NVIDIA DGX BasePOD

Deployment Guide

Featuring NVIDIA DGX A100 Systems

DG-11192-001 | April 2023

Document History

DG-11192-001

Version	Date	Authors	Description of Change
01	2022-11-12	Davinder Singh, Greg Zynda, Robert Sohigian, Robert Strober, Scott Ellis, and Yang Yang	Initial release
02	2022-11-17	Robert Sohigian and Yang Yang	Procedural updates
03	2023-03-15	Robert Sohigian and Yang Yang	Procedural and formatting updates
04	2023-04-07	Joe Handzik, Michael Levy, and Robert Sohigian	Further procedural and formatting updates

Abstract

Artificial intelligence (AI) infrastructure requires significant compute resources to train the latest state-of-the-art models efficiently, often requiring multiple nodes running in a distributed cluster.

While cloud computing provides an easy on-ramp to train AI models, many enterprises require an on-premises data center for a variety of technical or business reasons.

Building AI infrastructure on-premises can be a complex and confusing process. Careful planning and coordination will make the cluster deployment easier, and the job of the cluster administrators tasked with the day-to-day operation easier.

NVIDIA DGX BasePOD™ provides the underlying infrastructure and software to accelerate deployment and execution of these new AI workloads. By building upon the success of NVIDIA DGX™ systems, DGX BasePOD is a prescriptive AI infrastructure for enterprises, eliminating the design challenges, lengthy deployment cycle, and management complexity traditionally associated with scaling AI infrastructure.

The DGX BasePOD is built upon the NVIDIA DGX A100 system, which offers unprecedented compute performance with eight NVIDIA A100 Tensor Core GPUs connected with NVIDIA NVLink® and NVIDIA NVSwitch™ technologies for fast inter-GPU communication.

Powered by NVIDIA Base Command™, DGX BasePOD provides the essential foundation for AI development optimized for the enterprise.

Contents

Chapter 1. Architecture

1.1 Hardware Overview

The DGX BasePOD consists of DGX A100 compute nodes, five control plane servers (two for cluster management and three Kubernetes (K8s) master nodes), as well as associated storage and networking infrastructure.

An overview of the hardware is in Table 1. Details about the hardware that can be used and how it should be cabled are given in the *NVIDIA DGX BasePOD Reference Architecture*.

This deployment guide describes the steps necessary for configuring and testing a four-node DGX BasePOD after the physical installation has taken place.

Table 1. DGX BasePOD components

Component	Technology
Compute nodes	DGX A100 system
Compute fabric	NVIDIA Quantum QM8700 HDR 200 Gbps InfiniBand
Management fabric	NVIDIA SN4600 switches
Storage fabric	NVIDIA SN4600 switch for Ethernet attached storage NVIDIA Quantum QM8700 HDR 200 Gb/s for InfiniBand attached storage
Out-of-band management fabric	NVIDIA SN2201 switches
Control plane	Minimum Requirements (each server): > 64-bit x86 processor, AMD EPYC 7272 or equivalent > 256 GB memory > 1 TB SSD > Two 100 Gbps network ports

1.2 Networking

This section covers the DGX system network ports and an overview of the networks used by DGX BasePOD.

1.2.1 DGX A100 System Network Ports

Figure 1 shows the rear of the DGX A100 system with the network port configuration used in this solution guide.

The following ports are selected for DGX BasePOD networking:

- Four single-port ConnectX-6 cards are used for the InfiniBand compute fabric, two on either side of the chassis (marked in red).
- Two ports of the dual-port ConnectX-6 cards are configured as a bonded Ethernet interface for in-band management and storage networks. These are the bottom port from slot 4 and the right port from slot 5 (marked in blue).
- BMC network access is provided through the out-of-band network (marked in gray).

Figure 1. DGX A100 system rear

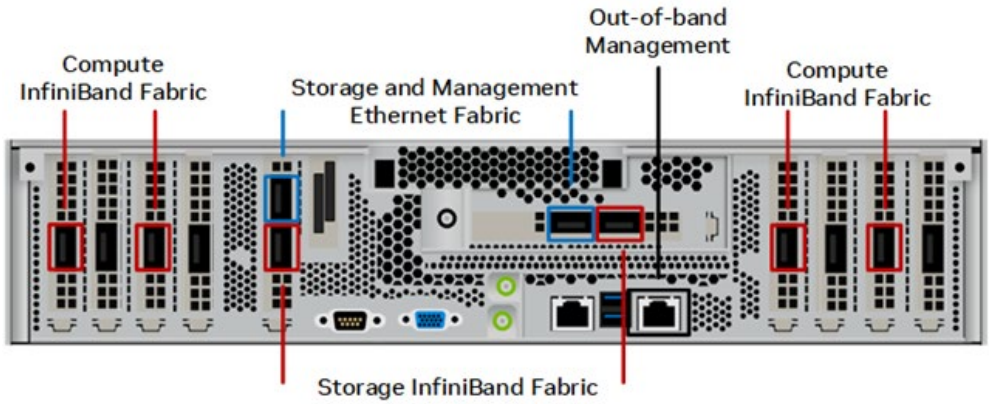


Table 2 details the naming and mapping for the ConnectX-6 interfaces.

Table 2. Interface naming and mapping

Slot Number	PCI Bus Number	Port Destination	InfiniBand RDMA Port
0	0000:4b:00.0	ibp75s0	mlx5_1
2	0000:ba:00.0	ibp186s0	mlx5_5
4 top	0000:e1:00.0	ibp225s0f0	mlx5_6
4 bottom	0000:e1:00.1	enp225s0f1	mlx5_7
5 left	0000:61:00.0	ibp97s0f0	mlx5_2
5 right	0000:61:00.1	enp97s0f1	mlx5_3
6	0000:0c:00.0	ibp12s0	mlx5_0
8	0000:8d:00.0	ibp141s0	mlx5_4

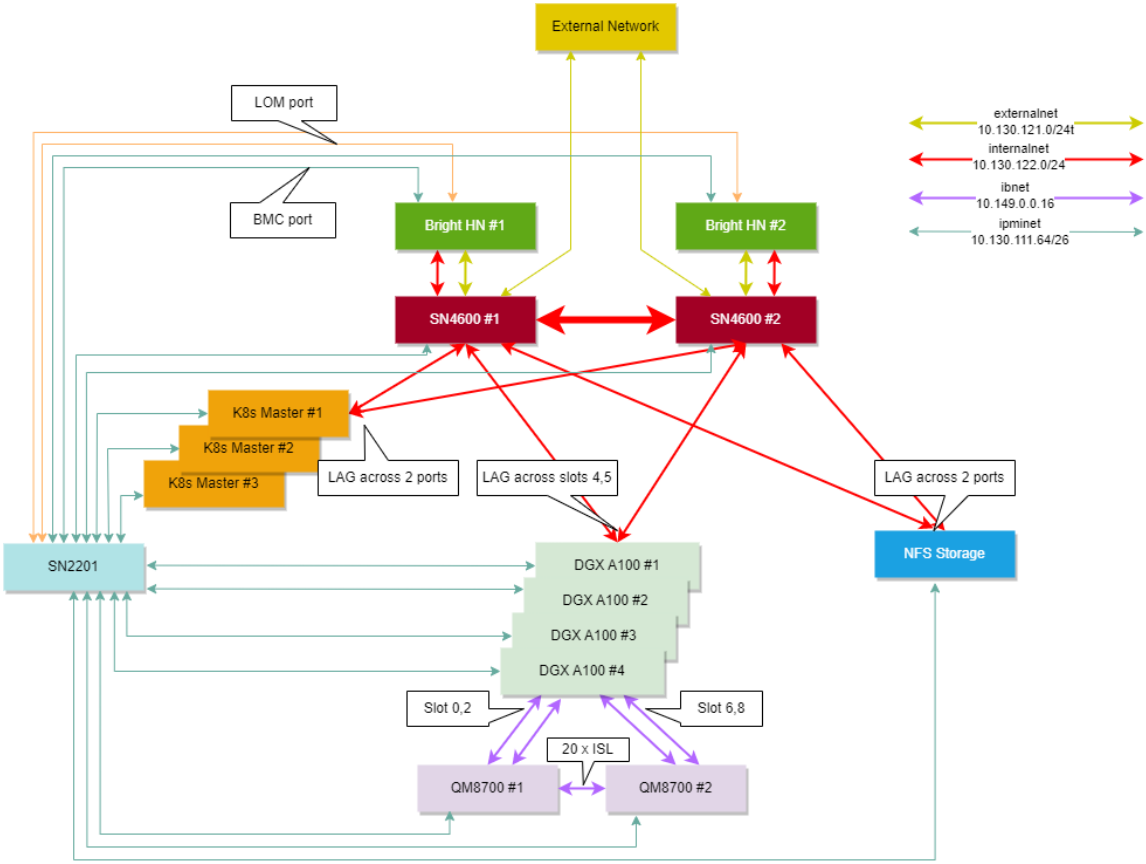
1.2.2 DGX BasePOD Network Overview

There are four networks in a DGX BasePOD configuration:

- `internalnet`—Network used exclusively within the cluster, for storage and in-band management.
- `externalnet`—Network connecting the DGX BasePOD to an external network, such as a corporate or campus network.
- `ipminet`—Network for out of band management, connecting BMCs.
- `ibnet`—InfiniBand network connecting all DGX systems ConnectX-6 Compute Fabric HCAs.

These are shown in Figure 2.

Figure 2. Network design and topology diagram



1.2.3 internalnet and externalnet

`internalnet` uses VLAN 122 and `externalnet` uses VLAN 121. Both VLANs are configured on the SN 4600 switches, which are the backbone of the DGX BasePOD Ethernet networking. Each DGX system connects to the SN4600 switches with a bonded interface consist of two physical interfaces, slot 4 bottom port (storage 4-2) and slot 5 right port (storage 5-2).

The K8s master nodes and the NFS storage device have a similar bonded interface configuration connected to SN4600 switches. Two SN4600 switches with Multi-chassis Link Aggregation (MLAG) provides the redundancy for DGX systems, K8s master nodes, and other devices with bonded interfaces. Trunk mode is used to bond the interface with VLAN 122 as its native VLAN. Access mode is used on the port connected to the Bright head node. BGP protocols used between interfaces are described in Table 3. All connected subnets are redistributed into BGP.

Table 3. BGP protocols

Protocol	Description
BGP	Used as required for routing between switches
iBGP	Configured between the two SN4600s using the MLAG `peerlink.4094` interface
eBGP	Configured between the uplink (SN4600) and IPMI (SN2201) switches

1.2.4 ipminet

On the `ipminet` switches, the gateway for VLAN 111 is configured and all the ports connected to the end hosts are configured as access ports for VLAN 111. Each Bright head node requires two interfaces connected to the IPMI switch; the first for IPMI interface of the host and the second to be used as HOST OS's direct access to the IPMI subnet. Uplinks are connected to TOR-01 and TOR-02 using unnumbered eBGP. All connected subnets are redistributed into BGP. IPMI switch can also be uplinked to separate management network if required, rather than the TOR switches; still IPMI subnet route must be advertised to the in-band network so that Bright can control hosts using the IPMI network.

1.2.5 ibnet

For the `ibnet`, NICs on physical DGX slot 0 and 2 are connected to QM8700-1 InfiniBand switch; and the NICs on physical DGX slot 6 and 8 are connected to QM8700-2 InfiniBand switch. To manage the InfiniBand fabric, a subnet manager is required; one of the 8700 switches must be configured as the subnet manager.

1.3 Software

Base Command Manager (BCM) is a key software component of DGX BasePOD. BCM is used to provision the OS on all hosts, deploy K8s, optionally deploy Jupyter, and provide monitoring and visibility of the cluster health.

An instance of BCM runs on a pair of head nodes in an HA configuration and is connected to all other nodes in the DGX BasePOD.

DGX systems within a DGX BasePOD have a DGX OS image installed by BCM. Similarly, the K8s master nodes are imaged by BCM with an Ubuntu LTS version equivalent to that of the DGX OS and the head nodes themselves.

1.3.1 K8s

K8s is an open-source platform for automating deployment, scaling, and operations of application containers across clusters of hosts. With K8s, it is possible to:

- Scale applications on the fly.
- Seamlessly update running services.
- Optimize hardware availability by using only the needed resources.

The cluster manager provides the administrator with the required packages, allows K8s to be set up, and manages and monitors K8s.

1.3.2 Jupyter (Optional)

BCM can optionally deploy and manage Jupyter, consisting of four major components and several extensions. The major components are: Jupyter Notebook, JupyterLab, JupyterHub, and Jupyter Enterprise Gateway.

These are the Jupyter extensions that BCM deploys:

- Template specialization extension—create custom Jupyter kernel without editing text files.
- Job management extension—manage jobs from within the Jupyter interface.
- VNC extension—interact with the X display of the execution server (including the desktop) from within the Jupyter interface.
- K8s operators—Jupyter kernel, PostgreSQL, and Spark operators.
- Jupyter dev server—Proxy server that enables developing applications in alternative editors while the computational workload is proxied to their Jupyter notebook running on the cluster.

1.4 Storage

An NFS server is required for a highly available (HA) BCM installation, and the required export path for that is described in this DGX BasePOD document. A DGX BasePOD typically also includes dedicated storage, but the configuration of that is outside the scope of this document. Contact the vendor of the storage solution being used for instructions on configuring the high-performance storage portions of a DGX BasePOD.

Chapter 2. Deployment

Deployment of a DGX BasePOD involves pre-setup, deployment, and use of BCM to provision the K8s cluster, and optionally deploy Jupyter.

1. Prepare the infrastructure.

   Physical installation should be completed before using this document, along with capturing information about the intended deployment in a site survey. Refer to Appendix A for the example site survey used by this document.

2. Configure the networking switches.

   Refer to Appendix B for the example configuration used by this document. Specifics on connecting to and configuring the switches can be found in their associated user guides.

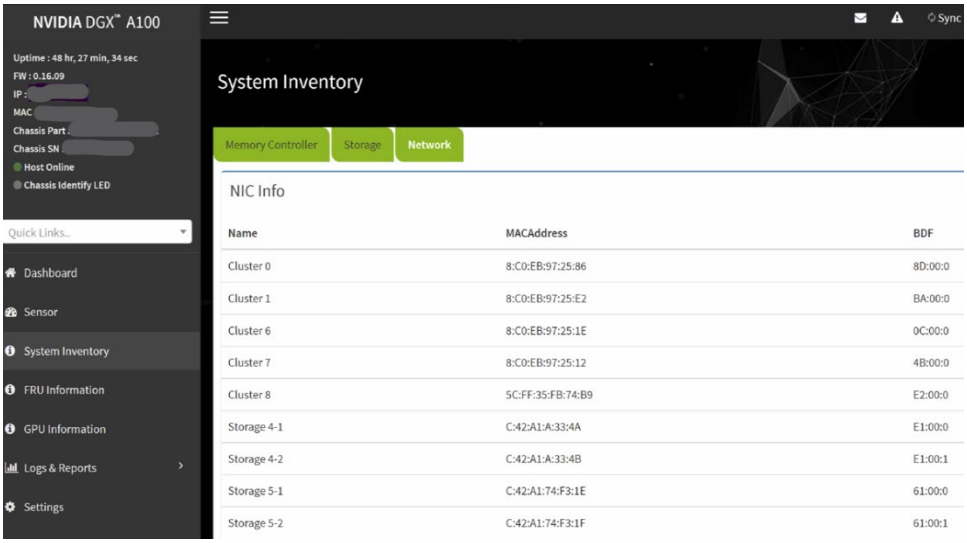
3. Configure the NFS server.
   a. As stated in Section 1.4, NFS configuration steps are not in scope for this document.
   b. This DGX BasePOD deployment uses the `/var/nfs/general`, which is the NFS export path provided in Table 3 of the Site Survey.
   c. Use the following parameters for the NFS server export file `/etc/exports`

```
/var/nfs/general *(rw,sync,no_root_squash,no_subtree_check)
```

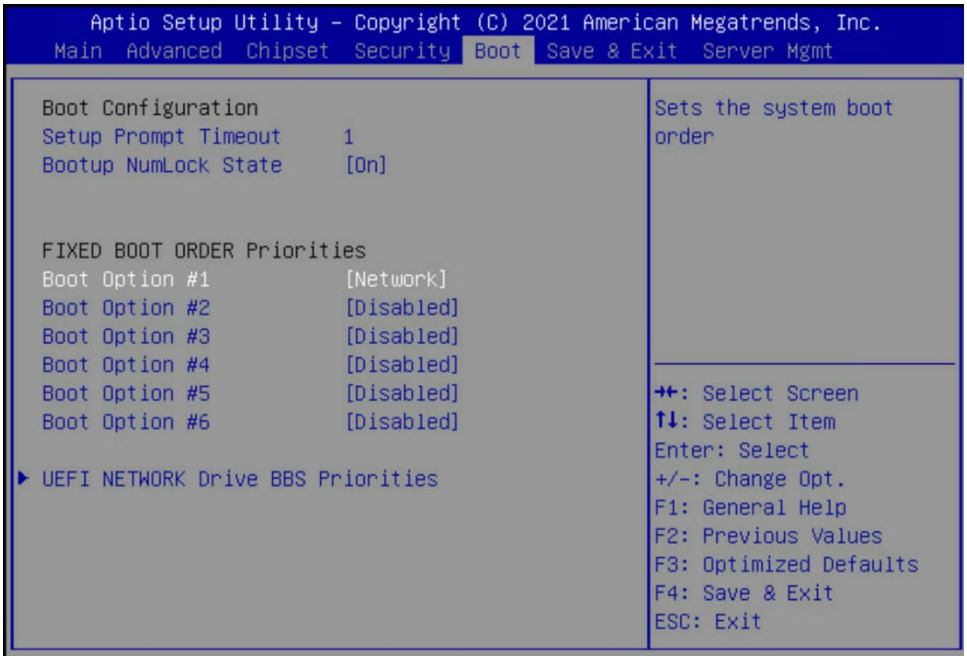
4. Set the DGX BIOS so that the DGX systems PXE boot by default.

   BCM requires DGX systems to PXE boot.
   a. Connect to the BMC of the DGX system.

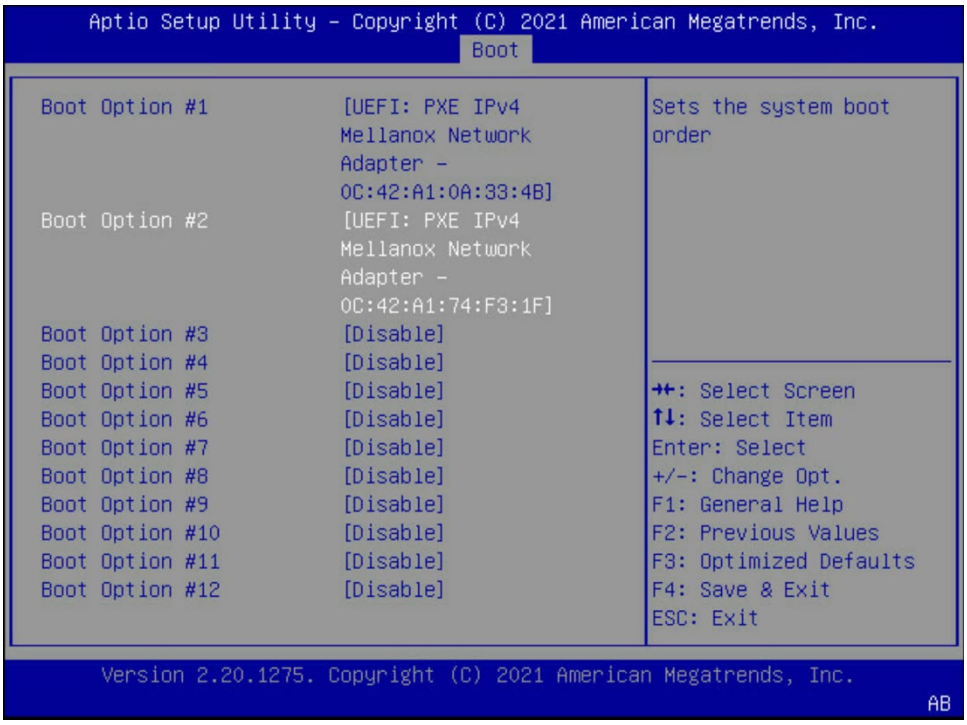
b. In the `Network` tab of the `System Inventory` window, locate the MAC addresses for the `Storage 4-2` and `Storage 5-2` interfaces.



c. In the DGX A100 system BIOS, configure `Boot Option #1` to be `[NETWORK]`. Set other Boot devices to `[DISABLED]`.

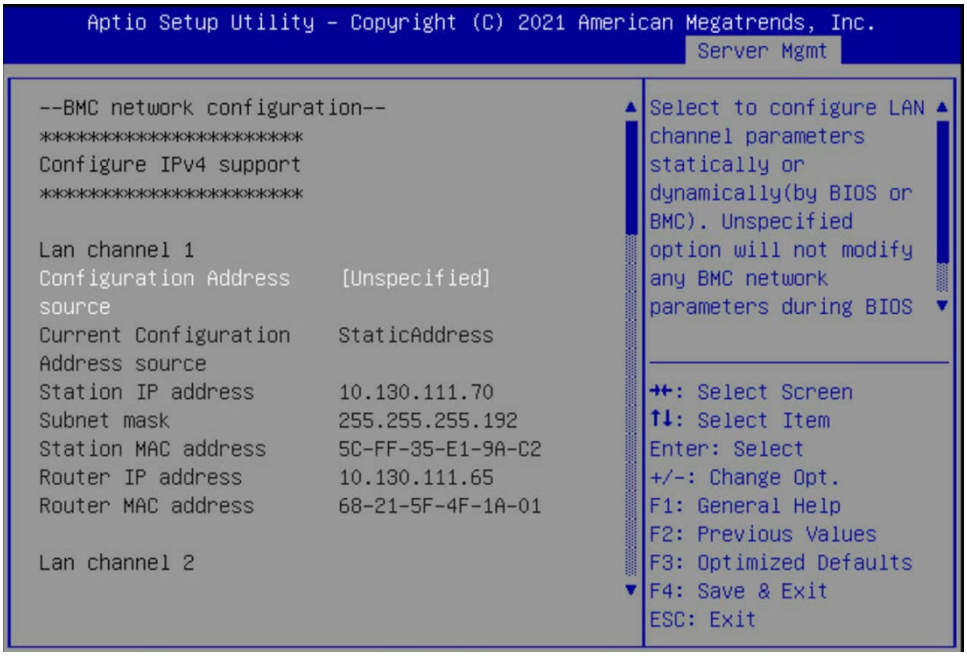


d. Disable PXE boot devices except for `Storage 4-2` and `Storage 5-2`. Set them to use IPv4.



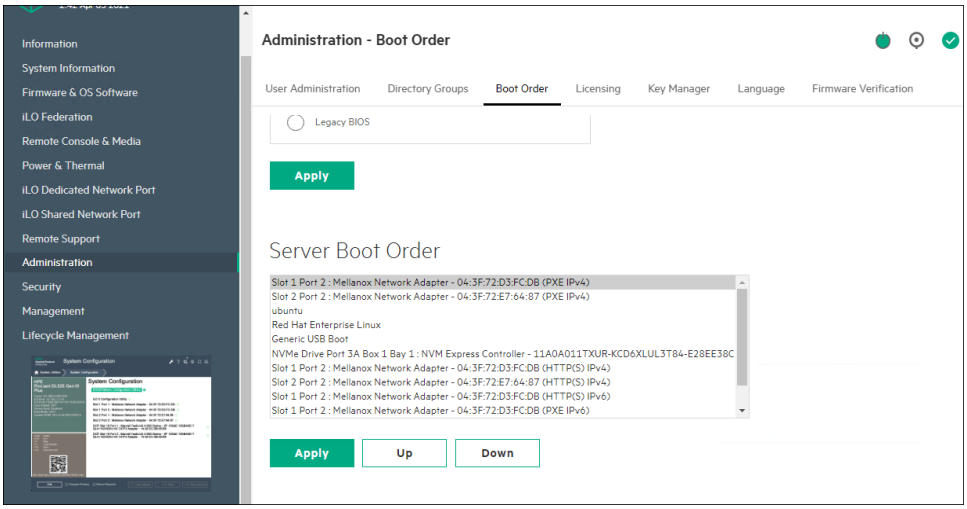
e. Configure a static IP address for the BMC.

Navigate to the `Server Mgmt` tab of the BIOS, enter the `BMC network configuration menu`, then set the IPv4 `Lan channel 1` Configuration Address Source option to `StaticAddress`, enter the IP address, subnet, and gateway/router information.

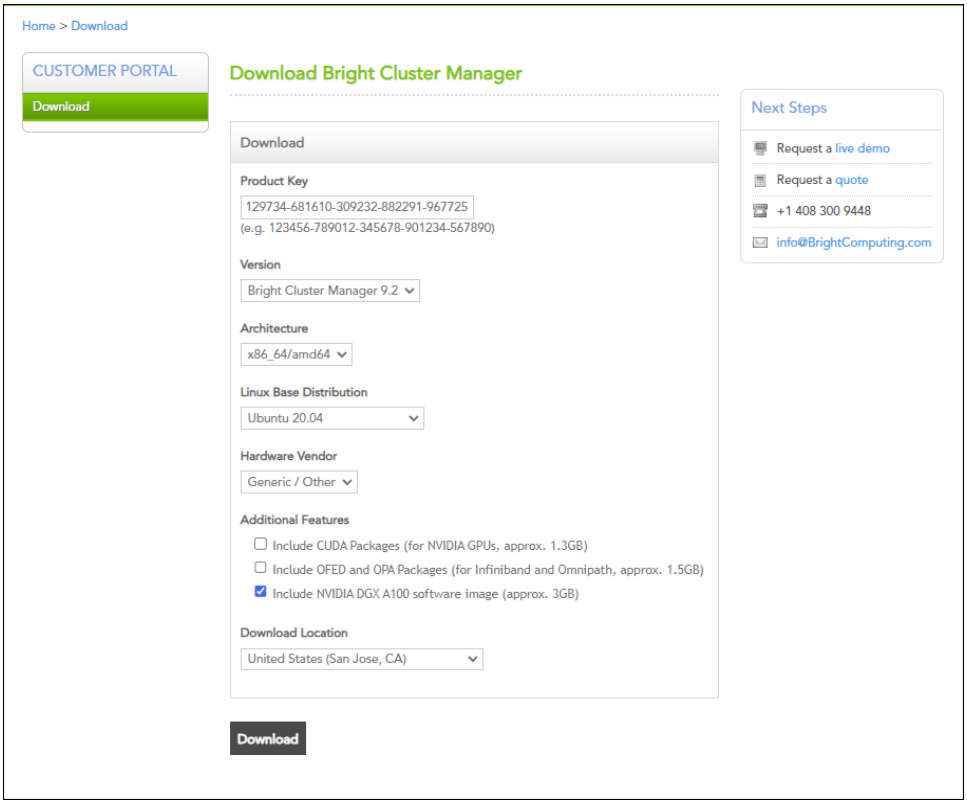


5. Ensure that the Network boot option is configured as the primary boot option for the K8s master nodes that are to be used for this cluster.

   This is an example of a system that will boot from the network with `Slot 1 Port 2` and `Slot 2 Port 2`.

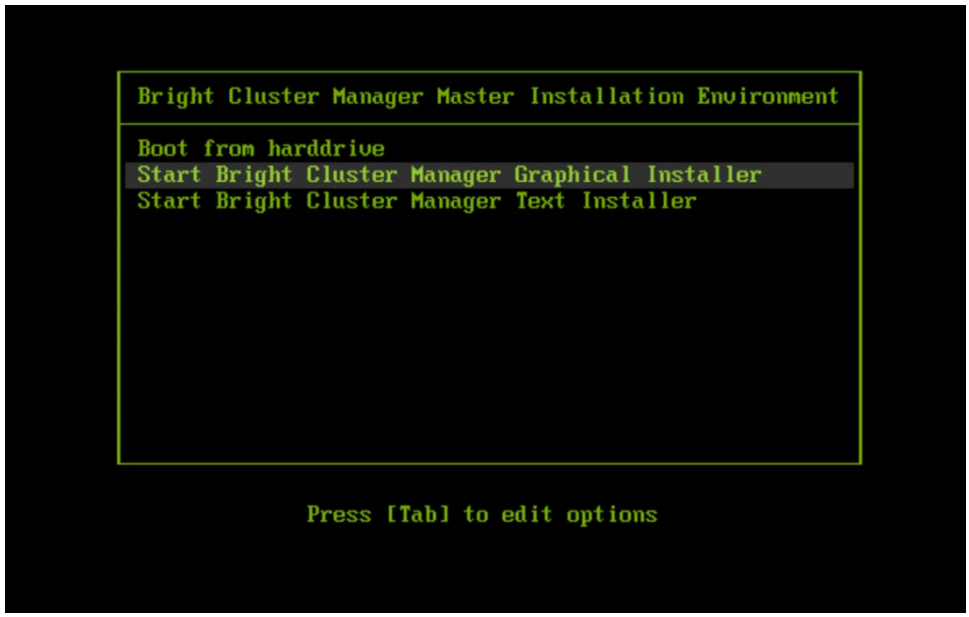


6. Download a BCM ISO from the Bright download site. Select Base Command Manager 9.2, Ubuntu 20.04, and check the `Include NVIDIA DGX A100 Software image` checkbox.



7. Burn the ISO to a DVD or to a bootable USB device.

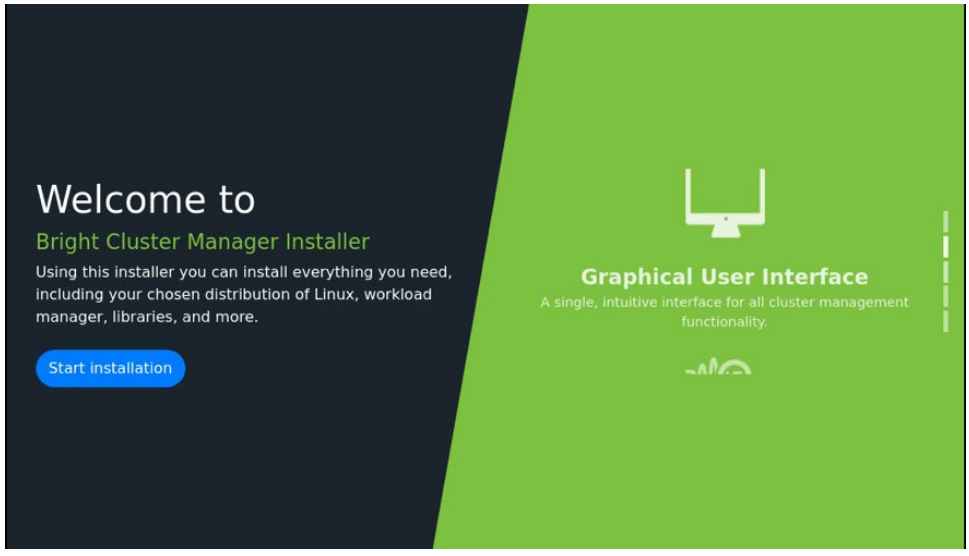
   It can also be mounted as virtual media and installed using the BMC. The specific mechanism for the latter will vary by vendor.
8. Ensure that the BIOS of the target head node is configured in UEFI mode and that its boot order is configured to boot the media containing the Bright installer image.
9. Boot the installation media.
10. At the grub menu, choose `Start Bright Cluster Manager Graphical Installer`.



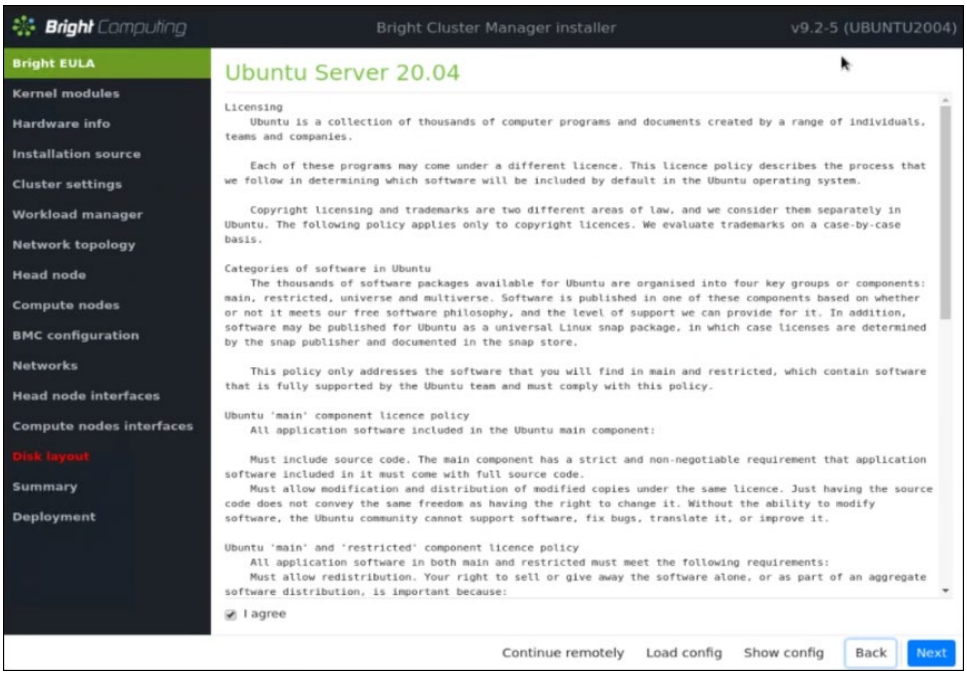
11. When the `Welcome to Base Command Manager` splash screen is displayed, select `Start installation`.

    If the splash screen does not launch, launch it from a terminal.

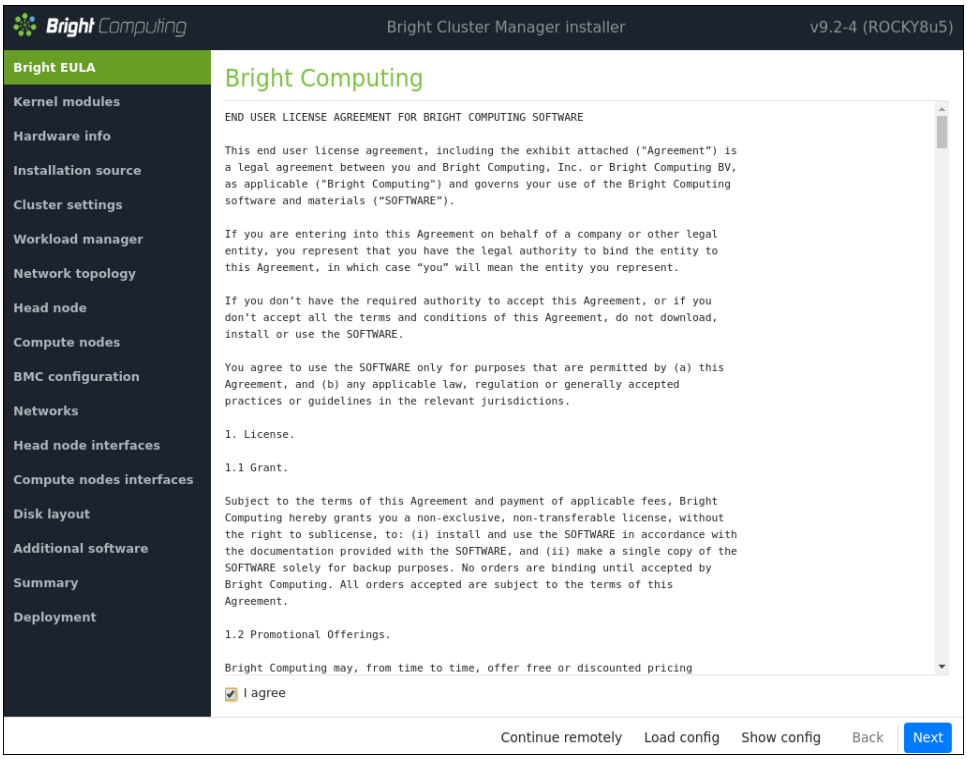
```
systemctl restart bright-installer-configure
```



12. Accept the Bright EULA.

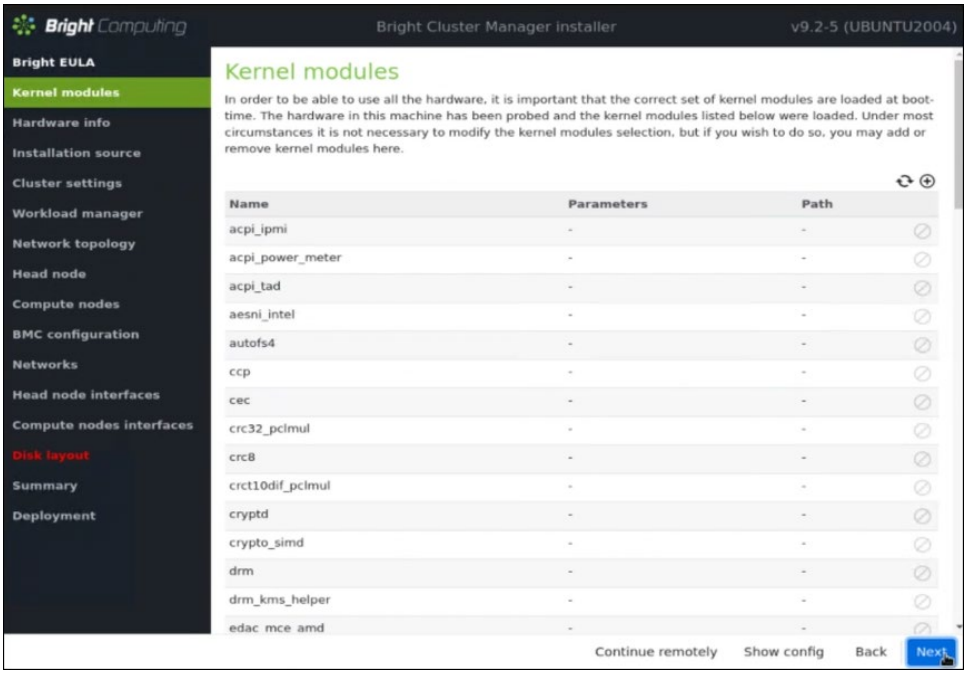


13. Accept the OS EULA.



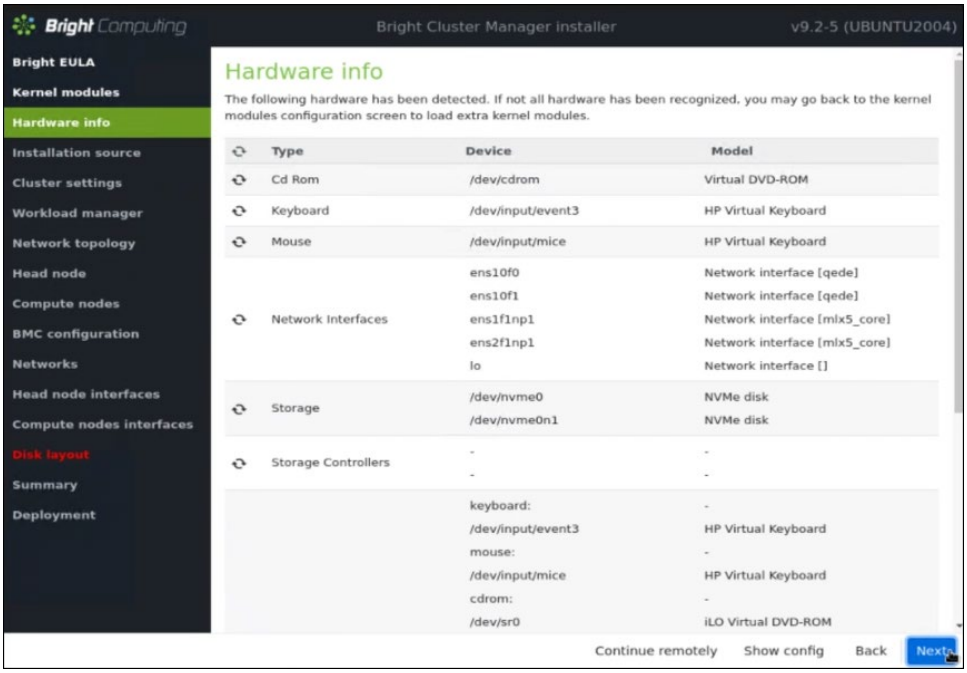
14. Configure the `Kernel modules` screen.

