NVIDIA Bright Cluster Manager 9.2

Containerization Manual

Revision: 79c0f2bc9

Date: Fri Jun 27 2025

©2024 NVIDIA Corporation & affiliates. All Rights Reserved. This manual or parts thereof may not be reproduced in any form unless permitted by contract or by written permission of NVIDIA Corporation.

Trademarks

Linux is a registered trademark of Linus Torvalds. PathScale is a registered trademark of Cray, Inc. Red Hat and all Red Hat-based trademarks are trademarks or registered trademarks of Red Hat, Inc. SUSE is a registered trademark of SUSE LLC. NVIDIA, CUDA, GPUDirect, HPC SDK, NVIDIA DGX, NVIDIA Nsight, and NVLink are registered trademarks of NVIDIA Corporation. FLEXlm is a registered trademark of Flexera Software, Inc. PBS Professional, and Green Provisioning are trademarks of Altair Engineering, Inc. All other trademarks are the property of their respective owners.

Rights and Restrictions

All statements, specifications, recommendations, and technical information contained herein are current or planned as of the date of publication of this document. They are reliable as of the time of this writing and are presented without warranty of any kind, expressed or implied. NVIDIA Corporation shall not be liable for technical or editorial errors or omissions which may occur in this document. NVIDIA Corporation shall not be liable for any damages resulting from the use of this document.

Limitation of Liability and Damages Pertaining to NVIDIA Corporation

The NVIDIA Bright Cluster Manager product principally consists of free software that is licensed by the Linux authors free of charge. NVIDIA Corporation shall have no liability nor will NVIDIA Corporation provide any warranty for the NVIDIA Bright Cluster Manager to the extent that is permitted by law. Unless confirmed in writing, the Linux authors and/or third parties provide the program as is without any warranty, either expressed or implied, including, but not limited to, marketability or suitability for a specific purpose. The user of the NVIDIA Bright Cluster Manager product shall accept the full risk for the quality or performance of the product. Should the product malfunction, the costs for repair, service, or correction will be borne by the user of the NVIDIA Bright Cluster Manager product. No copyright owner or third party who has modified or distributed the program as permitted in this license shall be held liable for damages, including general or specific damages, damages caused by side effects or consequential damages, resulting from the use of the program or the un-usability of the program (including, but not limited to, loss of data, incorrect processing of data, losses that must be borne by you or others, or the inability of the program to work together with any other program), even if a copyright owner or third party had been advised about the possibility of such damages unless such copyright owner or third party has signed a writing to the contrary.

Table of Contents

Preface

Welcome to the *Containerization Manual* for NVIDIA Bright Cluster Manager 9.2.

0.1 About This Manual

This manual is aimed at helping cluster administrators install, understand, configure, and manage the containerization integration capabilities of NVIDIA Bright Cluster Manager. The administrator is expected to be reasonably familiar with the *Administrator Manual*.

0.2 About The Manuals In General

Regularly updated versions of the NVIDIA Bright Cluster Manager 9.2 manuals are available on updated clusters by default at /cm/shared/docs/cm. The latest updates are always online at http://support.brightcomputing.com/manuals.

- The *Installation Manual* describes installation procedures for the basic cluster.
- The *Administrator Manual* describes the general management of the cluster.
- The *User Manual* describes the user environment and how to submit jobs for the end user.
- The *Cloudbursting Manual* describes how to deploy the cloud capabilities of the cluster.
- The *Developer Manual* has useful information for developers who would like to program with the cluster manager.
- The *Machine Learning Manual* describes how to install and configure machine learning capabilities with the cluster manager.
- The *Edge Manual* explains how the cluster manager can be used with edge sites.

If the manuals are downloaded and kept in one local directory, then in most pdf viewers, clicking on a cross-reference in one manual that refers to a section in another manual opens and displays that section in the second manual. Navigating back and forth between documents is usually possible with keystrokes or mouse clicks.

For example: <Alt>-<Backarrow> in Acrobat Reader, or clicking on the bottom leftmost navigation button of xpdf, both navigate back to the previous document.

The manuals constantly evolve to keep up with the development of the cluster manager environment and the addition of new hardware and/or applications. The manuals also regularly incorporate customer feedback. Administrator and user input is greatly valued at Bright Computing. So any comments, suggestions or corrections will be very gratefully accepted at manuals@brightcomputing.com.

There is also a feedback form available via Bright View, via the menu icon, ☰, following the navigation path:

☰>Help>Feedback

0.3 Getting Administrator-Level Support

If the reseller from whom the cluster manager was bought offers direct support, then the reseller should be contacted.

Otherwise the primary means of support is via the website `https://enterprise-support.nvidia.com/s/create-case`. This allows the administrator to submit a support request via a web form, and opens up a trouble ticket. It is a good idea to try to use a clear subject header, since that is used as part of a reference tag as the ticket progresses. Also helpful is a good description of the issue. The followup communication for this ticket goes via standard e-mail. Section 16.2 of the *Administrator Manual* has more details on working with support.

0.4 Getting Professional Services

Bright Computing normally differentiates between

- professional services (customer asks Bright Computing to do something or asks Bright Computing to provide some service), and
- support (customer has a question or problem that requires an answer or resolution).

Professional services can be provided after consulting with the reseller, or the Bright account manager.

©2024 NVIDIA Corporation & affiliates. All Rights Reserved. This manual or parts thereof may not be reproduced in any form unless permitted by contract or by written permission of NVIDIA Corporation.

Trademarks

Linux is a registered trademark of Linus Torvalds. PathScale is a registered trademark of Cray, Inc. Red Hat and all Red Hat-based trademarks are trademarks or registered trademarks of Red Hat, Inc. SUSE is a registered trademark of SUSE LLC. NVIDIA, CUDA, GPUDirect, HPC SDK, NVIDIA DGX, NVIDIA Nsight, and NVLink are registered trademarks of NVIDIA Corporation. FLEXlm is a registered trademark of Flexera Software, Inc. PBS Professional, and Green Provisioning are trademarks of Altair Engineering, Inc. All other trademarks are the property of their respective owners.

Rights and Restrictions

All statements, specifications, recommendations, and technical information contained herein are current or planned as of the date of publication of this document. They are reliable as of the time of this writing and are presented without warranty of any kind, expressed or implied. NVIDIA Corporation shall not be liable for technical or editorial errors or omissions which may occur in this document. NVIDIA Corporation shall not be liable for any damages resulting from the use of this document.

Limitation of Liability and Damages Pertaining to NVIDIA Corporation

The NVIDIA Bright Cluster Manager product principally consists of free software that is licensed by the Linux authors free of charge. NVIDIA Corporation shall have no liability nor will NVIDIA Corporation provide any warranty for the NVIDIA Bright Cluster Manager to the extent that is permitted by law. Unless confirmed in writing, the Linux authors and/or third parties provide the program as is without any warranty, either expressed or implied, including, but not limited to, marketability or suitability for a specific purpose. The user of the NVIDIA Bright Cluster Manager product shall accept the full risk for the quality or performance of the product. Should the product malfunction, the costs for repair, service, or correction will be borne by the user of the NVIDIA Bright Cluster Manager product. No copyright owner or third party who has modified or distributed the program as permitted in this license shall be held liable for damages, including general or specific damages, damages caused by side effects or consequential damages, resulting from the use of the program or the un-usability of the program (including, but not limited to, loss of data, incorrect processing of data, losses that must be borne by you or others, or the inability of the program to work together with any other program), even if a copyright owner or third party had been advised about the possibility of such damages unless such copyright owner or third party has signed a writing to the contrary.

1

Introduction To Containerization On NVIDIA Bright Cluster Manager

Containerization is a technology that allows processes to be isolated by combining *cgroups*, *Linux namespaces*, and *(container) images*.

- Cgroups are introduced in section 7.10 on workload management of the *Administrator Manual*
- Linux namespaces represent independent spaces for different operating system facilities: process IDs, network interfaces, mount points, inter-process communication resources and others. Cgroups and namespaces allow processes to be isolated from each other by separating the available resources as much as possible.
- A container image is a component of a container, and is a file that contains one or several layers. The layers cannot be altered as far the container is concerned, and a snapshot of the image can be used for other containers. A union file system is used to combine these layers into a single image. Union file systems allow files and directories of separate file systems to be transparently overlaid, forming a single coherent file system.

Cgroups, namespaces and image are the basis of a container. When the container is created, then a new process can be started within the container. Containerized processes running on a single machine all share the same operating system kernel, so they start immediately, without the delay of requiring a kernel to first boot up. No process is allowed to change the layers of the image. All changes are applied on a temporary layer created on top of the image, and these changes are destroyed when the container is removed.

There are several ways to manage the containers, but the most powerful approaches use Docker, also known as Docker Engine, and Kubernetes.

Docker manages containers on individual hosts, while Kubernetes manages containers across a cluster. The cluster manager integrates both of these solutions, so that setup, configuration and monitoring of containers becomes an easily-managed part of the cluster manager.

Chapter 2 describes how Docker integration with Bright works.

Chapter 3 covers how Docker registries are integrated.

Chapter 4 covers Kubernetes integration.

Chapter 5 covers Kubernetes application configuration and groups of Kubernetes applications.

Chapter 6 covers Kubernetes operators, which are a way to manage Kubernetes cluster applications.

Chapter 7 covers Kubernetes deployment on edge sites.

Chapter 8 describes the use of Singularity, which is an application containerization tool. Singularity is designed to execute containers as if they are just native applications on a host computer, and to work with HPC.

Chapter 9 describes OpenShift, which is Red Hat's container manager.

2

Docker Engine

Docker integration with NVIDIA Bright Cluster Manager 9.2 for Docker version 20.10.17 is available at the time of writing of this section (March 2022) on the `x86_64` architecture for all the Bright-supported Linux distributions. For a more up-to-date status, the features matrix at `https://support.brightcomputing.com/feature-matrix/` can be checked.

Docker Engine (or just Docker) is a tool for container management. Docker allows containers and their images to be created, controlled, and monitored on a host using Docker command line tools or the Docker API.

Swarm mode, which allows containers to spawn on several hosts, is not formally supported by NVIDIA Bright Cluster Manager 9.2. This is because NVIDIA Bright Cluster Manager 9.2 provides Kubernetes for this purpose instead.

Docker provides a utility called `docker`, and two daemons called `containerd` (the default provided by the cluster manager), and `dockerd`. Additional functionality includes pulling the container image from a specific image registry (Chapter 3), configuring the container network, setting `systemd` limits, and attaching volumes.

2.1 Docker Setup

The cluster manager provides the `cm-docker` package. The package includes the following components:

- Docker itself, that provides an API and delegates the container management to Containerd;
- Containerd runtime, that manages OCI images and OCI containers (via runC);
- runC, a CLI tool for spawning and running containers according to the OCI specification runtime;
- docker-py, a Python library for the Docker API.

Typically, however, the administrator is expected to simply run the `cm-docker-setup` utility, which is provided by the cluster manager's `cm-setup` package. Running `cm-docker-setup` takes care of the installation of the `cm-docker` package and also takes care of Docker setup. If run without options then the utility starts up a TUI dialog (figure 2.1).

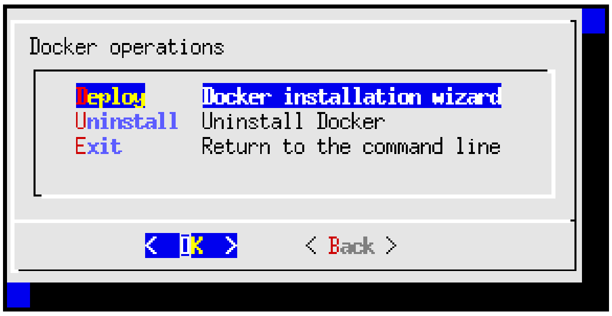

Figure 2.1: `cm-docker-setup` TUI startup

The `cm-docker-setup` utility asks several questions, such as which Docker registries are to be used, what nodes Docker is to be installed on, whether the NVIDIA container runtime should be installed, and so on. If `cm-docker-setup` is used with the `-c` option, and given a YAML configuration file <*YAMLfile*>, then a runtime configuration is loaded from that file. The YAML file is typically generated and saved from an earlier run.

When the questions in the TUI dialog have been answered and the deployment is carried out, the utility:

- installs the `cm-docker` package, if it has not been installed yet
- then assigns the DockerHost role to the node categories or head nodes that were specified
- adds health checks to the cluster manager monitoring configuration
- performs the initial configuration of Docker.

The regular nodes on which Docker is to run, are restarted by the utility, if needed. The restart operation provisions the updated images from the image directory onto the nodes.

The `cm-docker` package also includes a modules environment file, which must be loaded in order to use the `docker` command. The modules environment and modules are introduced in section 2.2 of the *Administrator Manual*.

By default only the administrator can run the `docker` commands after setup (some output ellipsized):

Example

```
[root@bright92 ~]# ssh node001
[root@node001 ~]# module load docker
[root@node001 ~]# docker info
Containers: 0
Images: 0
...
 Docker Root Dir: /var/lib/docker
 Debug Mode: false
 Registry: https://index.docker.io/v1/
 Labels:
 Experimental: false
 Insecure Registries:
  127.0.0.0/8
 Registry Mirrors:
  https://harbor-proxy.brightcomputing.com/
 Live Restore Enabled: false
[root@node001 ~]#
```

and the `hello-world` image can be run as usual with:

Example

```
[root@node001 ~]# docker run hello-world
Unable to find image 'hello-world:latest' locally
latest: Pulling from library/hello-world
2db29710123e: Pull complete
Digest: sha256:cc15c5b292d8525effc0f89cb299f1804f3a725c8d05e158653a563f15e4f685
Status: Downloaded newer image for hello-world:latest

Hello from Docker!
This message shows that your installation appears to be working correctly.
...
```

Or, for example, importing and running Apache containers with Docker may result in the following output:

Example

```
[root@node001 ~]# module load docker
[root@node001 ~]# docker run httpd & docker run httpd &
... runs a couple of Apache containers...
[root@node001 ~]# docker container ls
CONTAINER ID  IMAGE   COMMAND             CREATED         STATUS         PORTS   NAMES
acdbe2f3667b  httpd   "httpd-foreground"  13 seconds ago  Up 11 seconds  80/tcp  quizzical_bhabha
64787a8524dd  httpd   "httpd-foreground"  13 seconds ago  Up 11 seconds  80/tcp  funny_hypatia
...
[root@node001 ~]#
```

Using Docker directly means being root on the host. It is rarely sensible to carry out regular user actions as the root user at all times.

So, to make Docker available to regular users, Kubernetes provides a user management layer and restrictions.

After Docker has been installed, Kubernetes can be set up to allow regular user access to the Docker containers as covered in Chapter 4. It is a best practice for regular users to use Kubernetes instead of Docker commands directly.

2.2 Integration With Workload Managers

The cluster manager does not provide integration of Docker with workload managers. The administrator can however tune the workload managers in some cases to enable Docker support.

- LSF – An open beta version of LSF with Docker support is available from the IBM web site. This LSF version allows jobs to run in Docker containers, and monitors the container resources per job.
- PBS Professional – Altair provides a hook script that allows jobs to start in Docker containers. Altair should be contacted to obtain the script and instructions.

2.3 DockerHost Role

When `cm-docker-setup` is executed, the DockerHost role is assigned to nodes or categories. The DockerHost role is responsible for Docker service management and configuration.

From `cmsh`, the configuration parameters can be managed from the `Docker::Host` role:

Example

```
[root@bright92 ~]# cmsh
[bright92]% category use default
[bright92->category[default]]% roles
[bright92->category[default]->roles]% assign docker::host
[bright92->category*[default*]->roles*[Docker::Host*]]% show
Parameter                        Value
-------------------------------- ------------------------------------------------
Name                             Docker::Host
Revision
Type                             DockerHostRole
Add services                     yes
Provisioning associations        <0 internally used>
Spool                            /var/lib/docker
Tmp dir                          $spool/tmp
Enable SELinux                   yes
Default Ulimits
Debug                            no
Log Level                        info
Bridge IP
Bridge
MTU                              0
API Sockets                      unix:///var/run/docker.sock
Iptables                         yes
User Namespace Remap
Insecure Registries
Enable TLS                       no
Verify TLS                       no
TLS CA
TLS Certificate
TLS Key
Certificates Path                /etc/docker
Storage Backends                 <0 in submode>
Containerd Socket
Runtime                          runc
Options
[bright92->category*[default*]->roles*[Docker::Host*]]%
```

The Docker host parameters that CMDaemon can configure in the DockerHost role, along with a description, are shown in table 2.1:

Parameter	Description
`Add services`*	Add services to nodes belonging to this node. Care must be taken if setting this to `no`. (default: `yes`)
`Spool`	Root of the Docker runtime (default: `/var/lib/docker`)
`Tmp dir`	Location for temporary files. Default: $<*spool*>`/tmp`, where $<*spool*> is replaced by the path to the Docker runtime root directory

...continues

...continued

Parameter	Description
`Enable SELinux`*	Enable selinux support in Docker daemon (default: `yes`)
`Default Ulimits`	Set the default ulimit options for all containers
`Debug`*	Enable debug mode (default: `no`)
`Log Level`	Set the daemon logging level. In order of increasing verbosity: `fatal`, `error`, `warn`, `info`, `debug`. (default: `info`)
`Bridge IP`	Network bridge IP (not defined by default)
`Bridge`	Attach containers to a network bridge (not defined by default)
`MTU`	Set the containers network MTU, in bytes (default: `0`, which does not set the MTU at all)
`API Sockets`	Daemon socket(s) to connect to (default: `unix:///var/run/docker.sock`)
`Iptables`*	Enable iptables rules (default: `yes`)
`User Namespace Remap`	User/Group setting for user namespaces (not defined by default). It can be set to any of `<UID>`, `<UID:GID>`, <*username*>, <*username:groupname*>. If it is used, then `user_namespace.enable=1` must be set in the kernel options for the relevant nodes, and those nodes must be rebooted to pick up the new option.
`Insecure Registries`	If registry access uses HTTPS but does not have proper certificates distributed, then the administrator can make Docker accept this situation by adding the registry to this list (empty by default)
`Enable TLS`*	Use TLS (default: `no`)
`Verify TLS`*	Use TLS and verify the remote (default: `no`)
`TLS CA`	Trust only certificates that are signed by this CA (not defined by default)

...continues

...continued

Parameter	Description
TLS Certificate	Path to TLS certificate file (not defined by default)
TLS Key	Path to TLS key file (not defined by default)
Certificates Path	Path to Docker certificates (default: /etc/docker)
Storage Backends	Docker storage back ends. Storage types can be created and managed, in a submode under this mode. The available types are described in table 2.2. Each of these storage types has options that can be set from within the submode.
Containerd Socket	Path to the containerd socket (default: not used)
Runtime	Docker runtime
Options	Additional parameters for docker daemon

* Boolean (takes yes or no as a value)

Table 2.1: Docker::Host *role options*

2.4 Iptables

By default iptables rules have been added to nodes that function as a Docker host, to let network traffic go from the containers to outside the pods network. If this conflicts with other software that uses iptables, then this option can be disabled. For example, if the docker::host role has already been assigned to the nodes via the default category, then the iptables rules that are set can be disabled by setting the iptables parameter in the Docker::Host role to no:

Example

```
[root@bright92 ~]# cmsh
[bright92]% category use default
[bright92->category[default]]% roles
[bright92->category[default]->roles]% use docker::host
[bright92->category[default]->roles[Docker::Host]]% set iptables no
[bright92->category*[default*]->roles*[Docker::Host*]]% commit
```

2.5 Storage Backends

A core part of the Docker model is the efficient use of containers based on layered images. To implement this, Docker provides different storage back ends, also called storage drivers. These storage back ends rely heavily on various filesystem features in the kernel or volume manager. Some storage back ends perform better than others, depending on the circumstances.

