OpenHPC (v1.3)
OHPC on Openstack

CentOS7.3 Base OS

Openstack/SLURM Edition for Linux* (x86_64)

Document Last Update: 12MAY2017
Document Revision: 1.1

Legal Notice

Contents

1 Introduction

The term "HPC as a Service" refers to an on demand instantiation of HPC Service in a Cloud environment. This guide presents a simple "HPC cluster" instantiation procedure on an existing OpenStack (Mitaka) system. "HPC as a service" relies on two main principals to instantiate HPC service :

1. Providing pre-built OS images for compute nodes with HPC optimized software and
2. Use of cloud-init to configure and tune HPC services.

This document provides a simple guide to build HPC optimized OS images, prepare cloud-init recipes and finally instantiate a fully functional HPC System using those images and cloud-init.

Recipes will instantiate and HPC master node (a.k.a. sms node) and HPC compute nodes using pre-configured OpenStack images. The terms "master" and "sms" are used interchangeably in this guide.

OS Images are built using components from the 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 state-full 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.

1.1 Target Audience

This guide is targeted at experienced Linux system administrators, familiar with 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 (with neutron) node and four bare metal nodes. The Controller node serves as the central controller for OpenStack services and should have 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 CentOS7.3 in a state-full configuration.

This recipe is tested with OpenStack Mitaka release with CentOS 7.2. This document provides some examples for installing and configuring OpenStack using the Mitaka release of Packstack from RDO. 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.

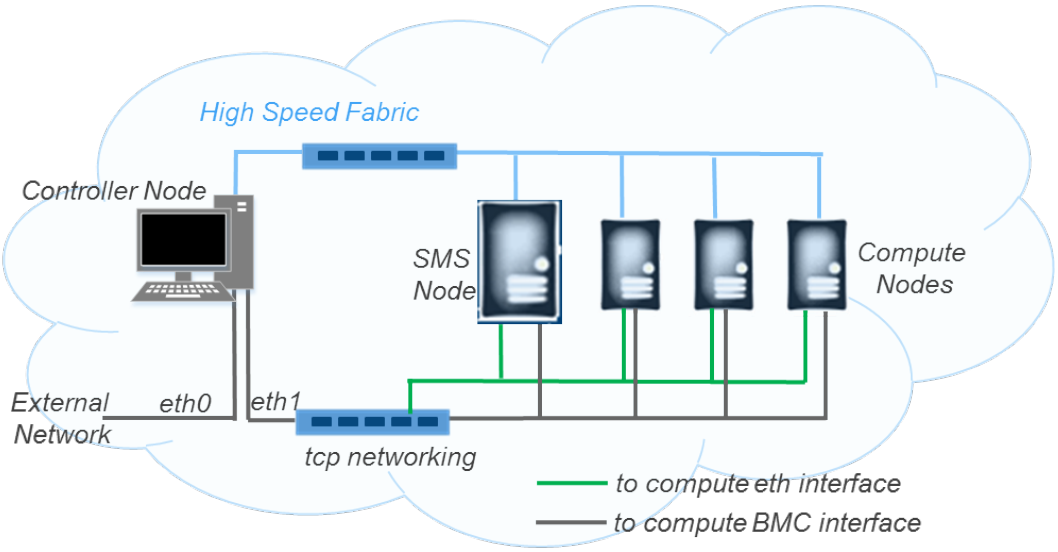

Figure 1: Overview of physical cluster architecture.

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.

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. In this recipe we will create utility functions for each functionality and then integrate them to make a complete receipe to create a compute node OS image. First setup a path for images to be stored.

```
[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. We will create common function to install diskimage-builder and its dependencies from base OS distro.

Create a function to install diskimage-builder

```
[ctrlr]# function setup_dib() {
```

install diskimage-builder and its dependencies

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

```
[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>
[ctrlr]# } # end of function
```

2.2 Setup common environment for diskimage-builder

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

we will create function setup_dib_hpc_base.