    Leave the default settings unless otherwise instructed by NVIDIA personnel.



15. Configure the `Hardware info` screen.

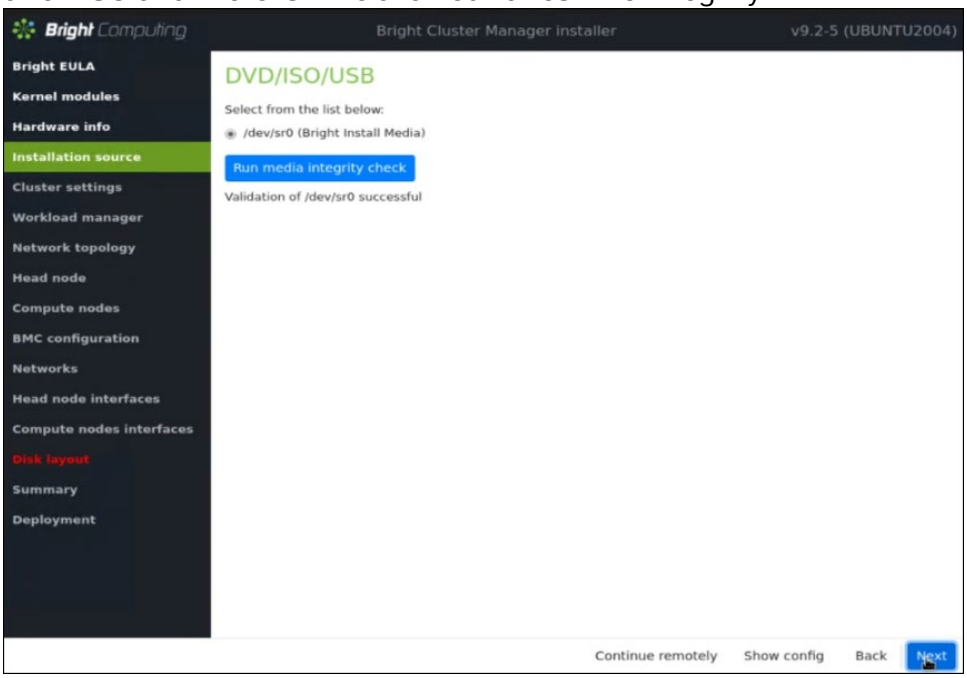
    Verify that the target storage device and the cabled host network interfaces are correctly displayed.



16. Configure the `Installation source` screen.

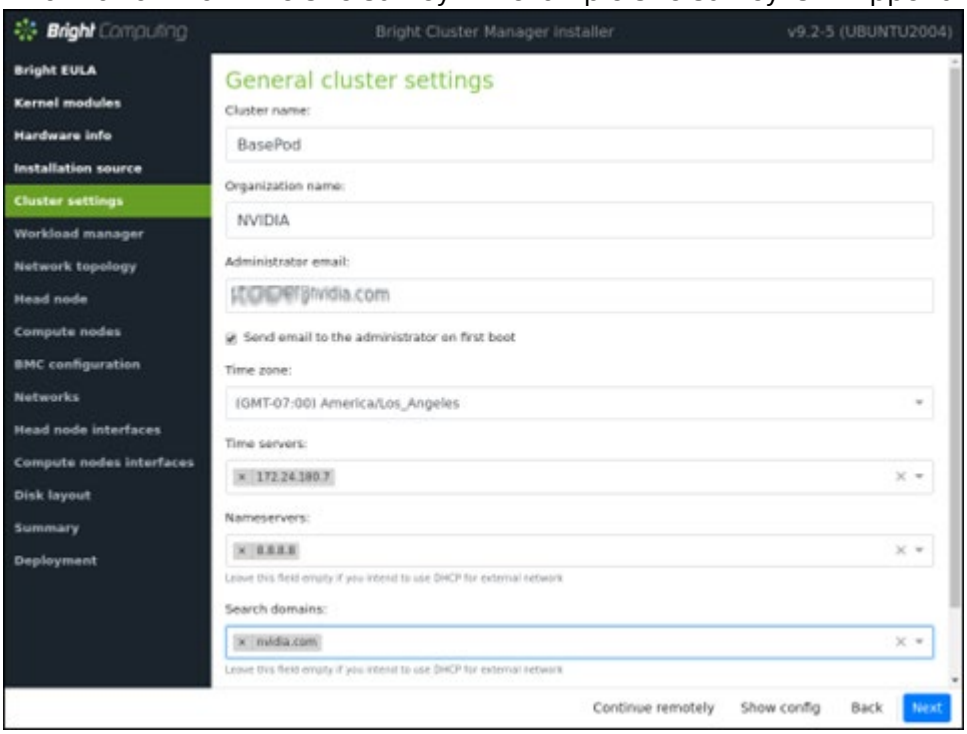
    Select the appropriate device.

    Run the media integrity check if there are doubts about the validity of the installation ISO and there is time allotted to test the integrity.



17. Configure the `Cluster settings` screen.

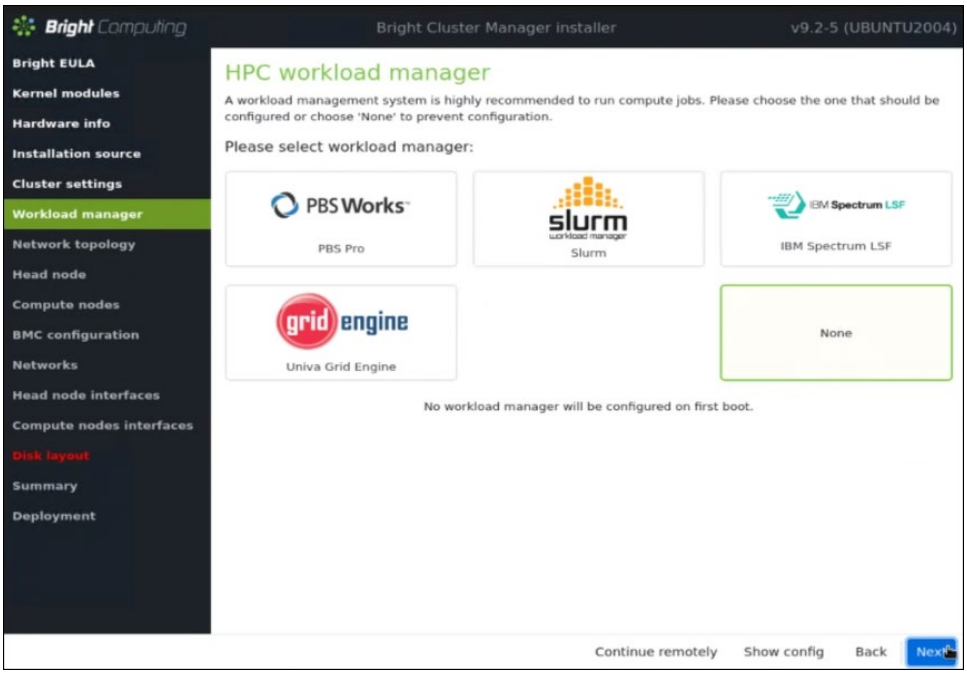
    Enter information from the site survey. An example site survey is in Appendix A.



18. Configure `Workload manager` screen.

    Select `None`.

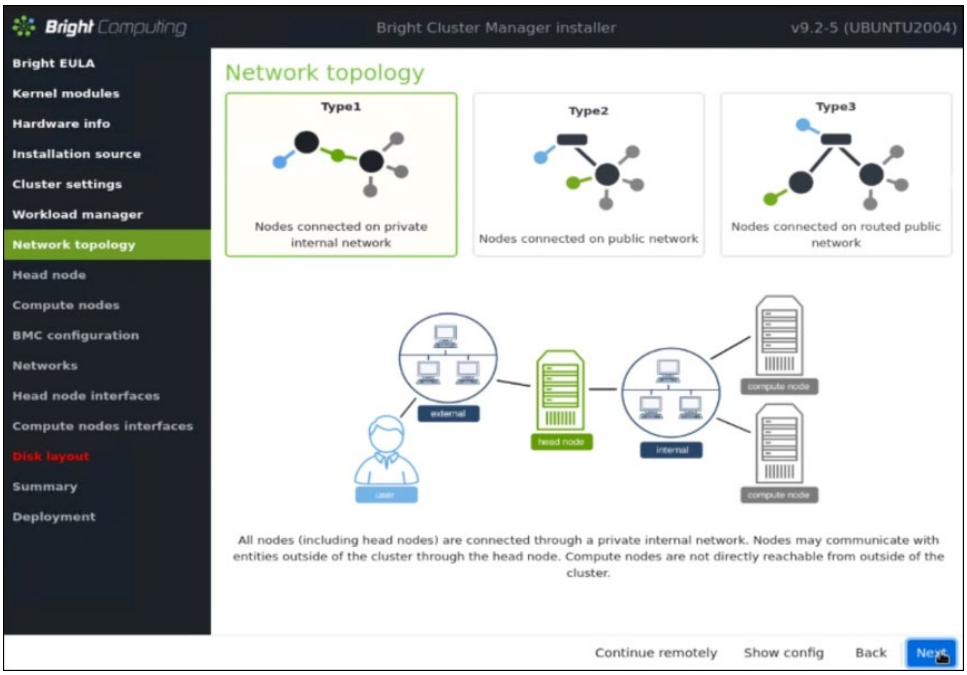
    After head node installation, K8s will be deployed for container orchestration.



19. Configure `Network topology` screen.

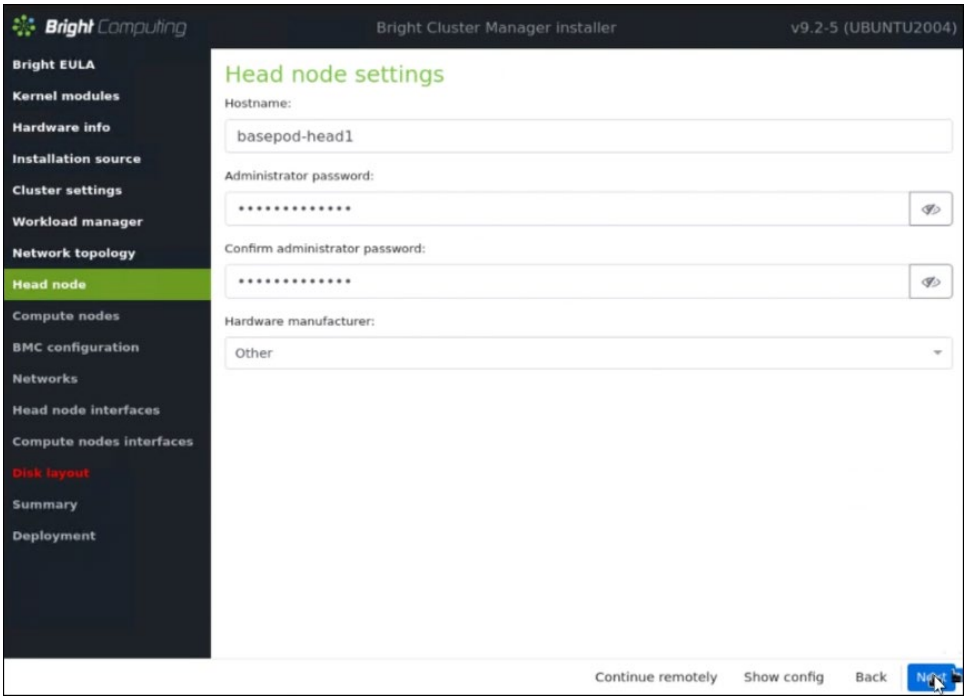
    Select `Type1`.

    In a DGX BasePOD architecture, the cluster nodes are connected to the head node over the internal network, with the head node serving as their default gateway.



20. Configure `Head node settings` screen.

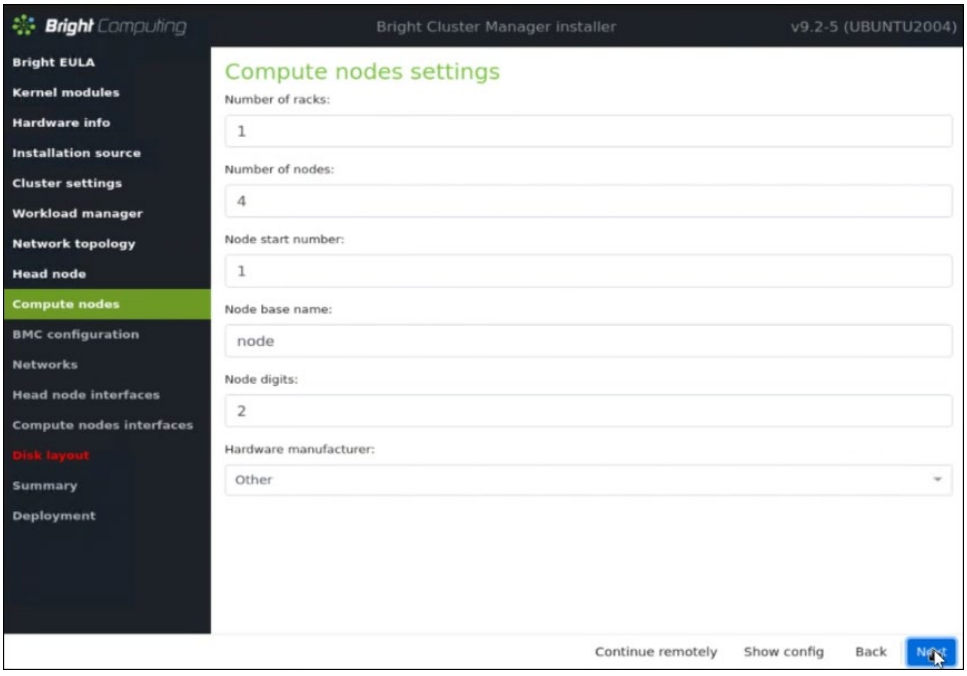
    Enter the `Hostname` and `Administrator password`.



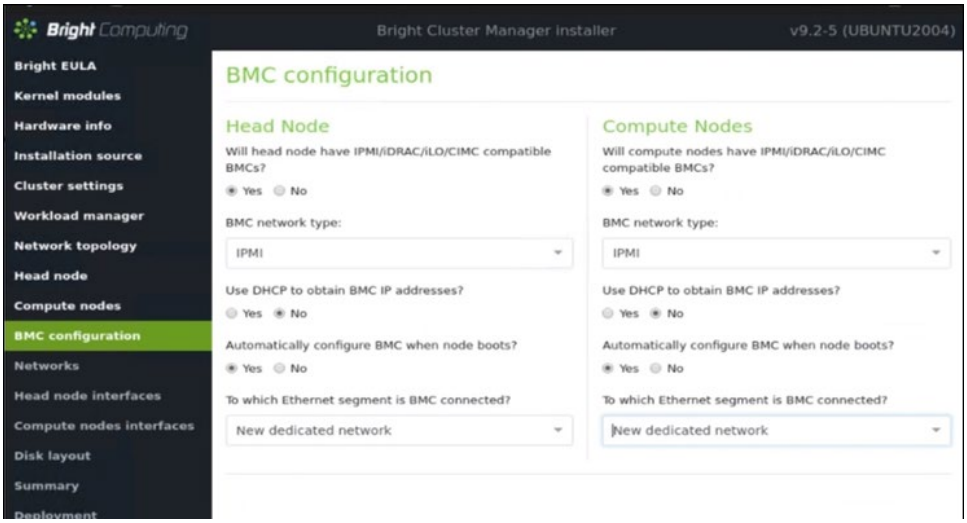
21. Configure `Compute nodes` screen.

    Set the `Number of nodes` to `4`

    Set `Node digits` to `2`.

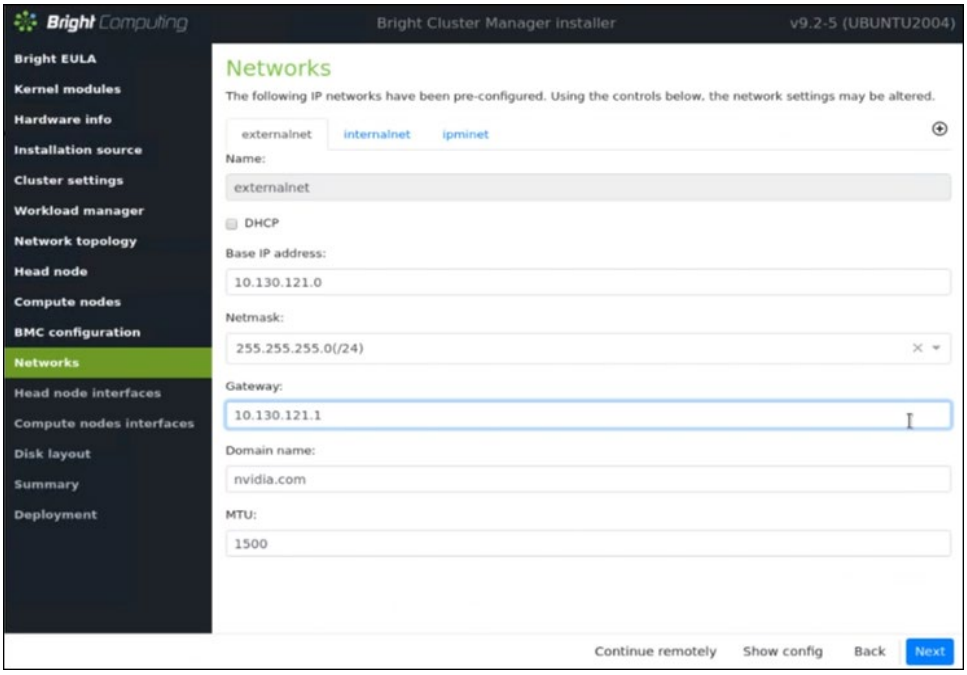


22. Configure `BMC configuration` screen.
    a. Select `Yes` for both the `Head Node` and the `Compute Nodes`.
    b. Select `IPMI` from the `BMC network type` select lists for both the `Head Node` and the `Compute Nodes`.
    c. Select `No` to the DHCP question for both node types.
    d. Select `Yes` for `Automatically configure BMC when node boots?`.
    e. Select `New dedicated network` from the `To which Ethernet segment is BMC connected?` select list.



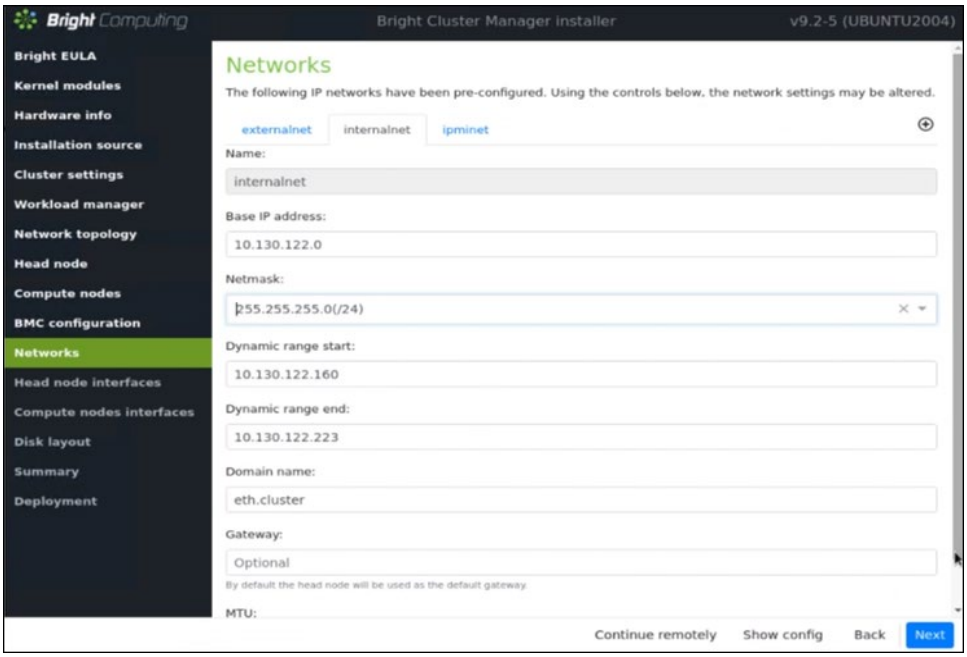
23. Configure the `Networks` screens.
    a. `externalnet`

    Set the `Base IP address`, `Netmask`, `Gateway`, and `Domain name` according to the site survey.



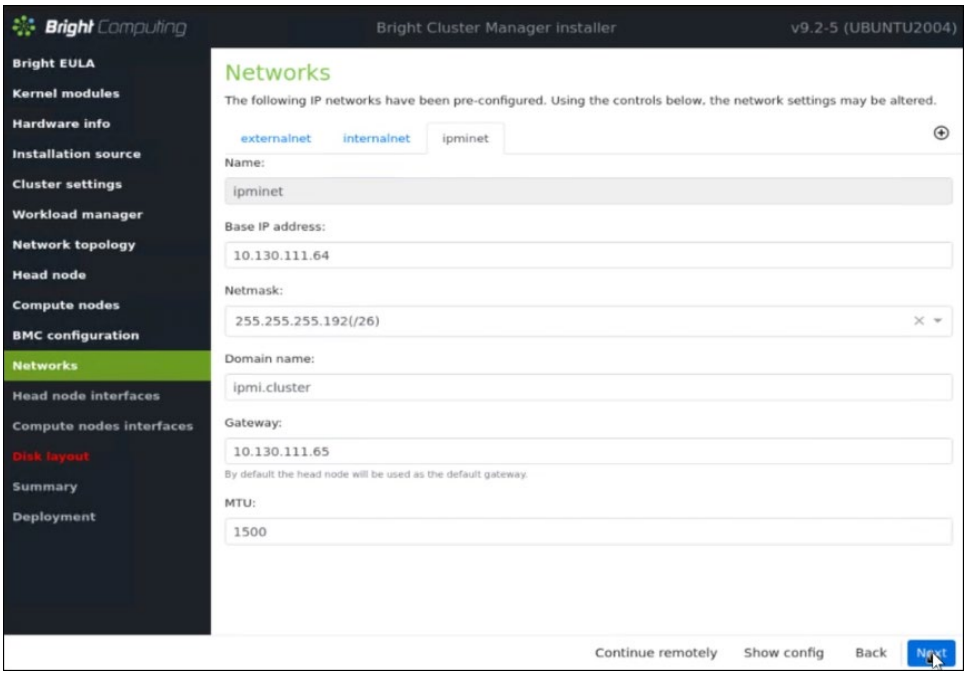
b. `internalnet`

Set the `Base IP address` and `Netmask` according to the site survey.



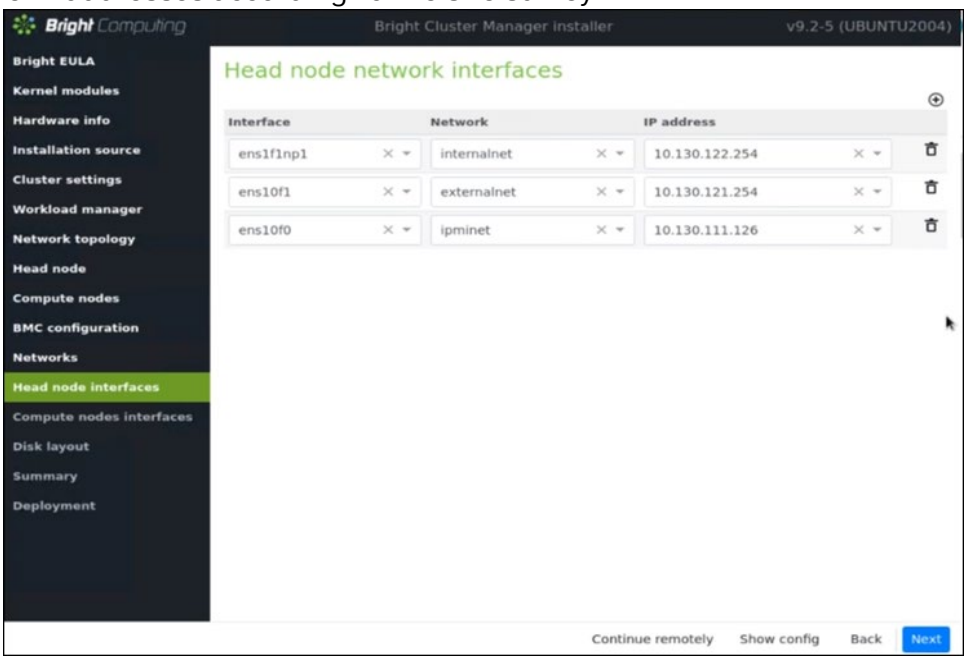
c. `ipminet`

Set the `Base IP address`, `Netmask`, and `Gateway` according to the site survey.



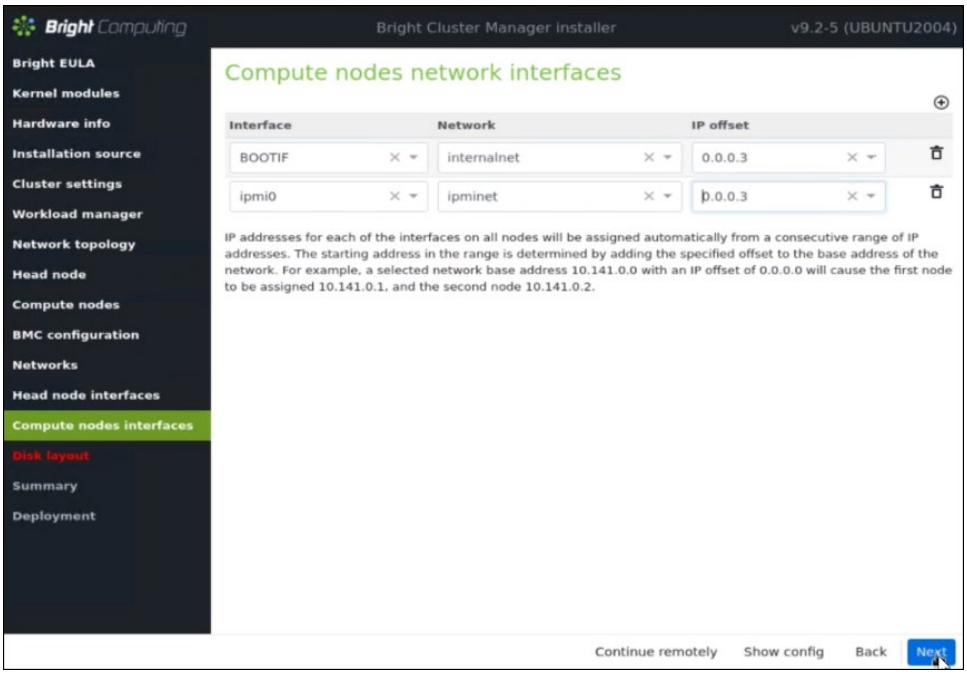
24. Configure the `Head node interfaces` screen.

    Set the IP addresses according to the site survey.

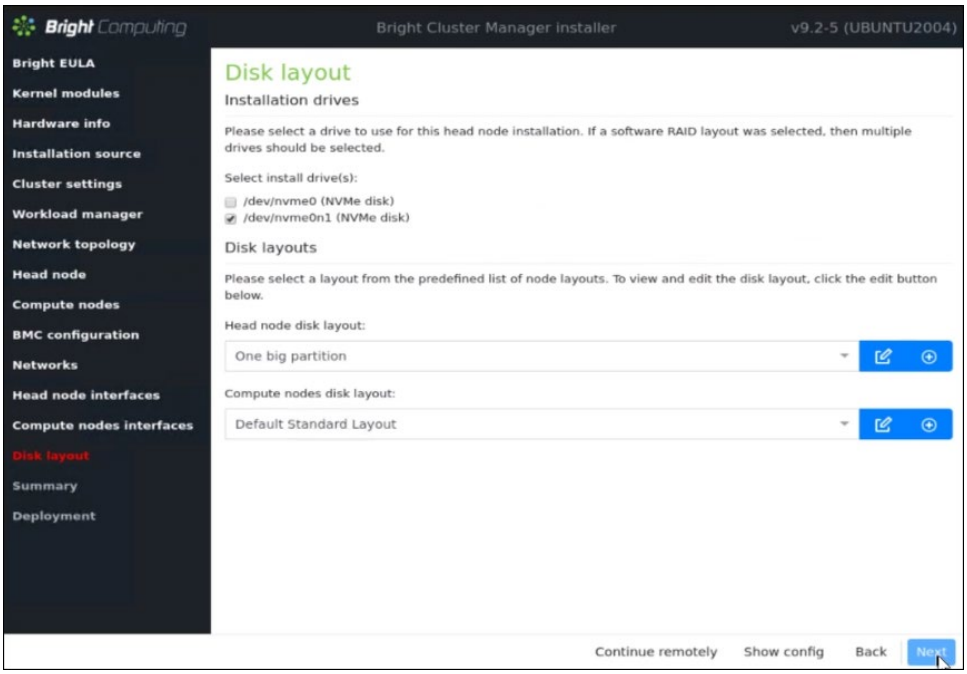


25. Configure the `Compute nodes network interfaces` screen.

    Set the offset for `BOOTIF` and `ipmi0` to `0.0.0.3`.

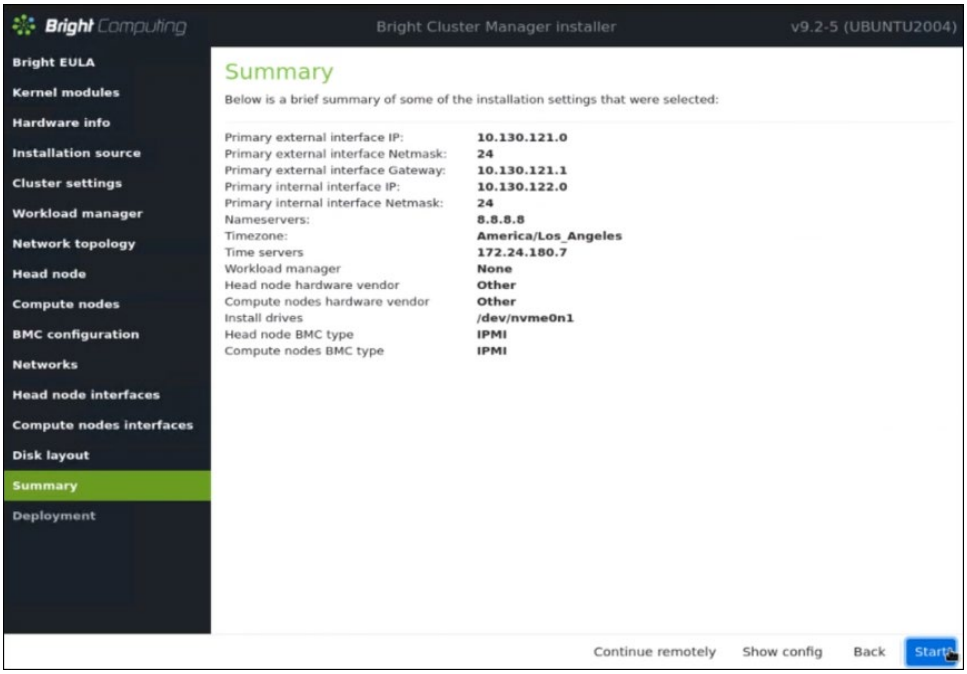


26. Configure the `Disk layout` screen.
    a. Set `Select install drives(s).`
    b. Set the `Head node disk layout`.

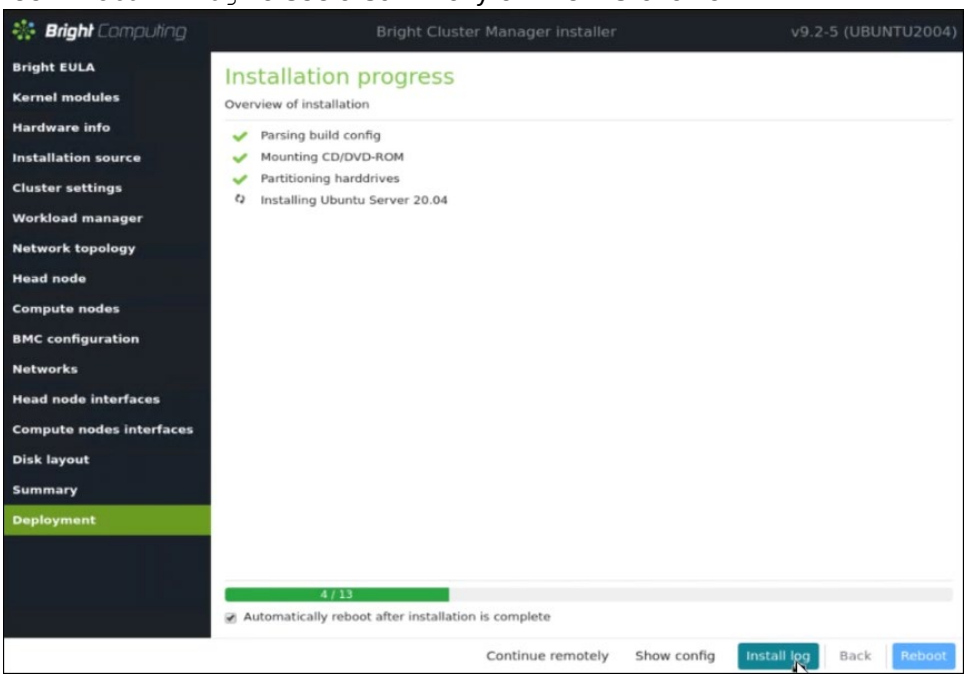


27. Review the `Summary` screen.

    This is a sample `Summary` screen.



28. Configure the `Deployment` screen.
    a. Check the `Automatically reboot after installation is complete` checkbox to reboot the host upon successful completion of the deployment.
    b. Select `Install log` to see a summary of the installation.



2.1 Cluster Configuration

1. Log in to the BCM head node assigned to `externalnet`.

```
ssh <externalnet>
```

2. Install the cluster license by running the `request-license` command.

   Because HA is used, specify the MAC address of the first NIC of the secondary head node so that it can also serve the Bright licenses in the event of a failover.

