Contents

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 an 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.

Prerequisites

- Ensure that the server that you plan to use as the build node meets the minimum server requirements described in Minimum Server Requirements.
- 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).

Procedure

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 OpenStack: Installing Ubuntu Linux or [1]

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

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

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:
`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.
`sudo su -`

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

```
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.

```
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.

d) Reboot the node.

Example:

```
reboot -f
```

e) Sync the build node with a time server.
`ntpdate ntp_server_name`

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

a) Retrieve and install git.

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

b) If necessary, configure git with your proxy.

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

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

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

Step 7: Check out the Cisco OSI.

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

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

```
export vendor=cisco
```

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

```
export scenario=scenario_name
```

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

```
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.

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
```

Note: See Rerunning Puppet on the Build Node and Enabling Puppet to Run as an Agent.

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 lower in the hierarchy. See YAML files and the Hiera Database.

Prerequisites

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

Procedure

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:

```
# local DNS. It doesn't have to be the name of your corporate DNS - a local
# This domain name will be the name your build and compute nodes use for the
# 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) 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 server to something other than build_server, 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_server_name`*`.yaml` file (where *build_server_name* is the short name of your build server) and set them to the appropriate values for your build server.

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.62tHDGB36EhiKF
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.

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.

Procedure

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.

Procedure

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:

```
cd /root
puppet agent --enable
puppet agent -td --server=your_build_node_fqdn --pluginsync
```

```
# Or if you wish to watch the installation live
puppet agent -t -v --no-daemonize --server=your_build_node_fqdn --pluginsync | tee output.log
```

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.

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.

Procedure

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
```

~=~

<- Previous: Installer Prereqs ~=~ ***Next: Testing ->***

Table of Contents ~=~ ***Overview***: *About* ~ *Components* ~ *Deployment* ~ *IP Networks* ~=~ ***Prerequisites***: *System Requirements* ~ *Choosing a Deployment Scenario* ~=~ ***Installing***: *Creating the Build Server* ~ *Building the Control and Compute Nodes* ~=~ ***Testing***: *Verifying Nodes* ~ *Using the Monitoring Interface* ~ *Creating a Network* ~ *Creating a Tenant Instance*