The default storage back end configured by cm-docker-setup is overlay2. Storage back ends supported by Docker are listed in table 2.2:

Technology	Description	Backend Name
`OverlayFS`	This is a modern union filesystem. It is the preferred storage driver for recent Docker versions. It has been in the mainline Linux kernel since version 3.18, with additional improvements for Docker in version 4.0. All of the distributions that the cluster manager supports have backported the kernel changes needed for this to work.	`overlay2`
`Device Mapper`	Deprecated since Docker Engine 18.09. It is a kernel-based framework that has been included in the mainline Linux kernel since version 2.6.9. It underpins many advanced volume management technologies on Linux. The driver stores every image and snapshot on its own virtual device, and works at the block level rather than the file level. • A loopback mechanism can be implemented using `loop-lvm` mode. This allows files on a local disk to be managed as if they are on a physical disk or block device. This is simpler than the thin pool mode, but is strongly discouraged for production use. In the cluster manager this mode is implemented by selecting the option `loopback (testing only)`. This is selected in the storage back end selection screen of the `cm-docker-setup` installation. • A thin pool mode can be implemented using `direct-lvm` mode. This uses a logical volume as a thin pool to use as backing for the storage pool, and uses a spare block device. Configuring this is normally more involved. In the cluster manager this mode is implemented by selecting the option `block (production ready)`. This is selected in the storage back end selection screen of the `cm-docker-setup` installation session. Device mapper options for these modes are described in Table 2.3	`devicemapper`

...continues

...continued

Technology	Description	Backend Name
AUFS	This was the first storage back end that Docker used. AUFS is not included in the mainline Linux kernel. Out of the distributions that NVIDIA Bright Cluster Manager 8.2 supports, it is only Ubuntu that supports it.	aufs

Table 2.2: *Docker storage back ends*

The `docker info` command, amongst many other items, shows the storage driver and related settings that are being used in Docker:

Example

```
[root@bright92 ~]# module load docker
[root@bright92 ~]# docker info

Client:
 Context:    default
 Debug Mode: false

Server:
 Containers: 18
  Running: 8
  Paused: 6
  Stopped: 4
 Images: 1
 Server Version: 20.10.9
 Storage Driver: overlay2
  Backing Filesystem: xfs
  Supports d_type: true
  Native Overlay Diff: true
  userxattr: false
 Logging Driver: json-file
 Cgroup Driver: cgroupfs
 Cgroup Version: 1
 Plugins:
  Volume: local
  Network: bridge host ipvlan macvlan null overlay
  Log: awslogs fluentd gcplogs gelf journald json-file local logentries splunk syslog
...
```

Docker data volumes are not controlled by the storage driver. Reads and writes to data volumes bypass the storage driver. It is possible to mount any number of data volumes into a container. Multiple containers can also share one or more data volumes.

More information about Docker storage back ends is available at `https://docs.docker.com/engine/userguide/storagedriver`.

2.5.1 Device Mapper Driver Settings Support

The cluster manager supports device mapper driver settings more explicitly than the other driver back end settings.

By default the device mapper storage back end is added automatically, and can be configured in the `storagebackends` submode of the DockerHost role:

Example

```
[bright92->device[bright92]->roles[docker::host]]% storagebackends
[bright92->device[bright92]->roles[docker::host]->storagebackends]% use devicemapper
[bright92->device[bright92]->roles[docker::host]->storagebackends[devicemapper]]% show
Parameter                        Value
-------------------------------- --------------------------------------------
Name                             devicemapper
Revision
Type                             devicemapper
Loop Data Size
Loop Metadata Size
Loop Device Size
Pool Device
Filesystem                       xfs
Block Size                       64K
Blk Discard                      yes
Mkfs Arguments
Mount Options
```

The parameters that are used in the Docker device mapper back end are described in table 2.3:

Parameter	Description	Option to `docker`
`Blk Discard`*	Enables or disables the use of `blkdiscard` when removing device mapper devices (default: `yes`)	`dm.blkdiscard`
`Block Size`	Custom blocksize to use for the thin pool (default: `64kB`)	`dm.blocksize`
`Filesystem`	Filesystem type to use for the base device (default: `xfs`)	`dm.fs`
`Loop Data Size`	Size to use when creating the loopback file for the data virtual device which is used for the thin pool (default: `100GB`)	`dm.loopdatasize`
`Loop Device Size`	Size to use when creating the base device, which limits the size of images and container (not set by default)	`dm.basesize`
`Loop Metadata Size`	Size to use when creating the loopback file for the metadata device which is used for the thin pool (default: `2GB`)	`dm.loopmetadatasize`
`Mkfs Arguments`	Extra mkfs arguments to be used when creating the base device	`dm.mkfsarg`
`Mount Options`	Extra mount options used when mounting the thin devices	`dm.mountopt`
`Pool Device`	Custom block storage device to use for the thin pool (not set by default)	`dm.thinpooldev`

...continues

...continued

Parameter	Description	Option to `docker`

* Boolean (takes `yes` or `no` as a value)

Table 2.3: *Device mapper back end Docker options*

For back end driver storage settings other than device mapper, such as `AUFS` or `OverlayFS`, settings can be added as options if needed. In `cmsh` this can be done by setting the `options` parameter in the `storagebackend` submode under the `docker::host` role.

2.6 Docker Monitoring

When `cm-docker-setup` runs, it configures and runs the following Docker health checks:

1. makes a test API call to the endpoint of the Docker daemon
2. checks containers to see that none is in a `dead` state

The Docker daemon can be monitored outside of the cluster manager with the usual commands directly.

The cluster manager ways to manage or check on Docker include the following:

In CMDaemon, the `docker` service can be checked:

Example

```
[bright92->device[node001]->services]% list
Service (key)            Monitored  Autostart
------------------------ ---------- ----------
docker                   yes        yes
nslcd                    yes        yes
[bright92->device[node001]->services]% show docker
Parameter                        Value
-------------------------------- ------------------------------------------------
Revision
Service                          docker
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
```

The `docker0` interface statistics can be checked within the `nodeoverview` output:

Example

```
[bright92->device[node001]]% nodeoverview
...

Interface    Received     Transmitted
------------ ------------ ------------
docker0      16.0 KiB     3.16 KiB
ens3         492 MiB      72.5 MiB
ens4         0 B          0 B
...
```

The measurable `Docker` checks if the `docker` service is healthy.

Example

```
[bright92->device[node001]]% dumpmonitoringdata -1h now Docker
Timestamp                  Value      Info
-------------------------- ---------- ----------
2021/11/29 11:52:44.146    PASS
2021/11/30 18:28:44.146    PASS
```

2.7 Docker Setup For NVIDIA

NVIDIA GPU Cloud (NGC) is a cloud platform that runs on NVIDIA GPUs. NGC containers are lightweight containers that run applications on NVIDIA GPUs. The applications are typically HPC, machine learning, or deep learning applications.

An NGC can be set up to run NGC containers from the registry `http://ngc.nvidia.com`.

Docker can be configured as an NGC running NGC containers by using the NVIDIA Container Toolkit.

The cluster manager package provided for this is: `cm-nvidia-container-toolkit`.

One way to install and deploy this package is as part of the Docker installation, when running `cm-docker-setup` (section 2.1), where the cluster administrator selects `yes` as the answer to the request: ``` ``Do you want to install the NVIDIA Runtime for Docker''. ```

Alternatively, if Docker has already been installed via `cm-docker-setup`, and if the package has not been installed, then it can be installed via the package manager, `yum` or `apt`. The toolkit has to be running on the compute nodes that have GPUs, which means that the installation must go to the appropriate node image (section 11.4 of the *Administrator Manual*). For example, if the appropriate image is `gpu-image`, then the package manager command for RHEL-based distributions would be:

Example

```
# yum install --installroot=/cm/images/gpu-image cm-nvidia-container-toolkit
```

The nodes that use that GPU image can then be rebooted to pick up the new package.

The GPU status can then be printed with the NVIDIA system management interface command. For example, if the image has been picked up by `node001`:

Example

```
[root@bright92 ~]# ssh node001
Last login: Thu Dec  2 09:24:03 2021 from bright92.cm.cluster
[root@node001 ~]# module load docker
[root@node001 ~]# docker run --runtime=nvidia --rm nvidia/cuda:11.4-base nvidia-smi
Unable to find image 'nvidia/cuda:11.4.0-base' locally
11.4.0-base: Pulling from nvidia/cuda
...
Digest: sha256:f0a5937399da5e4efb37fd7b75beb8c484b84dc381243c4b81fc5f9fcad42b66
Status: Downloaded newer image for nvidia/cuda:11.4.0-base
Mon Mar  7 17:30:48 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 440.33.01    Driver Version: 440.33.01    CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|===============================+======================+======================|
|   0  Tesla K40c          On   | 00000000:00:06.0 Off |                  Off |
| 23%   32C    P8    22W / 235W |      0MiB / 12206MiB |      0%      Default |
```

```
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                       GPU Memory |
|  GPU       PID   Type   Process name                             Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
[root@node001 ~]# logout
Connection to node001 closed.
```

The available CUDA Docker images can be found at `https://hub.docker.com/r/nvidia/cuda`.

3

Docker Registries

When a user creates a new container, an image specified by the user should be used. The images are kept either locally on a host, or in a registry. The image registry can be provided by a cloud provider or locally.

3.1 Docker And Harbor Registries: Introduction

3.1.1 Docker Hub, A Remote Registry

By default, Docker searches for images in Docker Hub, which is a cloud-hosted public and private image registry. Docker Hub serves a huge collection of existing images that users can make use of. Every user is allowed to create a new account, and to upload and share images with other users. Using the Docker client, a user can search for already-published images, and then pull them down to a host in order to build containers from them.

When an image is found in the registry, the Docker client verifies if the latest version of the image has already been downloaded. If not, then it downloads the images, and stores them locally. It also tries to synchronize them when a new container is created. When the latest image is downloaded, Docker creates a container from the image layers that are formatted to be used by a union file system. Docker can make use of several union file system variants, including AUFS, btrfs, vfs, and DeviceMapper.

3.1.2 Local Image Registry Options: Classic Docker Registry Vs Harbor

Besides using Docker Hub for the image registry, the administrator can also install a local image registry on the cluster. The cluster manager provides two ways to integrate such a local registry with the cluster, based on existing open source projects:

- The first one is the classic docker registry provided by Docker Inc, and can be useful if the registry is used by trusted users.
- The second registry, Harbor, developed by VMware and introduced in NVIDIA Bright Cluster Manager version 8.1-5, provides additional features such as security and identity management, and is aimed at the enterprise.

Both options can be installed with the `cm-container-registry-setup` utility, which comes with the cluster manager's `cm-setup` package.

3.2 Docker And Harbor Registries: Setup And Configuration

Docker Registry and Harbor can be installed via the `cm-container-registry-setup` command-line utility. They can also be installed via Bright View in NVIDIA Bright Cluster Manager for versions beyond 8.1-6 as follows:

- The Docker Registry Deployment Wizard is launched via the clickpath:
  `Containers→Docker→Docker Registry Wizard`

- Either Docker Registry, or Harbor, should be chosen as a registry.
- A single node is ticked for the deployment. The address, port, and the root directory for storing the container images are configured. If the head node is selected for Harbor, then the setup later on asks to open the related port on the head node. This is to make Harbor and the Harbor UI externally accessible.
- In the summary page, if the `Ready for deployment` box is ticked, then the administrator can go ahead with deploying the registry.
- When the deployment is complete, the Docker Registry becomes ready for use. Harbor can take a few additional minutes to be ready for use.

Similar to the case of etcd nodes (section 4.2), nodes that run Harbor or Docker Registry have the `datanode` option (page 240 of the *Administrator Manual*) when installed by the cluster manager utilities. The option helps prevent the registry being wiped out by accident, and is added when the `cm-container-registry-setup` utility is used to install Harbor or Docker Registry. This extra protection is put into place because if a registry is wiped out, then the state of images in the container becomes incoherent and cannot be restored.

Harbor UI

If the head node is where Harbor is to be installed, and is to be made externally accessible, then the Harbor UI can be accessed at `https://`<*head node hostname*>`:9443`.

If a different node is used for Harbor to be installed, then the related port must be forwarded locally.

Harbor can be logged into by default with the `admin` user and the default `Harbor12345` password.

It is recommended to change that password after the first login.

Dealing With A Pre-existing Kubernetes Or Harbor Installation

Since Harbor uses Docker internally, and because Kubernetes customizes Docker networking, it means that nodes where Kubernetes is running cannot be reused for Harbor, and that nodes where Harbor is running cannot be reused for Kubernetes.

3.2.1 Docker Registry Daemon Configuration Using The Docker Registry Role

The Docker Registry role is used to configure and manage the `docker-registry` daemon, and its parameters are described in table 3.1:

Parameter	Description
`Domain`	Main domain name (default: hostname of the node)
`Alt Domains`	Alternative domain names (default: FQDN of the node)
`Port`	Port (default: `5000`)
`Spool Dir`	Spool directory (default: `/var/lib/docker-registry`)

...*continues*

...continued

Parameter	Description

* Boolean (takes `yes` or `no` as a value)

Table 3.1: *Docker Registry role parameters*

The values stored in the Docker Registry role are not supposed to be changed, but they are useful for the uninstall procedure, and also as a record of the settings for the administrator.

```
[bright92->device[bright92]->roles[generic::docker_registry]]% environments
[bright92->device[bright92]->roles[generic::docker_registry]->environments]% list
Name (key)       Value                     Node Environment
---------------- ------------------------- ----------------
alt_domains      node001.cm.cluster        no
domain           node001                   no
port             5000                      no
spool_dir        /var/lib/docker-registry  no
```

Further details on the `docker-registry` daemon can be found at `https://github.com/docker/distribution`.

3.2.2 Harbor Daemon Configuration Using The Harbor Role

The Harbor role is used to configure and manage the `harbor` daemon. The parameters of the role are described in table 3.2:

Parameter	Description
`Domain`	Main domain name (default: hostname of the node)
`Alt Domains`	Alternative domain names (default: FQDN of the node)
`Port`	Port (default: `9443`)
`Spool Dir`	Spool directory (default: `/var/lib/harbor`)
`Default Password`	Default password of the Harbor admin user (default: `Harbor12345`)
`DB Password`	Password of the Harbor database (default: randomly generated)

* Boolean (takes `yes` or `no` as a value)

Table 3.2: *Harbor role parameters*

The values stored in the Harbor role are not supposed to be changed, but they are useful for the uninstall procedure, and also as reminder of the settings for the administrator.

```
[bright92->device[bright92]->roles[generic::harbor]]% environments
[bright92->device[bright92]->roles[generic::harbor]->environments]% list
Name (key)         Value                      Node Environment
------------------ -------------------------- ----------------
alt_domains        harbor,node001.cm.cluster  no
db_password        <generated password>       no
domain             node001                    no
external_network   True                       no
```

port	9443	no
redirect_port	65535	no
spool_dir	/var/lib/harbor	no

Further details on Harbor can be found at https://vmware.github.io/harbor.

4

Kubernetes

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

- scale applications on the fly
- seamlessly update running services
- optimize hardware usage by using only the resources that are needed

The cluster manager provides the administrator with the required packages, allows Kubernetes to be set up on a cluster, and manages and monitors Kubernetes. More information about the design of Kubernetes, its command line interfaces, and other Kubernetes-specific details, can be found at the official online documentation at `https://kubernetes.io/docs/`.

Kubernetes integration with NVIDIA Bright Cluster Manager 9.2 for Kubernetes v1.24 is available at the time of writing of this paragraph (November 2022) on the `x86_64` architecture for all the Bright-supported Linux distributions, except for SLES12. For a more up-to-date status, the features matrix at `https://support.brightcomputing.com/feature-matrix/` can be checked.

4.1 Reference Architecture

A reference architecture for Kubernetes in the cluster manager comprises:

- *etcd nodes*: An etcd cluster—the Kubernetes distributed key-value storage—runs on regular nodes only, and uses an odd number (1, 3, 5 ...) of nodes.
- *master nodes*: Kubernetes master units run on head or compute nodes. 2 (or 3) are recommended for High Availability (HA). In a Bright HA configuration, both or none of the head nodes should be selected. That is, it must not run on only one head node of an HA configuration.
- *worker nodes*: Kubernetes worker units run on regular nodes only.

Since NVIDIA Bright Cluster Manager version 8.2, multiple clusters of Kubernetes can be deployed. In such a configuration the same node cannot be shared across different Kubernetes clusters.

A Kubernetes API server proxy based on NGINX runs on every node, except for on nodes that run `etcd`. The proxy also runs on the head node(s).

Because of the server proxy, a port is reserved on the head node(s) for every Kubernetes cluster. This is required for Kubernetes HA (section 4.1.1), and it also allows `kubectl` and other tools such as Helm to be used from the head node, to access each Kubernetes cluster.

4.1.1 Kubernetes HA

For a Kubernetes HA setup the minimum node requirements are:

- at least 3 nodes for Etcd
- at least 2 nodes for Kubernetes Master

In an average cluster, Bright Computing recommends 3 nodes for the etcd cluster, and 3 nodes for the Kubernetes master.

Even without a Kubernetes master in an HA configuration, there is no downtime for existing pods running on the worker nodes. The worker nodes will still continue to work.

However, Kubernetes HA is needed to be able to schedule tasks, spawn new pods, and in general keep the cluster in the desired state.

4.2 Kubernetes Setup

The cluster manager allows the deployment of the several Kubernetes versions:

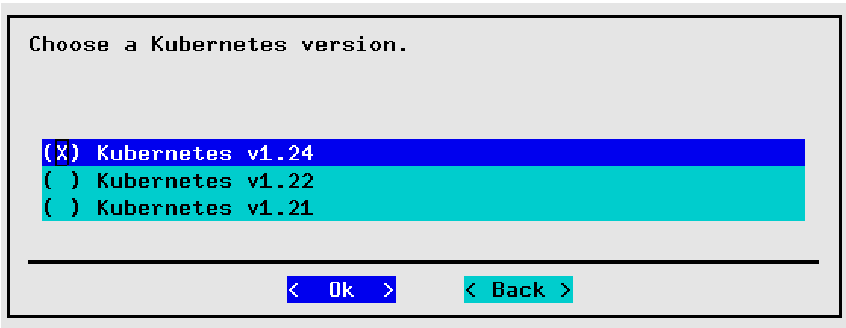

Figure 4.1: Kubernetes Setup TUI Session: Version selection screen

Some Kubernetes versions on older Linux kernels allow selection of the deprecated `docker` runtime instead of `containerd`:

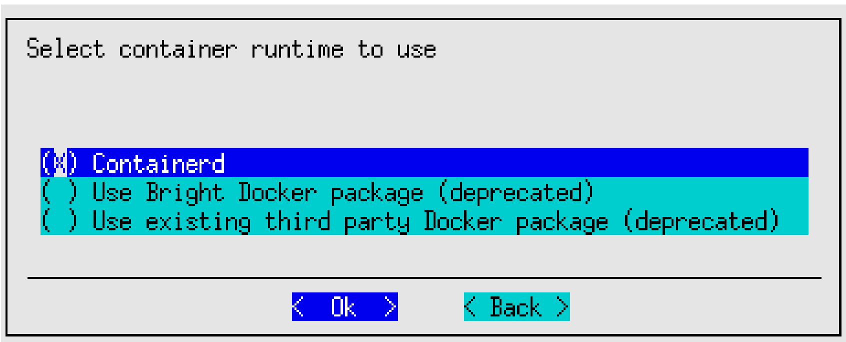

Figure 4.2: Kubernetes Setup TUI Session: Container Runtime selection screen

The container runtime options at the time of writing of this section (March 2024) are shown in table 4.1.

Kubernetes version	Package	Container runtime options
1.21	cm-kubernetes121	containerd, Bright docker (deprecated), existing third party docker (deprecated)
1.22	cm-kubernetes122	containerd, Bright docker (deprecated), existing third party docker (deprecated)
1.23	cm-kubernetes123	containerd, Bright docker (deprecated), existing third party docker (deprecated)
1.24	cm-kubernetes124	containerd
1.25	cm-kubernetes125	containerd
1.26	cm-kubernetes126	containerd

Table 4.1: *Kubernetes versions*

The cluster manager provides the following Kubernetes-related packages:

- The `conntrack` and `nginx` packages: These are always installed on the head node(s) and on the master and worker node(s).
- The `cm-etcd` package is installed on the nodes selected for etcd. In a similar way to the case of Harbor or Docker Registry (section 3.2), the nodes that run etcd are protected by the cluster manager with the `datanode` option (page 240 of the *Administrator Manual*). For etcd nodes, the option is added during the `cm-kubernetes-setup` installation. As in the case for the registries, the datanode option is set in order to help prevent the administrator from wiping out the existing state of etcd nodes. Wiping out the state of etcd nodes means that the Kubernetes cluster becomes incoherent and that it cannot be restored to where it was just before the etcd nodes were wiped. The etcd version installed by the cluster manager package is 3.5.0.

4.2.1 Kubernetes Networking

Early on during the wizard (figure 4.3), a name for the cluster is requested. The wizard pre-fills it with `default`, but this should not be confused with the Kubernetes `default` namespace. Here, the name is used instead, inside the cluster manager, to identify the cluster, configuration files, and other resources such as module files.

