Cisco OpenStack Installer Deployment Guide,Icehouse Release

First Published: May 21, 2014

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

CONTENTS

Preface

Audience

This guide is intended primarily for experienced data center and/or network administrators who want to use Cisco OpenStack Installer (Cisco OSI) to deploy an OpenStack cloud.

Document Conventions

This document uses the following conventions:

Convention	Description
^ or Ctrl	Both the ^ symbol and Ctrl represent the Control (Ctrl) key on a keyboard. For example, the key combination **^D** or **Ctrl-D** means that you hold down the Control key while you press the D key. (Keys are indicated in capital letters but are not case sensitive.)
bold font	Commands and keywords and user-entered text appear in **bold** font.
Italic font	Document titles, new or emphasized terms, and arguments for which you supply values are in *italic* font.
`Courier font`	Terminal sessions and information the system displays appear in `courier` font.
`Bold Courier font`	**`Bold Courier`** font indicates text that the user must enter.
[x]	Elements in square brackets are optional.

Convention	Description
...	An ellipsis (three consecutive nonbolded periods without spaces) after a syntax element indicates that the element can be repeated.
|	A vertical line, called a pipe, indicates a choice within a set of keywords or arguments.
[x | y]	Optional alternative keywords are grouped in brackets and separated by vertical bars.
{x | y}	Required alternative keywords are grouped in braces and separated by vertical bars.
[x {y | z}]	Nested set of square brackets or braces indicate optional or required choices within optional or required elements. Braces and a vertical bar within square brackets indicate a required choice within an optional element.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.
< >	Nonprinting characters such as passwords are in angle brackets.
[ ]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.

Reader Alert Conventions

This document uses the following conventions for reader alerts:

Note Means *reader take note*. Notes contain helpful suggestions or references to material not covered in the manual.

Tip Means *the following information will help you solve a problem.*

Caution Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.

Timesaver Means *the described action saves time.* You can save time by performing the action described in the paragraph.

Related Documentation

Best-Effort Support

As a community project, Cisco OpenStack Installer is supported on a best-effort basis by Cisco engineers. Our best-effort support mailer is openstack-support@cisco.com.

Cisco OpenStack Installer Resources

Information about installing, configuring, and maintaining a Cisco OpenStack Installer cloud is available at these locations:

- The latest information about Cisco OpenStack Installer can be found at the Cisco OpenStack documentation wiki.
- Release information can be found in Launchpad at https://launchpad.net/openstack-cisco/.

  Click on a release in the "Series and milestones" section to find information about individual releases, including projected release dates, targeted blueprints and bug lists, release notes, and change logs.
- The Cisco OSI bug tracker is in Launchpad at https://bugs.launchpad.net/openstack-cisco.

  Please contribute bug reports if your run into problems. You can find a guide to filing bug reports here.
- The data model approach to the Cisco OpenStack Installer Icehouse release is documented on GitHub in these two articles:
  - Configuring OpenStack as Data
  - Scenario Based Override Terminus
- Detailed instructions for installing Ubuntu Server are on the Cisco OpenStack Installer documentation GitHub repository at https://github.com/CiscoSystems/coi-docs/tree/icehouse/install_guides.

Other Resources

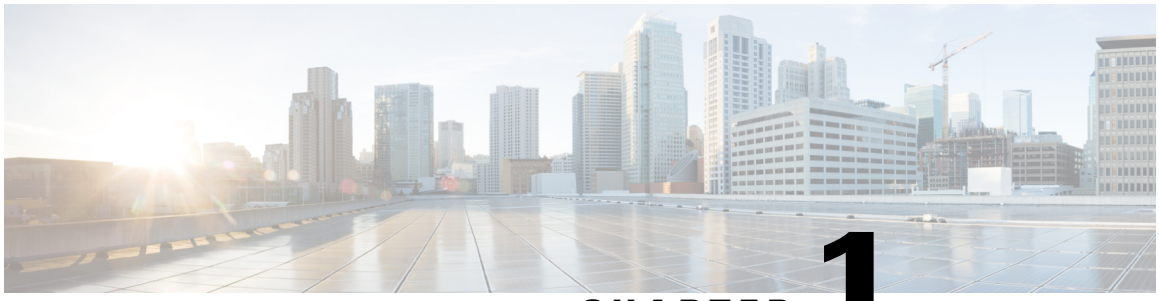
Information about the upstream community OpenStack project and Puppet install tools is available on the web:

- The community OpenStack project documentation page is at OpenStack documentation.
- For general OpenStack questions, you may also want to check http://ask.openstack.org and the OpenStack Wiki.

- Information about configuring and using Puppet can be found on the Puppet Labs website at Puppet Labs.
- Information about using Hiera can be found on the Puppet Labs website at Hiera documentation.

Finally, you can ask for help in the OpenStack community at large. Some of the most popular forums are:

- The OpenStack mailing lists at https://wiki.openstack.org/wiki/Mailing_Lists.
- The OpenStack IRC channels at https://wiki.openstack.org/wiki/IRC. There is a puppet-openstack channel on freenode as well.
- OpenStack user groups all over the world. Find them at https://wiki.openstack.org/wiki/OpenStack_User_Groups.
- The puppet-openstack Google Group at https://groups.google.com/a/puppetlabs.com/forum/#!forum/puppet-openstack .



CHAPTER 1

Overview

About the Cisco OpenStack Installer Project

Cisco OpenStack Installer (Cisco OSI) is Cisco's reference implementation of OpenStack, provided by Cisco free of charge and as open source software for the community. This guide documents the Cisco OSI Icehouse release.

Release Schedule and Policies

The release schedule of Cisco OSI parallels the community release. Where possible, Cisco OSI provides unmodified OpenStack code. Every new release of Cisco OSI follows the latest community stable release; however, in some cases Cisco might provide more recent patches that have been accepted into the OpenStack stable branches, but have not yet become part of an OpenStack stable release.

The Cisco OSI code update policy is to contribute code upstream to the OpenStack project and absorb patches into Cisco OpenStack Installer after they have been accepted upstream. Cisco deviates from this policy only when patches are unlikely to be reviewed and accepted upstream in time for a release or for a customer deadline (in such cases Cisco applies the patches to the repositories, submits them upstream, and replaces the local change with the upstream version when it is becomes accepted). Cisco also uses and contributes to modules from other upstream sources including Puppet Labs on StackForge.

Icehouse Release of Cisco OSI

The Icehouse release of Cisco OpenStack Installer continues the model-based approach introduced in the Havana release. Configuration parameters are edited in .yaml files and the configuration is applied using Puppet.

The Icehouse release contains several new features not present in the Havana release, including improved support for Modular Layer 2 (ML2) and using a new version of Ubuntu Linux (14.04 LTR, Trusty) as its base operating system. See the release notes for details of enhancements and bug fixes.

Puppet and Cobbler Installation Automation Tools

Cisco OSI uses Puppet to install and configure OpenStack components and Hiera to model the deployment configuration. Cisco's core OpenStack Puppet modules are the community OpenStack core modules available at the time of release on StackForge, where Cisco actively contributes code and reviews (see About the Cisco OpenStack Installer Project, on page 1).

Cisco OpenStack Installer installs and sets up Cobbler (for more information, see Cobbler) in order to provide provisioning of physical servers in the OpenStack cloud.

OpenStack Components

Supported Components

Cisco OpenStack Installer uses standard OpenStack components for most functions.

Cisco OSI deploys Nova compute nodes. By default, it uses Neutron for network services. Cisco OSI also installs the Keystone identity service, the Horizon OpenStack console, and, optionally, Heat for orchestration. You can also use Ceilometer for telemetry.

For object storage, Cisco OSI deploys **Swift**. **Ceph** provides block storage and is deployed as a backend for Glance. Ceph can also serve as a backend for Cinder.

In some deployment scenarios, you can choose to have Cisco OSI deploy your OpenStack cloud with active/active high availability for most core functions and other important components.

Note A reliable active/active highly available multiple-writer solution is not currently available for the database backends required by OpenStack. Galera, the only supported active/active database clustering solution, deadlocks when run with some OpenStack services, including Neutron and Nova. See the release notes for details about this known issue.

When deploying the high availability reference architecture, Cisco OSI deploys additional components such as MySQL, WSREP and Galera, HAProxy, and Keepalived. You can also set up messaging with the Advanced Message Queuing Protocol (AMQP) using RabbitMQ Clustering and Mirrored Queues.

In order to provide a system that can be managed after it is installed, Cisco OSI provides open source monitoring tools as a reference monitoring system. These include Collectd and Graphite. Each tool provides simple health monitoring or trending information on the physical nodes and important software services in the OpenStack cloud.

Unsupported Components

Cisco OSI does not support OpenStack incubator projects that have not been added to the core products.

Cisco OSI currently does not support Trove, even though it has been added as a core OpenStack product.

Component Versions

The package inventories for each release show the version of each component in the release.

The Cisco OSI is supplied in two repositories:

- The main repository, which contains the core OpenStack packages:
  - 64 bit (amd64): http://openstack-repo.cisco.com/openstack/cisco/dists/icehouse/snapshots/i.0/main/binary-amd64/Packages.
  - 32 bit (i386): http://openstack-repo.cisco.com/openstack/cisco/dists/icehouse/snapshots/i.0/main/binary-i386/Packages.
- The puppet repository, which contains packages used by the Puppet installer.
  - 64 bit (amd64): http://openstack-repo.cisco.com/openstack/puppet/dists/icehouse/snapshots/i.0/main/binary-amd64/Packages.
  - 32 bit (i386): http://openstack-repo.cisco.com/openstack/puppet/dists/icehouse/snapshots/i.0/main/binary-i386/Packages.

