Images From a Node

OpenStack services are deployed to target nodes as Docker containers. When you deploy services, the Docker daemon on the target node downloads the required Docker images from the Docker registry, and uses the images to create the containers.

You may want to remove all the images from a target node to ensure that fresh images are downloaded when you deploy services.

To remove all images, run the following command on the target node:

```
# docker rmi -f $(docker images -q)
```
### Appendix A kollacli Command-Line Reference

```
kollacli [ subcommand ]
[ --version ] [ -v | --verbose ] [ --log-file file_name ] [ -q | --quiet ]
[ -h | --help ] [ --debug ]
```

Starts the CLI shell, or runs CLI commands from the operating system prompt.

**subcommand**

A `kollacli` subcommand:

<table>
<thead>
<tr>
<th>Subcommand</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>complete</td>
<td>Print a Bash completion function.</td>
</tr>
<tr>
<td>deploy</td>
<td>Deploy services.</td>
</tr>
<tr>
<td>dump</td>
<td>Create a tar file of the configuration files and logs for debugging.</td>
</tr>
<tr>
<td>group add</td>
<td>Add a group.</td>
</tr>
<tr>
<td>group addhost</td>
<td>Add a host to a group.</td>
</tr>
<tr>
<td>group listhosts</td>
<td>List all groups and the hosts in each group.</td>
</tr>
<tr>
<td>group listservices</td>
<td>List all groups and the services in each group.</td>
</tr>
<tr>
<td>group remove</td>
<td>Remove a group.</td>
</tr>
<tr>
<td>group removehost</td>
<td>Remove a host from a group.</td>
</tr>
<tr>
<td>help</td>
<td>Display help on using CLI commands.</td>
</tr>
<tr>
<td>host add</td>
<td>Add a host to a deployment.</td>
</tr>
<tr>
<td>host check</td>
<td>Check host configuration.</td>
</tr>
<tr>
<td>host destroy</td>
<td>Remove services from host.</td>
</tr>
<tr>
<td>host list</td>
<td>List hosts and the groups they belong to.</td>
</tr>
<tr>
<td>host remove</td>
<td>Remove hosts from a deployment.</td>
</tr>
<tr>
<td>host setup</td>
<td>Set up a host.</td>
</tr>
<tr>
<td>password clear</td>
<td>Remove a password name and its value.</td>
</tr>
<tr>
<td>password list</td>
<td>List all password names.</td>
</tr>
<tr>
<td>password set</td>
<td>Set a password value.</td>
</tr>
<tr>
<td>property clear</td>
<td>Clear a property.</td>
</tr>
<tr>
<td>property list</td>
<td>List properties and their associated values.</td>
</tr>
<tr>
<td>property set</td>
<td>Set a property.</td>
</tr>
<tr>
<td>service addgroup</td>
<td>Add a service to a group.</td>
</tr>
<tr>
<td>service list</td>
<td>List all services and their associated subservices.</td>
</tr>
<tr>
<td>service listgroups</td>
<td>List all services and the groups they belong to.</td>
</tr>
<tr>
<td>service removegroup</td>
<td>Remove a service from a group.</td>
</tr>
<tr>
<td>upgrade</td>
<td>Upgrade a deployment.</td>
</tr>
</tbody>
</table>

**--version**

Display the CLI version number.
-v, --verbose

Set the verbosity of the output when running a command. Repeat -v, to increase the verbosity, so -vv gives you more than -v, and -vvv gives you more than -vv.

--log-file

Set the file name to log debug and error output. This is disabled by default. The output is in verbose format. If the log file exists, the results are appended to the existing file content.

-q, --quiet

Suppress all output, except warnings and errors.

-h, --help

Display CLI syntax help.

--debug

Display traceback on errors.

**kollacli complete**

```
kollacli complete [ --name function_name ] [ --shell bash | none ]
```

Print a Bash completion function.

--name

The name of the function to generate. The default function name is *kollacli*.

--shell

The shell type to use for the script. Use *none* for data only. The default is *bash*.

**kollacli deploy**

```
kollacli deploy
[ --serial ] [ --services service_list ]
[ --groups group_list | --hosts host_list ]
```

Deploy services.

--serial

Deploy services serially (one host at a time).

