Cisco OpenStack Installer Deployment Guide,Havana Release

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.

Warning **Means *reader be warned.* In this situation, you might perform an action that could result in bodily injury.**

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 Cisco OpenStack Installer cluster 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 Havana release is documented on GitHub in these two articles:
  - Configuring OpenStack as Data
  - Scenario Based Override Terminus

Other Resources

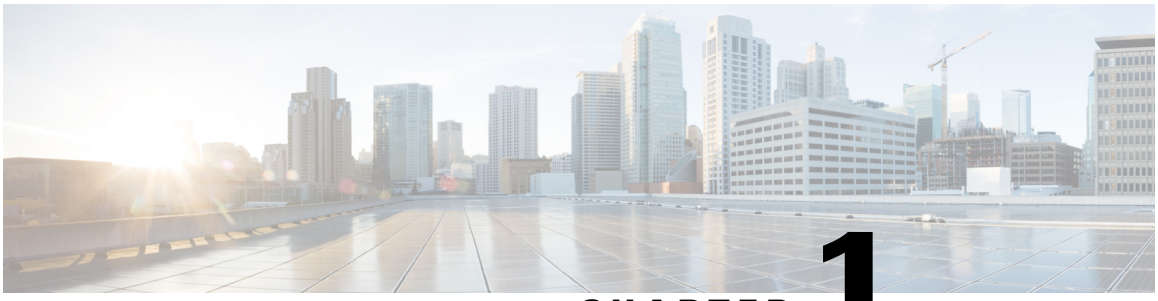
Information about the upstream community OpenStack project and Puppet install tools is available on the web:

- 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 Havana 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.

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. We deviate 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 we apply the patches to our repositories, submit them upstream, and replace the local change with the upstream version when it is becomes accepted). We also use and contribute to modules from other upstream sources including modules and manifests Puppet Labs on StackForge.

Cisco OpenStack Installer provides the most recent release of each of the OpenStack Client Libraries (for example, `python-neutronclient`, `python-novaclient`, and so on) unless doing so causes testing to fail. In such cases, Cisco OSI will include the latest community release that does not cause problems in testing. The version of each component supplied with this release is documented in Component Versions, on page 3.

Havana Release of Cisco OSI

The Havana release represents a significant change from previous Cisco OpenStack Installers. Past versions required users to edit the Puppet manifest file `site.pp`. The Havana release features Scenario Node Terminus, a new Puppet module. You use the Scenario Node Terminus module and Puppet Labs' Hiera tool to describe to describe a data model for your deployment. This new installation model provides you more flexibility to modify almost any aspect of your OpenStack install without needing to master the Puppet tool.

Puppet and Cobbler Installation Automation Tools

Cisco OSI installs and configures OpenStack components by using Puppet, and using 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 information, see Cobbler) in order to provide provisioning of physical servers in the OpenStack cloud.

OpenStack Components

Supported 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 and, optionally, Heat for orchestration. You can also deploy a Horizon dashboard and Ceilometer telemetry.

For object storage, you can choose to have Cisco OSI deploy either **Ceph** or **Swift**; either can be used as a backend for Glance. Ceph can provide block storage, and can serve as a backend for Cinder or as a standalone storage service.

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

OpenStack incubator projects that have not been added to the core products are not supported in Cisco OSI.

Component Versions

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

- The OpenStack and Puppet packages are listed in the main repository:
  - 64 bit (amd64):
    http://openstack-repo.cisco.com/openstack/cisco/dists/havana/snapshots/h.1/main/binary-amd64/Packages.
  - 32 bit (i386):
    http://openstack-repo.cisco.com/openstack/cisco/dists/havana/snapshots/h.1/main/binary-i386/Packages.
- Optional third-party components are listed in the supplemental repository:
  - 64 bit (amd64):
    http://openstack-repo.cisco.com/openstack/cisco_supplemental/dists/havana/snapshots/h.1/main/binary-amd64/Packages.
  - 32 bit (i386):
    http://openstack-repo.cisco.com/openstack/cisco_supplemental/dists/havana/snapshots/h.1/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 scenario that allows this (in terms of number of distinct nodes). See Choosing a Deployment Scenario, on page 9 for descriptions of scenarios that are included with the Havana 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 provides the following functions:

- As 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.
- As 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.
- As 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.

- As 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.
- As 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 three files:

- `/etc/puppet/hiera.yaml`

  Ths 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 will 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 will 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 deployment scenario 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 bare-metal 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 20.