OpenStack Deployment

Deployment Scenarios

A *scenario* is a collection of roles that perform different tasks within an OpenStack cluster. For example, if you want an environment where a single node runs all services, the scenario "all_in_one" would be suitable. If you want separate control and compute nodes, the "2_role" scenario is the smallest possible scenario (in terms of number of distinct nodes). See Choosing a Deployment Scenario, on page 11 for descriptions of scenarios that are included with the Icehouse release.

Build Node

When you deploy OpenStack with Cisco OSI, you configure a *build node* (or *build server*) outside the OpenStack cluster to manage and automate the OpenStack software deployment. The build server is not a part of the OpenStack installation. After the build server is installed and configured, you use it as an out-of-band automation and management workstation to bring up, control, and reconfigure (if necessary) the nodes of the OpenStack cluster. The build node serves the following roles:

- A Puppet Master server that deploys the software onto and manages the configuration of the OpenStack cluster. For more information about Puppet, see the Puppet Labs documentation.
- A repository for a model of your installation scenario using the Scenario Node Terminus Puppet module and Puppet's Hiera tool to decouple node-specific data from your class descriptions.
- A Cobbler installation server to manage the Pre-boot Execution Environment (PXE) boot that is used for rapid bootstrapping of the OpenStack cluster. For more information about Cobbler, see the Cobbler user documentation.

- A monitoring server to collect statistics about the health and performance of the OpenStack cluster and to monitor the availability of the servers and services of the OpenStack cluster.
- A cache for installing components to Puppet client nodes.

YAML Files and the Hiera Database

To set up and customize your Cisco OSI deployment, you will modify several YAML files in the Hiera database that controls the configuration of the deployment model. This section describes how those files define the model.

The deployment model is mostly defined in the following files:

- `/etc/puppet/hiera.yaml`

  This file specifies the order in which files are read for configuration information.

  Note You can refer to this file to confirm that correct configuration parameters have been assigned and not preempted in the file hierarchy. You do not typically need to modify this file.

  The `:yaml:` section identifies the root directory for the configuration files.

  The `:hierarchy:` section orders the files that are searched for configuration assignments. The files listed are searched first-to-last; the first occurrence found is used. The first several entries should resemble this example:

```
:hierarchy:
  - "hostname/%{hostname}"
  - "client/%{clientcert}"
  - user
  - jenkins
  - vendor/cisco_coi_user.%{scenario}
  - user.%{scenario}
  - vendor/cisco_coi_user.common
  - user.common
  ...
```

- `/etc/puppet/data/hiera_data/*.yaml`

  These files contain the model information. They are searched in the order described in the previous bullet. You configure your model by adding and modifying information in these files.

  Note Most of the changes you make will be in one of two files: `user.common.yaml` and `user.scenario.yaml`, where "scenario" is the deployment scenario that you are using.

- `/etc/puppet/data/role_mappings.yaml`

  This file defines roles for each node in your deployment. It is populated with default values for the supplied scenarios. You will need to modify this file to specify which role is used by each node in your deployment.

Facter Variables

Puppet uses the **Facter** tool to collect system information into a set of variables called Puppet *facts*. These variables have the form *%{fact_name}*. For example, a configuration might capture the eth1 IP address in the variable *%{ipaddress_eth1}* . See *Learning Puppet—Variables, Conditionals, and Facts* here.

Cobbler Configuration

Cobbler is used to provision baremetal nodes. Each node that you want to provision using Cobbler must be individually defined in `/etc/puppet/data/cobbler/cobbler.yaml`. Provisioning using Cobbler is optional in Cisco OSI. You can instead provision a machine yourself and deploy an OpenStack node by installing a Puppet Agent. See Building the Control and Compute Nodes Individually With Puppet Agent, on page 27.

IP Networks

Cisco OSI configures one or more of the following interface types depending on how you configure the deployment:

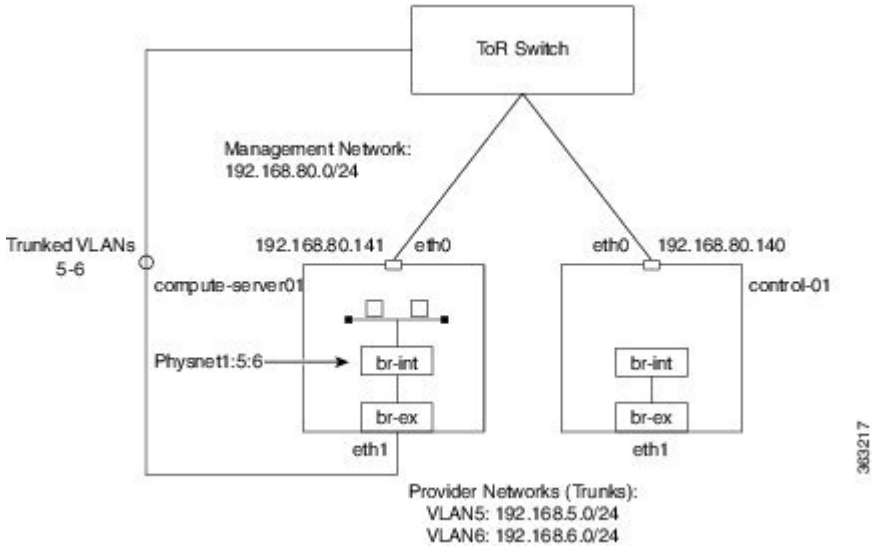
- A public interface that is reachable by all other OpenStack nodes—Used for API access, Horizon/VNC console access, and as a (GRE) tunnel endpoint. You can also optionally use the public interface for management tasks such as monitoring and Preboot Execution Environment (PXE) booting the baremetal node.
- (Optional) A separate management interface—Used for management tasks if you do not want to put the tasks on the public interface.
- An external interface attached to an upstream device that provides Layer 3 routing—Used to provide an uplink for the Layer 3 agent's external router (in a GRE-based tenant network) and by floating IP addresses (in a provider mode network).
- Private interfaces—Used for traffic between tenants or virtual machines (VMs).

Cisco OSI supports the installation of two different models for traffic between VMs and external networks.

The first model is a provider network (also called a VLAN model in Cisco documentation). In this model, each VM connects through a bridge to the compute node's external-facing adapter. Each VM thus has an IP address associated with the underlying physical network, so that VMs are connected directly to the network

that routes the external traffic. The following figure illustrates a control node and a compute node in a provider network configuration.

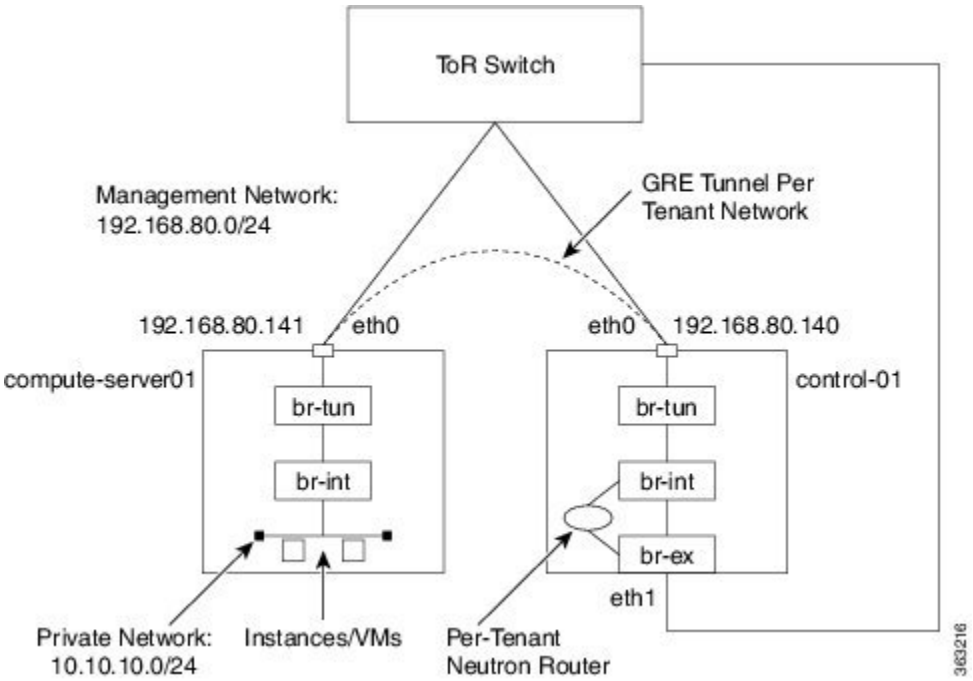
Figure 1: Provider Network Example

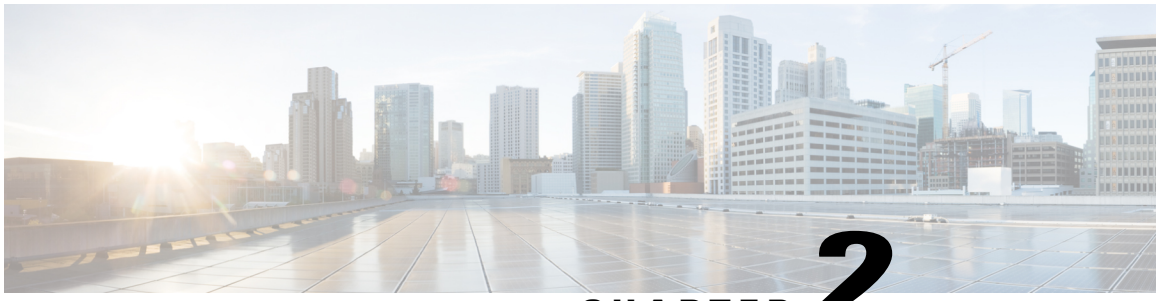


The second model is the tenant network or GRE model. In this model, VMs have direct access only to an internal virtual network. The Neutron controller provides external access by acting as a Network Address Translation (NAT) router, providing Port Address Translation (PAT) for outbound traffic and Floating IP

addresses for inbound traffic. The following figure illustrates a control node and a compute node in a GRE network configuration.