```
[ctrlr]# function setup_dib_hpc_base() {
```

Install dib if it is not already installed by calling setup_dib function, which we created in previous section

```
[ctrlr]#    setup_dib
```

```
[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"
[ctrlr]# } # end of the function
```

2.3 Preparing ironic deploy images

Ironic uses deploy images (aka kernel) to bootstrap the provisioning of user images. This section will create a function to build deploy images.

Start by creating a function prepare_deploy_image

```
[ctrlr]# function prepare_deploy_image() {
```

First check if user has requested to create images by verifying environment variables chpc_create_new_image, chpc_image_deploy_kernel and chpc_image_deploy_ramdisk. If user already supplied images then we will copy images to our common image location and setup environment variable for later use

```
[ctrlr]#   if [[ ${chpc_create_new_image} -ne 1 ]] && [[ -s $chpc_image_deploy_kernel ]] && [[ -s $chpc_image_deploy_ramdisk ]]
[ctrlr]#      # need to create an image, image is provided by user
[ctrlr]#      echo "Skiping cloud deploy-image build, Image provided:"
[ctrlr]#      echo "Deploy ramdisk Image:$chpc_image_deploy_ramdisk"
[ctrlr]#      # Store Images file
[ctrlr]#      CHPC_IMAGE_DEST=$CHPC_CLOUD_IMAGE_PATH/$(basename $chpc_image_deploy_kernel)
[ctrlr]#         if [[ ! -e $CHPC_IMAGE_DEST ]]; then
```

```
[ctrlr#         sudo cp -f $chpc_image_deploy_kernel $CHPC_CLOUD_IMAGE_PATH/
[ctrlr#      fi
[ctrlr#      chpc_image_deploy_kernel=$CHPC_IMAGE_DEST
[ctrlr#      CHPC_IMAGE_DEST=$CHPC_CLOUD_IMAGE_PATH/$(basename $chpc_image_deploy_ramdisk)
[ctrlr#      if [[ ! -e $CHPC_IMAGE_DEST ]]; then
[ctrlr#         sudo cp -f $chpc_image_deploy_ramdisk $CHPC_CLOUD_IMAGE_PATH/
[ctrlr#      fi
[ctrlr#      chpc_image_deploy_ramdisk=$CHPC_IMAGE_DEST
[ctrlr#   else
[ctrlr#      echo "Building new Cloud Deploy Image"
[ctrlr#      echo "========================================================================="
[ctrlr#      echo "=== Preparing cloud-hpc deploy images for ironic====================="
[ctrlr#      echo "========================================================================="
[ctrlr#      # prepare deploy images
[ctrlr#      # Install dib if it is not already installed
[ctrlr#      setup_dib
[ctrlr#      # Unset any previos envirnment flag
[ctrlr#      unset DIB_YUM_REPO_CONF
[ctrlr#      #  Install git if it is not already installed
[ctrlr#      yum -y install git
[ctrlr#      disk-image-create ironic-agent centos7 -o icloud-hpc-deploy-c7
[ctrlr#      echo "========================================================================="
[ctrlr#      echo "=== cloud-hpc deploy images Complete ================================="
[ctrlr#      echo "========================================================================="
[ctrlr#      chpc_image_deploy_kernel="$( realpath icloud-hpc-deploy-c7.kernel)"
[ctrlr#      chpc_image_deploy_ramdisk="$( realpath icloud-hpc-deploy-c7.initramfs)"
[ctrlr#      # 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)
[ctrlr#    fi
[ctrlr# } # end of function
```

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. we will create a function function prepare_sms_image

```
[ctrlr]# function prepare_sms_image() {
```

First check if user has requested to create images by verifying environment variables chpc_create_new_image and chpc_image_sms. If user already supplied images then we will copy images to our common image location and setup environment variable for later use

```
[ctrlr#    if [[ ${chpc_create_new_image} -ne 1 ]] && [[ -s $chpc_image_sms ]]; then
[ctrlr#       # No need to create an image, image is provided by user
[ctrlr#       echo -n "Skiping cloud sms-image build, Image provided:"
[ctrlr#       echo "$chpc_image_sms"
[ctrlr#       CHPC_IMAGE_DEST=$CHPC_CLOUD_IMAGE_PATH/$(basename $chpc_image_sms)
[ctrlr#       if [[ ! -e $CHPC_IMAGE_DEST ]]; then
```

```
[ctrlr]#           sudo cp $chpc_image_sms $CHPC_CLOUD_IMAGE_PATH
[ctrlr]#       fi
[ctrlr]#       chpc_image_sms=$CHPC_IMAGE_DEST
[ctrlr]#    else
```

. If user has not supplied images then we will build sms image here. disk-image-builder will supports two type of HPC images, sms and compute.

