OpenHPC (v1.3)
OHPC on Openstack

CentOS7.3 Base OS

Openstack/SLURM Edition for Linux* (x86_64)

Document Last Update: 2017-04-06
Document Revision: d964afc

Legal Notice

Contents

1 Introduction

Term "HPC as a Service" refers to an on demand instantiation of HPC Service in a Cloud. This guide presents a simple "HPC cluster" instantiation procedures in an existing OpenStack based System. "HPC as a service" relies on two main principals to instantiate HPC service 1. Providing pre-build OS images for compute nodes with HPC optimized software and 2. Uses of Cloud-Init to configure and tune HCP services. This recipe provides a simple guides to build HPC optimized OS images, prepare cloud-init recipes and finally instantiate fully functional HPC System using HPC optimized image and cloud-init. For HPC System recipe instantiate HPC master node (aka sms node) and HPC compute nodes using pre-configured OS images. The terms master and SMS are used interchangeably in this guide. OS Images are build using components from OpenHPC software stack. OpenHPC represents an aggregation of a number of common ingredients required to deploy and manage an HPC Linux* cluster including resource management, I/O clients, development tools, and a variety of scientific libraries. These packages have been pre-built with HPC integration in mind using a mix of open-source components. The documentation herein is intended to be reasonably generic, but uses the underlying motivation of a small, 4-node statefull cluster installation to define a step-by-step process. Several optional customizations are included and the intent is that these collective instructions can be modified as needed for local site customizations. Base Linux Edition: this edition of the guide highlights installation without the use of a companion configuration management system and directly uses distro-provided package management tools for component selection. The steps that follow also highlight specific changes to system configuration files that are required as part of the cluster install process.

Base Linux Edition: this edition of the guide highlights installation without the use of a companion configuration management system and directly uses distro-provided package management tools for component selection. The steps that follow also highlight specific changes to system configuration files that are required as part of the cluster install process.

1.1 Target Audience

This guide is targeted at experienced Linux system administrators for HPC environments. Knowledge of software package management, system networking, PXE booting and OpenStack system software is assumed. Command-line input examples are highlighted throughout this guide via the following syntax:

```
[sms]# echo "OpenHPC hello world"
```

Unless specified otherwise, the examples presented are executed with elevated (root) privileges. The examples also presume use of the BASH login shell, though the equivalent commands in other shells can be substituted. In addition to specific command-line instructions called out in this guide, an alternate convention is used to highlight potentially useful tips or optional configuration options. These tips are highlighted via the following format:

Tip

The solution is to increase the size of the manuals. –Mark V. Shaney

1.2 Requirements/Assumptions

This installation recipe assumes the availability of an OpenStack controller (+ network) node four bare metal nodess. The Controller node serve as central controller for OpenStack services and has all the required OpenStack services installed and configured (i.e. keystone, nova, neutron, ironic along with their dependent services) to provision bare metal nodes with CentOS7.2 in a statefull configuration.

This recipe is tested with OpenStack Mitaka release with CentOS 7.2. This document provides some examples for installing and configuring OpenStack using Mitaka release of packstack from RedHatÂő. More detail on using packstack can be found https://www.rdoproject.org/install/quickstart/.

For power management, we assume that the bare metal node baseboard management controllers (BMCs) are available via IPMI from the chosen controller host. For file systems, we assume that the master server (instantiated during provisioning âĂIJHPC as a serviceâĂİ) will host an NFS file system that is made available to the HPC compute nodes.

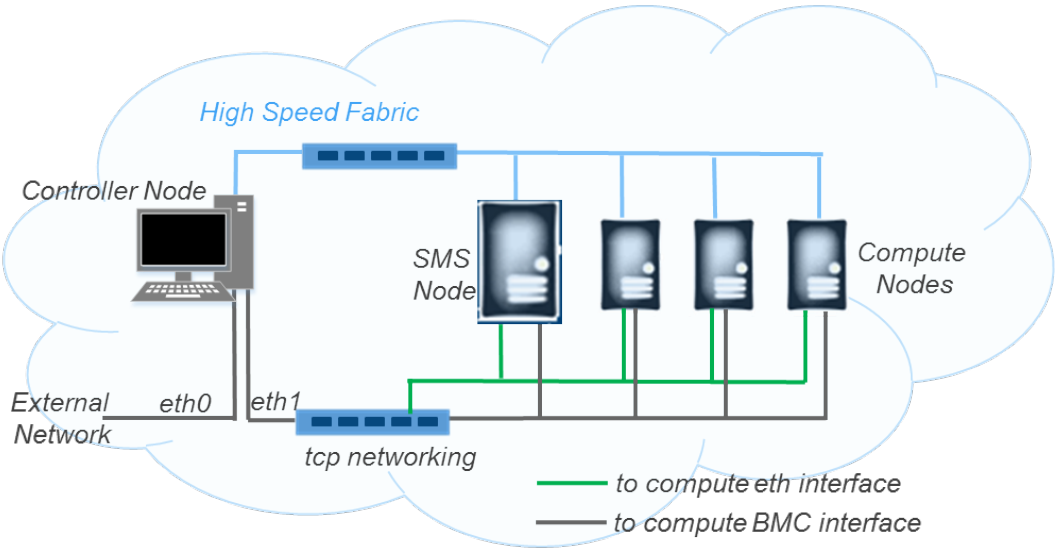

Figure 1: Overview of physical cluster architecture.

An outline of the physical architecture discussed is shown in Figure 1 and highlights the high-level networking configuration. The *master* host requires at least two Ethernet interfaces with *eth0* connected to the local data center network and *eth1* used to provision and manage the cluster backend (note that these interface names are examples and may be different depending on local settings and OS conventions). Two logical IP interfaces are expected to each compute node: the first is the standard Ethernet interface that will be used for provisioning and resource management. The second is used to connect to each host's BMC and is used for power management and remote console access. Physical connectivity for these two logical IP networks is often accommodated via separate cabling and switching infrastructure; however, an alternate configuration can also be accommodated via the use of a shared NIC, which runs a packet filter to divert management packets between the host and BMC.

In addition to the IP networking, there is a high-speed network (InfiniBand in this recipe) that is also connected to each of the hosts. This high speed network is used for application message passing and optionally for Lustre connectivity as well.

NOTE: This recipe sets various environment variables in one section and use them in other section. So users are expected to use single shell session for successful execution of this recipe. Appendix A provides reference to prebuilt recipe, and useful for users who wants to try out with minimum human interactions.

1.3 Inputs

As this recipe details installing a cluster starting from bare-metal, there is a requirement to define IP addresses and gather hardware MAC addresses in order to support a controlled provisioning process. These

values are necessarily unique to the hardware being used, and this document uses variable substitution (`${variable}`) in the command-line examples that follow to highlight where local site inputs are required. A summary of the required and optional variables used throughout this recipe are presented below. Note that while the example definitions above correspond to a small 4-node compute subsystem, the compute parameters are defined in array format to accommodate logical extension to larger node counts.

