OpenHPC (v2.3)

Recipes and Tutorials for Building Open-Source Mini-Clusters

CentOS8.3 Base OS

xCAT/SLURM Edition for Linux* (x86 64)

Isayah Reed

September 2021

Revision 1.0

Table of Contents

Version	Description
v1.0	Initial draft. XCAT setup, slurm setup/validation, nvidia driver+cuda
v1.1	Removed gpu driver, changed node names from s to c

Introduction

This document contains instructions to setup, validate, and manage an open-source HPC Linux mini-cluster. It is a variation of the Linux Foundation Collaborative OHPC Project xCAT installation recipe[1] for CentOS-8.3. The recipe is customized for a small, portable, and easily assembled cluster of Intel NUC machines. The initial setup calls for one master node – also referred to as the head or management node - with two compute nodes. The recipe includes setup of a scheduler and resource managemer, HPC development tools, and, optionally, additional HPC components such as high-speed interconnects and GPUs. This guide assumes a clean CentOS-8.3 minimal installation.

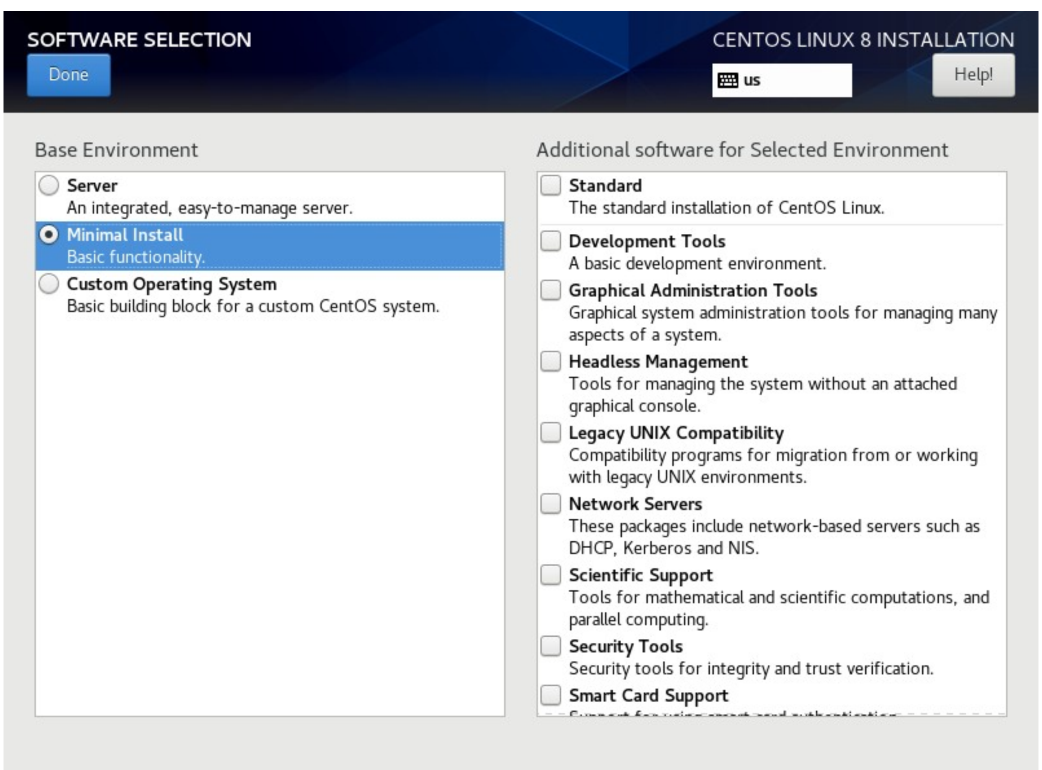
[1]https://github.com/openhpc/ohpc/releases/download/v2.3.GA/Install_guide-CentOS8-xCAT-Stateless-SLURM-2.3-x86_64.pdf