```
[ctrlr]#       # setup environment varioable to indicate sms image type
[ctrlr]#       export DIB\_HPC\_IMAGE\_TYPE=sms
```

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)
[ctrlr]#    fi # end of else of or if
[ctrlr]# } # end of function
```

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". then we will instruct disk-image-builder to install client hpc packages one by one. We will do all these in a shall function prepare_user_images.

```
[ctrlr]# function prepare_user_image() {
```

Check if user has already supplied compute node images by checking environment variable chpc_create_new_image and chpc_image_user. If image is supplied then we will copy image to ohpc image location and setup our environemnt variable to point to image location

```
[ctrlr]#   if [[ ${chpc_create_new_image} -ne 1 ]] && [[ -s $chpc_image_user ]]; then
[ctrlr]#      # No need to create an image, image is provided by user
[ctrlr]#      echo -n "Skiping cloud user-image build, Image provided:"
[ctrlr]#      echo "$chpc_image_user"
[ctrlr]#      CHPC_IMAGE_DEST=$CHPC_CLOUD_IMAGE_PATH/$(basename $chpc_image_user)
[ctrlr]#      if [[ ! -e $CHPC_IMAGE_DEST ]]; then
[ctrlr]#         sudo cp $chpc_image_user $CHPC_CLOUD_IMAGE_PATH
[ctrlr]#      fi
[ctrlr]#      chpc_image_user=$CHPC_IMAGE_DEST
[ctrlr]#   else
```

Creat new image if one is not supplied by a user. first set image type to 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)
[ctrlr]#    fi # end of else of or if
[ctrlr]# } # end of function
```

2.4.3 Putting all together

We created helper function. Call these function in order to generate images for compute node.
print banner to start image building

```
[ctrlr]# echo "################################################################################"
[ctrlr]# echo "############################ Starting SMS Image ############################"
[ctrlr]# echo "################################################################################"
```

First prepare image for sms node by calling function prepare_image_sms [ctrlr](*#*) prepare_sms_image print banner to indicate end of image creation

```
[ctrlr]# echo $chpc_image_sms
[ctrlr]# echo "################################################################################"
[ctrlr]# echo "############################ sms image is done ############################"
[ctrlr]# echo "################################################################################"
```

prepare image for compute node by calling function prepare_user_image [ctrlr](*#*) prepare_user_image print banner to indicate end of user creation

```
[ctrlr]# echo $chpc_image_user
[ctrlr]# echo "################################################################################"
[ctrlr]# echo "############################ user image is done ############################"
[ctrlr]# echo "################################################################################"
```

prepare deploye images for ironic provisioning [ctrlr](*#*) prepare_deploy_image print banner to indicate end of user creation

```
[ctrlr]# echo $chpc_image_deploy_kernel
[ctrlr]# echo $chpc_image_deploy_ramdisk
[ctrlr]# echo "################################################################################"
[ctrlr]# echo "############################ deploy image is done ############################"
[ctrlr]# echo "################################################################################"
```

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

```
[ctrlr]#chpcInitPath=/opt/ohpc/admin/cloud_hpc_init
[ctrlr]#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.