Figure 2: GRE Network Example





CHAPTER 2

Preparing to Install OpenStack

System Requirements for Cisco OpenStack Installer

Supported Hardware and Software

Cisco OSI has been tested against a system that includes Cisco UCS servers. Systems with nonsupported servers might encounter issues.

Cisco OpenStack Installer has been tested on the following infrastructure:

- Cisco UCS C-Series and B-Series Servers serve as physical compute and storage hardware.
- Cisco switches provide physical networking.
- Ubuntu 14.04 LTS (64-bit server edition) serves as a base operating system.
- KVM serves as the hypervisor.
- OpenStack Neutron provides the network services for the OpenStack cloud. You can select a variety of Neutron setup options, including support for OVS in GRE tunneling mode, OVS in VLAN mode, the Cisco Nexus plugin, and provider networks.

Recommended Release Levels

The following release levels are recommended for any server that is part of an OpenStack cluster:

- For blade servers and integrated rack-mount servers, Cisco UCS Manager, Release 2.1(1) and later releases
- For standalone rack-mount servers, Cisco Integrated Management Controller, Release 1.5 and later releases
- If Cobbler is used for bare-metal Linux installs, Cobbler 2.4 is recommended.

Proxy Configurations

If your network uses proxies, they must be configured properly in order to download the packages used by the installer.

How to configure your proxy is discussed in Creating the Build Node, on page 18.

Minimum Server Requirements

The following table lists the minimum requirements for the Cisco UCS servers that you use for the nodes in your OpenStack cluster:

Server/Node	Recommended Hardware	Notes
Build node	Processor: 64-bit x86 Server or VM with: • Memory: 4 GB (RAM) • Disk space: 20 GB	The build node must also have Internet connectivity to be able to download Cisco OSI modules and Puppet manifests. To ensure that the build node can build and communicate with the other nodes in your cluster, it must also have a network interface on the same network as the management interfaces of the other OpenStack cluster servers. A minimal build node (for example, a VM with 4 GB of RAM and a 20-GB disk) is sufficient for a test install. However, because the build node acts as the puppet master, caches client components, and logs all installation activity you might need a more powerful machine with more disk space for larger installs.
Control node	Processor: 64-bit x86 Memory: 12 GB (RAM) Disk space: 1 TB (SATA or SAS) Network: Two 1-Gbps network interface cards (NICs)	A quad core server with 12 GB of RAM is sufficient for a minimal control node. These are minimum requirements. Memory, disk, and interface speed will be greater for larger clusters.
Compute node	Processor: 64-bit x86 Memory: 128 GB (RAM) Disk space: 300 GB (SATA) Volume storage: Two disks with 2 TB (SATA) for volumes attached to the compute nodes Network: Two 1-Gbps NICs	These are minimum requirements. Memory, disk, and interface speed will be greater for larger clusters.

Server/Node	Recommended Hardware	Notes
HA proxy (load balance) node	Processor: 64-bit x86 Memory: 12 GB (RAM) Disk space: 20 GB (SATA or SAS) Network: One 1-Gbps NIC	These are minimum requirements. Memory, disk, and interface speed will be greater for larger clusters.
Swift storage proxy node	Processor: 64-bit x86 Memory: 12 GB (RAM) Disk space: 300 GB (SATA or SAS) Network: Two 1-Gbps NICs	These are minimum requirements. Memory, disk, and interface speed will be greater for larger clusters.
Swift storage node	Processor: 64-bit x86 Memory: 32 GB (RAM) Disk space: 300 GB (SATA) Volume storage: • For rack-mount servers, either 24 disks with 1 TB (SATA) or 12 disks with 3 TB (SATA) depending upon the model • For blade servers, two disks with 1 TB (SATA) for combined base OS and storage Network: Two 1-Gbps NICs	Three or more storage nodes are needed. These are minimum requirements. Memory, disk, and interface speed will be greater for larger clusters.

Choosing a Deployment Scenario

The following table lists the deployment scenarios that are currently available with Cisco OSI:

Scenario Name	Node Count	Roles	Description	Typical Use Case
all_in_one	1	all_in_one, compute	A single node that combines the control services and compute services with the build server. Optionally, you can add more compute-only nodes.	Evaluating OpenStack without tying up a lot of hardware. Providing a fully functional (though not scalable or redundant) OpenStack cloud with minimal resources.

Scenario Name	Node Count	Roles	Description	Typical Use Case
2_role	3	build, control, compute, swift_proxy (optional), swift_storage (optional)	Separate nodes (two in addition to the build node) for control and compute. Optionally, one or more nodes for swift storage services. High availability is not possible in this scenario.	Evaluating a multi-node installation with services separated to different machines without the added complexity of messaging, HA, and other production features.
3_role	4	build, control, compute, network_control, swift_proxy (optional), swift_storage (optional)	Same as 2_role, but adds a separate node for network control services that are handled by the control node in 2_role. Optionally, one or more nodes for Swift storage services. High availability is not possible in this scenario.	Evaluating a simplified multi-node installation similar to a 2-role scenario, but with separate networking services.
full_ha	14	build, control, compute, swift_proxy, swift_storage, load_balancer	Similar to the 2_role scenario, but includes a load balancer for high-availability deployment.	Deploying production environments that provide separation of services including dedicated load balancing nodes, storage nodes, compute nodes, and an active/passive highly available control plane.
compressed_ha	4	build, compressed_ha, compressed_ha_cephall, compressed_ha_cephosd, compressed_ha_cephmon	A high-availability deployment on three nodes with each node serving all functions. If Ceph is not used, the compressed_ha_cephall, compressed_ha_cephosd and compressed_ha_cephmon roles can be omitted.	Deploying a highly available control plane with a limited number of nodes and limited scalability. (See the note below regarding the nature of the high-availability solution.)

Note The node count includes the build node.

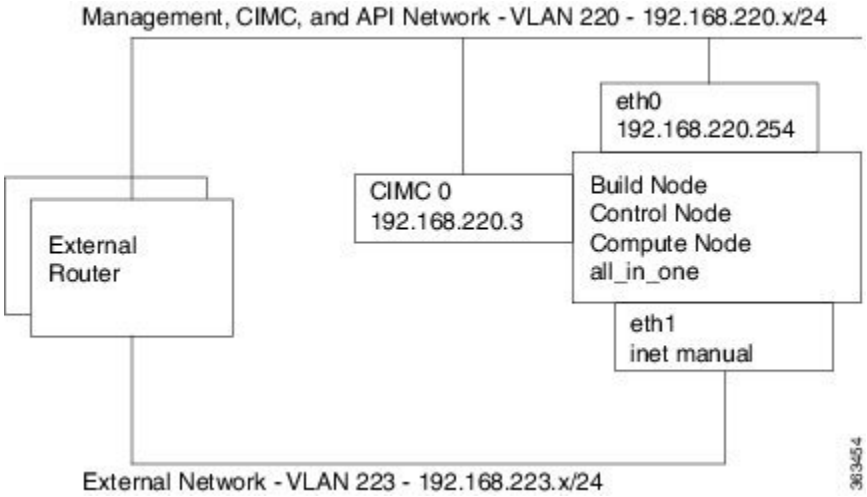
Note Roles refers to the roles defined in the data model. Each role defines a package of resources required to function in that role.

Note A reliable active/active highly available multiple-writer solution is not currently available for the database backends required by OpenStack. Galera, the only supported active/active database clustering solution, deadlocks when run with some OpenStack services, including Neutron and Nova. See the release notes for details about this known issue.

The deployment scenarios are illustrated below in order of increasing complexity.