- `${sms_name}` # Hostname for SMS server
- `${sms_ip}` # Internal IP address on SMS server
- `${sms_eth_internal}` # Internal Ethernet interface on SMS
- `${eth_provision}` # Provisioning interface for computes
- `${internal_netmask}` # Subnet netmask for internal network
- `${ntp_server}` # Local ntp server for time synchronization
- `${bmc_username}` # BMC username for use by IPMI
- `${bmc_password}` # BMC password for use by IPMI
- `${num_computes}` # Total # of desired compute nodes
- `${c_ip[0]}`, `${c_ip[1]}`, ... # Desired compute node addresses
- `${c_bmc[0]}`, `${c_bmc[1]}`, ... # BMC addresses for computes
- `${c_mac[0]}`, `${c_mac[1]}`, ... # MAC addresses for computes
- `${compute_regex}` # Regex matching all compute node names (e.g. "c*")
- `${compute_prefix}` # Prefix for compute node names (e.g. "c")
-

Optional:

- `${mgs_fs_name}` # Lustre MGS mount name
- `${sms_ipoib}` # IPoIB address for SMS server
- `${ipoib_netmask}` # Subnet netmask for internal IPoIB
- `${c_ipoib[0]}`, `${c_ipoib[1]}`, ... # IPoIB addresses for computes
- `${kargs}` # Kernel boot arguments

2 Preparing Bare Metal Node Operating System

This guide uses diskimage-builder utility to build and configure OS images for the sms (controller) node as well as compute nodes. Preparing images is an optional part of the overall recipe. If user has predefined images, then environment variable "chpc_create_new_image" must be reset and path to images must be provided using environment variable "chpc_image_deploy_kernel", "chpc_image_deploy_ramdisk", "chpc_image_user", and "chpc_image_sms".

In this example cloud images are built on controller node ("[ctrlr]# "). Once images are built, they are stored in the standard openHPC Path.

```
[ctrlr]# CHPC_CLOUD_IMAGE_PATH=/opt/ohpc/admin/images/cloud/
```

2.1 Install and Setup diskimage-builder

Images can be built on any supported OS. In this example we will install and build images on controller node, user can do same on a system independent of their production cluster. Install diskimage-builder and its dependencies from base OS distro

```
[ctrlr]# yum -y install diskimage-builder PyYAML
```

Install grub dependency

```
[ctrlr]# yum -y install parted
```

Diskimage-builder installed from base distro does not have group install feature. So add a patch (Note: this probably will be included into RPM and will be part of that rpm installation)

```
[ctrlr]# yum -y install <DIB patch>
```

2.2 Setup common environment for diskimage-builder

diskimage-builder or dib uses environment variables and elements to customize the images. For debugging purpose, we will create default user chpc with a password intel8086, with sudo privilege. These variables are used by element devuser.

```
[ctrlr]# export DIB_DEV_USER_USERNAME=chpc
[ctrlr]# export DIB_DEV_USER_PASSWORD=intel8086
[ctrlr]# export DIB_DEV_USER_PWDLESS_SUDO=1
```

Now add path to custom elements which are not part of base diskimage-builder. OpenHPC provides few HPC elements. [Note: This also can be part of openHPC provided rpm for dib. In that case remove this step]

```
[ctrlr]# export ELEMENTS_PATH="$(realpath ../../dib/hpc/elements)"
```

Add path to HPC specific files [note: same as earlier, this too can be part of rpm package]

```
[ctrlr]# export DIB_HPC_FILE_PATH="$(realpath ../../dib/hpc/hpc-files/)"
```

HPC elements are common for OpenHPC and Intel HPC Orchestrator, environment variable "DIB_HPC_BASE" tell dib which one to pick. For OpenHPC set environment variable

```
[ctrlr]# export DIB_ HPC_BASE="ohpc"
```

Make sure open hpc packages is installed. ohpc_pkg is one of the input setup earlier in this document.

```
[ctrlr]# yum -y install ${ohpc_pkg}
```

Export same to DIB.

```
[ctrlr]# export DIB_HPC_OHPC_PKG=${ohpc_pkg}
```

Create list of HPC elements needed to build HPC images, by starting hpc-env-base. This element will setup basic hpc environment to build hpc images.

```
[ctrlr]# DIB_HPC_ELEMENTS="hpc-env-base"
```

2.3 Preparing ironic deploy images

Ironic uses deploy images (aka kernel) to bootstrap the provisioning of user images. Unset any previous environment flag

```
[ctrlr]# DIB_YUM_REPO_CONF
```

Git is used by some of the elements in diskimage-builder.

```
[ctrlr]# yum -y install git
```

Create deploy images using disk-image-create cli. This will download base centos image from distro, install ironic-agent on it and create kernel and initiramfs images.

```
[ctrlr]# disk-image-create ironic-agent centos7 -o icloud-hpc-deploy-c7
```

Move deploy images to ohpc standard path.

```
[ctrlr]# chpc_image_deploy_kernel="$( realpath icloud-hpc-deploy-c7.kernel)"
[ctrlr]# chpc_image_deploy_ramdisk="$( realpath icloud-hpc-deploy-c7.initramfs)"
 #Store Images file
[ctrlr]# mkdir -p $CHPC_CLOUD_IMAGE_PATH/
[ctrlr]# sudo mv -f $chpc_image_deploy_kernel $CHPC_CLOUD_IMAGE_PATH/
[ctrlr]# chpc_image_deploy_kernel=$CHPC_CLOUD_IMAGE_PATH/$(basename $chpc_image_deploy_kernel)
[ctrlr]# sudo mv -f $chpc_image_deploy_ramdisk $CHPC_CLOUD_IMAGE_PATH/
[ctrlr]# chpc_image_deploy_ramdisk=$CHPC_CLOUD_IMAGE_PATH/$(basename $chpc_image_deploy_ramdisk)
```

2.4 Preparing user images for bare metal instances

For "HPC as a Service" we will building 2 user images (1 for sms node and 1 for compute node) and 2 deploy images. User images we build here will be customized with OpenHPC using HPC specific elements.