Hardware and BIOS Setup

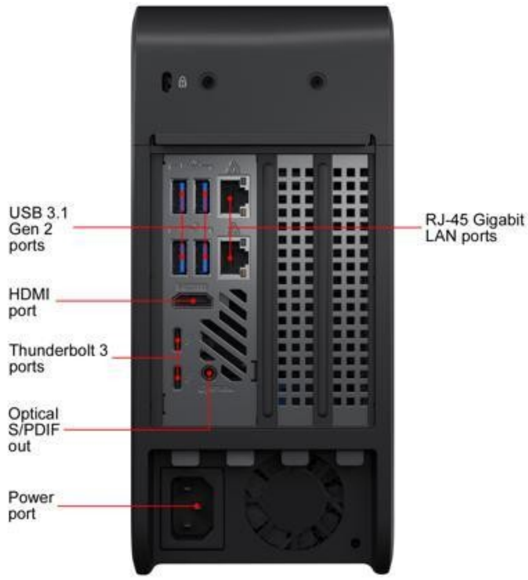
The equipment requirements for this documentation are:

- 3 Intel NUC machines. At least one NUC must have two ethernet ports
- 4 ethernet cords. One for each node, one for external internet
- 1 ethernet switch with at least 4 ports
- CentOS8.3 installation usb drive. Either ISO or netboot

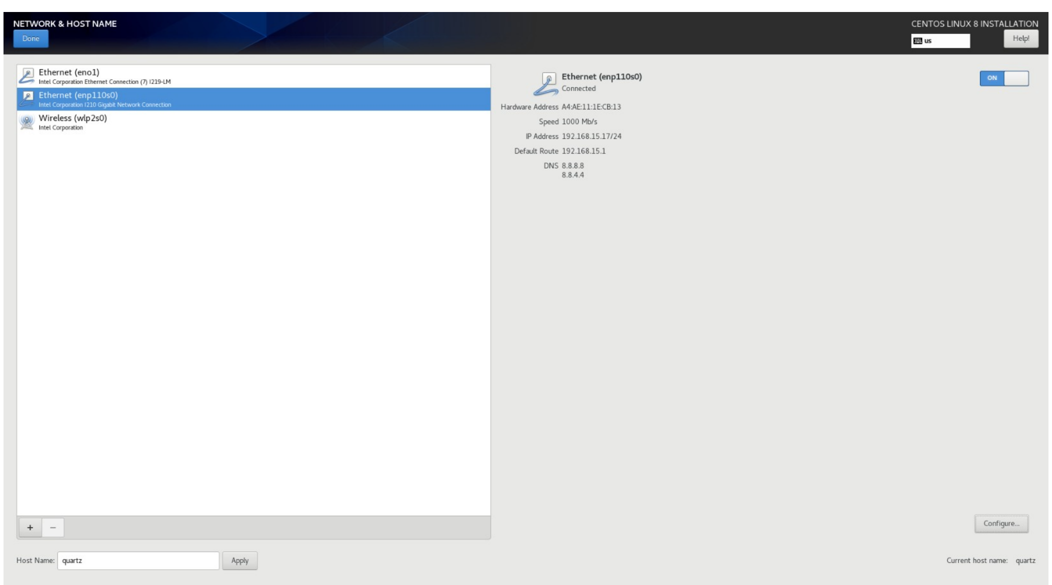
A barebones CentOS installation is used for this guide. This means a minimal install with no additional software selected.



This document is based on the 9th generation NUC - pictured below - validated with BIOS QXCFL579 on FW v12.0.40.1433. We use the lower powered I219-LM Ethernet port for the cluster environment. For the NUC9 pictured, this is port eno1 located on top. The other Ethernet port – I210, near the HDMI port – will serve as the connection to the external internet. External internet is only needed for the management node.



