NVIDIA DGX BasePOD: Multi-cloud Architecture with Amazon Web Services

Deployment Guide

Featuring NVIDIA DGX BasePOD and NVIDIA GPUs through Amazon Web Services

DU-11343-001 | April 2023

Document History

DU-11343-001

Version	Date	Authors	Description of Change
01	2023-04-07	Joe Handzik, Robert Sohigian, Jack Curtin, Fred Herard, and Zenobia Redeaux	Initial release
02	2023-04-19	Joe Handzik, Robert Sohigian, Yang Yang, and Chris Kawalek	Procedural and marketing updates

Abstract

As part of the NVIDIA DGX™ platform, NVIDIA DGX BasePOD™ provides on-premises infrastructure for artificial intelligence (AI) workloads. This infrastructure is an excellent fit for stable use cases and resource requirements.

However, demands can sometimes outstrip resource availability or users might need access to different resources than those provided by their DGX infrastructure.

Managing a separate pool of resources to support changing requirements typically involves the development of significant expertise in cloud management tools and interfaces. A separate pool of resources often requires user education to request the appropriate system or environment—leading to suboptimal resource utilization and user confusion.

Those scenarios are now resolved through the capabilities of NVIDIA Base Command™ Manager (BCM) software. Administrators can now integrate on-demand public cloud resources directly with an on-premises DGX BasePOD private cloud environment and make the combined resources available transparently in a multi-cloud architecture.

This document describes how to extend DGX BasePOD with additional NVIDIA GPUs from Amazon Web Services (AWS) and manage the entire infrastructure from a consolidated user interface. Given the breadth of instances offered by AWS for both general-purpose and accelerated computing with NVIDIA GPUs, it is a great option for use as the basis of cloud resource integration in BCM.

Providing concordant access to on-premises and public cloud resources through existing infrastructure drastically simplifies both the administrator and user experience and makes using the right tool for any job easy.

Contents

Chapter 1. Introduction

Deployment of public cloud integration into DGX BasePOD should be done after on-premises components and services have been deployed, according to the *NVIDIA DGX BasePOD Deployment Guide*. The tool within BCM that enables this integration is Cluster Extension (`cm-cluster-extension`). It allows an administrator to integrate a public cloud provider account into an on-premises deployment and configure what resources will be provisioned using that public cloud provider and what regions those resources will be provisioned in. The public cloud resources appear side-by-side with on-premises resources in administrator tools, with access to public cloud-specific configuration capabilities when necessary.

Chapter 2. softwareimage and Category Creation

Before configuring AWS using `cm-cluster-extension`, create the software images and non-director node categories that are necessary for the target public cloud environment.

1. `ssh` to the head node as root or a user capable of gaining root permissions.

   Specify the external network address or hostname of the head node to gain access.

   ```
   # ssh <head-node>
   ```

2. Enter the `cmsh` configuration shell.

```
# cmsh
```

> **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.

3. Enter the `softwareimage` menu and create three additional software images as clones of the `default-image`—one for each of the node types to be provisioned in the public cloud.

   ```
   % softwareimage
   % clone default-image cloud-director-image
   % clone default-image k8s-cloud-master-image
   % clone default-image k8s-cloud-gpu-worker-image
   % ..
   % commit
   ```

4. Enter the `category` menu and create categories for `k8s-cloud-master` and `k8s-cloud-gpu-worker`.

   The `cm-cluster-extension` tool automatically creates a category for the `cloud-director` (`softwareimage` is configured at a later step).

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

5. Augment the `disksetup` of the new categories as well.

   This guide was executed with a disk layout that maximized the root partition size to avoid scenarios where containers quickly fill a smaller partition. Save the following text to `/tmp/big-cloud-disk.xml`.

```
<diskSetup>
  <device>
    <blockdev>/dev/sda</blockdev>
    <blockdev>/dev/I</blockdev>
    <blockdev>/dev/vda</blockdev>
    <blockdev>/dev/xvda</blockdev>
    <blockdev>/dev/cciss/c0d0</blockdev>
    <blockdev>/dev/nvme0n1</blockdev>
    <blockdev mode="cloud">/dev/sdb</blockdev>
    <blockdev mode="cloud">/dev/hdb</blockdev>
    <blockdev mode="cloud">/dev/vdb</blockdev>
    <blockdev mode="cloud">/dev/xvdb</blockdev>
    <blockdev mode="cloud">/dev/xvdf</blockdev>
    <blockdev mode="cloud">/dev/nvme1n1</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>
    <partition id="a2">
      <size>12G</size>
      <type>linux swap</type>
    </partition>
  </device>
</diskSetup>
```

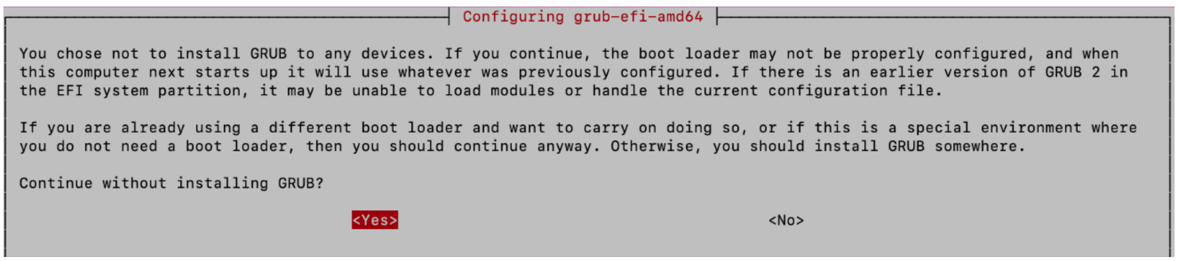
6. Assign the new disk layout file to the cloud categories in Step 4.

```
$ cmsh
% category
% use k8s-cloud-master
% set disksetup /tmp/big-cloud-disk.xml
% commit
% use k8s-cloud-gpu-worker
% set disksetup /tmp/big-cloud-disk.xml
% commit
```

7. Exit the `cmsh` configuration shell and update all three images.

```
# cm-chroot-sw-img /cm/images/k8s-cloud-master-image/
# apt update && apt -y upgrade
# exit
# cm-chroot-sw-img /cm/images/k8s-cloud-gpu-worker-image/
# apt update && apt -y upgrade
# exit
# cm-chroot-sw-img /cm/images/cloud-director-image/
# apt update && apt -y upgrade
# exit
```

8. When a terminal menu is displayed to confirm that GRUB does not need to be installed, select `Yes` to continue.



Chapter 3. Cluster Extension Configuration

With images and categories prepared, the environment is now ready for AWS integration and initial configuration. The AWS integration will be accomplished using the `cm-cluster-extension` command.

1. Create an AWS IAM group with an appropriate policy for a user account to integrate into the BCM on-premises head node.

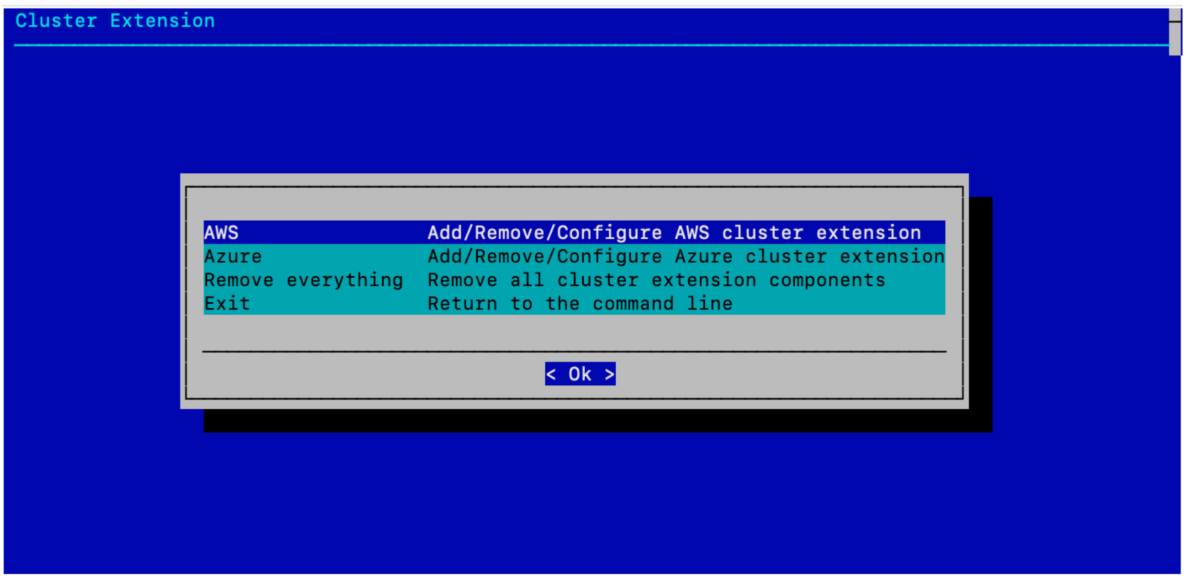
   To create a minimum viable policy set, refer to [this Bright Knowledge Base article]().

   Assign the policy to the target group and provision a new user in that group. Create a new access key and associated secret access key for that user for use with Bright. Securely document the access key and secret access key for use in this section.

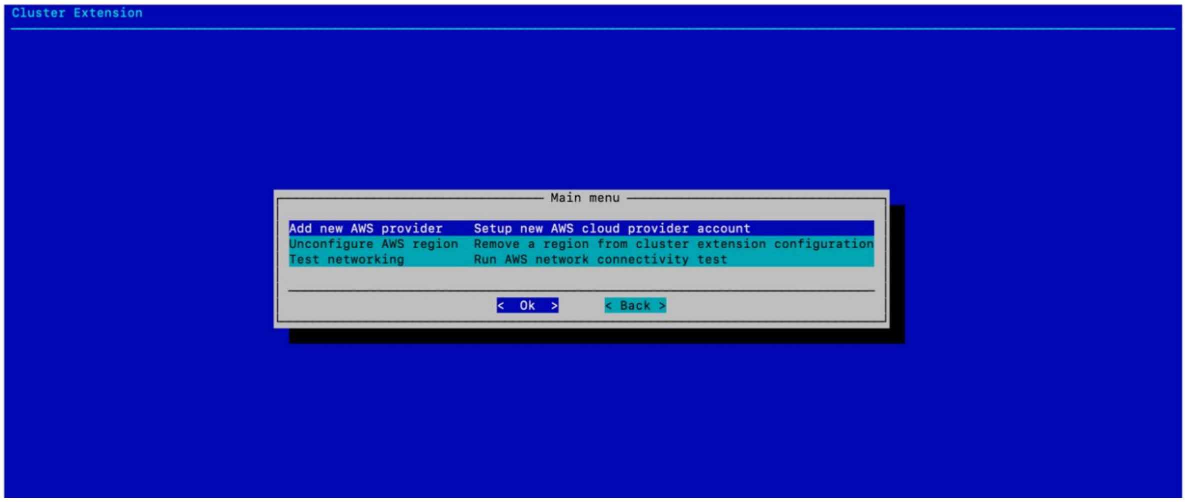
2. Run the `cm-cluster-extension` command to get started.

```
# cm-cluster-extension
```

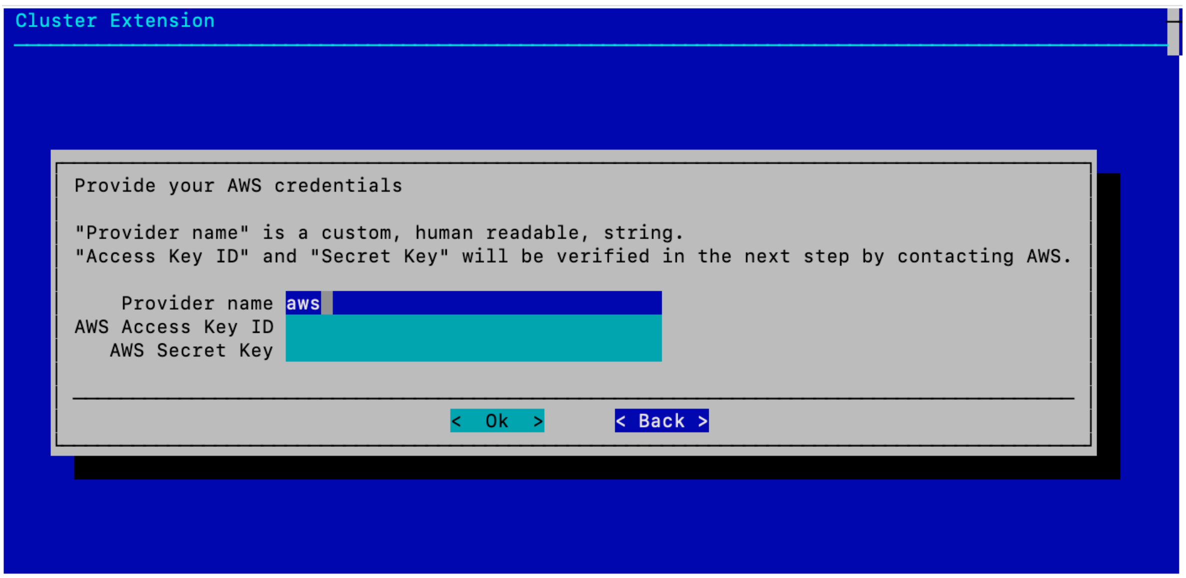
3. Choose the `AWS` extension and then select `Ok`.



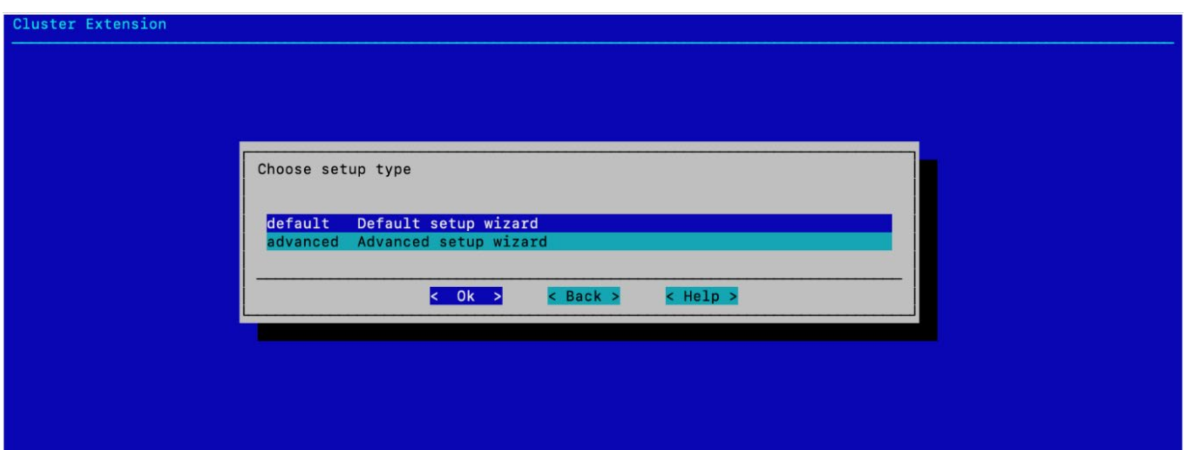
4. Choose `Add new AWS provider` and then select `Ok`.