   This example is for a head node with Internet access. For air-gapped clusters, see "Off-cluster WWW access" in Section 4.3.3 of the Bright Installation manual.

```
# request-license
Product Key (XXXXXX-XXXXXX-XXXXXX-XXXXXX-XXXXXX): 123456-123456-123456-123456
Country Name (2 letter code): US
State or Province Name (full name): California
Locality Name (e.g. city): Santa Clara
Organization Name (e.g. company): NVIDIA
Organizational Unit Name (e.g. department): MLOPs
Cluster Name: BasePOD
Private key data saved to /cm/local/apps/cmd/etc/cluster.key.new

Warning: Permanently added 'basepod-head1' (ECDSA) to the list of known hosts.
MAC Address of primary head node (basepod-head1) for ens1f1np1
[04:3F:72:E7:67:07]:
Will this cluster use a high-availability setup with 2 head nodes? [y/N] y
```

```
MAC Address of secondary head node for eth0 [XX:XX:XX:XX:XX:XX]:
14:02:EC:DA:AF:18

Certificate request data saved to /cm/local/apps/cmd/etc/cluster.csr.new
Submit certificate request to
http://licensing.brightcomputing.com/licensing/index.cgi ? [Y/n] Y
Contacting http://licensing.brightcomputing.com/licensing/index.cgi...

License granted.
License data was saved to /cm/local/apps/cmd/etc/cluster.pem.new
Install license? [Y/n] Y
========= Certificate Information ========
Version:                          7.0 and above
Edition:                          Advanced
Common name:                      BasePOD
Organization:                     NVIDIA
Organizational unit:              Sales
Locality:                         Santa Clara
State:                            California
Country:                          US
Serial:                           1243145
Starting date:                    11 Jan 2022
Expiration date:                  07 Jan 2023
MAC address / Cloud ID:           04:3F:72:E7:67:07|14:02:EC:DA:AF:18
Licensed nodes:                   32
Pay-per-use nodes:                Yes
Licensed nodes with accelerators: 1024
Accounting & Reporting:           Yes
Allow edge sites:                 Yes
License type:                     Commercial
==========================================
Is the license information correct ? [Y/n] Y
…
```

3. Backup the default software image.

   The backup image can be used to create additional software images.

```
# cmsh
% softwareimage
% clone default-image default-image-orig
% commit
```

Note: This document uses # to indicate commands executed as the root user on a head node, and % to indicate commands executed within `cmsh`. The prompt change is in the preceding block. If it is unclear where a command is being executed, check the prompt that precedes it.

   Wait for the ramdisk to be regenerated and the following text to be displayed.

```
Tue Oct 25 16:24:03 2022 [notice] basepod-head1: Initial ramdisk for image
default-image-orig was generated successfully
```

4. Backup the DGX software image.

   The backup image can be used to create additional software images.

```
% softwareimage
% clone dgx-a100-image dgx-a100-image-orig
% commit
```

   Wait for the ramdisk to be regenerated and the following text to be displayed.

```
Tue Oct 25 16:28:13 2022 [notice] basepod-head1: Initial ramdisk for image dgx-a100-image-orig was generated successfully
```

5. Create the K8s software image by cloning the default software image.

   This software image will be further configured and provisioned onto the K8s master nodes.

```
% softwareimage
% clone default-image k8s-master-image
% commit
```

6. Add the required kernel modules to the `dgx-a100-image` software image.

```
% softwareimage
% use dgx-a100-image
% kernelmodules
% add mlx5_core
% add bonding
% add raid0
% add raid1
% softwareimage commit
```

7. Add the required kernel modules to the `k8s-master-image` software image.

```
% /
% softwareimage
% use k8s-master-image
% kernelmodules
% add mlx5_core
% add bonding
% softwareimage commit
```

8. Create the `dgx` node category and assign the `dgx-a100-image` software image to it.

   All nodes assigned to the `dgx` category will be provisioned with the `dgx-100-image` software image.

```
% category
% clone default dgx
% set softwareimage dgx-a100-image
% commit
```

9. Create the `k8s-master` node category and assign the `k8s-master-image` software image to it.

   All nodes assigned to the `k8s-master` category will be provisioned with the `k8s-master-image` software image.

```
% category
% clone default k8s-master
% set softwareimage k8s-master-image
% commit
```

10. Create the DGX nodes.

    `node01` was created during head node installation. Clone `node01` to create the DGX nodes, which will initially be named `node02`, `node03`, `node04`, and `node05`.

```
% device
% foreach --clone node01 -n node02..node05 ()
% commit
```

11. Rename the DGX nodes so they are more easily identified later.

```
% use node02
% set hostname dgx01
% use node03
% set hostname dgx02
% use node04
% set hostname dgx03
% use node05
% set hostname dgx04
% device commit
```

12. Clone `node01` to create the K8s control plane nodes, which will initially be named `node05`, `node06` and `node07`.

```
% device
% foreach --clone node01 -n node06..node08 ()
% commit
```

13. Rename the K8s master nodes so they are more easily identifiable.

```
% device
% use node06
% set hostname knode01
% use node07
% set hostname knode02
% use node08
% set hostname knode03
% device commit
```

14. Rename `node01`.

    The purpose of this step is to specify that `node01` is only a template.

```
% device
% use node01
% set hostname template01
% commit
```

15. Assign the DGX nodes to the DGX node category.

```
% foreach -n dgx01..dgx04 (set category dgx)
```

16. Assign the K8S nodes to the `k8s-master` node category.

```
% foreach -n knode01..knode03 (set category k8s-master)
% commit
```

17. Check the nodes and their categories.

    Extra options are used for `device list` to make the format more readable.

```
% device list -f hostname:20,category:10,ip:20,status:15
hostname (key)       category       ip                   status
-------------------- ----------     -------------------- ---------------
basepod-head1                       10.130.122.254       [   UP   ]
dgx01                dgx            10.130.122.5         [  DOWN  ]
dgx02                dgx            10.130.122.6         [  DOWN  ]
dgx03                dgx            10.130.122.7         [  DOWN  ]
dgx04                dgx            10.130.122.8         [  DOWN  ]
knode01              k8s-master     10.130.122.9         [  DOWN  ]
knode02              k8s-master     10.130.122.10        [  DOWN  ]
knode03              k8s-master     10.130.122.11        [  DOWN  ]
template01           default        10.130.122.4         [  DOWN  ]
```

2.1.1 Network Configuration

1. Add a Network for InfiniBand (`ibnet`).

```
% network
% add ibnet
% set domainname ibnet.cluster.local
% set baseaddress 10.149.0.0
% set broadcastaddress 10.149.255.255
% set netmaskbits 16
% set mtu 2048
% commit
```

2. Verify the results.

```
% list -f name:20,type:10,netmaskbits:10,baseaddress:15,domainname:20
name (key)           type       netmaskbit baseaddress     domainname
-------------------- ---------- ---------- --------------- --------------------
externalnet          External   24         10.130.121.0    nvidia.com
globalnet            Global     0          0.0.0.0         cm.cluster
ibnet                Internal   16         10.149.0.0      ibnet.cluster.local
internalnet          Internal   24         10.130.122.0    eth.cluster
ipminet              Internal   26         10.130.111.64   ipmi.cluster
```

3. Set the IP address of the `ipmi0` (BMC) interface.

```
% use ipmi0
% set ip 10.130.111.66
% show
Parameter                      Value
------------------------------ ------------------------------------------
Revision
Type                           bmc
Network device name            ipmi0
Network                        ipminet
IP                             10.130.111.66
DHCP                           no
Alternative Hostname
Additional Hostnames
Start if                       always
BringUpDuringInstall           no
On network priority            10
Gateway                        0.0.0.0
VLAN ID                        0
LAN channel                    1
```

4. Verify that the OS-visible physical interface (`ens10f0`) is configured.

   This allows the head node to communicate on the `ipminet` network, and therefore, communicate with its own BMC.

```
% show
Parameter                      Value
------------------------------ ------------------------------------------
Revision
Type                           physical
Network device name            ens10f0
Network                        ipminet
IP                             10.130.111.126
DHCP                           no
Alternative Hostname
Additional Hostnames
```

```
Start if                          always
BringUpDuringInstall              no
On network priority               60
MAC                               00:00:00:00:00:00
Speed
Card Type
```

5. Add a physical interface if one is not shown in Step 4.

```
% add physical ens10f0
% set network ipminet
% set ip 10.130.111.126
% commit
```

6. Ensure that both head node interfaces are connected to `ipminet`.

```
% list
Type         Network device name  IP               Network          Start if
------------ -------------------- ---------------- ---------------- --------
bmc          ipmi0                10.130.111.66    ipminet          always
physical     ens10f0              10.130.111.126   ipminet          always
physical     ens10f1              10.130.121.254   externalnet      always
physical     ens1f1np1 [prov]     10.130.122.254   internalnet      always
```

7. Reboot the head node.

   A reboot of the head node is required because the network interfaces were changed.

```
% /
% device
% use master
% reboot
Reboot in progress for: basepod-head1
```

2.1.2 Configure Disk Layouts for Node Categories

Part of using BCM for managing nodes in a DGX BasePOD is to define the disk partitions. Each DGX BasePOD node category includes K8s master and DGX node categories, both of which must have their disk layout configured.

2.1.2.1 Configure Disk Layout for the k8s-master Category

1. Augment the `disksetup` of the k8s-master category.

   For the K8s master nodes, an EFI System Partition of 100 MB is created at the start of the disk, with the remainder of the disk dedicated to the OS as a single large partition. Note that this disk setup does not have a swap partition.

   The configuration file references `/dev/nvme0n1` as the block device used. This may need to be changed to match the specific device name used on systems intended as K8s master nodes.

   Save the following text to `/cm/local/apps/cmd/etc/htdocs/disk-setup/k8s-disksetup.xml`, factoring in any necessary changes specific to the target systems as noted.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<diskSetup xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <device>
```

```
    <blockdev>/dev/nvme0n1</blockdev>

    <partition id="a0" partitiontype="esp">
      <size>100M</size>
      <type>linux</type>
      <filesystem>fat</filesystem>
      <mountPoint>/boot/efi</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>
    <partition id="a1">
      <size>max</size>
      <type>linux</type>
      <filesystem>xfs</filesystem>
      <mountPoint>/</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>
  </device>
</diskSetup>
```

2. Assign this disk layout to the `k8s-master` node category.

```
$ cmsh
% category
% use k8s-master
% set disksetup /cm/local/apps/cmd/etc/htdocs/disk-setup/k8s-disksetup.xml
% commit
```

2.1.2.2 Configure Disk Layout for the DGX Node Category

The DGX A100 system contains two NVMe drives for a RAID 1 boot volume, and an additional eight NVMe drives that are used as a RAID 0 volume for local, high-performance storage. The following disk configuration creates that disk layout, making use of the `/raid` directory that was created in the `dgx-a100-image` software image as the mount point for the local RAID 0 array.

1. Copy the following disk setup XML file to a file on the head node.

   In this example, it is copied to `/cm/local/apps/cmd/etc/htdocs/disk-setup/dgxa100-disksetup.xml`.

```
<?xml version="1.0" encoding="UTF-8"?>
<diskSetup>
  <device>
    <blockdev>/dev/nvme2n1</blockdev>
    <partition id="boot" partitiontype="esp">
      <size>512M</size>
      <type>linux</type>
      <filesystem>fat</filesystem>
      <mountPoint>/boot/efi</mountPoint>
      <mountOptions>defaults,noatime,nodiratime</mountOptions>
    </partition>

    <partition id="slash1">
      <size>max</size>
      <type>linux raid</type>
    </partition>
  </device>

  <device>
```

```
    <blockdev>/dev/nvme3n1</blockdev>
    <partition id="slash2">
      <size>max</size>
      <type>linux raid</type>
    </partition>
  </device>

  <device>
  <blockdev>/dev/nvme0n1</blockdev>
  <partition id="raid1" partitiontype="esp">
    <size>max</size>
    <type>linux raid</type>
  </partition>
</device>

  <device>
  <blockdev>/dev/nvme1n1</blockdev>
  <partition id="raid2" partitiontype="esp">
    <size>max</size>
    <type>linux raid</type>
  </partition>
</device>

<device>
  <blockdev>/dev/nvme4n1</blockdev>
  <partition id="raid3" partitiontype="esp">
    <size>max</size>
    <type>linux raid</type>
  </partition>
</device>

<device>
  <blockdev>/dev/nvme5n1</blockdev>
  <partition id="raid4" partitiontype="esp">
    <size>max</size>
    <type>linux raid</type>
  </partition>
</device>

<device>
  <blockdev>/dev/nvme6n1</blockdev>
  <partition id="raid5" partitiontype="esp">
    <size>max</size>
    <type>linux raid</type>
  </partition>
</device>

<device>
  <blockdev>/dev/nvme7n1</blockdev>
  <partition id="raid6" partitiontype="esp">
    <size>max</size>
    <type>linux raid</type>
  </partition>
</device>

<device>
  <blockdev>/dev/nvme8n1</blockdev>
```

```
  <partition id="raid7" partitiontype="esp">
    <size>max</size>
    <type>linux raid</type>
  </partition>
 </device>

 <device>
  <blockdev>/dev/nvme9n1</blockdev>
  <partition id="raid8" partitiontype="esp">
    <size>max</size>
    <type>linux raid</type>
  </partition>
 </device>

 <raid id="slash">
  <member>slash1</member>
  <member>slash2</member>
  <level>1</level>
  <filesystem>ext4</filesystem>
  <mountPoint>/</mountPoint>
  <mountOptions>defaults,noatime,nodiratime</mountOptions>
 </raid>

 <raid id="raid">
  <member>raid1</member>
  <member>raid2</member>
  <member>raid3</member>
  <member>raid4</member>
  <member>raid5</member>
  <member>raid6</member>
  <member>raid7</member>
  <member>raid8</member>
  <level>0</level>
  <filesystem>ext4</filesystem>
  <mountPoint>/raid</mountPoint>
  <mountOptions>defaults,noatime,nodiratime</mountOptions>
 </raid>
</diskSetup>
```

2. Assign this `disksetup` to the `dgx-a100` node category.

```
$ cmsh
% category
% use dgx
% set disksetup /cm/local/apps/cmd/etc/htdocs/disk-setup/dgxa100-disksetup.xml
% commit
```

2.1.3 Configure Node Network Interfaces

2.1.3.1 Configure Bright to Allow MAC Addresses to PXE Boot

1. Use the root (not `cmsh`) shell.
2. In `/cm/local/apps/cmd/etc/cmd.conf`, uncomment the `AdvancedConfig` parameter.

```
AdvancedConfig = { "DeviceResolveAnyMAC=1" } # modified value
```

3. Restart the `CMDaemon` to enable reliable PXE booting from bonded interfaces.

```
# systemctl restart cmd
```

The `cmsh` session will be disconnected because of restarting the `CMDaemon`. Type `connect` to reconnect after the `CMDaemon` has restarted. Or enter `exit` and then restart `cmsh`.

2.1.3.2 Configure Provisioning Interfaces on the DGX Nodes

The steps that follow are performed on the head node and should be run on all DGX systems.

Note: Double check the MAC address for each interface, and the IP address for the `bond0` interface. Mistakes here will be difficult to diagnose.

1. Add additional physical interfaces and set their MAC addresses.

```
# cmsh
% device
% use dgx01
% interfaces
% add physical enp225s0f1
% set mac B8:CE:F6:2F:08:69
% add physical enp97s0f1
% set mac B8:CE:F6:2D:0E:A7
% add physical enp225s0f1np1
% set mac B8:CE:F6:2F:08:69
% add physical enp97s0f1np1
% set mac B8:CE:F6:2D:0E:A7
% % commit

% list
Type         Network device name  IP               Network          Start if
------------ -------------------- ---------------- ---------------- --------
bmc          ipmi0                10.130.111.68    ipminet          always
physical     BOOTIF [prov]        10.130.122.5     internalnet      always
physical     enp225s0f1           0.0.0.0                           always
physical     enp97s0f1            0.0.0.0                           always
physical     enp225s0f1np1        0.0.0.0                           always
physical     enp97s0f1np1         0.0.0.0                           always
```

2. Add a `bond0` interface and assign it an IP.

```
% /       # go to top level of cmsh
% device
% use dgx01
% interfaces
% add bond bond0
% set interfaces enp225s0f1 enp97s0f1 enp225s0f1np1 enp97s0f1np1
% set network internalnet
% set ip 10.130.122.5
% set mode 4
% set options miimon=100

% show
Parameter                      Value
------------------------------ ------------------------------------------
Revision
```

```
Type                            bond
Network device name             bond0
Network                         internalnet
IP                              10.130.122.5
DHCP                            no
Alternative Hostname
Additional Hostnames
Start if                        always
BringUpDuringInstall            no
On network priority             70
Mode                            4
Options                         miimon=100
Interfaces                      enp225s0f1,enp97s0f1
```

3. Remove the `bootif` interface (bond0 will be used instead).

```
% remove bootif
% exit         # go up two levels
% exit
```

4. Set the `provisioninginterface` property to `bond0`.

```
% set provisioninginterface bond0
% device commit
```

5. Verify the configuration.

```
% get provisioninginterface
bond0
% interfaces
% list
Type         Network device name  IP               Network          Start if
------------ -------------------- ---------------- ---------------- --------
bmc          ipmi0                10.130.111.68    ipminet          always
bond         bond0 [prov]         10.130.122.5     internalnet      always
physical     enp225s0f1 (bond0)   0.0.0.0                           always
physical     enp97s0f1 (bond0)    0.0.0.0                           always
```

6. Perform all the steps in this section on each of the remaining DGX nodes.

2.1.3.3 Configure Provisioning Interfaces on the K8s Nodes

The steps that follow are performed on the head node and shown for a single K8s node (`knode01`). But all the steps in this section must be run for each of the three K8s nodes.

1. Add additional physical interfaces and set their MAC addresses.

```
% /       # got to top level of CMSH
% device
% use knode01
% interfaces
% add physical ens1f1np1
% set mac 04:3F:72:E7:64:97
% add physical ens2f1np1
% set mac 0C:42:A1:79:9B:15
% commit
% list
Type         Network device name  IP               Network          Start if
------------ -------------------- ---------------- ---------------- --------
bmc          ipmi0                10.130.111.72    ipminet          always
physical     BOOTIF [prov]        10.130.122.9     internalnet      always
physical     ens1f1np1            0.0.0.0                           always
physical     ens2f1np1            0.0.0.0                           always
```

2. Add a `bond0` interface and assign it an IP address.

```
% /
% device
% use knode01
% interfaces
% add bond bond0
% set interfaces ens1f1np1 ens2f1np1
% set network internalnet
% set ip 10.130.122.9
% set mode 4
% set options miimon=100

% show
Parameter                      Value
------------------------------ ----------------------------------------------
Revision
Type                           bond
Network device name            bond0
Network                        internalnet
IP                             10.130.122.9
DHCP                           no
Alternative Hostname
Additional Hostnames
Start if                       always
BringUpDuringInstall           no
On network priority            70
Mode                           4
Options                        miimon=100
Interfaces                     ens1f1np1,ens2f1np1
```

3. Remove the `bootif` interface and set the provisioning interface.

```
% remove bootif
% exit         # go up two levels
% exit
```

4. Set the `provisioninginterface` property to `bond0`.

```
% set provisioninginterface bond0
% device commit
```

5. Perform the steps in this section on the remaining K8s master nodes.

2.1.3.4 Configure InfiniBand Interfaces on DGX Nodes

The following procedure adds four physical InfiniBand interfaces for a single DGX system (`dgx01`). This procedure must be run for each DGX node.

```
% /       # go to top level of CMSH
% device
% use dgx01
% interfaces
% add physical ibp12s0
% set ip 10.149.0.5
% set network ibnet
% add physical ibp75s0
% set ip 10.149.1.5
% set network ibnet
% add physical ibp141s0
% set ip 10.149.2.5
```

```
% set network ibnet
% add physical ibp186s0
% set network ibnet
% set ip 10.149.3.5

% list
Type         Network device name  IP               Network          Start if
------------ -------------------- ---------------- ---------------- --------
bmc          ipmi0                10.130.111.69    ipminet          always
bond         bond0 [prov]         10.130.122.5     internalnet      always
physical     enp225s0f1 (bond0)   0.0.0.0                           always
physical     enp97s0f1 (bond0)    0.0.0.0                           always
physical     ibp12s0              10.149.0.5       ibnet            always
physical     ibp141s0             10.149.2.5       ibnet            always
physical     ibp186s0             10.149.3.5       ibnet            always
physical     ibp75s0              10.149.1.5       ibnet            always

% device commit
```

2.1.3.5 Identify the Cluster Nodes

1. Identify the nodes by setting the MAC address for the provisioning interface for each node to the MAC address listed in the site survey.

```
% device
% use dgx01
% set mac b8:ce:f6:2f:08:69
% use dgx02
% set mac 0c:42:a1:54:32:a7
% use dgx03
% set mac 0c:42:a1:0a:7a:51
% use dgx04
% set mac 1c:34:da:29:17:6e
% use knode01
% set mac 04:3F:72:E7:64:97
% use knode02
% set mac 04:3F:72:D3:FC:EB
% use knode03
% set mac 04:3F:72:D3:FC:DB
% foreach -c dgx,k8s-master (get mac)
B8:CE:F6:2F:08:69
0C:42:A1:54:32:A7
0C:42:A1:0A:7A:51
1C:34:DA:29:17:6E
04:3F:72:E7:64:97
04:3F:72:D3:FC:EB
04:3F:72:D3:FC:DB
```

2. If all the MAC addresses are set properly, commit the changes.

```
% device commit
% quit
```

2.1.3.6 Update the Software Images

These steps update the K8s master nodes and DGX systems to the latest software images.

1. Update K8s master node software image on the BCM head node.
```
# cm-chroot-sw-img /cm/images/k8s-master-image/
/# apt update && apt -y upgrade
/# exit
```
2. Power on `dgx01`.

   Wait until it is recognized by BCM before proceeding.
3. Update the DGX software.

   The DGX OS is updated by updating a running DGX node in DGX BasePOD, then using the resultant system as the source for the DGX image.
```
# ssh dgx01
# apt update && apt install cuda-compute-repo && apt full-upgrade -y
# exit
```
4. Set the DGX system to use the `dgx-a100-image` and set the kernel version to the latest one available.
```
% device
% use dgx01
% grabimage -w
% softwareimage
% use dgx-a100-image
% set kernelversion 5.4.0-131-generic
% commit
```

2.2 Power On and Provision Cluster Nodes

Now that the required post-installation configuration has been completed, it is time to power on and provision the cluster nodes. After the initial provisioning, power control will be available from within Bright—using the `cmsh` or Bright View. But for this initial provisioning it is necessary to power them on outside of Bright (that is, using the power button or a KVM).

It will take several minutes for the nodes to go through their BIOS. After that, the node status will progress as the nodes are being provisioned. Watch the `/var/log/messages` and `/var/log/node-installer` log files to verify that everything is proceeding smoothly.

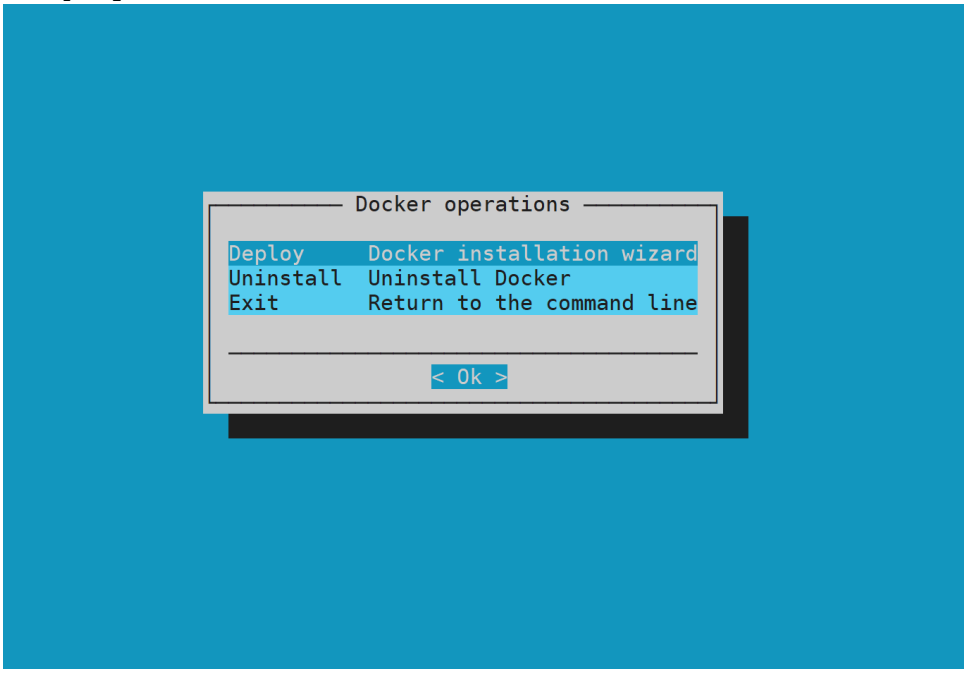
2.3 Deploy Docker

Install docker on the head node and K8s control plane nodes so that users can use docker functions on those nodes, for example, build containers.

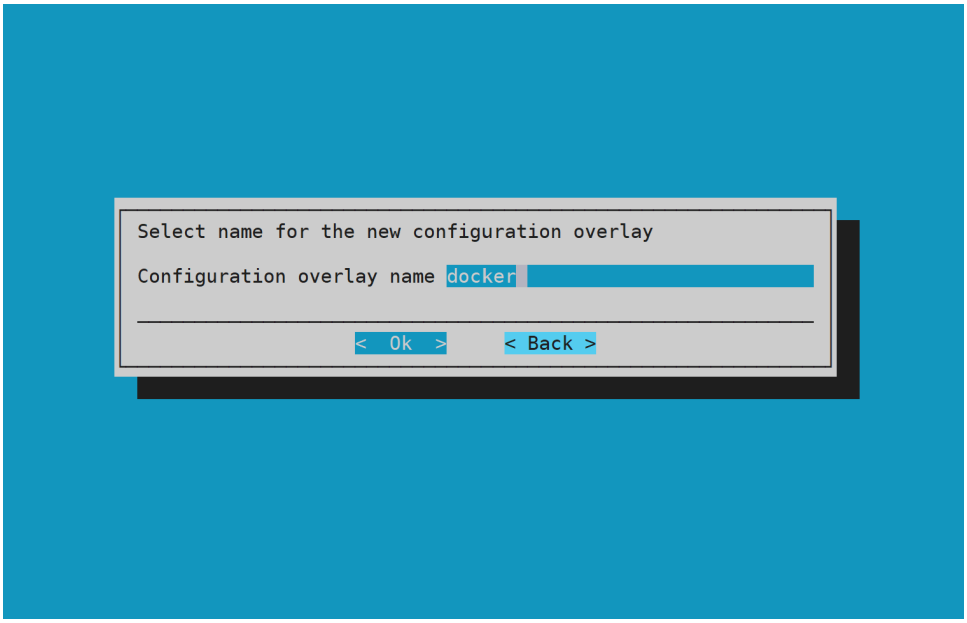
1. Run the `cm-docker-setup` CLI wizard on the head node as the root user.

```
# cm-docker-setup
```

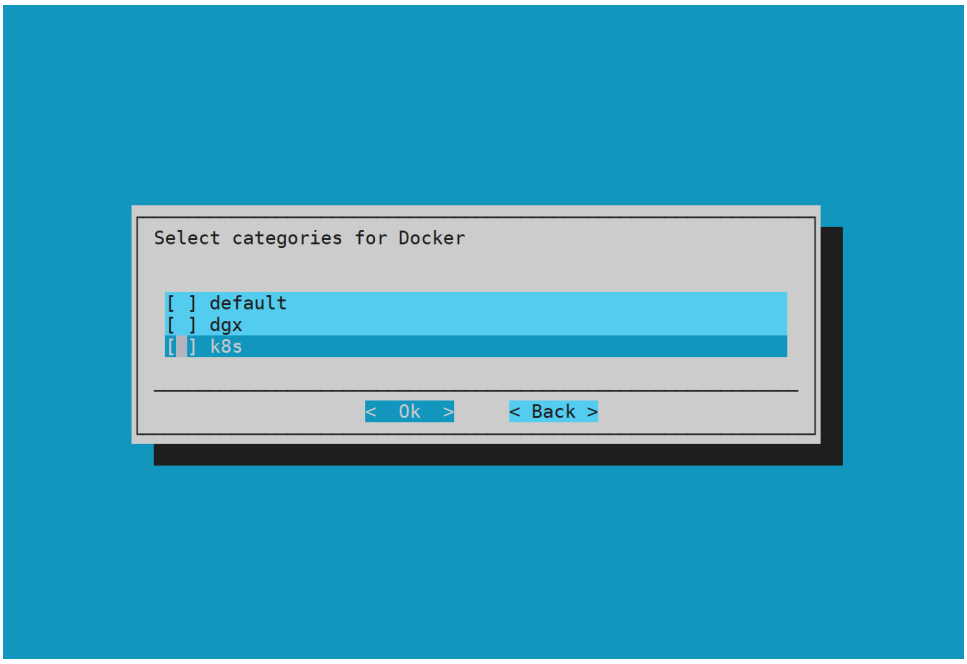
2. Choose `Deploy` to continue.



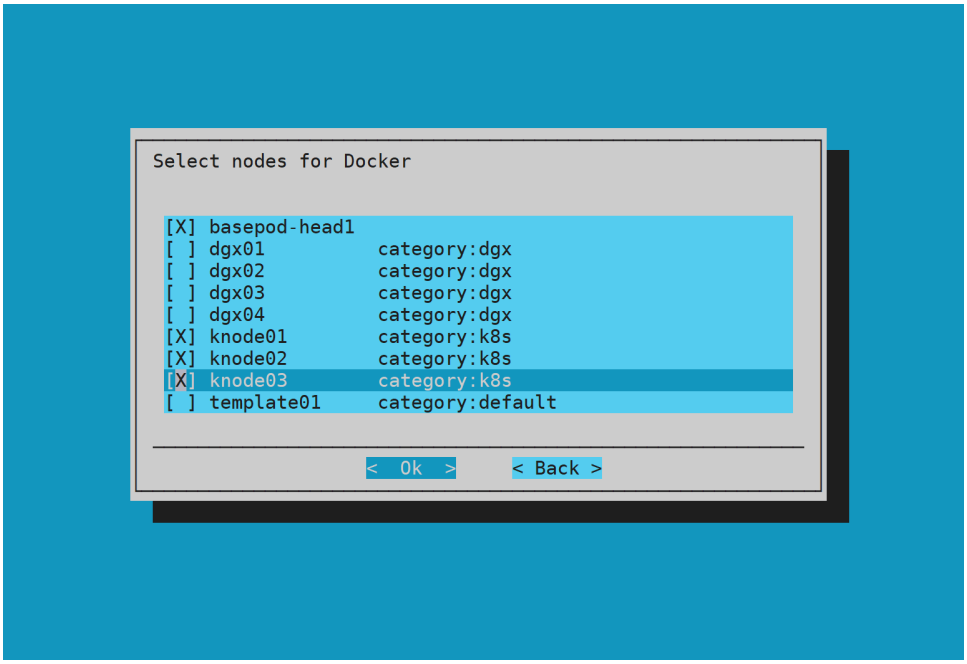
3. By default, the wizard will create a `docker` configuration overlay. This assigns the `Docker::Host` role to the nodes selected in the wizard.



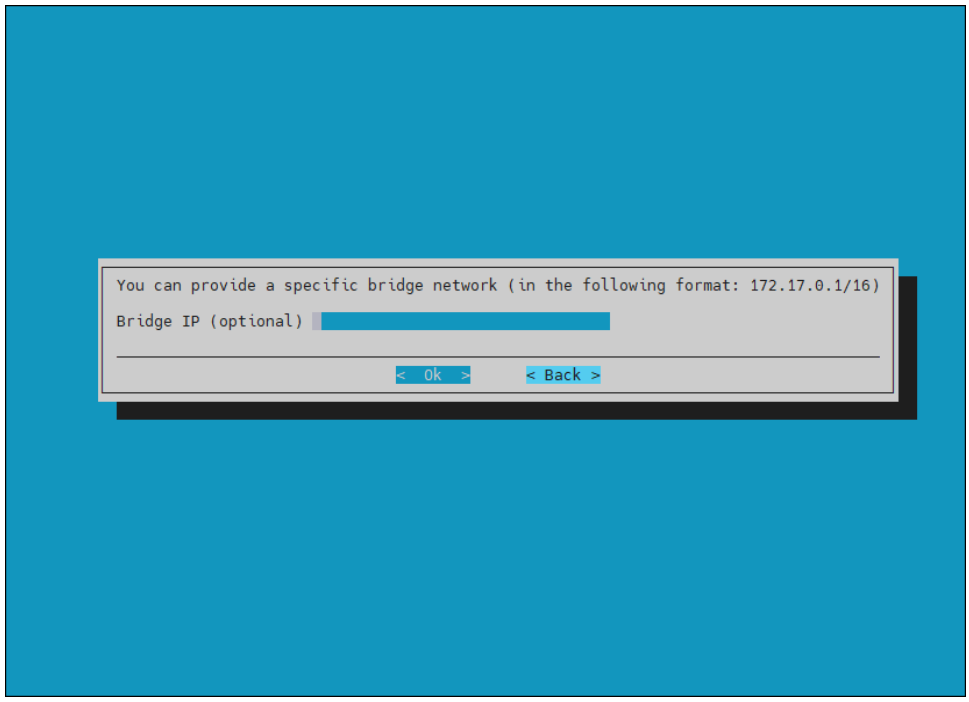
4. Leave items unselected in the screen because individual nodes will be specified in the next step.



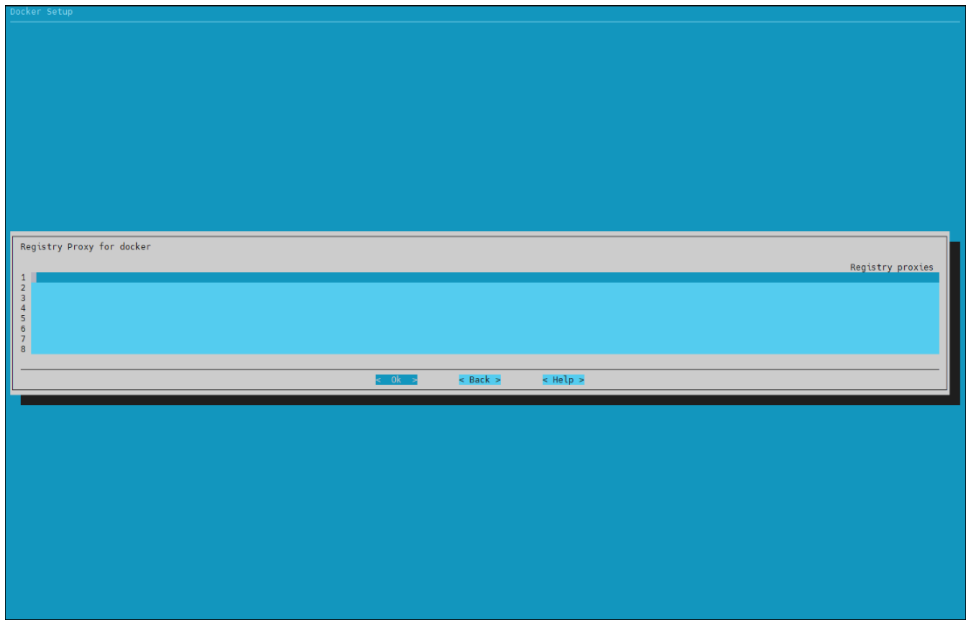
5. `basepod head1`, `knode01`, `knode02`, and `knode03` are selected to install docker.



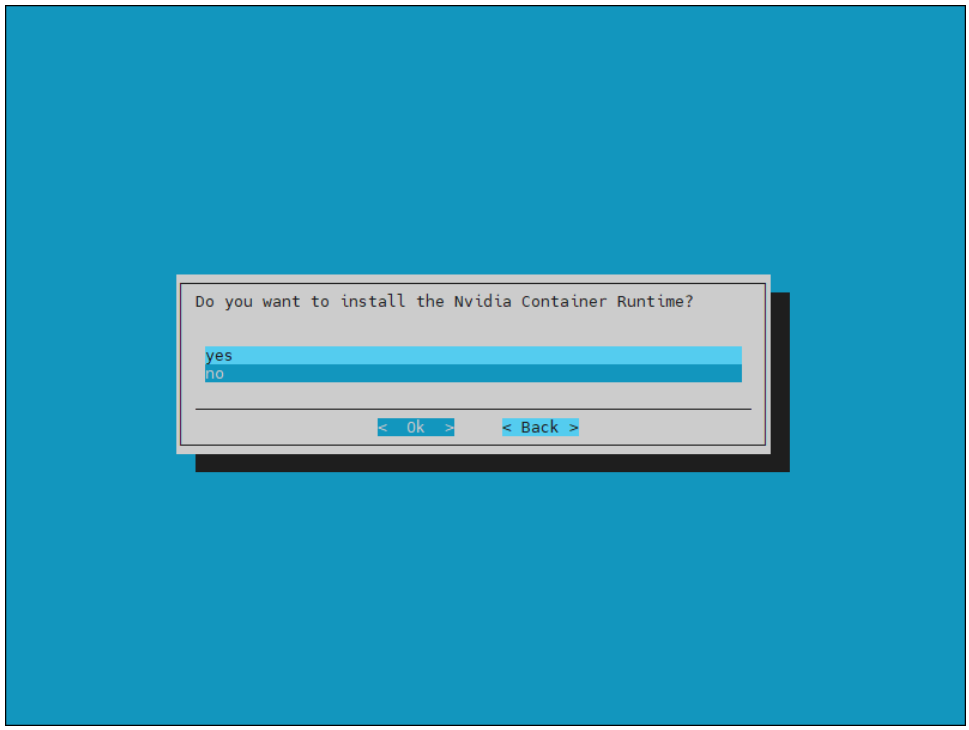
6. Optionally, specify a specific Docker bridge network. If you choose not to specify a bridge network, the default value of 172.17.0.0/16 will be used.



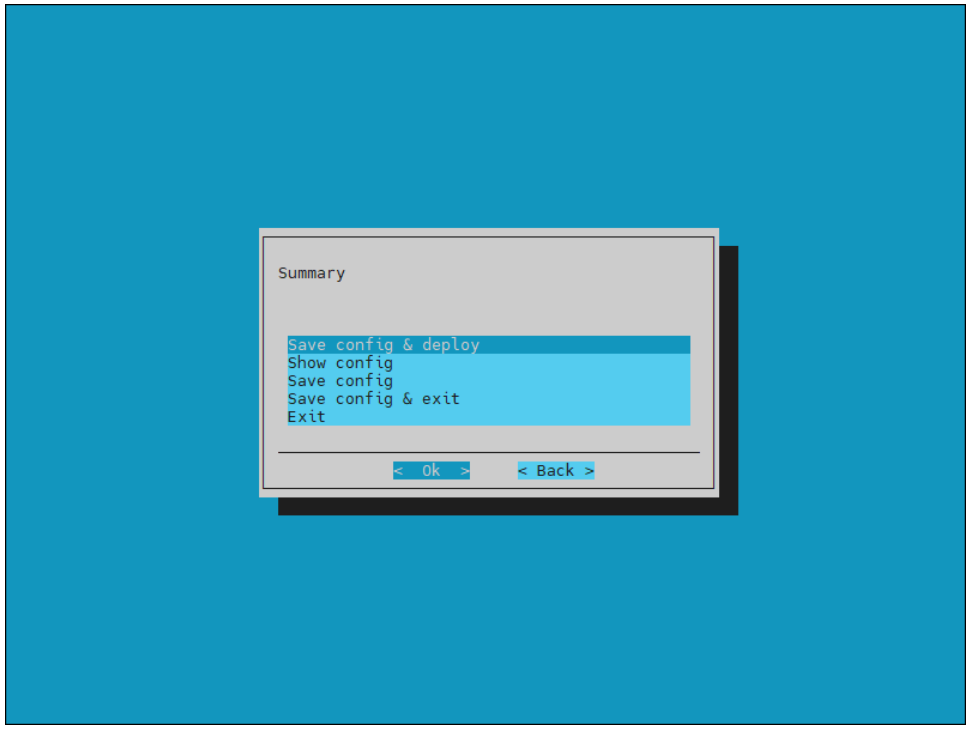
7. Enter any local Docker repositories this screen.



8. Do not install the NVIDIA Container Runtime on the head node and K8s control plane nodes since there are no GPUs on those nodes.

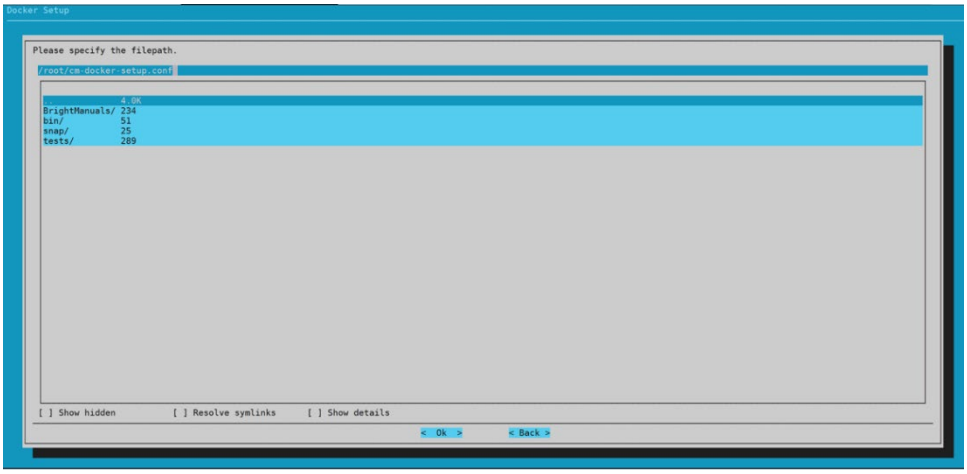


9. Select `Save config & deploy` to save the deployment configuration.



10. By default, the Docker wizard will save the deployment configuration in `/root/cm-docker-setup.conf`.

    This configuration file can be used to redeploy Docker in the future. Select `Ok` to start the installation.

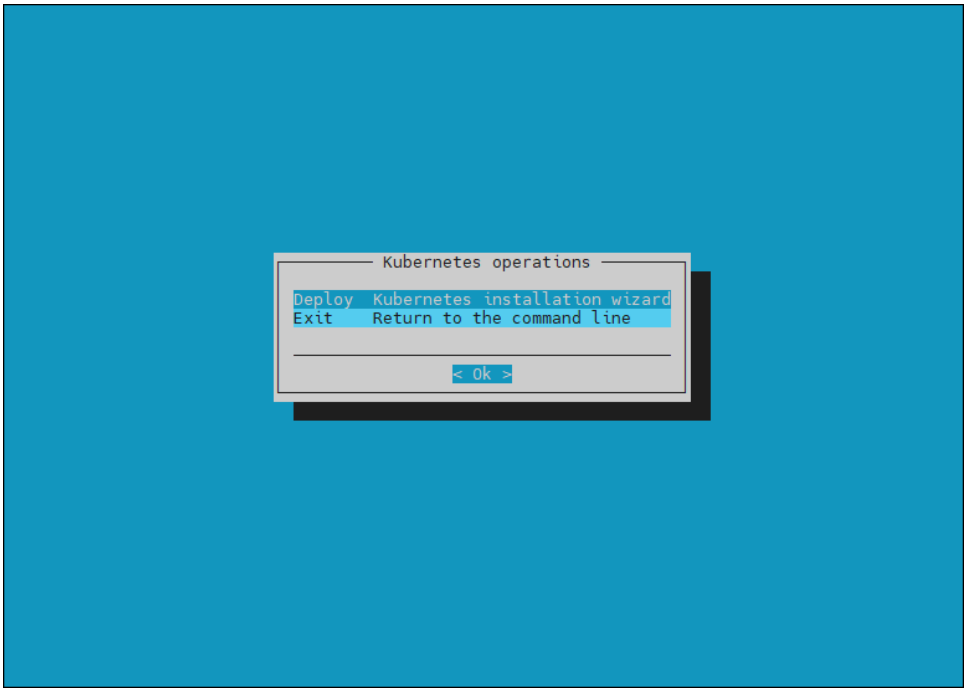


2.4 Deploy K8s

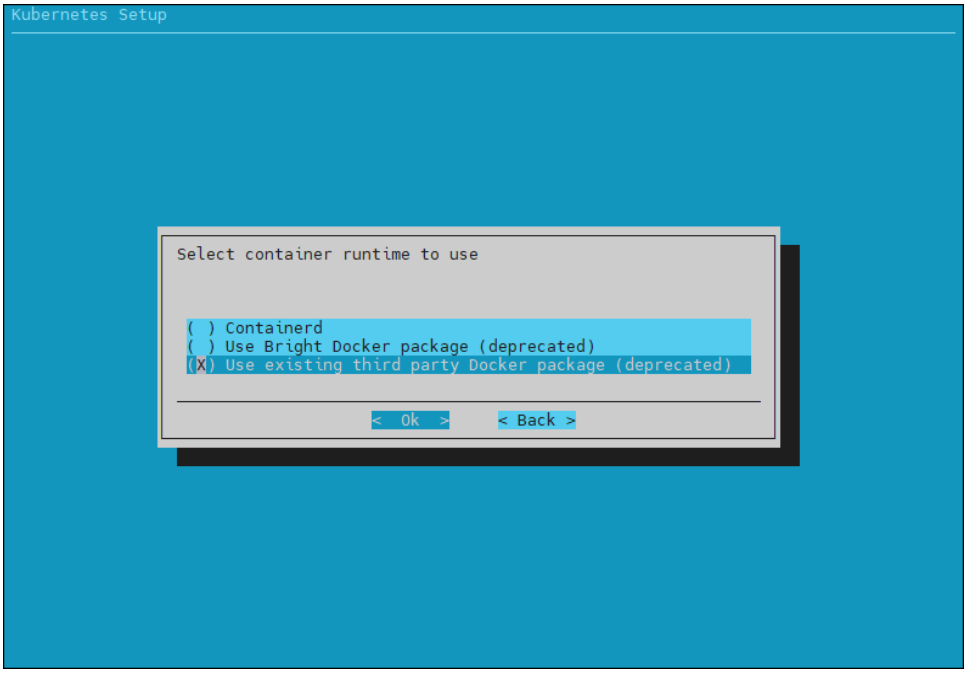
1. Run the `cm-kubernetes-setup` CLI wizard as the root user on the head node.