IP Networks

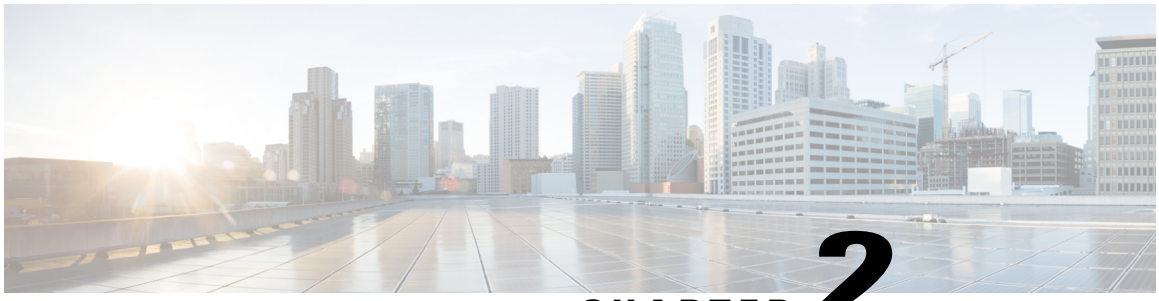
Cisco OSI configures one or more of the following interface types depending on how you configure the deployment:

- A public interface 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 PXE booting the bare-metal node.
- (Optional.) A separate management interface--Used for management tasks if you don't want to put them on the public interface.
- An external interface attached to an upstream devices 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 VMs.

Cisco OSI supports 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 has its own external interface with an IP address associated with the underlying physical network, so that VMs are connected directly to the network that routes external traffic.

The second model is the "tenant network" or "GRE model". In this model, VMs have direct access only to an internal virtual network. In this model, the Neutron controller provides external access by acting as a NAT router, providing PAT for outbound traffic and Floating IPs for inbound traffic.



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 may 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 12.04 LTS (64-bit PC 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
- For standalone rack-mount servers, Cisco Integrated Management Controller, Release 1.5 and later

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 12.

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, remember that the build node acts as the puppet master, caches client components and logs all installation activity. A more powerful machine with more disk space might be required for larger installs.
Control node	Processor: 64-bit x86 Memory: 12 GB (RAM) Disk space: 1 TB (SATA or SAS) Network: One 1-Gbps network interface card (NIC)	Two NICs are recommended but not required for basic functioning. 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: One 1-Gbps NIC	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 3TB (SATA) depending upon the model • For blade servers, 2 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

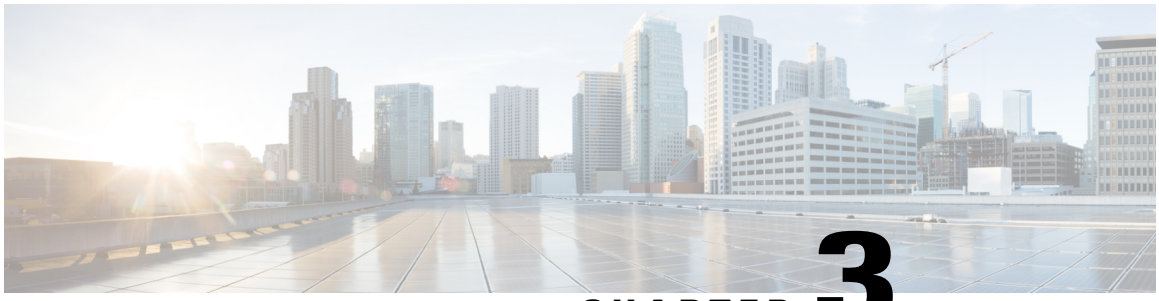
The following table lists the deployment scenarios that are currently available with Cisco OSI:

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, along with the build server. Optionally, you can add more compute-only nodes.	Trying out 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.	Trying 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.	Trying a simplified multi-node installation as in 2_role, 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. This scenario is experimental in h.1 and might require manual intervention to deploy properly.	Deploying production environments. Provides separation of services including dedicated load balancing nodes, storage nodes, compute nodes, and an active/active 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 an active/active highly available control plane with a limited number of nodes and limited scalability.

Note 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.



CHAPTER 3

Installing OpenStack

Summary of Steps for Installing OpenStack

This summary outlines the major steps required to deploy OpenStack using Cisco OSI. It can serve as a checklist for experienced installers.

Step 1 Ensure that your environment meets the system requirements for Cisco OSI as described in Preparing to Install OpenStack, on page 7.

Step 2 Choose a deployment scenario as described in Choosing a Deployment Scenario, on page 9.

Step 3 Create the build server that will serve as the build node for the OpenStack cluster by using the Cisco OSI install script as described in Creating the Build Node, on page 12.