2.4.1 Preparing user image for head node OS

To build head node (aka sms) images, we need to install server packages of HPC components. This is accomplished by setting image type to sms. Default image type in hpc elements is âĂIJcomputeâĂİ.

```
[ctrlr]# export DIB\_HPC\_IMAGE\_TYPE=sms
```

Now enable SLURM resource manager for head node.

```
[ctrlr]# DIB\_HPC\_ELEMENTS+=" hpc-slurm"
```

Add optional OpenHPC Components

```
[ctrlr]# if [[ \$\{enable\_mrsh\} -eq 1 ]];then
[ctrlr]#       DIB\_HPC\_ELEMENTS+=" hpc-mrsh"
[ctrlr]# fi
```

We will also setup HPC development environment on HPC head node. Enable gnu compiler

```
[ctrlr]# export DIB\_HPC\_COMPILER="gnu"
```

Enable openmpi & mvapich2

```
[ctrlr]# export DIB\_HPC\_MPI="openmpi mvapich2"
```

Enable performance tools

```
[ctrlr](*#*) export DIB\_HPC\_PERF\_TOOLS="perf-tools"
```

Enable 3rd party libraries serial-libs, parallel-libs, io-libs, python-libs and runtimes

```
[ctrlr]# export DIB\_HPC\_3RD\_LIBS="serial-libs parallel-libs io-libs python-libs runtimes"
```

Add hpc development environment element to list of elements

```
[ctrlr]# DIB\_HPC\_ELEMENTS+=" hpc-dev-env"
```

Now create a sms image with element local-config, dhcp-all-interfaces, devuser, selinux-permisive and all hpc specific elements. Element local-config copies your local environment into image, which is the local users, their password and permissions. Element devuser will create new user specified by environment variable "DIB_DEV_USER_USERNAME".

```
[ctrlr]# disk-image-create centos7 vm local-config dhcp-all-interfaces devuser \
 selinux-permissive \$DIB\_HPC\_ELEMENTS -o icloud-hpc-cent7-sms
```

It will take a while to build an image. Once image is built copy it to standard openHPC path.

```
[ctrlr]# chpc\_image\_sms="\$( realpath icloud-hpc-cent7.qcow2)"
[ctrlr]# mkdir -p \$CHPC\_CLOUD\_IMAGE\_PATH
[ctrlr]# mv -f \$chpc\_image\_sms \$CHPC\_CLOUD\_IMAGE\_PATH
[ctrlr]# chpc\_image\_sms=\$CHPC\_CLOUD\_IMAGE\_PATH/\$(basename \$chpc\_image\_sms)
```

2.4.2 Preparing user image for compute node OS

To build compute node images, we need to install client packages of HPC components. This is accomplished by setting image type to compute. Default image type in hpc elements is "compute".

```
[ctrlr]# export DIB_HPC_IMAGE_TYPE=compute
```

Now enable SLURM resource manager for compute node.

```
[ctrlr]# DIB_HPC_ELEMENTS+=" hpc-slurm"
```

Add optional OpenHPC Components

```
[ctrlr]# if [[ ${enable_mrsh} -eq 1 ]];then
[ctrlr]#       DIB_HPC_ELEMENTS+=" hpc-mrsh"
[ctrlr]# fi
```

Now create a compute node image with element local-config, dhcp-all-interfaces, devuser, selinux-permisive and all hpc specific elements. Element local-config copies your local environment into image, which is the local users, their password and permissions. Element devuser will create new user specified by environment variable "DIB_DEV_USER_USERNAME".

```
[ctrlr]# disk-image-create centos7 vm local-config dhcp-all-interfaces devuser \
 selinux-permissive $DIB_HPC_ELEMENTS -o icloud-hpc-cent7-sms
```

It will take a while to build an image. Once image is built, copy it to standard OpenHPC path.

```
[ctrlr]# chpc_image_sms="$( realpath icloud-hpc-cent7.qcow2)"
[ctrlr]# mkdir -p $CHPC_CLOUD_IMAGE_PATH
[ctrlr]# mv -f $chpc_image_user$CHPC_CLOUD_IMAGE_PATH
[ctrlr]# chpc_image_user=$CHPC_CLOUD_IMAGE_PATH/$(basename $chpc_image_sms)
```

2.5 Introduction to diskimage-builder

It is a utility to build and configure OS images for sms node as well compute node. It uses prebuild minimum OS images from base distro, which it further customizes as per user request. Diskimage-builder is a framework which uses many elements (similar to plug-ins) to customize the image. Base distribution of diskimage-builder comes with pre-defined elements. This recipe uses additional HPC elements which were built to customize images based on OpenHPC components.

2.5.1 HPC Elements to build OpenHPC Images

"HPC as a service" uses 4 HPC specific elements in addition to pre-packaged elements comes with diskimage-builder.

hpc-dev-env: This is mainly used to create sms images to create HPC development environment. It creates hpc development environment by installing following OpenHPC components within image: ohpc-autotools, valgrind-ohpc, easybuild-ohpc, spack-ohpc, r_base-ohpc mpi and compiler for chosen MPI and compiler via environment variable, $DIB_HPC_COMPILER, $DIB_HPC_MPI Performance Tools lmod-default with their 3rd party libraries.

hpc-env-base: TBC

hpc-mrsh:TBC

hpc-slurm: TBC

2.5.2 Editing HPC Elements

TBC

3 Preparing Cloud-Init

OpenStack uses cloud-init for boot time initialization of cloud instances. This recipe relies on cloud-init to initialize HPC instances in an OpenStack cloud. This recipe prepares cloud-init initialization template script, which is than updated with sms-ip and other environment variables just before the provisioning. This is than fed as user data to Nova during instance creation. Script generated here will be executed by root during boot.

3.1 Preparing template for compute node cloud-init

Create an empty chpc_init file and open for editing. You can also use existing template and modify.

Start editing by adding some environment variable, first one is to set path to shared folder for cloud-init

```
chpcInitPath=/opt/ohpc/admin/cloud_hpc_init
logger "chpcInit: Updating Compute Node with HPC configuration"
```

Update rsyslog configuration file to send all the syslog to sms. sms_ip is the tag used here is updated with IP of SMS node just before provisioning.

```
# Update rsyslog
cat /etc/rsyslog.conf | grep "<sms_ip>:514"
rsyslog_set=$?
if [ "${rsyslog_set}" -ne "0" ]; then
    echo "*.* @<sms_ip>:514" >> /etc/rsyslog.conf
fi
systemctl restart rsyslog
logger "chpcInit: rsyslog configuration complete, updating remaining HPC configuration"
```