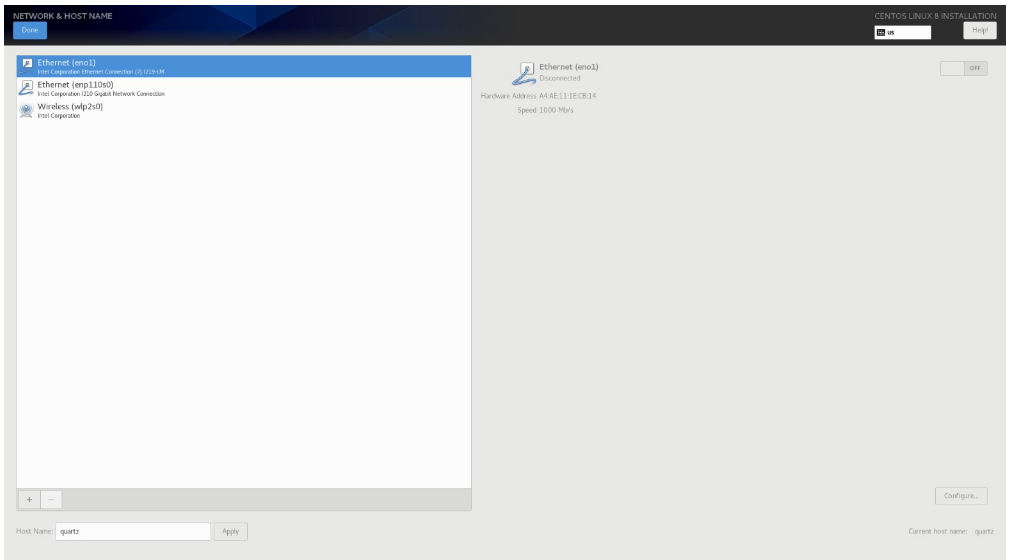
For simplicity, we will use DHCP to set the public IP address. A static IP address is recommended for login and administrating the cluster remotely.



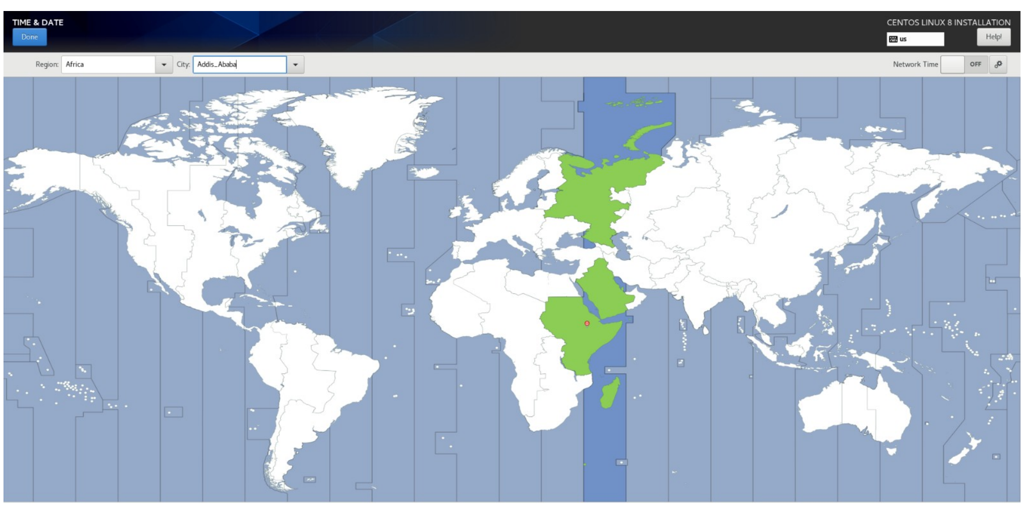
Only one Ethernet connection is needed for each compute node on the internal cluster network. Configure this port – eno1 – with the following settings for the management node:

- IP address: 10.10.1.10
- Subnet Mask: 8
- Default Gateway: 10.10.1.10
- DNS: 10.10.1.10

This connection is not needed at the moment, and should be deactivated after configuration. The hostname for the management node in the recipe will be called *quartz*. Wi-Fi must be turned off.