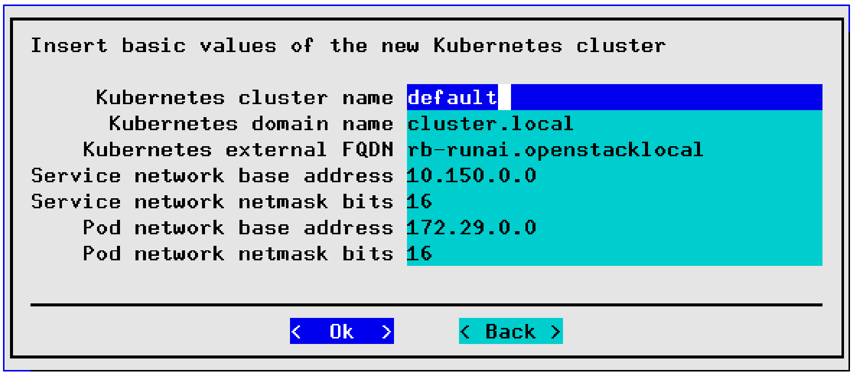

Figure 4.3: Kubernetes Setup TUI Session: Networking selection screen

This screen also allows the following important choices:

- `Kubernetes external FQDN`: This is the FQDN that is placed as one of the subjects in the public-facing certificates generated for this Kubernetes cluster.
- `Service network base address` and `Service network netmastk bits`: These define the CIDR for the service network. The wizard pre-fills the fields. It also tries to avoid pre-filling them with overlapping network ranges, by taking any existing network known to the cluster manager into account.
- `Pod network base address` and `Pod network netmask bits`: These define the CIDR for the pod network. The wizard pre-fills these. It also tries to avoid pre-filling them with overlapping network ranges, by taking any existing network known to The cluster manager into account. By default, entire /24 network ranges are assigned to individual Kubernetes nodes from the pod CIDR.

The packages are installed automatically from the repository when the administrator runs `cm-kubernetes-setup` from the command line.

The log file produced by the setup can be found in `/var/log/cm-kubernetes-setup.log`.

4.2.2 Kubernetes Core Add-ons

During setup, some critical add-on components such as CoreDNS and Calico are automatically deployed in the `kube-system` namespace. In the cluster manager all add-ons are treated as Kubernetes applications (Chapter 5), and belong to the default app group `system`.

A `cmsh` treeview illustrating the hierarchy to access these applications is:

```
[cmsh]
  |-- ...
  |
  |-- kubernetes[default]
  |                `-- appgroups[system]
  |                               `-- applications
  |-- ...
```

CoreDNS

CoreDNS is the DNS server add-on for internal service discovery. It reads the IP addresses of services and pods from Etcd, and resolves domain names for them. If a domain name is not found because the domain is external to the Kubernetes cluster, then CoreDNS forwards the request to the main DNS server. The cluster manager uses CoreDNS version 1.9.4 with Kubernetes version 1.24.

Calico

Calico is a modern SDN (Software-Defined Networking) add-on to manage a cluster-wide network for pods. Calico uses an agent called Felix to run on each node as a pod. The cluster manager uses Calico version 3.10.

Further details on Calico can be found at `https://docs.projectcalico.org/`.

If the Kubernetes cluster is composed of more than 50 nodes, then the Calico component Typha is also automatically deployed for better scalability. The number of Typha replicas is calculated by allocating one Typha replica per 150 nodes, with a minimum of 3 (above 50 nodes) and a maximum of 20.

If an initial deployment of the Kubernetes cluster has fewer than 50 nodes, but nodes are then added to the Kubernetes cluster so that the 50 node threshold is exceeded, then Typha is not automatically enabled. In this case, Typha can be enabled manually via `cmsh` as follows:

Example

```
[bright92->kubernetes[default]->appgroups[system]->applications[calico]]% environment
[bright92->kubernetes[default]->appgroups[system]->applications[calico]->environment]% list
Name (key)             Value            Nodes environment
---------------------- ---------------- ------------------
calico_typha_replicas  0                no
calico_typha_service   none             no
head_node_internal_ip  10.141.255.254   no
[bright92->kubernetes...[calico]->environment]% set calico_typha_service value calico-typha
[bright92->kubernetes*...[calico*]->environment*]% set calico_typha_replicas value 3
[bright92->kubernetes*[default*]->appgroups*[system*]->applications*[calico*]->environment*]% commit
[bright92->kubernetes[default]->appgroups[system]->applications[calico]->environment]%
```

4.2.3 Kubernetes Optional Add-ons

The following add-ons are installed by default unless otherwise noted. However, the user can choose to skip some or all of them during the setup.

Kubernetes Dashboard

Kubernetes Dashboard is the web user interface add-on for GUI cluster administration and metrics visualization. The cluster manager uses Kubernetes Dashboard version 2.0.

There are two ways to access the dashboard:

- Using `kubectl proxy` and accessing `http://localhost:8001/api/v1/namespaces/kubernetes-dashboard/services/https:kubernetes-dashboard:/proxy/` To use the proxy, `kubectl` must be set up locally (section 9.3.2 of the *User Manual*).
- Users on an external network can log in to `kubectl` or Kubernetes Dashboard by following the procedures described in section 4.17.

Kubernetes Metrics Server

The Kubernetes Metrics Server is an add-on that is a replacement for Heapster. It aggregates metrics, and provides container monitoring and performance analysis. It exposes metrics via an API. The cluster manager runs Metrics server version 1.0.0.

Helm

Helm is an add-on that manages *charts*, which are packages of pre-configured Kubernetes resources. The Helm component is installed and properly configured with the cluster manager's Kubernetes installation by default. It is initialized and ready for use for every Kubernetes user when the Kubernetes module is loaded. The cluster manager uses Helm version 3.

NGINX Ingress Controller

The official Kubernetes ingress controller add-on is built around the Kubernetes Ingress resource, using a ConfigMap to store the NGINX configuration. Ingress provides HTTP and HTTPS routes from outside a Kubernetes cluster to services within the cluster. Traffic routing is controlled by rules defined in the Ingress resource.

By default, the cluster manager provides an ingress for Kubernetes Dashboard during the `cm-kubernetes-setup` run, so that the Dashboard works. Port 30080 is the default that is set for the HTTP, and port 30443 is the default that is set for HTTPS.

These 2 ports are exposed on every Kubernetes node, both masters and workers.

The Ingress Controller is deployed as a `NodePort` which means it comes with a default range of possible port values of `30000-32767`.

NVIDIA Device Plugin For Kubernetes

An older alternative to the Kubernetes NVIDIA GPU Operator (section 6.3) is the NVIDIA device plugin for Kubernetes, which is an add-on option in the `cm-kubernetes-setup` run. By default it is not selected.

To be used, in addition to being selected, it requires that the NVIDIA GPU drivers are first installed (section 7.4 of the *Installation Manual*) on the head node and regular node container hosts.

This means that, as described in that section, the regular nodes on which the GPUs are located must have the `cuda-driver` and `cuda-dcgm` packages installed, inside the software image.

After booting up the container host regular nodes, the functioning of the DCGM (Data Center GPU Manager) tools should be checked by running `module load cuda-dcgm; dcgmi config --get`.

The plugin add-on can then be installed as a selection option in the `cm-kubernetes-setup` session that is run by the system administrator after the NVIDIA GPU drivers have been installed. The plugin is somewhat beta at the time of writing (March 2019).

If `cuda-dcgm` and `cuda-driver` are installed for GPUs on the head node, then to ensure that the head node GPUs are detected by CMDaemon, it is recommended to restart CMDaemon on the head node with `service cmdaemon restart`.

The plugin then allows the GPUs to be consumed from the containers. Overcommitting GPUs (sharing) is not possible from containers or pods. Multiple GPUs can be requested by the container. Some documentation on its use can be found at `https://github.com/NVIDIA/k8s-device-plugin`.

The cluster manager provides NVIDIA device plugin version 1.11

4.2.4 Kubernetes Setup From The Command Line

The `cm-kubernetes-setup` command line utility has the following usage synopsis:

```
[root@bright92 ~]# cm-kubernetes-setup -h
usage: Kubernetes Setup cm-kubernetes-setup [-c <config_file>]
                                            [--cluster CLUSTER_NAME]
                                            [--skip-docker] [--skip-reboot]
                                            [--skip-image-update]
                                            [--default-cni-bin-dir]
                                            [--add-user USERNAME_ADD]
                                            [--list-users]
                                            [--get-user GET_USER]
                                            [--modify-user USERNAME_MODIFY]
                                            [--remove-user USERNAME_REMOVE]
                                            [--namespace NAMESPACE]
                                            [--add-to-namespace]
                                            [--remove-from-namespace]
                                            [--role edit,admin,view,cluster-admin]
                                            [--runas-uid RUNAS_UID]
                                            [--runas-gids RUNAS_GIDS]
                                            [--user-paths USER_PATHS]
                                            [--allow-all-uids]
                                            [--operators OPERATORS]
                                            [--backup-permissions FILE]
                                            [--restore-permissions FILE]
                                            [--regenerate-certs]
                                            [--list-operators] [--psp]
                                            [--apparmor] [--disable-psp]
                                            [--update-addons] [--remove]
                                            [--yes-i-really-mean-it]
                                            [--remove-ceph-storage] [--pull]
                                            [--images IMAGES] [--nodes NODES]
                                            [--node-selector NODE_SELECTOR]
                                            [--allow-device-mapper] [-v]
                                            [--store-name-aliases]
                                            [--no-distro-checks] [--json]
                                            [--output-remote-execution-runner]
```

```
                                          [--on-error-action debug,remotedebug,undo,abort]
                                          [--skip-packages]
                                          [--min-reboot-timeout <reboot_timeout_seconds>]
                                          [--allow-running-from-secondary]
                                          [--dev] [-h]

optional arguments:
  --cluster CLUSTER_NAME
                        Name of the referred Kubernetes cluster
  -h, --help            Print this screen

common:
  Common arguments

  -c <config_file>      Load runtime configuration for plugins from a YAML config file

installing Kubernetes clusters:
  Flags for installing or managing Kubernetes clusters

  --skip-docker         Skip the Docker installation steps.
  --skip-reboot         Skip the reboot steps.
  --skip-image-update   Skip the image update steps.
  --default-cni-bin-dir
                        Setup CNI with default /opt/cni/bin directory

user management:
  Flags for adding a new user in a Kubernetes cluster

  --add-user USERNAME_ADD
                        Create a new user in a Kubernetes cluster
  --list-users          Get information about configured Kubernetes users
  --get-user GET_USER   Get information about configured Kubernetes users
  --modify-user USERNAME_MODIFY
                        Modify user in a Kubernetes cluster
  --remove-user USERNAME_REMOVE
                        Remove existing user from a Kubernetes cluster
  --namespace NAMESPACE
                        Specify namespace for user (--get-user, --modify-user) role binding
  --add-to-namespace    Indicate if permissions to manage namespace needs to be granted for a given user
                        (--modify-user)
  --remove-from-namespace
                        Indicate if permissions to manage namespace needs to be revoked for a given user
                        (--modify-user)
  --role edit,admin,view,cluster-admin
                        Specify role for the new (--add-user) and existing (--modify-user) role binding
                        (Default: edit). For 'cluster-admin' namespace flag is ignored
  --runas-uid RUNAS_UID
                        UID is allowed to be used in unprivileged pods (--add-user, --modify-user)
  --runas-gids RUNAS_GIDS
                        Comma-separated list of GIDs allowed to be used in unprivileged pods
                        (--add-user, --modify-user)
  --user-paths USER_PATHS
                        Comma-separated list of paths user is able to mount in pods
                        (--add-user, --modify-user)
  --allow-all-uids      Allow user to run processes in pods as any user (--add-user, --modify-user)
```

```
                        hostPath volumes will be disabled for such pods
  --operators OPERATORS
                        Comma-separated list of operators user has access to (--add-user, --modify-user)

backup or restore Permission Manager user configurations:
  Flag for managing permission manager user configuration

  --backup-permissions FILE
                        Save permissions to file
  --restore-permissions FILE
                        Restore permissions from file

regenerate Kubernetes certificates:
  Flag for regenerating the Kubernetes certificates

  --regenerate-certs    Regenerate certificates for a Kubernetes cluster

list available operators:
  Flag to list available Kubernetes operators

  --list-operators      List available Kubernetes operators

pod security policies:
  Flags for setting up Pod Security Policies

  --psp                 Setup PSP
  --apparmor            Use AppArmor (Default: false)

disable pod security policies:
  Flags for removing Pod Security Policies

  --disable-psp         Remove PSP

update kubernetes addons:
  Flags for updating Kubernetes addons

  --update-addons       Update Addons

removing Kubernetes clusters:
  Flags for removing a Kubernetes cluster

  --remove              Remove a Kubernetes cluster
  --yes-i-really-mean-it
                        Required for additional safety
  --remove-ceph-storage
                        Remove Kubernetes osd pool from Ceph cluster

pulling images to the nodes:
  Flags for pulling images to the nodes

  --pull                Pull images to the nodes
  --images IMAGES       Comma-separated list of images to pull (--pull)
  --nodes NODES         Comma-separated list of nodes to pull images to (--pull)
  --node-selector NODE_SELECTOR
                        Selector (label query) to filter on, supports '=', '==', and '!='.
```

```
                        (e.g. key1=value1,key2=value2) (--pull)

Docker storage backend specific:
  --allow-device-mapper
                        Allow to select DeviceMapper (DEPRECATED) storage in wizard

advanced:
  Various *advanced* configuration options flags.

  -v, --verbose         Verbose output
  --store-name-aliases  Store hostname aliases for head nodes (active and passive) and default category
  --no-distro-checks    Disable distribution checks based on ds.json
  --json                Use json formatting for log lines printed to stdout
  --output-remote-execution-runner
                        Format output for CMDaemon
  --on-error-action debug,remotedebug,undo,abort
                        Upon encountering a critical error, instead of asking the user for choice, setup
                        will do selected action.
  --skip-packages       Skip the any stages which install packages. Requires packages to be already installed.
  --min-reboot-timeout <reboot_timeout_seconds>
                        How long to wait for nodes to finish reboot (default and minimum allowed: 300 seconds).
  --allow-running-from-secondary
                        Allow to run the wizard from the secondary when it is the active head node.
  --dev                 Enables additional command line arguments
```

The `cm-kubernetes-setup` utility should be executed on the console.

Dealing With A Pre-existing Docker Installation

Docker is required for Kubernetes configured by NVIDIA Bright Cluster Manager. The setup wizard checks if Docker has been installed (page 5), and automatically installs Docker, if needed. However, if Docker has already been configured on the same category of nodes on which Kubernetes is to be installed, then the installation stops, because overriding the existing Docker configuration may not be what is wanted. To override the existing Docker configuration, Docker for that category should first be removed with the `cm-docker-setup --remove` command.

Dealing With A Pre-existing Etcd Cluster

Etcd is required by Kubernetes to store all the key-value states of the Kubernetes cluster. If no Etcd cluster is found, then the setup wizard prompts to deploy an Etcd cluster. If Etcd is already installed, or present from a previous Kubernetes cluster, then the setup wizard prompts on whether to use the existing Etcd cluster.

Dealing With A Pre-existing Harbor Installation

Since Harbor uses Docker internally, and because Kubernetes customizes Docker networking, it means that nodes where Harbor is running cannot be reused for Kubernetes.

4.2.5 Kubernetes Setup From A TUI Session

When the Kubernetes installation is carried out using `cm-kubernetes-setup` without any options, a TUI wizard starts up. The administrator can answer several questions in the wizard. Questions that are asked include questions about which of the node categories or which of the individual nodes should be configured to run the Kubernetes services. There are also questions about the service and pod networks parameters, the port numbers that will be configured for the daemons, whether to install specific add-ons, and so on. After the wizard has been completed, a configuration file can be saved that can be used to set up Kubernetes.

The configuration file can be deployed immediately from the wizard, or it can be deployed later by specifying it as an option to `cm-kubernetes-setup`, in the form `-c` <*file*>.

If no deployment has been carried out earlier, then the main operations screen of the wizard shows just two options, `Deploy` and `Exit`.

If deployment has already been carried out, then the further options that are available are also displayed (figure 4.4):

Example

```
[root@bright92 ~]# cm-kubernetes-setup
```

TUI session starts up:

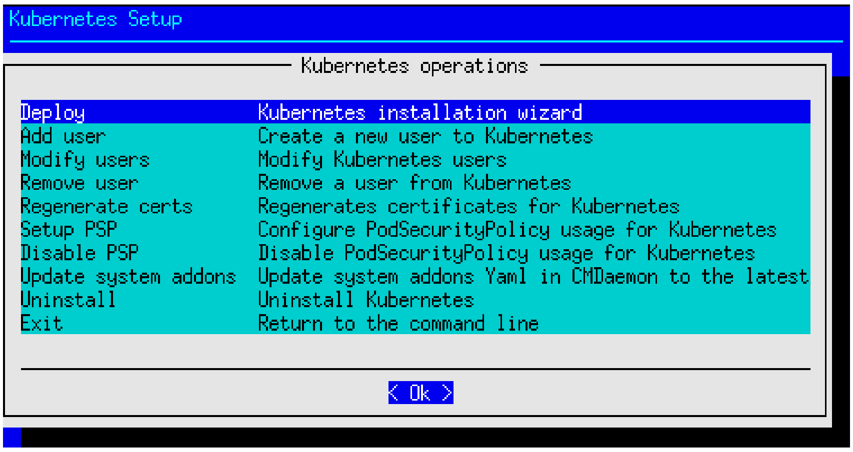

Figure 4.4: Kubernetes Setup TUI Session: Main Operations Screen After A Deployment

The deployment via CLI or via TUI assigns the appropriate roles, and adds the new Kubernetes cluster. The deployment adds health checks to the monitoring configuration, and it generates certificates for the Kubernetes daemons.

It installs a container runtime:

- The container runtime deployed by default is Containerd (figure 4.5).
- Alternatively, the following Docker runtimes can be deployed:
  - The NVIDIA Bright Cluster Manager Docker runtime
  - A third party Docker package can also be used, for example: an existing (non-Bright) Docker already on the cluster.

  However, Docker is deprecated since Kubernetes version 1.21, so neither of the Docker runtime options is recommended.

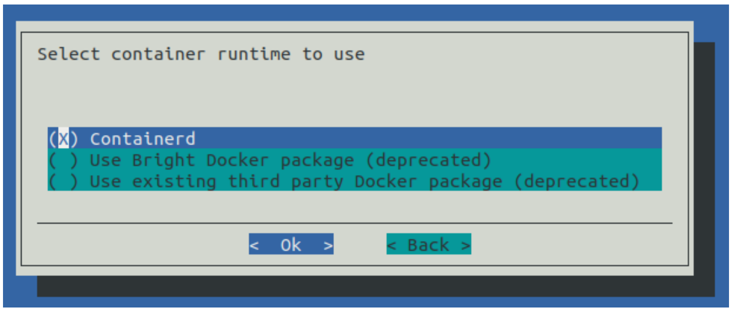

Figure 4.5: Kubernetes Setup TUI Session: Container Runtime Screen

Calico is set as the Kubernetes network plugin by default. Flannel is an option.

The master, worker, and etcd nodes can be assigned to specific nodes or categories.

The network configuration settings for the Kubernetes cluster can be specified. Ports have default assignments, but can be re-assigned as needed. The `etcd` spool file path can be set.

The following options are also possible:

- a registry mirror from DockerHub can be specified
- the Kubernetes API server can be exposed to the external network
- the internal network used by Kubernetes nodes can be selected

Add-ons that are available are:

- Ingress Controller (Nginx)
- Kubernetes Dashboard
- Kubernetes Metrics Server
- Kubernetes State Metrics
- NVIDIA device plugin for Kubernetes

Operator packages are application managers, and are described further in Chapter 6. Operators that can be installed are:

- `cm-jupyter-kernel-operator`
- `cm-kubernetes-postgresql-operator`
- `cm-kubernetes-spark-operator`

The permission manager—an application for role-based access control—can also be configured.