```
# cm-kubernetes-setup
```

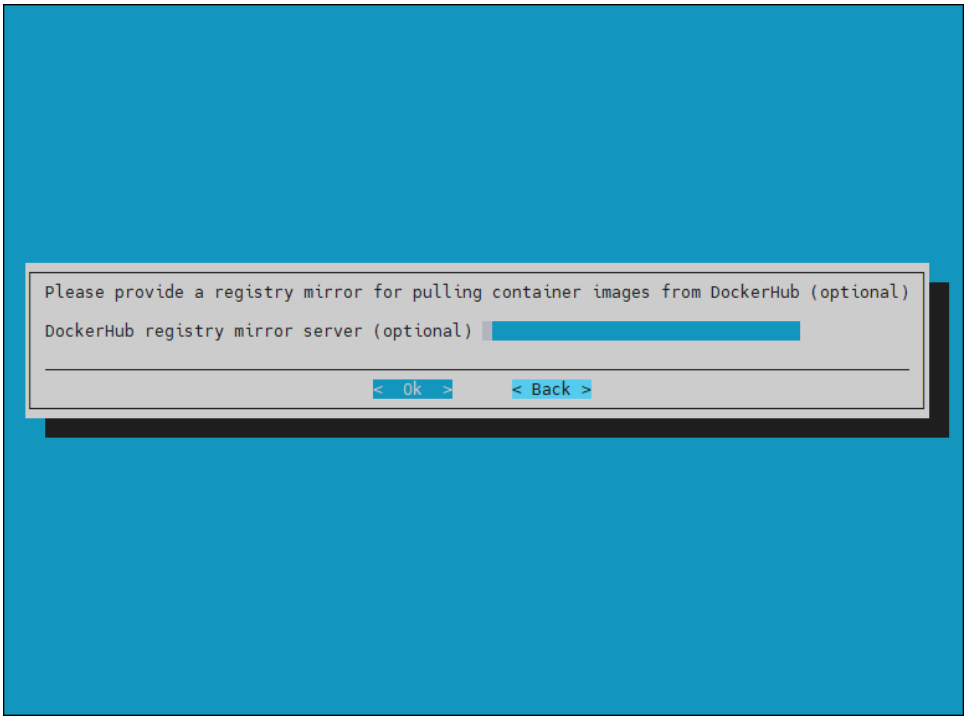
2. Choose `Deploy` to start the deployment.



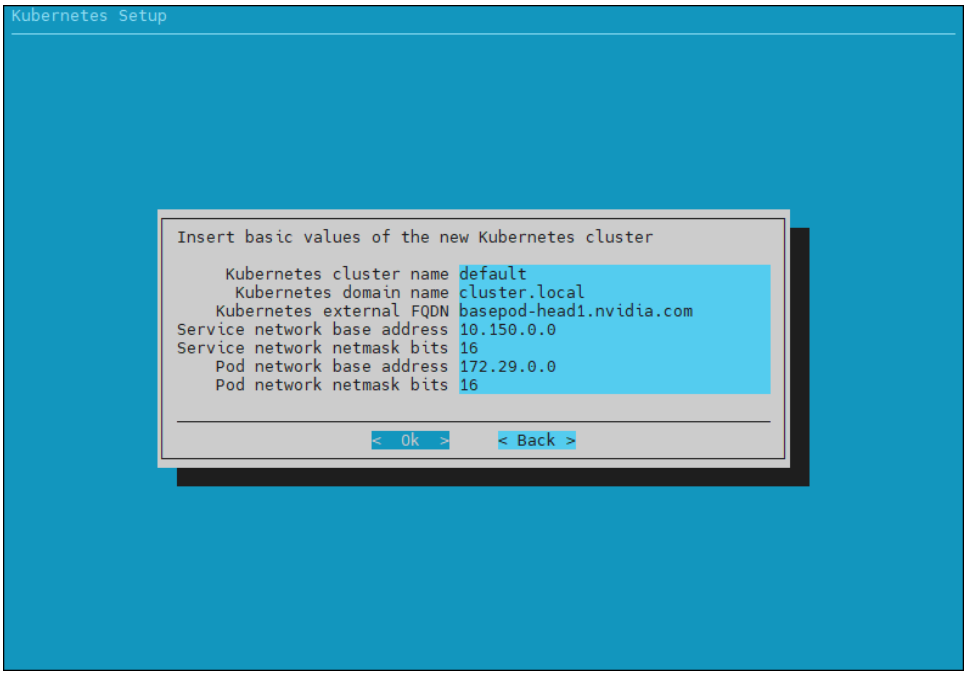
3. Select `Use existing third party Docker package`.
   The DGX A100 software image comes with Docker.



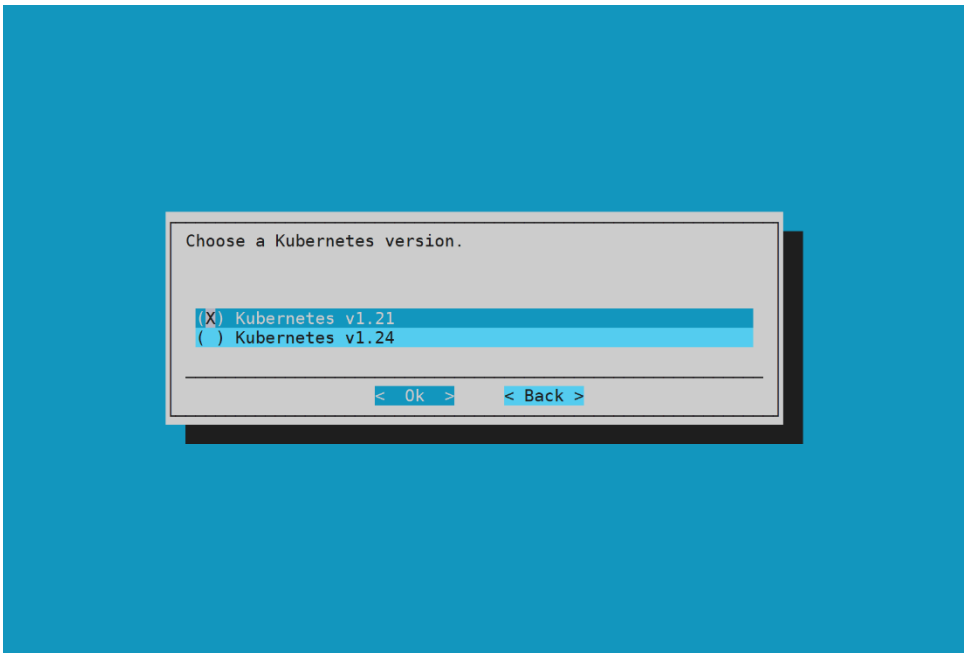
4. Select `Ok` to continue.



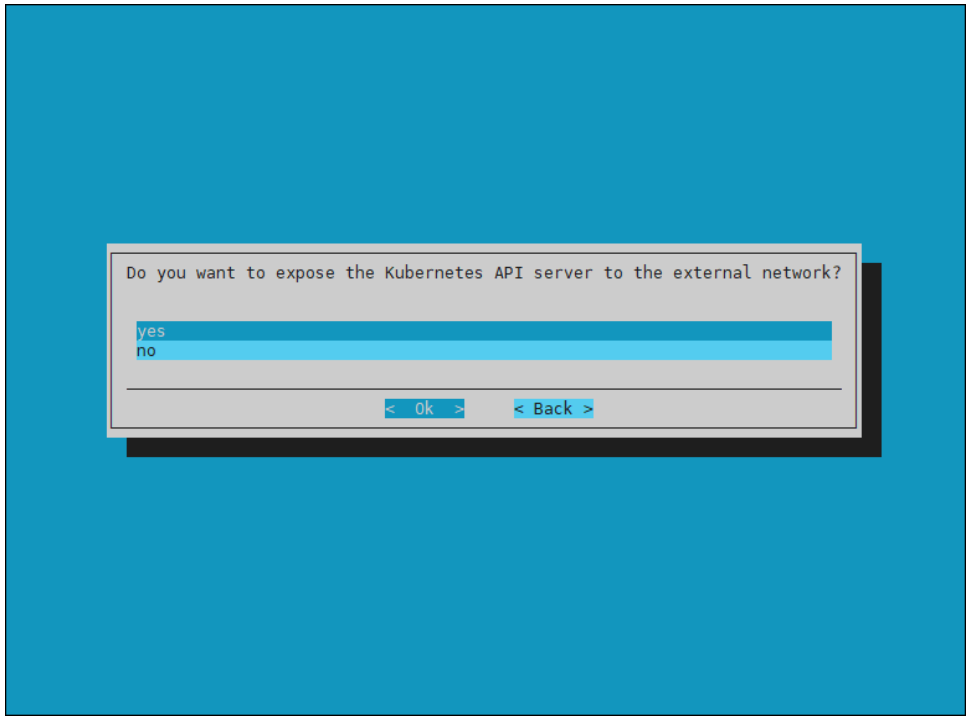
5. Accept the default settings for this K8s cluster.



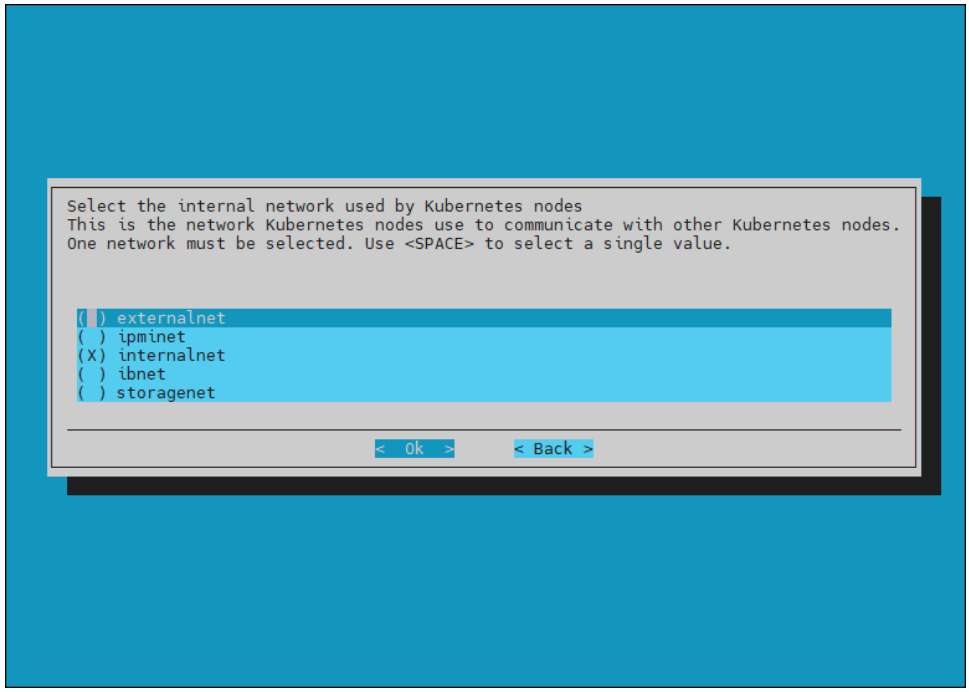
6. Select K8s version v.1.21 in the dialog that appears next.



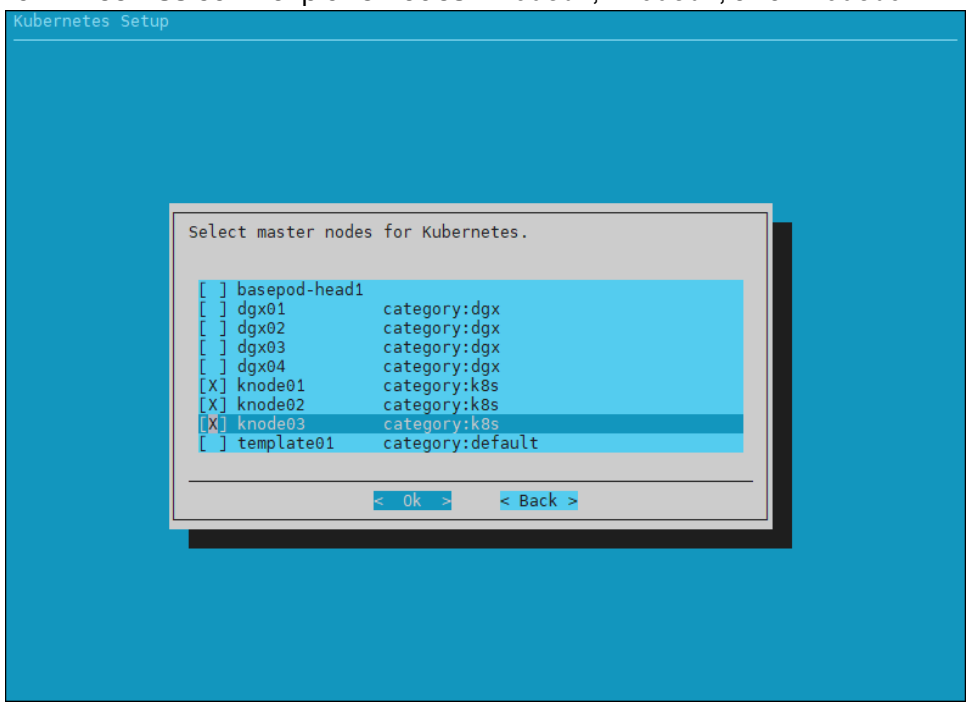
7. Select `yes` to expose the K8s API server on the head node.
   This allows users to use the K8s cluster from the head node.



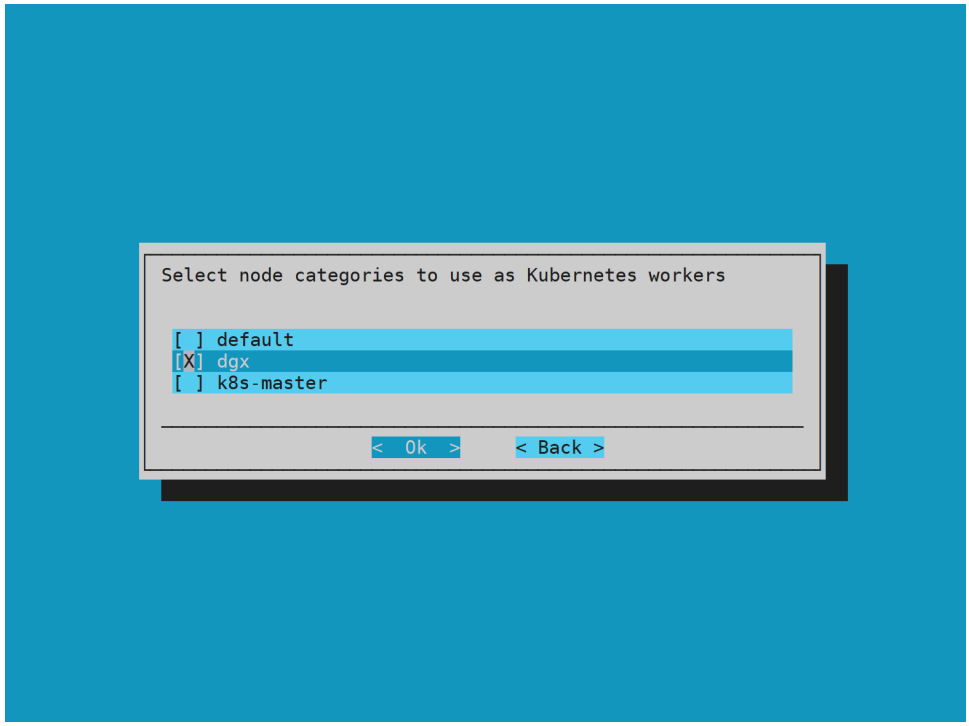
8. Select `internalnet` since the K8s master nodes and the DGX nodes, which are the K8s worker nodes, are all connected on `internalnet`.



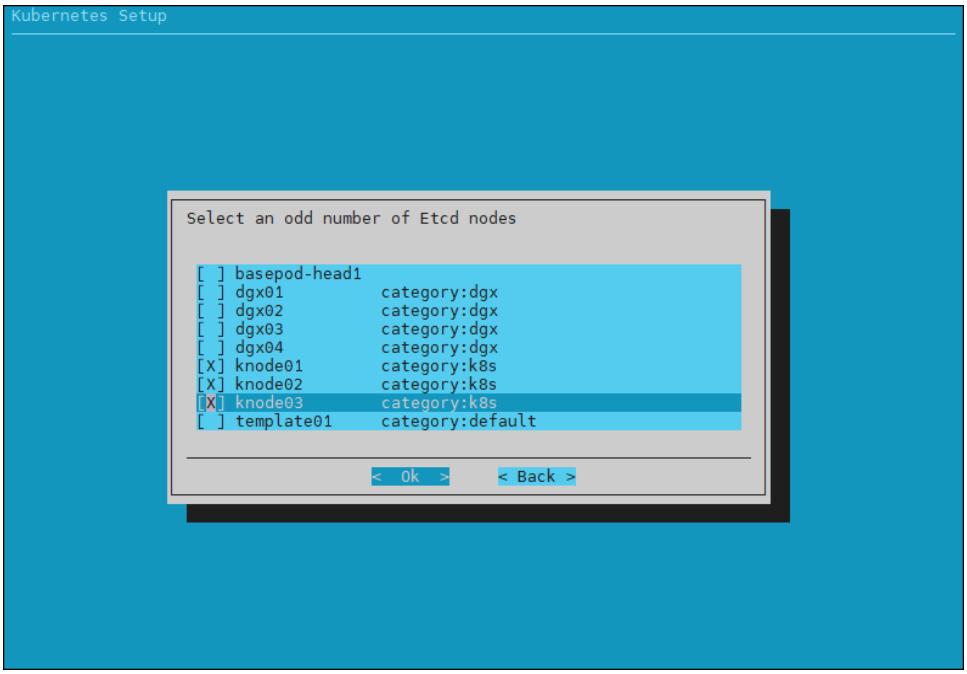
9. Select all three K8s control plane nodes: `knode01`, `knode02`, and `knode03`.



10. Select `dgx` for the node categories.

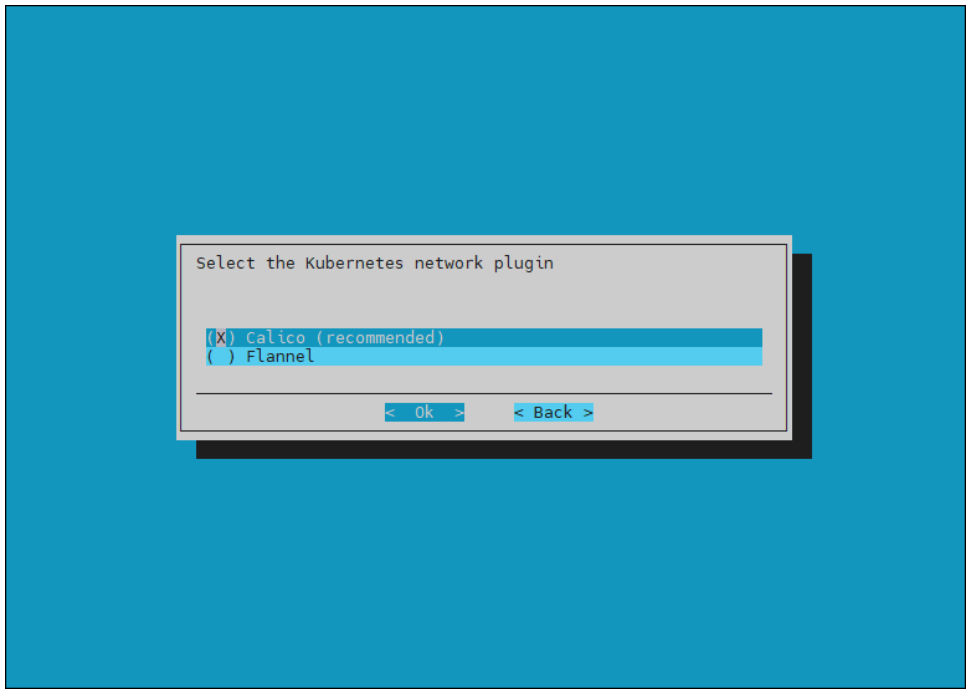


11. Select all three K8s control plane nodes: `knode01`, `knode02`, and `knode03` for the `Etcd` nodes.

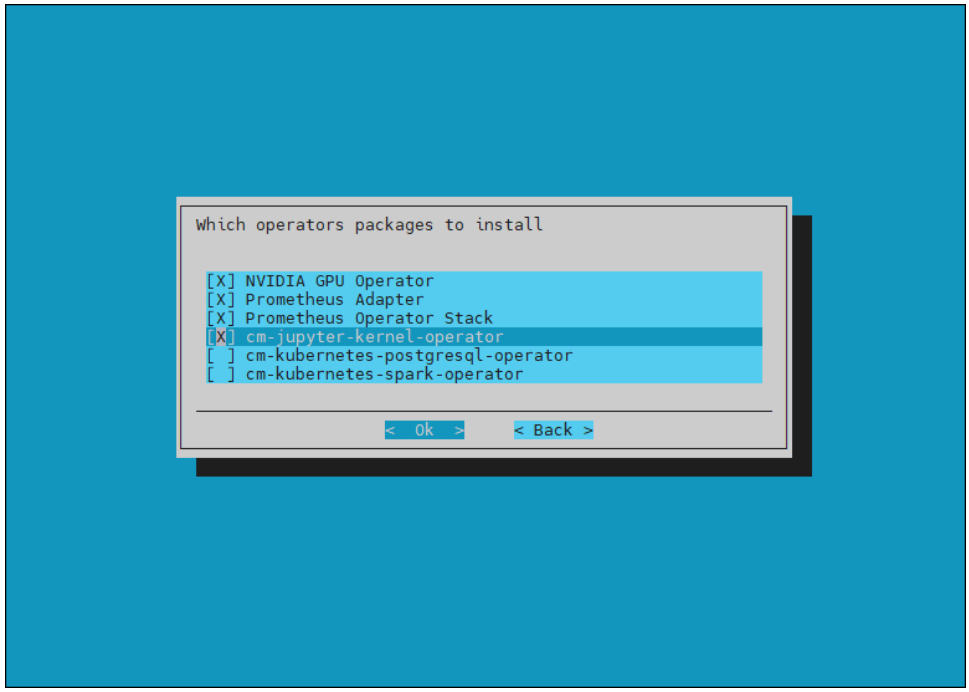


12. Accept the defaults unless the organization requires specific ports.

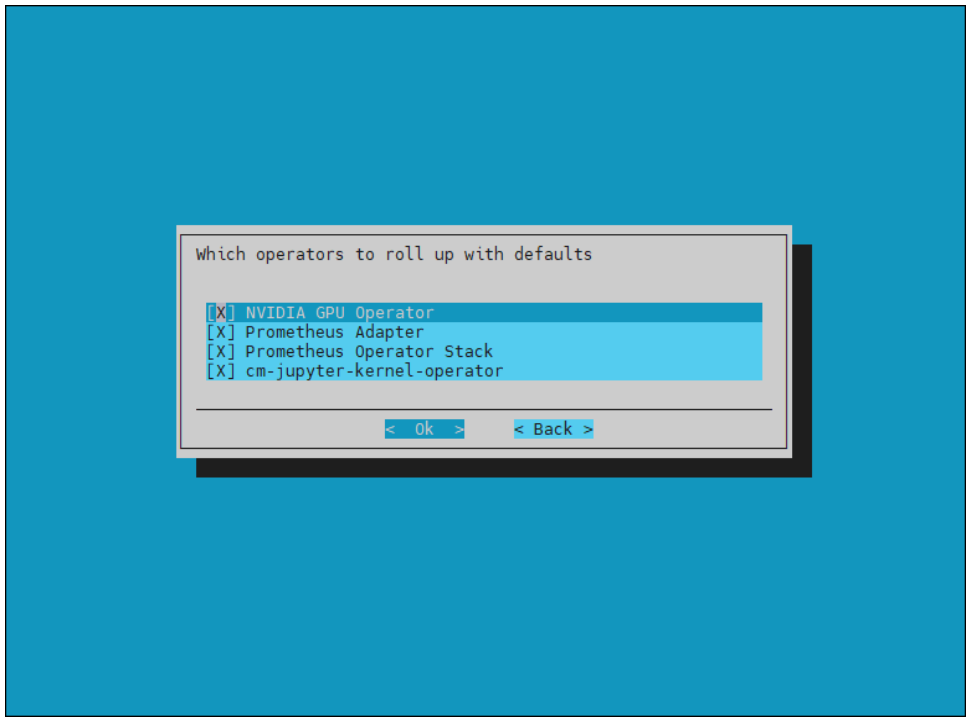
13. Select the `Calico network plugin` when prompted.



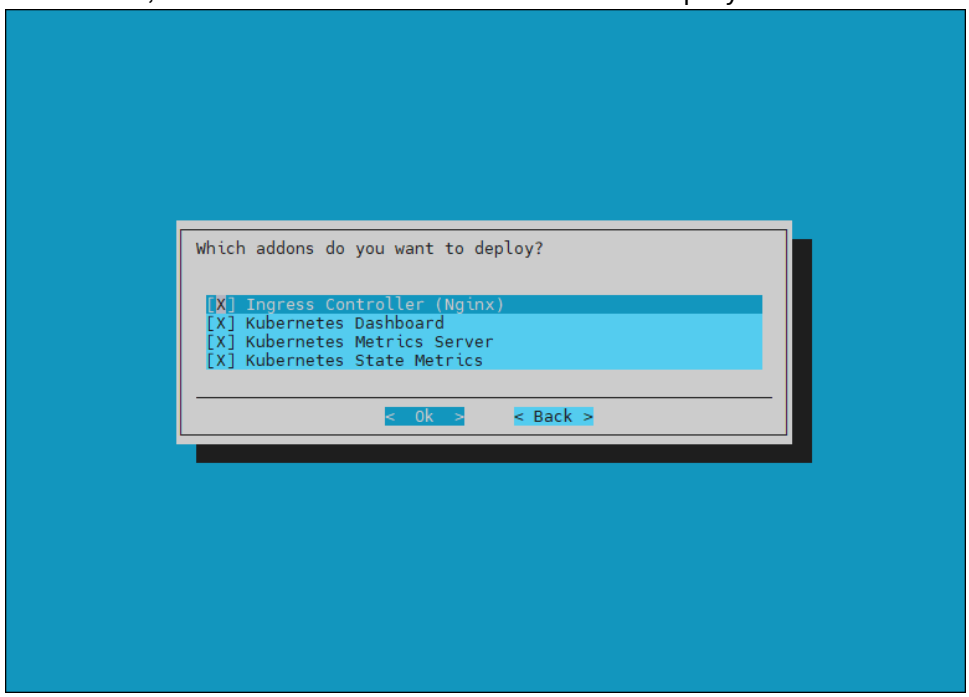
14. Select the following operators: `NVIDIA GPU Operator`, `Prometheus Adapter`, `Prometheus Adapter Stack`, and the `cm-jupyter-kernel-operator` to install.



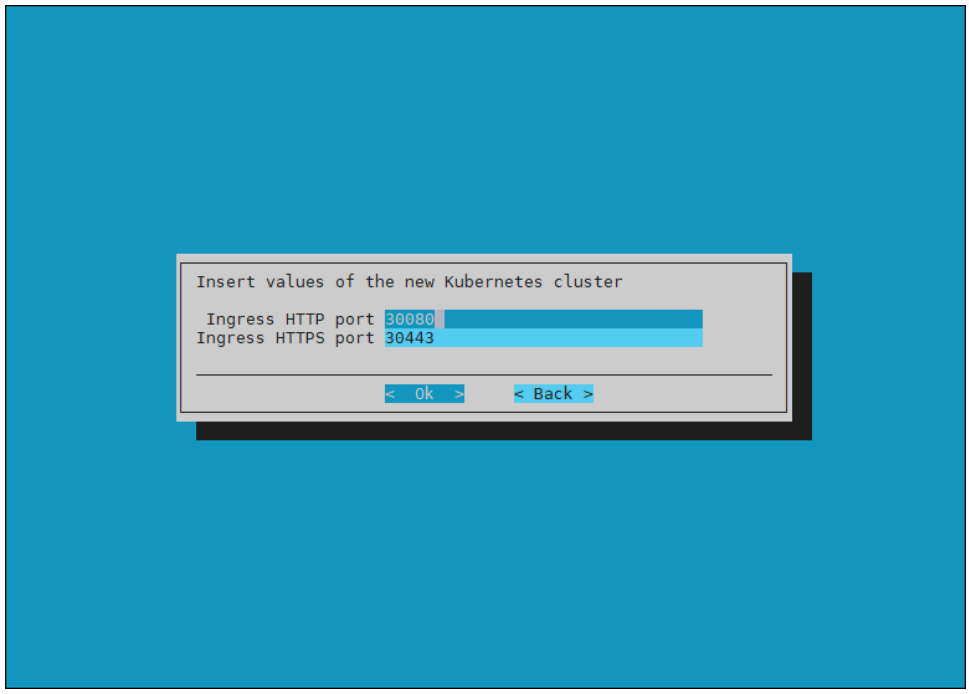
15. Select the same four operators to be rolled-up with the defaults.



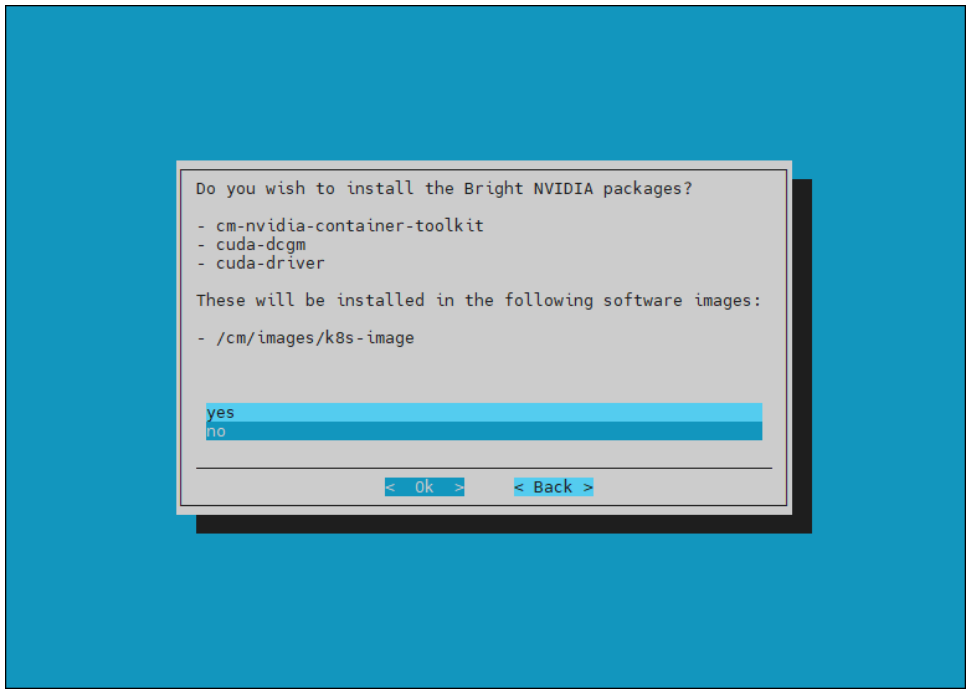
16. Select the `Ingress Controller (Nginx)`, `Kubernetes Dashboard`, `Kubernetes Metrics Server`, and `Kubernetes State Metrics` to deploy.



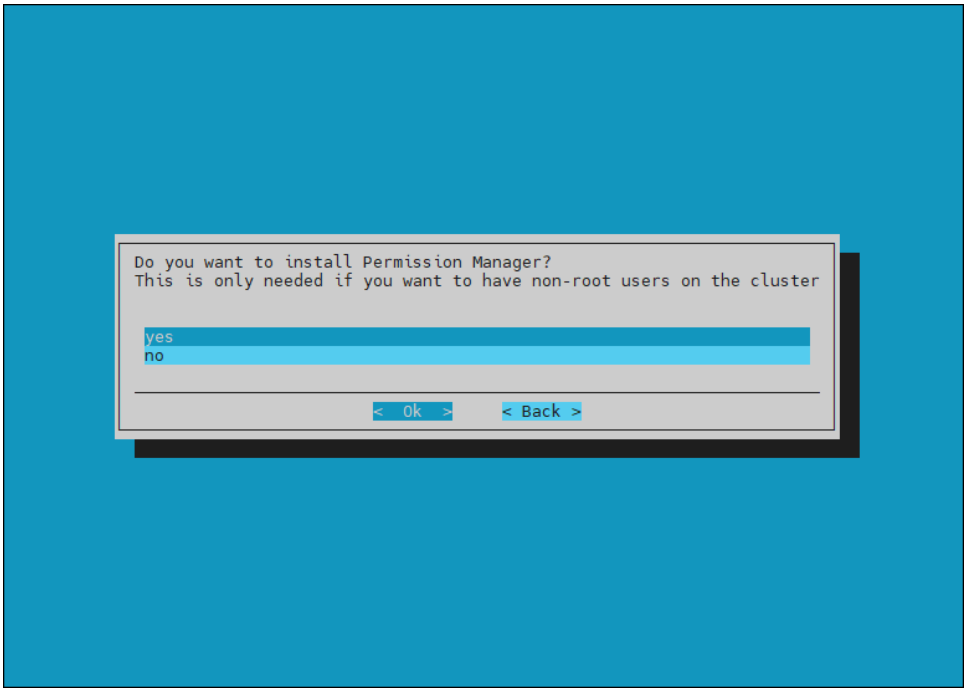
17. Select the defaults unless specific ingress ports are to be used.



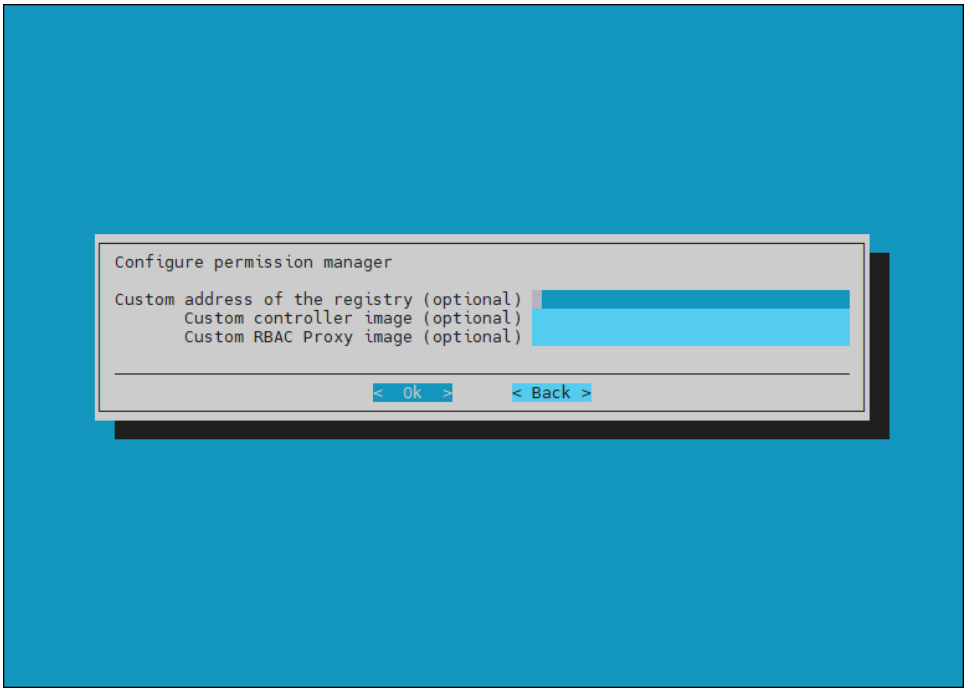
18. Select `no` since the K8s control plane nodes do not have GPUs.



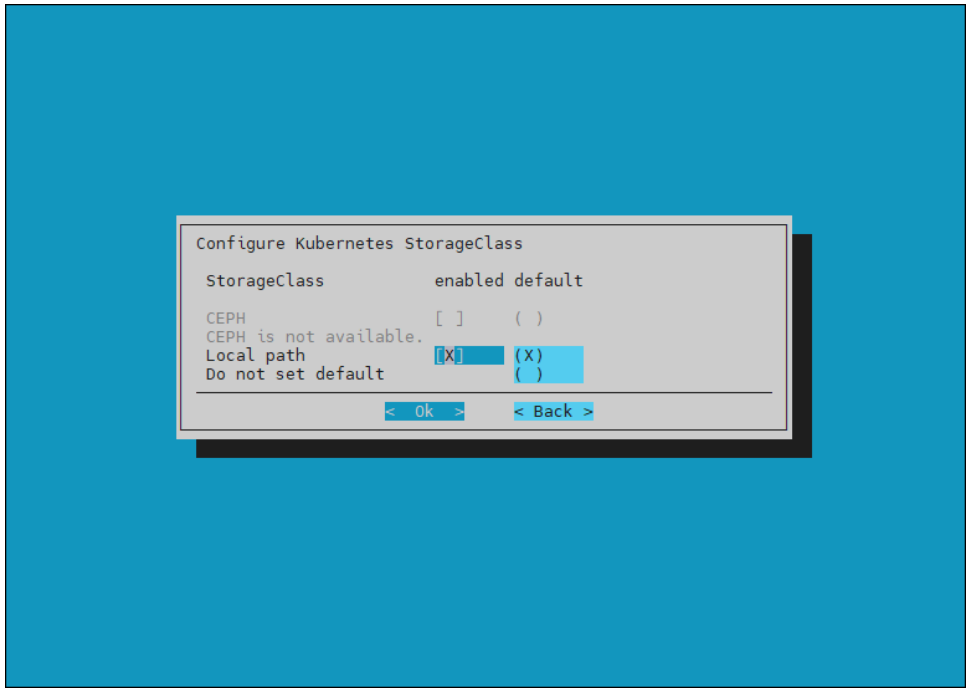
19. Select `yes` to deploy the `Permission Manager`.



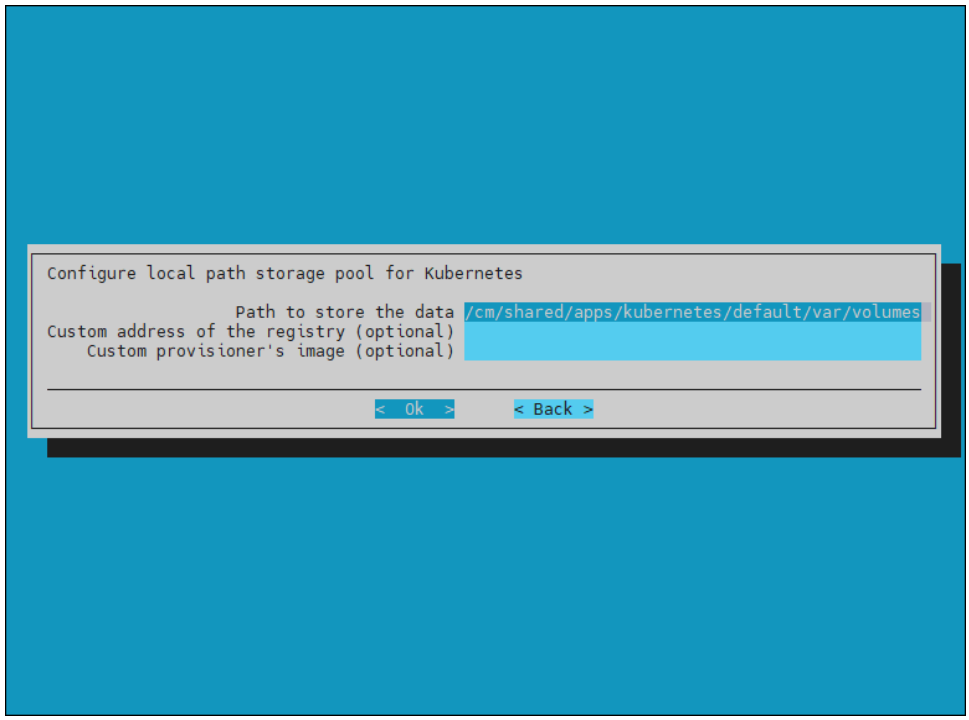
20. Leave these fields blank, which is the default.



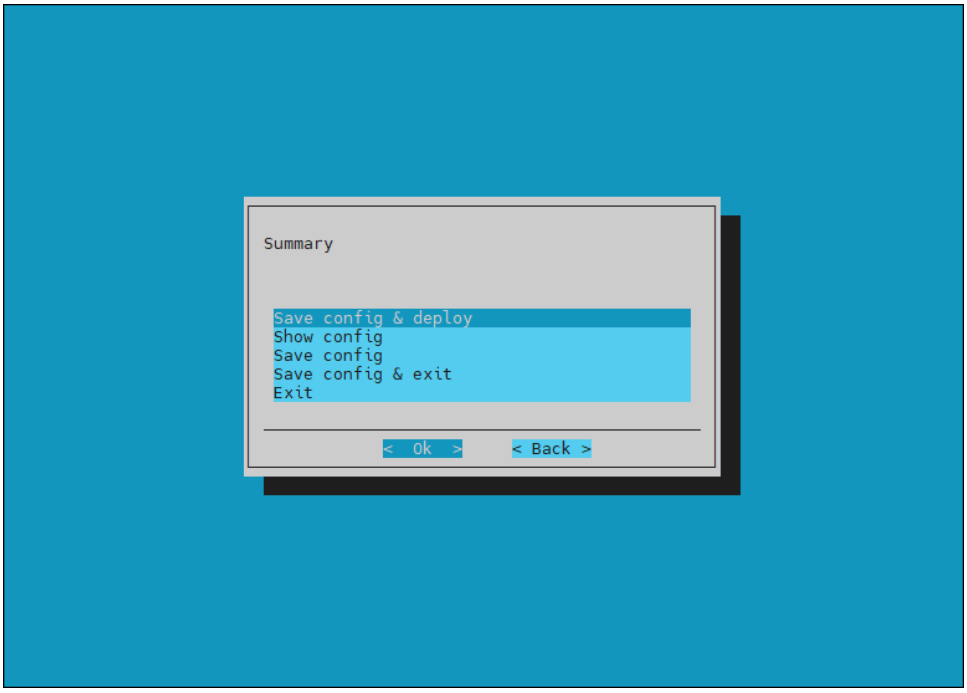
21. Select both `enabled` and `default` for the `Local path` storage class.



22. Accept the default data storage path and leave the other two fields blank, which is the default.

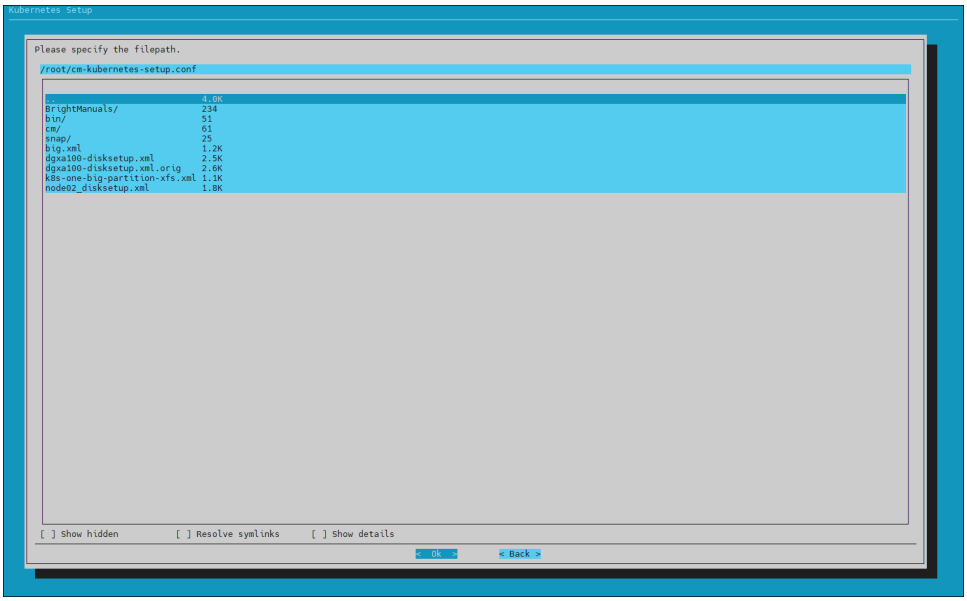


23. Select `Save config & deploy`.



24. Accept the default location for the `cm-kubernetes-setup.conf` file.

    This file can be used to deploy K8s or edited to deploy additional K8s clusters.



    Wait for the installation to finish.

25. Verify that the K8s cluster is installed properly.