5. Enter the required AWS credential information and then select `Ok`.

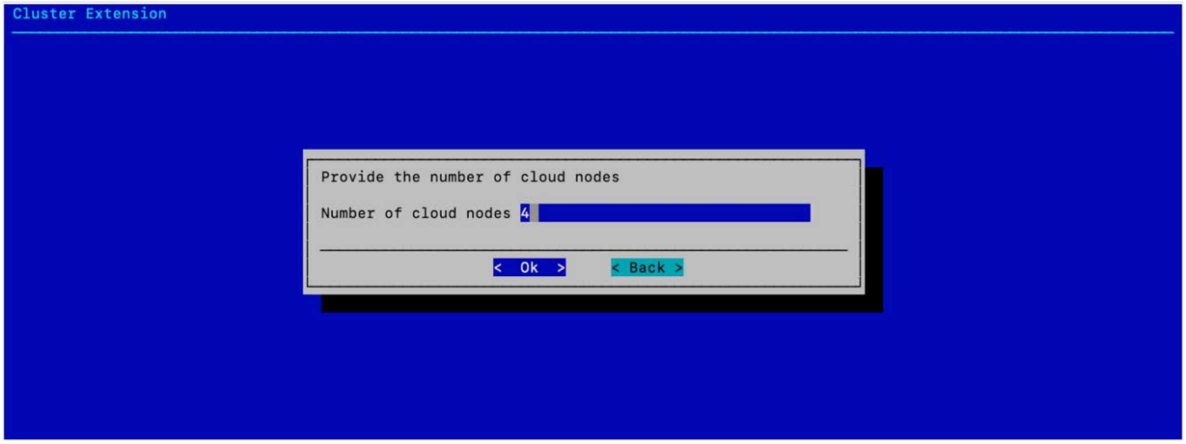


6. Add the provider to the new region by choosing the `default` setup type and then select `Ok`.



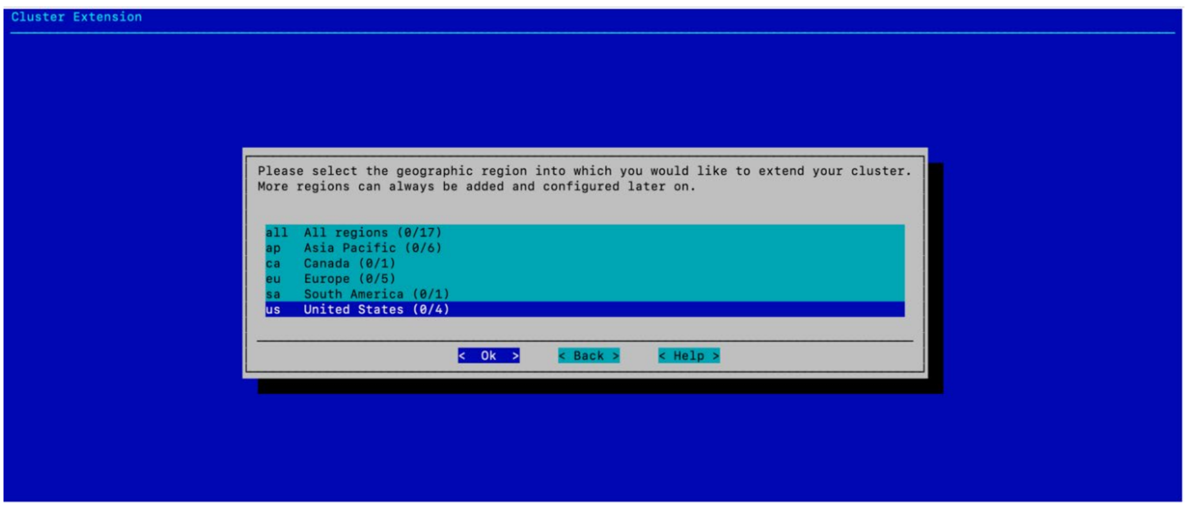
7. Enter `4` for the quantity of cloud nodes and then select `Ok`.

   There will be three nodes for the Kubernetes (K8s) control plane and one node as a GPU worker. More nodes can be added later.



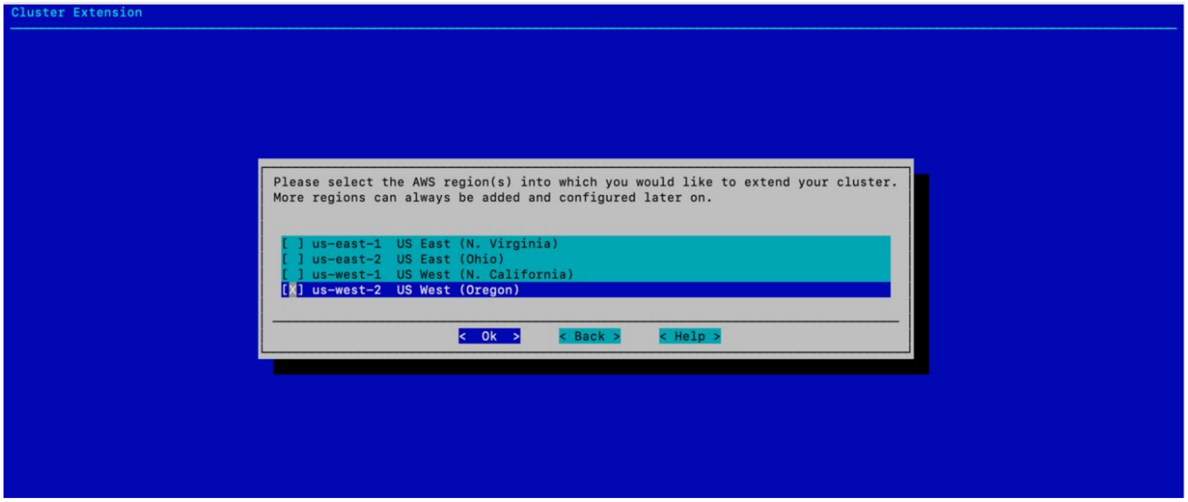
8. Choose the appropriate geographic region and then select `Ok`.

   Choosing a region near the on-premises cluster typically increases network performance. If the configuration is designed for regional fault tolerance, choose a more distant region. Because not all instances are available in all regions, the type of instance needed should also be considered.



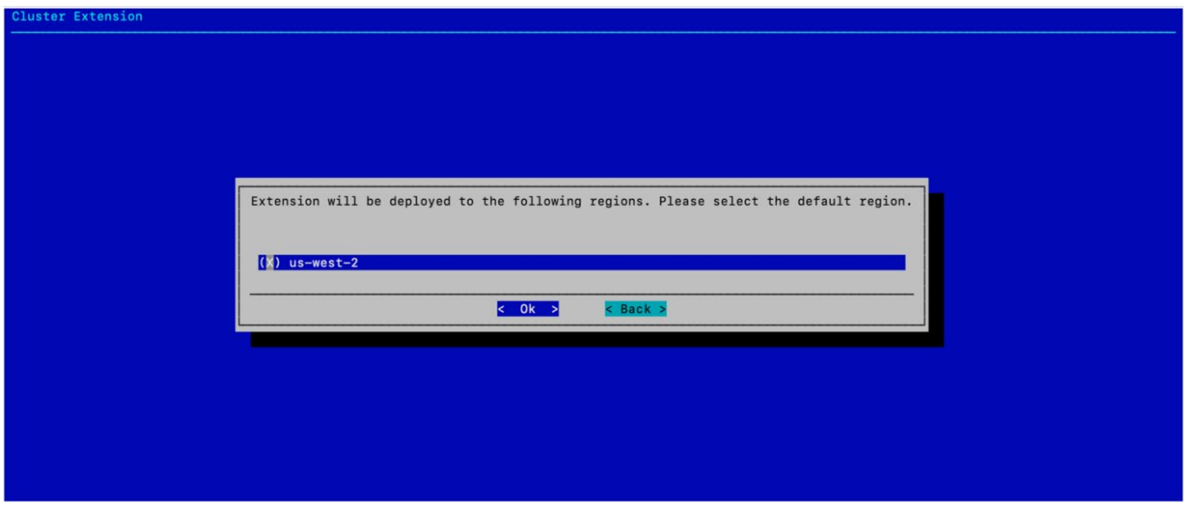
9. Choose a region in the subsequent screen and then select `Ok`.

   `us-west-2` is used in this example.



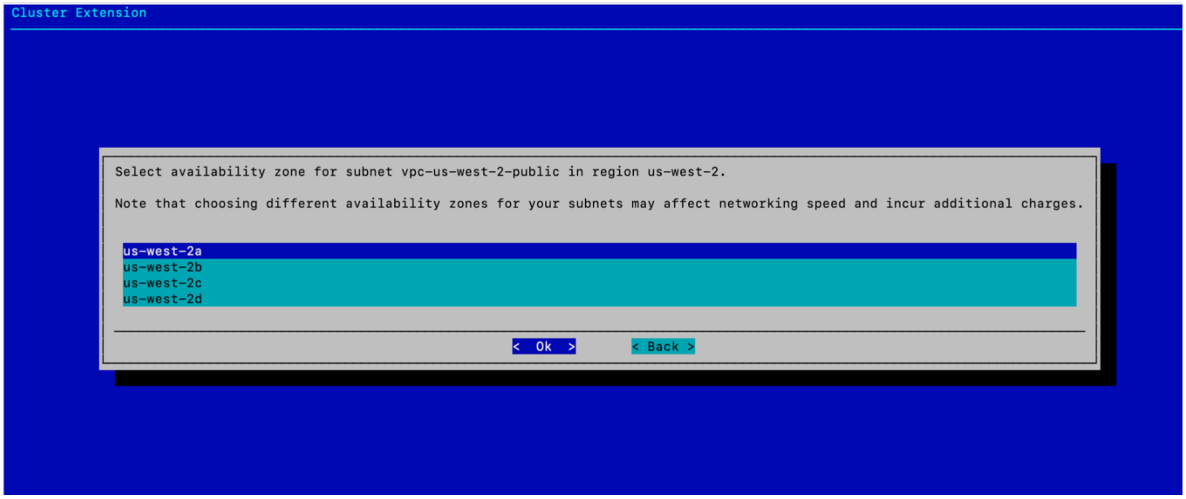
10. Choose a default region and then select `Ok`.

    In this example, the only option is `us-west-2` because no other regions were configured.



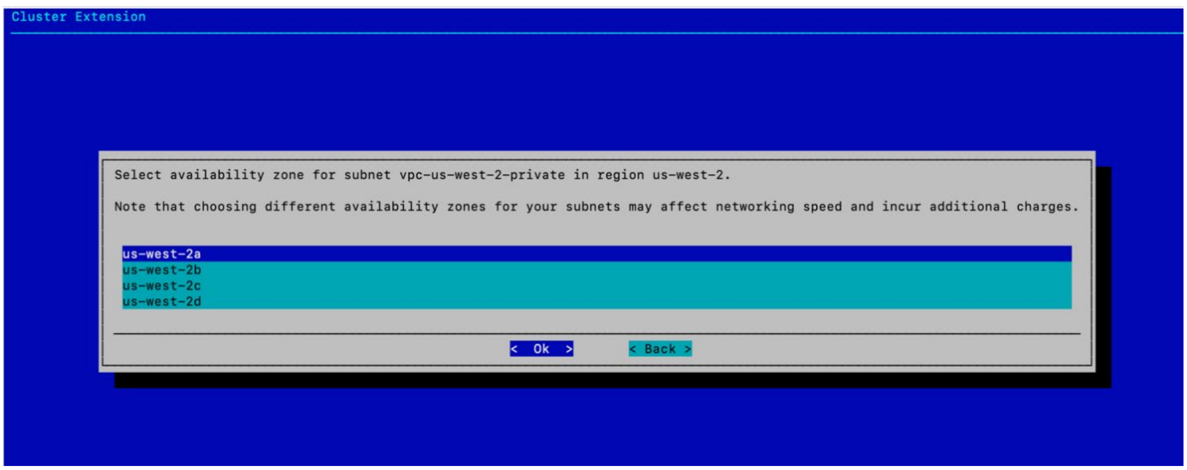
11. Choose an availability zone for the public subnet that Cluster Extension will create and then select `Ok`.

    `us-west-2a` was selected in this example.



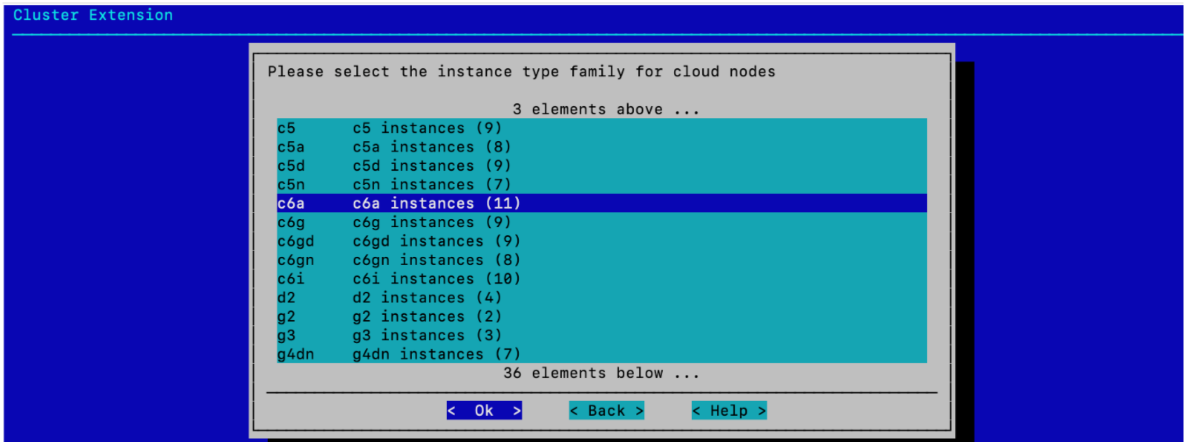
12. Choose an availability zone for the private subnet that Cluster Extension will create and then select `Ok`.

    `us-west-2a` was again selected.