Finally, enable the root account, initialize a password, and set the OS system time.



Install xCAT OHPC

Setting up the openHPC environment requires the given linux commands to be executed with root privileges - either through the use of sudo or, preferably, directly as the root user.

From a clean CentOS minimal install, update the OS packages. The OS update may impact kernel files and drivers, so a reboot is required.

```
$> dnf -y update
$> reboot
```

Install the EPEL (Extra Packages for Enterprise Linux) repo and enable the OHPC repository through installation of the ohpc-release RPM. This repo contains the requisite provisioning tools and libraries, including the required xCAT packages we will use.

```
$> dnf -y --enablerepo=extras install epel-release
$> dnf -y install http://repos.openhpc.community/OpenHPC/2/CentOS_8/x86_64/ohpc-release-2-1.el8.x86_64.rpm
$> dnf -y install wget
$> wget -P /etc/yum.repos.d https://xcat.org/files/xcat/repos/yum/latest/xcat-core/xcat-core.repo
$> wget -P /etc/yum.repos.d http://xcat.org/files/xcat/repos/yum/devel/xcat-dep/rh8/x86_64/xcat-dep.repo
$> dnf -y install dnf-plugins-core
```

From the OHPC repo, install the base packages.

```
$> dnf config-manager --set-enabled powertools
$> dnf -y install ohpc-base xCAT
```

Load the xCAT environment variables. It is recommended to put this in the .bashrc file of the cluster administrator, to load by default during every login.

```
$> source /etc/profile.d/xcat.sh
```

Use the chrony NTP client to synchronize the system clock with an NTP server. To do this, enable the chrony service to start automatically during boot, set the service to synchronize with nearby NTP pools, and restart the service to use these new settings.

```
$> systemctl enable chronyd.service
$> echo "server 0.africa.pool.ntp.org" >> /etc/chrony.conf
$> echo "server 1.africa.pool.ntp.org" >> /etc/chrony.conf
$> echo "server 2.africa.pool.ntp.org" >> /etc/chrony.conf
$> echo "server 3.africa.pool.ntp.org" >> /etc/chrony.conf
$> echo "allow all" >> /etc/chrony.conf
$> systemctl restart chronyd
```

There is an additional setting in the chrony configurations process `$> echo "allow all" >> /etc/chrony.conf`

that allows the management node to also act as an NTP server, in addition to an NTP client. This is necessary to allow compute nodes to synchronize their local system clocks with the management node, as we will see later in this guide.

Next, install packages for the open-source SLURM scheduler.

```
$> dnf -y install ohpc-slurm-server
```

Activate the Ethernet connection eno1 that will be used for communication with compute nodes, and add an entry for this interface into the xCAT database

```
$> ip link set dev eno1 up
$> systemctl reload NetworkManager
$> chdef -t site dhcpinterfaces="xcatmn|eno1"
```

The next step is to create the base CentOS image. Download the full CentOS ISO, then place the linux image into /install with copycds:

```
$> wget http://centos.mirror.liquidtelecom.com/8.3.2011/isos/x86_64/CentOS-8.3.2011-x86_64-dvd1.iso
$> copycds ./CentOS-8.3.2011-x86_64-dvd1.iso
```

A list of available boot images can be displayed with lsdef. For a stateless cluster, netboot-compute will be used.

```
$> lsdef -t osimage
```

Set the location of image files. It is recommended to add this to the .bashrc of the system administrator.

```
$> export CHROOT=/install/netboot/centos8/x86_64/compute/rootimg/
```

Initialize the installation image to provide a minimal OS configuration.

```
$> genimage centos8-x86_64-netboot-compute
```

The newly creates minimal OS image needs to be customized for this cluster environment. This requires adding additional repositories, tools, and packages, as well as customizing OS system settings for inter-node synchronization and communication.