Assuming sms node nfs share /home, /opt/ohpc/pub, =/opt/ohpc/admin/cloud_hpc_init lets mount them during boot

```
# nfs mount directory from SMS head node to Compute Node
cat /etc/fstab | grep "<sms_ip>:/home"
home_exists=$?
if [ "${home_exists}" -ne "0" ]; then
    echo "<sms_ip>:/home /home nfs nfsvers=3,rsize=1024,wsize=1024,cto 0 0" >> /etc/fstab
fi
cat /etc/fstab | grep "<sms_ip>:/opt/ohpc/pub"
ohpc_pub_exists=$?

if [ "${ohpc_pub_exists}" -ne "0" ]; then
    echo "<sms_ip>:/opt/ohpc/pub /opt/ohpc/pub nfs nfsvers=3 0 0" >> /etc/fstab
    # Make sure we have directory to mount
    # Clean up if required
    if [ -e /opt/ohpc/pub ]; then
        echo "chpcInit: [WARNING] /opt/ohpc/pub already exists!!"
    fi
fi
mkdir -p /opt/ohpc/pub
mount /home
mount /opt/ohpc/pub

# mount cloud_hpc_init
cat /etc/fstab | grep "sms_ip:$chpcInitPath"
CloudHPCInit_exist=$?
if [ "${CloudHPCInit_exist}" -ne "0" ]; then
    echo "<sms_ip>:$chpcInitPath $chpcInitPath nfs nfsvers=3 0 0" >> /etc/fstab
fi
```

```
mkdir -p $chpcInitPath
mount $chpcInitPath
# Restart nfs
systemctl restart nfs

have ntp sync with sms node.
# Restart ntp at CN
systemctl enable ntpd
# Update ntp server
cat /etc/ntp.conf | grep "server <sms_ip>"
ntp_server_exists=$?
if [ "${ntp_server_exists}" -ne "0" ]; then
    echo "server <sms_ip>" >> /etc/ntp.conf
fi
systemctl restart ntpd
# time sync
Ntpstat
Sync sms node with compute nodes. sync users, slurm and enable munge by copying munge keys
# Sync following files to compute node
# Assuming nfs is setup properly
if [ -d $chpcInitPath ]; then
    # Update the slurm file
    cp -f -L $chpcInitPath/slurm.conf /etc/slurm/slurm.conf
    # Sync head node configuration with Compute Node
    cp -f -L $chpcInitPath/passwd /etc/passwd
    cp -f -L $chpcInitPath/group /etc/group
    cp -f -L $chpcInitPath/shadow /etc/shadow
    cp -f -L $chpcInitPath/slurm.conf /etc/slurm/slurm.conf
    cp -f -L $chpcInitPath/slurm /etc/pam.d/slurm
    cp -f -L $chpcInitPath/munge.key /etc/munge/munge.key
    # For hostname resolution
    cp -f -L $chpcInitPath/hosts /etc/hosts
    # make sure that hostname mentioned into /etc/hosts matches machine hostname. TBD
    # Start slurm and munge
    systemctl enable munge
    systemctl restart munge
    systemctl enable slurmd
    systemctl restart slurmd
else
    logger "chpcInit:ERROR: cannot stat nfs shared /opt directory, cannot copy HPC system files"
fi
```

Update the hostname as per sms node.

```
# Setup hostname as per the head node
#Find the hostname of this machine from the copied over /etc/hosts file
cc_ipaddrs=(‘hostname -I‘)
for cc_ipaddr in ${cc_ipaddrs[@]}; do
    cat /etc/hosts | grep ${cc_ipaddr} > /dev/null
    result=$?
    if [ "$result" -eq "0" ]; then
        cc_hostname=‘cat /etc/hosts | grep ${cc_ipaddr} | cut -d$’\t’ -f2‘
        break
    fi
done

if [ -z "${cc_hostname}" ]; then
    logger "chpcInit:ERROR: No resolved hostname found for any IP address in /etc/hosts"
    exit 1
fi

#set the hostname
```

```
if [ $(hostname) != ${cc_hostname} ]; then
    hostnamectl set-hostname ${cc_hostname}
fi
```

By now all pre-requisite for slurm is taken care, lets start slurm daemon.

```
# Start slurm and munge
systemctl enable munge
systemctl restart munge
systemctl enable slurmd
systemctl restart slurmd
```

One last step to make sure ssh is working and enabled on compute nodes. Update the permissions of ssh.

```
#Change file permissions in /etc/ssh to fix ssh into compute node
chmod 0600 /etc/ssh/ssh_host_*_key
```

Save the file with name chp_cinit, we will use this file during baremetal node instance creation.

3.2 Preparing template for sms node cloud-init

Cloud init script for sms node is little different than compute node. Sms node, when instantiated within openstack, serve as a head node for HPC as a service and hosts all the services as a sms node in an independent hpc clusters. This will host server side of applications, resource manager, and share users. For more detail on sms node functionality please refer to OpenHPC documentation.

In this recipe, we will prepare cloud-init template script for sms node, which than is updated with compute node IP, ntp server and other environmental variables, just before provisioning. Create an empty chpc_init file and open for editing. You can also use existing template and modify. Start editing by adding some environment variable, which will be updated later, just before provisioning.

```
# Get the Compute node prefix and number of compute nodes
cnodename_prefix=<update_cnodename_prefix>
num_ccomputes=<update_num_ccomputes>
ntp_server=<update_ntp_server>
sms_name=<update_sms_name>
```

Now setup nfs share for cloud-init and files which want to send to compute nodes.

```
# setup cloudinit directory
chpcInitPath=/opt/ohpc/admin/cloud_hpc_init
# create directory of not exists
mkdir -p $chpcInitPath
chmod 700 $chpcInitPath
To create same user environment, copy user files
# Copy other files needed for Cloud Init
sudo cp -fpr /etc/passwd $chpcInitPath
sudo cp -fpr /etc/shadow $chpcInitPath
sudo cp -fpr /etc/group $chpcInitPath
```

Share /home, /opt/ohpc/pub and /opt/ohpc/admin/cloud_hpc_init over nfs

```
# export CloudInit Path to nfs share
cat /etc/exports | grep "$chpcInitPath"
chpcInitPath_exported=$?

if [ "${chpcInitPath_exported}" -ne "0" ]; then
```

```
    echo "$chpcInitPath *(rw,no_subtree_check,no_root_squash)" >> /etc/exports
fi
# share /home from HN
if ! grep "^/home" /etc/exports; then
    echo "/home *(rw,no_subtree_check,fsid=10,no_root_squash)" >> /etc/exports
fi
# share /opt/ from HN
if ! grep "^/opt/ohpc/pub" /etc/exports; then
    echo "/opt/ohpc/pub *(ro,no_subtree_check,fsid=11)" >> /etc/exports
fi
exportfs -a
# Restart nfs
systemctl restart nfs
systemctl enable nfs-server
logger "chpcInit: nfs configuration complete, updating remaining HPC configuration"
```

Configure ntp sever on sms node, as per the site setting.

```
# configure NTP
systemctl enable ntpd
if [[ ! -z "$ntp_server" ]]; then
   echo "server ${ntp_server}" >> /etc/ntp.conf
fi
systemctl restart ntpd
systemctl enable ntpd.service
# time sync
ntpstat
logger "chpcInit:ntp configuration done"
```

Distribute munge keys with compute nodes and then Update SLURM resource manager with hpc compute nodes.

```
### Update Resource manager configuration ###
# Update basic slurm configuration at sms node
perl -pi -e "s/ControlMachine=\S+/ControlMachine=${sms_name}/" /etc/slurm/slurm.conf
perl -pi -e "s/^NodeName=(\S+)/NodeName=${cnodename_prefix}[1-${num_ccomputes}]/" /etc/slurm/slurm.conf
perl -pi -e "s/^PartitionName=normal Nodes=(\S+)/PartitionName=normal \
Nodes=${cnodename_prefix}[1-${num_ccomputes}]/" /etc/slurm/slurm.conf
# copy slurm file from sms node to Cloud Comute Nodes
cp -fpr -L /etc/slurm/slurm.conf $chpcInitPath
cp -fpr -L /etc/pam.d/slurm $chpcInitPath
cp -fpr -L /etc/munge/munge.key $chpcInitPath
# Start slurm and munge
systemctl enable munge
systemctl restart munge
systemctl enable slurmctld
systemctl restart slurmctld
#systemctl enable slurmd
#systemctl restart slurmd
logger "chpcInit:slurm configuration done"
```