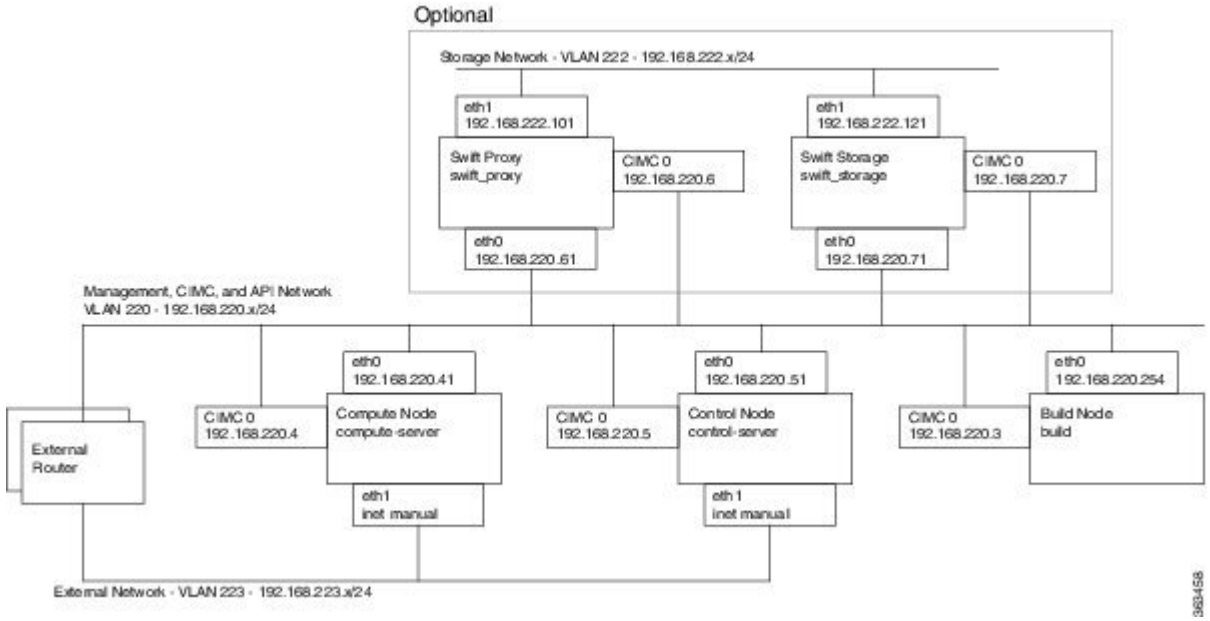
The following figure illustrates the all-in-one scenario.

Figure 3: All-In-One Scenario



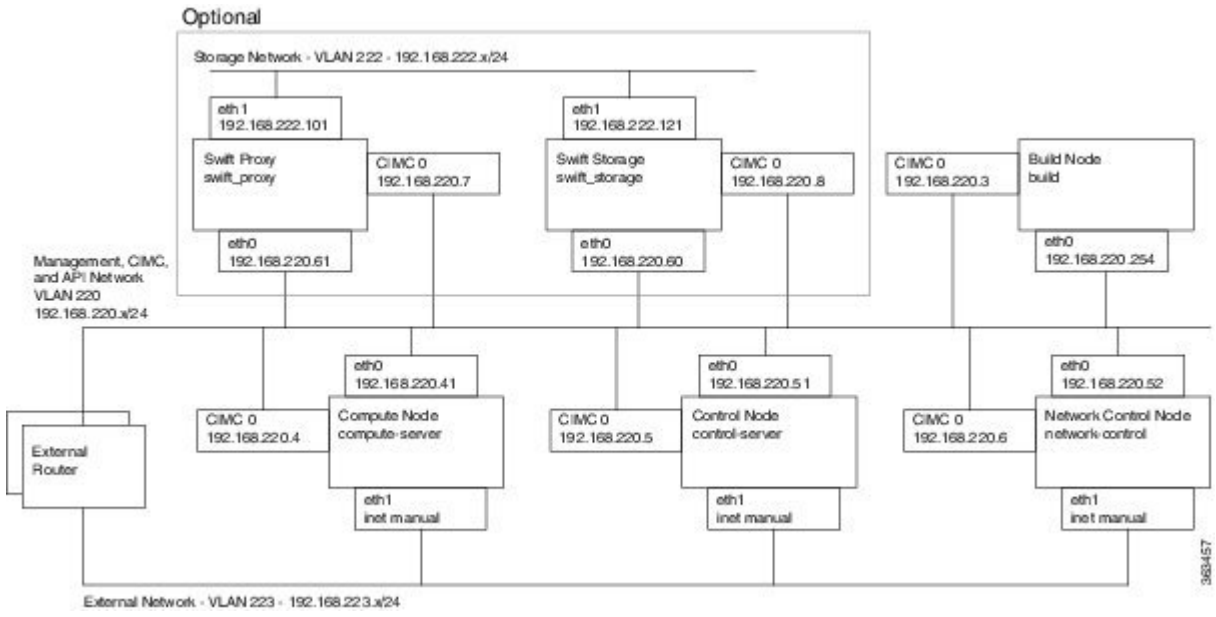
The following figure shows a two-role scenario.

Figure 4: Two-Role Scenario



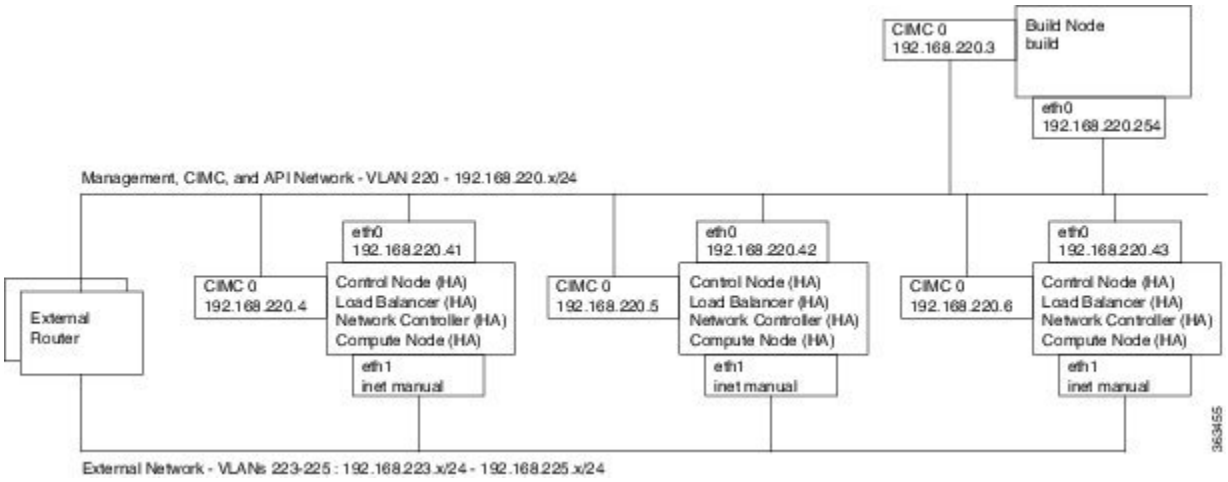
The following figure shows a three-role scenario.

Figure 5: Three-Role Scenario



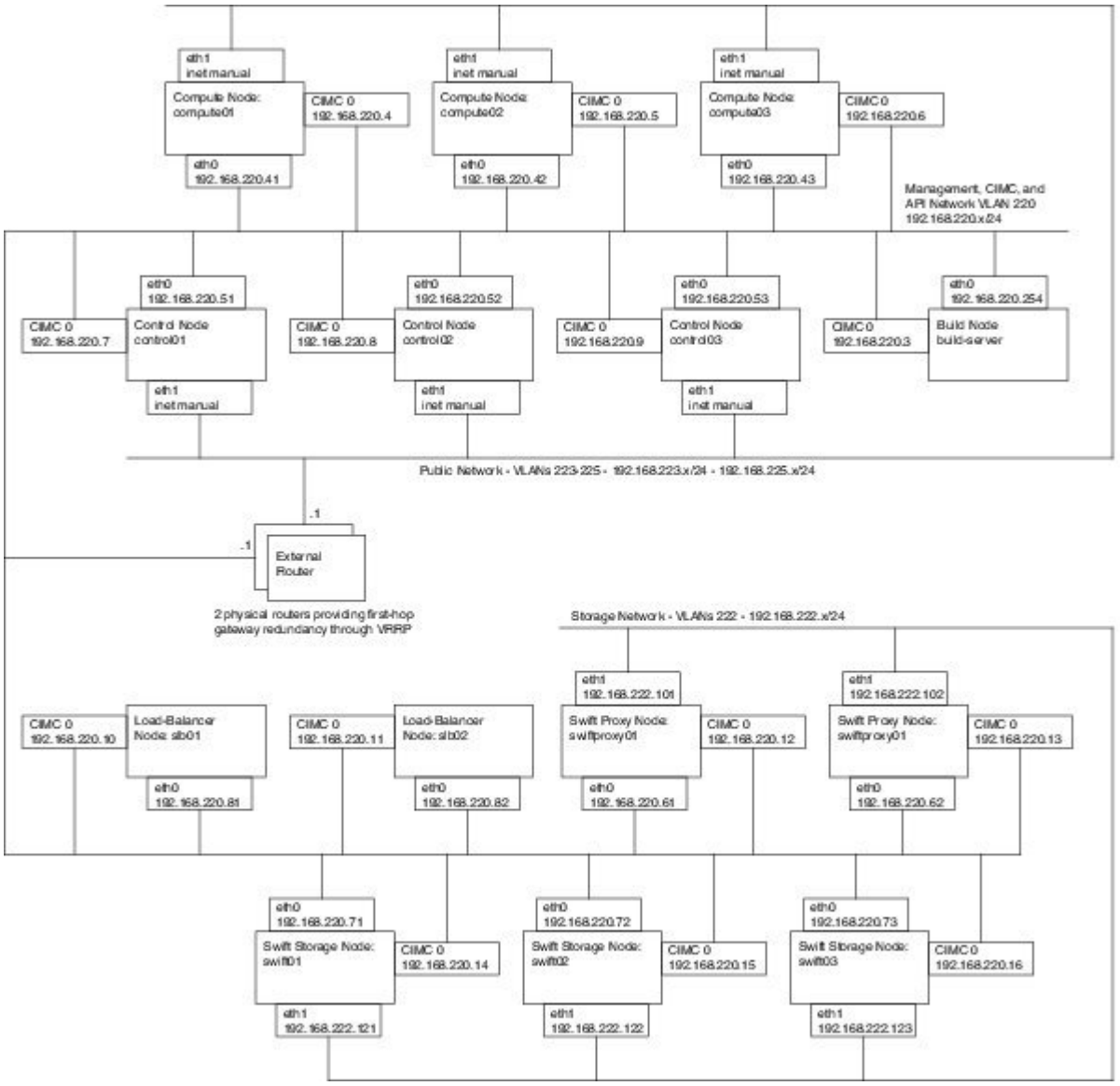
The following figure shows a compressed high-availability (HA) scenario.