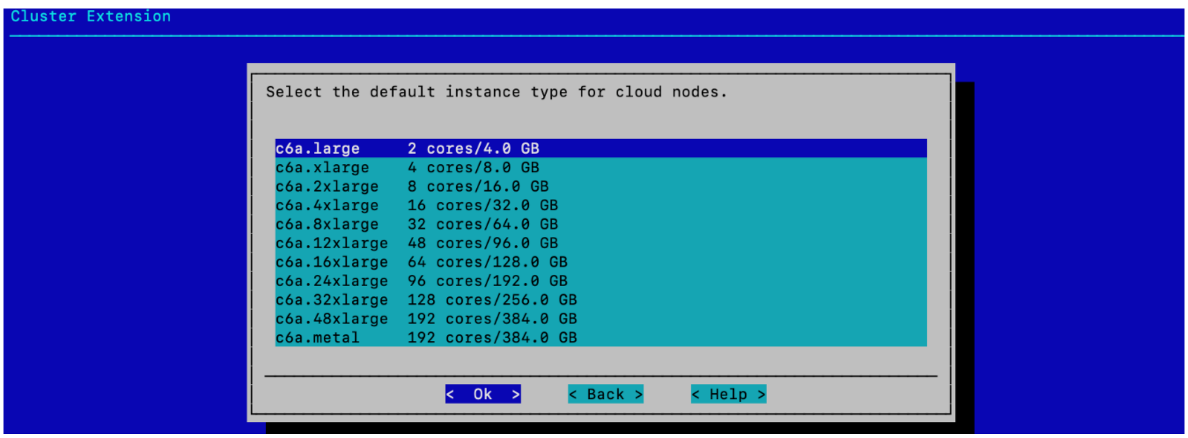


13. Choose `c6a` for `instance type family for cloud nodes` and then select `Ok`.

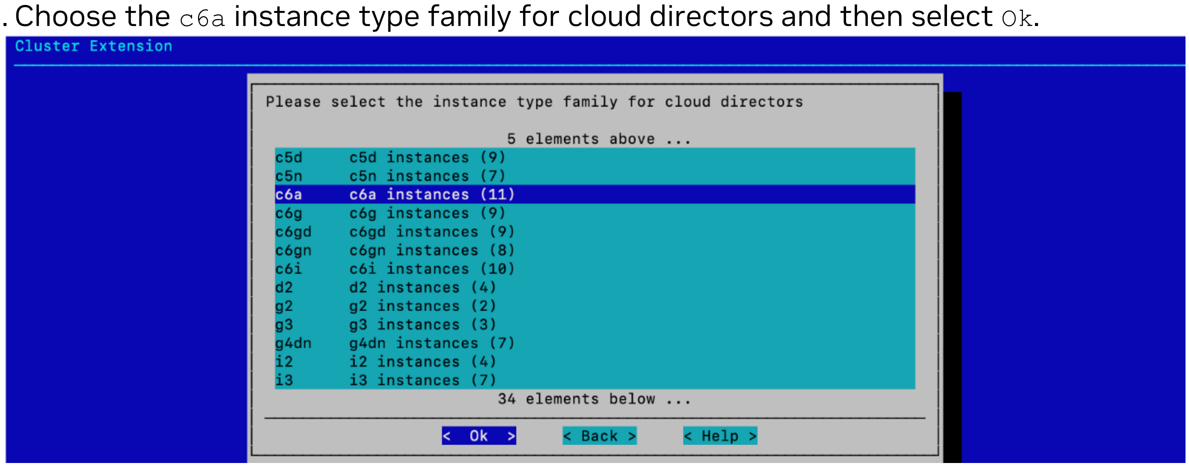
    `c6a` instances are widely available and provide good performance and value for this use case. At a later step, one of the preallocated public cloud nodes will be configured to use an instance type with NVIDIA GPUs.



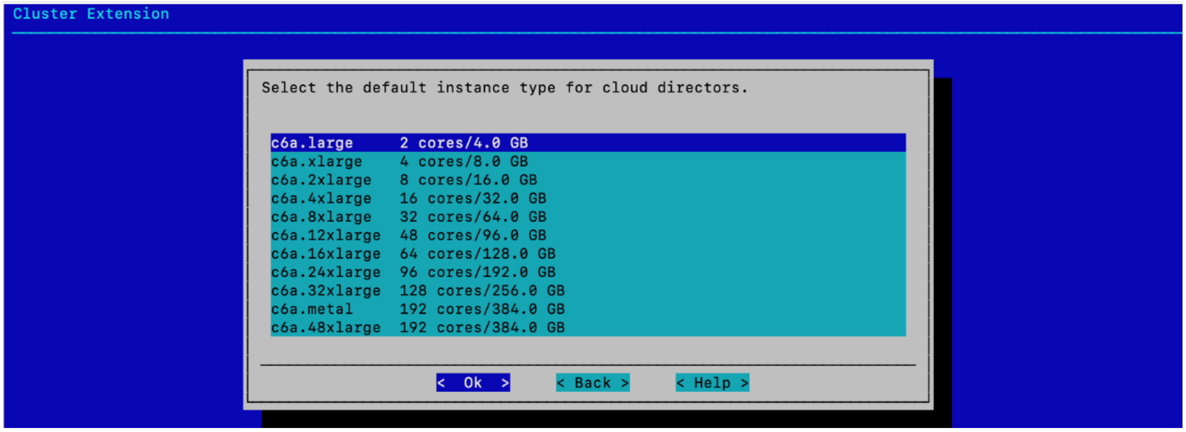
14. Choose `c6a.large` instances and then select `Ok`.



15. Choose the `c6a` instance type family for cloud directors and then select `Ok`.

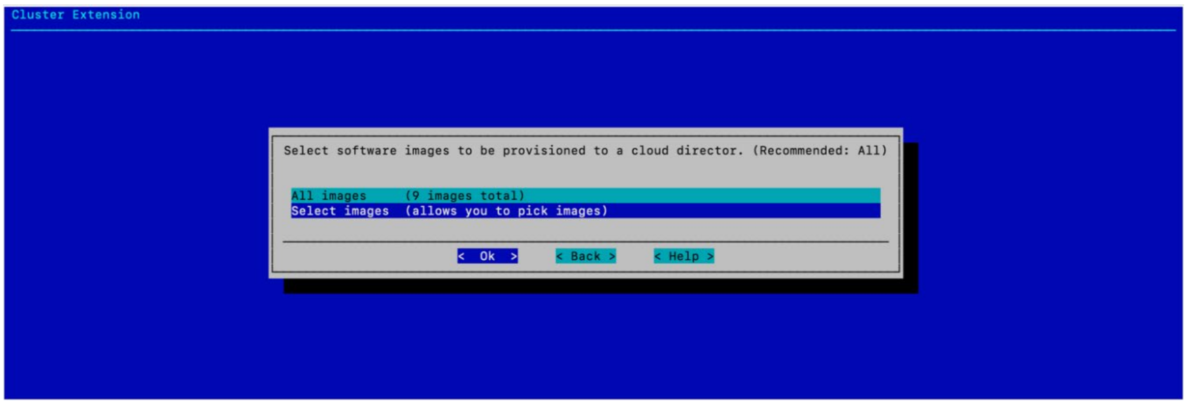


16. Choose the `c6a.large` instance type and then select `Ok`.



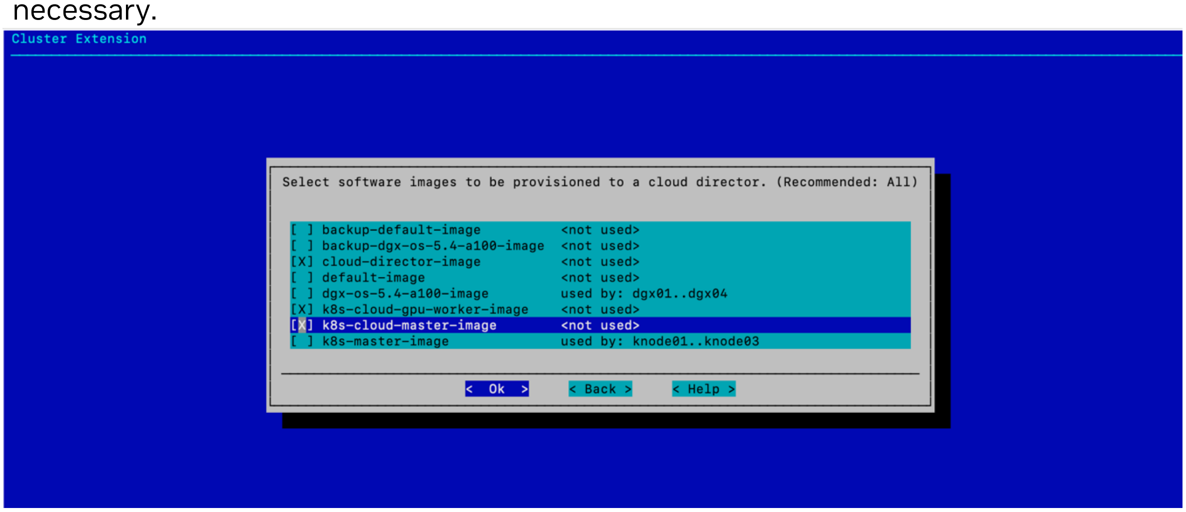
17. Choose `Select images` and then select `Ok`.

    This selects the subset of images that can be used in the public cloud and eliminates those that cannot be used (such as DGX OS).

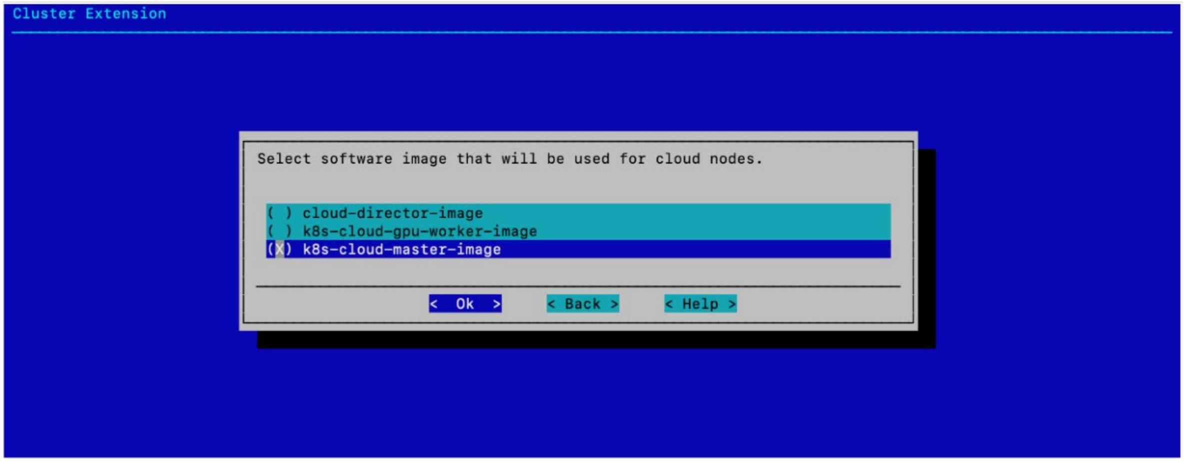


18. Choose the images that were created for this deployment and then select `Ok`.

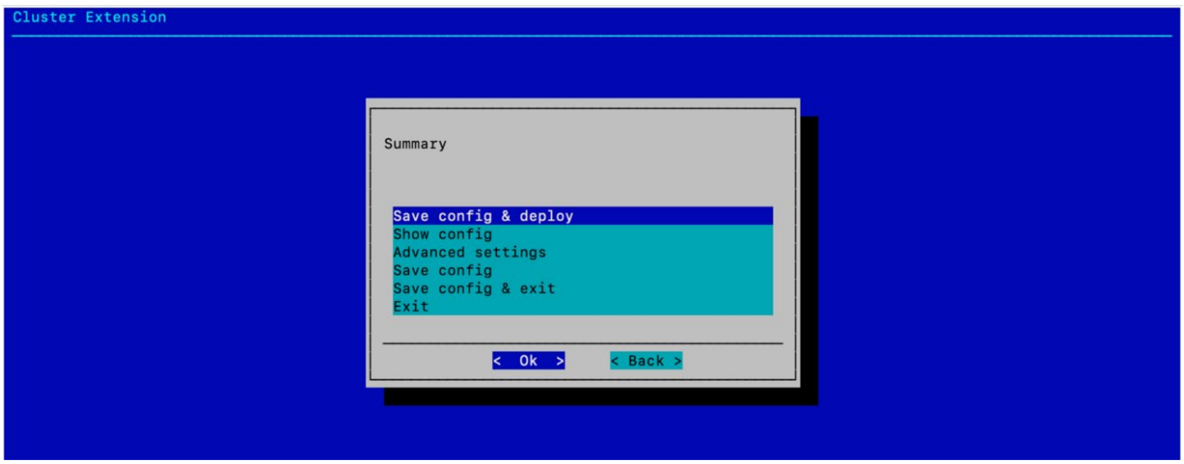
    `cloud-director-image`, `k8s-cloud-gpu-worker-image`, and `k8s-cloud-master-image` should be checked. Additional images can be added later if necessary.



19. Choose `k8s-cloud-master-image` for the default cloud node image and then select `Ok`.

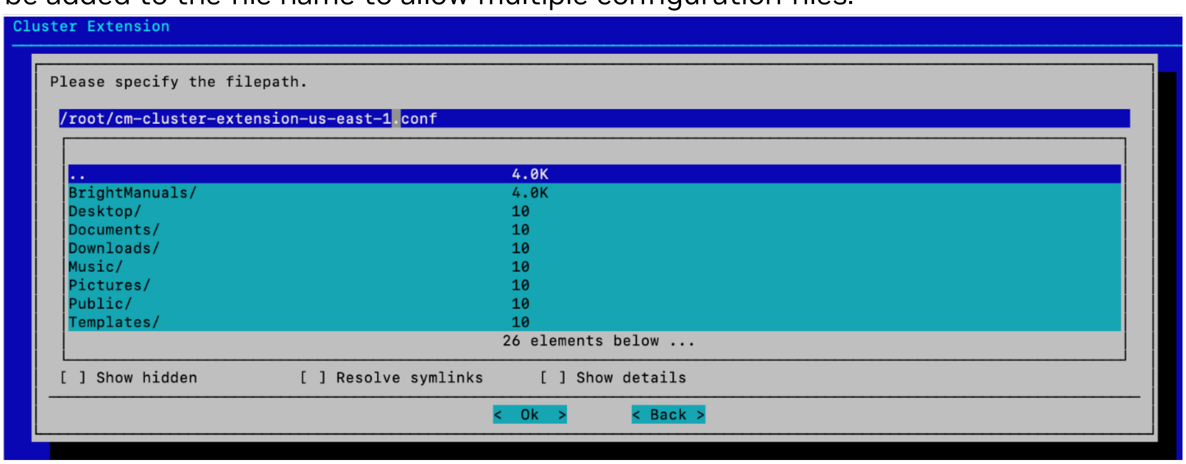


20. Choose `Save config & deploy` on the `Summary` screen and then select `Ok`.



21. Specify the filepath and then select `Ok`.

    A default filepath is displayed. A region name or other identifying information should be added to the file name to allow multiple configuration files.



22. The configuration begins executing on the BCM head node.

    When completed, output like the following should be displayed.

```
## Progress: 100

Took:     04:09 min.
Progress: 100/100
################### Finished execution for ‘Cluster Extension’, status: completed

Cluster Extension finished!
```

23. Verify that the initial setup was successful.

    Run `list -f` in `cmsh` as shown in the screenshot and compare it to the output provided—it should be similar (additional listed systems are redacted, and the exact IP subnet may be slightly different).

```
[hybridbasepod-b-primary->device]% list -f Type,Hostname,Category,Ip,Network,Status
Type       Hostname (key)      MAC                Category             Ip            Network    Status
---------  ------------------  -----------------  -------------------  ------------  ---------  ----------
CloudNode  us-west-2-cnode001  00:00:00:00:00:00  us-west-2-cloud-node  172.17.0.1     us-west-2  [  DOWN  ]
CloudNode  us-west-2-cnode002  00:00:00:00:00:00  us-west-2-cloud-node  172.17.0.2     us-west-2  [  DOWN  ]
CloudNode  us-west-2-cnode003  00:00:00:00:00:00  us-west-2-cloud-node  172.17.0.3     us-west-2  [  DOWN  ]
CloudNode  us-west-2-cnode004  00:00:00:00:00:00  us-west-2-cloud-node  172.17.0.4     us-west-2  [  DOWN  ]
CloudNode  us-west-2-director  00:00:00:00:00:00  aws-cloud-director    172.17.255.251 us-west-2  [  DOWN  ]
```

24. Augment the OpenVPN port if needed.

    The Cluster Extension functionality relies on OpenVPN to run a VPN tunnel between the on-premises head node and the targeted public cloud environment. The default configuration uses UDP port 1194. To configure a different protocol or port, refer to [this Bright Knowledge Base article](#).