Add the required repositories to the compute node image. This will allow dnf package installations directly into the compute node image.

```
$> cp /etc/yum.repos.d/OpenHPC.repo $CHROOT/etc/yum.repos.d/
$> cp /etc/yum.repos.d/epel.repo $CHROOT/etc/yum.repos.d/
```

Install the OHPC client packages into the compute image.

```
$> dnf -y --installroot=$CHROOT install ohpc-base-compute
$> dnf -y --installroot=$CHROOT update
```

Compute nodes should only connect to the head node and other compute nodes - never the internet. Therefore, a firewall is not needed for compute nodes.

```
$> chroot $CHROOT systemctl disable firewalld
```

We want to use the same user names, passwords, and groups from the head node across any compute nodes. Therefore, we copy the information from the head node to the compute node image.

```
$> /bin/cp -f /etc/passwd /etc/group $CHROOT/etc
```

Install the SLURM packages for the compute nodes.

```
$> dnf -y --installroot=$CHROOT install ohpc-slurm-client
```

Note that some SLURM client packages are not the same as the packages for the head node. One example is the slurm background process used – the head node runs the slurmctld process, while compute nodes use the listening daemon slurmd. Compute nodes also do not need the configurations files in /etc/slurm.

```
$> echo SLURMD_OPTIONS="--conf-server 10.10.1.10" > $CHROOT/etc/sysconfig/slurmd
$> dnf -y --installroot=$CHROOT install chrony
$> echo "server 10.10.1.10" >> $CHROOT/etc/chrony.conf
```

Generate the base OS image for compute nodes. Before making the image, ensure the kernel version for all packages in the compute node image matches the kernel version of the head node.

```
$> dnf -y --installroot=$CHROOT install kernel-`uname -r`
$> genimage centos8-x86_64-netboot-compute -k `uname -r`
$> dnf config-manager --installroot=$CHROOT --enable baseos
$> dnf -y --installroot=$CHROOT install --enablerepo=powertools lmod-ohpc
```

There are some folders from the head node that we want to share with compute nodes. Examples include the user home directories, and a central location for shared applications (/opt/ohpc/pub).

Use NFS to automatically mount and share these folders across the internal network with the compute nodes.

```
$> echo "10.10.1.10:/home /home nfs nfsvers=3,nodev,nosuid 0 0" >> $CHROOT/etc/fstab
$> echo "10.10.1.10:/opt/ohpc/pub /opt/ohpc/pub nfs nfsvers=3,nodev 0 0" >> $CHROOT/etc/fstab
$> perl -pi -e "s|/tftpboot|#/tftpboot|" /etc/exports
$> perl -pi -e "s|/install|#/install|" /etc/exports
$> echo "/home *(rw,no_subtree_check,fsid=10,no_root_squash)" >> /etc/exports
$> echo "/opt/ohpc/pub *(ro,no_subtree_check,fsid=11)" >> /etc/exports
$> exportfs -a
$> systemctl restart nfs-server
$> systemctl enable nfs-server
```

Use the head node as a reference clock to synchronize the system clock of compute nodes. Recall that we previously set the head node to synchronize with external NTP servers over the public internet. Since we want to avoid exposure to the public internet by compute nodes, the head node will be used as a reference instead of an external NTP server. Additionally, edit the sshd file to limit compute nodes access by allowing ssh access only to users who have an active job on the node.

```
$> echo "server 10.10.1.10" >> $CHROOT/etc/chrony.conf
$> chroot $CHROOT systemctl enable chronyd
$> echo "server 10.10.1.10" >> $CHROOT/etc/chrony.conf
$> echo "account required pam_slurm.so" >> $CHROOT/etc/pam.d/sshd
```