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

```
[ctrlr]# # nfs mount directory from SMS head node to Compute Node
[ctrlr]# cat /etc/fstab | grep "<sms_ip>:/home"
[ctrlr]# home_exists=$?
[ctrlr]# if [ "${home_exists}" -ne "0" ]; then
[ctrlr]#    echo "<sms_ip>:/home /home nfs nfsvers=3,rsize=1024,wsize=1024,cto 0 [ctrlr]# 0" >> /etc/fstab
[ctrlr]# fi
[ctrlr]# cat /etc/fstab | grep "<sms_ip>:/opt/ohpc/pub"
[ctrlr]# ohpc_pub_exists=$?
[ctrlr]#
[ctrlr]# if [ "${ohpc_pub_exists}" -ne "0" ]; then
[ctrlr]#    echo "<sms_ip>:/opt/ohpc/pub /opt/ohpc/pub nfs nfsvers=3 0 0" >> /etc/fstab
[ctrlr]#    # Make sure we have directory to mount
[ctrlr]#    # Clean up if required
[ctrlr]#    if [ -e /opt/ohpc/pub ]; then
[ctrlr]#        echo "chpcInit: [WARNING] /opt/ohpc/pub already exists!!"
[ctrlr]#    fi
[ctrlr]# fi
[ctrlr]# mkdir -p /opt/ohpc/pub
[ctrlr]# mount /home
[ctrlr]# mount /opt/ohpc/pub
[ctrlr]#
[ctrlr]# # Mount cloud_hpc_init
[ctrlr]# cat /etc/fstab | grep "sms_ip:$chpcInitPath"
[ctrlr]# CloudHPCInit_exist=$?
[ctrlr]# if [ "${CloudHPCInit_exist}" -ne "0" ]; then
[ctrlr]#    echo "<sms_ip>:$chpcInitPath $chpcInitPath nfs nfsvers=3 0 0" >> /etc/fstab
[ctrlr]# fi
```

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

Have ntp sync with sms node.

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

Update the hostname as per sms node.

```
[ctrlr]# # Setup hostname as per the head node
[ctrlr]# #Find the hostname of this machine from the copied over /etc/hosts file
[ctrlr]# cc_ipaddrs=(‘hostname -I‘)
[ctrlr]# for cc_ipaddr in ${cc_ipaddrs[@]}; do
[ctrlr]#    cat /etc/hosts | grep ${cc_ipaddr} > /dev/null
[ctrlr]#    result=$?
[ctrlr]#    if [ "$result" -eq "0" ]; then
[ctrlr]#        cc_hostname=‘cat /etc/hosts | grep ${cc_ipaddr} | cut -d$’\t’ -f2‘
[ctrlr]#        break
[ctrlr]#    fi
[ctrlr]# done
[ctrlr]#
[ctrlr]# if [ -z "${cc_hostname}" ]; then
[ctrlr]#    logger "chpcInit:ERROR: No resolved hostname found for any IP address in /etc/hosts"
[ctrlr]#    exit 1
```

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

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

```
[ctrlr]# # Start slurm and munge
[ctrlr]# systemctl enable munge
[ctrlr]# systemctl restart munge
[ctrlr]# systemctl enable slurmd
[ctrlr]# systemctl restart slurmd
```

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

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

Save the file with name chpc_init, 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.

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

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

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

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

```
[ctrlr]# # export CloudInit Path to nfs share
[ctrlr]# cat /etc/exports | grep "$chpcInitPath"
```

```
[ctrlr]# chpcInitPath_exported=$?
[ctrlr]#
[ctrlr]# if [ "${chpcInitPath_exported}" -ne "0" ]; then
[ctrlr]#   echo "$chpcInitPath *(rw,no_subtree_check,no_root_squash)" >> /etc/exports
[ctrlr]# fi
[ctrlr]# # share /home from HN
[ctrlr]# if ! grep "^/home" /etc/exports; then
[ctrlr]#    echo "/home *(rw,no_subtree_check,fsid=10,no_root_squash)" >> /etc/exports
[ctrlr]# fi
[ctrlr]# # share /opt/ from HN
[ctrlr]# if ! grep "^/opt/ohpc/pub" /etc/exports; then
[ctrlr]#    echo "/opt/ohpc/pub *(ro,no_subtree_check,fsid=11)" >> /etc/exports
[ctrlr]# fi
[ctrlr]# exportfs -a
[ctrlr]# # Restart nfs
[ctrlr]# systemctl restart nfs
[ctrlr]# systemctl enable nfs-server
[ctrlr]# logger "chpcInit: nfs configuration complete, updating remaining HPC configuration"
```

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

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

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

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

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

```
[ctrlr]# #Change file permissions in /etc/ssh to fix ssh into compute node
[ctrlr]# 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.

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

```
[ctrlr]# 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.