Chapter 4. Host Preparation After Cluster Extension Configuration

With AWS cloud resource access configured, the next step is to modify the device entries created using the `cm-cluster-extension` to leverage the correct categories and make changes when necessary to public cloud settings.

1. Rename the nodes according to their expected usage as follows from the device sub-menu.

```
# cmsh
% device
% rename us-west-2-cnode001 us-west-2-knode001
% rename us-west-2-cnode002 us-west-2-knode002
% rename us-west-2-cnode003 us-west-2-knode003
% rename us-west-2-cnode004 us-west-2-gpu-node001
% commit
```

2. Update the categories for all four public cloud nodes.

```
% set us-west-2-knode001 category k8s-cloud-master
% set us-west-2-knode002 category k8s-cloud-master
% set us-west-2-knode003 category k8s-cloud-master
% set us-west-2-gpu-node001 category k8s-cloud-gpu-worker
% commit
```

3. Increase the EBS volume size to 100 GiB on the `knode` systems.

```
% cloudsettings us-west-2-knode001
% storage
% set ebs size 100GiB
% commit
% ..
% ..
% ..
% cloudsettings us-west-2-knode002
% storage
% set ebs size 100GiB
% commit
% ..
% ..
% ..
% cloudsettings us-west-2-knode003
% storage
% set ebs size 100GiB
% commit
% ..
% ..
% ..
```

4. Increase the EBS volume size of `us-west-2-gpu-node001` to 100 GiB and change the `instancetype` for `us-west-2-gpu-node001` to `g4dn.xlarge` (a lower-cost GPU instance with a single T4 GPU).

   If requirements justify a higher instance spec, use the appropriate instance type—any NVIDIA GPU instance should work.

   ```
   % cloudsettings us-west-2-gpu-node001
   % set instancetype g4dn.xlarge
   % commit
   % storage
   % set ebs size 100GiB
   % commit
   ```

5. Increase the EBS volume size to 200 GiB on the `director` system.

   ```
   % ..
   % ..
   % cloudsettings us-west-2-director
   % storage
   % set ebs size 200GiB
   % commit
   ```

6. Update the `softwareimage` for the `aws-cloud-director` category.

   ```
   % category
   % use aws-cloud-director
   % set softwareimage cloud-director-image
   ```

7. Update `disksetup` for the cloud director to use the same partitioning scheme set for the other public cloud nodes.

   ```
   % set disksetup /tmp/big-cloud-disk.xml
   % commit
   ```

Chapter 5. Power On and Provision the Cloud Nodes

Now that the required post-installation configuration has been completed, it is time to power on and provision the public cloud nodes. Public cloud node behavior is slightly different from on-premises equipment—the systems will not be provisioned in the target public cloud until they are first powered on. Additionally, the director node must be powered on and provisioned first—until it is fully provisioned, it is not possible to deploy the public cloud nodes it manages in a region. Just as with on-premises deployments, the public cloud nodes can be accessed through ssh during the installation process.

Watch the `/var/log/messages` and `/var/log/node-installer` log files to verify that everything is proceeding smoothly if you are unsure of a given node's deployment state.

1. Power on the cloud director.

   It will enter a `[ PENDING ]` state, then transition to `[ DOWN ]` (Instance has started).

   ```
   # cmsh
   % power on us-west-2-director
   ```

   The provisioning of the cloud director may take two or more hours due to the tens of gigabytes of software image data that must be synchronized to the public cloud. The process is complete when the cloud director moves to an `[ UP ]` state.

2. Power on the four public cloud nodes concurrently.

   Once the cloud director is fully provisioned, bringing up the other four public cloud nodes is much faster because their base images are already stored in the target region with the cloud director.

   ```
   % power on -n us-west-2-knode00[1-3],us-west-2-gpu-node001
   ```

3. Run `device` then `list` to ensure all public cloud nodes are in an `[ UP ]` state.

   Disregard any trailing `Status` output.

   ```
   % device
   % list
   Type                   Hostname (key)           MAC                Category             Ip              Network        Status
   ---------------------- ------------------------ ------------------ -------------------- --------------- -------------- ----------
   CloudNode              us-west-2-director       00:00:00:00:00:00  aws-cloud-director   172.17.255.251  us-west-2      [   UP   ]
   CloudNode              us-west-2-gpu-node001    00:00:00:00:00:00  k8s-cloud-gpu-worker 172.17.0.4      us-west-2      [   UP   ]
   CloudNode              us-west-2-knode001       00:00:00:00:00:00  k8s-cloud-master     172.17.0.1      us-west-2      [   UP   ]
   CloudNode              us-west-2-knode002       00:00:00:00:00:00  k8s-cloud-master     172.17.0.2      us-west-2      [   UP   ]
   CloudNode              us-west-2-knode003       00:00:00:00:00:00  k8s-cloud-master     172.17.0.3      us-west-2      [   UP   ]
   ```

4. Install the NVIDIA driver on `us-west-2-gpu-node001`.

   ssh to it as root and run all subsequent commands from the node in AWS.

```
# ssh us-west-2-gpu-node001
# apt install linux-headers-$(uname -r)
# distribution=$(. /etc/os-release;echo $ID$VERSION_ID | sed -e 's/\.//g')
# wget
https://developer.download.nvidia.com/compute/cuda/repos/$distribution/x86_64/cud
a-keyring_1.0-1_all.deb
# dpkg -I cuda-keyring_1.0-1_all.deb
# apt update
# apt install -y cuda-drivers –no-install-recommends
# rm cuda-keyring_1.0-1_all.deb
# nvidia-smi
```

5. Look for output from `nvidia-smi`, which like this, shows a successful installation.

   Expect possible variations in software versions and device utilization.

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            On   | 00000000:00:1E.0 Off |                    0 |
| N/A   36C    P8    15W /  70W |      2MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

6. Log out of the public cloud GPU node and back into the on-premises head node.
7. Execute the following to capture the modifications made to the public cloud GPU node, which will then be present in the image of any additional public cloud GPU nodes provisioned in this environment.

```
$ cmsh
# device
# use us-west-2-gpu-node001
# grabimage -w
```

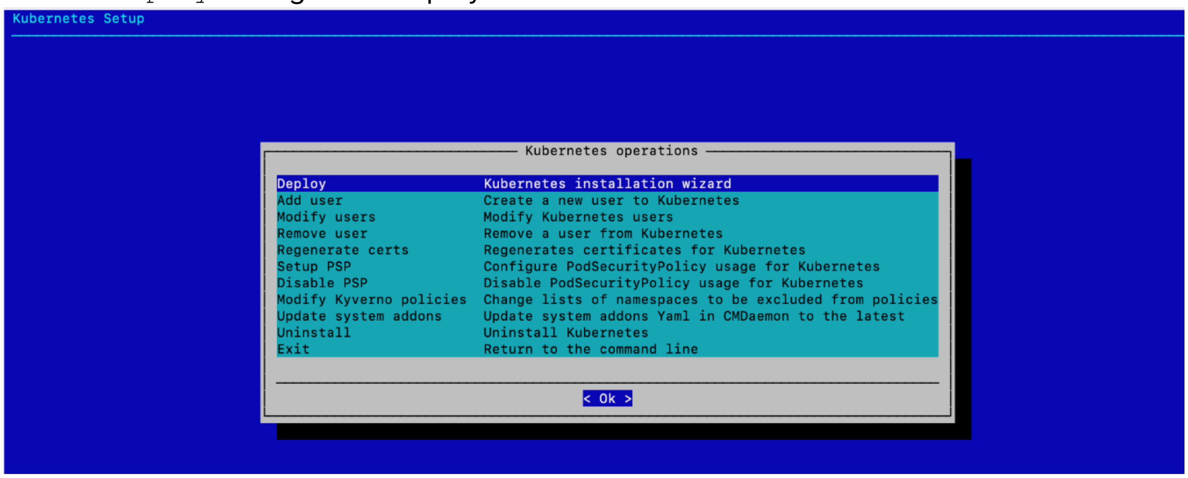
Chapter 6. Deploy Kubernetes

With all required public cloud instances deployed and configured for general use, the environment is ready for K8s deployment. In a hybrid environment, the same tool used to deploy on-premises K8s is used to deploy K8s in the public cloud as well.

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

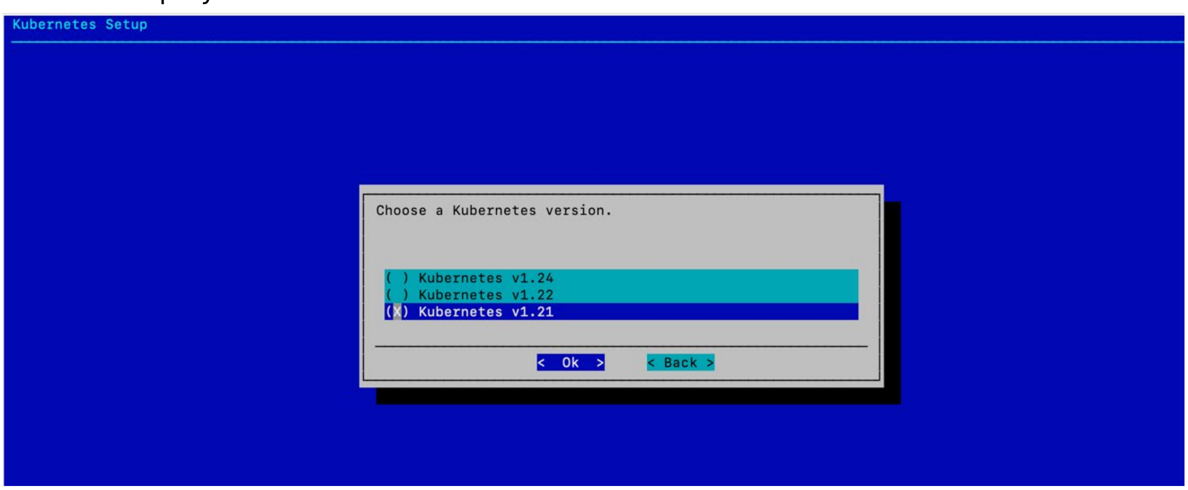
```
# cm-kubernetes-setup
```

2. Choose `Deploy` to begin the deployment and then select `Ok`.

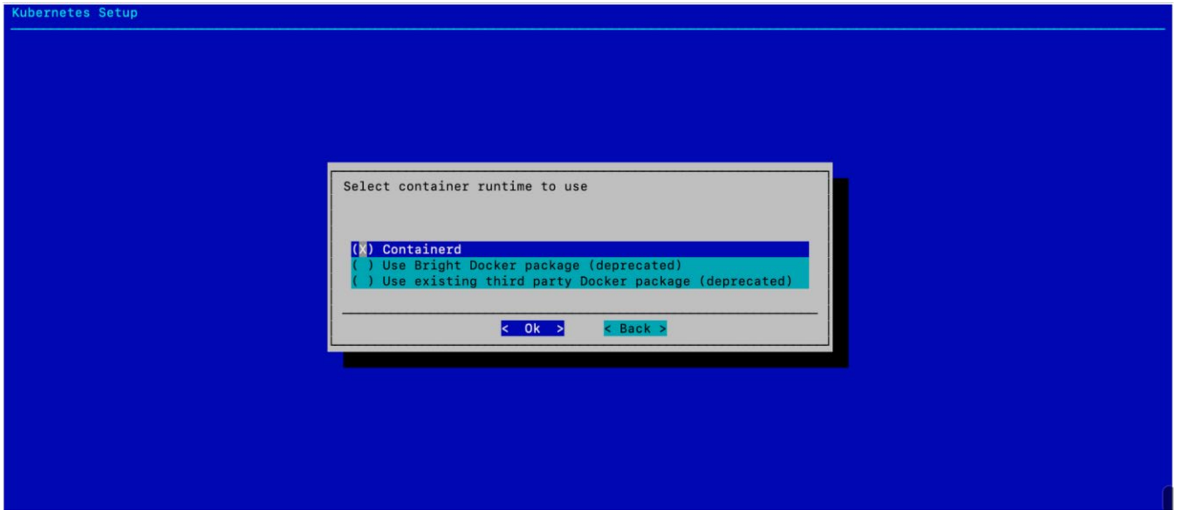


3. Choose `Kubernetes v1.21` and then select `Ok`.

   K8s version 1.21 was selected to match the version deployed in the on-premises DGX BasePOD deployment.

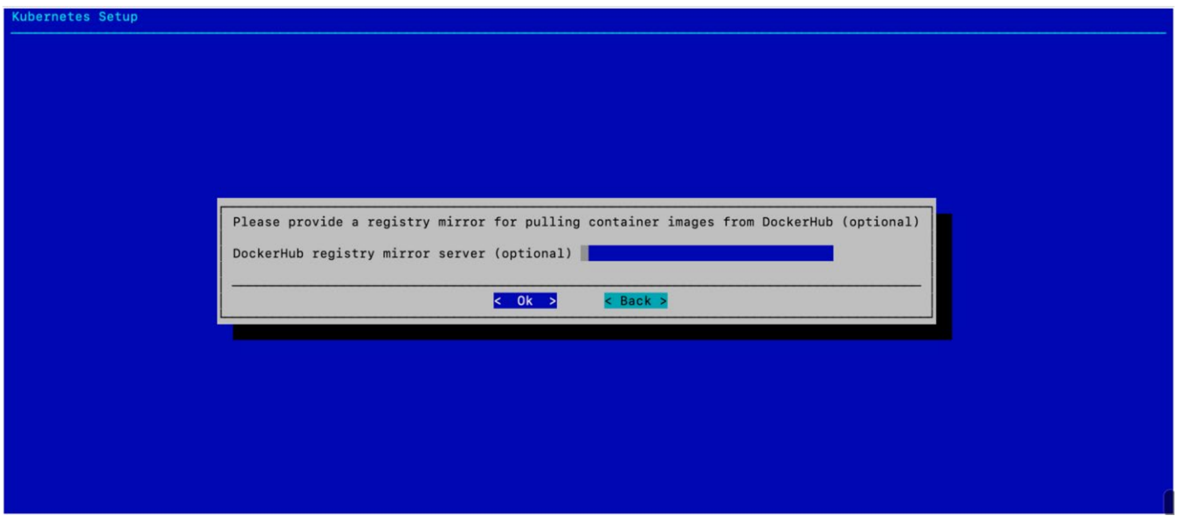


4. Choose `Containerd` (it should be selected by default) and then select `Ok`.



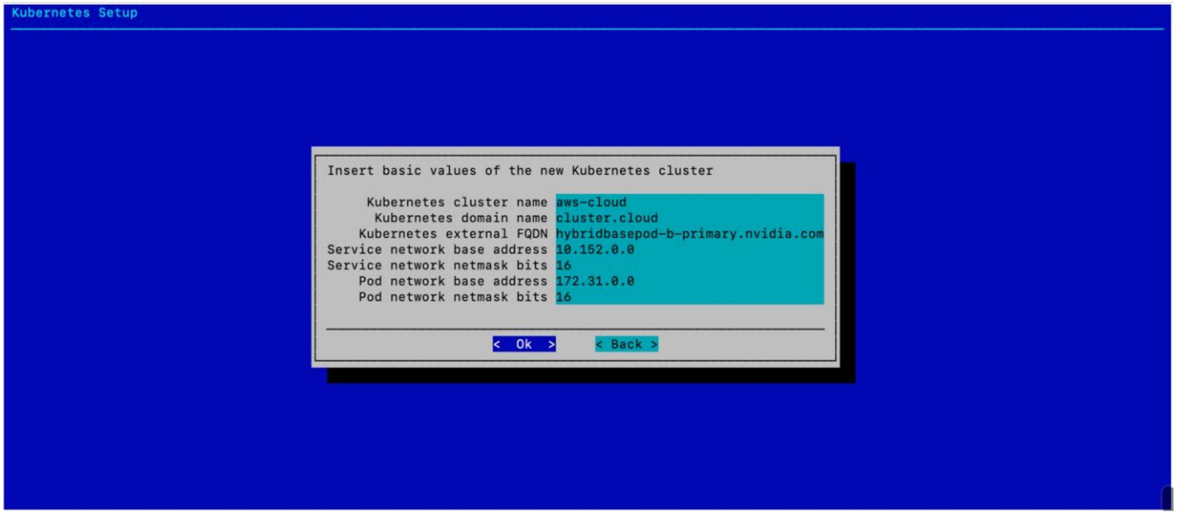
5. Optionally, provide a registry mirror and then select `Ok`.

