**OpenHPC**

**OpenHPC as a Service Recipes**

**CentOS 7.2 Base OS**

**OpenStack/SLURM for Linux\* (x86\_64)**

Legal Notice

---

# 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.

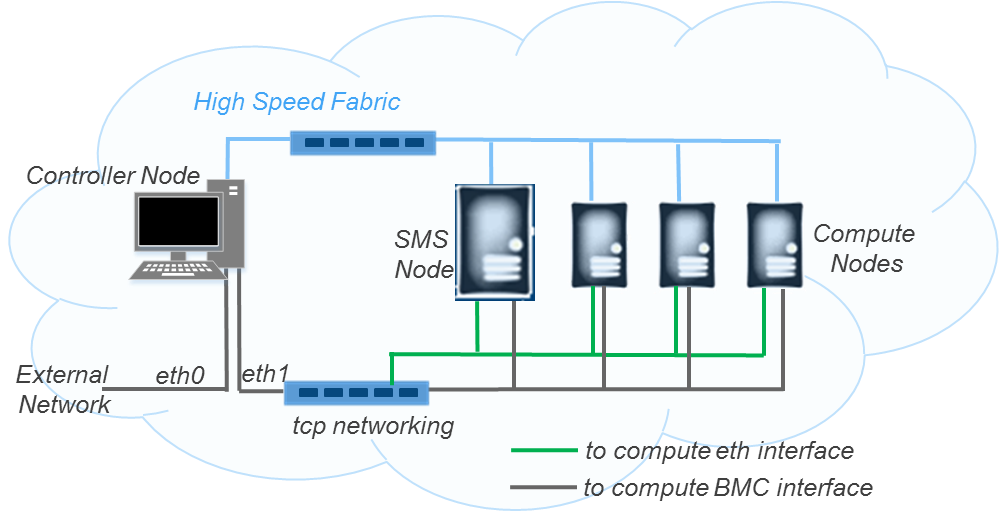
## Target Audience

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  
Tip: The solution is to increase the size of the manuals. {Mark V. Shaney

## Requirement/Assumption

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 RedHat®. 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 “HPC as a service”) will host an NFS file system that is made available to the HPC compute nodes.  


An outline of the physical architecture discussed is shown in Figure 1 and highlights the high-level  
networking configuration. The *Controller node* requires at least two Ethernet interfaces with *eth0* connected to  
the external network (public or 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.

This recipe sets various environment variables in one section and use them in other section. So Users is 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.

## Input

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   
($*f*variable*g*) 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.  
6 Rev: ac5491c  
Install Guide (v1.2.1): CentOS7.2/x86 64 + OpenStack + SLURM

* Same as input.tex plus
* ohpc\_pkg="${ohpc\_pkg:-https://github.com/openhpc/ohpc/releases/download/v1.1.GA/ohpc-release-centos7.2-1.1-1.x86\_64.rpm}"
* nodename\_prefix="${nodename\_prefix:-c}"
* chpc\_base=”ohpc”
* hpc\_slurm\_partition=normal
* # Prefix for compute node hostnames
* cnodename\_prefix= "${cnodename\_prefix:-cc}"
* # Local (internal) hostname on SMS
* controller\_name="${controller\_name=sun-hn3}"
* # Local (internal) IP address on SMS
* controller\_ip="${controller\_ip=192.168.46.13}"
* # cloud subnet CIDR
* cc\_subnet\_cidr="${cc\_subnet\_cidr=192.168.46.0/24}"
* # cloud subnet DHCP range
* cc\_subnet\_dhcp\_start="${cc\_subnet\_dhcp\_start=192.168.46.130}"
* cc\_subnet\_dhcp\_end="${cc\_subnet\_dhcp\_end=192.168.46.135}"
* sms\_mac=a4:bf:01:0c:e1:42
* sms\_bmc=192.168.46.58
* # BMC user credentials for use by IPMI
* sms\_bmc\_username="${sms\_bmc\_username:-root}"
* sms\_bmc\_password="${sms\_bmc\_password:-root}"
* cloud compute node characterstics (aka nova flavor)
* SMS\_RAM\_MB=24456
* SMS\_CPU=16
* SMS\_DISK\_GB=500
* SMS\_ARCH=x86\_64
* # cloud node BMC username and password
* cc\_bmc\_username="${cc\_bmc\_username=root}"
* cc\_bmc\_password="${cc\_bmc\_password=root}"
* #cloud compute node characterstics (aka nova flavor)
* RAM\_MB=24456
* CPU=16
* DISK\_GB=500
* ARCH=x86\_64
* SOCKETS=2
* CORES\_PER\_SOCKET=8
* THREADS\_PER\_CORE=2
* # Cloud Compute Node Images, optional part.
* chpc\_image\_deploy\_kernel="${chpc\_image\_deply=/opt/ohpc/admin/images/cloud/ironic-deploy.kernel}"
* chpc\_image\_deploy\_ramdisk="${chpc\_image\_ramdisk=/opt/ohpc/admin/images/cloud/ironic-deploy.initramfs}"
* chpc\_image\_user="${chpc\_image\_user=/opt/ohpc/admin/images/cloud/icloud-c7-cn.qcow2}"
* #chpc\_image\_sms="${chpc\_image\_sms=/opt/ohpc/admin/images/cloud/icloud-hpc-cent7-sms.qcow2}"
* chpc\_image\_sms="${chpc\_image\_sms=/opt/ohpc/admin/images/cloud/icloud-sms1.qcow2}"
* chpc\_create\_new\_image="${chpc\_create\_new\_image=0}"

# Preparing bare metal Node OS Images

This guide uses diskimage-builder utility to build and configure OS images for sms node as well compute node. Preparing images is an optional part of overall recipe. If user has predefined images then environment varioable “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 build on controller node (“[ctrlr]#”). Once images are build, they are stored at standard openHPC Path.

[ctrlr]# CHPC\_CLOUD\_IMAGE\_PATH=/opt/ohpc/admin/images/cloud/

## Install & Setup disimage-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>

## 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"

## 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)

## 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.

## 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 “compute”.

[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)

## 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)

## 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.

### 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 & compiler via environment variable, $DIB\_HPC\_COMPILER, DIB\_HPC\_MPI

#### Performance Tools lmod-default with their 3rd party libraries.

#### Hpc-env-base:

#### Hpc-mrsh

#### Hpc-slurm

### Editing HPC Elements

# 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.

## 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.

## Preparing template for sms node cloud-init (chpc\_sms\_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*

## 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*

# 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 & HPC compute node using nova.

It is assumed that system admin has installed OpenStack controller services & OpenStack network services (i.e. keystone, nova, ironic, glance, neutron, mongodb, rabbitmq server, heat etc). Controller node is configured with Openvswith Bridge on internal network port. Two tenant name admin & 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=<>*

## Prepare OpenStack for bare metal provisioning with ironinc

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 let’s 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*

## Instantiate bare metal nodes

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 beremtal 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 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*

### 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 ironicn 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 node’s 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}'`*

### 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*

### Boot SMS nodes

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 previous section we 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.

Bofore 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*

### 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*

# Customizing bare metal images

# Sample installation of OpenStack Services using packstack

# HPC Cloud bursting

This section should be integrated with existing openhpc recipe.