One last step to make sure ssh is working and enabled on compute nodes. Update the permissions of ssh.

```
#Change file permissions in /etc/ssh to fix ssh into compute node
chmod 0600 /etc/ssh/ssh_host_*_key
```

Save the file with name chp_sms_cinit, we will use this file during sms node instance creation. Prepare optional part of cloud-init Update mrsh during cloud-init Create a new file sms/update_mrsh and add mrsh configuration to enable mrsh on sms node. And save it.

```
# Update mrsh
# check if it is already configured grep mshell /etc/services will return non-zero, else configure"
cat /etc/services | grep mshell
mshell_exists=$?
if [ "${mshell_exists}" -ne "0" ]; then \
    echo "mshell        21212/tcp    \
    # mrshd" >> /etc/services
fi
cat /etc/services | grep mlogin
mlogin_exists=$?
if [ "${mlogin_exists}" -ne "0" ]; then \
    echo "mlogin          541/tcp    \
    # mrlogind" >> /etc/services
fi
```

Updating cluster shell during cloud -init Create a new file sms/update_clustershell and add configuration to enable clustershell on sms node. And save it.

```
sed -i -- 's/all: @adm,@compute/compute: cc[1-${num_ccomputes}]\n&/' /etc/clustershell/groups.d/local.cfg
```

3.3 Prepare optional part of cloud-init

3.3.1 Update mrsh during cloud-init

Create a new file sms/update_mrsh and add mrsh configuration to enable mrsh on sms node. And save it.

```
# Update mrsh
# check if it is already configured grep mshell /etc/services will return non-zero, else configure"
cat /etc/services | grep mshell
mshell_exists=$?
if [ "${mshell_exists}" -ne "0" ]; then
    echo "mshell        21212/tcp               # mrshd" >> /etc/services
fi
cat /etc/services | grep mlogin
mlogin_exists=$?
if [ "${mlogin_exists}" -ne "0" ]; then
    echo "mlogin          541/tcp               # mrlogind" >> /etc/services
fi
```

3.3.2 Updating cluster shell during cloud -init

Create a new file sms/update_clustershell and add configuration to enable clustershell on sms node. And save it.

```
sed -i -- 's/all: @adm,@compute/compute: cc[1-${num_ccomputes}]\n&/' /etc/clustershell/groups.d/local.cfg
```

3.4 Configuring overall cloud-init

In previous section we created template for cloud-init for hpc head node and hpc compute nodes. We need to update these template with user defined inputs like IP Address, node names. With these updates, cloud-init script is ready to deploy with OpenStack Nova.

Copy cloud-init template to working folder

```
chpcInitPath=/opt/ohpc/admin/cloud_hpc_init
# if directory exists then mv to Old directory. TBD
mkdir -p $chpcInitPath
#copy Cloud HPC files to temp working directory
sudo cp -fr -L < ${SCRIPTDIR} >/ cloud_hpc_init/${chpc_base}/* $chpcInitPath/
export chpcInit=$chpcInitPath/chpc_init
export chpcSMSInit=$chpcInitPath/chpc_sms_init
```

Update sms_ip in compute node cloud-init template with HPC head node.

```
sudo sed -i -e "s/<sms_ip>/${sms_ip}/g" $chpcInit
```

Update HPC head node cloud-init template with compute name prefix as defined by user

```
sudo sed -i -e "s/<update_cnodename_prefix>/${cnodename_prefix}/g" $chpcSMSInit
sudo sed -i -e "s/<update_num_ccomputes>/${num_ccomputes}/g" $chpcSMSInit
Update hostname of HPC head node & NTP server information
sudo sed -i -e "s/<update_ntp_server>/${controller_ip}/g" $chpcSMSInit
sudo sed -i -e "s/<update_sms_name>/${sms_name}/g" $chpcSMSInit
```

Optionally if user enabled mrsh or clusteshell, then update cloud-init accordingly

```
if [[ ${enable_mrsh} -eq 1 ]];then
   # update mrsh for sms node
   cat $CHPC_SCRIPTDIR/sms/update_mrsh >> $chpcSMSInit
fi
if [[ ${enable_clustershell} -eq 1 ]];then
   # update clustershell for sms node
   cat $CHPC_SCRIPTDIR/sms/update_clustershell >> $chpcSMSInit
fi
```

4 Instantiating OpenHPC System in OpenStack Cloud

To instantiate OpenHPC system, we will first prepare openstack components with HPC images, networking and other relevant configurations. After the configuration we will instantiate HPC head node and HPC compute node using nova.

It is assumed that system admin has installed OpenStack controller services and OpenStack network services (i.e. keystone, nova, ironic, glance, neutron, mongodb, rabbitmq server, heat etc). Controller node is configured with OpenVSwitch Bridge on internal network port. Two tenants, named `admin` and `services` are created in keystone to manage the services. All the services are created by system admin.

Below is expected endpoint list.

```
Openstack service list
+----------------------------------+-----------+---------------+---------------+
| ID                                 | Region   | Service Name | Service Type |
+----------------------------------+-----------+---------------+---------------+
| d5aeeb54713745c29ed3c2e4a97f59bd | RegionOne | ironic  | baremetal    |
| 86c71badbf8b4446a1b699eef05f3f41 | RegionOne | nova    | compute      |
| 70d138db26214d0bbc6b3ade8bf6f6f8 | RegionOne | gnocchi | metric       |
| f34c3a58b9c648aaacabeeefd589a0d2 | RegionOne | neutron | network      |
| 789c3fb6f9ae4e249ee4023484ccb5fc | RegionOne | aodh    | alarming     |
| 2531392e2d084b4582b364572e79a7b5 | RegionOne | heat    | orchestration |
| c183b73f654e454eaf5784c4b98149d8 | RegionOne | Image Service | image |
| 850b3c2943df4dca99338ff2013f657b | RegionOne | cinder  | volume       |
| 81cefa79212a4780abe5a1da281a0172 | RegionOne | novav3  | computev3    |
| 36a6a7c7968a4a94bea07c8e30fa5c4b | RegionOne | keystone | identity    |
| db70a8676dd44dd09b3ada7475e67383 | RegionOne | cinderv3 | volumev3    |
| c4161d4c9c6b4080b2cb66c2f580853d | RegionOne | ceilometer | metering |
| d5714e8adb094671ad0388d04214c44d | RegionOne | cinderv2 | volumev2    |
+----------------------------------+-----------+---------------+---------------+

openstack project list
+----------------------------------+----------+
| ID                                 | Name    |
+----------------------------------+----------+
| 7464fcc8f1b34048bd09fe165d18647b | admin |
| b1ed7efb53cc44c8b06daaee15b6a296 | services |
+----------------------------------+----------+
```