   This example deployment did not require one.



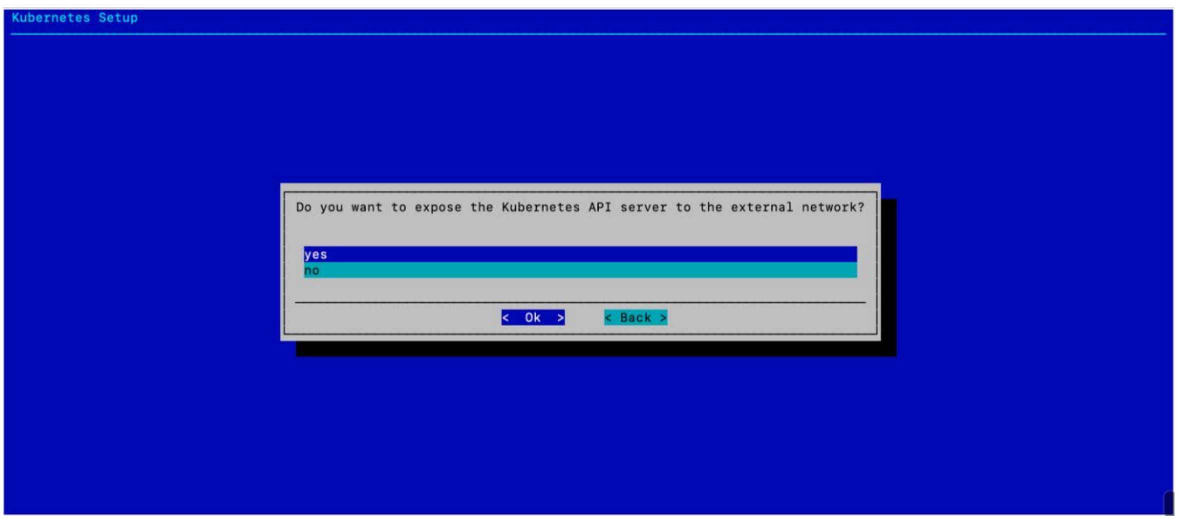
6. Configure the basic values of the K8s cluster and select `Ok`.

   Choose names that make it easy to understand that the K8s deployment is using public cloud resources. In addition, ensure that the service and pod network subnets do not overlap with existing subnets in the cluster.



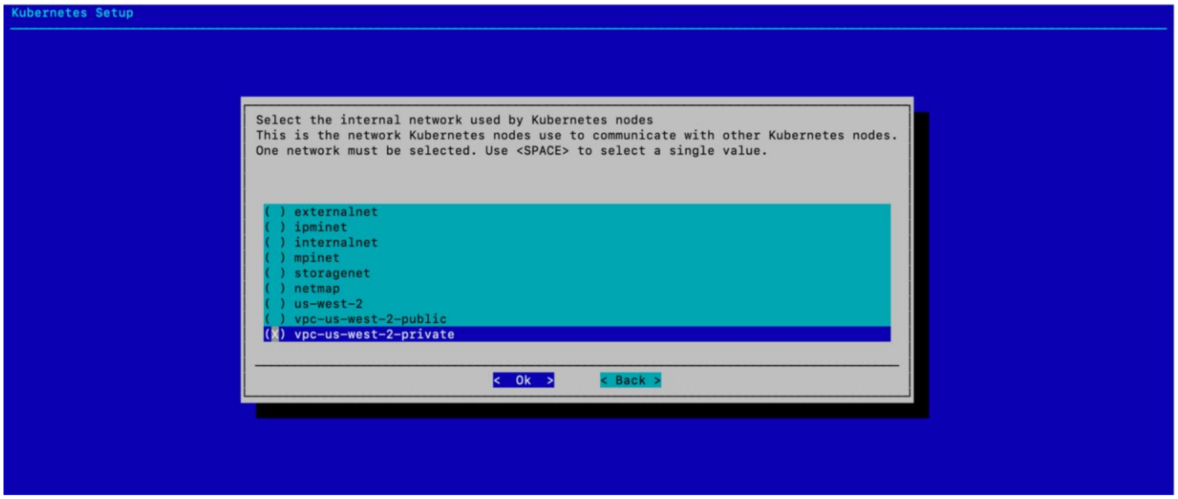
7. Choose `yes` to expose the K8s API server to the external network and then select `Ok`.

   This allows users to use the K8s cluster from the head node.

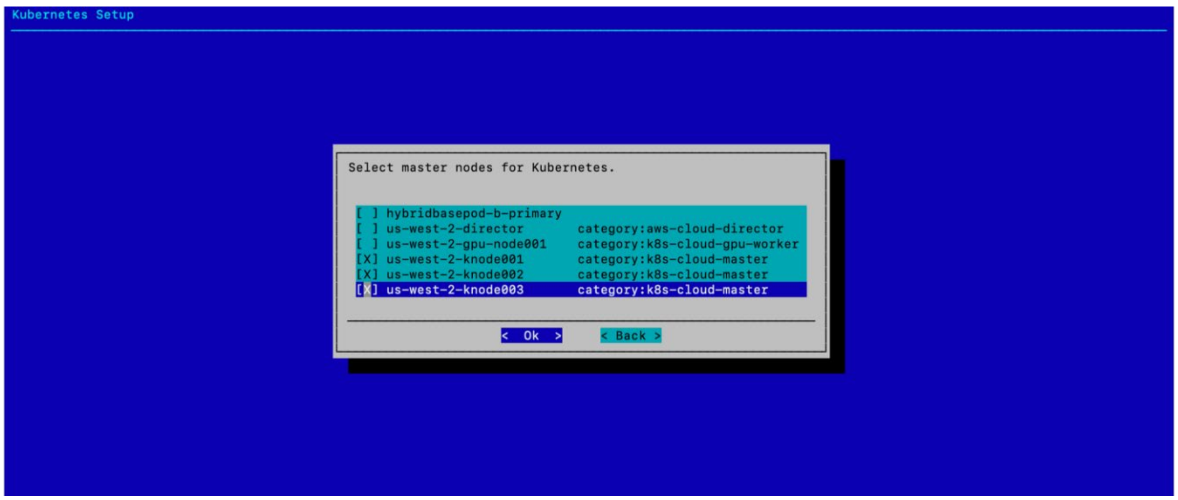


8. Choose `vpc-us-west-2-private` for the public cloud-based K8s environment and then select `Ok`.

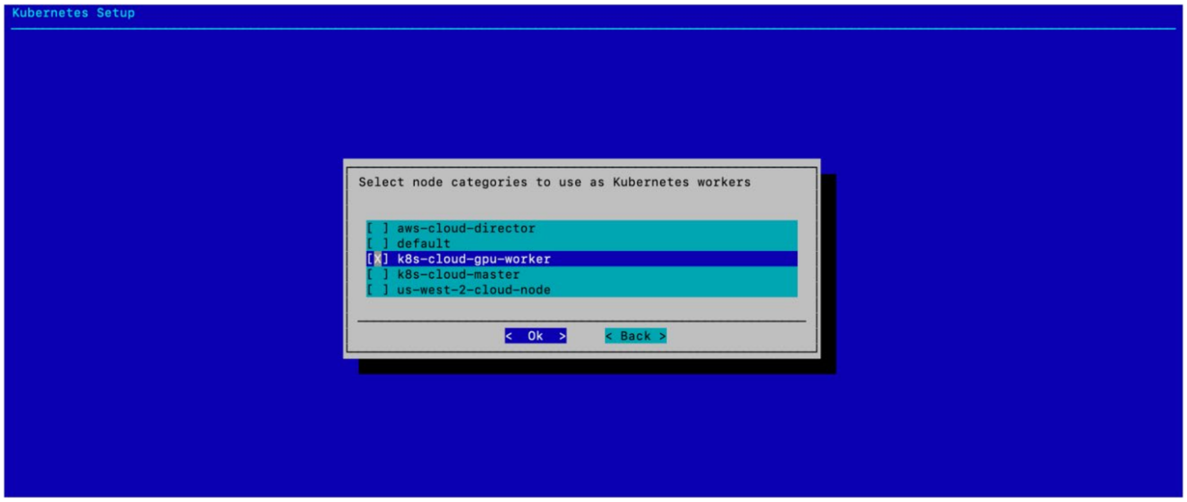
   This keeps internal K8s traffic entirely in the public cloud.



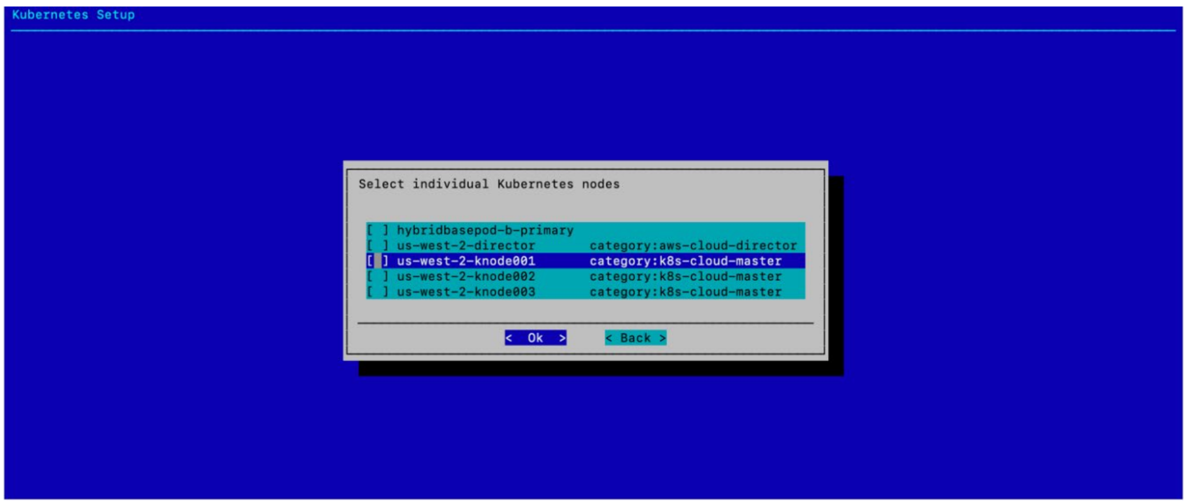
9. Choose the three `k8s-cloud-master` nodes and then select `Ok`.



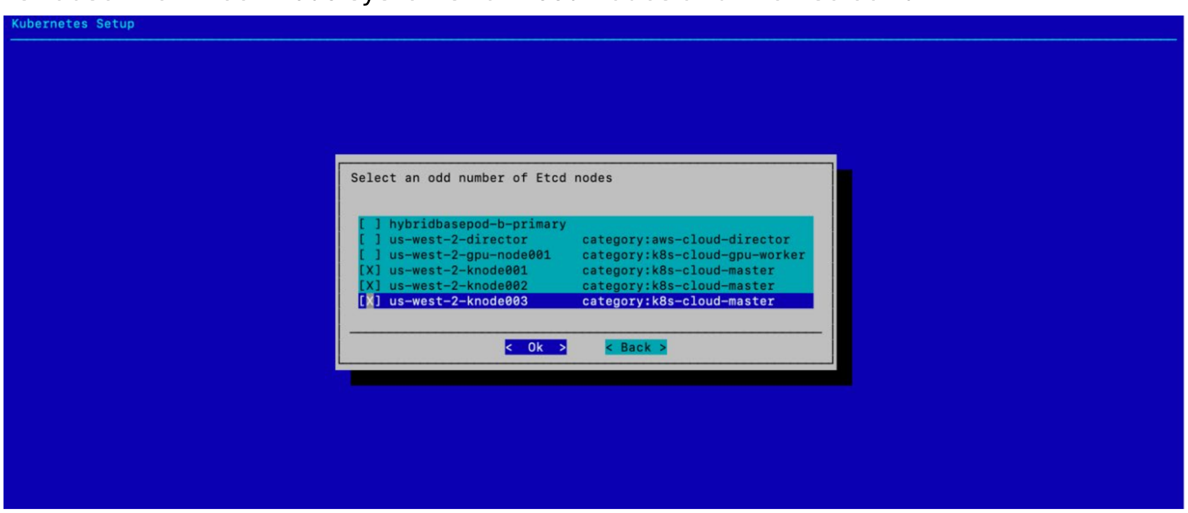
10. Choose `k8s-cloud-gpu-worker` for the worker node category and then select `Ok`.