Step 4 Customize the build server as described in About Building the Control and Compute Nodes, on page 18.

Step 5 Build the control and compute nodes as described in About Building the Control and Compute Nodes, on page 18. You can perform this step in one of the following ways:

- Build each control and compute node individually as described in Building the Control and Compute Nodes Individually With Cobbler, on page 19 or Building the Control and Compute Nodes Individually With Puppet Agent, on page 20.
- Build the control and compute nodes using Cobbler as described in Building the Control and Compute Nodes with Cobbler, on page 20.

Step 6 Rerun Puppet on the build node to enable Puppet to run as an agent as described in Rerunning Puppet on the Build Node and Enabling Puppet to Run as an Agent, on page 23.

Step 7 Test the OpenStack deployment as described in Testing the OpenStack Deployment, on page 23.

Creating the Build Server

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. Double-check the names in `/etc/hosts` before proceeding.
- Time synchronize the nodes. Use ntpdate to check this.

If you are using Cobbler to deploy your nodes, then these two steps will be 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 you plan to use as the build node meets the minimum server requirements described in Minimum Server Requirements, on page 8.
- Set your proxy configuration as described in Proxy Configurations, on page 8.
- Validate forward and reverse DNS lookups for the build server IP address and fully qualified domain name (FQDN).

Step 1 Install Ubuntu 12.04 LTS on the server, with the following features:

- Deploy no less than a minimal installation.
- Ensure that 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.

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 may 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` containing 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 as shown here:

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
```

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.

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 havana https://github.com/CiscoSystems/puppet_openstack_builder
cd puppet_openstack_builder
git checkout h.1
```

Step 7 Set Cisco as the vendor by setting the *vendor* environment variable:

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

Step 8 Specify a deployment scenario. See Choosing a Deployment Scenario, on page 9 to determine which scenario is appropriate to your application.

Example:
```
export scenario=<scenario_name>
```

Step 9 Run the install script. This script will install Puppet and prepare modules and repositories on the build node:

Example:
```
cd ~/puppet_openstack_builder/install-scripts
./install.sh 2>&1 | tee install.log
```

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 a log file `/root/puppet_openstack_builder/install-scripts/install.log` containing 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 install.sh only needs to be run 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 23.

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`.

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, instead set the configuration parameters in `/etc/puppet/data/hiera_data/user.common.yaml` and in `/etc/puppet/data/hiera_data/user.scenario.yaml` where scenario is your deployment scenario.

Before You Begin

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

Step 1 Configure host names.

Example:

```
# Set the hostname of the build node
coe::base::build_node_name: build-server

# Set the hostname of the control node
coe::base::controller_hostname: control-server

# Set the IP address of the control node
coe::base::controller_node_internal: controller_ip
```

Note Supply only the short host name to this parameter, not the fully qualified domain name.

Step 2 Specify the software repository (or *repo*) from which you will copy packages. Use the Cisco repo if you have a choice. The cloud archive is not tested as regularly by Cisco engineers and may contain inconsistencies. Use http if possible as your download protocol, especially if you are behind a proxy. Choose either the ftp or http configuration in each case; under no circumstances include both configurations.

Example:

```
# Configure which repository to get packages from. Can be 'cisco_repo' or 'cloud_archive'
# The former is an apt repo maintained by Cisco and the latter is the Ubuntu Cloud Archive
coe::base::package_repo: 'repo_label'

# Which Openstack release to install - can be 'grizzly' or 'havana'. Other versions currently untested.
coe::base::openstack_release: 'havana'

# If cisco_repo is used, the mirror location can be configured as either ftp or http:
# coe::base::openstack_repo_location: 'ftp://ftpeng.cisco.com/openstack'
coe::base::openstack_repo_location: 'http://openstack-repo.cisco.com/openstack'

# Cisco maintains a supplemental repo with packages that aren't core to Openstack, but
# are frequently used, such as ceph and mysql-galera. It can be enabled using ftp or http
# coe::base::supplemental_repo: 'ftp://ftpeng.cisco.com/openstack/cisco_supplemental'
coe::base::supplemental_repo: 'http://openstack-repo.cisco.com/openstack/cisco_supplemental'

# If you are using the ubuntu repo, you can change from 'main' (default) to 'updates'
coe::base::ubuntu_repo: 'updates'

# Set a proxy server
coe::base::proxy: 'proxy_ip'
```

```
# Set a gateway server
coe::base::node_gateway: 'gateway_ip'
```

Step 3 Configure connectivity.