```
# module load kubernetes/default/1.21.4
# kubectl cluster-info
Kubernetes control plane is running at https://localhost:10443
CoreDNS is running at https://localhost:10443/api/v1/namespaces/kube-
system/services/kube-dns:dns/proxy

To further debug and diagnose cluster problems, use 'kubectl cluster-info dump'.

# kubectl get nodes
NAME      STATUS   ROLES                  AGE     VERSION
knode01   Ready    control-plane,master   5m11s   v1.21.4
knode02   Ready    control-plane,master   5m11s   v1.21.4
knode03   Ready    control-plane,master   5m11s   v1.21.4
dgx01     Ready    worker                 5m5s    v1.21.4
dgx02     Ready    worker                 5m5s    v1.21.4
dgx03     Ready    worker                 4m54s   v1.21.4
dgx04     Ready    worker                 4m58s   v1.21.4
```

2.4.1 Install the Network Operator

2.4.1.1 Preparation

1. Verify that the NVIDIA Mellanox OFED package version on the DGX systems matches the version listed in DGX OS release notes: https://docs.nvidia.com/dgx/dgx-os-release-notes/index.html

```
# cmsh
% device
% pexec -c dgx -j "ofed_info -s"
[dgx01..dgx04]
MLNX_OFED_LINUX-5.4-3.4.0.0:
```

The correct InfiniBand interfaces used in the compute fabric must be identified and their operation status checked. As noted in Table 2, `mlx5_0`, `mlx5_1`, `mlx5_4`, and `mlx5_5` are used and should be verified in working condition. Each interface on each node should be `State:Active`, `Physical stat: LinkUp`, and `Link layer: InfiniBand`.

2. Verify that the interfaces are working properly with the following command:

```
[basepod-head1->device]% pexec -c dgx -j "for i in 0 1 4 5; do ibstat -d \
mlx5_${i} | grep -i \"mlx5_\\|state\\|infiniband\"; done"
[dgx01..dgx04]
CA 'mlx5_0'
                State: Active
                Physical state: LinkUp
                Link layer: InfiniBand
CA 'mlx5_1'
                State: Active
                Physical state: LinkUp
                Link layer: InfiniBand
CA 'mlx5_4'
                State: Active
                Physical state: LinkUp
                Link layer: InfiniBand
CA 'mlx5_5'
                State: Active
                Physical state: LinkUp
                Link layer: InfiniBand
```

3. Check the SRIOV interface status.

   The output should include these values:

   - `NUM_OF_VFS` should be set to `8`.
   - `SRIOV_EN` should be `True(1)`.
   - `Link_TYPE_P1` should be `IB(1)`.

   In this example, only the `Link_TYPE_P1` is set correctly. The others need to be set in the next step.

```
[basepod-head1->device]% pexec -c dgx -j "for i in 0 1 4 5; do mst start; \
mlxconfig -d /dev/mst/mt4123_pciconf${i} q; done | grep -e \
\"SRIOV_EN\\|LINK_TYPE\\|NUM_OF_VFS\""
[dgx01..dgx04]
         NUM_OF_VFS                          0
         SRIOV_EN                            False(0)
         LINK_TYPE_P1                        IB(1)
         NUM_OF_VFS                          0
         SRIOV_EN                            False(0)
         LINK_TYPE_P1                        IB(1)
         NUM_OF_VFS                          0
         SRIOV_EN                            False(0)
         LINK_TYPE_P1                        IB(1)
         NUM_OF_VFS                          0
         SRIOV_EN                            False(0)
         LINK_TYPE_P1                        IB(1)
```

4. Enable SRIOV and set `NUM_OF_VFS` to `8` for each interface.

   Since `Link_TYPE_P1` was set correctly, only the two other values are set below.

```
[basepod-head1->device]% pexec -c dgx -j "for i in 0 1 4 5; do mst start; \
mlxconfig -d /dev/mst/mt4123_pciconf${i} -y set SRIOV_EN=1 NUM_OF_VFS=8; done"
[dgx01..dgx04]
Starting MST (Mellanox Software Tools) driver set
Loading MST PCI module - Success
[warn] mst_pciconf is already loaded, skipping
Create devices
Unloading MST PCI module (unused) - Success

Device #1:
----------

Device type:    ConnectX6
Name:           MCX653105A-HDA_Ax
Description:    ConnectX-6 VPI adapter card; HDR IB (200Gb/s) and 200GbE; single-
port QSFP56; PCIe4.0 x16; tall bracket; ROHS R6
Device:         /dev/mst/mt4123_pciconf0

Configurations:                                 Next Boot       New
         SRIOV_EN                               False(0)        True(1)
         NUM_OF_VFS                             0               8

 Apply new Configuration? (y/n) [n] : y
Applying... Done!
-I- Please reboot machine to load new configurations.
. . . some output omitted . . .
```

5. Reboot the DGX nodes to load the configuration.

```
% reboot -c dgx
```

6. Wait for the DGX nodes to be `UP` before continuing to the next step.

```
% list -c dgx -f hostname:20,category:10,ip:20,status:10
hostname (key)       category   ip                   status
-------------------- ---------- -------------------- ----------
dgx01                dgx        10.130.122.5         [   UP   ]
dgx02                dgx        10.130.122.6         [   UP   ]
dgx03                dgx        10.130.122.7         [   UP   ]
dgx04                dgx        10.130.122.8         [   UP   ]
```

7. Configure eight SRIOV VFs on the InfiniBand ports.

```
[basepod-head1->device]% pexec -c dgx -j "for i in 0 1 4 5; do echo 8 > \
/sys/class/infiniband/mlx5_${i}/device/sriov_numvfs; done"
```

2.4.1.2 Deploy the Network Operator

Perform these steps on the primary head node.

1. Load the Kubernetes environment module.

```
# module load kubernetes/default/1.21.4
```

2. Add and install the Network Operator Helm repo.

```
# helm repo add nvidia-networking https://mellanox.github.io/network-operator
"nvidia-networking" has been added to your repositories

# helm repo update
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "nvidia-networking" chart repository
...Successfully got an update from the "prometheus-community" chart repository
...Successfully got an update from the "nvidia" chart repository
Update Complete. ⎈Happy Helming!⎈
```

3. Create `values.yaml` file for Helm to install Network Operator.

```
root@basepod-head1:~# cat ./network-operator/values.yaml
nfd:
  enabled: true
sriovNetworkOperator:
  enabled: true

# NicClusterPolicy CR values:
deployCR: true
ofedDriver:
  deploy: false
rdmaSharedDevicePlugin:
  deploy: false
sriovDevicePlugin:
  deploy: false

secondaryNetwork:
  deploy: true
  multus:
    deploy: true
  cniPlugins:
    deploy: true
  ipamPlugin:
    deploy: true
root@basepod-head1:~#
```

4. Use Helm to install Network Operator.

```
# helm install -f ./network-operator/values.yaml -n \
network-operator --create-namespace --wait nvidia-networking/network-operator --generate-
name
NAME: network-operator-1665598416
LAST DEPLOYED: Wed Oct 12 11:13:42 2022
NAMESPACE: network-operator
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Get Network Operator deployed resources by running the following commands:
# kubectl -n network-operator get pods
NAME                                                READY   STATUS    RESTARTS   AGE
cni-plugins-ds-9bv8j                                        1/1  Running  0  96s
cni-plugins-ds-f87tg                                        1/1  Running  0  96s
cni-plugins-ds-s76pw                                        1/1  Running  0  96s
```

```
cni-plugins-ds-w2lhl                                              1/1  Running  0  96s
kube-multus-ds-9gw52                                              1/1  Running  0  96s
kube-multus-ds-kbkvf                                              1/1  Running  0  96s
kube-multus-ds-s2n8h                                              1/1  Running  0  96s
kube-multus-ds-t8hfs                                              1/1  Running  0  96s
network-operator-1668351491-b58d45f44-qkn7v                       1/1  Running  0 102s
network-operator-1668351491-node-feature-discovery-master-5fgrv 1/1  Running  0 102s
network-operator-1668351491-node-feature-discovery-worker-2vnjp 1/1  Running  0 102s
network-operator-1668351491-node-feature-discovery-worker-77pnc 1/1  Running  0 102s
network-operator-1668351491-node-feature-discovery-worker-8pc2k 1/1  Running  0 102s
network-operator-1668351491-node-feature-discovery-worker-l7r4r 1/1  Running  0 102s
network-operator-1668351491-sriov-network-operator-5675d88rpqhn 1/1  Running  0 102s
sriov-network-config-daemon-l4ch9                                 3/3  Running  0  83s
sriov-network-config-daemon-l99rq                                 3/3  Running  0  83s
sriov-network-config-daemon-zvcpv                                 3/3  Running  0  83s
sriov-network-config-daemon-zvsxq                                 3/3  Running  0  83s
whereabouts-g5lkv                                                 1/1  Running  0  96s
whereabouts-hd4zt                                                 1/1  Running  0  96s
whereabouts-jjmcv                                                 1/1  Running  0  96s
whereabouts-kll9p                                                 1/1  Running  0  96s
root@basepod-head1:/nfs/general/network-operator#
```

5. Create the `sriov-ib-network-node-policy.yaml` and `sriovibnetwork.yaml` configuration files.

   The interface names must match the interface names in Table 2.

```
root@basepod-head1:~# cat ./network-operator/sriov-ib-network-node-policy.yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: ibp12s0
  namespace: network-operator
spec:
  deviceType: netdevice
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  nicSelector:
    vendor: "15b3"
    pfNames: ["ibp12s0"]
  linkType: ib
  isRdma: true
  numVfs: 8
  priority: 90
  resourceName: resibp12s0

---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: ibp75s0
  namespace: network-operator
spec:
  deviceType: netdevice
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  nicSelector:
    vendor: "15b3"
    pfNames: ["ibp75s0"]
  linkType: ib
  isRdma: true
```

```
  numVfs: 8
  priority: 90
  resourceName: resibp75s0

---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: ibp141s0
  namespace: network-operator
spec:
  deviceType: netdevice
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  nicSelector:
    vendor: "15b3"
    pfNames: ["ibp141s0"]
  linkType: ib
  isRdma: true
  numVfs: 8
  priority: 90
  resourceName: resibp141s0

---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovNetworkNodePolicy
metadata:
  name: ibp186s0
  namespace: network-operator
spec:
  deviceType: netdevice
  nodeSelector:
    feature.node.kubernetes.io/network-sriov.capable: "true"
  nicSelector:
    vendor: "15b3"
    pfNames: ["ibp186s0"]
  linkType: ib
  isRdma: true
  numVfs: 8
  priority: 90
  resourceName: resibp186s0

root@basepod-head1:~# cat ./network-operator/sriovibnetwork.yaml
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: ibp12s0
  namespace: network-operator
spec:
  ipam: |
    {
      "type": "whereabouts",
      "datastore": "kubernetes",
      "kubernetes": {
        "kubeconfig": "/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig"
      },
      "range": "192.168.1.0/24",
```

```
      "log_file": "/var/log/whereabouts.log",
      "log_level": "info"
    }
  resourceName: resibp12s0
  linkState: enable
  networkNamespace: default

---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: ibp75s0
  namespace: network-operator
spec:
  ipam: |
    {
      "type": "whereabouts",
      "datastore": "kubernetes",
      "kubernetes": {
        "kubeconfig": "/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig"
      },
      "range": "192.168.2.0/24",
      "log_file": "/var/log/whereabouts.log",
      "log_level": "info"
    }
  resourceName: resibp75s0
  linkState: enable
  networkNamespace: default

---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: ibpi141s0
  namespace: network-operator
spec:
  ipam: |
    {
      "type": "whereabouts",
      "datastore": "kubernetes",
      "kubernetes": {
        "kubeconfig": "/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig"
      },
      "range": "192.168.3.0/24",
      "log_file": "/var/log/whereabouts.log",
      "log_level": "info"
    }
  resourceName: resibp141s0
  linkState: enable
  networkNamespace: default

---
apiVersion: sriovnetwork.openshift.io/v1
kind: SriovIBNetwork
metadata:
  name: ibp186s0
  namespace: network-operator
```

```
spec:
  ipam: |
    {
      "type": "whereabouts",
      "datastore": "kubernetes",
      "kubernetes": {
        "kubeconfig": "/etc/cni/net.d/whereabouts.d/whereabouts.kubeconfig"
      },
      "range": "192.168.4.0/24",
      "log_file": "/var/log/whereabouts.log",
      "log_level": "info"
    }
  resourceName: resibp186s0
  linkState: enable
  networkNamespace: default
root@basepod-head1:~#
```

6. Deploy the configuration files.

```
# kubectl apply -f ./network-operator/sriov-ib-network-node-policy.yaml
sriovnetworknodepolicy.sriovnetwork.openshift.io/ibp12s0 created
sriovnetworknodepolicy.sriovnetwork.openshift.io/ibp75s0 created
sriovnetworknodepolicy.sriovnetwork.openshift.io/ibp141s0 created
sriovnetworknodepolicy.sriovnetwork.openshift.io/ibp186s0 created

# kubectl apply -f ./network-operator/sriovibnetwork.yaml
sriovibnetwork.sriovnetwork.openshift.io/ibp12s0 created
sriovibnetwork.sriovnetwork.openshift.io/ibp75s0 created
sriovibnetwork.sriovnetwork.openshift.io/ibpi141s0 created
sriovibnetwork.sriovnetwork.openshift.io/ibp186s0 created
```

7. Deploy the `mpi-operator`.

```
# kubectl apply -f \
https://raw.githubusercontent.com/kubeflow/mpi-\
operator/master/deploy/v2beta1/mpi-operator.yaml
namespace/mpi-operator created
customresourcedefinition.apiextensions.k8s.io/mpijobs.kubeflow.org
created
serviceaccount/mpi-operator created
clusterrole.rbac.authorization.k8s.io/kubeflow-mpijobs-admin created
clusterrole.rbac.authorization.k8s.io/kubeflow-mpijobs-edit created
clusterrole.rbac.authorization.k8s.io/kubeflow-mpijobs-view created
clusterrole.rbac.authorization.k8s.io/mpi-operator created
clusterrolebinding.rbac.authorization.k8s.io/mpi-operator created
deployment.apps/mpi-operator created
```

8. Copy the Network Operator `/opt/cni/bin` directory to `/cm/shared`, where it will be accessed by the head nodes.

```
# ssh dgx01
# cp -r /opt/cni/bin /cm/shared/dgx_opt_cni_bin
# exit
```

9. Make two backup directories.

```
root@basepod-head1:~ # mv /cm/images/dgx-a100-\
image/cm/local/apps/kubernetes/current/bin/cni /cm/images/dgx-a100-\
image/cm/local/apps/kubernetes/current/bin/cni_orig
root@basepod-head1:~# mv /cm/images/dgx-a100-image/opt/cni/bin \
/cm/images/dgx-a100-image/opt/cni/bin_orig
```

10. Copy the directory from `/cm/shared` directory to `/cm/image` where it is used by the BCM DGX image.

```
root@basepod-head1:~ # cp -r /cm/shared/dgx_opt_cni_bin \
/cm/images/dgx-a100-image/cm/local/apps/kubernetes/current/bin/cni
# cp -r /cm/shared/dgx_opt_cni_bin  /cm/images/dgx-a100-image/opt/cni/bin
```

11. Create the `network-validation.yaml` file and run a simple validation test.

```
root@basepod-head1:~ # cat network-operator/network-validation.yaml
apiVersion: v1
kind: Pod
metadata:
  name: network-validation-pod
  annotations:
    k8s.v1.cni.cncf.io/networks: ibp75s0,ibp186s0,ibp12s0,ibp141s0
spec:
  containers:
    - name: network-validation-pod
      image: docker.io/deepops/nccl-tests:latest
      imagePullPolicy: IfNotPresent
      command:
        - sh
        - -c
        - sleep inf
      securityContext:
        capabilities:
          add: [ "IPC_LOCK" ]
      resources:
        requests:
          nvidia.com/resibp75s0: "1"
          nvidia.com/resibp186s0: "1"
          nvidia.com/resibp12s0: "1"
          nvidia.com/resibp141s0: "1"
        limits:
          nvidia.com/resibp75s0: "1"
          nvidia.com/resibp186s0: "1"
          nvidia.com/resibp12s0: "1"
          nvidia.com/resibp141s0: "1"
root@basepod-head1:~ # kubectl apply -f \
./network-operator/network-validation.yaml
pod/network-validation-pod created
```

2.4.1.3 Run a Multi-node NCCL Test

The NVIDIA Collective Communication Library (NCCL) implements multi-GPU and multi-node communication primitives optimized for NVIDIA GPUs and networking that is the foundation for many AI/ML training and deep learning applications. A successful run of a multi-node NCCL test is a good indicator that multi-node MPI and NCCL communication between GPUs is operating correctly.

1. Build the `nccl_test` K8s job file.

```
# cat ./nccl_test.yaml
apiVersion: kubeflow.org/v2beta1
kind: MPIJob
metadata:
  name: nccltest
spec:
  slotsPerWorker: 8
  runPolicy:
    cleanPodPolicy: Running
  mpiReplicaSpecs:
    Launcher:
      replicas: 1
      template:
         spec:
           containers:
           - image: docker.io/deepops/nccl-tests:latest
             name: nccltest
             imagePullPolicy: IfNotPresent
             command:
             - sh
             - "-c"
             - |
               /bin/bash << 'EOF'
               mpirun --allow-run-as-root \
                 -np 16 \
                 -bind-to none -map-by slot \
                 -x NCCL_DEBUG=INFO \
                 -x NCCL_DEBUG_SUBSYS=NET \
                 -x NCCL_ALGO=RING \
                 -x NCCL_IB_DISABLE=0 \
                 -x LD_LIBRARY_PATH \
                 -x PATH \
                 -mca pml ob1 \
                 -mca btl self,tcp \
                 -mca btl_tcp_if_include 192.168.0.0/16 \
                 -mca oob_tcp_if_include 172.29.0.0/16 \
                 /nccl_tests/build/all_reduce_perf -b 8 -e 4G -f2 -g 1 \
                 && sleep infinity
               EOF

    Worker:
      replicas: 2
      template:
        metadata:
          annotations:
            k8s.v1.cni.cncf.io/networks: ibp12s0,ibp75s0,ibp141s0,ibp186s0
        spec:
```

```
          containers:
          - image: docker.io/deepops/nccl-tests:latest
            name: nccltest
            imagePullPolicy: IfNotPresent
            securityContext:
              capabilities:
                add: [ "IPC_LOCK" ]
            resources:
              limits:
                nvidia.com/resibp12s0: "1"
                nvidia.com/resibp75s0: "1"
                nvidia.com/resibp141s0: "1"
                nvidia.com/resibp186s0: "1"
                nvidia.com/gpu: 8
```

2. Run the `nccl_test` file.

```
# kubectl apply -f /network-operator/nccl_test.yaml
mpijob.kubeflow.org/nccltest created
root@basepod-head1:~#
# kubectl get pods
NAME                         READY   STATUS    RESTARTS   AGE
nccltest-launcher-9pp28      1/1     Running   0          3m6s
nccltest-worker-0            1/1     Running   0          3m6s
nccltest-worker-1            1/1     Running   0          3m6s
```

A sample logfile follows.