11. Select `Ok` without configuring any individual K8s nodes.

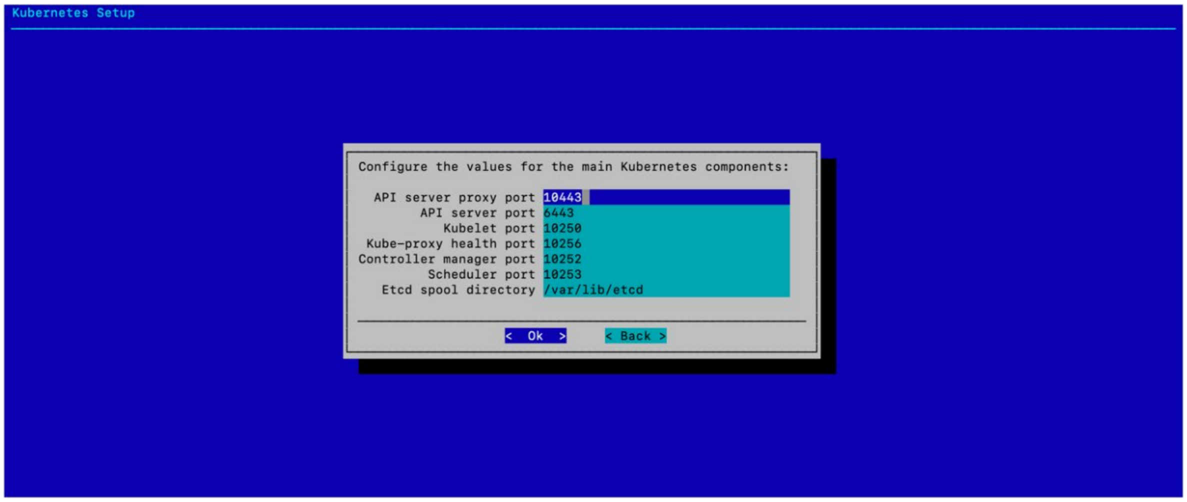


12. Choose the three `knode` systems for `Etcd` nodes and then select `Ok`.

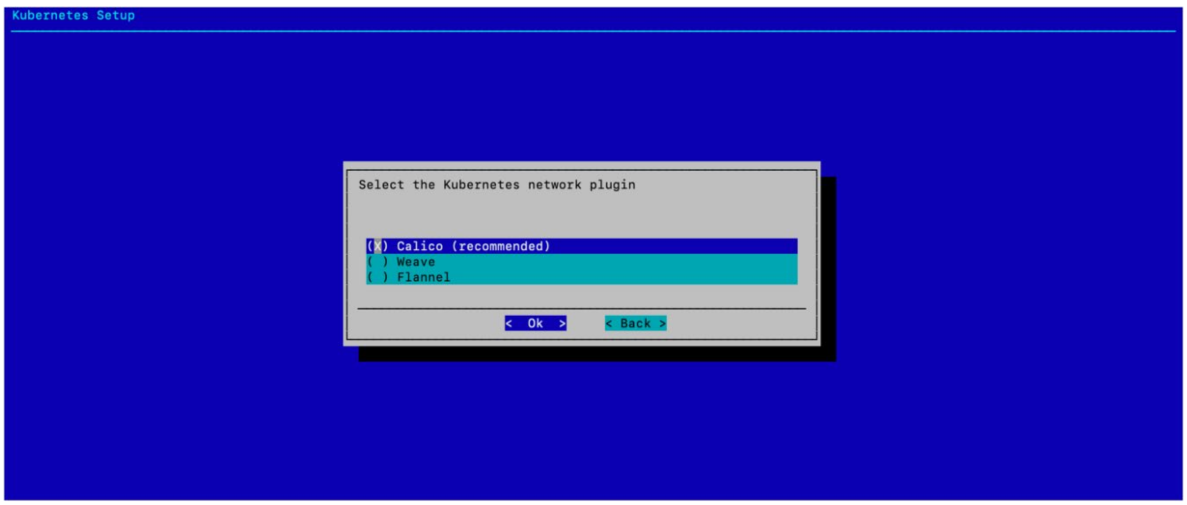


13. Configure the K8s main components and then select `Ok`.

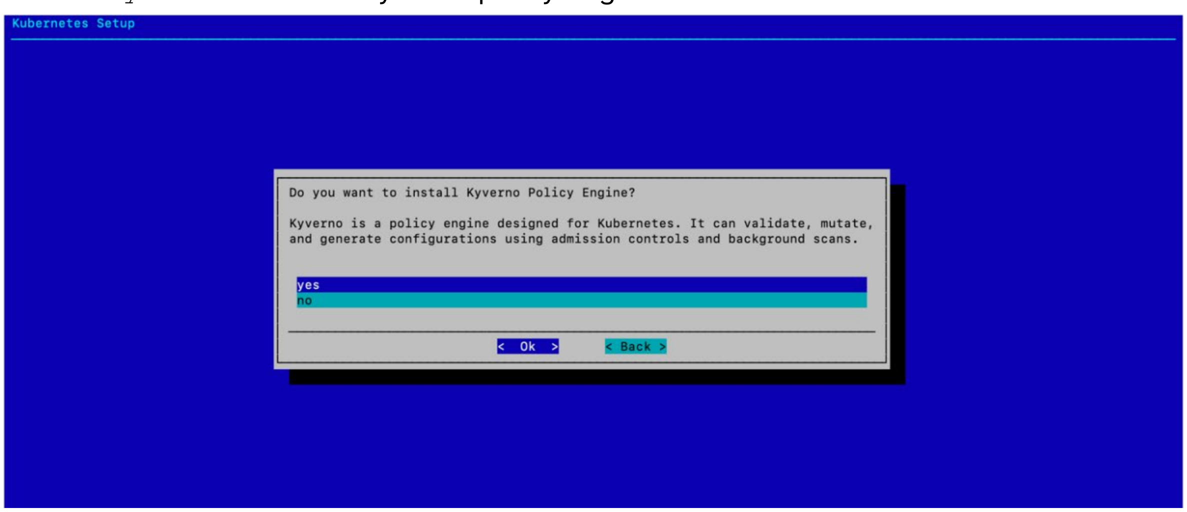
    Use the default ports and path here unless the environment requires different values. The default values were used in this deployment.



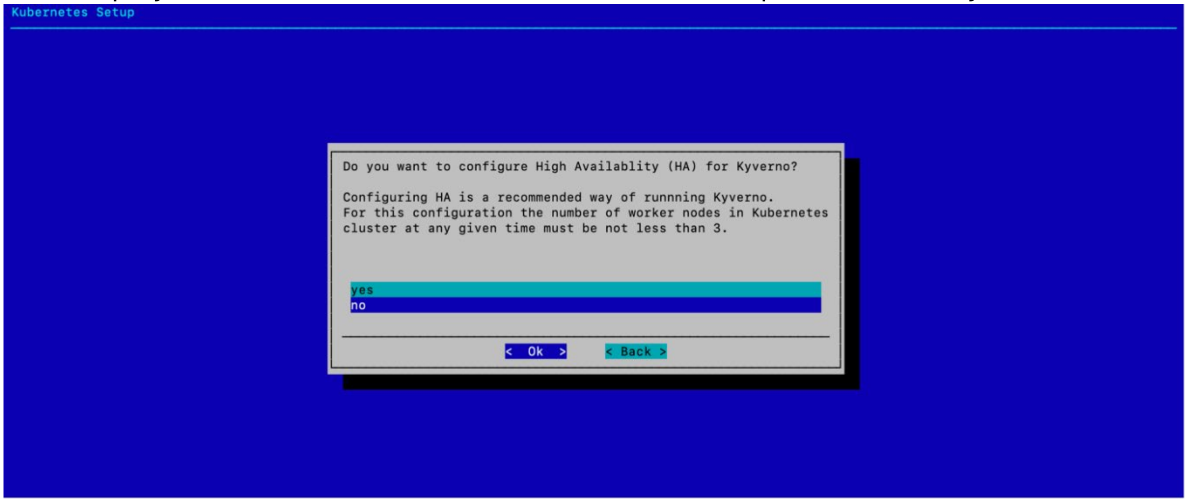
14. Choose the `Calico network plugin` and then select `Ok`.



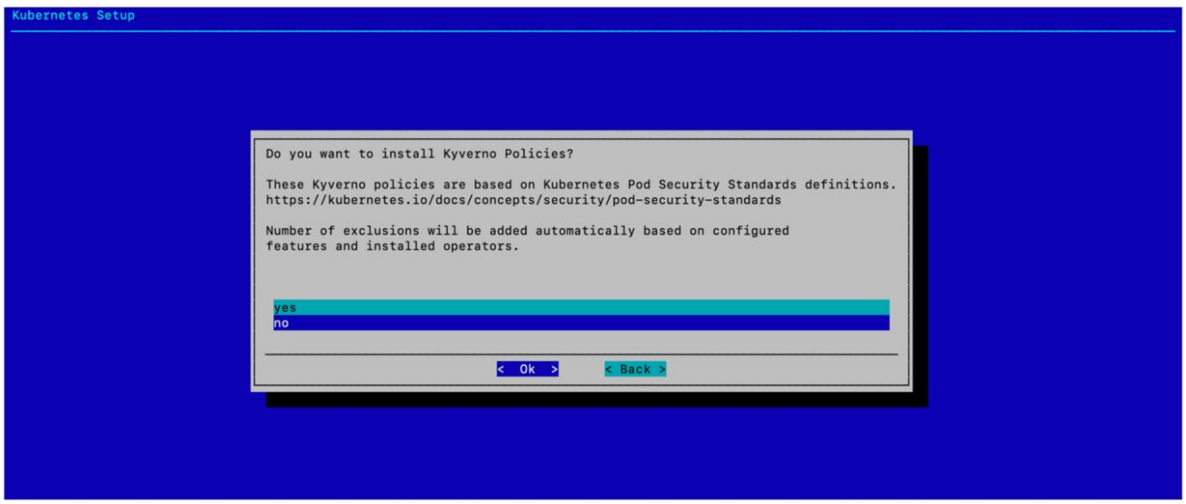
15. Choose `yes` to install the Kyverno policy engine and then select `Ok`.



16. Choose `no` to decline to configure HA for Kyverno and then select `Ok`.
    This deployment does not meet the minimum node requirement for Kyverno HA.

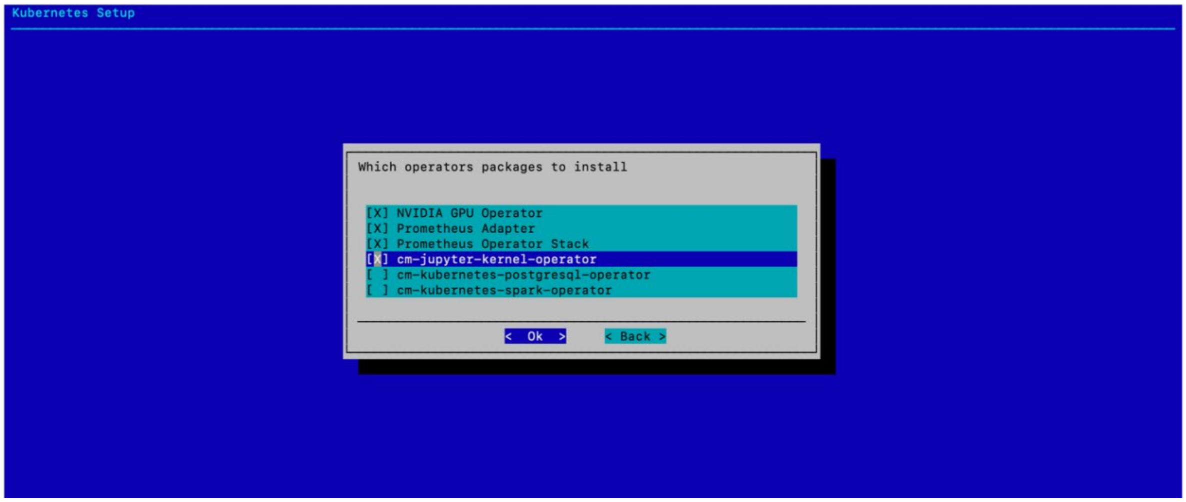


17. Choose whether to install Kyverno Policies and then select `Ok`.
    Unless required for the configuration, choose `no`.

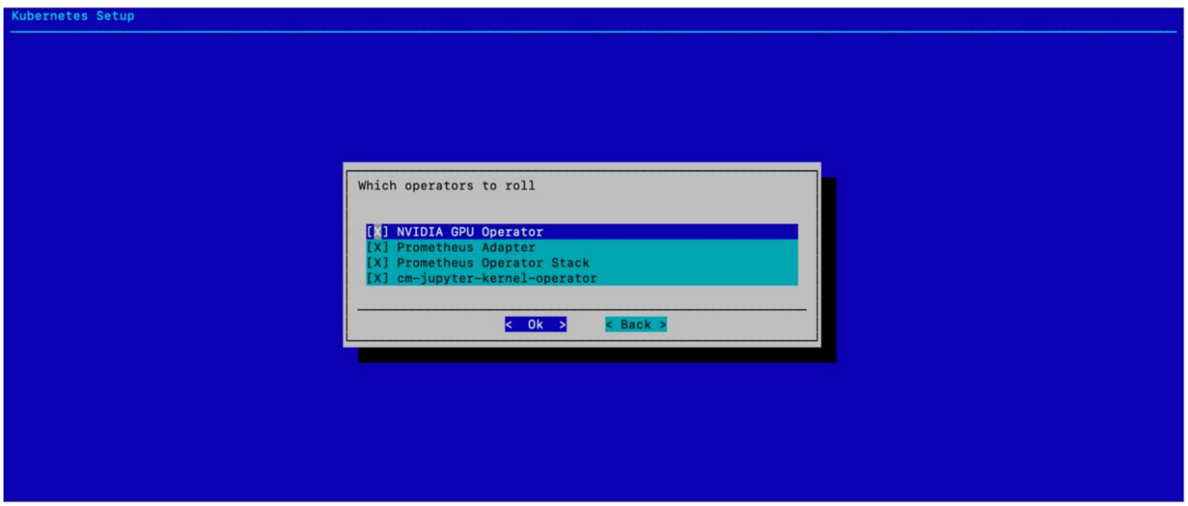


18. Choose the operator packages to install and then select `Ok`.

    As shown in the screenshot, choose `NVIDIA GPU Operator`, `Prometheus Adapter`, `Prometheus Adapter Stack`, and the `cm-jupyter-kernel-operator`.

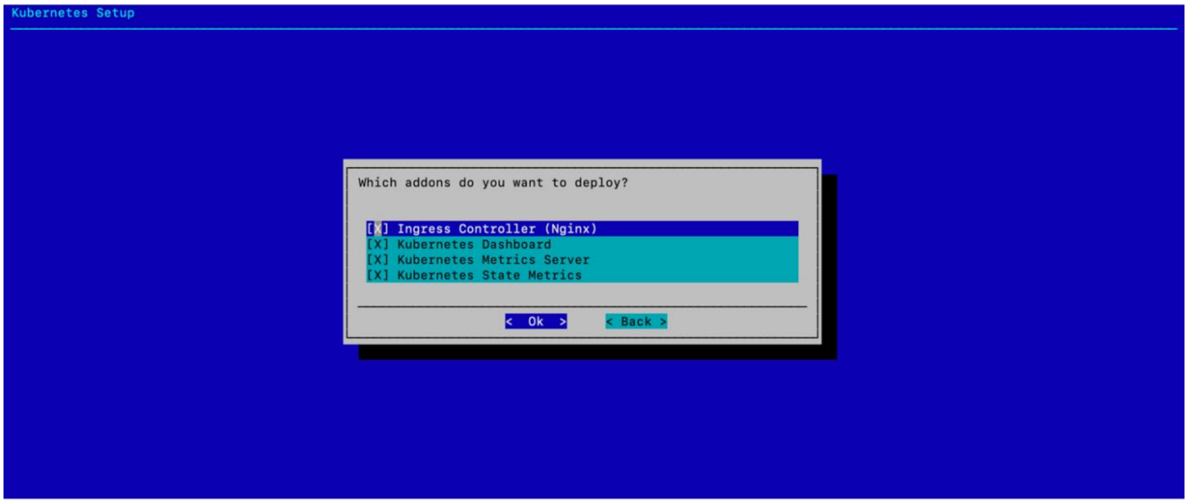


19. Choose the same four operators to be rolled up with the defaults and then select `Ok`.



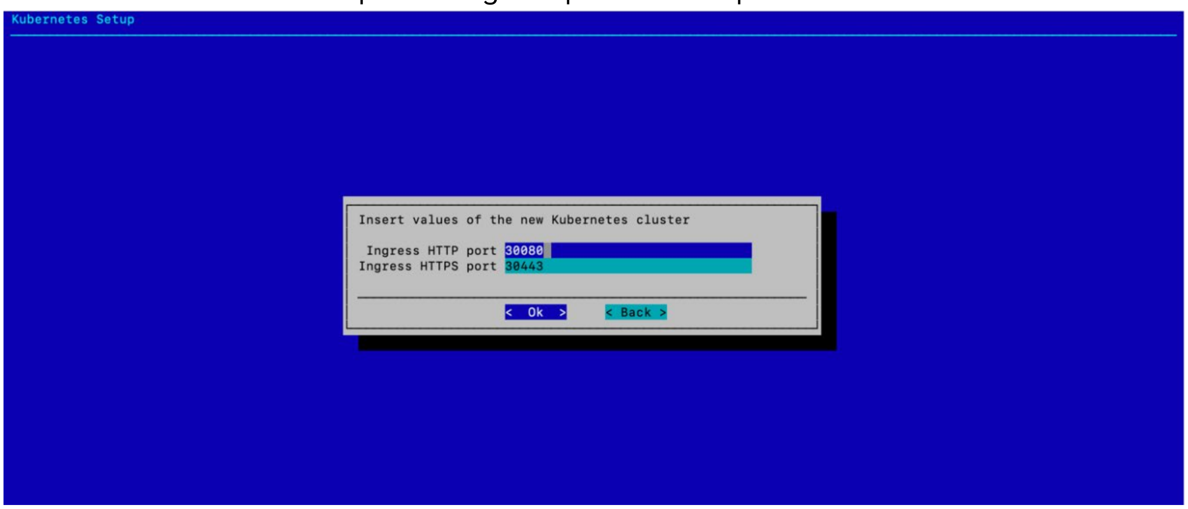
20. Choose the addons to deploy and then select `Ok`.