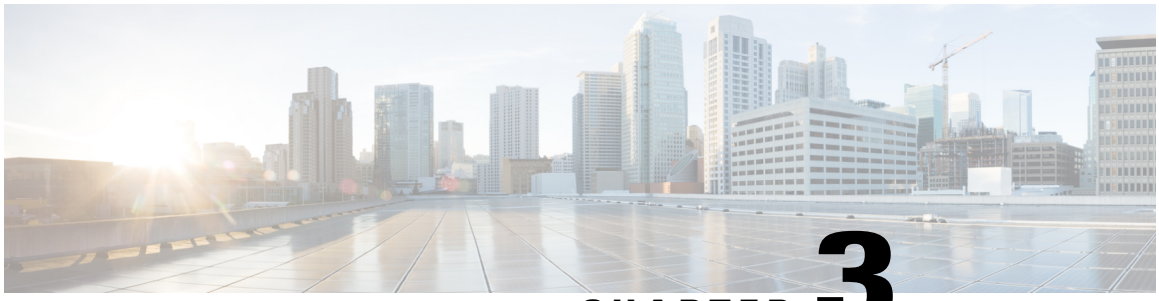
Figure 6: Compressed High-Availability Scenario



The following figure shows a full HA scenario.

Figure 7: Full High-Availability Scenario





CHAPTER 3

Installing OpenStack

Creating the Build Server

About Configuring an All-In-One Deployment

The all-in-one (AIO) scenario has some crucial differences from the other scenarios. If you are not deploying an AIO scenario, you can skip this section.

You can install the AIO scenario using the default values supplied with the installer. If you have non-standard interface definitions, hostnames, proxies, or other parameters, you can change the baseline AIO configuration.

If you want to modify the AIO configuration, we recommend that you do so before running the `install.sh` script. For the AIO install, all changes to the configuration are picked up and installed during the run of the `install.sh` script.

Two example customizations of the AIO Model 1 setup are as follows:

- Using an interface other than eth0 for SSH/Management. Export the default_interface value to the correct interface definition. For example, to use eth1, enter the following:
  ```
  export default_interface=eth1
  ```
- Using an interface other than eth1 on your node for external (public) access. Export the external_interface value to the correct interface definition. For example, to use eth2, enter the following:
  ```
  export external_interface=eth2
  ```

Note You can modify configuration parameters after running the `install.sh` script, but be aware of the following:

- The `install.sh` script does a Puppet catalog run and will reflect changes in the yaml files. If you change parameters after running the `install.sh` script, you must run **puppet apply** on the all-in-one node to pick up the changes.
- If you run the `install.sh` script before customizing the installation, you might have to clean up some artifacts of the original parameter settings. For example, you might need to reconfigure SSL certificates, delete misconfigured network endpoints, and so on.
- The installer creates the directory `/etc/puppet` and installs the configuration files there. Therefore, you must make changes in the `/root/puppet_openstack_builder/data` directory before you run the install. After you run the install, you make changes in the `/etc/puppet/data` directory.

Creating the Build Node

The server that you use for the build node can be a physical server or a virtual machine (VM).

Note Puppet relies on SSL certificates for authentication. Before you run Puppet for the first time, you must do the following:

- Ensure that the node's domain and hostname are consistent with the name to be distributed as the puppet master to all OpenStack component nodes. Check the names in `/etc/hosts` before proceeding.
- Obtain the name of a ntp time server. You will use this name during the install procedure.

If you are using Cobbler to deploy your nodes, these two steps are automatically included for you, but you must still make sure that the domain and time are correctly set on your build node before running the install script. See the Puppet documentation regarding certification management for more information.

Before You Begin

- Ensure that the server that you plan to use as the build node meets the minimum server requirements described in Minimum Server Requirements, on page 10.
- If you use a proxy, have your proxy information available; you will use it during the install procedure.
- Validate forward and reverse DNS lookups for the build server IP address and fully qualified domain name (FQDN).

Step 1 Install Ubuntu 14.04 LTS on the server, with the following features:

- Ensure that the OpenSSH server is installed.

- Configure the network interface on the OpenStack cluster management segment with a static IP address.
- When partitioning the storage, choose a partitioning scheme that provides at least 15 GB of free space under `/var`, because the installation packages and ISO images that are used to deploy OpenStack are cached there.

For detailed instructions on installing Ubuntu Linux for use as a build node, see https://github.com/CiscoSystems/coi-docs/tree/icehouse/install_guides.

Step 2 When the installation is complete, log in as root.
sudo su -

Step 3 If your environment includes a proxy server, configure the package manager to use the proxy.

a) For **apt**, add the following to the `/etc/apt/apt.conf.d/00proxy` file:

Note Even if you are using **git** to install packages, you will use **apt** to install **git**, so do not skip this step.

Example:
```
Acquire::https::proxy https://your-proxy.address.com:443/
```

b) Because some transparent proxy servers corrupt package indexes, Cisco OSI turns off HTTP pipelining in the apt configuration. Since Cisco OSI uses apt-get to download packages to the build node, you might want to disable pipelining on your build node before downloading Cisco OSI.
To disable pipelining, create a file called `/etc/apt/apt.conf.d/00no_pipelining` that contains the following line:

Example:
```
Acquire::http::Pipeline-Depth "0";
```

Step 4 Ensure that the build node has the proper host and domain names and that it is time synchronized:

a) Log in as root.

Example:
```
sudo su -
```

b) Check these names on all nodes before you begin to make sure they are what you expect.

Example:
```
hostname -d
hostname -f
```

c) If you want to change a host or domain name, make the change now. Change the `/etc/hostname` and `/etc/hosts` files to include the correct host and domain names.

Example:
```
sudo su -                 # must be root to change the hosts and hostname files

/etc/hostname:            # edit as shown for build server
build-server

/etc/hosts:               # edit as shown for build server
127.0.1.1           your.domain.name          build-server
host-ip-address     build-server.your.domain.name
```

Note For all-in-one installs, recall that the default build node name is `all-in-one`, not `build-server`. See About Configuring an All-In-One Deployment, on page 17
.

d) Reboot the node.

Example:
```
reboot -f
```

e) Sync the build node with a time server.

Example:
```
ntpdate ntp_server_name
```

Step 5 Log in as root and install git as follows:

a) Retrieve and install git.

Example:
```
sudo su -
apt-get install -y git
```

b) If necessary, configure git with your proxy.

Example:
```
git config --global https.proxy https://your-proxy.address.com:443
```

Step 6 Clone the Cisco OpenStack installer repository into `/root`.

Example:
```
cd /root
git clone -b icehouse https://github.com/CiscoSystems/puppet_openstack_builder
```

Step 7 Check out the Cisco OSI.

Example:
```
cd puppet_openstack_builder
git checkout i.0
```

Step 8 Set Cisco as the vendor by setting the *vendor* environment variable.

Example:
```
export vendor=cisco
```

Step 9 Specify a deployment scenario. See Choosing a Deployment Scenario, on page 11 to determine which scenario is appropriate to your application.

Example:
```
export scenario=scenario_name
```

Step 10 Run the `install.sh` script. The script installs Puppet and prepares modules and repositories on the build node.

Example:
```
cd ~/puppet_openstack_builder/install-scripts
./install.sh
```

The Puppet Master, modules, and repositories are installed on the build node. At this point, the data model has been installed on the build node. Any required customization should be made to the data model and not to the Puppet manifests.

Running the install command as given above generates the `/var/log/puppet_openstack_builder_install.log` file that contains all the output that was sent to the terminal window. It contains information that can be useful should you need to debug the installation.

Note Run the `install.sh` script only once to install Puppet. If you make changes to the data model after running `install.sh`, you can make them take effect by running Puppet directly as follows:
`puppet apply -v /etc/puppet/manifests/site.pp`
See Rerunning Puppet on the Build Node and Enabling Puppet to Run as an Agent, on page 31.

What to Do Next

Customize the data model and install the OpenStack nodes for the deployment scenario that you have chosen.

Customizing the Build Server

If you selected the all_in_one scenario, an example configuration has already been placed in `/etc/puppet/data/hiera_data/user.yaml`. See Installing An All-In-One Deployment.