4.2.6 Testing Kubernetes

To test that Kubernetes works, the `cluster-info` command can be run. For example, on the head node, `bright92`:

Example

```
[root@bright92 ~]# module load kubernetes
[root@bright92 ~]# 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'.
```

After `cm-kubernetes-setup` finishes, and the regular nodes have been rebooted, the state of the nodes can be found by running the `get nodes` command:

Example

```
[root@bright92 ~]# kubectl get nodes
NAME       STATUS   ROLES    AGE   VERSION
node001    Ready    <none>   25m   v1.24
node002    Ready    <none>   25m   v1.24
node003    Ready    <none>   25m   v1.24
node004    Ready    <none>   25m   v1.24
node005    Ready    <none>   25m   v1.24
node006    Ready    <none>   25m   v1.24
```

A six node cluster should show the following Kubernetes installed add-ons, when using `kubectl` with the `get all -n kube-system` option (some lines truncated):

Example

```
[root@bright92 ~]# module load kubernetes
[root@bright92 ~]# kubectl get all -n kube-system
NAME                                         READY   STATUS    RESTARTS   AGE
pod/calico-kube-controllers-58497c65d5-skhgw 1/1     Running   0          26m
pod/calico-node-27xj7                        1/1     Running   0          26m
pod/calico-node-6hmm5                        0/1     Running   1          26m
pod/calico-node-987qv                        1/1     Running   0          26m
pod/calico-node-gcbcm                        1/1     Running   0          26m
pod/calico-node-hlsrj                        1/1     Running   0          26m
pod/calico-node-q7k4v                        1/1     Running   0          26m
pod/calico-node-qdbq5                        0/1     Running   0          26m
pod/calico-node-v2dxj                        1/1     Running   0          26m
pod/coredns-6768db756-819fs                  1/1     Running   0          26m
pod/coredns-6768db756-cs58q                  1/1     Running   0          26m
pod/kube-state-metrics-758ccc75d6-75dsn      1/1     Running   0          26m
pod/metrics-server-7b477dd7b9-2drkg          1/1     Running   0          26m
pod/metrics-server-7b477dd7b9-z6nch          1/1     Running   0          26m

NAME                          TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)                  AGE
service/calico-typha          ClusterIP   10.150.121.25   <none>        5473/TCP                 26m
service/kube-dns              ClusterIP   10.150.255.254  <none>        53/UDP,53/TCP,9153/TCP   26m
service/kube-state-metrics    ClusterIP   None            <none>        8080/TCP,8081/TCP        26m
service/metrics-server        ClusterIP   10.150.99.149   <none>        443/TCP                  26m

NAME                          DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE SELECTOR            AGE
daemonset.apps/calico-node    8         8         6       8            6           kubernetes.io/os=linux   26m

NAME                                      READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/calico-kube-controllers   1/1     1            1           26m
deployment.apps/calico-typha              0/0     0            0           26m
deployment.apps/coredns                   2/2     2            2           26m
deployment.apps/kube-state-metrics        1/1     1            1           26m
deployment.apps/metrics-server            2/2     2            2           26m

NAME                                                 DESIRED   CURRENT   READY   AGE
replicaset.apps/calico-kube-controllers-58497c65d5   1         1         1       26m
replicaset.apps/calico-typha-68857595fc              0         0         0       26m
replicaset.apps/coredns-6768db756                    2         2         2       26m
```

```
replicaset.apps/kube-state-metrics-758ccc75d6         1         1         1       26m
replicaset.apps/metrics-server-7b477dd7b9             2         2         2       26m
```

The administrator can now configure the cluster to suit the particular site requirements.

4.3 Using GPUs With Kubernetes: NVIDIA GPUs

4.3.1 Prerequisites

The GPUs have to be recognized by the nodes, and have the appropriate drivers (such as `cuda-driver`) installed. Details on how to do this are given in section 7.4 of the *Installation Manual*.

To verify the GPUs are recognized and have drivers in place, the `nvidia-smi` command can be run. The response displayed for a GPU should look similar to the following:

Example

```
root@node001:~# nvidia-smi
Tue Dec  7 11:25:21 2021
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 470.57.02    Driver Version: 470.57.02    CUDA Version: 11.4     |
|-------------------------------+----------------------+----------------------+
| 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 K40c          On   | 00000000:00:06.0 Off |                  Off |
| 23%   37C    P8    21W / 235W |      0MiB / 12206MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

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

Docker as a container runtime is deprecated since Kubernetes v1.21 (page 30). It is still possible to use Docker with Kubernetes but it is recommended to deploy Containerd for new Kubernetes setups. The setups for NVIDIA GPUs are described next for both Docker and Containerd:

4.3.2 Existing Containerd Deployment

If a non-Bright Containerd has already been deployed before Kubernetes is deployed, then `cm-kubernetes-setup` may replace an existing Containerd configuration file in order to enable NVIDIA GPU integration via a Kubernetes CNI plugin. This is because Containerd is configured by `cm-kubernetes-setup`, overwriting any previous configuration.

4.3.3 Existing Docker Deployment

If a non-Bright Docker has already been deployed before Kubernetes is deployed, then the instructions from section 2.7 must be followed first. These are the instructions on making sure that GPUs can be used inside Docker. The NVIDIA container toolkit (`cm-nvidia-container-toolkit`) has to be present on the nodes.

To verify Docker is working with GPUs, `nvidia-smi` can be run from inside a container:

Example

```
root@node001:~# module load docker
root@node001:~# docker run --runtime=nvidia --rm nvidia/cuda nvidia-smi
Mon Sep 28 13:13:39 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.06    Driver Version: 450.51.06    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| 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 K40c          On   | 00000000:00:06.0 Off |                  Off |
| 23%   28C    P8    22W / 235W |      0MiB / 12206MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

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

4.3.4 New Kubernetes Installation

If Containerd is selected as container runtime, then `cm-kubernetes-setup` assigns a new role to the Kubernetes worker nodes: `generic::containerd`.

The role has a `Configurations` submode, in which the `containerd-cri` object can be configured. The entry for `Filename` specifies the path to the `cri.toml` file, which contains content used by the container runtime interface on the Kubernetes worker nodes that have been assigned the role.

Example

```
[bright92->configurationoverlay[kube-default-worker]->roles]% use generic::containerd
[...]->roles[generic::containerd]]% show
Parameter                        Value
-------------------------------- ------------------------------------------------
Name                             generic::containerd
Type                             GenericRole
Add services                     yes
Provisioning associations        <0 internally used>
Services                         containerd
Configurations                   <2 in submode>
Environments                     <1 in submode>
Exclude list snippets            <1 in submode>
Data node                        no
[...]->roles[generic::containerd]]% configurations
[...]->roles[generic::containerd]->configurations]% use containerd-cri
[...]->roles[generic::containerd]->configurations[containerd-cri]]% show
Parameter                        Value
-------------------------------- --------------------------------------------------
Name                             containerd-cri
Type                             static
Create directory                 yes
Filename                         /cm/local/apps/containerd/var/etc/conf.d/cri.toml
Filemask directory               0644
User name
```

```
Group name
Disabled                         no
Service action on write          RESTART
Service stop on failure          yes
Content                          <645B>
Filemask                         0644
```

The file with the CRI (Container Runtime Interface) configuration is created in directory:

```
/cm/local/apps/containerd/var/etc/conf.d
```

and included into the main Containerd configuration file:

```
/cm/local/apps/containerd/var/etc/config.toml
```

with the `imports` statement:

```
imports = ["/cm/local/apps/containerd/var/etc/conf.d/*.toml"]
```

Whatever the container runtime that is selected, if NVIDIA GPU integration is required then the NVIDIA container toolkit is taken care of by the installer.

There are two ways to integrate NVIDIA GPUs into Kubernetes:

1. With the NVIDIA GPU operator for Kubernetes. This is discussed further in section 6.3.
2. With an add-on called the NVIDIA device plugin for Kubernetes. This is the way discussed in this section.

During the installation of Kubernetes via `cm-kubernetes-setup` (section 4.2.5) there is a step where custom add-ons can be selected (figure 4.6):

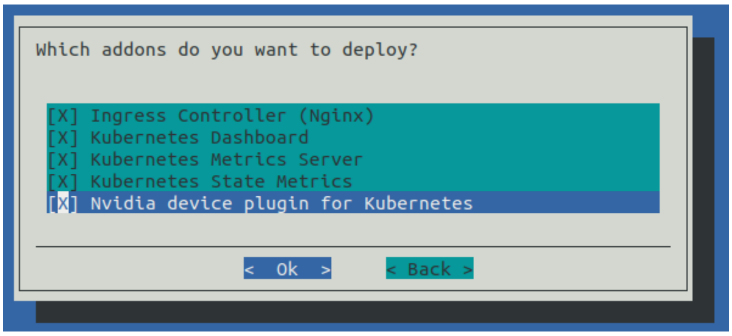

Figure 4.6: Kubernetes Setup TUI Session: Deployment Of Add-ons

The add-on `NVIDIA device plugin for Kubernetes` should be enabled in that step. This ensures that `cm-nvidia-container-toolkit` is installed in the software image.

This is the default approach. It can be verified to work by deploying a pod that requests a GPU, as requested using the `nvidia.com/gpu: 1` line inside the yaml file. Alternatively, the GPU operator can be deployed via Helm (page 95).

If more GPUs are available on a single host, then only one GPU should be made visible, and recognized inside the pod, when requesting a single GPU, as in the example:

Example

```
root@cluster:~> cat gpu.yaml
apiVersion: v1
kind: Pod
metadata:
```

```
  name: gpu-pod
spec:
  restartPolicy: Never
  containers:
  - name: cuda-container
    image: nvidia/cuda:9.2-runtime
    command: ["nvidia-smi"]
    resources:
      limits:
        nvidia.com/gpu: 1
root@cluster:~> kubectl apply -f gpu.yaml
pod/gpu-pod configured
root@cluster:~> kubectl logs gpu-pod
Mon Sep 28 12:12:46 2020
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 450.51.06    Driver Version: 450.51.06    CUDA Version: 11.0     |
|-------------------------------+----------------------+----------------------+
| 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 K40c          On   | 00000000:00:06.0 Off |                  Off |
| 23%   29C    P8    22W / 235W |      0MiB / 12206MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

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

4.3.5 Existing Kubernetes Installation

The NVIDIA device plugin can be deployed either as an add-on (this chapter) or as an operator (section 6.3). If during the installation the add-on was not selected (section 4.3.4 shows a screenshot), then it can be enabled in cmsh or Bright View afterwards.

```
[cluster->kubernetes[default]->appgroups[system]->applications[nvidia]]% set enabled yes
[cluster->kubernetes*[default*]->appgroups*[system*]->applications*[nvidia*]]% commit
```

This add-on deploys a DaemonSet that runs the NVIDIA device plugin for Kubernetes on nodes with the brightcomputing.com/gpu-accelerator label.

NVIDIA Bright Cluster Manager is responsible for setting this GPU label on the appropriate Kubernetes nodes, with parameter values specified for the labelset:

Example

```
[cluster->kubernetes[default]->labelsets[nvidia]]% show
Parameter                        Value
-------------------------------- ------------------------------------------------
Name                             nvidia
Revision
Labels                           brightcomputing.com/gpu-accelerator=
Nodes                            node001
```

```
Categories
Overlays
```

These labels should appear in Kubernetes Node resources as a result, and the device plugin running on these particular nodes (some columns of output have been removed for clarity):

Example

```
[root@head ~]# kubectl get nodes --show-labels
NAME      STATUS   ROLES    AGE   VERSION   LABELS
node001   Ready    <none>   16h   v1.21.4   beta.kube...com/gpu-accelerator=...hostname=node001...
node002   Ready    <none>   16h   v1.21.4   beta.kube...com/node-category=de...hostname=node002...
node003   Ready    <none>   16h   v1.21.4   beta.kube...com/node-category=de...hostname=node003...
node004   Ready    <none>   16h   v1.21.4   beta.kube...com/node-category=de...hostname=node004...
node005   Ready    <none>   16h   v1.21.4   beta.kube...com/node-category=de...hostname=node005...
node006   Ready    <none>   16h   v1.21.4   beta.kube...com/node-category=de...hostname=node006...
node007   Ready    <none>   16h   v1.21.4   beta.kube...com/node-category=de...hostname=node007...
head      Ready    master   16h   v1.21.4   beta.kube...com/ingress-controll...hostname=head...


[root@head ~]# kubectl get nodes -l brightcomputing.com/gpu-accelerator=
NAME      STATUS   ROLES    AGE   VERSION
node001   Ready    <none>   16h   v1.21.4

[root@head ~]#  kubectl get pod -l name=nvidia-device-plugin-ds -n kube-system -o wide
NAME                                   READY  STATUS    ... NODE      NOMINATED NODE   READINESS GATES
nvidia-device-plugin-daemonset-p9x2p   1/1    Running   ... node001   <none>           <none>
```

4.4 Using GPUs With Kubernetes: AMD GPUs

4.4.1 Prerequisites

The GPUs have to be recognized by the node. One way to check this from within the cluster manager is to run `sysinfo` for the node:

Example

```
[bright92->device[bright92]]% sysinfo | grep GPU
Number of GPUs                          1
GPU Driver Version                      4.18.0-193.el8.x86_64
GPU0 Name                               Radeon Instinct MI25
```

In order to make Kubernetes aware of nodes that have AMD GPUs, the AMD GPU device plugin has to be deployed as a DaemonSet inside Kubernetes. The official GitHub repository that hosts this plugin can be found at:

`https://github.com/RadeonOpenCompute/k8s-device-plugin`

The device plugin requires Kubernetes v1.16+, which has been around since NVIDIA Bright Cluster Manager version 9.0. With some extra instructions, the plugin can also be made a part of NVIDIA Bright Cluster Manager version 8.2.

The DaemonSet YAML file can be deployed with:

Example

```
kubectl create -f https://raw.githubusercontent.com/RadeonOpenCompute/k8s-device-plugin/v1.16/
k8s-ds-amdgpu-dp.yaml
```

4.4.2 Managing The YAML File Through CMDaemon

The plugin can be added by the user via the Kubernetes appgroups as an application. In the session that follows, it is given the arbitrary name `device-plugin`:

Example

```
[root@bright92 ~]# wget https://raw.githubusercontent.com/RadeonOpenCompute/k8s-device-plugin/
v1.16/k8s-ds-amdgpu-dp.yaml -O /tmp/k8s-ds-amdgpu-dp.yaml
[root@bright92 ~]# cmsh
[bright92]% kubernetes
[bright92->kubernetes[default]]% appgroups
[bright92->kubernetes[default]->appgroups]% add amd
[bright92->kubernetes*[default*]->appgroups*[amd*]]% applications
[bright92->kubernetes*[default*]->appgroups*[amd*]->applications]% add device-plugin
```

The configuration of the plugin can be set to the YAML file, by setting the `config` parameter to take the value of the YAML file path.

Example

```
[bright92->...[amd*]->applications*[device-plugin*]]% set config /tmp/k8s-ds-amdgpu-dp.yaml
[bright92->kubernetes*[default*]->appgroups*[amd*]->applications*[device-plugin*]]% show
Parameter                        Value
-------------------------------- ------------------------------------------------
Name                             device-plugin
Revision
Format                           Yaml
Enabled                          yes
Config                           <914B>
Environment                      <0 in submode>
Exclude list snippets            <0 in submode>
[bright92->kubernetes*[default*]->appgroups*[amd*]->applications*[device-plugin*]]% commit
```

The YAML file can also be edited within `cmsh` after it has been set, by running `set config` without a value.

There are older releases available, starting from Kubernetes v1.10, if needed. Saving this device-plugin YAML should result in pods being scheduled on all the non-tainted nodes, as seen by listing the pods (some columns elided):

```
[root@bright92 ~]# module load kubernetes/default/
[root@bright92 ~]# kubectl get pod -n kube-system -l name=amdgpu-dp-ds -o wide
NAME                                  READY   STATUS     ... IP              NODE    ...
amdgpu-device-plugin-daemonset-66jl7   1/1     Running    ... 172.29.112.135  gpu001 ...
amdgpu-device-plugin-daemonset-8mh9w   1/1     Running    ... 172.29.152.130  gpu002 ...
```

4.4.3 Including Head Nodes as part of the DaemonSet:

The cluster manager taints head nodes, so that they do not run non-critical pods. The taint can be removed with the "`-`" operator to allow non-critical pods to run:

Example

```
kubectl taint nodes bright92 node-role.kubernetes.io/master-
```

However, a more specific exception can be configured in the DaemonSet itself.

Within the YAML file, the following existing `tolerations` definition has to be modified, from:

```
tolerations:
- key: CriticalAddonsOnly
  operator: Exists
```

to:

```
tolerations:
- key: node-role.kubernetes.io/master
  effect: NoSchedule
  operator: Exists
```

The modified toleration tolerates this taint, and therefore has the device plugin run on such tainted nodes.

Verifying That AMD GPUs Are Recognized By Kubernetes

If Kubernetes is aware of the AMD GPUs for a node then several mentions of `amd.com/gpu` are displayed when running the `kubectl describe node` command for the node. The following session shows output for a node `gpu01`, ellipsized for clarity:

Example

```
[root@bright92 ~]# kubectl describe node gpu01
Name:                 gpu01
...
Capacity:
  amd.com/gpu:        3
  cpu:                64
  ephemeral-storage:  1813510Mi
  hugepages-1Gi:      0
  hugepages-2Mi:      0
  memory:             527954676Ki
  pods:               50
...
```

4.4.4 Running The DaemonSet Only On Specific Nodes

The AMD GPU device plugin, unlike the NVIDIA GPU device plugin Daemonset, is scheduled to run on each Kubernetes host. This means that it runs even if the host has no GPU.

This can be prevented with the following steps:

A LabelSet can be created via `cmsh`, and the nodes or categories that have GPUs are assigned within the `labelsets` mode:

Example

```
[root@bright92 ~]# cmsh
[bright92]% kubernetes
[bright92->kubernetes[default]]% labelsets
[bright92->kubernetes[default]->labelsets]% use nvidia
[bright92->kubernetes[default]->labelsets[nvidia]]% ..   #but, we're using AMD GPUs, so let's go back up:
[bright92->kubernetes[default]->labelsets]% add amd
[bright92->kubernetes*[default*]->labelsets*[amd*]]% set labels brightcomputing.com/amd-gpu-accelerator=
[bright92->kubernetes*[default*]->labelsets*[amd*]]% append categories gpu-nodes
[bright92->kubernetes*[default*]->labelsets*[amd*]]% commit
```

This assigns the labels to the nodes with GPUs. This can be verified with:

Example

```
kubectl get nodes -l brightcomputing.com/amd-gpu-accelerator=
NAME     STATUS   ROLES    AGE   VERSION
gpu001   Ready    master   66m   v1.18.8
gpu002   Ready    master   66m   v1.18.8
...
```

The DaemonSet YAML can now be adjusted to only run the device plugin on nodes with this new label. This can be done by adding an affinity block after the tolerations block:

Example

```
      tolerations:
      - key: CriticalAddonsOnly  # toleration may be different, if changes were made to it
        operator: Exists
      affinity:
        nodeAffinity:
          requiredDuringSchedulingIgnoredDuringExecution:
            nodeSelectorTerms:
              - matchExpressions:
                  - key: 'brightcomputing.com/amd-gpu-accelerator'
                    operator: Exists
```

This results in the device plugin pods being removed immediately from all nodes that do not have the label.

4.4.5 Running An Example Workload

An example workload can be run as described in the official AMD GPU Kubernetes device plugin documentation at:

https://github.com/RadeonOpenCompute/k8s-device-plugin/tree/v1.16#example-workload

Thus it should now be possible to run:

```
[root@bright92 ~]# kubectl create -f https://raw.githubusercontent.com/RadeonOpenCompute/
k8s-device-plugin/v1.16/example/pod/alexnet-gpu.yaml
```

The YAML requests only one GPU at the bottom of the YAML file:

```
apiVersion: v1
kind: Pod
metadata:
  name: alexnet-tf-gpu-pod
  labels:
    purpose: demo-tf-amdgpu
spec:
  containers:
    - name: alexnet-tf-gpu-container
      image: rocm/tensorflow:latest
      workingDir: /root
      env:
      - name: HIP_VISIBLE_DEVICES
        value: "0" # # 0,1,2,...,n for running on GPU and select the GPUs, -1 for running on CPU
      command: ["/bin/bash", "-c", "--"]
args: ["python3 benchmarks/scripts/tf_cnn_benchmarks/tf_cnn_benchmarks.py --model=alexnet;\
trap : TERM INT; sleep infinity & wait"]
```

```
      resources:
        limits:
          amd.com/gpu: 1 # requesting a GPU
```

Container creation might take a while due to the image size. Once scheduled, it prints out that it found exactly one GPU, and proceeds to run a TensorFlow workload.

Example

```
[root@bright92 ~]# kubectl logs -f alexnet-tf-gpu-pod
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/compat/v2_compat.py:96:
disable_resource_variables (from tensorflow.python.ops.variable_scope) is deprecated and will be removed
in a future version.
Instructions for updating:
non-resource variables are not supported in the long term
2021-01-08 21:03:29.222293: I tensorflow/core/platform/profile_utils/cpu_utils.cc:104]
CPU Frequency: 2495445000 Hz
2021-01-08 21:03:29.222398: I tensorflow/compiler/xla/service/service.cc:168]
XLA service 0x39f62f0 initialized for platform Host (this does not guarantee that XLA will be used). Devices:
2021-01-08 21:03:29.222420: I tensorflow/compiler/xla/service/service.cc:176]
      StreamExecutor device (0): Host, Default Version
2021-01-08 21:03:29.223754: I tensorflow/stream_executor/platform/default/dso_loader.cc:48]
Successfully opened dynamic library libamdhip64.so
2021-01-08 21:03:31.635339: I tensorflow/compiler/xla/service/service.cc:168]
XLA service 0x3a40bb0 initialized for platform ROCM (this does not guarantee that XLA will be used). Devices:
2021-01-08 21:03:31.635363: I tensorflow/compiler/xla/service/service.cc:176]
      StreamExecutor device (0): Device 738c, AMDGPU ISA version: gfx908
2021-01-08 21:03:31.931125: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1734]
Found device 0 with properties:
          pciBusID: 0000:27:00.0 name: Device 738c     ROCm AMD GPU ISA: gfx908
          coreClock: 1.502GHz coreCount: 120 deviceMemorySize: 31.98GiB deviceMemoryBandwidth: 1.12TiB/s
...
TensorFlow:  2.3
Model:       alexnet
Dataset:     imagenet (synthetic)
Mode:        training
SingleSess:  False
Batch size:  512 global
             512 per device
Num batches: 100
Num epochs:  0.04
Devices:     ['/gpu:0']
...
```

Had more GPUs been requested, more would have been made available to the container.

For comparison, a CPU version of the container is also available. The official instructions can be referred to for these, too.

4.5 Kubernetes Configuration Overlays

A list of configuration overlays can be seen from within `configurationoverlay` mode:

Example