    As shown in the screenshot, choose `Ingress Controller (Nginx)`, `Kubernetes Dashboard`, `Kubernetes Metrics Server`, and `Kubernetes State Metrics`.



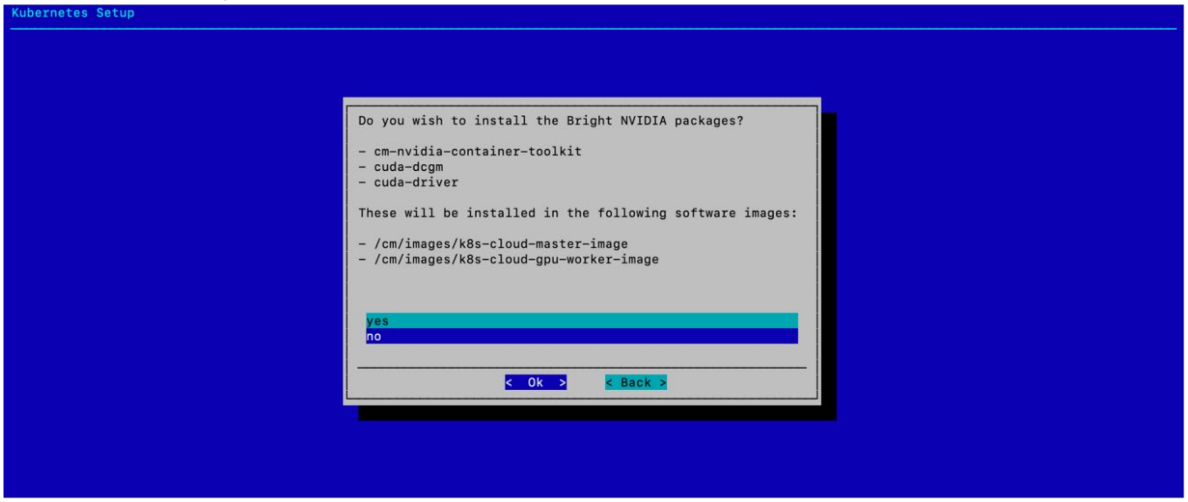
21. Choose the Ingress ports for the cluster and then select `Ok`.

    Use the defaults unless specific ingress ports are required.

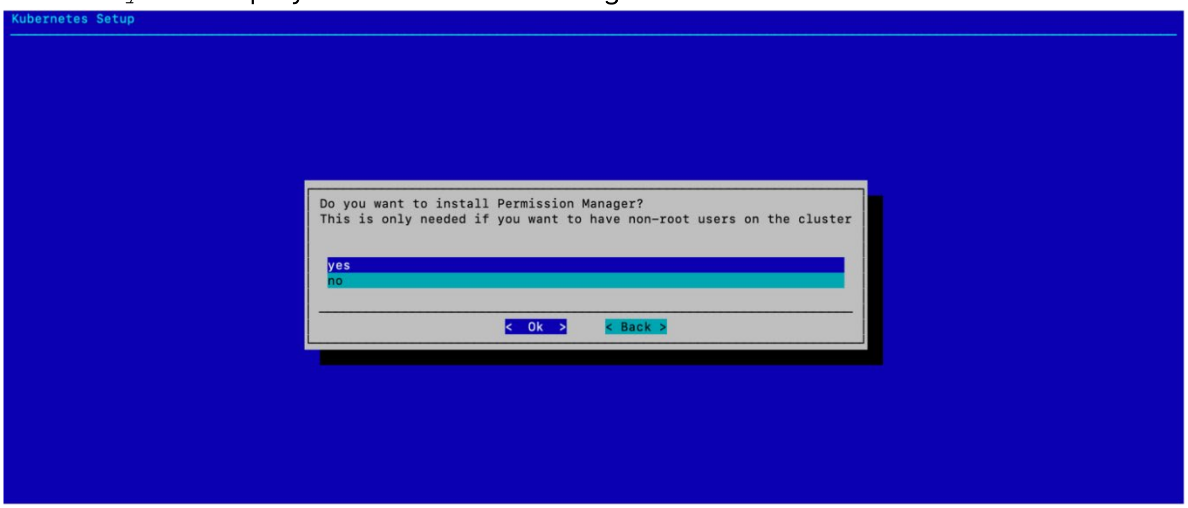


22. Choose `no` when asked to install the Bright NVIDIA packages and then select `Ok`.

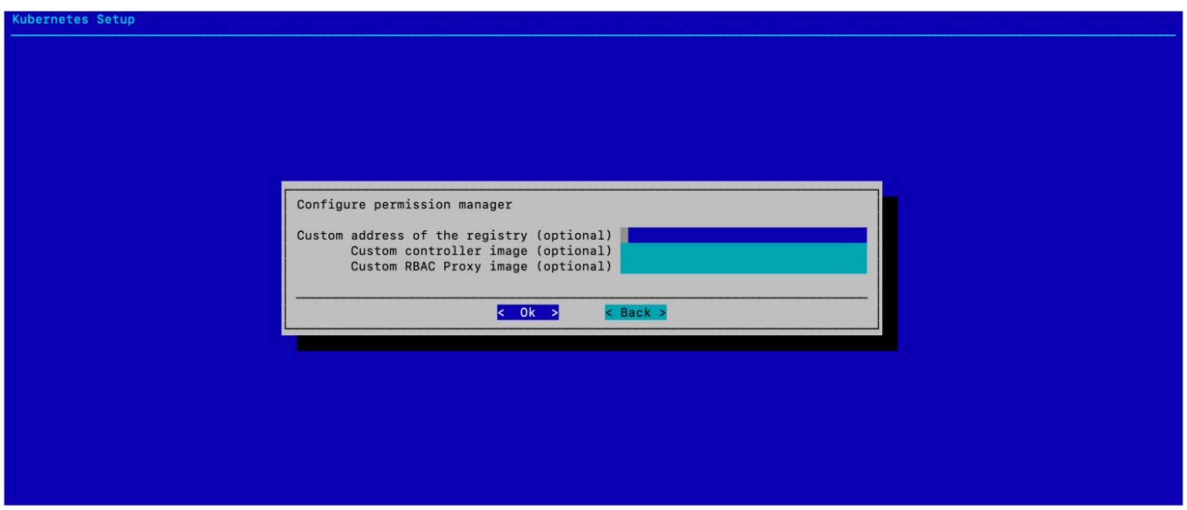
    Since the K8s control plane nodes do not have GPUs, the GPU Operator manages NVIDIA OS components.



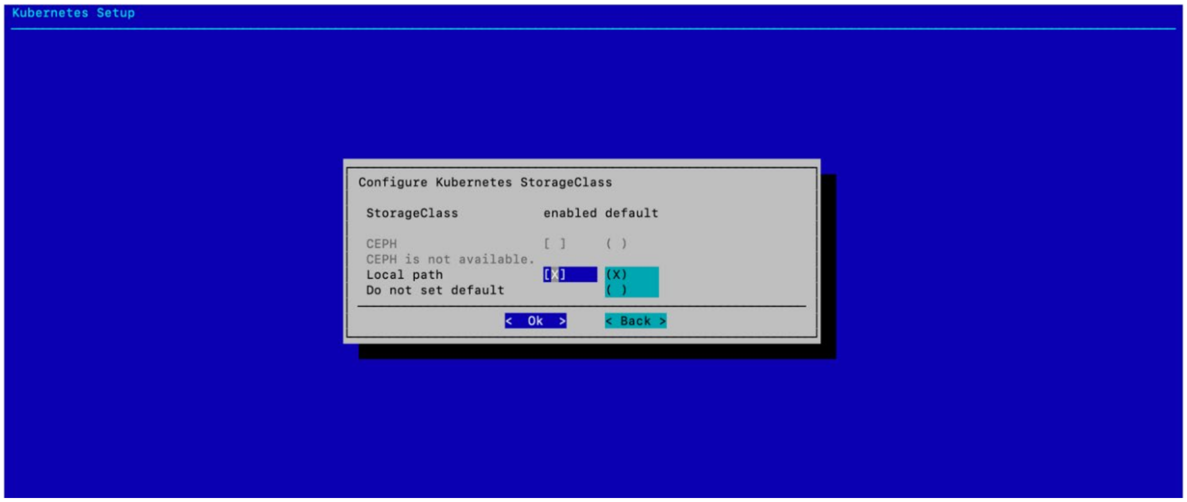
23. Choose `yes` to deploy the Permission Manager and then select `Ok`.



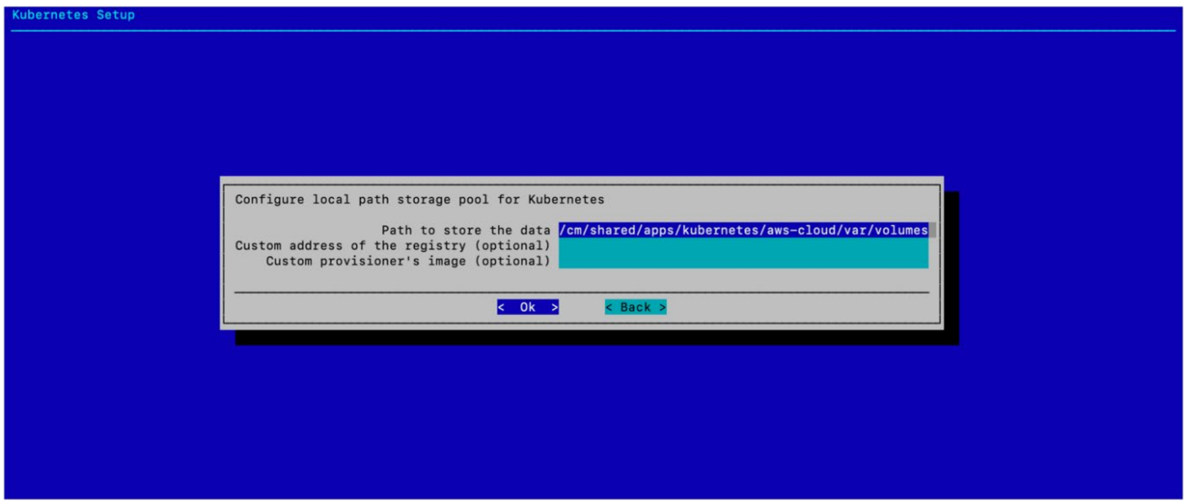
24. Select `Ok` without configuring any optional values.



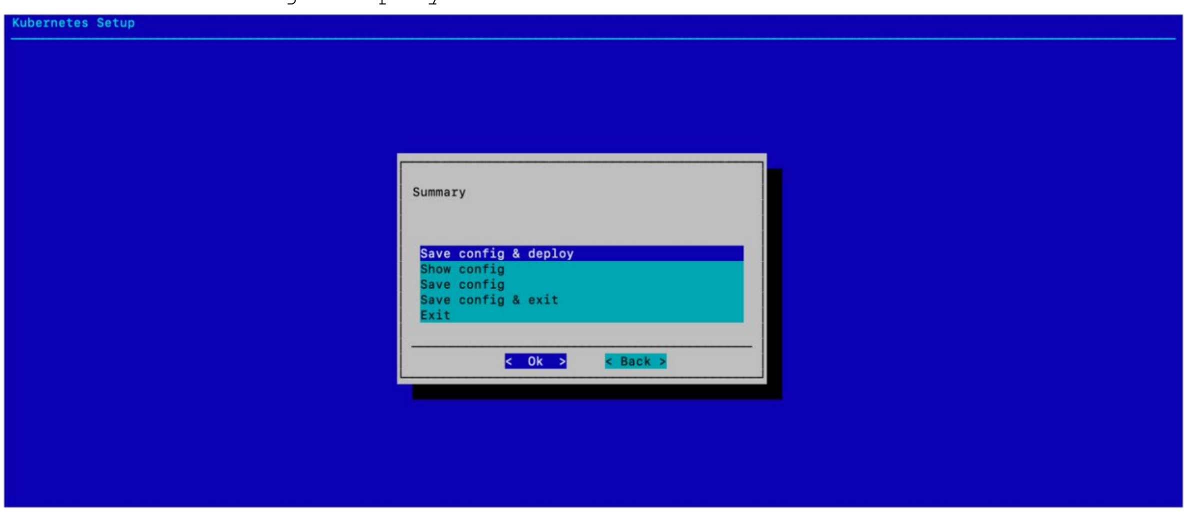
25. Choose both `enabled` and `default` for the `Local path` storage class and then select `Ok`.



26. Select `Ok` without changing any of the default values.

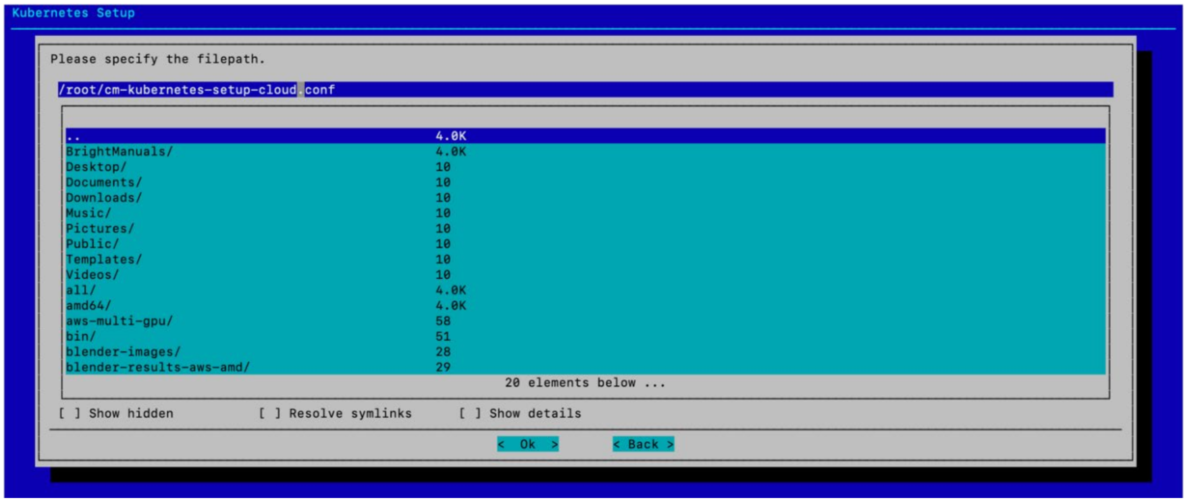


27. Choose `Save config & deploy` and then select `Ok`.



28. Change the filepath to `/root/cm-kubernetes-setup-cloud.conf` and then select `Ok`.

    The filepath was changed to avoid name conflicts with the existing K8s configuration file from the initial on-premises deployment. Wait for the installation to finish.