Regardless of which scenario you have chosen, you will customize the installation by changing the data model.

If you selected the all_in_one scenario, edit the configuration parameters in the `/etc/puppet/data/hiera_data/user.yaml` file.

If you selected any other scenario, you can replace default configuration parameters in `/etc/puppet/data/hiera_data/user.common.yaml`, `/etc/puppet/data/hiera_data/common.yaml`, and `/etc/puppet/data/hiera_data/user.scenario.yaml` where scenario is your deployment scenario. Alternatively, you can set parameters in `/etc/puppet/data/hiera_data/user.yaml`, which will override the default value lower in the hierarchy. See YAML Files and the Hiera Database, on page 4.

Before You Begin

Set up the build node and run the `install.sh` script on it.

Step 1 Configure hostnames in `/etc/puppet/data/hiera_data/user.common.yaml` (for non-all-in-one installs).

Example:

```
######### Node Addresses ##############
# Change the following to the short host name you have given your build node.
# This name should be in all lower case letters due to a Puppet limitation
# (refer to http://projects.puppetlabs.com/issues/1168).
build_node_name: build-server

# Change the following to the host name you have given to your control
# node. This name should be in all lower case letters due to a Puppet
# limitation (refer to http://projects.puppetlabs.com/issues/1168).
coe::base::controller_hostname: control-server
```

Note Supply only the short hostname to the build_node parameter, not the fully qualified domain name.

Step 2 Specify the software repository (or *repo*) from which you will copy packages. This configuration is in `common.yaml`. Use the Cisco repo if you have a choice. The cloud archive is not tested as regularly by Cisco engineers and might contain inconsistencies.

Example:

```
# The package repository to be used. Valid values include 'cloud_archive'
# (use the Ubuntu Cloud Archive) and 'cisco_repo' (use the Cisco Systems
# OpenStack repository).
coe::base::package_repo: repo_label

# The base version of OpenStack to be installed (e.g. 'havana').
openstack_release: icehouse

# The 'openstack_repo_location' parameter should be the complete
# URL of the repository you want to use to fetch OpenStack
# packages (e.g. http://openstack-repo.cisco.com/openstack/cisco).
# This setting is not used by all vendors.
openstack_repo_location: 'http://openstack-repo.cisco.com/openstack/cisco'

# The 'supplemental_repo' parameter should be the complete URL
# of the repository you want to use for supplemental packages
# (e.g. http://openstack-repo.cisco.com/openstack/cisco_supplemental).
# This setting is not used by all vendors.
supplemental_repo: 'http://openstack-repo.cisco.com/openstack/cisco_supplemental'
```

The following settings are in `user.common.yaml`:

```
# If you use an HTTP/HTTPS proxy to reach the apt repositories that
# packages will be installed from during installation, uncomment this
# setting and specify the correct proxy URL. If you do not use an
# HTTP/HTTPS proxy, leave this setting commented out.
#proxy: 'http://proxy-server.domain.com:8080'

# The default gateway that should be provided to DHCP clients that acquire
# an address from Cobbler.
node_gateway: 'gateway_ip'
```

Step 3 Configure connectivity in `user.common.yaml`.

Example:

```
# This domain name will be the name your build and compute nodes use for the
# local DNS. It doesn't have to be the name of your corporate DNS - a local
# DNS server on the build node will serve addresses in this domain - but if
# it is, you can also add entries for the nodes in your corporate DNS
# environment they will be usable *if* the above addresses are routeable
# from elsewhere in your network.
domain_name: domain.name

############ NTP Configuration ############
# Change this to the location of a time server or servers in your
# organization accessible to the build server. The build server will
# synchronize with this time server, and will in turn function as the time
# server for your OpenStack nodes.
ntp_servers:
  - time-server.domain.name

# This sets the IP for the private(internal) interface of controller nodes
# (which is predefined already in $controller_node_internal, and the internal
# interface for compute nodes. It is generally also the IP address
# used in Cobbler node definitions.
internal_ip: "%{ipaddress_eth3}"

# The IP address on which vncserver proxyclient should listen.
# This should generally be an address that is accessible via
# horizon. You can set it to an actual IP address (e.g. "192.168.1.1"),
```

```
# or use facter to get the IP address assigned to a particular interface.
nova::compute::vncserver_proxyclient_address: "%{ipaddress_eth3}"

# The external_interface is used to provide a Layer2 path for
# the l3_agent external router interface. It is expected that
# this interface be attached to an upstream device that provides
# a L3 router interface, with the default router configuration
# assuming that the first non "network" address in the external
# network IP subnet will be used as the default forwarding path
# if no more specific host routes are added.
external_interface: eth2

# The public_interface will have an IP address reachable by
# all other nodes in the openstack cluster. This address will
# be used for API Access, for the Horizon UI, and as an endpoint
# for the default GRE tunnel mechanism used in the OVS network
# configuration.
public_interface: eth1

# The interface used for VM networking connectivity. This will usually
# be set to the same interface as public_interface.
private_interface: eth1

# The IP address to be used to connect to Horizon and external
# services on the control node. In the compressed_ha or full_ha scenarios,
# this will be an address to be configured as a VIP on the HAProxy
# load balancers, not the address of the control node itself.
controller_public_address: controller_public_ip

# The IP address used for internal communication with the control node.
# In the compressed_ha or full_ha scenarios, this will be an address
# to be configured as a VIP on the HAProxy load balancers, not the address
# of the control node itself.
controller_internal_address: controller_internal_ip
```

Note The variable *%{ipaddress_eth1}* is a Puppet fact supplied by the Facter tool. See http://docs.puppetlabs.com/learning/variables.html.

Step 4 Configure passwords in `user.common.yaml`.

Example:

```
### The following are passwords and usernames used for
### individual services. You may wish to change the passwords below
### in order to better secure your installation.
cinder_db_password: cinder_pass
glance_db_password: glance_pass
keystone_db_password: key_pass
nova_db_password: nova_pass
network_db_password: quantum_pass
database_root_password: mysql_pass
cinder_service_password: cinder_pass
glance_service_password: glance_pass
nova_service_password: nova_pass
ceilometer_service_password: ceilometer_pass
admin_password: Cisco123
admin_token: keystone_admin_token
network_service_password: quantum_pass
rpc_password: openstack_rabbit_password
metadata_shared_secret: metadata_shared_secret
horizon_secret_key: horizon_secret_key
ceilometer_metering_secret: ceilometer_metering_secret
ceilometer_db_password: ceilometer
heat_db_password: heat
heat_service_password: heat_pass
heat::engine::auth_encryption_key: 'notgood but just long enough i think'

# Set this parameter to use a single secret for the Horizon secret
# key, neutron agents, Nova API metadata proxies, swift hashes,etc.
# This prevents you from needing to specify individual secrets above,
# but has some security implications in that all services are using
```

```
# the same secret (creating more vulnerable services if it should be
# compromised).
secret_key: secret

# Set this parameter to use a single password for all the services above.
# This prevents you from needing to specify individual passwords above,
# but has some security implications in that all services are using
# the same password (creating more vulnerable services if it should be
# compromised).
password: password123
```

Step 5 Configure disk partitioning in `user.common.yaml`.

Example:

```
#### Disk partitioning options ###########
# The /var directory is where logfiles and instance data are stored
# on disk. If you wish to have /var on it's own partition (considered
# a best practice), set enable_var to true.
enable_var: true

# The Cinder volume service can make use of unallocated space within
# the "cinder-volumes" volume group to create iscsi volumes for
# export to instances. If you wish to leave free space for volumes
# and not preallocate the entire install drive, set enable_vol_space
# to true.
enable_vol_space: true

# Use the following two directives to set the size of the / and /var
# partitions, respectively. The var_part_size directive will be ignored
# if enable_var is not set to true above.
root_part_size: 65536
var_part_size: 432000
```

Step 6 Provide the following advanced options to your build node to enable special features during baremetal provisioning. The advanced options can be placed in a host override file such as `/etc/puppet/data/hiera_data/hostname/`*`your_hostname`*`.yaml` or in `/etc/puppet/data/hiera_data/user.common.yaml`. Not all of these options are required for all scenarios.

a) In user.common.yaml, install a specific kernel package and set it to be the kernel booted by default by setting the `load_kernel_pkg` directive.

Example:

```
load_kernel_pkg: 'linux-image-3.2.0-51-generic'
```

b) Specify command-line options that should be passed to the kernel at boot time by setting the kernel_boot_params directive.

Note We recommend that you use elevator=deadline if you use ISCSI volumes in Cinder. Some I/O issues have been reported using the default elevator.

Example:

```
kernel_boot_params: 'quiet splash elevator=deadline'
```

c) Set the time zone on the clock for nodes booted by using Cobbler and setting this directive in `user.common.yaml`:

Example:

```
# The time zone that clocks should be set to. See /usr/share/zoneinfo
# for valid values, such as "UTC" and "US/Eastern".
time_zone: your_timezone
```

Note We recommend that you use UTC on all OpenStack nodes.

d) For the full_ha and compressed_ha scenarios only: On the command line, copy the `/etc/puppet/data/hiera_data/hostname/build_server.yaml` file to a build node-specific filename.

Example:
```
cd /etc/puppet/data/hiera_data/hostname
cp build_server.yaml build_server_name.yaml
```
where *build_server_name* is the short name of your build server.