--services

A comma-separated list of services.

--groups

A comma-separated list of groups.

--hosts

A comma-separated list of hosts.
**kollacli dump**

Create a tar file of the configuration files and logs for debugging.

**kollacli group add**

Add a group.

`group_name`

The name of the group.

**kollacli group addhost**

Add a host to a group.

`group_name`

The name of the group.

`host`

The fully qualified domain name or IP address of the host.

**kollacli group listhosts**

List all groups and the hosts in each group.

**kollacli group listservices**

List all groups and the services in each group.

**kollacli group remove**

Remove a group.

`group_name`

The name of the group.
**kollacli group removehost**

```
kollacli group removehost group_name host
```

Remove a host from a group.

- **group_name**
  - The name of the group.

- **host**
  - The fully qualified domain name or IP address of the host.

**kollacli help**

```
kollacli help [ command ]
```

Display help on using CLI commands.

- **command**
  - Help on a specific `kollacli` command.

**kollacli host add**

```
kollacli host add host
```

Add a host to a deployment.

- **host**
  - The fully qualified domain name or IP address of the host.

**kollacli host check**

```
kollacli host check { host | all } [ --predeploy ]
```

Check host configuration.

- **host | all**
  - The fully qualified domain name or IP address of the host, or `all` for all hosts.

- **--predeploy**
  - Perform predeployment check. Only use this option if no services are deployed to the host.

**kollacli host destroy**

```
kollacli host destroy { host | all } [ --stop ] [ --includedata ]
```

Remove services from host.

- **host | all**
  - The fully qualified domain name or IP address of the host, or `all` for all hosts.
--stop

Perform a graceful shutdown (docker stop) of the containers before removing them.

--includedata

Remove the OpenStack data.

**kollacli host list**

```bash
kollacli host list [ host ]
[ { -f | --format } csv | html | json | table | value | yaml ]
[ { -c | --column } column_name ] [ --max-width max_width ]
[ --quote all | minimal | none | nonnumeric ]
```

List hosts and the groups they belong to.

**host**

The fully qualified domain name or IP address of the host. If no `host` is provided, all hosts are listed.

**kollacli host remove**

```bash
kollacli host remove [ host | all ]
```

Remove hosts from a deployment.

**host | all**

The fully qualified domain name or IP address of the host, or `all` for all hosts.

**kollacli host setup**

```bash
kollacli host setup { host | { -f | --file } file_name }
```

Set up a host.

**host**

The fully qualified domain name or IP address of the host.

**-f | --file**

Absolute path of a YAML format file that contains the hosts to set up and the credentials to use.

**kollacli password clear**

```bash
kollacli password clear password_name
```

Remove a password name and its value.

**password_name**

The name of the password.

**kollacli password list**

```bash
kollacli password list
```
List all password names.

**kollacli password set**

```
kollacli password set password_name
```

Set a password value.

```
password_name
```

The name of the password.

**kollacli property clear**

```
kollacli property clear [ --groups group_list | --hosts host_list ] property_name
```

Clear a property.

```
property_name
```

The name of the property.

```
--groups
```

A comma-separated list of groups.

```
--hosts
```

A comma-separated list of hosts.

**kollacli property list**

```
kollacli property list
{ { -f | --format } csv | html | json | table | value | yaml }
{ { -c | --column } column_name } [ --max-width max_width ]
[ --quote all | minimal | none | nonnumeric ]
[ --all ] [ --long ]
[ --groups group_list | --hosts host_list ]
```

List properties and their associated values.

```
--all
```

List all properties and their values, regardless of their length.

```
--long
```

List properties, their current values, and their default values, if overridden by a global property.

```
--groups
```

A comma-separated list of groups.

```
--hosts
```

A comma-separated list of hosts.

**kollacli property set**

```
```

Set a property.
### kollacli service addgroup

Add a service to a group.

**service_name**

The name of the service.

**group_name**

The name of the group.

### kollacli service list

List all services and their associated subservices.

### kollacli service listgroups

List all services and the groups they belong to.

### kollacli service removegroup

Remove a service from a group.

**service_name**

The name of the service.
kollacli upgrade

*group_name*

The name of the group.

**kollacli upgrade**

Upgrade a deployment.