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

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

3.3.3 Enable genders during cloud-init

Create a new file sms/enable_genders and add configuration to enable genders on compute nodes. And save it.

```
[ctrlr]# #Update the genders at sms with compute node information
[ctrlr]# # first check if genders is install TBD
[ctrlr]# for ((i=0; i<$num_ccomputes; i++)) ; do
[ctrlr]# cat /etc/genders | grep ${cc_name[$i]}
[ctrlr]# gender_exists=$?
[ctrlr]# if [ "${gender_exists}" -ne "0" ]; then
[ctrlr]# echo -e "${cc_name[$i]}\tcompute,bmc=${cc_bmc[$i+1]}"
[ctrlr]# fi
[ctrlr]# done >> /etc/gender
```

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

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

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

```
[ctrlr]# 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

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

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

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

```
[ctrlr]# openstack service list

#Expected Output#
+----------------------------------+-----------+--------------+---------------+
| 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     |
+----------------------------------+-----------+--------------+---------------+

[ctrlr]# openstack project list

#Expected Output#
+----------------------------------+----------+
| 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.

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

```
[ctrlr]# #Set SELinux to permissive
[ctrlr]# setenforce 0

[ctrlr]# #Source the keystonerc_admin file
[ctrlr]# source ${HOME}/keystonerc_admin

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

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

```
[ctrlr]# yum install -y tftp-server syslinux-tftpboot xinetd
```

Make the directory for tftp and give it the ironic owner

```
[ctrlr]# mkdir -p /tftpboot
[ctrlr]# chown -R ironic /tftpboot
```

Configure tfpt server

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

#Restart the xinetd service
[ctrlr]# systemctl restart xinetd

#Copy the PXE linux files to the tftpboot directory we created
[ctrlr]# cp /var/lib/tftpboot/pxelinux.0 /tftpboot
```

```
[ctrlr]# cp /var/lib/tftpboot/chain.c32 /tftpboot

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

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

```
[ctrlr]# 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:

```
[ctrlr]# sed --in-place "s|#tftp_root=/tftpboot|tftp_root=/tftpboot|" /etc/ironic/ironic.conf
[ctrlr]# sed --in-place "s|#ip_version=4|ip_version=4|" /etc/ironic/ironic.conf
[ctrlr]# 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

```
[ctrlr]# 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.

```
[ctrlr]# sed --in-place "s|reserved_host_memory_mb=512|reserved_host_memory_mb=0|" /etc/nova/nova.conf
[ctrlr]# 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.

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

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

```
[ctrlr]#    # Entry is present, check if dns is already present, if not then enable
[ctrlr]#    if ! grep "^extension_drivers" $ml2file|grep -q dns; then
[ctrlr]#        current_dns=‘grep "^extension_drivers" $ml2file‘
[ctrlr]#        new_dns="$current_dns,dns"
[ctrlr]#        sed -in-place "s|^extension_drivers.*|$new_dns|" $ml2file
[ctrlr]#    fi
[ctrlr]# fi
```

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

```
[ctrlr]# systemctl restart neutron-dhcp-agent
[ctrlr]# systemctl restart neutron-openvswitch-agent
[ctrlr]# systemctl restart neutron-metadata-agent
[ctrlr]# systemctl restart neutron-server
[ctrlr]# systemctl restart openstack-nova-scheduler
[ctrlr]# systemctl restart openstack-nova-compute
[ctrlr]# 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"

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

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

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

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

Create a bare metal flavor with nova.

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

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

Export keypair name for use it later in other sections

```
[ctrlr]#    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.

```
[ctrlr]# # Create sms node image
[ctrlr]#   glance image-list | grep "|\s*sms-image\s*|"
[ctrlr]#   img_exists=$?
[ctrlr]#   if [ "${img_exists}" -ne "0" ]; then
[ctrlr]#       glance image-create --name sms-image --visibility public --disk-format qcow2 --container-format bare < ${chpc_im
[ctrlr]#   fi
[ctrlr]#   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.

```
[ctrlr]# #Create a sms node in the bare metal service ironic.
[ctrlr]#    ironic node-list | grep "|\s*${sms_name}$\s*|"
[ctrlr]#    node_exists=$?
[ctrlr]#    if [ "${node_exists}" -ne "0" ]; then
[ctrlr]#        ironic node-create -d pxe_ipmitool -i deploy_kernel=${DEPLOY_VMLINUZ_UUID} -i deploy_ramdisk=${DEPLOY_INITRD_UU
[ctrlr]#    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.