Along with sharing files between head and compute nodes, we want the files to automatically update on the compute node whenever there is a change in the corresponding on the head node file. An example would be creating a new linux user – the compute node should automatically recognize the new credentials without needing a reboot to update the corresponding files.

```
$> mkdir -p /install/custom/netboot
$> chdef -t osimage -o centos8-x86_64-netboot-compute \
        synclists="/install/custom/netboot/compute.synclist"
$> echo "/etc/passwd -> /etc/passwd" > /install/custom/netboot/compute.synclist
$> echo "/etc/group -> /etc/group" >> /install/custom/netboot/compute.synclist
$> echo "/etc/shadow -> /etc/shadow" >> /install/custom/netboot/compute.synclist
$> echo "/etc/slurm/slurm.conf -> /etc/slurm/slurm.conf " >> \
        /install/custom/netboot/compute.synclist
$> echo "/etc/munge/munge.key -> /etc/munge/munge.key " >> \
        /install/custom/netboot/compute.synclist
$> echo "/etc/hosts -> /etc/hosts " >> /install/custom/netboot/compute.synclist
```

Here, we are synchronizing the files for usernames, user passwords, user groups, SLURM node configurations, munge authentication keys, and hostnames.

The final steps are adding information about the cluster environment to the xCAT database.

First, register the domain name for cluster node name resolution

```
$> chdef -t site domain=aau
$> chdef -t network net1 net=10.10.1.0 mask=255.0.0.0 gateway=10.10.1.1
$> echo 10.10.1.10 quartz quartz.aau >> /etc/hosts
```

Next, add relevant system information for the compute nodes to the xCAT database. For this cluster environment, the compute nodes will be named c1 and c2.

```
$> mkdef -t node c1 groups=nuc,all ip=10.10.1.72 mac=<MAC> netboot=xnba arch=x86_64
$> mkdef -t node c2 groups=nuc,all ip=10.10.1.71 mac=<MAC> netboot=xnba arch=x86_64
```

Enable the Ethernet used for compute nodes to automatically activate during boot.

```
$> perl -pi -e "s/ONBOOT=\S+/ONBOOT=yes/" /etc/sysconfig/network-scripts/ifcfg-eno1
```

Complete the configuration for registration of network information in the xCAT database.

```
$> makehosts
$> makenetworks
$> makedhcp -n
$> makedns -n
```

Rebuild compute image from chroot environment to include the newly added information about the compute environment, then set the compute nodes to use this re-built netboot compute image.

```
$> packimage centos8-x86_64-netboot-compute
$> nodeset all osimage=centos8-x86_64-netboot-compute
```

The compute nodes should be ready for use. Boot the compute nodes and ping to ensure they are online and able to communicate with the head node.

```
$> ping c1
```

Setup SLURM Scheduler

Now that we have compute nodes that can communicate with the management node in the cluster environment, set up the SLURM manager to enable job submission to the compute nodes. We will use the sample configuration file – slurm.conf.example - from the slurm directory as a template for making edits to customize for our environment.

```
$> cp /etc/slurm/slurm.conf.example /etc/slurm/slurm.conf
```

Edit the sample template to reflect the specifications of the cluster we are building. First, set the machine name to be the name of our cluster: quartz. We want any node that goes down to automatically become available as soon as a valid configuration is detected. This requires changing the ReturnToService parameter.

```
$> perl -pi -e "s/ControlMachine=\S+/ControlMachine=quartz/" /etc/slurm/slurm.conf
$> perl -pi -e "s|ReturnToService=0|#ReturnToService=2|" /etc/slurm/slurm.conf
```

Next, add information about the compute nodes. Create a partition called *normal* and add our compute nodes to this partition.

```
$> perl -pi -e "s|PartitionName|#PartitionName|" /etc/slurm/slurm.conf
$> echo PartitionName=normal Nodes=c1,c2 Default=YES MaxTime= 2:00:00 MaxNodes=2 State=UP >> \
    /etc/slurm/slurm.conf
$> perl -pi -e "s|NodeName|#NodeName|" /etc/slurm/slurm.conf
$> echo NodeName=c1 Sockets=1 CoresPerSocket=2 ThreadsPerCore=2 State=UNKNOWN >> \
    /etc/slurm/slurm.conf
$> echo NodeName=c2 Sockets=1 CoresPerSocket=2 ThreadsPerCore=2 State=UNKNOWN >> \
    /etc/slurm/slurm.conf
$> echo SlurmctldParameters=enable_configless >> /etc/slurm/slurm.conf
```