```
# kubectl logs -f nccltest-launcher-9pp28
...
#                                                              out-of-place                       in-place
#       size         count      type   redop    root     time   algbw   busbw #wrong     time   algbw   busbw #wrong
#        (B)    (elements)                               (us)  (GB/s)  (GB/s)            (us)  (GB/s)  (GB/s)

           8             2     float     sum      -1    41.28    0.00    0.00      0    40.99    0.00    0.00      0
          16             4     float     sum      -1    38.58    0.00    0.00      0    40.33    0.00    0.00      0
          32             8     float     sum      -1    40.15    0.00    0.00      0    39.73    0.00    0.00      0
          64            16     float     sum      -1    39.11    0.00    0.00      0    38.47    0.00    0.00      0
         128            32     float     sum      -1    39.66    0.00    0.01      0    40.57    0.00    0.01      0
         256            64     float     sum      -1    39.52    0.01    0.01      0    39.50    0.01    0.01      0
         512           128     float     sum      -1    41.58    0.01    0.02      0    40.09    0.01    0.02      0
        1024           256     float     sum      -1    42.30    0.02    0.05      0    41.71    0.02    0.05      0
        2048           512     float     sum      -1    45.81    0.04    0.08      0    44.70    0.05    0.09      0
        4096          1024     float     sum      -1    48.77    0.08    0.16      0    49.13    0.08    0.16      0
        8192          2048     float     sum      -1    55.33    0.15    0.28      0    55.52    0.15    0.28      0
       16384          4096     float     sum      -1    58.32    0.28    0.53      0    56.24    0.29    0.55      0
       32768          8192     float     sum      -1    60.59    0.54    1.01      0    58.35    0.56    1.05      0
       65536         16384     float     sum      -1    71.26    0.92    1.72      0    71.29    0.92    1.72      0
      131072         32768     float     sum      -1    93.71    1.40    2.62      0    103.5    1.27    2.38      0
      262144         65536     float     sum      -1    103.5    2.53    4.75      0    101.0    2.60    4.87      0
      524288        131072     float     sum      -1    137.8    3.80    7.13      0    148.8    3.52    6.60      0
     1048576        262144     float     sum      -1    107.9    9.72   18.22      0    109.6    9.57   17.94      0
     2097152        524288     float     sum      -1    120.7   17.38   32.59      0    120.9   17.34   32.51      0
     4194304       1048576     float     sum      -1    153.4   27.34   51.26      0    153.3   27.36   51.31      0
     8388608       2097152     float     sum      -1    230.4   36.40   68.26      0    229.6   36.54   68.50      0
    16777216       4194304     float     sum      -1    403.4   41.59   77.98      0    405.1   41.42   77.66      0
    33554432       8388608     float     sum      -1    735.5   45.62   85.54      0    736.4   45.57   85.44      0
    67108864      16777216     float     sum      -1   1426.1   47.06   88.23      0   1459.4   45.98   86.22      0
   134217728      33554432     float     sum      -1   2639.9   50.84   95.33      0   2682.3   50.04   93.82      0
   268435456      67108864     float     sum      -1   5202.4   51.60   96.75      0   5187.5   51.75   97.03      0
   536870912     134217728     float     sum      -1    10469   51.28   96.15      0    10327   51.99   97.48      0
  1073741824     268435456     float     sum      -1    20588   52.15   97.79      0    20786   51.66   96.86      0
  2147483648     536870912     float     sum      -1    41240   52.07   97.64      0    41165   52.17   97.81      0
  4294967296    1073741824     float     sum      -1    82270   52.21   97.89      0    82325   52.17   97.82      0
# Out of bounds values : 0 OK
# Avg bus bandwidth    : 34.0031
```

2.5 (Optional) Deploy Bright Jupyter

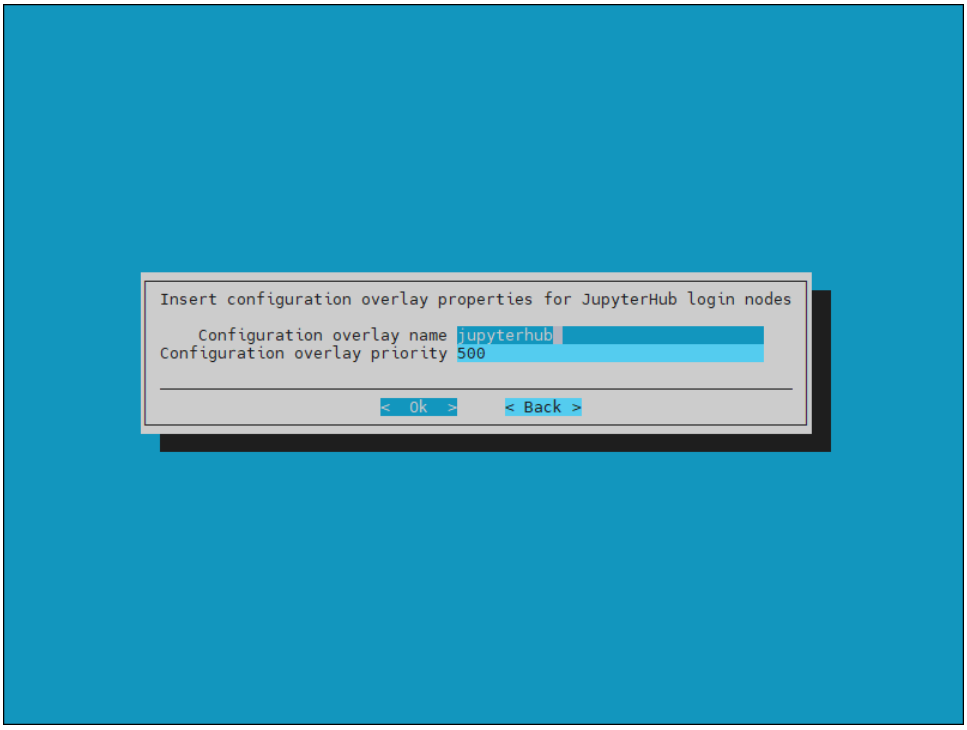
BCM provides a robust and popular Jupyter integration. Because the Bright Jupyter integration distributes the kernel across the cluster through the HPC workload management system or Kubernetes, Jupyter is generally installed on the head node or on a login node.

2.5.1 Install Jupyter Using the CLI Wizard

1. Run the `cm-jupyter-setup` CLI wizard on the head node as the root user.
   ```
   # cm-jupyter-setup
   ```
2. Choose `Deploy` to continue.

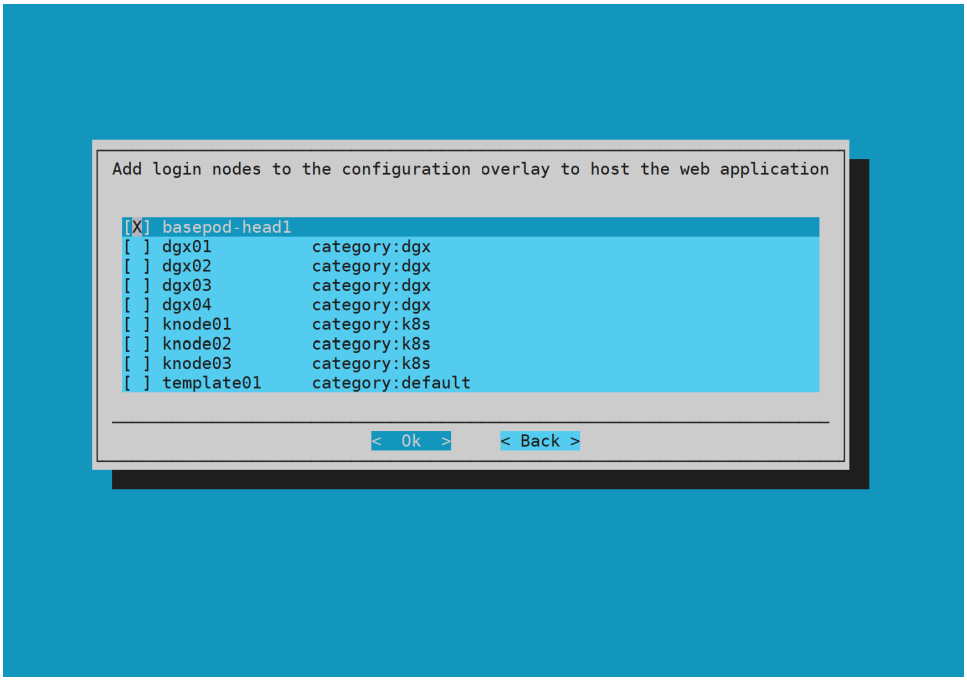
3. Specify the overlay name and priority for the JupyterHub login nodes.

   By default, the Jupyter wizard will create a configuration overlay named `jupyterhub` with a priority of `500`. Use the defaults unless there is an existing `jupyterhub` overlay.

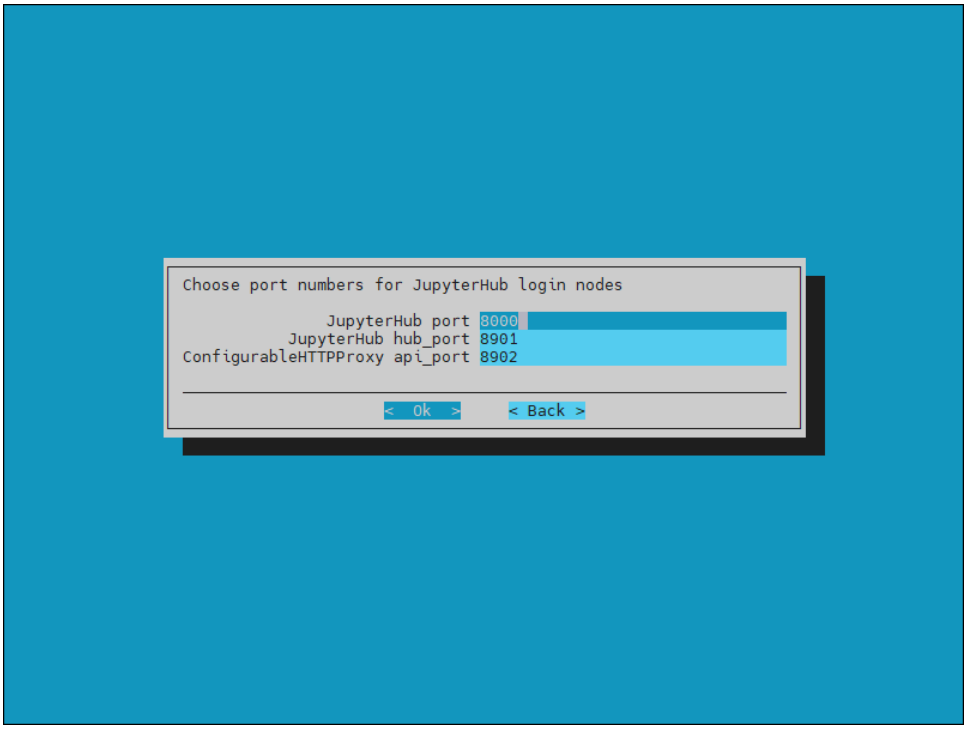


4. Select `basepod-head1` and then `Ok`.

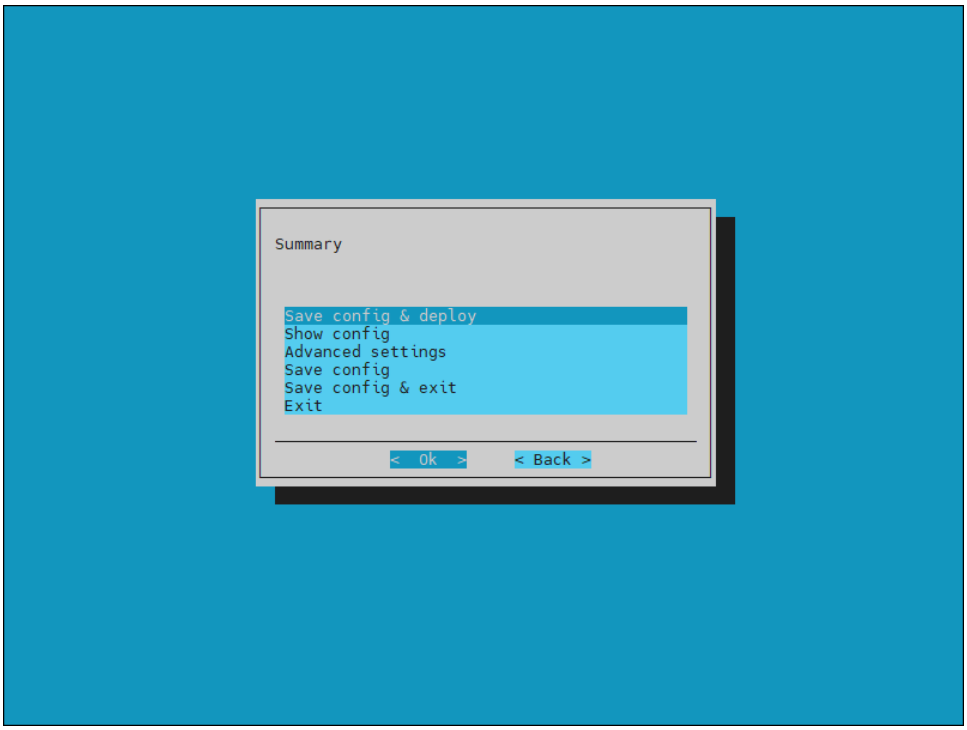
   After HA is configured, the `cm-jupyterhub` service will be set to always run on the active head node.



5. Select the default ports of `8000`, `8901`, and `8902` and select `Ok`.
   Users will access it on the active head node on port `8000`.

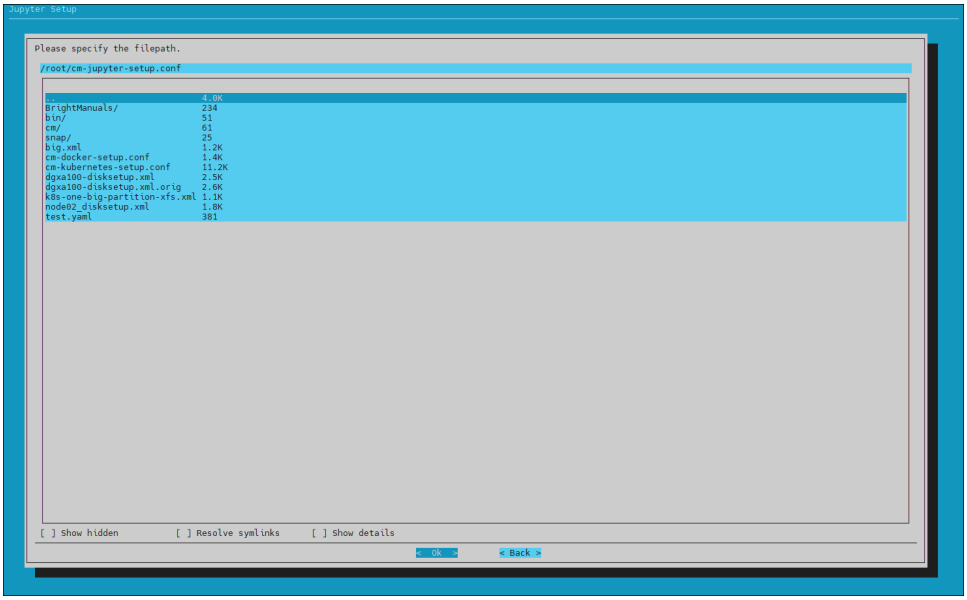


6. Select `Save config & deploy` and then `Ok`.



7. Select `Ok` to start the installation.

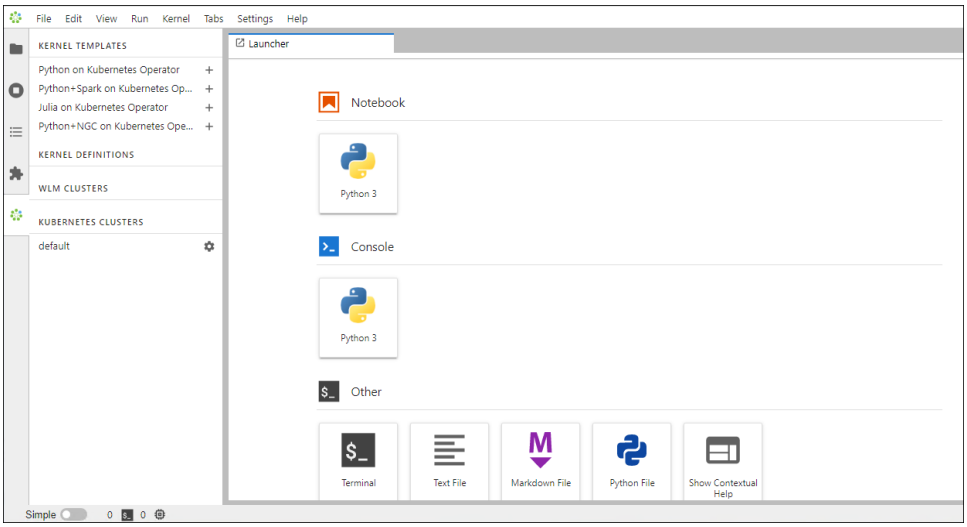
   By default, the Jupyter wizard will save the deployment configuration in `/root/cm-jupyter-setup.conf`. This configuration file can be used to redeploy Jupyter in the future.



8. When the installation completes, the `cm-jupyter` service will automatically be started on the selected node.

   All users in the cluster (except the root user) will be able to login to Jupyterhub using a web browser at `http://<head-node-ip or FQDN>:8000`.

   Example: http://10.130.121.254:8000



9. If needed, a test user can be created with following command:

```
# cmsh -c "user; add jupyterhubuser; set password jupyterhubuser; commit"
```

10. Add the user to K8s.

```
# cm-kubernetes-setup --add-user jupyterhubuser --operators cm-jupyter-kernel-operator
```

Chapter 3. High Availability

This section covers how to configure high availability (HA) using `cmha-setup` CLI wizard.

1. Ensure that both head nodes are licensed.

   We provided the MAC address for the secondary head when we installed the cluster license (section 3.2.2.1).

```
% main licenseinfo  | grep ^MAC
MAC address / Cloud ID               04:3F:72:E7:67:07|14:02:EC:DA:AF:18
```

2. Configure the NFS shared storage.

   Mounts configured in `fsmounts` will be automatically mounted by the `CMDaemon`.

```
% device
% use master
% fsmounts
% add /nfs/general
% set device 10.130.122.252:/var/nfs/general
% set filesystem nfs
% commit
% show
Parameter                      Value
------------------------------ ------------------------------------------------
Device                         10.130.122.252:/var/nfs/general
Revision
Filesystem                     nfs
Mountpoint                     /nfs/general
Dump                           no
RDMA                           no
Filesystem Check               NONE
Mount options                  defaults
```

3. Verify that the shared storage is mounted.

```
# mount | grep '/nfs/general'
10.130.122.252:/var/nfs/general on /nfs/general type nfs4
(rw,relatime,vers=4.2,rsize=1048576,wsize=1048576,namlen=255,hard,proto=tcp,timeo
=600,retrans=2,sec=sys,clientaddr=10.130.122.254,local_lock=none,addr=10.130.122.
252)
```

4. Verify that head node has power control over the cluster nodes.