```
[ctrlr]# #Add the associated port(s) MAC address to the created node(s)
[ctrlr]#    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

```
[ctrlr]#    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.

```
    [ctrlr]# #Setup neutron port for static IP addressing of sms node, this is an optional part
    [ctrlr]# neutron port-create sharednet1 --dns_name $sms_name --fixed-ip ip_address=$sms_ip --name $sms_name --mac-address $
[ctrlr]#    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.

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

```
[ctrlr]# # Setup Compute nodes
[ctrlr]# # Note: if installed from the rpm, the following script is installed as setup_compute_nodes.sh
[ctrlr]#
[ctrlr]#    for ((i=0; i < ${num_ccomputes}; i++)); do
[ctrlr]#        ##Create compute nodes in the bare metal service
[ctrlr]#        ironic node-list | grep "|\s*${cnodename_prefix}$((i+1))\s*|"
[ctrlr]#        node_exists=$?
[ctrlr]#        if [ "${node_exists}" -ne "0" ]; then
[ctrlr]#            ironic node-create -d pxe_ipmitool -i deploy_kernel=${DEPLOY_VMLINUZ_UUID} -i deploy_ramdisk=${DEPLOY_INITRD
[ctrlr]#       fi
[ctrlr]#       NODE_UUID_CC[$i]=`ironic node-list | grep "|\s*${cnodename_prefix}$((i+1))\s*|" | awk '{print $2}'`

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

[ctrlr]#        #Add the instance_info/image_source and instance_info/root_gb
[ctrlr]#        ironic node-update ${NODE_UUID_CC[$i]} add instance_info/image_source=${USER_DISK_IMAGE_UUID} instance_info/roo
[ctrlr]#
[ctrlr]#        #Setup neutron port for static IP addressing of compute nodes
[ctrlr]#        cn_name=${cnodename_prefix}$((i+1))
[ctrlr]#        neutron port-create sharednet1 --dns_name $cn_name --fixed-ip ip_address=${cc_ip[$i]} --name $cn_name --mac-addr
[ctrlr]#        NEUTRON_PORT_ID_CC[$i]=`neutron port-list | grep "|\s*${cnodename_prefix}$((i+1))\s*|" | awk '{print $2}'`
[ctrlr]#    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.

```
[ctrlr]# # Wait for the Nova hypervisor-stats to sync with available Ironic resources
[ctrlr]# 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.

```
[ctrlr]# #Boot the sms node with nova. chpcInit is set from prepare_cloudInit
[ctrlr]# echo "nova boot --config-drive true --flavor ${FLAVOR_UUID} --image ${SMS_DISK_IMAGE_UUID} --key-name ${KEYPAIR_NAME}
```

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

```
[ctrlr]# nova boot --config-drive true --flavor ${FLAVOR_UUID} --image ${SMS_DISK_IMAGE_UUID} --key-name ${KEYPAIR_NAME} --meta
```

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

```
[ctrlr]# 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).

```
[ctrlr]# for ((i=0; i < ${num_ccomputes}; i++)); do
[ctrlr]# filename="cn$((i+1))"
[ctrlr]# echo "nova boot --config-drive true --flavor ${FLAVOR_UUID} --image ${USER_DISK_IMAGE_UUID} --key-name ${KEYPAIR_NAME}
[ctrlr]# nova boot --config-drive true --flavor ${FLAVOR_UUID} --image ${USER_DISK_IMAGE_UUID} --key-name ${KEYPAIR_NAME} --met
[ctrlr]# #wait for 5 sec before booting other compute node
[ctrlr]# sleep 5
[ctrlr]# done
```

5 Appendix

This file generates the parent scripts form the parse_doc