```
[bright92->configurationoverlay]% list
Name (key)           Priority   Nodes             Categories    Roles
-------------------- ---------- ----------------- ------------- ------------------...
```

```
kube-default-etcd    500       node001..node003                Etcd::Host
kube-default-master  510       node001..node003                Docker::Host, Kube...
kube-default-worker  500       node004..node006  default       Docker::Host, Kube...
```

Configuration overlays can be used to manage the Kubernetes services used with a particular configuration. For example, when managing the Kubernetes services used for a Kubernetes engine within an Auto Scale tracker (section 8.4.9 of the *Administrator Manual*).

4.6 Removing A Kubernetes Cluster

A Kubernetes cluster can be removed using `cm-kubernetes-setup` with the `--remove` and `--yes-i-really-mean-it` options. Also, if there more than one cluster present, then the cluster name must be specified using the `--cluster` parameter.

A removal run looks as follows (some output ellipsized):

Example

```
[root@bright92 ~]# cm-kubernetes-setup --remove --cluster default --yes-i-really-mean-it

Connecting to CMDaemon
Executing 20 stages
################### Starting execution for 'Kubernetes Setup'
  - kubernetes
  - docker
## Progress: 0
#### stage: kubernetes: Get Kube Cluster
## Progress: 5
#### stage: kubernetes: Check Kube Cluster Exists
## Progress: 10
#### stage: kubernetes: Find Installed Components
## Progress: 15
#### stage: kubernetes: Find Files On Headnodes
## Progress: 20
#### stage: kubernetes: Firewall Zone Close
## Progress: 25
#### stage: kubernetes: Firewall Interface Close
## Progress: 30
#### stage: kubernetes: Firewall Policy Close
## Progress: 35
#### stage: kubernetes: Nginx Reverse Proxy Close
## Progress: 40
#### stage: kubernetes: IP Ports Close
## Progress: 60
#### stage: kubernetes: Remove Installed Components
## Progress: 65
#### stage: kubernetes: Remove Files On Headnodes
## Progress: 70
#### stage: kubernetes: Remove Etcd Spool
## Progress: 80
#### stage: kubernetes: Set Reboot Required
You need to reboot 2 nodes to cleanup the network configuration
## Progress: 85
#### stage: kubernetes: Collection Update Provisioners
## Progress: 100

Took:     00:08 min.
```

```
Progress: 100/100
################### Finished execution for 'Kubernetes Setup', status: completed

Kubernetes Setup finished!
```

Using the `--remove` option removes the Kubernetes cluster configuration from the cluster manager, unassigns Kubernetes-related roles—including the EtcdHost role—and removes Kubernetes health checks. The command does not remove packages that were installed with a `cm-kubernetes-setup` command before that.

After the disabling procedure has finished, the cluster has no Kubernetes configured and running.

4.7 Kubernetes Cluster Configuration Options

Kubernetes allows many Kubernetes clusters to be configured. These are separated sets of hosts with different certificates, users and other global settings.

When carrying out the Kubernetes setup run, a Kubernetes cluster name will be asked, and a new object with the cluster settings is then added into the CMDaemon configuration. The administrator can change the settings of the cluster from within the `kubernetes` mode of `cmsh` or within the `Kubernetes Clusters` options window of Bright View, accessible via the clickpath `Containers`→`Kubernetes Clusters`.

The `cmsh` equivalent looks like:

Example

```
[root@bright92 ~]# cmsh
[bright92]% kubernetes list
Name (key)
------------------
default
[bright92]% kubernetes use default
[bright92->kubernetes[default]]% show
Parameter                                    Value
-------------------------------------------- ------------------------------------------------------
Kubernetes role bindings                      <0 in submode>
Notes
Revision
Name                                         default
Authorization Mode                           Node,RBAC
Kube Config                                  /cm/local/apps/kubernetes/var/etc/node.kubeconfig
Kube Client Config                           /cm/local/apps/kubernetes/var/etc/kubelet.kubeconfig
Kube Config Template                         <409 bytes>
CA                                           /cm/local/apps/kubernetes/var/etc/kubeca-default.pem
CA Key                                       /cm/local/apps/kubernetes/var/etc/kubeca-default.key
Kubernetes Certificate                       /cm/local/apps/kubernetes/var/etc/node.pem
Kubernetes Key                               /cm/local/apps/kubernetes/var/etc/node.key
Kubernetes Client Certificate                /cm/local/apps/kubernetes/var/etc/kubelet.pem
Kubernetes Client Key                        /cm/local/apps/kubernetes/var/etc/kubelet.key
Service Accounts Certificate                 /cm/local/apps/kubernetes/var/etc/sa-default.pem
Service Accounts Certificate Key             /cm/local/apps/kubernetes/var/etc/sa-default.key
Kubernetes API Aggregator Certificate        /cm/local/apps/kubernetes/var/etc/apiaggregator.pem
Kubernetes API Aggregator Certificate Key    /cm/local/apps/kubernetes/var/etc/apiaggregator.key
Cluster Domain                               cluster.local
Etcd Cluster                                 kube-default
Etcd Prefix                                  /kube-apiserver
Etcd Servers
```

```
Service Network                        kube-default-service
Trusted domains                        bright92.local,kubernetes,kubernetes.default,kubernetes.\
                                       default.svc,master,localhost
Pod Network                            kube-default-pod
Pod Network Node Mask
Internal Network                       internalnet
KubeDNS IP                             10.150.255.254
Kubernetes API server
Kubernetes API server proxy port       10444
App Groups                             <1 in submode>
Label Sets                             <2 in submode>
Module file template                   <1130 bytes>
[bright92->kubernetes[default]]%
```

The preceding `kubernetes` mode parameters are described in table 4.2:

Parameter	Description
`Authorization Mode`	Selects how to authorize on the secure port (default: RBAC (Role-Based Access Control) and Node Authorization modes)
`Kube Config`	Path to a kubeconfig file, specifying how nodes authenticate to the API server
`Kube Client Config`	Path to a kubeconfig file, specifying how kubelets authenticate to the API server
`Kube Config Template`	Template the cluster manager uses to generate kubeconfig files for services and users.
`CA`	Path to PEM-encoded RSA or ECDSA certificate used for the CA
`CA Key`	Path to PEM-encoded RSA or ECDSA private key used for the CA
`Kubernetes Certificate`	File containing x509 certificate used by the kubelets
`Kubernetes Key`	File containing x509 private key used by the Kubernetes kubelets
`Kubernetes Client Certificate`	File containing x509 certificate used for the kubelets
`Kubernetes Client Key`	File containing x509 private key used for the Kubernetes kubelets
`Service Accounts Certificate`	File containing x509 certificate used for Kubernetes service accounts. This certificate value will be used as `--service-account-key-file` in the Kubernetes API service.

...continues

...continued

Parameter	Description
`Service Accounts Key`	File containing x509 private key used for Kubernetes Service Accounts. This key value will be used as `--service-account-private-key-file` in the controller manager service.
`Cluster Domain`	Domain for this cluster
`Etcd Cluster`	The Etcd cluster instance.
`Etcd Prefix`	The prefix for all resource paths in `etcd`
`Etcd Servers`	List of `etcd` servers to watch (format: `http://`<*IP address*>`:`<*port number*>)
`Service Network`	Network from which the service cluster IP addresses will be assigned, in IPv4 CIDR format. Must not overlap with any IP address ranges assigned to nodes for pods. Default: `172.29.0.0/16`
`Pod Network`	Network where pod IP addresses will be assigned from
`Internal Network`	Network to back the internal communications.
`Trusted Domains`	Trusted domains to be included in Kubernetes-related certificates as Alternative Subject Names.
`KubeDNS IP`	CoreDNS IP Address.
`Kubernetes API Server`	Kubernetes API server address (format: `https://`<*host*>`:`<*port number*>).
`Kubernetes API Server Proxy Port`	Kubernetes API server proxy port.
`Addons`	Kubernetes Add-ons managed by CMDaemon.

Table 4.2: `kubernetes` *mode parameters*

4.8 EtcdCluster

The EtcdCluster mode sets the global Etcd cluster settings. It can be accessed via the top level `etcd` mode of `cmsh`.

Parameter	Description	Option to `etcd`
`Name`	Etcd Cluster Name.	`--initial-cluster-token`
`Election Timeout`	Election Timeout, in milliseconds.	`--election-timeout`
`Heart Beat Interval`	Heart Beat Interval, in milliseconds.	`--heartbeat-interval`
`CA`	The Certificate Authority (CA) Certificate path for Etcd, used to generate certificates for Etcd.	`--peer-trusted-ca-file`
`CA Key`	The CA Key path for Etcd, used to generate certificates for Etcd.	
`Member Certificate`	The Certificate path to use for Etcd cluster members, signed with the Etcd CA. The EtcdHost Role can specify a Member CA as well, and in that case it overwrites any value set here.	`--peer-cert-file`
`Member Certificate Key`	The Key path to use for Etcd cluster members, signed with the Etcd CA. The EtcdHost Role can specify a Member CA as well, and in that case it overwrites any value set here.	`--peer-key-file`
`Client CA`	The CA used for client certificates. When set it is assumed client certificate and key will be generated and signed with this CA by another party. Etcd still expects the path to be correct for the Client Certificate and Key.	`--trusted-ca-file`
`Client Certificate`	The Client Certificate, used by Etcdctl for example.	`--cert-file`
`Client Certificate Key`	The Client Certificate Key, used by Etcdctl for example.	`--key-file`

* Boolean (takes `yes` or `no` as a value)

Table 4.3: *EtcdCluster role parameters and* `etcd` *options*

4.9 Kubernetes Roles

Kubernetes roles include the following roles:

- `EtcdHost` (page 47)
- `KubernetesApiServerProxy` (page 48)
- `KubernetesApiServer` (page 48)
- `KubernetesController` (page 50)
- `KubernetesScheduler` (page 52)
- `KubernetesProxy` (page 53)
- `KubernetesNode` (page 54)

When nodes are configured using Kubernetes roles, then settings in these roles may sometimes use the same pointer variables—for example the Kubernetes or Etcd cluster instance. Pointer variables such as these have definitions that are shared across the roles, as indicated by the parameter description tables for the roles, and which are described in the following pages.

In `cmsh`, the roles can be assigned:

- for individual nodes via the `roles` submode of `device` mode
- for a category via the `roles` submode of a category
- for a configuration overlay via the `roles` submode of `configurationoverlay` mode

4.9.1 EtcdHost Role

The EtcdHost role is used to configure and manage the `etcd` service for a node.

The `etcd` service manages the `etcd` database, which is a hierarchical distributed key-value database. The database is used by Kubernetes to store its configurations. The EtcdHost role parameters are described in table 4.4:

Parameter	Description	Option to `etcd`
`Member Name`	The human-readable name for this `etcd` member (`$hostname` will be replaced by the node hostname)	`--name`
`Spool`	Path to the data directory (default: `/var/lib/etcd`)	`--data-dir`
`Advertise Client URLs`	List of client URLs for this member to advertise publicly (default: `http://$hostname:5001`)	`--advertise-client-urls`
`Advertise Peers URLs`	List of peer URLs for this member to advertise to the rest of the cluster (default: `http://$hostname:5002`)	`--initial-advertise-peer-urls`
`Listen Client URLs`	List of URLs to listen on for client traffic (default: `http://$hostname:5001`, `http://127.0.0.1:2379`)	`--listen-client-urls`
`Listen Peer URLs`	List of URLs to listen on for peer traffic (default: `http://$hostname:5002`)	`--listen-peer-urls`
`Snapshot Count`	Number of committed transactions that trigger a snapshot to disk (default: 5000)	`--snapshot-count`
`Debug`*	Drop the default log level to DEBUG for all subpackages? (default: `no`)	`--debug`
`Member Certificate`	Etcd member certificate, signed with CA specified in the Etcd Cluster. When set it will overrule the value from the EtcdCluster object. Default empty.	`--peer-cert-file`

...continues

...continued

Parameter	Description	Option to `etcd`
`Member Certificate Key`	Etcd member certificate key, signed with CA specified in the Etcd Cluster. When set it will overrule the value from the EtcdCluster object. Default empty.	`--peer-keyt-file`
`Options`	Additional parameters for the `etcd` daemon (empty by default)	

* Boolean (takes `yes` or `no` as a value)

Table 4.4: *EtcdHost role parameters and* `etcd` *options*

The `etcd` settings are updated by the cluster manager in `/cm/local/apps/etcd/current/etc/cm-etcd.conf`.

4.9.2 The KubernetesAPIServerProxy Role

The KubernetesApiServerProxy role sets up a proxy that provides the entry point for one or more instances of the Kubernetes API server. The proxy runs on every node of a Kubernetes cluster instance, including the head node.

If multiple Kubernetes master nodes are present, then it enables HA for the Kubernetes master components, as described in section 4.1.

4.9.3 The KubernetesApiServer Role

The KubernetesApiServer role is used to configure and manage the `kube-apiserver` daemon. The `kube-apiserver` daemon is a Kubernetes API server that validates and configures data for the Kubernetes API objects. The API objects include pods, services, and replication controllers. The API Server processes REST operations, and provides a front end to the shared state of the cluster through which all the other components interact.

The KubernetesApiServer role parameters are described in table 4.5:

Parameter	Description	Option to `kube-apiserver`
`Kubernetes Cluster`	The Kubernetes cluster instance (pointer)	
`Insecure API Port`	The port on which to serve unsecured, unauthenticated access (disabled by default)	`--insecure-port`
`Secure API Port`	The port on which to serve HTTPS with authentication and authorization. If 0, then HTTPS will not be served at all. (default: `6443`)	`--secure-port`
`Advertise Address`	The IP address on which to advertise the API server to members of the cluster with `--advertise-address`. If set to `0.0.0.0`, then the IP address of the management network of the head node is used. (default: `0.0.0.0`)	
`Insecure Bind Address`	IP address to serve on (default: `127.0.0.1`)	`--insecure-bind-address`

...continues

...continued

Parameter	Description	Option to `kube-apiserver`
`Secure Bind Address`	The IP address on which to serve the read- and secure ports (default: `0.0.0.0`)	`--bind-address`
`Admission Control`	Ordered list of plug-ins to control the admission of resources into the cluster) (default: `NamespaceLifecycle,LimitRanger, ServiceAccount,PersistentVolumeLabel, DefaultStorageClass,ValidatingAdmissionWebhook, ResourceQuota,DefaultTolerationSeconds, MutatingAdmissionWebhook`)	`--admission-control`
`Allowed Privileged`*	If `yes`, then allow privileged containers. (default: `no`)	`--allow-privileged`
`Event TTL`	Time period that events are retained. Empty by default. A format example: `1h0m0s`	`--event-ttl`
`Kubelet Timeout`	Kubelet port timeout (default: `5s`)	`--kubelet-timeout`
`Log Level`	Log level (default: `0`)	`--v`
`Log To StdErr`*	Logging to stderr means it goes into the systemd journal (default: `yes`)	`--logtostderr`
`Options`	Additional parameters for the `kube-apiserver` daemon (empty by default)	

* Boolean (takes `yes` or `no` as a value)

***Table 4.5:** KubernetesApiServer role parameters and* `kube-apiserver` *options*

Further details on the Kubernetes API Server can be found at `https://kubernetes.io/docs/admin/kube-apiserver/`.

4.9.4 KubernetesController Role

The Kubernetes Controller role is used to configure and manage the `kube-controller-manager` daemon that embeds the core control loops shipped with Kubernetes. In Kubernetes, a controller is a control loop that watches the shared state of the cluster through the API server, and it makes changes in order to try to move the current state towards the desired state. Examples of controllers that ship with Kubernetes at the time of writing (January 2018) are:

- the replication controller
- the endpoints controller,
- the namespace controller,
- the serviceaccounts controller

The KubernetesController role parameters are described in table 4.6:

Parameter	Description	Option to kube-controller-manager
Kubernetes Cluster	The Kubernetes cluster instance (pointer)	
Address	IP address to serve on (default: 0.0.0.0)	--address
Port	Port to serve on (default: 10252)	--port
Concurrent Endpoint Syncs	Number of endpoint syncing operations that will be done concurrently. (default: 5	--concurrent-endpoint-syncs
Concurrent Rc Syncs	The number of replication controllers that are allowed to sync concurrently. 5	--concurrent-rc-syncs
Namespace Sync Period	Period for syncing namespace life-cycle updates	--namespace-sync-period
Node Monitor Grace Period	Period for syncing NodeStatus in NodeController	--node-monitor-grace-period
Node Monitor Period	Period the running Node is allowed to be unresponsive before marking it unhealthy	--node-monitor-period
Node Startup Grace Period	Period the starting Node is allowed to be unresponsive before marking it unhealthy	--node-startup-grace-period
Node Sync Period	Period for syncing nodes from cloud-provider	--node-sync-period
Pod Eviction Timeout	Grace period for deleting pods on failed nodes	--pod-eviction-timeout
Pv Claim Binder Sync Period	Period for syncing persistent volumes and persistent volume claims	--pvclaimbinder-sync-period
Register Retry Count	Number of retries for initial node registration	--register-retry-count
Resource Quota Sync Period	Period for syncing quota usage status in the system	--resource-quota-sync-period
Log Level	Log level (default: 0)	--v
Log To StdErr*	Logging to stderr means getting it into the systemd journal (default: yes)	--logtostderr
Options	Additional parameters for the kube-controller-manager daemon	
Cluster signing Cert file	Filename containing a PEM-encoded X509 CA certificate used to issue cluster-scoped certificates. (leave empty to use the value of CA defined in the Kubernetes Cluster instance).	--cluster-signing-cert-file

...continues

...continued

Parameter	Description	Option to `kube-controller-manager`
`Cluster signing Cert Key file`	Filename containing a PEM-encoded RSA or ECDSA private key used to sign cluster-scoped certificates. (leave empty to use the value of CA Key defined in the Kubernetes Cluster instance).	`--cluster-signing-key-file`
`Use Service Account Credentials`	Flag to enable or disable use of Service Account Credentials.	`--use-service-account-credentials`
`Allocate Node Cidrs`	Allocate node CIDR in cluster using Pod Network and Node Mask size defined in Kubernetes Cluster Object.	`--allocate-node-cidrs`
`Kube Config`	Path to a kubeconfig file, specifying how to authenticate to API server.	`--kubeconfig`
`Kubernetes Certificate`	File containing x509 Certificate used by Kubernetes Controller Manager. This will be embedded in the Kube Config file.	(1)
`Kubernetes Key`	File containing x509 private key used by Kubernetes Controller Manager. This will be embedded in the Kube Config file.	(2)

* Boolean (takes `yes` or `no` as a value)

Table 4.6: *KubernetesController role parameters and* `kube-controller-manager` *options*

Further details on the Kubernetes controller manager can be found at `https://kubernetes.io/docs/admin/kube-controller-manager/`.

4.9.5 KubernetesScheduler Role

The KubernetesScheduler role is used to configure and manage the `kube-scheduler` daemon. The Kubernetes scheduler defines pod placement, taking into account the individual and collective resource requirements, quality of service requirements, hardware/software/policy constraints, affinity and anti-affinity specifications, data locality, inter-workload interference, deadlines, and so on.

The KubernetesScheduler role parameters are described in table 4.7:

Parameter	Description	Option to `kube-scheduler`
`Kubernetes Cluster`	The Kubernetes cluster instance (pointer)	
`Address`	IP address to serve on (default: `0.0.0.0`)	`--address`
`Scheduler Port`	Port to serve on (default: `10253`)	`--port`
`Algorithm Provider`	The scheduling algorithm provider to use (default: `DefaultProvider`)	`--algorithm-provider`
`Policy Config`	File with scheduler policy configuration (default: `/cm/local/apps/kubernetes/var/etc/scheduler-policy.json`)	`--policy-config-file`

...continues

...continued

Parameter	Description	Option to `kube-scheduler`
`Log Level`	Log level (default: `0`)	`--v`
`Log To StdErr`*	Logging to STDERR means getting it into the systemd journal (default: `yes`)	`--logtostderr`
`Options`	Additional parameters for the `kube-scheduler` daemon	
`Kube Config`	Path to a kubeconfig file, specifying how to authenticate to API server.	`--kubeconfig`
`Kubernetes Certificate`	File containing x509 Certificate used by Kubernetes Scheduler. This certificate will be embedded in the Kube Config file.	
`Kubernetes Key`	File containing x509 private key used by Kubernetes Scheduler. This certificate key will be embedded in the Kube Config file.	

* Boolean (takes `yes` or `no` as a value)

Table 4.7: *KubernetesScheduler role parameters and* `kube-scheduler` *options*

Further details on the Kubernetes scheduler can be found at `https://kubernetes.io/docs/admin/kube-scheduler/`.

4.9.6 KubernetesProxy Role

The KubernetesProxy role is used to configure and manage `kube-proxy` daemon. The `kube-proxy` daemon runs on each node, and reflects services as defined in the Kubernetes API. It can do simple TCP and UDP stream-forwarding or round-robin TCP and UDP forwarding across a set of back ends.

The KubernetesProxy role parameters are described in table 4.8:

Parameter	Description	Configuration Parameter Passed To `kube-proxy`
`Kubernetes Cluster`	The Kubernetes cluster instance (pointer)	
`Address`	IP address to serve on (default: `0.0.0.0`)	`--address`
`Proxy Port Range Start`	Bottom of range of host ports that may be consumed in order to proxy service traffic (not set by default)	`--proxy-port-range`
`Proxy Port Range End`	Top of range of host ports that may be consumed in order to proxy service traffic (not set by default)	`--proxy-port-range`
`Masquerade All`	If using the pure iptables proxy, SNAT everything (default: `yes`)	`--masquerade-all`
`Health Check Address`	IP address for the health check server to serve on	`--healthz-port`
`Health Check Port`	Port to bind the health check server to serve on (default: `10251`, use 0 to disable)	`--healthz-port`
`Oom Score Adjust`	The `oom_score_adj` value for the kube-proxy process, in range [-999, 999] (default: `-999`)	`--oom-score-adj`
`Kube Config`	Path to a kubeconfig file, specifying how to authenticate to the API server.	`--kubeconfig`
`Kubernetes Certificate`	File containing x509 Certificate used by kube-proxy. This certificate is embedded in the Kube Config.	
`Kubernetes Key`	File containing x509 private key used by Kubernetes API server. This certificate key is embedded in the Kube Config.	
`Log Level`	Log level (default: `0`)	`--v`
`Log To StdErr`*	Logging to STDERR means it goes in the systemd journal (default: `yes`)	`--logtostderr`

* Boolean (takes `yes` or `no` as a value)

Table 4.8: *KubernetesProxy role parameters and* `kube-proxy` *options*

Further details on the Kubernetes network proxy can be found at `https://kubernetes.io/docs/admin/kube-proxy/`.

4.9.7 KubernetesNode Role

The KubernetesNode role is used to configure and manage the `kubelet` daemon, which is the primary node agent that runs on each node. The `kubelet` daemon takes a set of pod specifications, called *PodSpecs*, and ensures that the containers described in the PodSpecs are running and healthy.

The KubernetesNode role parameters are described in table 4.9:

Parameter	Description	Option to `kubelet`
`Kubernetes Cluster`	The Kubernetes cluster instance (pointer)	
`Address`	IP address to serve on (default: `0.0.0.0`)	`--address`
`Kubelet Port`	Port that the HTTP service of the node runs on (default: `10250`)	`--port`
`CNI plugin binaries path`	The full path of the directory in which to search for CNI plugin binaries. (default: `/cm/local/apps/kubernetes/current/bin/cni`)	`--cni-bin-dir`
`Enable Server`*	Enable server mode of Kubelet	`--enable-server`
`Host Network Sources`	List of sources from which Kubelet allows pods use of the host network (default: `file`)	`--host-network-sources`
`Hostname Override`	If non-empty, use this string as identification instead of the actual hostname (not set by default)	`--hostname-override`
`Manifests Path`	Path to the config file or directory of files (default: `/cm/local/apps/kubernetes/var/etc/manifests`)	`--pod-manifest-path`
`Network plugin`	The name of the network plugin to be invoked for various events in kubelet/pod lifecycle. (default: `cni`)	`--network-plugin`
`Spool`	Directory path for managing Kubelet files (default: `/cm/local/apps/kubernetes/var/kubelet`)	`--root-dir`
`Cgroup Root`	Optional root cgroup to use for pods	`--cgroup-root`
`Docker Endpoint`	Docker endpoint address to connect to (default: `unix:///var/run/docker.sock`)	`--docker-endpoint`
`Docker Spool`	Absolute path to the Docker state root directory (default: `/var/lib/docker`)	`--docker-root`
`Resource Container`	Absolute name of the resource-only container to create and run the Kubelet in (default: `/kubelet`)	`--resource-container`
`Allowed Privileged`*	If true, allow privileged containers. (default: `no`)	`--allow-privileged`
`Labels`	List of node labels	
`Register On Start`*	Register the node with the API server (default: `yes`)	`--register-node`
`Eviction minimum reclaim`	Minimum amount of resources reclaimed in an eviction (default: `imagefs.available=1Gi`)	`--eviction-minimum-reclaim`

...continues

...continued

Parameter	Description	Option to `kubelet`
`Hard eviction`	Hard eviction constraints (default: `imagefs.available<1%`)	`--eviction-hard`
`Max Pods`	Number of pods that can run on this node	`--max-pods`
`Max pod eviction grace period`	Maximum allowed grace period (in seconds) allowed to terminated pods (default: 60)	`--eviction-max-pod-grace-period`
`Soft eviction`	Soft eviction constraints (default: `imagefs.available<5%`)	`--eviction-soft`
`Soft eviction grace period`	Soft eviction grace period (default: `imagefs.available=1m30s`)	`--eviction-soft-grace-period`
`File Check Frequency`	Duration between checking configuration files for new data (default: `20s`)	`--file-check-frequency`
`HTTP Flush Frequency`	Duration between checking HTTP for new data (default: `20s`)	`--http-check-frequency`
`Node Status Update Frequency`	The absolute free disk space, in MB, to maintain (default: `10s`)	`--node-status-update-\ frequency`
`Run Once`*	If true, exit after spawning pods from local manifests or remote URLs (default: `no`)	`--runonce`
`Streaming Connection Idle Timeout`	Maximum time a streaming connection connection can be idle before the connection is automatically closed (default: `1h`)	`--streaming-connection-\ idle-timeout`
`Sync Frequency`	Maximum period between synchronizing running containers and config (default: `10s`)	`--sync-frequency`
`Image GC High Threshold`	Percent of disk usage after which image garbage collection is always run (default: `90`)	`--image-gc-high-\ threshold`
`Image GC Low Threshold`	Percent of disk usage before which image garbage collection is never run (default: `80`)	`--image-gc-low-\ threshold`
`Threshold`	maintain (default: `256`)	`threshold-mb`
`Oom Score Adjust`	The `oom_score_adj` value for the kube-proxy process, in range [-999, 999] (default: `-999`)	`--oom-score-adj`
`Log Level`	Log level (default: `0`)	`--v`
`Log To StdErr`*	Logging to STDERR means it gets into the systemd journal (default: `yes`)	`--logtostderr`
`Options`	Additional parameters for the kube-scheduler daemon	

...continues

...continued

Parameter	Description	Option to `kubelet`

* Boolean (takes `yes` or `no` as a value)

Table 4.9: KubernetesNode role parameters and `kubelet` *options*

Further details on the kubelet daemon can be found at `https://kubernetes.io/docs/admin/kubelet/`.

4.10 Security Model

The Kubernetes security model allows authentication using a certificate authority (CA), with the user and daemon certificates signed by a Kubernetes CA. The Kubernetes CA should not be confused with the cluster manager CA.

The cluster manager will create a CA specifically for issuing all Kubernetes-related certificates. The certificates are put into `/cm/local/apps/kubernetes/var/etc/` by default, and `/etc/kubernetes/` is made a link to this directory.

In Kubernetes terminology a user is a unique identity accessing the Kubernetes API server. The user may be a human or an automated process. For example an admin or a developer are human users, but kubelet represents an infrastructure user. Both types of users are authorized and authenticated in the same way against the API server.

Kubernetes uses client certificates, tokens, or HTTP basic authentication methods to authenticate users for API calls. The cluster manager configures client certificate usage by default. The authentication is performed by the API server which validates the user certificate using the common name part of the certificate subject.

In Kubernetes, authorization happens as a separate step from authentication. Authorization applies to all HTTP accesses on the main (secure) API server port. The cluster manager by default enables RBAC (Role-Based Access Control) combined with Node Authorization. The authorization check for any request thus takes the common name and/or organization part of the certificate subject to determine which roles the user or service has associated. Roles carry a certain set of privileges for resources within Kubernetes.

4.10.1 Kyverno

The cluster manager has support for the Kyverno policy engine (`https://kyverno.io/`). If Kyverno is installed, then Kubernetes Permissions Manager (section 4.16) creates policy manifests packed as a Helm chart for every user added to Kubernetes via `cm-kubernetes-setup`. In addition, a kyverno-policy chart is installed in enforce mode to implement Pod Security Standards (`https://kyverno.io/policies/`). During installation, some exclusions are added to the policies automatically to make chosen features of Kubernetes cluster work.

For every created user the following defaults are applied:

- The user has an associated service account with the same name
- A <*username*>-restricted namespace is created. So, for a user `john` the namespace is `john-restricted`.
- An `edit` cluster role is bound to the service account in <*username*>-restricted namespace. The user is allowed to create pods, services, configmaps, etc. in the namespace
- The user is allowed to list nodes in the cluster
- Kyverno policies are applied to the resources in <*username*>-restricted namespace or to pod created or updated by the associated user

  - If hostPath is not the home directory of the user (of the format `/home/`*<username>*) then the creation of the resource is denied
  - The UID and GID of the running process are set to the same value as the UID and GID of the PAM user

Modifications from the defaults are:

- If the `Allow any UID process in pods` checkbox is ticked, or if the `--allow-all-uids` argument is specified, then the UID and GID of the running process becomes the user's UID and GID only if the hostPath volume is specified. Otherwise it can be set to any UID and GID.
- Cluster roles can be set not only to
  - `edit`

  but also to
  - `view`
  - `admin`
  - `cluster-admin`

  More details on these roles can be found at:
  `https://kubernetes.io/docs/reference/access-authn-authz/rbac/#user-facing-roles`.
- In addition, the user can be given access to custom CRDs, such as Zalando Postgres Operator, Jupyter Operator or Google Spark Operator

4.10.2 PodSecurityPolicy

The cluster manager also has support for PodSecurityPolicy (PSP) in the Kubernetes API Server. PSP can be explicitly enabled or disabled (section 4.14). However, it should be noted that PodSecurityPolicy was deprecated in Kubernetes v1.21, and removed from Kubernetes in v1.25.

For each user, a PSP is generated and assigned using Helm charts generated by Kubernetes Permissions Manager. (section 4.16).

The following defaults are applied:

- Users can only mount their own home directory. They cannot mount other paths such as `/etc`
- Users cannot run privileged pods.
- Users can only bind on ports higher than 1024.
- Users can run with their own `uid` and `gid`.
- It is possible to allow users to run as `root`, *without* using `hostPath` volumes.

There are also

- default role bindings that grant access to the resources covered by the `kubectl get all` command.

A higher-level explanation about each of the resources is given in section 4.13.

These policies only do something if PodSecurityPolicies are enabled, or enforced by Kubernetes, which is not the default. Section 4.14 can be referred to for enabling and disabling PodSecurityPolicies.

4.11 Addition Of New Kubernetes Users

Bright users can use Kubernetes by making them Kubernetes users. This means having Kubernetes configuration and access set up for them. This can be carried out a via the `cm-kubernetes-setup` TUI utility, and choosing the `Add user` option (figure 4.4). The utility then prompts for

- a Kubernetes cluster
- a user name
- a namespace that the privileges are to be assigned to
- a role for the user, with choices provided from:
  - `cluster-admin`: cluster-wide administrator
  - `admin`: administrator
  - `edit`: regular user
  - `view`: read-only user
- a switch if the user is allowed to run as any user, including root, inside pods
- a comma-separated list of paths that the user is able to mount to pods
- the UIDs and GIDs for user processes in pods
- a list of the Kubernetes operators that a user can use

Based on the input, a YAML for the Kubernetes Permission Manager is generated. This in turn, creates a Helm chart with all the required roles, role bindings, and PSP or Kyverno rules.

Creation of the user also triggers CMDaemon to create certificate and configuration files in the `~/.kube` directory

4.11.1 Adding Users Non-Interactively With `cm-kubernetes-setup`

The `cm-kubernetes-setup` CLI wizard provides the following options:

```
cm-kubernetes-setup -h
usage: Kubernetes Setup cm-kubernetes-setup
        [-c <config_file>]
        [--cluster CLUSTER_NAME]
        [--skip-docker] [--skip-reboot]
        [--skip-image-update]
        [--add-user USERNAME_ADD] [--list-users] [--get-user GET_USER]
        [--modify-user USERNAME_MODIFY] [--remove-user USERNAME_REMOVE]
        [--namespace NAMESPACE] [--add-to-namespace] [--remove-from-namespace]
        [--role edit,admin,view,cluster-admin]
        [--runas-uid RUNAS_UID] [--runas-gids RUNAS_GIDS]
        [--user-paths USER_PATHS]
        [--allow-all-uids]
        [--operators OPERATORS]
        [--regenerate-certs]
        [--list-operators] [--psp]
        [--apparmor] [--disable-psp]
        [--update-addons] [--remove]
        [--yes-i-really-mean-it]
        [--remove-ceph-storage] [--pull]
        [--images IMAGES] [--nodes NODES]
        [--node-selector NODE_SELECTOR]
```

```
        [--allow-device-mapper] [-v]
        [--no-distro-checks] [--json]
        [--output-remote-execution-runner]
        [--on-error-action debug,remotedebug,undo,abort]
        [--skip-packages]
        [--min-reboot-timeout <reboot_timeout_seconds>]
        [--dev] [-h]
...
```

The user has to be a user that exists on the cluster already and available via PAM.

If `--add-to-namespace` is specified, then the namespace has to exist on the Kubernetes cluster already.

Example

```
cm-kubernetes-setup --add-user john
```

The preceding example creates a user `john` for the default `john-restricted` namespace. It also assigns the `edit` role, and gives permission to run processes in the pod with the current UID/GIDs of the user. The ability to mount `~/john` as a hostPath is also provided.

A way to assign any of the default Kubernetes user-facing roles is also provided by using `--role` key, as documented at `https://kubernetes.io/docs/reference/access-authn-authz/rbac/#user-facing-roles`

The possible roles are: `view`, `edit`, `admin`, and `cluster-admin`.

Example

```
cm-kubernetes-setup --add-user john --role view
```

The preceding example creates a user `john` with `view` privileges only, for the default `john-restricted` namespace.

Example

```
cm-kubernetes-setup --add-user john --user-paths /home/john,/scratch --allow-all-uids \
   --operators cm-jupyter-kernel-operator
```

The preceding example creates a user `john` with the following privileges:

- `edit` privileges
- able to mount `/home/john` and `/scratch` as hostPath volume, when the process runs with UID/GIDs taken from the PAM subsystem on the moment of creation
- able to run as any user, including root (attempts to mount any hostPath volume will be rejected)
- access to the Jupyter Kernel Operator, i.e. with access to the resource `kind: CmKubernetesOperatorPermissionsJupyterKernel`

4.12 Getting Information And Modifying Existing Kubernetes Users

It is possible to edit user properties and permissions. `cm-kubernetes-setup` provides 2 ways of doing it: interactively or via CLI options.

Modifying users can be done interactively by choosing `Modify User` in the `cm-kubernetes-setup` main menu. Guidance is then given on choosing the cluster, users, and on modifying permissions.

Modifying users can also be done via CLI options, by specifying the `--modify-user` argument:

Example

```
cm-kubernetes-setup --add-user john --user-paths /home/john \
    --allow-all-uids --operators cm-jupyter-kernel-operator
```

In addition to what is specified on creation with the `--add-user` argument in the preceding example, the following example adds permission to mount the `/scratch` hostPath into pods, and gives access to the Zalando PostgreSQL Operator:

```
cm-kubernetes-setup --modify-user john --user-paths /home/john,/scratch \
    --allow-all-uids --operators cm-jupyter-kernel-operator,cm-kubernetes-postgresql-operator
```

Information about existing users can be found with:

Example

```
cm-kubernetes-setup --list-users
```

Permission for user `john` to operate in the `dev` namespace can be added with:

Example

```
kubectl create namespace dev
cm-kubernetes-setup --modify-user john --namespace dev --add-to-namespace
```

Permission for the user `john` to operate in the `dev` namespace can be revoked with:

Example

```
kubectl create namespace dev
cm-kubernetes-setup --modify-user john --namespace dev --remove-from-namespace
```

4.13 List Of Resources Defined For Users

These resources are rendered by the Permission Manager Operator, and can therefore be found inside Kubernetes.

The Role Bindings Deployed For Every User By Default

By default, the role bindings deployed for the user `john` created in the preceding section are:

- `ClusterRole/john-nodes` (in namespace `john-restricted`)
- `ClusterRoleBinding/john-nodes` (in namespace `john-restricted`)

User `john` is given read-only rights for the Nodes resource (for `kubectl get nodes`).

The Secure Namespace Related Resources

The secure namespace for user `john` is:

- `Namespace/john-restricted`

The service account used by `john`:

- `ServiceAccount/john` (in namespace `john-restricted`)

This is found referenced, for example, in `john`'s `$HOME/.kube/config`.

The PodSecurityPolicy that defines the user can run non-privileged pods, and use only ports above `1024`, and so on:

- `PodSecurityPolicy/john-restricted` (in namespace `john-restricted`)

More details on this can be found in section 4.10, page 58. This policy will only do something as soon as the `PodSecurityPolicy` Admission Controller is enabled in the API server.

A PodSecurityPolicy that defines the user can run as root as well, but *without* hostPath volumes:

- `PodSecurityPolicy/john-restricted-root` (in namespace `john-restricted`)

To give the aforementioned privileges to `john`'s secure namespace, so that `john` can run workloads, execute `kubectl get all`, and more:

- `Role/john-restricted` (in namespace `john-restricted`)
- `RoleBinding/john-restricted` (in namespace `john-restricted`)

The RoleBinding assigns it to the user `john` and ServiceAccount account for `john`. The upstream documentation at `https://kubernetes.io/docs/reference/access-authn-authz/service-accounts-admin` has more details on this.

The user `john` can be given the ability to use the PodSecurityPolicy defined earlier in his secure namespace, but also in other namespaces:

- `ClusterRole/john-psp` (in namespace `john-restricted`)
- `ClusterRoleBinding/john-psp` (in namespace `john-restricted`)

The same ability can be given for the second `root but no hostPath` PodSecurityPolicy:

- `ClusterRole/john-psp-root` (in namespace `john-restricted`)
- `ClusterRoleBinding/john-psp-root` (in namespace `john-restricted`)

If the Kyverno engine is installed then several policies are added:

- `clusterpolicies.kyverno.io/john-*-drop-privs-w-hostpath*`: Policy to modify pod manifests to run process with specified UID/GID
- `clusterpolicies.kyverno.io/john-*-limit-hostpath-vols`: Policy to deny pods if hostPath volumes does not match specified paths

The full content of all the documents created for the user can be viewed by checking the generated Helm manifest:

Example

```
helm get manifest -n cm-permissions john-XXXXXX
```

4.14 Pod Security Policies

4.14.1 Enabling Pod Security Policies For Kubernetes

Without additional policy engines, Kubernetes has very few restrictions on users. For a more fine-grained authorization it is possible to enable Pod Security Policies (PSPs).

This can be done via the `cm-kubernetes-setup` TUI wizard and choosing "`Enable PSP`". Alternatively, it can be done non-interactively using `cm-kubernetes-setup --psp`.

Optionally, AppArmor can be enabled as well, using non-interactive command option `-apparmor`.

Enabling PSP creates two new Applications within the Kubernetes AppGroup `system`:

- `psp`: this defines the policy and roles for privileged services. The cluster administrator needs to bind these to services typically running in the `kube-system` namespace.
- `psp_system`: this is auto-generated by the wizard. It binds the previously-mentioned privileges to service accounts, for services defined in the system namespaces. This way Calico, CoreDNS, Ingress, and so on, can still function.

The wizard also removes access to the default namespace for existing users, and it restarts the Kubernetes API Server with the PodSecurityPolicy feature enabled to enforce all privileges.

Each user should already have their own <*user*>-restricted namespace and privileges to work within this namespace. After the Kubernetes API Server is enabled with the PodSecurityPolicy feature, these policies are enforced after the API server has restarted.

A useful command to check exactly what a user—for example, the user test—can do is the following:

```
kubectl --kubeconfig=/home/test/.kube/config auth can-i --list --namespace=test-restricted
```

4.14.2 Disabling Pod Security Policies For Kubernetes

Disabling PSP can be done via the cm-kubernetes-setup TUI wizard, and choosing Disable PSP. It can alternatively be carried out non-interactively using cm-kubernetes-setup --disable-psp.

When disabling, it should be noted that:

- Existing users are not automatically re-added to the default namespace.
- Policies are still defined as resources, but are no longer enforced. This may result in more privileges for users then they had before. That is, they may be able to run as root in containers again.

4.14.3 Enabling Manually Via cmsh Instead

The PodSecurityPolicy feature is an admissioncontrol setting that can be added via cmsh:

```
[cluster->configurationoverlay[kube-default-master]->roles[Kubernetes::ApiServer]]% get admissioncontrol
NamespaceLifecycle
LimitRanger
ServiceAccount
DefaultStorageClass
DefaultTolerationSeconds
MutatingAdmissionWebhook
ValidatingAdmissionWebhook
ResourceQuota
PodSecurityPolicy
```

Adding or removing PodSecurityPolicy from here in cmsh triggers CMDaemon to restart the kube-apiserver services.

It could be that running pods are not affected. However, if the cluster administrator re-creates them, then it may be that new pods are not created by ReplicaSets, DaemonSets and similar. Errors may show up as follows:

```
Warning  FailedCreate  5m22s (x20 over 5m33s)  replicaset-controller  Error creating:\
pods "coredns-b5cdc886c-" is forbidden: unable to validate against any pod security policy: \
[spec.containers[0].securityContext.capabilities.add: Invalid value: "NET_BIND_SERVICE": \
capability may not be added spec.containers[0].securityContext.capabilities.add: Invalid \
value: "NET_BIND_SERVICE": capability may not be added spec.containers[0].securityContext.capabilities.add: \
Invalid value: "NET_BIND_SERVICE": capability may not be added]
```

Also, the kubelet services themselves will not have the proper privileges to manage their pods. If the PodSecurityPolicy value is enabled in the admissioncontrol settings, then the cluster administrator must be explicit and define the Pod Security Policy (PSP).

4.14.4 The psp Application

The purpose of this application is to define a PSP for these more privileged components. It starts by assigning it to the kubelet component. Other system components such as CoreDNS or Calico are not yet assigned to this PSP.

In cmsh, parts of the configuration setting can be seen with:

```
[cluster->kubernetes[default]->appgroups[system]->applications[psp]]% get config | grep -E "^kind|^  name" \
 |grep kind -A 2
kind: PodSecurityPolicy
  name: privileged
kind: ClusterRole
  name: privileged-psp
kind: RoleBinding
  name: privileged-psp-nodes
  namespace: kube-system
```

In the preceding configuration, resources define `PodSecurityPolicy/privileged`, which the cluster administrator binds to `Group/system:nodes` with a `RoleBinding/privileged-psp-nodes`, and using a role `ClusterRole/privileged-psp`. The `Group/system:nodes` allows access to resources required by the `kubelet` component, including read access to secrets, and write access to pods.

The `psp` Application Configuration

The full YAML configuration for the psp application follows:

```
# privileged psp to be used for kube system services only
apiVersion: policy/v1beta1
kind: PodSecurityPolicy
metadata:
  name: privileged
  annotations:
    seccomp.security.alpha.kubernetes.io/allowedProfileNames: '*'
spec:
  privileged: true
  allowPrivilegeEscalation: false
  allowedCapabilities: ['*']
  volumes: ['*']
  hostNetwork: true
  hostPorts:
    - min: 0
      max: 65535
  hostIPC: true
  hostPID: true
  runAsUser:
    rule: 'RunAsAny'
  seLinux:
    rule: 'RunAsAny'
  supplementalGroups:
    rule: 'RunAsAny'
  fsGroup:
    rule: 'RunAsAny'
---
# cluster role privileged psp
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: privileged-psp
rules:
  - apiGroups: ['policy']
    resources: ['podsecuritypolicies']
    verbs:     ['use']
    resourceNames: ['privileged']
  - apiGroups: ['extensions']
```

```
    resources: ['podsecuritypolicies']
    verbs:     ['use']
    resourceNames: ['privileged']
---
# role binding for privileged psp to system:nodes
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: privileged-psp-nodes
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: privileged-psp
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: Group
  apiGroup: rbac.authorization.k8s.io
  name: system:nodes
```

4.14.5 The psp_system Application:

The purpose of this application is to also assign the more privileged PodSecurityPolicy created in the psp application to all system applications. To be more specific, this is done by binding it to all the ServiceAccounts and appropriate Namespaces for those services.

In cmsh the binding configuration can be viewed with:

```
[cluster->kubernetes[default]->appgroups[system]->applications[psp_system]]% get config|grep -E "^kind|^  name" \
        |grep kind -A 2
kind: RoleBinding
  name: privileged-psp-calico-kube-controllers
  namespace: kube-system
--
kind: RoleBinding
  name: privileged-psp-nginx-ingress-serviceaccount
  namespace: ingress-nginx
--
kind: RoleBinding
  name: privileged-psp-metrics-server
  namespace: kube-system
--
kind: RoleBinding
  name: privileged-psp-coredns
  namespace: kube-system
--
kind: RoleBinding
  name: privileged-psp-kubernetes-dashboard
  namespace: kubernetes-dashboard
--
kind: RoleBinding
  name: privileged-psp-calico-node
  namespace: kube-system
```

This psp_system is generated by reading all the existing system add-ons from cmsh, and binding the ClusterRole/privileged-psp to all ServiceAccounts used by those services with a RoleBinding/privileged-psp-<*serviceaccount*> for each of them.

The psp_system Application Configuration

The YAML configuration for the psp_system application can be seen with:

```
[cluster->kubernetes[default]->appgroups[system]->applications[psp_system]]% get config
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: privileged-psp-calico-kube-controllers
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: privileged-psp
  apiGroup: rbac.authorization.k8s.io
subjects:
- kind: ServiceAccount
  name: calico-kube-controllers
  namespace: kube-system
---
...
```

The display in the preceding session is truncated, but it is followed by a long list of similar blocks for each of the service accounts used by Calico, DNS, metrics server, device plugins, and so on.

The role has been assigned specifically to ServiceAccounts. One alternative way could be to assign once to the system:serviceaccounts group:

```
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: privileged-psp-system-serviceaccounts
  namespace: kube-system
roleRef:
  kind: ClusterRole
  name: privileged-psp
  apiGroup: rbac.authorization.k8s.io
subjects:
- apiGroup: rbac.authorization.k8s.io
  kind: Group
  name: system:serviceaccounts
```

It should be noted that a few more bindings may still be required for multiple namespaces (kube-system, kube-dashboard, and so on).

4.14.6 Users And PodSecurityPolicies

When PodSecurityPolicies have not been enabled from the start, it is possible that users are already running pods that do not necessarily adhere to the new policy. For example pods that have mounted paths outside of their home directory, or are running privileged containers. These may keep working, at least until the Kubernetes scheduler decides to re-schedule them, or until they are terminated.

4.15 Kyverno

Kyverno (https://kyverno.io/) is a policy engine designed for Kubernetes. With Kyverno, policies are managed as Kubernetes resources, and no new language is required to write policies. This allows the use of familiar tools such as kubectl, git, and kustomize to manage policies. Kyverno policies can validate, mutate, and generate Kubernetes resources, as well as ensure OCI image supply chain security.

4.15.1 Kyverno Installation

Kyverno engine and Kyverno policy Helm charts can be installed as a part of `cm-kubernetes-setup`:

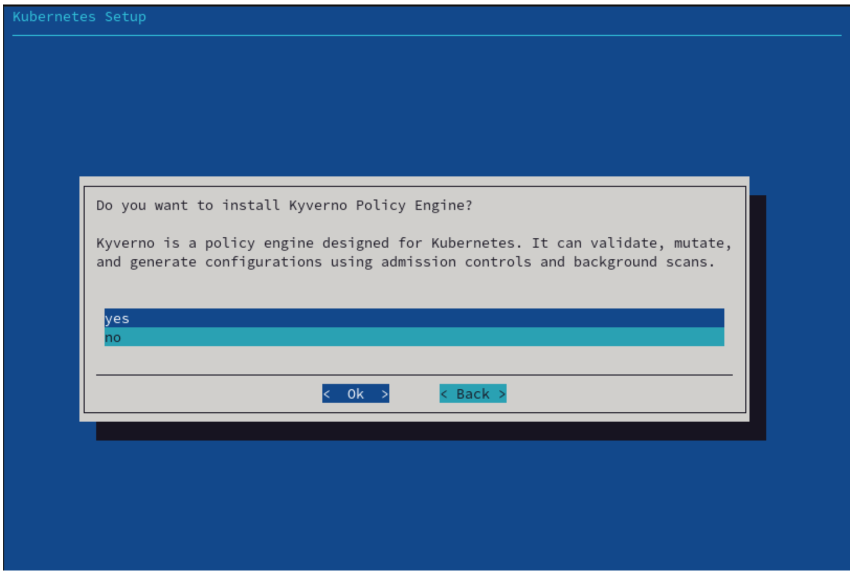

Figure 4.7: Choosing Kyverno installation

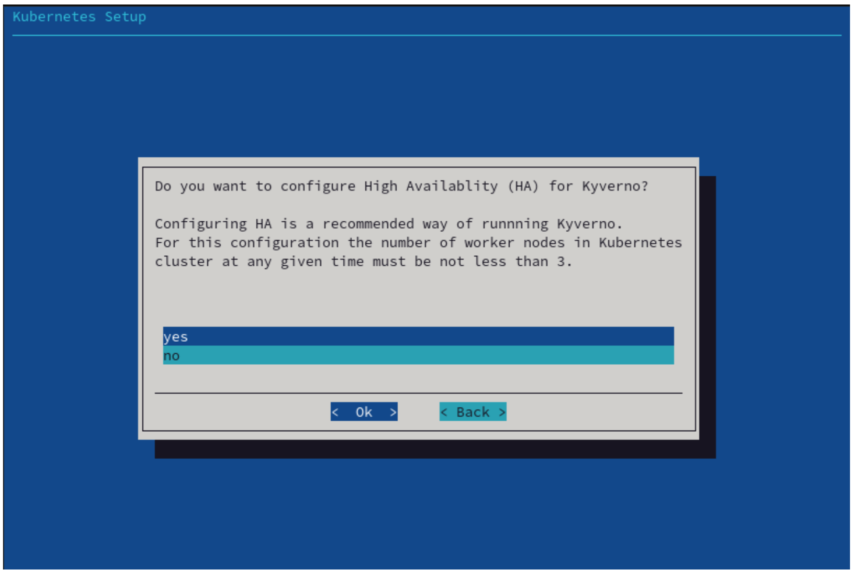

Figure 4.8: Kyverno high availability setup

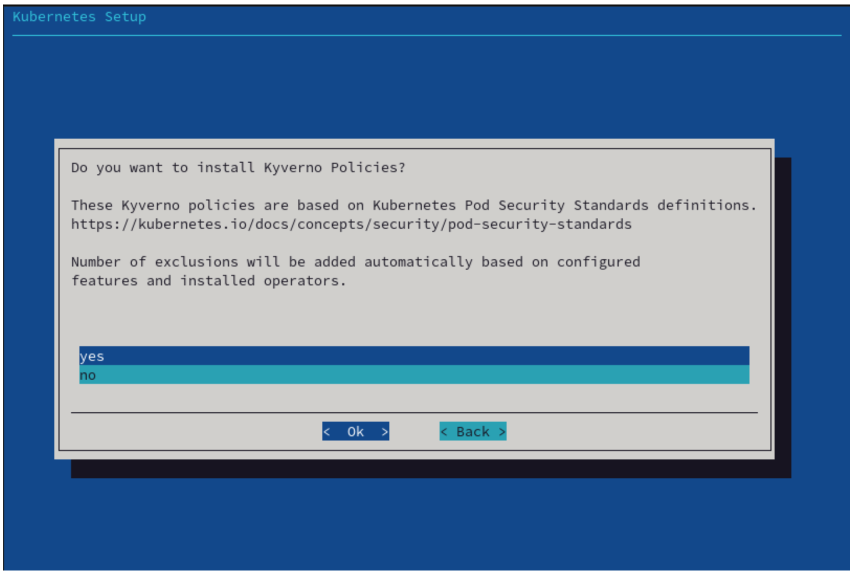

Figure 4.9: Kyverno policy setup

The installation adds 2 Helm charts in the namespace 'kyverno':

```
[root@bright92 ~]# module load kubernetes/
[root@bright92 ~]# helm list -n kyverno
NAME             NAMESPACE  STATUS   ...  CHART                   APP VERSION
kyverno          kyverno    deployed ...  kyverno-v2.5.2          v1.7.2
kyverno-policies kyverno    deployed ...  kyverno-policies-v2.5.2 v1.7.2
```

If the HA option is chosen, then the replica count value is set to 3.

```
[root@bright92 ~]# helm get values -n kyverno kyverno
USER-SUPPLIED VALUES:
replicaCount: 3
```

This means that at any given time Kubernetes scheduler tries to run 3 pods at the same time:

```
[root@bright92 ~]# kubectl get pods -n kyverno
NAME                       READY   STATUS    RESTARTS   AGE
kyverno-5bfb99b9c9-ddmmw   1/1     Running   0          1h
kyverno-5bfb99b9c9-hgfsc   1/1     Running   0          1h
kyverno-5bfb99b9c9-n67rv   1/1     Running   0          1h
```

4.15.2 Kyverno Policies

It is also recommended to install Kyverno policies in order to enforce Pod Security Standards `https://kyverno.io/policies/`. NVIDIA Bright Cluster Manager configures Kyverno policies in 'enforce' mode, adding service namespaces as exclusions. The list of namespaces to be excluded from particular policies depend on the selected features during install:

```
[root@bright92 ~]# helm get values -n kyverno kyverno-policies
USER-SUPPLIED VALUES:
validationFailureAction: enforce
policyExclude:
  disallow-host-namespaces:
    any:
    - resources:
        kinds:
```

```
        - Pod
        namespaces:
        - default
        - prometheus
  disallow-host-path:
    any:
    - resources:
        kinds:
        - Pod
        namespaces:
        - default
        - local-path-storage
        - '*-restricted'
        - prometheus
        - kube-system
        - gpu-operator
  disallow-host-ports:
    any:
    - resources:
        kinds:
        - Pod
        namespaces:
        - default
        - prometheus
```

In the preceding output, all namespaces that match the wildcard `*-restricted` are excluded from the policy named 'disallow-host-path' (`https://kyverno.io/policies/pod-security/baseline/disallow-host-path/disallow-host-path/`). This means that, without additional restrictions, all pods in the user namespaces can mount any host path from an underlying node.

To prevent that Kubernetes Permission Manager creates a Kyverno Cluster Policy for every newly-created user, and restricts the hostPath to only the home directory of the user:

```
[root@bright92 ~]# kubectl get clusterpolicies.kyverno.io | grep john
john-n730sr0-drop-privs-w-hostpath                         false        enforce    true
john-n730sr0-drop-privs-w-hostpath-containers              false        enforce    true
john-n730sr0-drop-privs-w-hostpath-initcontainers          false        enforce    true
john-n730sr0-limit-hostpath-vols                           false        enforce    true
```

4.16 Kubernetes Permission Manager

The Kubernetes permission manager is a custom operator based on Helm. It helps to manage user and system account permissions, roles, role bindings and pod security policies. The operator itself is packed and distributed as a Helm chart, so it can be installed during Kubernetes cluster creation via the `cm-kubernetes-setup` TUI. The Helm chart for the operator is located in `/cm/shared/apps/kubernetes-permissions-manager/current/helm`. The output to the following command shows if it is installed:

```
[root@bright92 ~]# module load kubernetes/
[root@bright92 ~]# helm list -n cm
NAME                    NAMESPACE  STATUS    ...  CHART                                        APP VERSION
local-path-provisioner  cm         deployed  ...  cm-kubernetes-local-path-provisioner-0.0.20  0.0.20
permissions-manager     cm         deployed  ...  cm-kubernetes-permissions-manager-0.0.1      0.0.1
```

The Helm chart of the operator includes custom resource definitions (CRD), and makes it possible for the administrator to manage resources using the `kubectl` tool:

Example

```
[root@bright92 ~]# cat > permissions.yaml<<EOF
apiVersion: charts.brightcomputing.com/v1alpha1
kind: CmKubernetesPermissionUser
metadata:
  labels:
    namespace: cmsupport-restricted
    username: cmsupport
  name: cmsupport-c7tk7ft
  namespace: cm-permissions
spec:
  allow_all_uids: false
  allowPrivilegeEscalation: false
  allowPrivileged: false
  create_namespace: true
  create_service_account: true
  gids:
  - 1000
  namespace: cmsupport-restricted
  psp_spec_override:
  role: edit
  uid: 1000
  user_paths:
  - /home/cmsupport
  username: cmsupport
EOF
[root@bright92 ~]# kubectl apply -f permissions.yaml
cmkubernetespermissionuser.charts.brightcomputing.com/cmsupport-c7tk7ft created
[root@bright92 ~]# kubectl get cmkubernetespermissionusers -A
NAMESPACE        NAME                AGE
cm-permissions   cmsupport-c7tk7ft   22s
```

At the time of writing of this section (December 2021), the permission manager handles these 4 CRDs:

1. cmkubernetespermissionusers to manage user access to generic resources of the cluster, such as pods, services, secrets, configmaps, etc.

2. cmkubernetesoperatorpermissionsjupyterkernels to manage access to the Jupyter Kernels.

3. cmkubernetesoperatorpermissionspostgresqls to manage access to the Zalando PostreSQL operator (https://github.com/zalando/postgres-operator).

4. cmkubernetesoperatorpermissionssparks to manage access to the Google Spark operator (https://github.com/GoogleCloudPlatform/spark-on-k8s-operator).

Providing access to third party operators is necessary if pod security policy is enabled. This is because, by default, not only does the user have no access to CRDs, but also the service accounts of the third party operators have no access to the user namespace.

The following example is a YAML document that provides access to the Jupyter Kernel Operator:

```
apiVersion: charts.brightcomputing.com/v1alpha1
kind: CmKubernetesOperatorPermissionsJupyterKernel
metadata:
  labels:
    namespace: cmsupport-restricted
    username: cmsupport
```

```
  name: cmsupport-unz4wlf
  namespace: cm-permissions
spec:
  namespace: cmsupport-restricted
  username: cmsupport
```

Every installed CRD document triggers the Kubernetes permission operator to create a corresponding Helm chart:

```
# helm get values -n cm-permissions cmsupport-unz4wlf
USER-SUPPLIED VALUES:
namespace: cmsupport-restricted
username: cmsupport

# helm get manifest -n cm-permissions cmsupport-unz4wlf
---
# Source: cm-kubernetes-operator-permissions-jupyter-kernel/templates/user-permissions.yaml
# Bind policy to service user
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: cmsupport-unz4wlf-cmsupport-psp
  labels:
    helm.sh/chart: cmsupport-unz4wlf
    app.kubernetes.io/name: cm-kubernetes-operator-permissions-jupyter-kernel
    app.kubernetes.io/instance: cmsupport-unz4wlf
    app.kubernetes.io/version: "0.0.1"
    app.kubernetes.io/managed-by: Helm
subjects:
  - kind: ServiceAccount
    name: default
    namespace: cmsupport-restricted
roleRef:
  kind: ClusterRole
  name: cmsupport-psp
  apiGroup: rbac.authorization.k8s.io
...
```

It is also possible to customize the resulting Helm chart by specifying additional values to specify a section of the CRD. Available values for the Jupyter kernel can be checked using the following command:

```
kubectl exec -it  -n cmkpm-system \
    $(kubectl get pods -n cmkpm-system -l control-plane=controller-manager -o name) \
    -c manager -- \
    cat /opt/helm/helm-charts/cm-kubernetes-operator-permissions-jupyter-kernel/values.yaml
```

Similarly, tunables for the generic user permissions of the user are available via:

```
kubectl exec -it  -n cmkpm-system \
    $(kubectl get pods -n cmkpm-system -l control-plane=controller-manager -o name) \
    -c manager -- cat /opt/helm/helm-charts/cm-kubernetes-permission-user/values.yaml
```

The output should be similar to:

```
username: ""  # name of the user
create_service_account: true  # whether to create kubernetes serviceaccount for the user
role: edit  # user role
user_paths: []  # hostPath user able to mount to pods
```

```
uid: -1  # UID to run process inside pods
gids: [-1]  # list of the GIDs for process inside pods
namespace: ""  # namespace to give user permissions to
create_namespace: true  # create or not the namespace
allowPrivilegeEscalation: false
allowPrivileged: false
allow_all_uids: true  # allow or not to run process as any user including root
                       # if 'true' and process is run not with user's UID, then
                       # all hostPath volumes are denied
psp_spec_override:  # custom PSP definition for the user
```

4.17 Providing Access To External Users

To provide access to users on an external network, the requirements are:

- for `kubectl`, an entry in the company/internal DNS server should resolve the external FQDN to the head node or to one of the nodes where Kubernetes is running;
- for the Kubernetes Dashboard, `dashboard` is a subdomain that must be included as a DNS entry under the external FQDN.

The external FQDN, which is set during the Kubernetes cluster setup, is the first item in the list of trusted domains. This can be retrieved from the Kubernetes cluster entity with `cmsh` as follows:

Example