Step 7 In `/etc/puppet/data/role_mappings.yaml`, map the roles in your selected scenario to hostnames. The format for each entry is `host_name: role_name`. You must supply an entry for every node in your deployment. It is not necessary to remove unused node names.

Example:
```
control-server: controller
control-server01: controller
control-server02: controller
control-server03: controller

compute-server: compute
compute-server01: compute
compute-server02: compute
compute-server03: compute

all-in-one: all_in_one

build-server: build

cache: cache

load-balancer01: load_balancer
load-balancer02: load_balancer

swift-proxy01: swift_proxy
swift-proxy02: swift_proxy

swift-storage01: swift_storage
swift-storage02: swift_storage
swift-storage03: swift_storage

#Stacktira roles
build.+: build
compute.+: compute
control.+: control

#compressed_ha
node01: compressed_ha
node02: compressed_ha
node03: compressed_ha
```
If you selected the all_in_one scenario, the install.sh script automatically adds the all-in-one node to this file.

Step 8 If you set the hostname of your build node to something other than *build-node*, set Cobbler-related directives in a host override file as follows:
In `/etc/puppet/data/hiera_data/hostname/build-server.yaml`, you see several directives used by Cobbler. Copy these into your `/etc/puppet/data/hiera_data/hostname/build_node_name.yaml` file and set them to the appropriate values for your build node.

Example:
```
# set my puppet_master_address to be fqdn
puppet_master_address: "%{fqdn}"
cobbler_node_ip: 'cobbler_node_ip'
node_subnet: 'cobbler_node_subnet'
node_netmask: 'cobbler_node_netmask'
```

```
node_gateway: 'cobbler_node_gateway'
admin_user: localadmin
# Will look something like
"$6$UfgWxrIv$k4KfzAEMqMg.fppmSOTd0usI4j6gfjs0962.JXsoJRWa5wMz8yQk4SfInn4.WZ3L/MCt5u.62tHDGB36EhiKF1"
password_crypted: encrypted_password
autostart_puppet: true
ucsm_port: 443
install_drive: /dev/sda
#ipv6_ra: 1
#interface_bonding = 'true'
```

What to Do Next

Build the control and compute nodes.

Building the Control and Compute Nodes

About Building the Control and Compute Nodes

You can provision and configure your control and compute nodes individually or as a group using Cobbler.

If you build the control and compute nodes individually, you can control the order in which the nodes are built. If one or more nodes have a dependency on an application that is running on another node, you can ensure that the correct node is built first so that the other nodes do not fail. For example, you might want to build the control node first if you need Keystone fully configured on that node to ensure that other nodes can reach Keystone during their own installations.

If you provision the nodes individually you can do the following:

- Use Cobbler to provision the node if you are starting from baremetal.
- Clone Cisco OSI from GitHub, install a Puppet agent, and use the agent to configure the node, if Ubuntu is already installed.

Building the Control and Compute Nodes Individually with Cobbler

Provision a node with Cobbler if you want to remove everything from the node and start from baremetal. If you do not want to erase the node before provisioning, use the procedure described in Building the Control and Compute Nodes Individually With Puppet Agent, on page 27 to provision the node.

Note Setting up the control node might require more than one Puppet catalog run, especially if there are proxies in the path or if you are installing an HA or compressed HA scenario. Proxies can have issues with apt-get installations and updates. HA control requires more than one Puppet run to resolve circular redundancy references.

You can verify that the control node configuration has converged completely to the configuration defined in Puppet by looking at the log files in the `/var/log/syslog` directory on the control node.

Before You Begin

Set up your build server before you begin building other nodes.

Run the `clean_node.sh` script with a node name as the only argument.

Example:

```
cd /root/puppet_openstack_builder/scripts
bash clean_node.sh node_name
```

The script performs several actions in sequence:

1. Removes any previously installed Puppet certificates.
2. Enables netboot for the node.
3. Power cycles the node.
4. When the machine reboots, starts a PXE install of the baremetal operating system.
5. After the operating system is installed, reboots the machine and starts a Puppet Agent.
6. The agent immediately begins installing OpenStack.

What to Do Next

Repeat these procedures to install all the nodes in your deployment scenario.

Note The `setup.sh` script should only be run once. To force an update later, restart the Puppet Agent.

Building the Control and Compute Nodes Individually With Puppet Agent

To configure the node without reinstalling Ubuntu, you can use the following procedure. These steps set up Puppet and then perform a catalog run.

Step 1 If necessary, install git on the node.

Example:

```
apt-get install -y git
```

Step 2 Clone Cisco OSI onto the node.

Example:

```
cd /root
git clone -b icehouse https://github.com/CiscoSystems/puppet_openstack_builder
```

Step 3 Check out the Cisco OSI.

Example:

```
cd puppet_openstack_builder
git checkout i.0
```

Step 4 Run the setup.sh script.

Example:

```
cd install_scripts
export build_server_ip=your_build_node_ip
bash setup.sh
```

Step 5 Start the Puppet Agent.

Example:

```
puppet agent -td --server=your_build_node_fqdn --pluginsync
```

What to Do Next

Repeat these procedures to install all the nodes in your deployment scenario.

Building Multiple Control and Compute Nodes with Cobbler

The 2_role and full_ha scenarios configure a build server that provides Puppet Master and optionally Cobbler baremetal deployment services for managing the remaining nodes in the OpenStack cluster. These services do the following:

- Remove any existing Puppet certificates.
- Remove any existing SSH parameters.
- Restart the Cobbler build system.
- Use Cobbler to power control the node according to the configuration in the `/etc/puppet/data/cobbler/cobbler.yaml` file.

Step 1 Set up role mappings in `/etc/puppet/data/role_mappings.yaml` as described in Customizing the Build Server, on page 21.

Step 2 Provide hardware information, including the MAC addresses of the network boot interfaces, machine hostnames, machine IP addresses, and management account information by editing the `/etc/puppet/data/cobbler/cobbler.yaml file`.
The `cobbler.yaml` file has four major sections:

- Preseed
- Profile
- Node-global preseed
- Individual node definitions

a) Edit the preseed section.
A preseed section defines parameters that customize the preseed file that is used to install and configure Ubuntu on the servers. Parameters that you might need to adjust include default repos to include in hosts and locations of these repos.

Note Cobbler uses Intelligent Platform Management Interface (IPMI) for power management of Cisco UCS C-Series Servers. Install the IPMI package if using Cisco UCS C-Series servers: `apt-get install -y ipmitool .`

Example:
```
preseed:
  repo: "http://openstack-repo.cisco.com/openstack/cisco icehouse main"
```

b) Verify the profile section.
A profile section defines options that specify the Cobbler profile parameters to apply to the servers. This section typically does not require customization.

Example:
```
profile:
  name: "precise"
  arch: "x86_64"
  kopts: "log_port=514 \
priority=critical \
local=en_US \
log_host=192.168.242.100 \
netcfg/choose_interface=auto"
```

c) Edit the node-global section.
Node-global specifies configuration parameters that are common across all servers in the cluster, such as gateway addresses, netmasks, and DNS servers.

Power parameters that are standardized also are included in this section. You must change several parameters in this section.

Example:
```
node-global:
  profile: "precise-x86_64"
  netboot-enabled: "1"
  power-type: "ipmilan"
  power-user: "admin"
  power-pass: "password"
  kickstart: "/etc/cobbler/preseed/cisco-preseed"
  kopts: "netcfg/get_nameservers=2.4.1.254 \
netcfg/confirm_static=true \
netcfg/get_ipaddress={$eth0_ip-address} \
netcfg/get_gateway=192.168.242.100 \
netcfg/disable_autoconfig=true \
netcfg/dhcp_options=\"Configure network manually\" \
netcfg/no_default_route=true \
partman-auto/disk=/dev/sda \
netcfg/get_netmask=255.255.255.0 \
netcfg/dhcp_failed=true"
```

Note The power_type parameter should be set to *ipmilan* for servers with an IPMI interface, including Cisco UCS C-series in standalone mode. Set power_type to *ucs* for Cisco UCS B- and C-series servers managed by Cisco UCSM.

```
# Power configuration parameters for standalone nodes
  ...
  power_type: "ipmilan"
  power_user: "user1"
  power_pass: "user1_pass"
```

```
  ...

# Power configuration for managed nodes
  ...
  power_type: "ucs"
  power_user: "domain/useraccount"
  power_pass: "useraccount_domain_password"
  ...
```

d) Create one section for each node, using a format as shown in this example.
Each node managed by Cobbler is listed as a separate definition. The node definition for each host defines, at a minimum, the hostname, power parameters, and interface configuration information for each server, as well as any parameters not defined in node-global.