The scheduler must know basic information about the compute nodes in order to efficiently allocate resources for job submissions. For example, a job that requires 4 cores cannot be scheduled on a node with only 2 cores.

Restart the slurm controller and munge authenticator for the edits to take effect.

```
$> systemctl restart munge
$> systemctl restart slurmctld
```

Set the slurm daemon and munge authenticator to automatically start when the head and compute nodes boot. This allows us to execute jobs on the compute nodes immediately after boot instead of manually starting the required processes after each boot.

```
$> systemctl enable munge
$> systemctl enable slurmctld
$> chroot $CHROOT systemctl enable slurmd
$> chroot $CHROOT systemctl enable munge
```

The pdsh command is an executable included in the OHPC packages that allows distributed commands to each compute node. In this example, we send the systemctl command to restart the slurm daemon on nodes c1 and c2.

```
$> pdsh -w c[1-2] systemctl restart slurmd
```

Check the status of the compute nodes.

```
$> sinfo
```

If any node state is listed as 'down', send a command to resume the compute node(s).

```
$> scontrol update nodename=c1 state=resume
$> scontrol update nodename=c2 state=resume
```

The compute nodes are now ready for job submissions.

Submit Distributed Jobs to Compute Nodes

At this point, the cluster environment should be setup for distributed communication, with the ability to submit jobs through the slurm scheduler. To test this functionality, we create a sample MPI application and submit it to the compute nodes.

Install the packages we will need on both the management and compute node. This includes MPI tools and libraries. To install and use these packages on the compute nodes, the compute image will need to be re-built and compute nodes will require a reset.

```
$> dnf -y install openmpi-devel gcc-c++
$> dnf -y --installroot=$CHROOT install openmpi openmpi-devel gcc-c++
$> packimage centos8-x86_64-netboot-compute
$> pdsh -w c[1-2] reboot
```

At this point, it is recommended to continue as a regular user instead of root.

The sample MPI code below is a simple application that can pass messages between multiple nodes.

```
#include "mpi.h"
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(int argc, char *argv[])
{
  int num_procs, num_local;
  char mach_name[MPI_MAX_PROCESSOR_NAME];
  int mach_len;

  MPI_Init (&argc,&argv);
  MPI_Comm_size (MPI_COMM_WORLD, &num_procs);
  MPI_Comm_rank (MPI_COMM_WORLD, &num_local);
  MPI_Get_processor_name(mach_name,&mach_len);

  MPI_Barrier(MPI_COMM_WORLD);

  if(num_local == 0)
    printf("\n Hello, world (%i procs total)\n",num_procs);

  MPI_Barrier(MPI_COMM_WORLD);

  printf("    --> Process # %3i of %3i is alive. -> %s\n",
  num_local,num_procs,mach_name);

  MPI_Finalize();
  return 0;
}
```

Copy the sample code to a file named hello.c. Compile the sample code.

```
$> export PATH=/usr/lib64/openmpi/bin:$PATH
$> mpicc ./hello.c -o hello
```

Create a file called batch_script and add the sample code below.

```
#!/bin/bash -l
#SBATCH -N 2
#SBATCH -J sample_run
#SBATCH -o output.txt

/usr/lib64/openmpi/bin/mpirun -n 2 -N 1 ./hello
```

Execute the script by submitting to the slurm scheduler.

```
$> sbatch < ./batch_script
```

Check the status with squeue. If the output is empty then the job in complete and results are written to ./output.txt.

```
$> squeue
$> cat ./output.txt
```