Example:

```
# The DNS Domain
domain: domain.name

# A list of NTP servers
ntp_servers:
  - time-server.domain.name

# Used to tell Neutron agents which IP to bind to.
# We can get the ip address of a particular interface
# using a fact.
internal_ip: "%{ipaddress_eth1}"

# Similarly, VNC needs to be told the IP to bind to
nova::compute::vncserver_proxyclient_address: "%{ipaddress_eth1}"

# This interface will be used to NAT to the outside world. It
# only needs to be on the control node(s)
external_interface: eth2

# This interface is used for communication between openstack
# components, such as the database and message queue
public_interface: eth1

# This interface is used for VM network traffic
private_interface: eth1

# This will be used to tell openstack services where the DB and
# other internally consumed services are
controller_internal_address: controller_ip

# This is used in the HA scenario to set the public keystone
# endpoint
controller_public_address: controller_public_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.

Example:

```
# Users can set either a single password for all services, plus
# the secret token for keystone
secret_key: secret
password: password123

# Or passwords can be specified for each service
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
```

Step 5 Configure disk partitioning.

Example:
```
root_part_size: 65536
var_part_size: 432000
enable_var: true
enable_vol_space: true
```

Step 6 The following advanced options can be provided to your build node to enable special features during bare-metal provisioning. They 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 will be required for all scenarios.

a) To install a specific kernel package and set it to be the kernel booted by default, set this directive:

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

b) To specify command-line options that should be passed to the kernel at boot time, set the kernel_boot_params directive.

Note Using elevator=deadline is recommended for those using ISCSI volumes in Cinder. Some I/O issues have been reported using the default elevator.

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

c) To set the timezone on the clock for nodes booted by using Cobbler, set this directive:

Example:
```
time_zone: US/Your_timezone
```

Step 7 In `/etc/puppet/data/role_mappings.yaml`, map the roles in your selected scenario to host names. The format for each entry is `host_name: role_name`. You must supply an entry for every node in your deployment.

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

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
```

If you selected the all_in_one scenario, the install.sh script will automatically add your build 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 will find 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. Therefore, if one or more nodes have a dependency on an application 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:

- Use Cobbler to provision the node If you are starting from bare metal.
- 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 bare metal. 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 20 to provision the node.

Note Setting up the control node might require more than one Puppet run, especially if there are proxies in the path, because some proxies can have issues with apt-get installations and updates. 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

Your build server should be completely set up 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 bare-metal 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, simply restart the Puppet agent.

Building the Control and Compute Nodes Individually With Puppet Agent

To configure the node without re-installing Ubuntu, you can use the following procedure.

Step 1 If necessary, install git on the node.

Example:
```
apt-get install -y git
```
These steps will set up Puppet and then perform a catalog run.

Step 2 Clone Cisco OSI onto the node.

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

Step 3 Check out the Cisco OSI.

Example:
```
cd puppet_openstack_builder
git checkout h.1
```

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 the Control and Compute Nodes with Cobbler

The 2_role and full_ha scenarios configure a build server that provides Puppet Master and optionally Cobbler bare-metal 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 15.

Step 2 Provide hardware information, including the MAC addresses of the network boot interfaces, machine host names, 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.
Preseed 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 IPMI for power management of UCS C-Series servers. Install the IPMI package if using C-Series servers: `apt-get install -y ipmitool .`

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

b) Verify the profile section.
Profile 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 will need to change several parameters in this section.

Example:
```
node-global:
  profile: "precise-x86_64"
  netboot-enabled: "1"
  power-type: "ipmitool"
  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 *ipmitool* for servers with an IPMI interface, including UCS C-series in standalone mode. Set power_type to *ucs* for B- and C-series servers managed by UCSM.

```
# Power configuration parameters for standalone nodes
  ...
  power_type: "ipmitool"
  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) Finally, 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 sub-organization 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.

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

After puppet has been installed on a node by running install.sh, all further modifications to the node should be made by modifying the model and running Puppet directly.

If you've made changes to the model (for example in `cobbler.yaml` or in a `.yaml` file in `/etc/puppet/data`), the changes will normally 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.

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

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

- To force an update of changes from the build node, restart the Puppet service on the build node.

```
service puppet restart
```

Testing the OpenStack Deployment

After you build the nodes in the OpenStack cluster and the Puppet runs have completed on all nodes, you should test the OpenStack deployment.

Step 1 Log into the OpenStack Horizon interface:

a) In your browser, navigate to `http://control-node-IP/horizon/`

b) 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 2 Load an image into the control node:

a) Log into the console of the control node:

- Username—localadmin
- Password—ubuntu

b) Log in as root.

c) In `/root/` run **source openrc**

d) Launch a test file in `/tmp/test_nova.sh`