Recipe below is tested with controller node installed and configured using packstack. Reference section provide more detail on packstack installation of OpenStack. Before starting with HPC instantiation, please export openstack credential as a root or system admin, we will be using them during openstack configuration.

```
unset OS_SERVICE_TOKEN
export OS_USERNAME=admin
export OS_PASSWORD=<>
export OS_AUTH_URL=<>
export PS1='[\u@\h \W(keystone_admin)]\$ '

export OS_TENANT_NAME=admin
export OS_REGION_NAME=<>
```

4.1 Prepare OpenStack for bare metal provisioning with ironic

This section we will create generic configuration, required for baremetal provisioning. We will use ironic as a provisioner and nova as a scheduler. Have selinux in permissive mode Setenforce 0

Create baremetal admin and baremetal observer role, and restart ironic API

```
openstack role list | grep -i baremetal_admin
role_exists=$?
if [ "${role_exists}" -ne "0" ]; then
    openstack role create baremetal_admin
fi

openstack role list | grep -i baremetal_observer
role_exists=$?
if [ "${role_exists}" -ne "0" ]; then
    openstack role create baremetal_observer
fi
systemctl restart openstack-ironic-api
```

Install tftp and other packages required for pxe boot via ironinc Ensure the utilities for baremetal are installed

```
yum install -y tftp-server syslinux-tftpboot xinetd
```

Make the directory for tftp and give it the ironic owner

```
mkdir -p /tftpboot
chown -R ironic /tftpboot
```

Configure tfpt server

```
#Configure tftp
#Configure /etc/xinet.d/tftp
echo "service tftp" > /etc/xinetd.d/tftp
echo "{" >> /etc/xinetd.d/tftp
echo " protocol      = udp" >> /etc/xinetd.d/tftp
echo " port          = 69" >> /etc/xinetd.d/tftp
echo " socket_type   = dgram" >> /etc/xinetd.d/tftp
echo " wait          = yes" >> /etc/xinetd.d/tftp
echo " user          = root" >> /etc/xinetd.d/tftp
echo " server        = /usr/sbin/in.tftpd" >> /etc/xinetd.d/tftp
echo " server_args   = -v -v -v -v -v --map-file /tftpboot/map-file /tftpboot" >> /etc/xinetd.d/tftp
echo " disable       = no" >> /etc/xinetd.d/tftp
echo " # This is a workaround for Fedora, where TFTP will listen only on" >> /etc/xinetd.d/tftp
echo " # IPv6 endpoint, if IPv4 flag is not used." >> /etc/xinetd.d/tftp
echo " flags         = IPv4" >> /etc/xinetd.d/tftp
echo "}" >> /etc/xinetd.d/tftp

#Restart the xinetd service
systemctl restart xinetd

#Copy the PXE linux files to the tftpboot directory we created
cp /var/lib/tftpboot/pxelinux.0 /tftpboot
cp /var/lib/tftpboot/chain.c32 /tftpboot

#Generate a map file for the PXE files
echo 're ^(/tftpboot/) /tftpboot/\2' > /tftpboot/map-file
echo 're ^/tftpboot/ /tftpboot/' >> /tftpboot/map-file
echo 're ^(^/) /tftpboot/\1' >> /tftpboot/map-file
```

```
echo 're ^([^/]) /tftpboot/\1' >> /tftpboot/map-file
```

Update Ironic configuration with tftp information. First update controller IP address for tftp server in ironic configuration.

```
sed --in-place "s|#tftp_server=\$my_ip|tftp_server=${controller_ip}|" /etc/ironic/ironic.conf
```

Update other additional tftp settings in ironic configuration file:

```
sed --in-place "s|#tftp_root=/tftpboot|tftp_root=/tftpboot|" /etc/ironic/ironic.conf
sed --in-place "s|#ip_version=4|ip_version=4|" /etc/ironic/ironic.conf
sed --in-place "s|#automated_clean=true|automated_clean=false|" /etc/ironic/ironic.conf
```

Now inform Nova to use ironic for bare metal provisioning, by configuring NOVA.conf

```
sed --in-place "s|#scheduler_use_baremetal_filters=false|scheduler_use_baremetal_filters=true|" \
 /etc/nova/nova.conf
```

In our sample, we will not use controller node for any compute resource so, lets mark reserved host memory as 0.

```
sed --in-place "s|reserved_host_memory_mb=512|reserved_host_memory_mb=0|" /etc/nova/nova.conf
sed --in-place "s|#scheduler_host_subset_size=1|scheduler_host_subset_size=9999999|" /etc/nova/nova.conf
```

For cloud-init we need to enable meta data server, which is done via neutron configuration.

```
# Enable meta data
# Edit /etc/neutron/dhcp_agent.ini
sed --in-place "s|enable_isolated_metadata\ =\ False|enable_isolated_metadata\ =\ True|" \
 /etc/neutron/dhcp_agent.ini
sed --in-place "s|#force_metadata\ =\ false|force_metadata\ =\ True|" \ /etc/neutron/dhcp_agent.ini
```

We will enable internal dns server to assign host name to the instances as requested by user.

```
#####
# Enable internal dns for hostname resolution, if it already not set
# manipulating configuration file via shell, alternate is to use openstack-config (TODO)
####
# setup dns domain first
if grep -q "^dns_domain.*openstacklocal$" /etc/neutron/neutron.conf; then
   sed -in-place "s|^dns_domain.*|dns_domain = oslocal|" /etc/neutron/neutron.conf
else
   if ! grep -q "^dns_domain" neutron.conf; then
       sed -in-place "s|^#dns_domain = openstacklocal$|dns_domain = oslocal|" /etc/neutron/neutron.conf
   fi
fi
# configure ml2 dns driver for neutron
ml2file=/etc/neutron/plugins/ml2/ml2_conf.ini
if ! grep -q "^extension_drivers" $ml2file; then
    # Assuming there is a place holder in comments, replace that string
    sed -in-place "s|^#extension_drivers.*|extension_drivers = port_security,dns|" $ml2file
else
    # Entry is present, check if dns is already present, if not then enable
    if ! grep "^extension_drivers" $ml2file|grep -q dns; then
        current_dns=`grep "^extension_drivers" $ml2file`
        new_dns="$current_dns,dns"
        sed -in-place "s|^extension_drivers.*|$new_dns|" $ml2file
```

```
    fi
fi
```

We are pretty much done with initial configuration, so letâĂŹs restart all the services at controller node.

```
systemctl restart neutron-dhcp-agent
systemctl restart neutron-openvswitch-agent
systemctl restart neutron-metadata-agent
systemctl restart neutron-server
systemctl restart openstack-nova-scheduler
systemctl restart openstack-nova-compute
systemctl restart openstack-ironic-conductor
```

4.2 Instantiate bare metal nodes

4.2.1 Setup generic bare metal instance

This section configure open stack for bare metal instance according to HPC images and user inputs. Before starting this it is assumed that system administrator has installed openstack and its services and has done appropriate configuration for bare metal provisioning, which includes installing ironinc, keystone, nova, neutron, glance. It is assumed that keystone is configured with

Before instantiating bare metal nodes with HPC, we need to do little bit more configuration. Setup generic bare metal instance

This section configures the network for "HPC as a services", upload compute OS images to glance, create a flavor for bare metal and upload public keys for ssh session.