```
% device
% power -c dgx,k8s-master status
[basepod-head1->device]% power -c dgx,k8s-master status
ipmi0 ..................... [   ON    ] dgx01
ipmi0 ..................... [   ON    ] dgx02
ipmi0 ..................... [   ON    ] dgx03
ipmi0 ..................... [   ON    ] dgx04
ipmi0 ..................... [   ON    ] knode01
ipmi0 ..................... [   ON    ] knode02
ipmi0 ..................... [   ON    ] knode03
[basepod-head1->device]%
```

5. Power off the cluster nodes.

   The cluster nodes must be powered off before configuring HA.

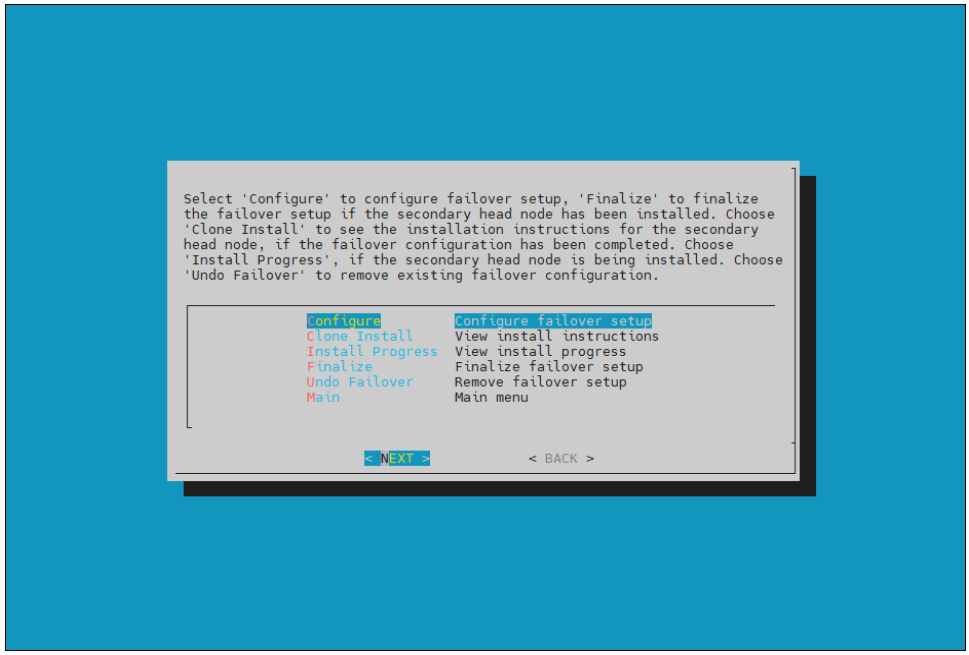
```
% power -c k8s-master,dgx off
ipmi0 ..................... [   OFF   ] knode01
ipmi0 ..................... [   OFF   ] knode02
ipmi0 ..................... [   OFF   ] knode03
ipmi0 ..................... [   OFF   ] dgx01
ipmi0 ..................... [   OFF   ] dgx02
ipmi0 ..................... [   OFF   ] dgx03
ipmi0 ..................... [   OFF   ] dgx04
```

6. Start the `cmha-setup` CLI wizard as the root user on the primary head node.

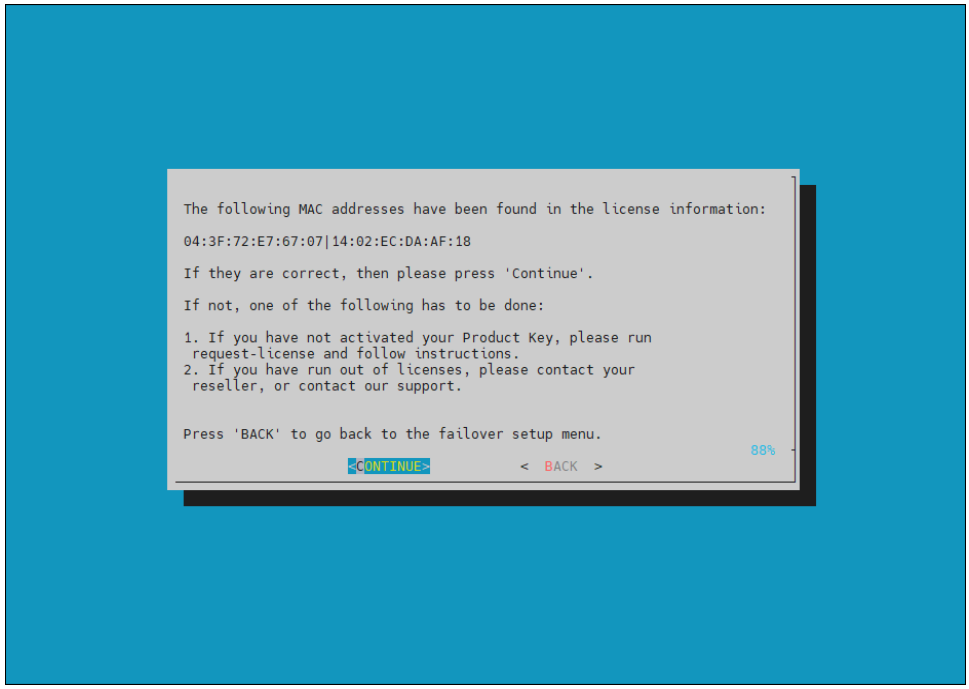
```
# cmha-setup
```

7. Select `Setup`.

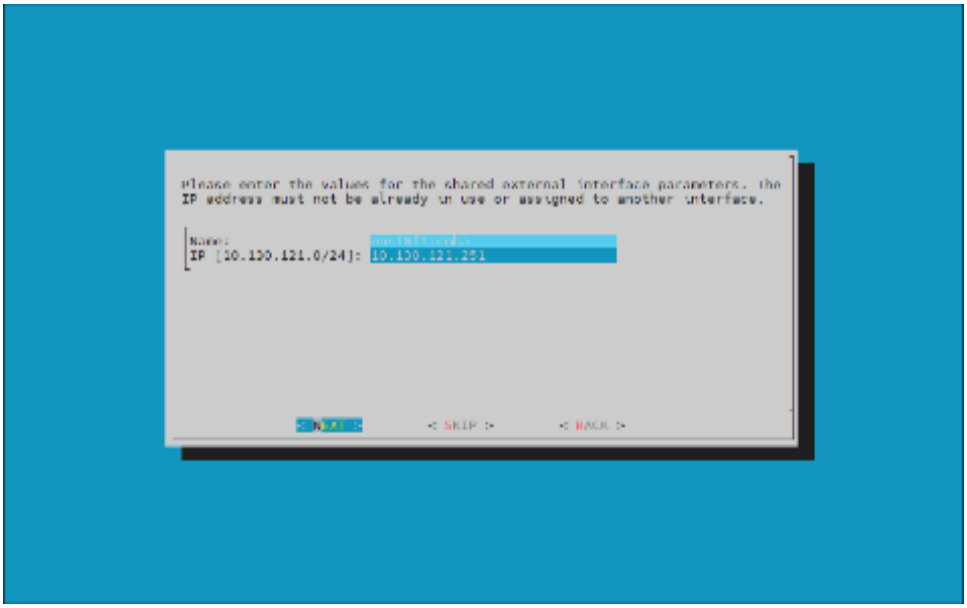
8. Select `Configure`.



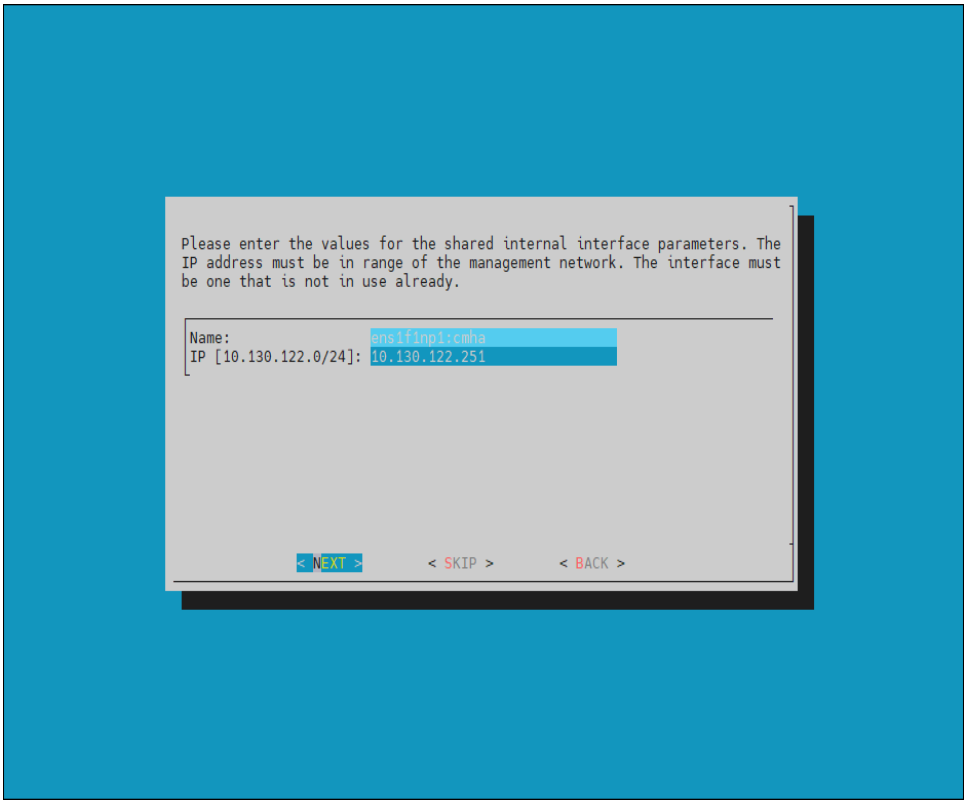
9. Verify that the cluster license information found by the wizard is correct.



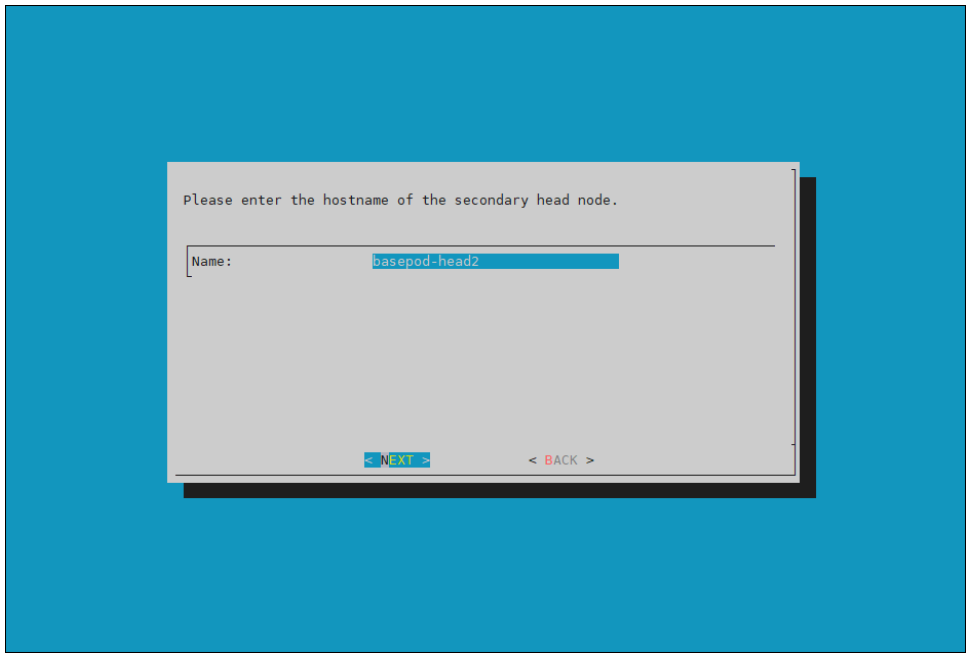
10. Configure an external virtual IP address that will be used by the active head node in the HA configuration. (This will be the IP that should always be used for accessing the active head nodes.)



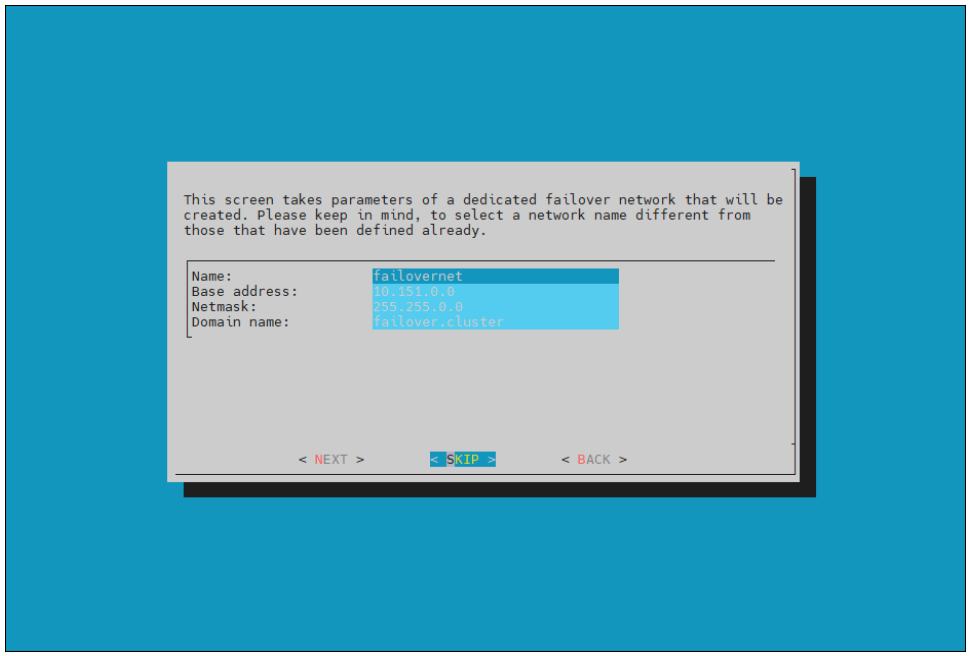
11. Provide an internal virtual IP address that will be used by the active head node in the HA configuration.



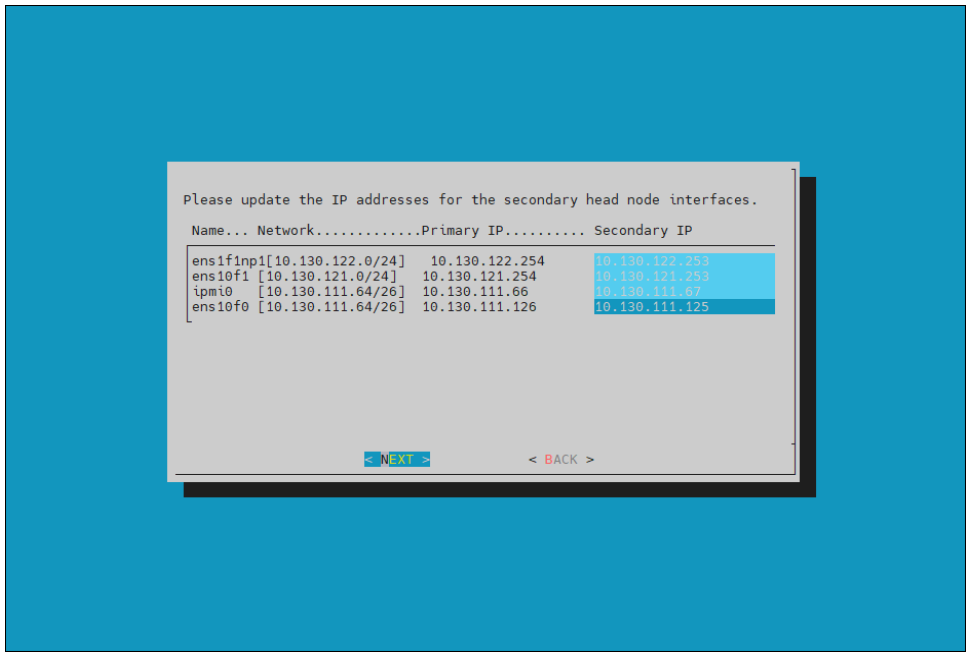
12. Provide the name of the secondary head node.



13. DGX BasePOD uses the internal network as the failover network, so select `SKIP` to continue.

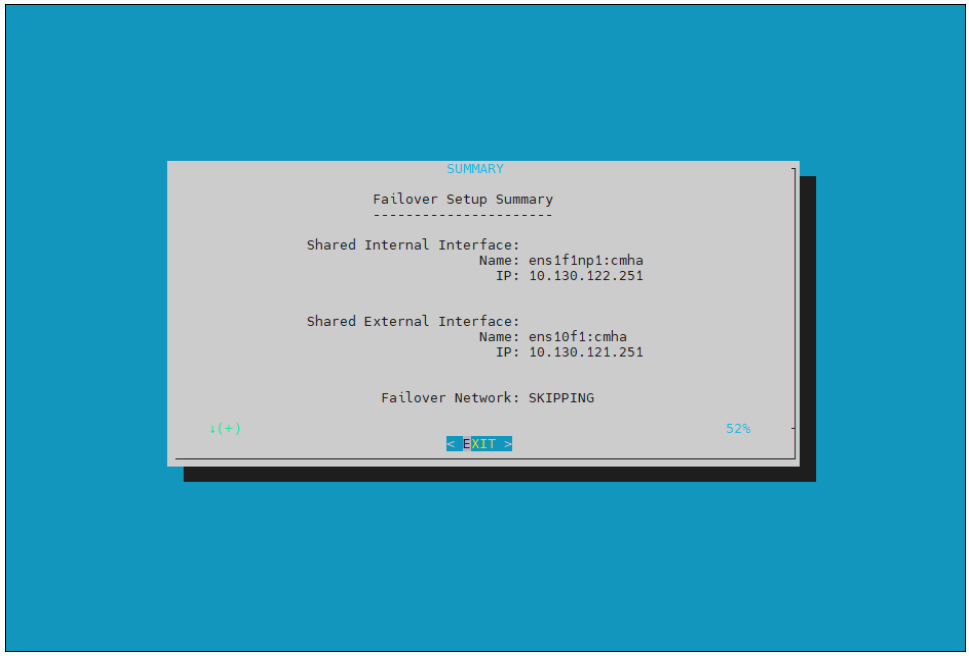


14. Configure the IP addresses for the secondary head node.

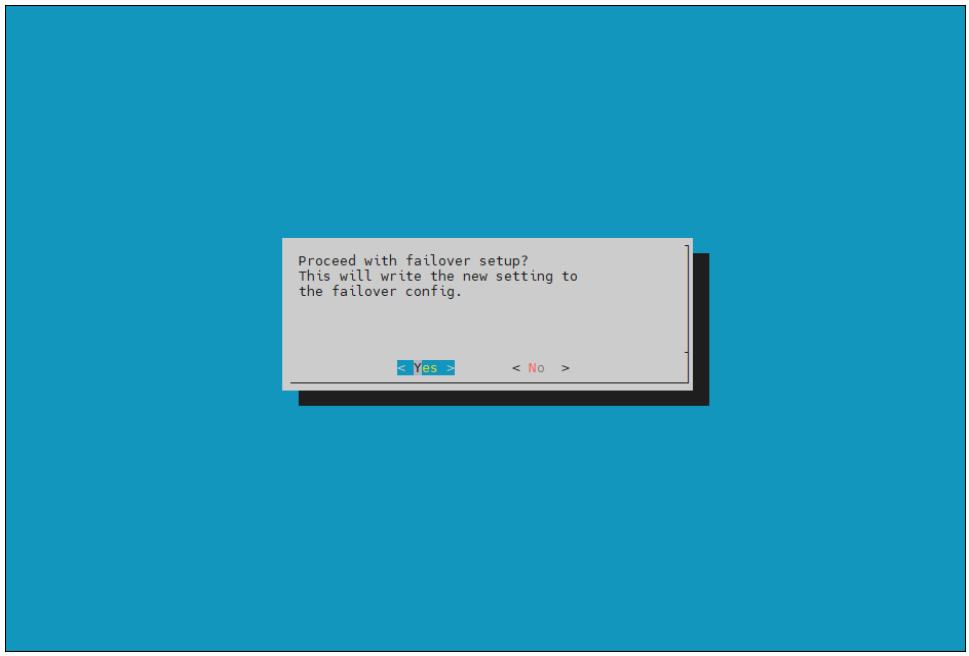


15. The wizard shows a summary of the information that it has collected.

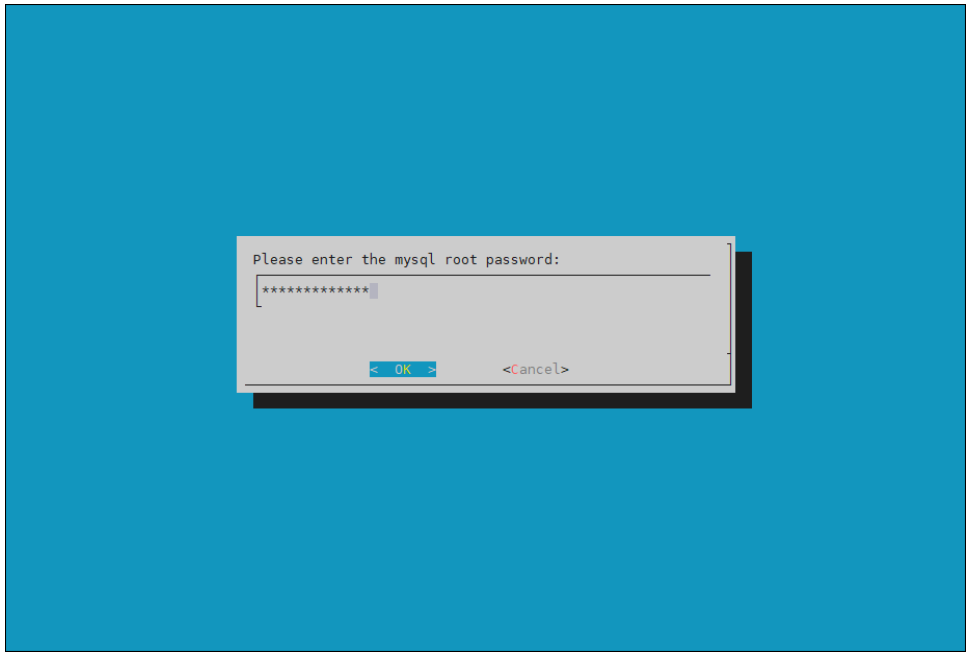
    This screen shoes the VIP that will be assigned to the internal and external interfaces.



16. Select `Yes` to proceed with the failover configuration.



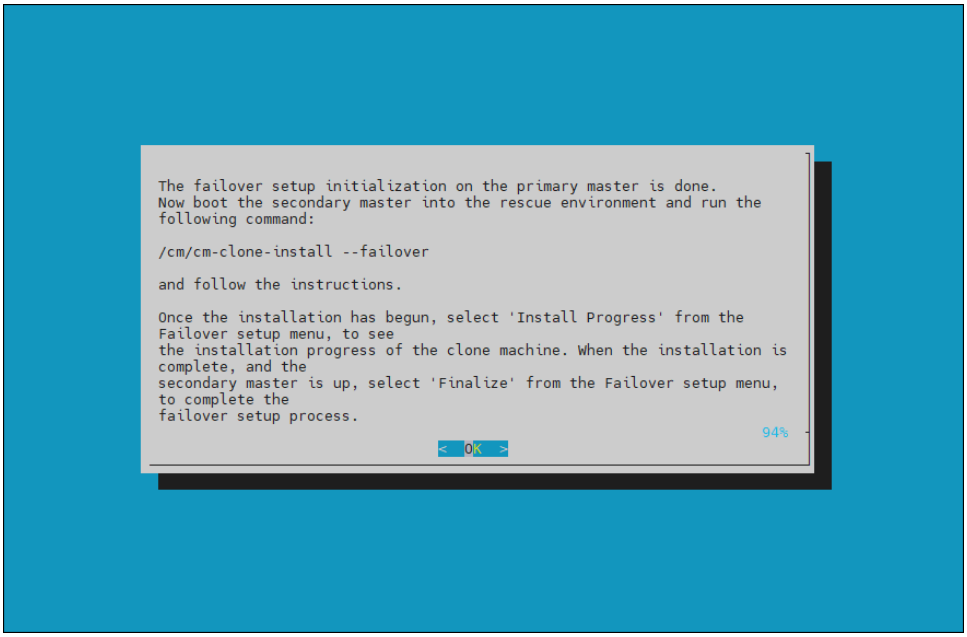
17. Enter the MySQL root password. This should be the same as the root password.



18. The wizard implements the first steps in the HA configuration. If all the steps show `OK`, press `ENTER` to continue. The progress is shown below:

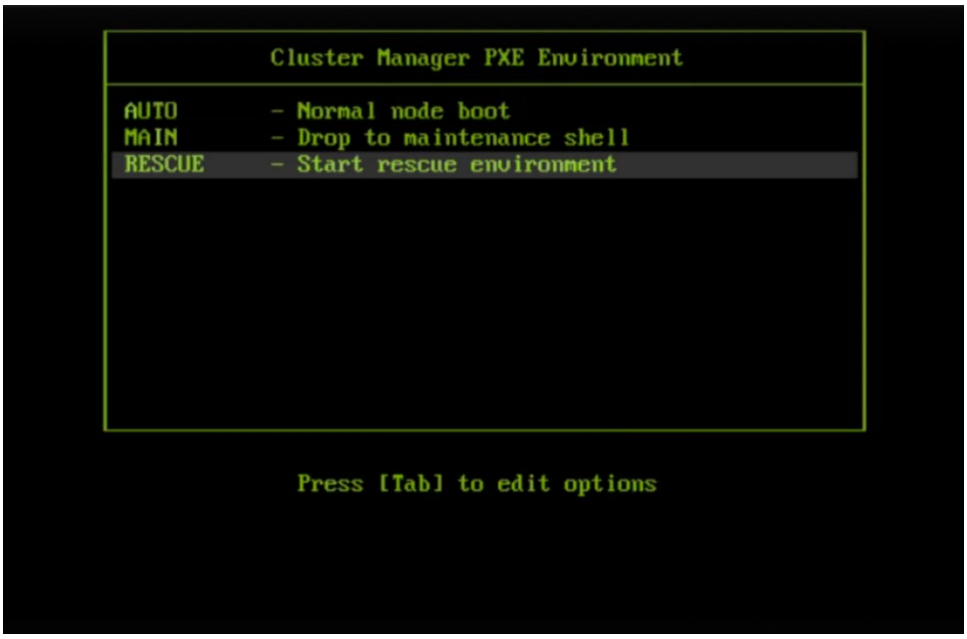
```
Initializing failover setup on master..... [  OK  ]
Updating shared internal interface........ [  OK  ]
Updating shared external interface........ [  OK  ]
Updating extra shared internal interfaces.... [  OK  ]
Cloning head node......................... [  OK  ]
Updating secondary master interfaces...... [  OK  ]
Updating Failover Object.................. [  OK  ]
Restarting cmdaemon....................... [  OK  ]
Press any key to continue
```

19. Run the `/cm/cm-clone-install -failover` command on the secondary head node. Note that this should be a one-time network boot.



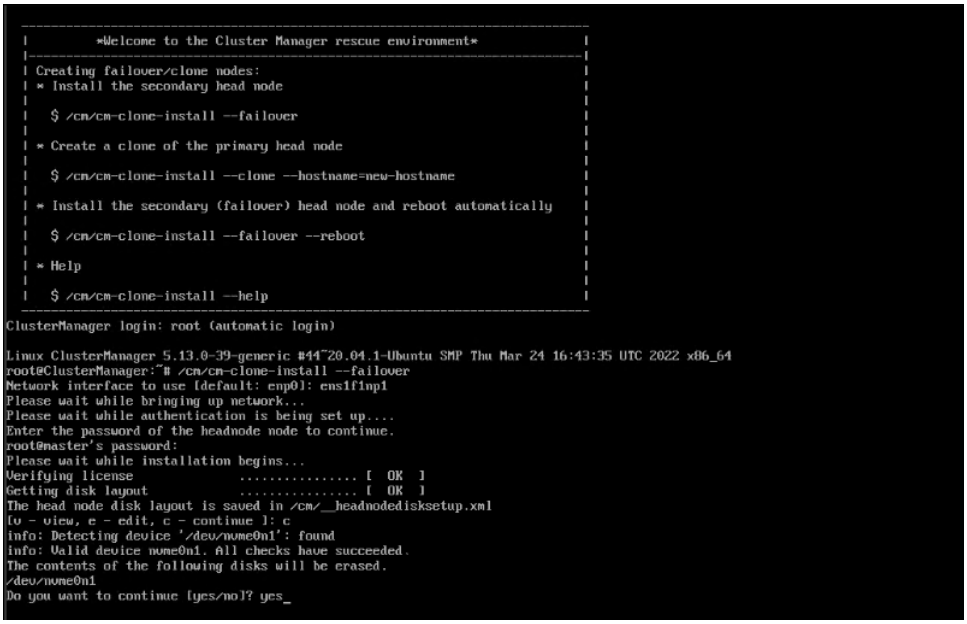
20. PXE boot the secondary head node, then select `RESCUE` from the grub menu.

    Since this is the initial boot of this node, it must be done outside of Base Command Manager (BMC or physical power button).

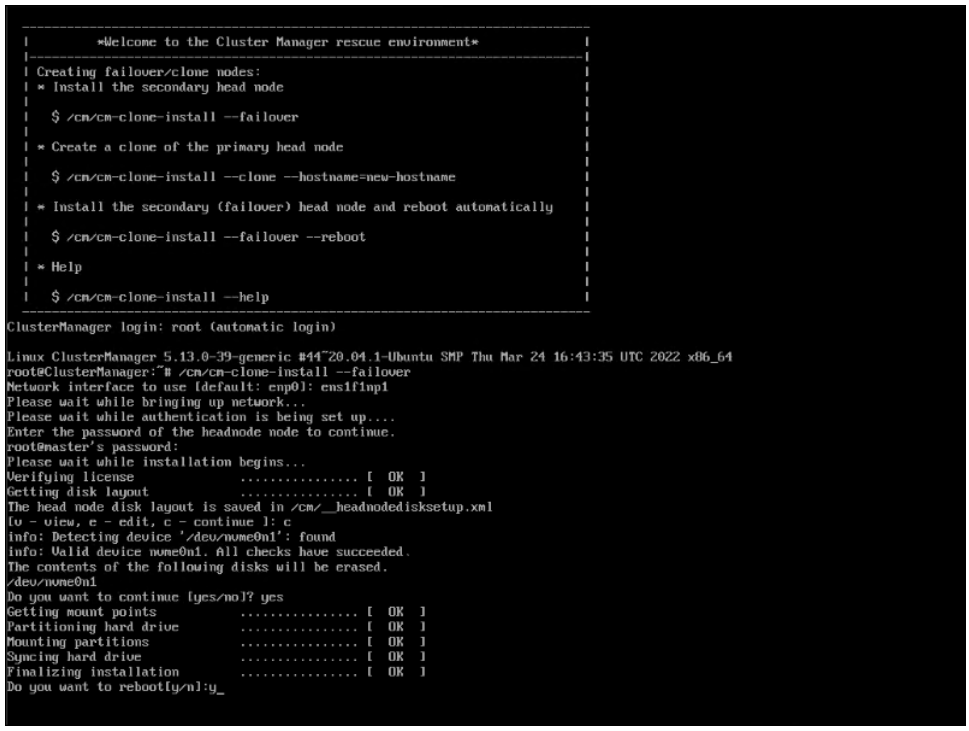


21. Once the secondary head node has booted into the rescue environment, run the `/cm/cm-clone-install -failover` command, then enter yes when prompted.

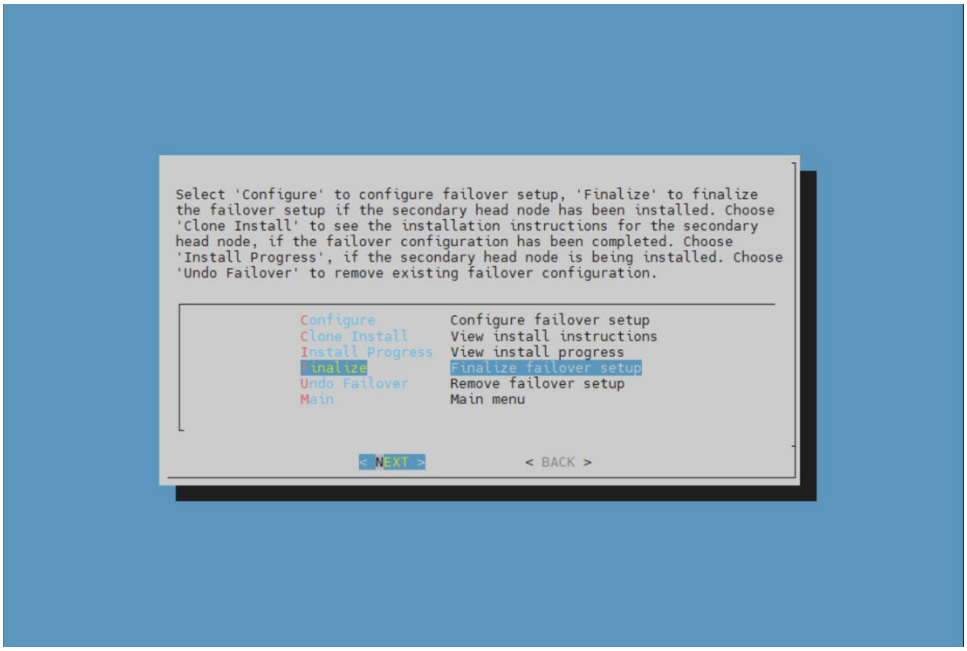
    The secondary head node will be cloned from the primary.



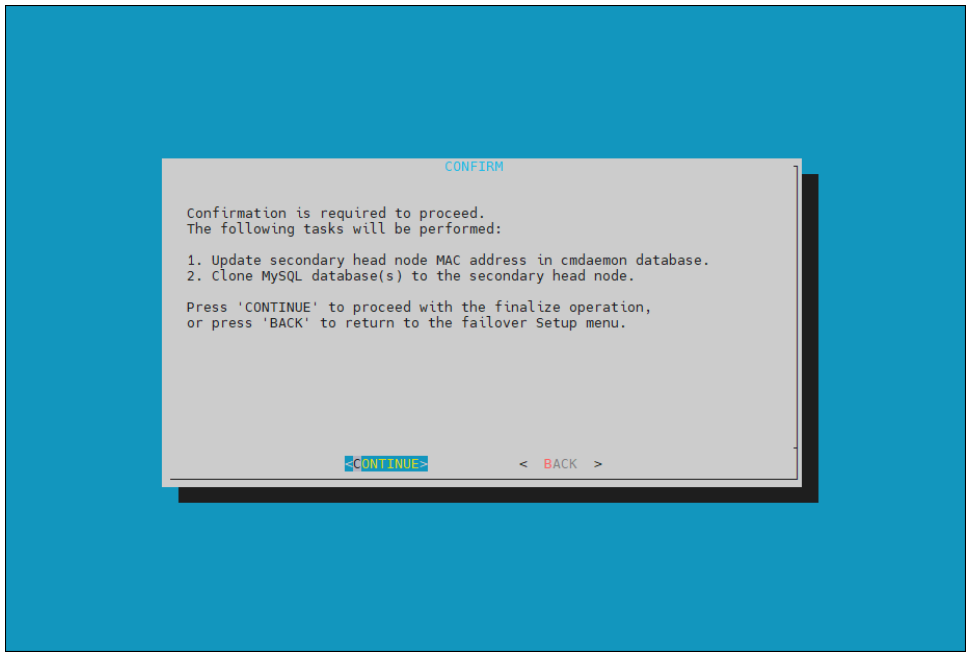
22. When cloning is completed, enter `y` to reboot the secondary head node.
    The secondary needs to boot from its hard drive. PXE boot should not be enabled.



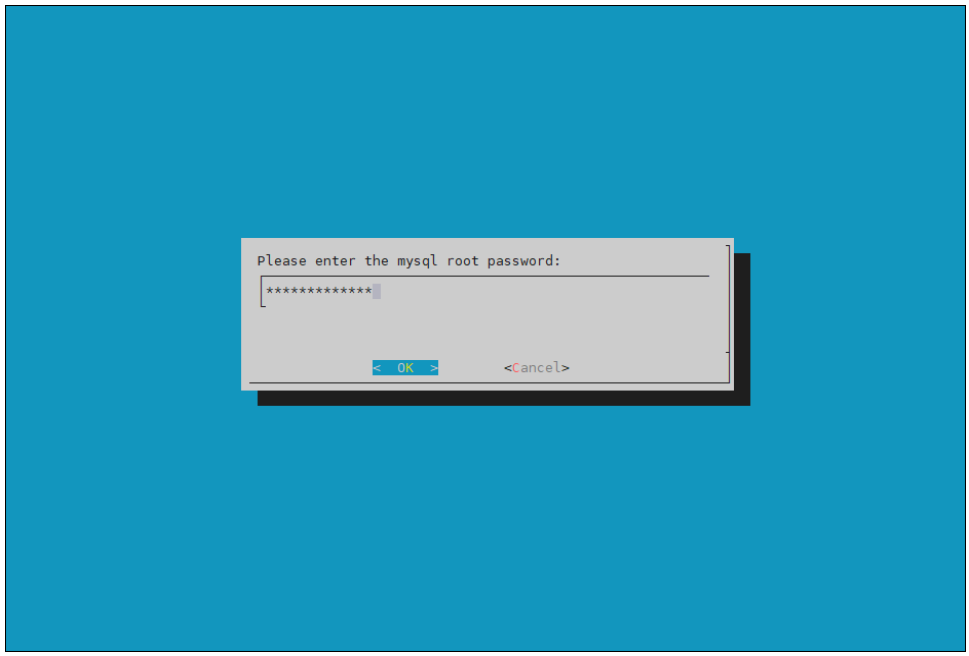
23. Wait for the secondary head node to reboot and then continue the HA setup procedure on the primary head node.
24. Select `finalize` from the `cmha-setup` menu.
    This will clone the MySQL database from the primary to the secondary head node.



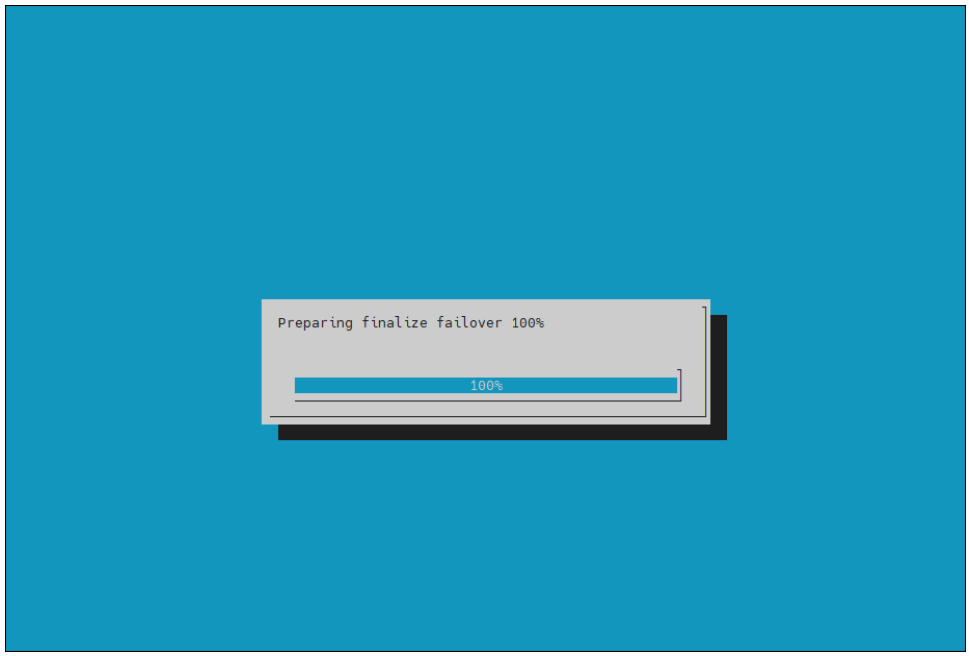
25. Select <CONTINUE> on the confirmation screen.



26. Enter the MySQL root password. This should be the same as the root password.



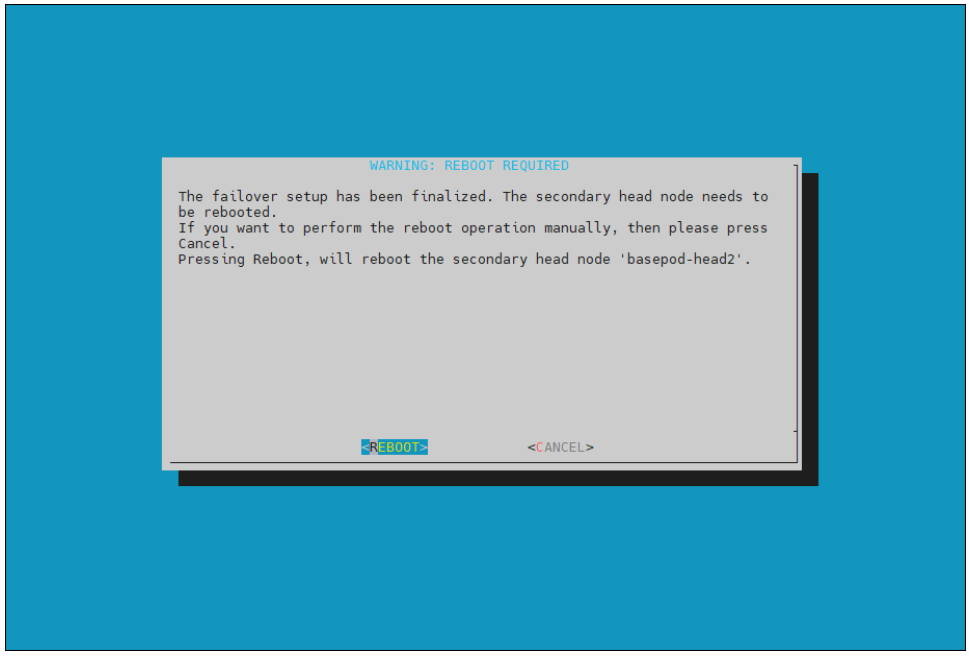
27. The `cmha-setup` wizard continues. Press `ENTER` to continue when prompted.



The progress is shown below:

```
Updating secondary master mac address.............. [  OK  ]
Initializing failover setup on basepod-head2....... [  OK  ]
Stopping cmdaemon.................................. [  OK  ]
Cloning cmdaemon database.......................... [  OK  ]
Checking database consistency...................... [  OK  ]
Starting cmdaemon, chkconfig services.............. [  OK  ]
Cloning workload manager databases................. [  OK  ]
Cloning additional databases....................... [  OK  ]
Update DB permissions.............................. [  OK  ]
Checking for dedicated failover network............ [  OK  ]
Press any key to continue
```

28. The `Finalize` step is now completed. Select `<REBOOT>` and wait for the secondary head node to reboot.

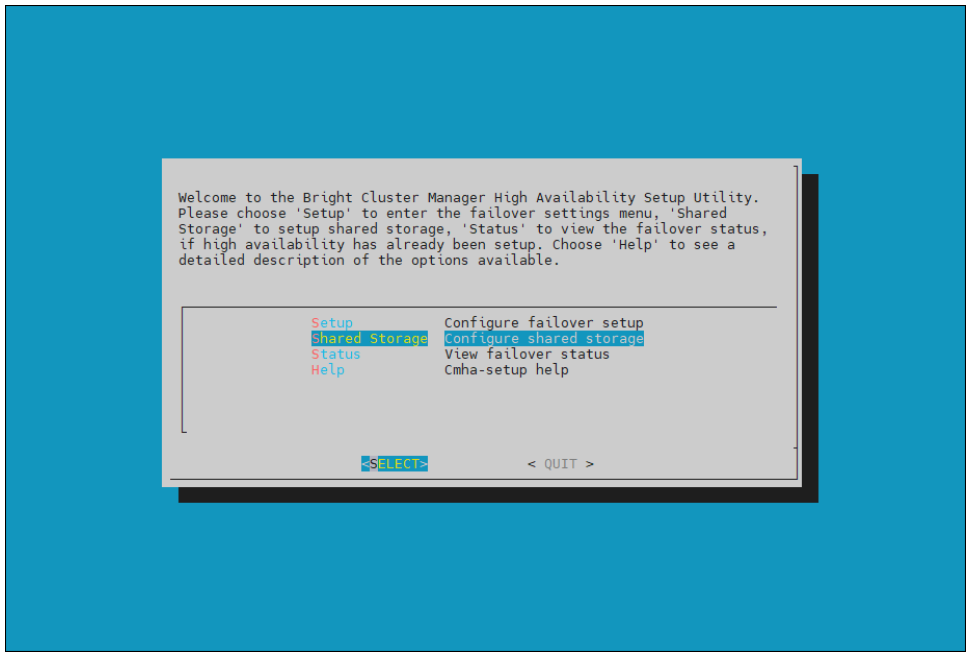


29. The secondary head node is now UP.

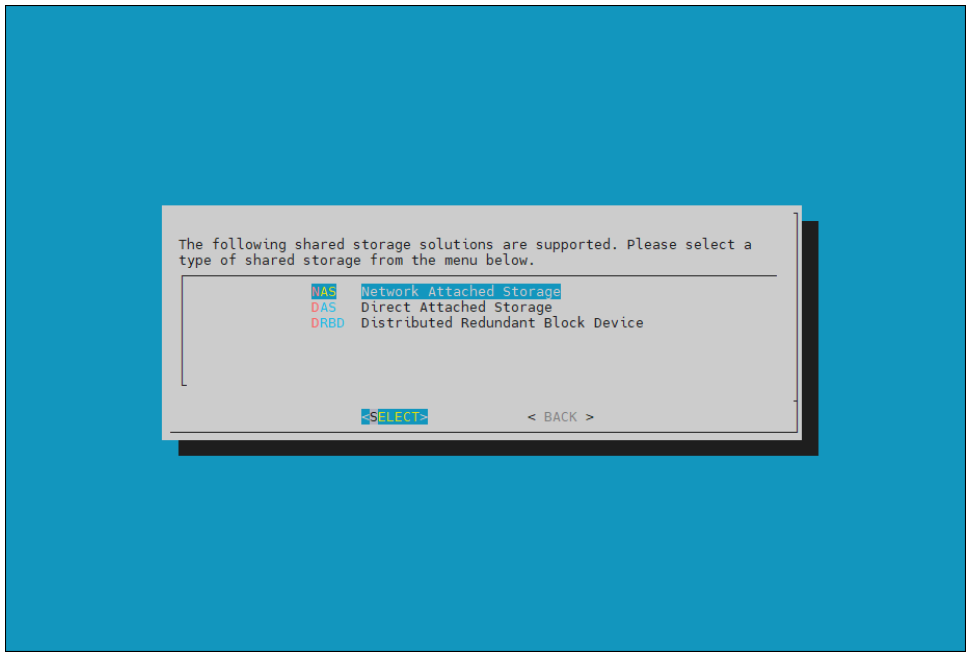
```
% device list -f hostname:20,category:12,ip:20,status:15
hostname (key)       category     ip                   status
-------------------- ---------- -------------------- ---------------
basepod-head1                     10.130.122.254       [   UP   ]
basepod-head2                     10.130.122.253       [   UP   ]
knode01              k8s-master   10.130.122.9         [  DOWN  ]
knode02              k8s-master   10.130.122.10        [  DOWN  ]
knode03              k8s-master   10.130.122.11        [  DOWN  ]
dgx01                dgx          10.130.122.5         [  DOWN  ]
dgx02                dgx          10.130.122.6         [  DOWN  ]
dgx03                dgx          10.130.122.7         [  DOWN  ]
dgx04                dgx          10.130.122.8         [  DOWN  ]
```

30. Select `Shared Storage` from the `cmha-setup` menu.

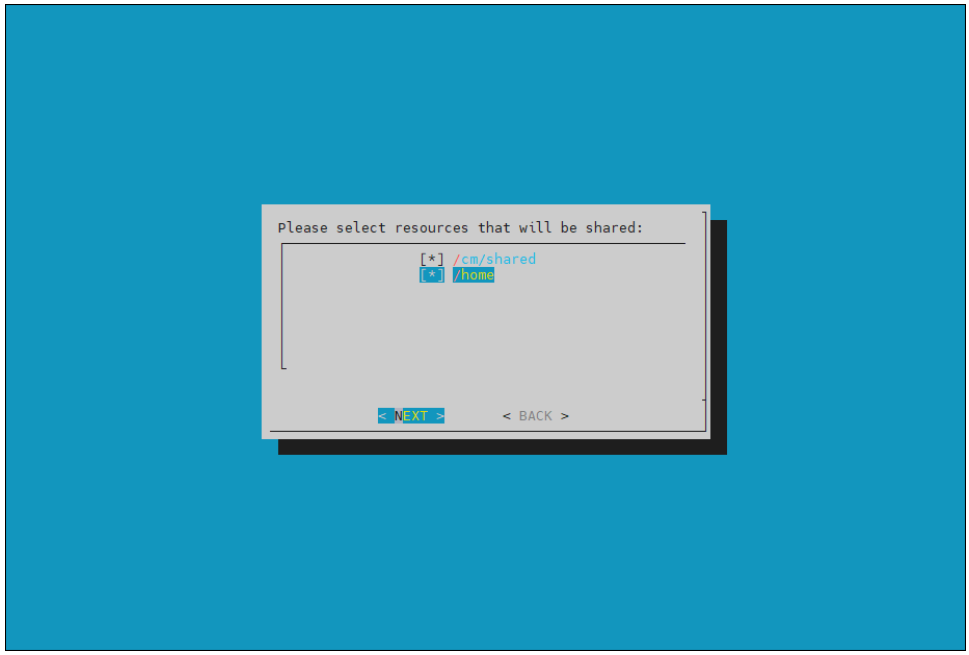
    In this final HA configuration step, `cmha-setup` will copy the `/cm/shared` and `/home` directories to the shared storage, and it configures both head nodes and all cluster nodes to mount it.



31. Select `NAS`.

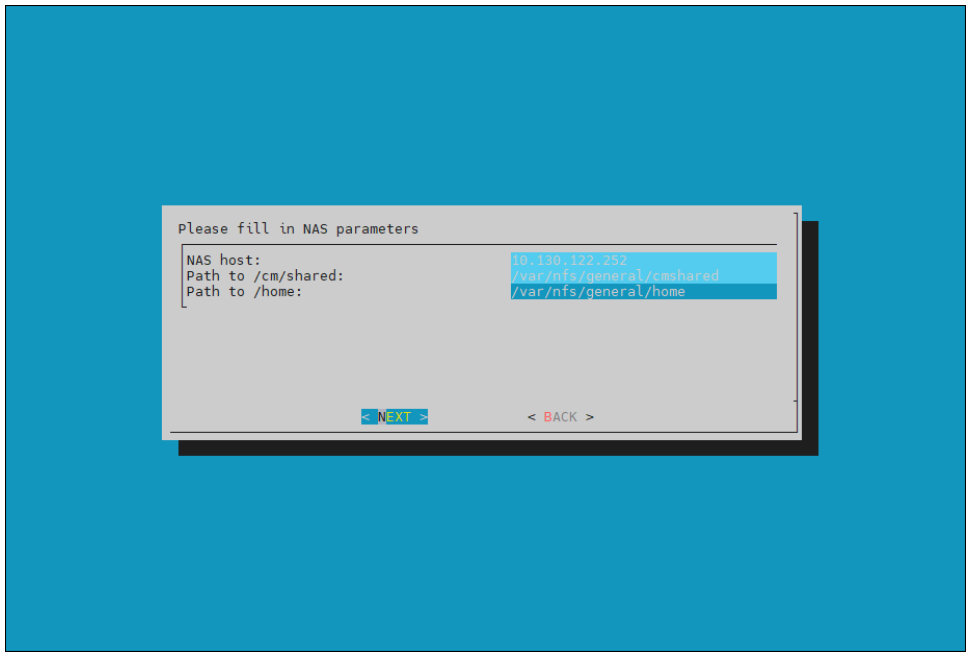


32. Select both `/cm/shared` and `/home`.

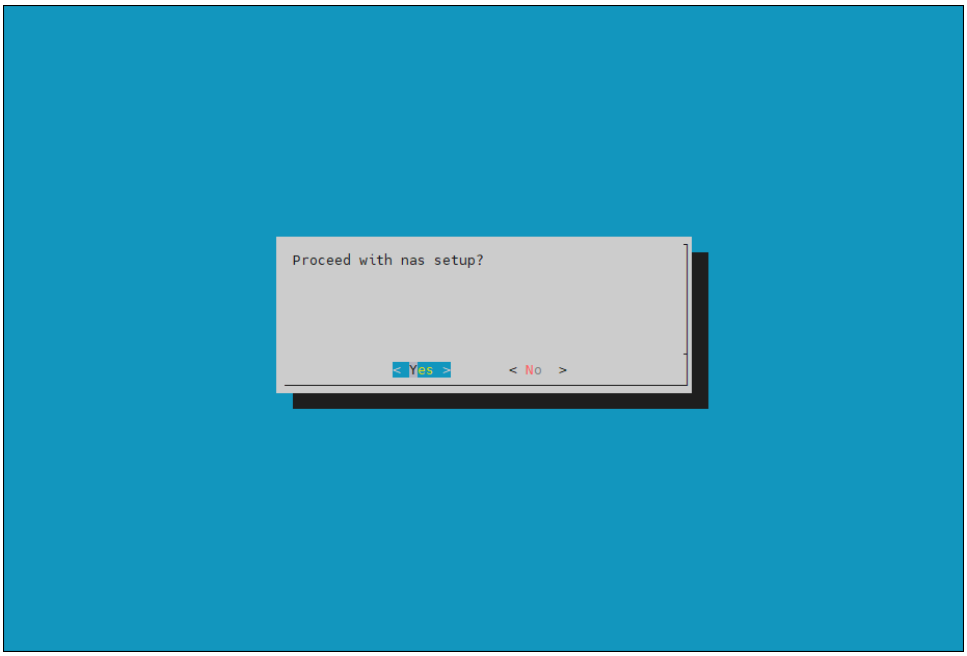


33. Provide the IP address of the NAS host, and the path that the `/cm/shared` and `/home` directories should be copied to on the shared storage.

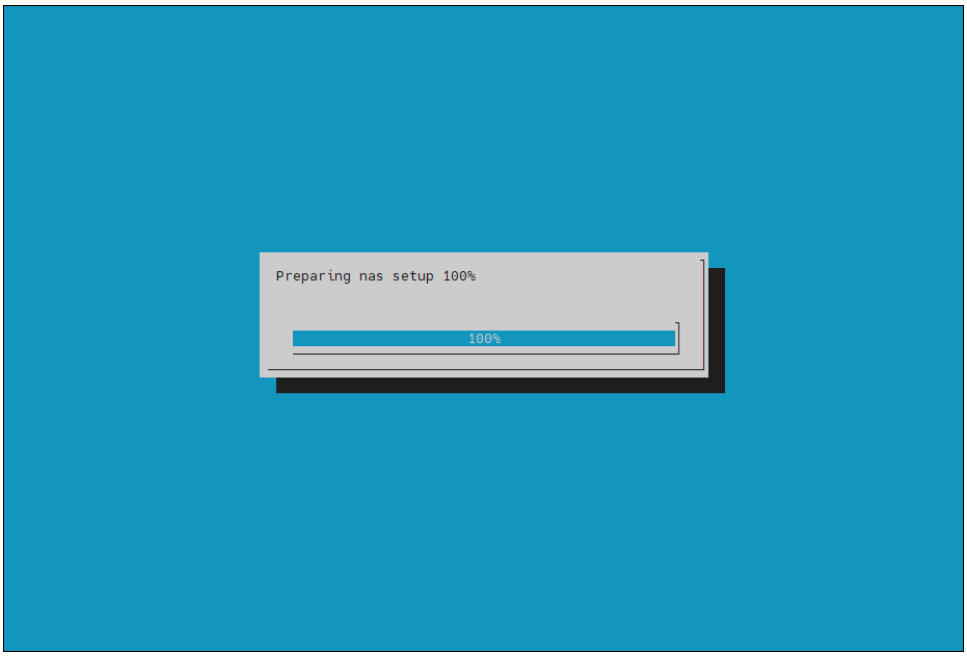
    In this case, /`var/nfs/general` is exported, so the `/cm/shared` directory will be copied to `10.130.122.252:/var/nfs/general/cmshared`, and it will be mounted over `/cm/shared` on the cluster nodes.



34. The wizard shows a summary of the information that it has collected. Press `ENTER` to continue.
35. Select YES when prompt to proceed with the setup.



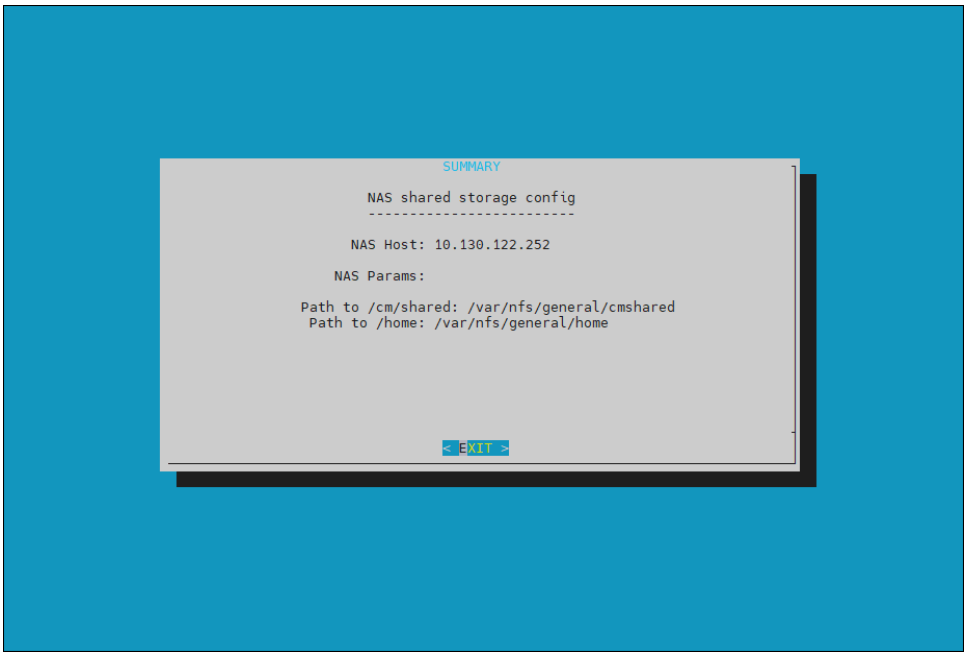
36. The `cmha-setup` wizard proceeds with its work. When it completes, select `ENTER` to finish HA setup.



The progress is shown below:

```
Copying NAS data..................................... [  OK  ]
Mount NAS storage.................................... [  OK  ]
Remove old fsmounts.................................. [  OK  ]
Add new fsmounts..................................... [  OK  ]
Remove old fsexports................................. [  OK  ]
Write NAS mount/unmount scripts...................... [  OK  ]
Copy mount/unmount scripts........................... [  OK  ]
Press any key to continue
```

37. `cmha-setup` is now complete. <`EXIT`> the wizard to return to the shell prompt.



3.1.1 Verify HA Setup

1. Run the `cmha status` command to verify that the failover configuration is correct and working as expected.

   Note that the command tests the configuration from both directions: from the primary head node to the secondary, and from the secondary to the primary. The active head node is indicated by an asterisk.