29. Verify the K8s cluster is installed properly.

    The K8s module may need to be unloaded for the on-premises deployment if already loaded or use the switch command as a shortcut to unload on-premises and load the public cloud module.

```
# module load kubernetes/aws-cloud/
# 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
us-west-2-gpu-node001  Ready    worker                 6m48s   v1.21.4
us-west-2-knode001     Ready    control-plane,master   6m48s   v1.21.4
us-west-2-knode002     Ready    control-plane,master   6m48s   v1.21.4
us-west-2-knode003     Ready    control-plane,master   6m48s   v1.21.4
```

30. Verify that a GPU job can be run on the K8s cluster.

   a. Save the following text to a file named `gpu.yaml`.

```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod-pytorch
spec:
  restartPolicy: Never
  containers:
  - name: pytorch-container
    image: nvcr.io/nvidia/pytorch:22.08-py3
    command: ["nvidia-smi"]
    resources:
      limits:
             nvidia.com/gpu: 1
```

   b. Execute the code using `kubectl apply`.

```
# kubectl apply -f gpu.yaml
```

   c. Use `kubectl logs` to check the result.

   The output should be like the following.

```
# kubectl logs gpu-pod-pytorch
Tue Feb 14 22:25:53 2023
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.85.12    Driver Version: 525.85.12    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla T4            On   | 00000000:00:1E.0 Off |                    0 |
| N/A   28C    P8    14W /  70W |      2MiB / 15360MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

Chapter 7. Create Additional Worker Nodes

The steps in this section cover how to extend the pool of worker nodes.

1. Access `cmsh` and enter the `device` sub-menu.

```
# cmsh
% device
```

2. Clone the single worker node.

   Maintaining the naming convention will automate the IP address increment.

```
% clone us-west-2-gpu-node001 us-west-2-gpu-node002
% commit
```

3. Power the additional worker node on and wait until it enters the `[ UP ]` state.

```
% power on us-west-2-gpu-node002
```

4. Verify the worker nodes are ready by using `kubectl`.

   The worker node should automatically be available as part of the public cloud Kubernetes worker pool because the entire category is marked as worker nodes. The resources should be ready to use immediately.

```
# kubectl get nodes
NAME                     STATUS   ROLES                    AGE   VERSION
us-west-2-gpu-node001    Ready    worker                   25h   v1.21.4
us-west-2-gpu-node002    Ready    worker                   10m   v1.21.4
us-west-2-knode001       Ready    control-plane,master     25h   v1.21.4
us-west-2-knode002       Ready    control-plane,master     25h   v1.21.4
us-west-2-knode003       Ready    control-plane,master     25h   v1.21.4
```

Chapter 8. (Optional) Enable Jupyter Operator Use in Cloud K8s Cluster

In the on-premises DGX BasePOD deployment guide, `cm-jupyter-setup` can be optionally configured and integrated into K8s. The same service, running from the head node, can be used to provide Jupyter access to the public cloud-based K8s cluster as well.

1. Validate `cm-jupyterhub` is set up and running correctly.

```
# service cm-jupyterhub status
```

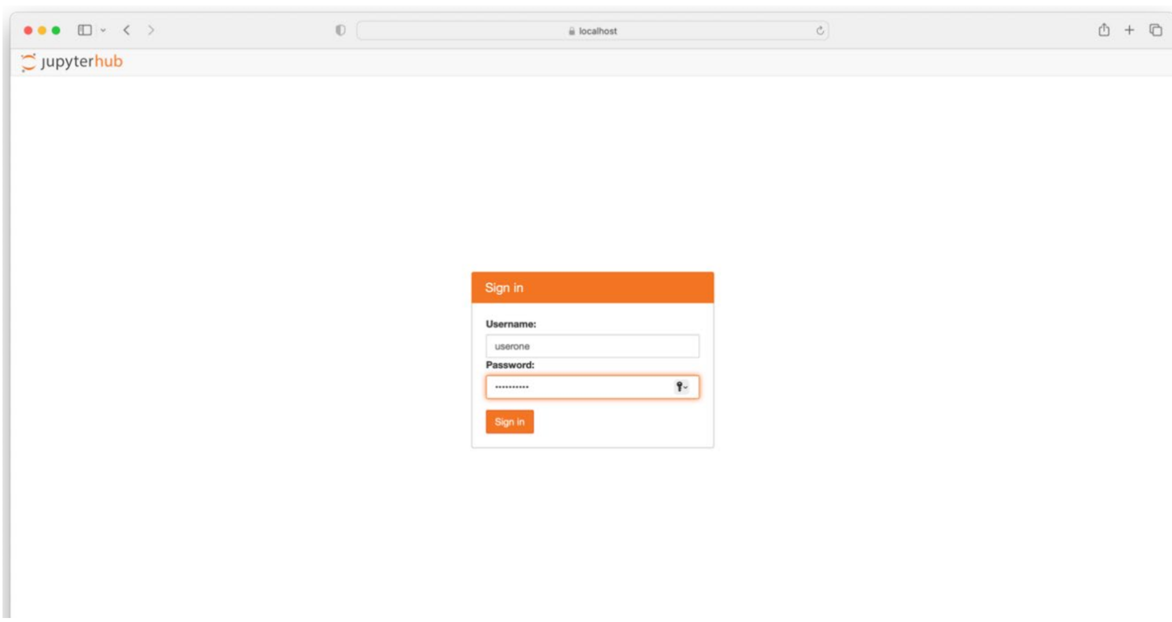
2. Configure a user and provide access to the appropriate K8s cluster.

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

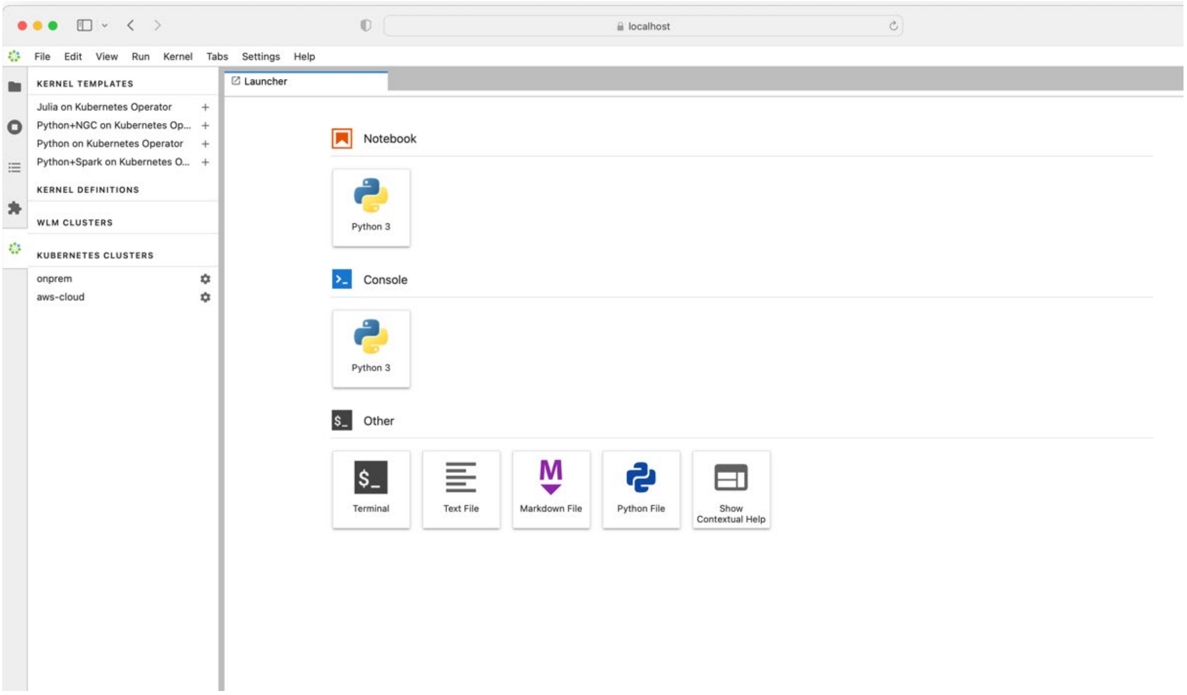
3. When using K8s via Jupyter, users must be added separately using K8s with the following commands. Users must have permission to access the Jupyter kernel operator in both K8s clusters to use the kernel templates.

```
# apt install cm-python39
# cm-kubernetes-setup --add-user userone --cluster aws-cloud --operators cm-jupyter-kernel-operator
# cm-kubernetes-setup --add-user userone --cluster onprem --operators cm-jupyter-kernel-operator
```

4. Sign in to the Jupyter web interface using the account configured with Jupyter kernel operator permissions.

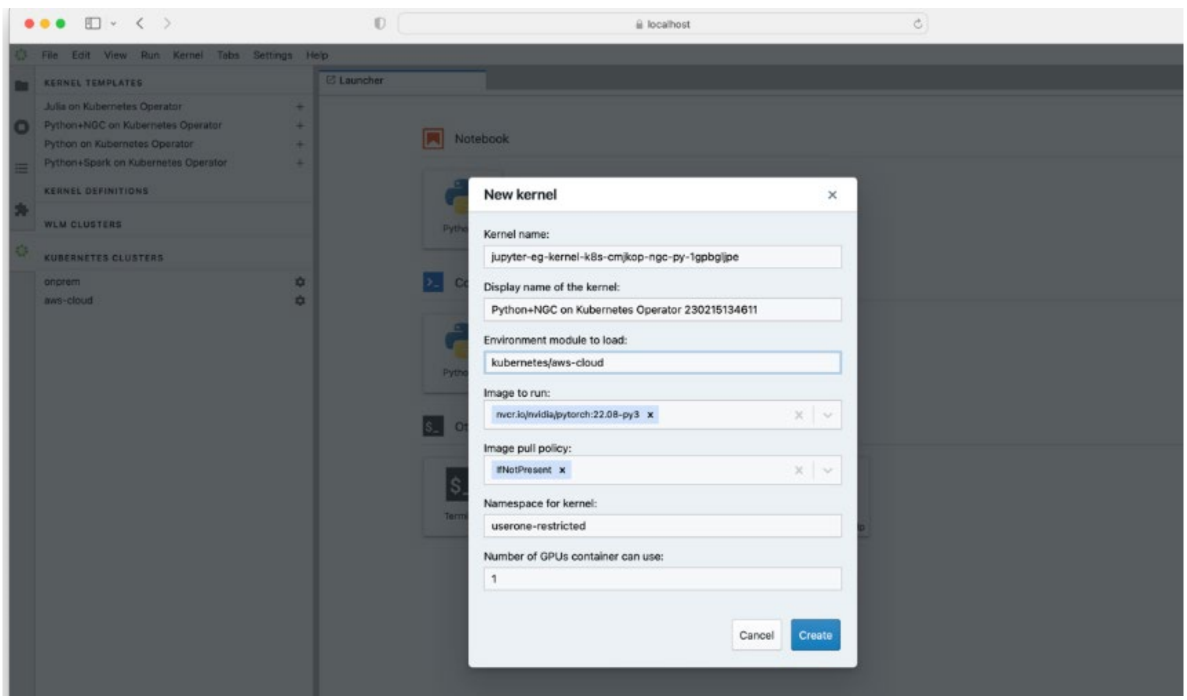


5. Navigate to the Bright tab, choose the `Python+NGC on Kubernetes Operator` kernel template, and then select `Ok`.

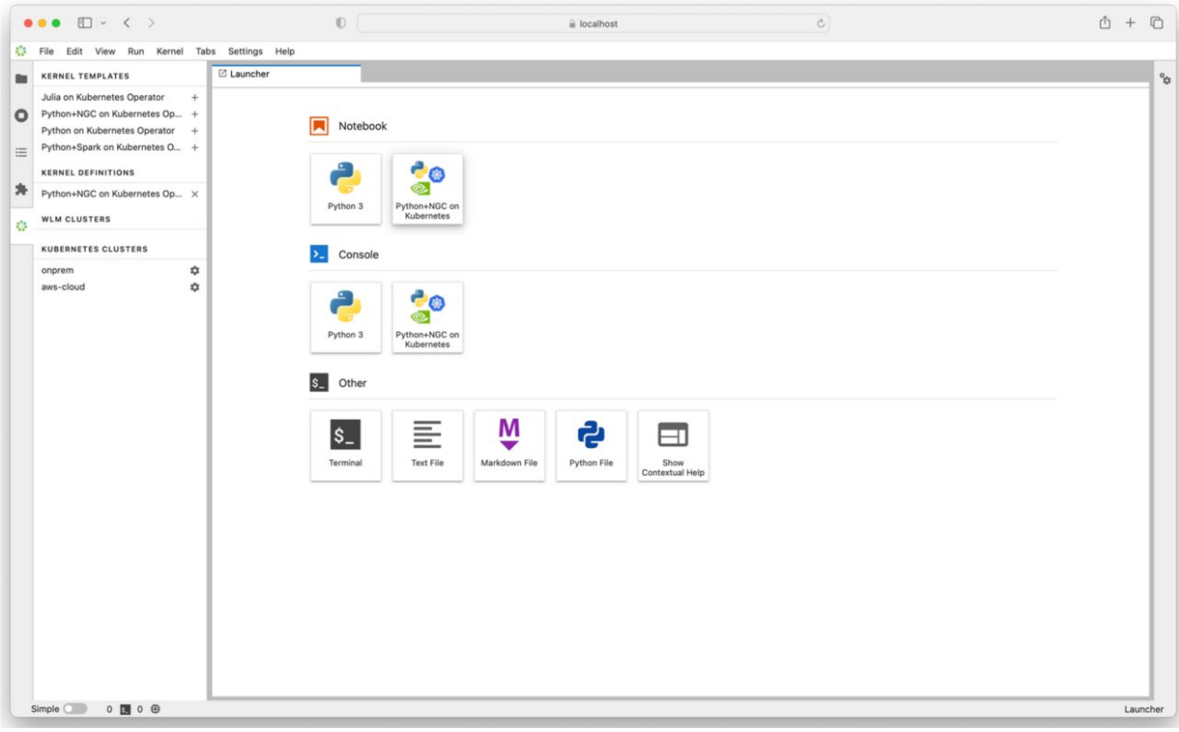


6. Fill in the required fields on the resulting `New kernel` window and then select `Create`.

   In this example, the public cloud-based K8s deployment was targeted by adding the cluster name (aws-cloud) as a path extension to the K8s environment module, and it was specified that the container could use a single GPU.



7. Select `Python+NGC on Kubernetes` in the `Notebook` section.



8. Once the state of the operator becomes Idle, run `nvidia-smi` to confirm the notebook is running on a T4 GPU instance.