First create a generic network for "HPC as a service" with a name "sharednet1"

```
#Get the tenant ID for the services tenant
    SERVICES_TENANT_ID=‘keystone tenant-list | grep "|\s*services\s*|" | awk ’{print $2}’‘

    #Create the flat network on which you are going to launch instances
    neutron net-list | grep "|\s*sharednet1\s*|"
    net_exists=$?
    if [ "${net_exists}" -ne "0" ]; then
        neutron net-create --tenant-id ${SERVICES_TENANT_ID} sharednet1 --shared \
         --provider:network_type flat --provider:physical_network physnet1
    fi
    NEUTRON_NETWORK_UUID=‘neutron net-list | grep "|\s*sharednet1\s*|" | awk ’{print $2}’‘
```

Create a subnet for our cluster with user defined start and end IP addresses. Make the controller as a gateway for our instances.

```
#Create the subnet on the newly created network
    neutron subnet-list | grep "|\s*subnet01\s*|"
    subnet_exists=$?
    if [ "${subnet_exists}" -ne "0" ]; then
        neutron subnet-create sharednet1 --name subnet01 --ip-version=4 \
         --gateway=${controller_ip} --allocation-pool \
          start=${cc_subnet_dhcp_start},end=${cc_subnet_dhcp_end} --enable-dhcp \
           ${cc_subnet_cidr}
    fi
    NEUTRON_SUBNET_UUID=‘neutron subnet-list | grep "|\s*subnet01\s*|" | awk ’{print $2}’‘
```

Upload kernel and initrd images to glance service so that they are available to ironic while deploying node.

```
#Create the deploy-kernel and deploy-initrd images
    glance image-list | grep "|\s*deploy-vmlinuz\s*|"
    img_exists=$?
    if [ "${img_exists}" -ne "0" ]; then
        glance image-create --name deploy-vmlinuz --visibility public --disk-format \
        aki --container-format aki < ${chpc_image_deploy_kernel}
    fi
    DEPLOY_VMLINUZ_UUID=‘glance image-list | grep "|\s*deploy-vmlinuz\s*|" | awk ’{print $2}’‘

    glance image-list | grep "|\s*deploy-initrd\s*|"
    img_exists=$?
    if [ "${img_exists}" -ne "0" ]; then
        glance image-create --name deploy-initrd --visibility public --disk-format \
        ari --container-format ari < ${chpc_image_deploy_ramdisk}
    fi
    DEPLOY_INITRD_UUID=‘glance image-list | grep "|\s*deploy-initrd\s*|" | awk ’{print $2}
```

Create a bare metal flavor with nova.

```
#Create the baremetal flavor and set the architecture to x86_64
    # This will create common baremetal flavor, if SMS node & compute has different
    # characteristic than user shall create multiple flavor one each characterisitc
    nova flavor-list | grep "|\s*baremetal-flavor\s*|"
    flavor_exists=$?
    if [ "$flavor_exists" -ne "0" ]; then
        nova flavor-create baremetal-flavor baremetal-flavor ${RAM_MB} ${DISK_GB} ${CPU}
        nova flavor-key baremetal-flavor set cpu_arch=$ARCH
    fi
    FLAVOR_UUID=‘nova flavor-list | grep "|\s*baremetal-flavor\s*|" | awk ’{print $2}’‘
#Increase the Quota limit for admin to allow nova boot
    openstack quota set --ram 512000 --cores 1000 --instances 100 admin
```

Finally register public ssh keys with nova, so that admin can ssh to the node.

```
#Register SSH keys with Nova
 nova keypair-list | grep "|\s*ostack_key\s*|"
 keypair_exists=$?
 if [ "${keypair_exists}" -ne "0" ]; then
    nova keypair-add --pub-key ${HOME}/.ssh/id_rsa.pub ostack_key
 fi
```

Export keypay name for use it later in other sections

```
    KEYPAIR_NAME=ostack_key
```

4.2.2 Setup HPC head node

Previous section we created generic bare metal setup. In this section we will create configuration for HPC head node in an OpenStack cloud. We created HPC head node OS images in previous sections, let’s upload this image to glance, and store IMAGE id in environment variable SMS_DISK_IMAGE_UUID, to be used during boot.

```
# Create sms node image
   glance image-list | grep "|\s*sms-image\s*|"
   img_exists=$?
   if [ "${img_exists}" -ne "0" ]; then
       glance image-create --name sms-image --visibility public --disk-format \
```

```
    qcow2 --container-format bare < ${chpc_image_sms}
fi
SMS_DISK_IMAGE_UUID=‘glance image-list | grep "|\s*sms-image\s*|" | awk ’{print $2}’‘
```

For provisioning sms node with ironic, we need to register node with ironic. This is done by registering node's BMC, node characteristic (aka flavor) like memory, cpu, disk space and node architecture. And registering kernel boot images. We will use pxe_ipmitool as a provisioning driver in ironic with a boot mode as bios.

```
#Create a sms node in the bare metal service ironic.
    ironic node-list | grep "|\s*${sms_name}$\s*|"
    node_exists=$?
    if [ "${node_exists}" -ne "0" ]; then
        ironic node-create -d pxe_ipmitool -i deploy_kernel=${DEPLOY_VMLINUZ_UUID} -i \
         deploy_ramdisk=${DEPLOY_INITRD_UUID} -i ipmi_terminal_port=8023 -i \
          ipmi_address=${sms_bmc} -i ipmi_username=${sms_bmc_username} -i \
           ipmi_password=${sms_bmc_password} -p cpus=${CPU} -p memory_mb=${RAM_MB} -p \
         local_gb=${DISK_GB} -p cpu_arch=${ARCH} -p capabilities="boot_mode:bios" \
         -n ${sms_name}
    fi
    SMS_UUID=‘ironic node-list | grep "|\s*${sms_name}\s*|" | awk ’{print $2}’‘
```

Now we need tell ironic about the network port on which node will perform pxe boot by configuring MAC Address.

```
    #Add the associated port(s) MAC address to the created node(s)
    ironic port-create -n ${SMS_UUID} -a ${sms_mac}
```

Add the instance info and disk space for root Add the instance_info/image_source and instance_info/root_gb

```
    ironic node-update $SMS_UUID add instance_info/image_source=${SMS_DISK_IMAGE_UUID} \
     instance_info/root_gb=50
```

We will assign a fixed IP address to sms node. This is done by associating sms nodeâĂŹs MAC address with neutron port. We will store this information in the neutron with sms_name. we will also set environment SMS_PORT_ID variable with this port id, to be used during boot.

```
    #Setup neutron port for static IP addressing of sms node, this is an optional part
    neutron port-create sharednet1 --dns_name $sms_name --fixed-ip ip_address=$sms_ip \
     --name $sms_name --mac-address $sms_mac
    SMS_PORT_ID=‘neutron port-list | grep "|\s*$sms_name\s*|" | awk ’{print $2}’‘
```

4.2.3 Setup HPC compute nodes

In previous section we configured Openstack to instantiate sms node. In this section we will be configuring openstack to instantiate HPC compute nodes.