```
# cmha status
Node Status: running in active mode

basepod-head1* -> basepod-head2
  mysql         [  OK  ]
  ping          [  OK  ]
  status        [  OK  ]

basepod-head2 -> basepod-head1*
  mysql         [  OK  ]
  ping          [  OK  ]
  status        [  OK  ]
```

2. Verify that the `/cm/shared` and `/home` directories are being mounted from the NAS server.

```
# mount
. . . some output omitted . . .
10.130.122.252:/var/nfs/general/cmshared on /cm/shared type nfs4
(rw,relatime,vers=4.2,rsize=32768,wsize=32768,namlen=255,hard,proto=tcp,timeo=600
,retrans=2,sec=sys,clientaddr=10.130.122.253,local_lock=none,addr=10.130.122.252)
10.130.122.252:/var/nfs/general/home on /home type nfs4
(rw,relatime,vers=4.2,rsize=32768,wsize=32768,namlen=255,hard,proto=tcp,timeo=600
,retrans=2,sec=sys,clientaddr=10.130.122.253,local_lock=none,addr=10.130.122.252)
```

3. Login to the head node to be made active and run `cmha makeactive`.

```
# ssh basepod-head2
# cmha makeactive
================================================================================
This is the passive head node. Please confirm that this node should
become
the active head node. After this operation is complete, the HA status of
the head nodes will be as follows:

basepod-head2 will become active head node (current state: passive)
basepod-head1 will become passive head node (current state: active)
================================================================================

Continue(c)/Exit(e)? c

Initiating failover.............................. [  OK  ]

basepod-head2 is now active head node, makeactive successful
```

4. Run the `cmha status` command again to verify that the secondary head node has become the active head node.

```
# cmha status
Node Status: running in active mode

basepod-head2* -> basepod-head1
  mysql          [  OK  ]
  ping           [  OK  ]
  status         [  OK  ]

basepod-head1 -> basepod-head2*
  mysql          [  OK  ]
  ping           [  OK  ]
  status         [  OK  ]
```

5. Manually failover back to the primary head node.

```
# ssh basepod-head1
# cmha makeactive

================================================================================
This is the passive head node. Please confirm that this node should become
the active head node. After this operation is complete, the HA status of
the head nodes will be as follows:

basepod-head1 will become active head node (current state: passive)
basepod-head2 will become passive head node (current state: active)
================================================================================

Continue(c)/Exit(e)? c

Initiating failover.............................. [  OK  ]

basepod-head1 is now active head node, makeactive successful
```

6. Run `cmsh status` again to verify that the primary head node has become the active head node.

```
# cmha status
Node Status: running in active mode

basepod-head1* -> basepod-head2
  mysql          [  OK  ]
  ping           [  OK  ]
  status         [  OK  ]

basepod-head2 -> basepod-head1*
  mysql          [  OK  ]
  ping           [  OK  ]
  status         [  OK  ]
```

7. Power on the cluster nodes.

```
# cmsh -c "power -c k8s-master,dgx on"
ipmi0 ..................... [   ON    ] knode01
ipmi0 ..................... [   ON    ] knode02
ipmi0 ..................... [   ON    ] knode03
ipmi0 ..................... [   ON    ] dgx01
ipmi0 ..................... [   ON    ] dgx02
ipmi0 ..................... [   ON    ] dgx03
ipmi0 ..................... [   ON    ] dgx04
```

8. (Optionally) Configure Jupyter HA

   If Jupyter was deployed on the primary head node before HA was configured, configure the Jupyter service to run on the active head node.

```
% device
% use basepod-head1
% services
% use cm-jupyterhub
% show
Parameter                        Value
-------------------------------- ------------------------------------------------
Revision
Service                          cm-jupyterhub
Run if                           ALWAYS
Monitored                        yes
Autostart                        yes
Timeout                          -1
Belongs to role                  yes
Sickness check script
Sickness check script timeout    10
Sickness check interval          60
```

9. Set the `runif` parameter to `active`.

```
% set runif active
% commit

% show
Parameter                        Value
-------------------------------- ------------------------------------------------
Revision
Service                          cm-jupyterhub
Run if                           ACTIVE
Monitored                        yes
Autostart                        yes
Timeout                          -1
Belongs to role                  yes
Sickness check script
Sickness check script timeout    10
Sickness check interval          60
```

10. Configure the Jupyter service on the secondary head node.

```
% device
% use basepod-head2
% services
% use cm-jupyterhub
% set runif active
```

Chapter 4. Basic User Management

BCM uses its own LDAP service to manage users and groups with a centralized LDAP database server running on the head node, and not by entries in `/etc/passwd` or `/etc/group` files. An external LDAP server can be setup for authentication services to replace the existing Bright LDAP service, but it is outside of the scope of this document.

Only the basic user management tasks are outlined in this guide to provide a starting point. Refer to the *Bright Cluster Manager Administrator Manual* for complete options and additional details.

Although user management can be done in both `cmsh` and Bright View, `cmsh` is used in this chapter.

4.1 Configuring a User

1. Add a user (`userone` in this case).

```
# cmsh
% user
% add userone
% set password 7adGnv0!K
% commit
```

2. `userone` will reset the password after successfully logging in.

```
userone@basepod-head2:~$ passwd
(current) LDAP Password:
New password:
Retype new password:
passwd: password updated successfully
userone@basepod-head2:~$
```

3. Use `show` to view user parameters and values.

```
[basepod-head2->user[userone]]% show
Parameter                       Value
------------------------------- ------------------------------------------
Accounts
Managees
Name                            userone
Primary group                   userone
Revision
Secondary groups
ID                              1004
Common name                     userone
Surname                         userone
Group ID                        1004
Login shell                     /bin/bash
Home directory                  /home/ userone
Password                        *********
email
Profile
Create cmjob certificate        no
Write ssh proxy config          no
Shadow min                      0
Shadow max                      999999
Shadow warning                  7
Inactive                        0
Last change                     2022/10/20
Expiration date                 2037/12/31
Project manager                 <submode>
Notes                           <0B>
```

4. Use `set` to change parameters.

```
[basepod-head2->user[userone]]% set
commonname             expirationdate         id                     name
profile                shadowmax              surname
createcmjobcertificate groupid                inactive               notes
projectmanager         shadowmin              writesshproxyconfig
email                  homedirectory          loginshell             password
revision               shadowwarning
```

4.1.1 Procedures to Remove a User

This block of code will delete a user.

```
# cmsh
% user
% remove userone
% commit
```

Adding the `-d` option to `remove` will also delete the home directory.

4.2 Adding a User to K8s

To use K8s services, a user must also be added the K8s cluster.

Add each K8s user with `cm-kubernetes-setup`.

```
root@basepod-head1:~# cm-kubernetes-setup --add-user userone
Connecting to CMDaemon
Executing 10 stages
################### Starting execution for 'Kubernetes Setup'
  - kubernetes
  - docker
## Progress: 0
#### stage: kubernetes: Get Kube Cluster
## Progress: 10
#### stage: kubernetes: Check Permissions User Chart
## Progress: 20
#### stage: kubernetes: Check User
## Progress: 30
#### stage: kubernetes: Check Add User
## Progress: 40
#### stage: kubernetes: Check Namespace Does Not Exist
## Progress: 50
#### stage: kubernetes: Check Cluster Admin Has No Operators
## Progress: 60
#### stage: kubernetes: Deploy user
User userone created successfully!
## Progress: 70
#### stage: kubernetes: List Installed Operators
## Progress: 80
#### stage: kubernetes: Update Operator Permissions
## Progress: 90
#### stage: kubernetes: Log Text
User added successfully!
## Progress: 100

Took:     00:06 min.
Progress: 100/100
################### Finished execution for 'Kubernetes Setup', status: completed

Kubernetes Setup finished!
```

4.3 Removing a User from K8s

To remove a user (`userone`) from K8s, execute this command:

```
# cm-kubernetes-setup --remove-user userone
```

The user will no longer be able to use the K8s service.

If an attempt is made, this error message will be shown:

```
Error from server (Forbidden): nodes is forbidden: User "userone" cannot list
resource "nodes" in API group "" at the cluster scope
```

Appendix A. Site Survey

The tables in this section represent responses to a completed site survey and are used as examples in this deployment guide.

Table 4. General information

Item	Value
NFS server IP	10.130.122.252
NFS server export	/var/nfs/general
Head node drive path	nvme0n1
Cluster name	BasePOD
Organization	NVIDIA
Timezone	US/Los_Angeles
Nameservers	8.8.8.8
Search domains	example.com

Table 5. BCM head node information

Item	Value
Head node 1 name	basepod-head1
Head node 1 and 2 administrator password	ExamplePassword1234!@#$
Head node 1 BMC IP (ipminet)	10.130.111.66
Head node 1 Ethernet device (externalnet)	enp10
Head node 1 IP (externalnet)	10.130.121.254
Head node 1 Ethernet device (internalnet)	enp10
Head node 1 IP (internalnet)	10.130.122.254
Head node 2 name	basepod-head2
Head node 2 BMC IP (ipminet)	10.130.111.67
Head node 2 Ethernet device (externalnet)	enp10
Head node 2 IP (externalnet)	10.130.121.253
Head node 2 Ethernet device (internalnet)	enp10
Head node 2 IP (internalnet)	10.130.122.253

Table 6. Network information

Item	Value
K8s node name template	knode##
ipminet base IP	10.130.111.64
ipminet netmask	255.255.255.192
ipminet gateway	10.130.111.65
ipminet switch ASN	
internalnet switch #1 ASN (for externalnet)	
internalnet switch #2 ASN (for externalnet)	
externalnet base IP	10.130.121.0
externalnet netmask	255.255.255.0
externalnet gateway	10.130.121.1
Domain	example.com
internalnet base IP	10.130.122.0
internalnet netmask	255.255.255.0
ibnet base IP	10.149.0.0
ibnet netmask	255.255.0.0

Table 7. DGX node information

Item	Value
DGX Node 1 name	dgx01
DGX Node 1 MAC (enp225s0f0/Management)	B8:CE:F6:2F:08:69
DGX Node 1 MAC (enp97s0f0/Management)	B8:CE:F6:2D:0E:A7
DGX Node 2 name	dgx02
DGX Node 2 MAC (enp225s0f0/Management)	B8:CE:F6:2F:08:69
DGX Node 2 MAC (enp97s0f0/Management)	B8:CE:F6:2D:0E:A7
DGX Node 3 name	dgx03
DGX Node 3 MAC (enp225s0f0/Management)	B8:CE:F6:2F:08:69
DGX Node 3 MAC (enp97s0f0/Management)	B8:CE:F6:2D:0E:A7
DGX Node 4 name	dgx04
DGX Node 4 MAC (enp225s0f0/Management)	B8:CE:F6:2F:08:69
DGX Node 4 MAC (enp97s0f0/Management)	B8:CE:F6:2D:0E:A7

Table 8. K8s node information

Item	Value
K8S Node 1 interface 1 (Management)	ens1f1np1
K8S Node 1 interface 2 (Management)	ens2f1np1
K8S Node 1 interface 1 (Management) MAC	04:3F:72:E7:64:97
K8S Node 1 interface 2 (Management) MAC	0C:42:A1:79:9B:15
K8S Node 2 interface 1 (Management)	ens1f1np1
K8S Node 2 interface 2 (Management)	ens2f1np1
K8S Node 2 interface 1 (Management) MAC	04:3F:72:E7:64:97
K8S Node 2 interface 2 (Management) MAC	0C:42:A1:79:9B:15
K8S Node 3 interface 1 (Management)	ens1f1np1
K8S Node 3 interface 2 (Management)	ens2f1np1
K8S Node 3 interface 1 (Management) MAC	04:3F:72:E7:64:97
K8S Node 3 interface 2 (Management) MAC	0C:42:A1:79:9B:15

Appendix B. Switch Configurations

Switch configuration files are captured in this section.

B.1 SN4600 #1 (In-band Management Switch)

```
# Note Make sure to update the IP addresses in the sample config below
# eth0 mgmt interface configs
nv set interface eth0 ip address 10.130.111.78/24
nv set interface eth0 ip gateway 10.130.111.65
nv set interface eth0 ip vrf mgmt
nv set interface eth0 type eth

# Creating SVI interfaces and adding the VLANs to the bridge
# Note Make sure to update the IP addresses
nv set bridge domain br_default vlan 122
nv set bridge domain br_default vlan 121
nv set interface vlan121 type svi
nv set interface vlan121 ip vrr mac-address 44:38:39:ff:ff:ff
nv set interface vlan121 ip vrr address 10.130.121.1/24
nv set interface vlan121 ip address 10.130.121.2/24
nv set interface vlan122 type svi
nv set interface vlan122 ip vrr mac-address 44:38:39:ff:ff:ff
nv set interface vlan122 ip vrr address 10.130.122.1/24
nv set interface vlan122 ip address 10.130.122.2/24

# MLAG configs
nv set interface peerlink bond member swp57
nv set interface peerlink bond member swp58
nv set interface peerlink type peerlink
nv set interface peerlink.4094 base-interface peerlink
nv set interface peerlink.4094 type sub
nv set interface peerlink.4094 vlan 4094
nv set mlag backup 10.130.111.79 vrf mgmt
nv set mlag enable on
nv set mlag mac-address 44:38:39:ff:ff:ff
nv set mlag peer-ip linklocal
nv set mlag priority 2048

### bright headnode 01
nv set interface swp1 bridge domain br_default access 121
### bright headnode 02
nv set interface swp2 bridge domain br_default access 121

### bond4-20 used to connect to DGX and Kubernetes master nodes
nv set interface bond4 bond member swp4
nv set interface bond4 bond mlag id 4
nv set interface bond4 bridge domain br_default untagged 122
nv set interface bond4 bridge domain br_default vlan all
nv set interface bond4 bond mlag enable on
nv set interface bond4 bond lacp-bypass on
nv set interface bond5 bond member swp5
nv set interface bond5 bond mlag id 5
nv set interface bond5 bridge domain br_default untagged 122
```

```
nv set interface bond5 bridge domain br_default vlan all
nv set interface bond5 bond mlag enable on
nv set interface bond5 bond lacp-bypass on
nv set interface bond6 bond member swp6
nv set interface bond6 bond mlag id 6
nv set interface bond6 bridge domain br_default untagged 122
nv set interface bond6 bridge domain br_default vlan all
nv set interface bond6 bond mlag enable on
nv set interface bond6 bond lacp-bypass on
nv set interface bond7 bond member swp7
nv set interface bond7 bond mlag id 7
nv set interface bond7 bridge domain br_default untagged 122
nv set interface bond7 bridge domain br_default vlan all
nv set interface bond7 bond mlag enable on
nv set interface bond7 bond lacp-bypass on
nv set interface bond8 bond member swp8
nv set interface bond8 bond mlag id 8
nv set interface bond8 bridge domain br_default untagged 122
nv set interface bond8 bridge domain br_default vlan all
nv set interface bond8 bond mlag enable on
nv set interface bond8 bond lacp-bypass on
nv set interface bond9 bond member swp9
nv set interface bond9 bond mlag id 9
nv set interface bond9 bridge domain br_default untagged 122
nv set interface bond9 bridge domain br_default vlan all
nv set interface bond9 bond mlag enable on
nv set interface bond9 bond lacp-bypass on
nv set interface bond10 bond member swp10
nv set interface bond10 bond mlag id 10
nv set interface bond10 bridge domain br_default untagged 122
nv set interface bond10 bridge domain br_default vlan all
nv set interface bond10 bond mlag enable on
nv set interface bond10 bond lacp-bypass on
nv set interface bond11 bond member swp11
nv set interface bond11 bond mlag id 11
nv set interface bond11 bridge domain br_default untagged 122
nv set interface bond11 bridge domain br_default vlan all
nv set interface bond11 bond mlag enable on
nv set interface bond11 bond lacp-bypass on
nv set interface bond12 bond member swp12
nv set interface bond12 bond mlag id 12
nv set interface bond12 bridge domain br_default untagged 122
nv set interface bond12 bridge domain br_default vlan all
nv set interface bond12 bond mlag enable on
nv set interface bond12 bond lacp-bypass on
nv set interface bond13 bond member swp13
nv set interface bond13 bond mlag id 13
nv set interface bond13 bridge domain br_default untagged 122
nv set interface bond13 bridge domain br_default vlan all
nv set interface bond13 bond mlag enable on
nv set interface bond13 bond lacp-bypass on
nv set interface bond14 bond member swp14
nv set interface bond14 bond mlag id 14
nv set interface bond14 bridge domain br_default untagged 122
nv set interface bond14 bridge domain br_default vlan all
nv set interface bond14 bond mlag enable on
nv set interface bond14 bond lacp-bypass on
```

```
nv set interface bond15 bond member swp15
nv set interface bond15 bond mlag id 15
nv set interface bond15 bridge domain br_default untagged 122
nv set interface bond15 bridge domain br_default vlan all
nv set interface bond15 bond mlag enable on
nv set interface bond15 bond lacp-bypass on
nv set interface bond16 bond member swp16
nv set interface bond16 bond mlag id 16
nv set interface bond16 bridge domain br_default untagged 122
nv set interface bond16 bridge domain br_default vlan all
nv set interface bond16 bond mlag enable on
nv set interface bond16 bond lacp-bypass on
nv set interface bond17 bond member swp17
nv set interface bond17 bond mlag id 17
nv set interface bond17 bridge domain br_default untagged 122
nv set interface bond17 bridge domain br_default vlan all
nv set interface bond17 bond mlag enable on
nv set interface bond17 bond lacp-bypass on
nv set interface bond18 bond member swp18
nv set interface bond18 bond mlag id 18
nv set interface bond18 bridge domain br_default untagged 122
nv set interface bond18 bridge domain br_default vlan all
nv set interface bond18 bond mlag enable on
nv set interface bond18 bond lacp-bypass on
nv set interface bond19 bond member swp19
nv set interface bond19 bond mlag id 19
nv set interface bond19 bridge domain br_default untagged 122
nv set interface bond19 bridge domain br_default vlan all
nv set interface bond19 bond mlag enable on
nv set interface bond19 bond lacp-bypass on
nv set interface bond20 bond member swp20
nv set interface bond20 bond mlag id 20
nv set interface bond20 bridge domain br_default untagged 122
nv set interface bond20 bridge domain br_default vlan all
nv set interface bond20 bond mlag enable on
nv set interface bond20 bond lacp-bypass on

### BGP unnumbered configuration (NOTE: no IPs need to be configured on the BGP
interfaces, when using BGP unnumbered)
nv set router bgp autonomous-system 4200000003
nv set router bgp enable on
nv set router bgp router-id 10.130.111.78
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected enable
on
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor swp64 remote-as external
nv set vrf default router bgp neighbor swp64 type unnumbered
nv set vrf default router bgp neighbor swp63 remote-as external
nv set vrf default router bgp neighbor swp63 type unnumbered
nv set vrf default router bgp neighbor swp60 remote-as external
nv set vrf default router bgp neighbor swp60 type unnumbered
nv set vrf default router bgp neighbor peerlink.4094 remote-as internal
nv set vrf default router bgp neighbor peerlink.4094 type unnumbered
```

B.2 SN4600 #2 (In-band Management Switch)

```
# Note Make sure to update the IP addresses in the sample config below
# eth0 mgmt interface configs
nv set interface eth0 ip address 10.130.111.79/24
nv set interface eth0 ip gateway 10.130.111.65
nv set interface eth0 ip vrf mgmt
nv set interface eth0 type eth

# Creating SVI interfaces and adding the VLANs to the bridge
# Note Make sure to update the IP addresses
nv set bridge domain br_default vlan 122
nv set bridge domain br_default vlan 121
nv set interface vlan121 type svi
nv set interface vlan121 ip vrr mac-address 44:38:39:ff:ff:ff
nv set interface vlan121 ip vrr address 10.130.121.1/24
nv set interface vlan121 ip address 10.130.121.3/24
nv set interface vlan122 type svi
nv set interface vlan122 ip vrr mac-address 44:38:39:ff:ff:ff
nv set interface vlan122 ip vrr address 10.130.122.1/24
nv set interface vlan122 ip address 10.130.122.3/24

# MLAG configs
nv set interface peerlink bond member swp57
nv set interface peerlink bond member swp58
nv set interface peerlink type peerlink
nv set interface peerlink.4094 base-interface peerlink
nv set interface peerlink.4094 type sub
nv set interface peerlink.4094 vlan 4094
nv set mlag backup 10.130.111.78 vrf mgmt
nv set mlag enable on
nv set mlag mac-address 44:38:39:ff:ff:ff
nv set mlag peer-ip linklocal

### bright headnode 01
nv set interface swp1 bridge domain br_default access 122
### bright headnode 02
nv set interface swp2 bridge domain br_default access 122

### bond4-20 used to connect to DGX and Kubernetes master nodes
nv set interface bond4 bond member swp4
nv set interface bond4 bond mlag id 4
nv set interface bond4 bridge domain br_default untagged 122
nv set interface bond4 bridge domain br_default vlan all
nv set interface bond4 bond mlag enable on
nv set interface bond4 bond lacp-bypass on
nv set interface bond5 bond member swp5
nv set interface bond5 bond mlag id 5
nv set interface bond5 bridge domain br_default untagged 122
nv set interface bond5 bridge domain br_default vlan all
nv set interface bond5 bond mlag enable on
nv set interface bond5 bond lacp-bypass on
nv set interface bond6 bond member swp6
nv set interface bond6 bond mlag id 6
nv set interface bond6 bridge domain br_default untagged 122
nv set interface bond6 bridge domain br_default vlan all
```

```
nv set interface bond6 bond mlag enable on
nv set interface bond6 bond lacp-bypass on
nv set interface bond7 bond member swp7
nv set interface bond7 bond mlag id 7
nv set interface bond7 bridge domain br_default untagged 122
nv set interface bond7 bridge domain br_default vlan all
nv set interface bond7 bond mlag enable on
nv set interface bond7 bond lacp-bypass on
nv set interface bond8 bond member swp8
nv set interface bond8 bond mlag id 8
nv set interface bond8 bridge domain br_default untagged 122
nv set interface bond8 bridge domain br_default vlan all
nv set interface bond8 bond mlag enable on
nv set interface bond8 bond lacp-bypass on
nv set interface bond9 bond member swp9
nv set interface bond9 bond mlag id 9
nv set interface bond9 bridge domain br_default untagged 122
nv set interface bond9 bridge domain br_default vlan all
nv set interface bond9 bond mlag enable on
nv set interface bond9 bond lacp-bypass on
nv set interface bond10 bond member swp10
nv set interface bond10 bond mlag id 10
nv set interface bond10 bridge domain br_default untagged 122
nv set interface bond10 bridge domain br_default vlan all
nv set interface bond10 bond mlag enable on
nv set interface bond10 bond lacp-bypass on
nv set interface bond11 bond member swp11
nv set interface bond11 bond mlag id 11
nv set interface bond11 bridge domain br_default untagged 122
nv set interface bond11 bridge domain br_default vlan all
nv set interface bond11 bond mlag enable on
nv set interface bond11 bond lacp-bypass on
nv set interface bond12 bond member swp12
nv set interface bond12 bond mlag id 12
nv set interface bond12 bridge domain br_default untagged 122
nv set interface bond12 bridge domain br_default vlan all
nv set interface bond12 bond mlag enable on
nv set interface bond12 bond lacp-bypass on
nv set interface bond13 bond member swp13
nv set interface bond13 bond mlag id 13
nv set interface bond13 bridge domain br_default untagged 122
nv set interface bond13 bridge domain br_default vlan all
nv set interface bond13 bond mlag enable on
nv set interface bond13 bond lacp-bypass on
nv set interface bond14 bond member swp14
nv set interface bond14 bond mlag id 14
nv set interface bond14 bridge domain br_default untagged 122
nv set interface bond14 bridge domain br_default vlan all
nv set interface bond14 bond mlag enable on
nv set interface bond14 bond lacp-bypass on
nv set interface bond15 bond member swp15
nv set interface bond15 bond mlag id 15
nv set interface bond15 bridge domain br_default untagged 122
nv set interface bond15 bridge domain br_default vlan all
nv set interface bond15 bond mlag enable on
nv set interface bond15 bond lacp-bypass on
nv set interface bond16 bond member swp16
```

```
nv set interface bond16 bond mlag id 16
nv set interface bond16 bridge domain br_default untagged 122
nv set interface bond16 bridge domain br_default vlan all
nv set interface bond16 bond mlag enable on
nv set interface bond16 bond lacp-bypass on
nv set interface bond17 bond member swp17
nv set interface bond17 bond mlag id 17
nv set interface bond17 bridge domain br_default untagged 122
nv set interface bond17 bridge domain br_default vlan all
nv set interface bond17 bond mlag enable on
nv set interface bond17 bond lacp-bypass on
nv set interface bond18 bond member swp18
nv set interface bond18 bond mlag id 18
nv set interface bond18 bridge domain br_default untagged 122
nv set interface bond18 bridge domain br_default vlan all
nv set interface bond18 bond mlag enable on
nv set interface bond18 bond lacp-bypass on
nv set interface bond19 bond member swp19
nv set interface bond19 bond mlag id 19
nv set interface bond19 bridge domain br_default untagged 122
nv set interface bond19 bridge domain br_default vlan all
nv set interface bond19 bond mlag enable on
nv set interface bond19 bond lacp-bypass on
nv set interface bond20 bond member swp20
nv set interface bond20 bond mlag id 20
nv set interface bond20 bridge domain br_default untagged 122
nv set interface bond20 bridge domain br_default vlan all
nv set interface bond20 bond mlag enable on
nv set interface bond20 bond lacp-bypass on

### BGP unnumbered configuration (NOTE: no IPs need to be configured on the BGP
interfaces, when using BGP unnumbered)
nv set router bgp autonomous-system 4200000003
nv set router bgp enable on
nv set router bgp router-id 10.130.111.79
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected enable
on
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor swp63 remote-as external
nv set vrf default router bgp neighbor swp63 type unnumbered
nv set vrf default router bgp neighbor swp64 remote-as external
nv set vrf default router bgp neighbor swp64 type unnumbered
nv set vrf default router bgp neighbor swp60 remote-as external
nv set vrf default router bgp neighbor swp60 type unnumbered
nv set vrf default router bgp neighbor peerlink.4094 remote-as internal
nv set vrf default router bgp neighbor peerlink.4094 type unnumbered
```

B.3 SN2201 (Out-of-band Management Switch)

```
#eth0
nv set interface eth0 ip address 10.130.111.77/24
nv set interface eth0 ip gateway 10.130.111.65
nv set interface eth0 ip vrf mgmt
nv set interface eth0 type eth

# Creating SVI interfaces and adding the VLANs to the bridge
# Note Make sure to update the IP addresses
nv set interface vlan111 ip address 10.130.111.65/26
nv set bridge domain br_default vlan 111

### BGP configurations
nv set router bgp autonomous-system 4200000004
nv set router bgp enable on
nv set router bgp router-id 10.130.111.77
nv set vrf default router bgp address-family ipv4-unicast enable on
nv set vrf default router bgp address-family ipv4-unicast redistribute connected enable
on
nv set vrf default router bgp enable on
nv set vrf default router bgp neighbor swp51 remote-as external
nv set vrf default router bgp neighbor swp51 type unnumbered
nv set vrf default router bgp neighbor swp52 remote-as external
nv set vrf default router bgp neighbor swp52 type unnumbered

# interfaces connected to IPMI interfaces of different servers
nv set interface swp1-40 bridge domain br_default access 111
```

B.4 Ethernet Network Configuration Verifications

Some of the lines in the output have been truncated for readability.

```
### MLAG verifications:
root@TOR-01:mgmt:~# net show clag
The peer is alive
     Our Priority, ID, and Role: 2048 48:b0:2d:cc:b2:bc primary
    Peer Priority, ID, and Role: 32768 48:b0:2d:ca:93:7b secondary
          Peer Interface and IP: peerlink.4094 fe80::4ab0:2dff:feca:937b (linklocal)
                      Backup IP: 10.130.111.79 vrf mgmt (active)
                     System MAC: 44:38:39:ff:ff:ff
root@TOR-01:mgmt:~#
CLAG Interfaces
Our Interface   Peer Interface  CLAG Id   Conflicts        Proto-Down Reason
-------------   --------------  -------   --------------   -----------------
       bond10   -                 10        -                      -
       bond11   -                 11        -                      -
       bond12   -                 12        -                      -
       bond13   bond13           13        -                      -
```

```
       bond14   -                14        -                     -
       bond15   -                15        -                     -
       bond16   bond16           16        -                     -
       bond17   -                17        -                     -
       bond18   -                18        -                     -
       bond19   -                19        -                     -
       bond20   -                20
       bond4    bond4            4         -                     -
       bond5    -                5         -                     -
       bond6    bond6            6         -                     -
       bond7    -                7         -                     -
       bond8    -                8         -                     -
       bond9    -                9         -                     -

### verifying an access port (below is a access port with VLAN set to 111)

network-admin@IPMI-01:mgmt:~$ net show int swp1
    Name  MAC                Speed  MTU   Mode
--  ----  -----------------  -----  ----  ---------
UP  swp1  68:21:5f:4f:14:81  1G     1500  Access/L2

Alias
-----
bcm-bootstrap:eth0

All VLANs on L2 Port
--------------------
111

Untagged
--------
111

### verifying a bonded interface in trunk mode (note the native VLAN is set to 122)
network-admin@TOR-01:mgmt:~$ net show int bond9
    Name    MAC                Speed  MTU   Mode
--  ------  -----------------  -----  ----  -------
UP  bond9  1c:34:da:29:17:54  100G   9216  802.3ad

Bond Details
------------------  --------
Bond Mode:          802.3ad
Load Balancing:     layer3+4
Minimum Links:      1
LACP Sys Priority:
LACP Rate:          1
LACP Bypass:        Active

All VLANs on L2 Port
--------------------
1,121,122

Untagged
--------
122

#### BGP verifications
```

```
cumulus@TOR-01:mgmt:~$ net show bgp summary
show bgp ipv4 unicast summary
============================
BGP router identifier 10.130.111.78, local AS number 4200000003 vrf-id 0
BGP table version 3
RIB entries 5, using 1000 bytes of memory
Peers 4, using 91 KiB of memory

Neighbor              V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down
State/PfxRcd   PfxSnt
TOR-02(peerlink.4094)  4 4200000003      2705      2706        0    0    0 02:15:00
Spine-02(swp64)        4 4200000002      2655      2656        0    0    0 02:12:29
Spine-01(swp63)        4 4200000001      2753      2754        0    0    0 02:17:22
IPMI-01(swp60)         4 4200000004      2480      2482        0    0    0 02:03:48

Total number of neighbors 4


cumulus@TOR-02:mgmt:~$ net show bgp summary
show bgp ipv4 unicast summary
============================
BGP router identifier 10.130.111.79, local AS number 4200000003 vrf-id 0
BGP table version 3
RIB entries 5, using 1000 bytes of memory
Peers 4, using 91 KiB of memory

Neighbor              V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down
State/PfxRcd   PfxSnt
TOR-01(peerlink.4094) 4 4200000003      2692      2692        0    0    0 02:14:20
Spine-01(swp63)        4 4200000001      2692      2692        0    0    0 02:14:21
Spine-02(swp64)        4 4200000002      2641      2642        0    0    0 02:11:48
IPMI-01(swp60)         4 4200000004      2467      2469        0    0    0 02:03:07

Total number of neighbors 4

cumulus@IPMI-01:mgmt:~$ net show bgp summary
show bgp ipv4 unicast summary
============================
BGP router identifier 10.130.111.77, local AS number 4200000004 vrf-id 0
BGP table version 3
RIB entries 5, using 1000 bytes of memory
Peers 2, using 46 KiB of memory

Neighbor        V         AS   MsgRcvd   MsgSent   TblVer  InQ OutQ  Up/Down State/PfxRcd
PfxSnt
TOR-01(swp51)    4 4200000003      2495      2494        0    0    0 02:04:30
TOR-02(swp52)   4 4200000003      2495      2494        0    0    0 02:04:30

Total number of neighbors 2
```

Notice

This document is provided for information purposes only and shall not be regarded as a warranty of a certain functionality, condition, or quality of a product. NVIDIA Corporation (“NVIDIA”) makes no representations or warranties, expressed or implied, as to the accuracy or completeness of the information contained in this document and assumes no responsibility for any errors contained herein. NVIDIA shall have no liability for the consequences or use of such information or for any infringement of patents or other rights of third parties that may result from its use. This document is not a commitment to develop, release, or deliver any Material (defined below), code, or functionality.

NVIDIA reserves the right to make corrections, modifications, enhancements, improvements, and any other changes to this document, at any time without notice.

Customer should obtain the latest relevant information before placing orders and should verify that such information is current and complete.

NVIDIA products are sold subject to the NVIDIA standard terms and conditions of sale supplied at the time of order acknowledgement, unless otherwise agreed in an individual sales agreement signed by authorized representatives of NVIDIA and customer (“Terms of Sale”). NVIDIA hereby expressly objects to applying any customer general terms and conditions with regards to the purchase of the NVIDIA product referenced in this document. No contractual obligations are formed either directly or indirectly by this document.

No license, either expressed or implied, is granted under any NVIDIA patent right, copyright, or other NVIDIA intellectual property right under this document. Information published by NVIDIA regarding third-party products or services does not constitute a license from NVIDIA to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property rights of the third party, or a license from NVIDIA under the patents or other intellectual property rights of NVIDIA.

Reproduction of information in this document is permissible only if approved in advance by NVIDIA in writing, reproduced without alteration and in full compliance with all applicable export laws and regulations, and accompanied by all associated conditions, limitations, and notices.

THIS DOCUMENT AND ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, “MATERIALS”) ARE BEING PROVIDED “AS IS.” NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL NVIDIA BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF NVIDIA HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA’s aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the Terms of Sale for the product.

Trademarks

NVIDIA, the NVIDIA logo, DGX, DGX NVIDIA BasePOD, and NVIDIA Base Command Manager are trademarks and/or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

NVIDIA Corporation | 2788 San Tomas Expressway, Santa Clara, CA 95051
http://www.nvidia.com