Example:

```
build-server:
  hostname: "build-server.cisco.com"
  power_address: "192.168.2.101"
  interfaces:
    eth0:
    mac-address: "a1:bb:cc:dd:ee:ff"
    dns-name: "build-server.cisco.com"
    ip-address: "192.168.242.100"
    static: "0"
```

For IPMI, the power_address parameter identifies separate CIMC addresses. For ucs, the power_address parameter identifies the UCS-M server and organization or suborganization within UCS-M. UCS has an additional field, power_id, to identify the node's service profile as specified in UCS-M.

```
# Individual power parameters for standalone nodes

first_server:
  hostname: "first_server"
  power_address: "CIMC of first_server"
  ...

second_server:
  hostname: "second_server"
  power_address: "CIMC of second_server"
  ...

# Individual power parameters for managed nodes

first_server:
  hostname: "first_server"
  power_address: "UCS-M server:org-NAME"
  power_id: "FirstServerServiceProfileName"
  ...

second_server:
  hostname: "second_server"
  power_address: "UCS-M server:org-NAME"
  power_id: "SecondServerServiceProfileName"
  ...
```

Step 3 Run the setup script:

a) Export the IP address of your build node.

```
export build_server_ip=build_node_ip
```

b) Change to the install_scripts directory.

```
cd ~/puppet_openstack_builder/install-scripts/
```

c) To prepare the node for the Puppet agent run, run the `setup.sh` script.

```
bash setup.sh
```

Rerunning Puppet on the Build Node and Enabling Puppet to Run as an Agent

After you install puppet has been installed on a node by running `install.sh` (on the build node) or `setup.sh` (on control nodes), all further modifications to the node should be made by modifying the model and running Puppet directly.

If you have made changes to the model (for example in `cobbler.yaml` or in a `.yaml` file in `/etc/puppet/data`), the changes take effect when the Puppet Agent does an automatic catalog run (by default, the Puppet Agent performs a catalog run every 30 minutes). If you need to update a node immediately, you can force Puppet to perform a catalog run.

Before You Begin

Set up the build node using install.sh. Provision and set up Cisco OSI nodes for your scenario.

Run Puppet on an agent node for a second time.

Note Depending on configuration parameters set in the Puppet YAML files on the build node, you might need to enable the Puppet Agent before you can run it manually on OpenStack nodes:

```
puppet agent --enable
```

- If Puppet is not set up to run automatically on an OpenStack node, force an update by running the Puppet Agent manually:

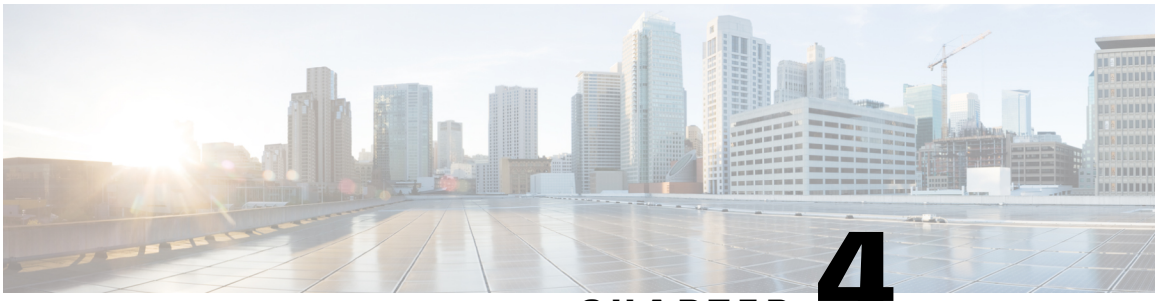
  ```
  puppet agent -td --server=build_node_FQDN
      --pluginsync
  ```

- To force an update from the build node, restart the Puppet service on the build node:

  ```
  service puppet restart
  ```

- Alternatively, you can force an immediate catalog run directly on the build node:

  ```
  puppet apply -v /etc/puppet/manifests/site.pp
  ```



CHAPTER 4

Testing the OpenStack Deployment

About Testing the OpenStack Deployment

This section provides the following procedures for testing that your OpenStack deployment is installed and working correctly:

- Verifying the running compute instances.
- Opening the monitoring interface and checking the status of the components.
- Creating a virtual network.
- Creating a virtual machine instance.

Verifying the Compute Instances

After you complete the installation, all the defined compute nodes should be running.

At the command line on the control node, view the running compute nodes by entering:

```
nova-manage service list
```

This command verifies that the OpenStack Nova services were installed and are running correctly.

The system returns a table that looks like the following:

```
Binary   Host                 Zone             Status     State Updated_At
nova-consoleauth all-in-one          internal         enabled    :-)   2014-03-11 17:34:17
nova-scheduler   all-in-one          internal         enabled    :-)   2014-03-11 17:34:16
nova-conductor   all-in-one          internal         enabled    :-)   2014-03-11 17:34:13
```

```
nova-compute     all-in-one          nova             enabled    :-)   2014-03-11 17:34:13
nova-cert        all-in-one          internal         enabled    :-)   2014-03-11 17:34:17
```

Using the Monitoring Interface

With the OpenStack deployment running, the Horizon monitoring interface is available. To log into the monitoring interface, do the following:

Step 1 In your browser, navigate to `http://control-node-IP/horizon/`

Step 2 Log into Horizon with the admin username and password in the `site.pp` file.
If you did not change the defaults, the username is admin, and the password is Cisco123.

Step 3 Examine the compute nodes in the interface:

a) In the **Navigation** pane on the right side of the interface, click the **Admin** tab.

b) In the **System Panel** on the **Admin** tab, choose **Hypervisors**.
The **Work** pane shows a table of all the running compute nodes.

Creating a Network

This section describes how to create a public network to be used for instances (also called virtual machines, or VMs) to gain external (public) connectivity. VMs are connected externally through a router you create on the control node and are connected to the router through a private GRE network.

Step 1 Create a public network. On a control node, enter the following:

```
neutron net-create public_network_name \
--router:external=True
```

Step 2 Create a subnet that is associated with the new public network.

Note The range of IP addresses in your subnet must not conflict with other network nodes on the subnet. For example, if you have a gateway upstream using addresses in the public subnet ranges (192.168.81.1, 192.168.81.2, and so on) your allocation range must start in a nonoverlapping range.

IP addresses are examples only. Use IP addresses that are consistent with your network configuration and policies.

```
neutron subnet-create --name public_subnet_name \
--allocation-pool start=192.168.220.20,end=192.168.220.253 \
public_network_name 192.168.220.0/24
```

Step 3 Create a private network and subnet to attach instances to.

```
neutron net-create private_network_name
neutron subnet-create --name private_subnet_name \
```

```
private_network_name 10.10.10.0/24 \
--dns_nameservers nameserver1 nameserver2
```

Step 4 Create a Neutron router.

```
neutron router-create os_router_name
```

Step 5 Associate the Neutron router interface with the previously created private subnet.

```
neutron router-interface-add os_router_name \
private_subnet_name
```

Step 6 Set the default gateway (previously created public network) for the Neutron router.

```
neutron router-gateway-set os_router_name \
public_network_name
```

Step 7 Modify the default Neutron security group to allow for ICMP and SSH (for access to the instances).

```
neutron security-group-rule-create --protocol icmp --direction ingress default
neutron security-group-rule-create --protocol tcp --port-range-min 22 \
--port-range-max 22 --direction ingress default
```

Creating a Tenant Instance

If you have one or more compute nodes running, you can create a VM on the OpenStack cloud.

Before You Begin

- You must have a running compute node and a node running the Glance image database.
- You must have control node with private and public networks as described in Creating a Network, on page 34.

Step 1 Load a VM image into Glance:

a) Download an image to deploy.
A popular small test image is Cirros.

```
wget http://download.cirros-cloud.net/0.3.1/cirros-0.3.1-x86_64-disk.img
```

b) Store the image in Glance.

```
glance image-create --name cirros-x86_64 --is-public True \
--disk-format qcow2 --container-format ovf --file cirros-0.3.1-x86_64-disk.img \
--progress
```

Step 2 Boot an Instance:

a) Enter the **neutron net-list** command to get a list of networks.

```
# neutron net-list
```

b) Boot the instance. Using the ID in the --nic net-id= field for the private network (*private_network_name* from the example in Creating a Network, on page 34, for example) from the table displayed by the net-list command in the previous step, enter the following:

```
nova boot --image cirros-x86_64 --flavor m1.tiny --key_name aio-key \
--nic net-id=32-byte_net-id test_vm_name
```

Step 3 Verify that your instance has spawned successfully.

Note The first time an instance is launched on the system can take a bit longer to boot than subsequent launches of instances.

Enter:

```
nova show test_vm_name
```

Step 4 Get the internal fixed IP of your instance with the following command:

```
nova show test_vm_name
```

Step 5 Verify connectivity to the instance from the control node.

Note Because namespaces are being used in this model, you must run the commands from the context of the qrouter using the ip netns exec qrouter syntax

a) List the qrouter to get its routerid.

```
neutron router-list
```

Alternatively, you can get the qrouter ID using the **ip** command.

```
ip netns
```

b) Connect to the qrouter and get a list of its addresses.

```
ip netns exec qrouter-neutron_router_id ip addr list
```

c) Ping the instance from the qrouter.

```
ip netns exec qrouter-neutron_router_id ping fixed_ip_of_instance
```

d) Use SSH to get into the instance from the qrouter.

```
ip netns exec qrouter-neutron_router_id ssh cirros@fixed_ip_of_instance
```

Step 6 Create a floating IP address for the VM.

a) Get a list of the networks.

```
neutron net-list
```

b) Get a list of the ports.

```
neutron port-list
```

c) Copy the correct IDs.

```
neutron floatingip-create --port_id internal_VM_port-id \
public_net-id
```

Step 7 From an external host, ping and SSH to your instance using the floating_ip_address.

```
ping floating_ip
```