For HPC compute nodes, we created compute node images, upload hpc compute node image to glance as a user image, and store IMAGE id in environment variable USER_DISK_IMAGE_UUID, to be used during boot.

```
#Create the whole-disk-image from the user’s qcow2 file
    glance image-list | grep "|\s*user-image\s*|"
    img_exists=$?
    if [ "${img_exists}" -ne "0" ]; then
        glance image-create --name user-image --visibility public --disk-format qcow2 \
```

```
        --container-format bare < ${chpc_image_user}
    fi
    USER_DISK_IMAGE_UUID=‘glance image-list | grep "|\s*user-image\s*|" | awk ’{print $2}’‘
```

Similar to sms node, create setup for all compute nodes including creating ironic node, associating node MAC address, adding instance information and assigning fix IP address. In our example we used 4 hpc compute nodes. to store the information in each OpenStack component we will assign compute node host name as a name, which is host name prefix (as chosen by user in inputs), followed by a node counter.

```
# Setup Compute nodes
    for ((i=0; i < ${num_ccomputes}; i++)); do
        ##Create compute nodes in the bare metal service
        ironic node-list | grep "|\s*${cnodename_prefix}$((i+1))\s*|"
        node_exists=$?
        if [ "${node_exists}" -ne "0" ]; then
            ironic node-create -d pxe_ipmitool -i deploy_kernel=${DEPLOY_VMLINUZ_UUID} -i \
             deploy_ramdisk=${DEPLOY_INITRD_UUID} -i ipmi_terminal_port=8023 -i \
              ipmi_address=${cc_bmc[$i]} -i ipmi_username=${cc_bmc_username} -i \
               ipmi_password=${cc_bmc_password} -p cpus=${CPU} -p memory_mb=${RAM_MB} -p \
                local_gb=${DISK_GB} -p cpu_arch=${ARCH} -p capabilities="boot_mode:bios" -n \
                 ${cnodename_prefix}$((i+1))
        fi
        NODE_UUID_CC[$i]=‘ironic node-list | grep "|\s*${cnodename_prefix}$((i+1))\s*|" | \
        awk ’{print $2}’‘

        # update for compute nodes node MAC
        ironic port-create -n ${NODE_UUID_CC[$i]} -a ${cc_mac[$i]}

        #Add the instance_info/image_source and instance_info/root_gb
        ironic node-update ${NODE_UUID_CC[$i]} add \
         instance_info/image_source=${USER_DISK_IMAGE_UUID} instance_info/root_gb=50

        #Setup neutron port for static IP addressing of compute nodes
        cn_name=${cnodename_prefix}$((i+1))
        neutron port-create sharednet1 --dns_name $cn_name --fixed-ip ip_address=${cc_ip[$i]} \
         --name $cn_name --mac-address ${cc_mac[$i]}
        NEUTRON_PORT_ID_CC[$i]=‘neutron port-list | grep "|\s*${cnodename_prefix}$((i+1))\s*|" \
         | awk ’{print $2}’‘
    Done
```

Ironic periodically sync with Nova with available nodes. Nova then updates its record for all available hosts. So before booting the node with Nova allow some time to sync ironic with it.

```
# Wait for the Nova hypervisor-stats to sync with available Ironic resources
sleep 121
```

4.2.4 Boot SMS node

In previous section we completed the bare metal configuration. User can request any available baremetal nodes by specifying the flavor they want and image they want to boot node with. For bare metal we created a flavor with name baremetal-flavor, we will provide this to nova with a CLI option –flavor. In our situation we will request 1 bare metal node with a baremetal flavor (–flavor) and SMS node image to boot (–image). We also would like to reserve the IP address of this node.

In previous section (setup sms) we associated one of the nodes MAC address with IP address, we will request this from nova by indicating port-id we created earlier (port-id=$SMS_PORT_ID).

In a previous section we also created cloud-init script for sms nodes. We will provide cloud-init script

(chpcSMSInit) to nova CLI option –user-data. For cloud init we will use metadata server, which will be provided by "–meta role= option". We will provide sms public key with "–key-name" option. At the end we will give our node a name. This name will be a host name of booted bare metal node.

Before booting, save boot command to a script, which will useful later on if user wants to re-instantiate same node.

```
#Boot the sms node with nova. chpcInit is set from prepare_cloudInit
echo "nova boot --config-drive true --flavor ${FLAVOR_UUID} --image ${SMS_DISK_IMAGE_UUID} \
 --key-name ${KEYPAIR_NAME} --meta role=webservers --user-data=$chpcSMSInit --nic \
 port-id=${SMS_PORT_ID} ${sms_name}" > boot_sms
```

Issue a boot command to nova to boot a SMS node:

```
nova boot --config-drive true --flavor ${FLAVOR_UUID} --image ${SMS_DISK_IMAGE_UUID} --key-name
 ${KEYPAIR_NAME} --meta role=webservers --user-data=$chpcSMSInit --nic port-id=${SMS_PORT_ID} \
 ${sms_name}
```

Wait around 15 seconds before we boot compute nodes. This will allow enough time to boot SMS node before compute nodes starts.

```
sleep 15
```

4.2.5 Boot compute nodes

Booting compute nodes are very similar to SMS nodes. In our case we will boot 4 compute nodes (as specified in user inputs. Host name of compute node will use prefix defined by cnodename_prefix variable, followed by node counter. For compute node we will use compute node image (USER_DISK_IMAGE_UUID) and compute node cloud-init script (chpcInit).

```
for ((i=0; i < ${num_ccomputes}; i++)); do
filename="cn$((i+1))"
echo "nova boot --config-drive true --flavor ${FLAVOR_UUID} --image ${USER_DISK_IMAGE_UUID} \
 --key-name ${KEYPAIR_NAME} --meta role=webservers --user-data=$chpcInit --nic \
  port-id=${NEUTRON_PORT_ID_CC[$i]} ${cnodename_prefix}$((i+1))" > boot_$filename
nova boot --config-drive true --flavor ${FLAVOR_UUID} --image ${USER_DISK_IMAGE_UUID} \
 --key-name ${KEYPAIR_NAME} --meta role=webservers --user-data=$chpcInit --nic \
  port-id=${NEUTRON_PORT_ID_CC[$i]} ${cnodename_prefix}$((i+1))
#wait for 5 sec before booting other compute node
sleep 5
done
```