```
[bright92->kubernetes[default]]% get trusteddomains
bright92.example.com
kubernetes
kubernetes.default
kubernetes.default.svc
master
localhost
```

In the preceding example, the FQDN of the cluster is `bright92.example.com`. The cluster administrator managing their own cluster will have another FQDN, and not this FQDN.

For `kubectl`, the Kubernetes API server proxy port should be open to the external network. The proxy port can be retrieved from the Kubernetes cluster entity as follows:

```
[bright92->kubernetes[default]]% get kubernetesapiserverproxyport
10443
```

For the Kubernetes Dashboard, the Ingress Controller HTTPS port should be open to the external network. This port, by default with a value of 30443, can be retrieved from the `ingress_controller` add-on environment:

Example

```
[bright92->kubernetes[default]]% appgroups
[bright92->kubernetes[default]->appgroups]% applications system
[bright92->kubernetes[default]->appgroups[system]->applications]% environment ingress_controller
[bright92->...applications[ingress_controller]->environment]% list
Name (key)                   Value                                  Nodes environment
---------------------------- -------------------------------------- ------------------
CM_KUBE_EXTERNAL_FQDN        bright92.example.com                   yes
CM_KUBE_INGRESS_HTTPS_PORT   30443                                  yes
CM_KUBE_INGRESS_HTTP_PORT    30080                                  yes
ingress_controller_label     brightcomputing.com/ingress-controller no
replicas                     1                                      no
```

If exposing the Kubernetes API server to the external network is selected during setup with cm-kubernetes-setup, then the HTTPS and HTTP ports in the preceding example are opened on the Shorewall service that runs on the head node. Exposure to the external network is enabled by default.

Convention Of Using A Domain Name As A Prefix Label

In the preceding example, the brightcomputing.com prefix that is part of the value for ingress_controller_label is just a label rather than a domain. The reason that prefix is used is that it simply follows the convention of using domain names as labels, such as is done by the Kubernetes community (domain: kubernetes.io) and RHEL OpenShift (domain: openshift.io). The prefix brightcomputing.com could equally well have been the prefix brightaccess instead. However it is probably less confusing now to follow the established convention. So that is what is done here for the label.

Users Can Access The Kubernetes Dashboard

Users can access the Kubernetes Dashboard using dashboard. By default, the URL takes the FQDN and the port value along with the dashboard subdomain, and has the form:

https://dashboard.<*CM_KUBE_EXTERNAL_FQDN*>:<*CM_KUBE_INGRESS_HTTPS_PORT*>

So, for example, it could be something like:

Example

```
https://dashboard.bright92.example.com:30443
```

Ingress Configuration For Dashboard In cmsh

The default Ingress rule described earlier can be found as an object within cmsh:

```
[bright92->kubernetes[default]->appgroups[system]->applications[dashboard_ingress]]% get config
apiVersion: networking.k8s.io/v1beta1
kind: Ingress
metadata:
  name: kubernetes-dashboard
  namespace: kubernetes-dashboard
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/secure-backends: "true"
    nginx.ingress.kubernetes.io/ssl-passthrough: "true"
    nginx.ingress.kubernetes.io/backend-protocol: "HTTPS"
spec:
  rules:
   - host: "dashboard.$CM_KUBE_EXTERNAL_FQDN"
     http:
       paths:
       - path: /
         backend:
           serviceName: kubernetes-dashboard
           servicePort: 443
```

Using kubectl, the Ingress resource can be found with:

```
bash$ kubectl get ingress -n kubernetes-dashboard
NAME                   HOSTS                            ADDRESS          PORTS    AGE
kubernetes-dashboard   dashboard.cluster1.local         10.150.153.251   80       45h
```

The official documentation for Ingress, at `https://v1-16.docs.kubernetes.io/docs/concepts/services-networking/ingress/`, explains it well. Path rewrites without domain names can also be used to set up Ingress with multiple backends (`serviceName` and servicePort pairs), without having to deal with setting up a DNS.

Ingress Controller Running On Compute Nodes

For scenarios where the head node is not involved in a Kubernetes setup, the cluster manager does not currently set up any forwarding for the Ingress Controller. The cluster manager does set up an NGINX proxy to expose the Kubernetes API Server in such cases, and accessing the Dashboard can then be done with the `kubectl proxy` approach.

For now a workaround to forward Ingress to a compute node can be achieved with port-forwarding, for example by adding the following line to `/etc/shorewall/rules` in Shorewall (section 7.2 of the *Installation Manual*):

Example

```
DNAT     net            nat:10.141.0.1:30443   tcp     30443
```

Using One Ingress Controller For Multiple Kubernetes Clusters

The cluster manager does not offer an out-of-the-box solution for one Ingress Controller with multiple Kubernetes clusters. This configuration can be achieved by configuring software such as NGINX to proxy, based on the domain name to the appropriate backend(s).

4.18 Networking Model

Kubernetes expects all pods to have unique IP addresses, which are reachable from within the cluster. This can be implemented in several ways, including adding pod network interfaces to a network bridge created on each host, or by using 3rd party tools to manage pod virtual networks.

Since NVIDIA Bright Cluster Manager version 9.2, the pod network provider is Calico (`https://www.projectcalico.org/`). Calico uses the Border Gateway Protocol (BGP) to distribute routes for every Kubernetes pod. This allows the Kubernetes cluster to be integrated without the need for overlays (IP-in-IP). Calico is particularly suitable for large Kubernetes deployments on bare metal, or in private clouds. This is because for larger deployments the performance and complexity costs of overlay networks can become significant.

4.19 Kubernetes Monitoring

When `cm-kubernetes-setup` is run, it configures the following Kubernetes-related health checks:

1. `KubernetesChildNode`: checks if all the expected agents and services are up and running for active nodes
2. `KubernetesComponentsStatus`: checks if all the daemons running on a node are healthy
3. `KubernetesNodesStatus`: checks if Kubernetes nodes have a status of `Ready`
4. `KubernetesPodsStatus`: checks if all the pods are in one of these states: `Running`, `Succeeded`, or `Pending`

4.20 Local Path Storage Class

For storage, instead of creating Kubernetes PersistentVolumes every time, a modern and practical way is to use the StorageClass feature.

Further documentation on StorageClass is available at:

- `http://blog.kubernetes.io/2016/10/dynamic-provisioning-and-storage-in-kubernetes.html`

- https://kubernetes.io/docs/concepts/storage/persistent-volumes/#storageclasses

As a part of initial installation it is possible to choose a Local Path Storage class to utilize the shared storage mounted on every node of the Kubernetes cluster. Possible options include any POSIX shared filesystems, such as NFS, BeeGFS, LustreFS, etc.

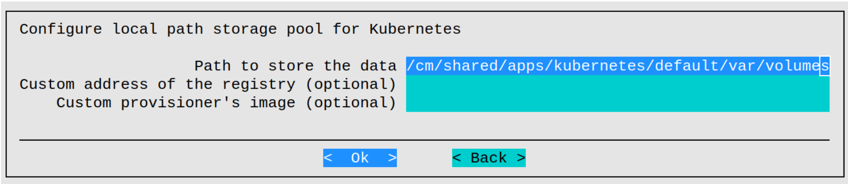

Figure 4.10: Kubernetes TUI Session: Local Storage Configuration

During setup, the installation wizard asks for a path for where Kubernetes physical volumes (PV) will be physically located. This path should be located on a shared filesystem accessible from all nodes.

After installation, the storage class can be seen to be available with:

```
# kubectl get storageclasses.storage.k8s.io
NAME                  PROVISIONER           RECLAIMPOLICY   VOLUMEBINDINGMODE   ALLOWVOLUMEEXPANSION   AGE
local-path (default)  rancher.io/local-path  Delete          Immediate           false                  1h
```

Users of the cluster can then freely create persistent volume claims (PVC) resources and use them in running pods.

4.21 Setup Of A Storage Class For Ceph

Pods running on Kubernetes can use Ceph as a distributed storage system to store data in a persistent way.

This section assumes a working Ceph cluster. Ceph installation for the cluster manager is covered in Chapter 9 of the *Administrator Manual*.

A new pool kube can be created with a replication factor of 3:

Example

```
[root@bright92 ~]# ceph osd pool create kube 100 100
pool 'kube' created
[root@bright92 ~]# ceph osd pool set kube size 3
set ppol 1 size to 3
[root@bright92 ~]# ceph osd pool set kube min_size 1
set pool 1 min_size to 1
```

The parameters settings in the preceding example are documented at the Ceph website, at

- http://docs.ceph.com/docs/master/rados/operations/pools/ for documentation on Ceph operations
- http://docs.ceph.com/docs/master/rados/configuration/pool-pg-config-ref/ for documentation on Ceph pool and PG (placement group) configuration

The pods of a given namespace have to have access to the Ceph RBD images created to back the volumes.

A kube client can be created with:

Example

```
[root@bright92 ~]# ceph auth get-or-create client.kube mon 'allow r' osd 'allow rwx pool=kube'
[client.kube]
        key = AQCnOvdZpYewBBAAWv1d7c7/XbEvj7QO7N0THg==
```

A list of the current users, and their access control can be viewed with (some output elided):

Example

```
[root@bright92 ~]# ceph auth list
installed auth entries:

osd.0
        key: AQD9M/dZw8HPNRAAT+X8mGSgRUkjLnQo38j4EA==
        caps: [mon] allow rwx
        caps: [osd] allow *
osd.1
...
client.admin
        key: AQCnM/dZONOPMxAAwqY9ADbJV+6i2Uq/ZNqh5A==
        auid: 0
        caps: [mds] allow *
        caps: [mgr] allow *
        caps: [mon] allow *
        caps: [osd] allow *
...
client.kube
        key: AQCnOvdZpYewBBAAWv1d7c7/XbEvj7QO7N0THg==
        caps: [mon] allow r
        caps: [osd] allow rwx pool=kube
```

The `admin` user must be able to create images in the pool. The `admin` configuration must therefore look like the section for `client.admin` in the preceding example.

Similarly, the `kube` user must be able to map images. The `kube` configuration must therefore look similar to the section for `client.kube` in the preceding example.

A Kubernetes secret must be created in the `kube-system` namespace, using the Ceph admin key:

```
[root@bright92 ~]# kubectl create secret generic ceph-secret --type="kubernetes.io/rbd" \
 --from-literal=key=$(ceph auth get-key client.admin) --namespace=kube-system
secret "ceph-secret" created
```

A Kubernetes secret must be created in the `default` namespace, and in every Kubernetes namespace that needs storage, using the Ceph user key:

```
[root@bright92 ~]# kubectl create secret generic ceph-secret-user --type="kubernetes.io/rbd" \
 --from-literal=key=$(ceph auth get-key client.kube) --namespace=default
secret "ceph-secret-user" created
```

Ceph monitor <*IP address*>:<*port*> values can be found by running `ceph mon stat`:

Example

```
[root@bright92 ~]# ceph mon stat
e1: 3 mons at {node001=10.141.0.1:6789/0,node002=10.141.0.2:6789/0,node003=10.141.0.3:6789/0},\
 election epoch 38, quorum 0,1,2 node001,node002,node003
```

A `storage-class.yml` file can then be created, similar to:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: fast
provisioner: kubernetes.io/rbd
parameters:
  monitors: 10.141.0.1:6789,10.141.0.2:6789,10.141.0.3:6789
  adminId: admin
  adminSecretName: ceph-secret
  adminSecretNamespace: kube-system
  pool: kube
  userId: kube
  userSecretName: ceph-secret-user
```

Details about the `StorageClass` parameters can be found at: `https://kubernetes.io/docs/concepts/storage/persistent-volumes/#ceph-rbd`

The Kubernetes storage class for Ceph RBD can now be created:

```
[root@bright92 ~]# kubectl apply -f storage-class.yml
storageclass "fast" created
```

To verify it has been created, the new StorageClass can be listed with:

```
[root@bright92 ~]# kubectl get sc
NAME      PROVISIONER
fast      kubernetes.io/rbd
```

4.22 Integration With Harbor

In order to spawn pods that use images from the Harbor registry, a secret must first be created with the credentials:

```
[root@bright92 ~]# kubectl create secret docker-registry myregistrykey \
 --docker-server=node001:9443 --docker-username=admin --docker-password=Harbor12345
```

The secret must then be referenced from the pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: foo
spec:
  containers:
  - name: foo
    image: node001:9443/library/nginx
  imagePullSecrets:
  - name: myregistrykey
```

Further information on this is available at `https://kubernetes.io/docs/concepts/containers/images/#specifying-imagepullsecrets-on-a-pod`.

5

Kubernetes Apps

Kubernetes add-ons were introduced in NVIDIA Bright Cluster Manager version 8.1, and could be managed in that version as part of the `addons` submode of the `kubernetes` mode in `cmsh`. In NVIDIA Bright Cluster Manager version 8.2 this feature was expanded into the *Kubernetes Applications & Groups* feature. Kubernetes Applications & Groups, less formally called app groups, can be accessed via the `appgroups` submode of `cmsh`:

Example

```
root@bright92 ~# cmsh
[bright92]% kubernetes
[bright92->kubernetes[default]]% appgroups
[bright92->kubernetes[default]->appgroups]% list
Name (key)   Applications
------------ ------------------------------
system       <13 in submode>
[bright92->kubernetes[default]->appgroups]%
```

The version 8.1, `addons` mode parameters are now accessed from version 8.2 onward via a default `system` app group instance. The `system` instance is accessed in the `appsgroup` submode.

Example

```
[bright92->kubernetes[default]->appgroups]% use system
[bright92->kubernetes[default]->appgroups[system]]% show
Parameter                         Value
--------------------------------- ------------------------------------------------
Name                              system
Revision
Enabled                           yes
applications                      <13 in submode>
[bright92->kubernetes[default]->appgroups[system]]% applications
[bright92->kubernetes[default]->appgroups[system]->applications]% list
Name (key)          Format Enabled
------------------- ------ -------
bootstrap           Yaml   yes
calico              Yaml   yes
dashboard           Yaml   yes
dashboard_ingress   Yaml   yes
dns                 Yaml   yes
flannel             Yaml   no
ingress_controller  Yaml   yes
```

```
kubernetes_ingress  Yaml   no
kubestatemetrics    Yaml   yes
metrics_server      Yaml   yes
nvidia              Yaml   no
root                Yaml   yes
```

A Kubernetes application can span multiple namespaces. A name in `appgroups` therefore only exists to group logically-related applications. Each application contains a YAML configuration file, which the cluster manager synchronizes to the Kubernetes API.

The default `system` app group is pre-defined. Other app groups can be created as needed. For example, an app group called `monitoring` could be created to group applications for running Prometheus, node exporters, and anything else related to exposing or viewing Prometheus metrics.

Toggling the `Enable` parameter of an app group enables or disables all of its application components in Kubernetes. Finer-grained control is possible within the `applications` mode level, by toggling the `enabled` parameter per application component instance. For example, within the `calico` application component instance:

Example

```
[bright92->kubernetes[default]->appgroups[system]->applications]% use calico
[bright92->kubernetes[default]->appgroups[system]->applications[calico]]% show
Parameter                        Value
-------------------------------- ------------------------------------------------
Name                             calico
Revision
Format                           Yaml
Enabled                          yes
Config                           <244KiB>
Environment                      <3 in submode>
Exclude list snippets            <2 in submode>
```

A large YAML configuration file for each application component instance can be configured via the `Config` parameter property, using the `set` option of `cmsh`. This opens up a text editor and allows the environment variables in the YAML configuration file to be managed.

Exclude list snippets are short exclude lists that can be set up for Kubernetes apps computing within the `excludelistsnippets` submode. They are used to prevent software image updates from overwriting the provisioned files or directories of the container image that are important to the associated Kubernetes application.

Using exclude list snippets within an `excludelistsnippets` submode is discussed in detail in section 4.4.1 of the *Cloudbursting Manual*. Similar to the case of Kubernetes apps images, in cloud computing exclude list snippets are used to prevent overwriting of the provisioned files and directories of cloud images.

Environment entries can be set via the `Environment` submode. Environment entries are similar to environment variables, and are used to replace variables inside the YAML configuration file. Environment entries can be added to the environment as well, if the `Nodes environment` value inside the `Environment` submode is set to `yes`.

Example

```
[bright92->kubernetes[default]->appgroups[system]->applications[calico]]% environment
[bright92->kubernetes[default]->appgroups[system]->applications[calico]->environment]% list
Name (key)             Value            Nodes environment
---------------------- ---------------- ------------------
calico_typha_replicas  0                no
calico_typha_service   none             no
head_node_internal_ip  10.141.255.254   no
```

5.1 Providing Custom Docker Images

Allowing users to work with custom Docker images on the cluster requires adding the user to the `docker` group. This can be carried out with `usermod -aG docker` <*user*>.

6

Kubernetes Operators

Kubernetes operators are the modern way to manage Kubernetes cluster applications (`https://kubernetes.io/docs/concepts/extend-kubernetes/operator/`). It is usually recommended that Kubernetes operators are used instead of the legacy applications.

At the time of writing of this section (December 2022), Bright provides and packages several operators which are validated to perform basic functionalities on a Kubernetes Bright setup.

The operators available can be categorized into two groups.

- *Cloud* (or *non-airgapped*) operators:
  - the NVIDIA GPU Operator (section 6.3)
  - the Prometheus Stack Operator
  - the Prometheus Adapter Operator
  - the Run:ai Operator (section 6.4)

  These are installed from upstream Helm repositories, and expect internet activity for deployment.

- *Bright* (or *air-gapped*) operators:
  - the Jupyter Kernel Operator (section 6.2)
  - the Spark Operator (section 6.5)
  - the Zalando PostreSQL Operator (`https://github.com/zalando/postgres-operator`)

  These support air-gapped environments by being deployed in two phases:

  1. the .deb or .rpm package being deployed
  2. the actual installation, or roll-out, phase.

6.1 Helm Charts For The Bright Operators

Third party operators should be patched to support working in a pod security policy (PSP) environment. Not only do users need to have access to the custom resource definitions (CRDs) that these operators bring to the cluster, but the service accounts of the operators also require access to the secure namespaces of the users.

During the initial setup, the installation wizard displays a menu to select which operators are to be installed (figure 6.1).

In the case of Bright operators this results in .deb or .rpm packages being deployed. In the case of cloud operators, this already results in a Helm chart being deployed:

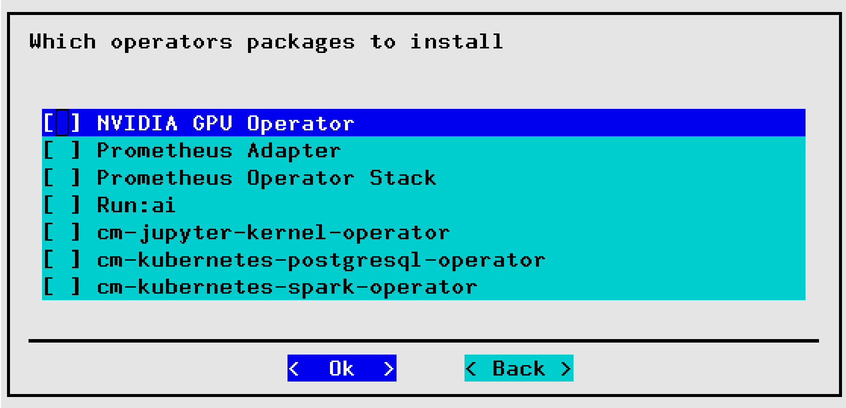

Figure 6.1: Kubernetes TUI Session: Selection of operator packages to be installed

Then, based on the selection, for the Bright operators, the wizard asks which of the installed Bright operators to install (roll out) (figure 6.2):

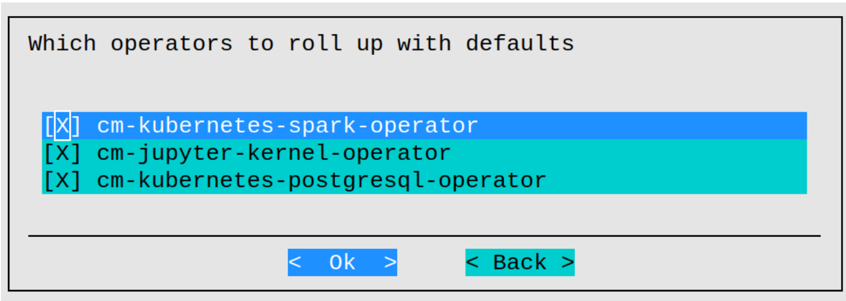

Figure 6.2: Kubernetes TUI Session: Selection of operator Helm charts to be installed

The Helm charts that are selected are installed with sensible defaults. If additional tuning is needed, then the charts can be installed manually after `cm-kubernetes-setup` finishes:

```
[root@bright92 ~]# yum install cm-kubernetes-postgresql-operator -y
[root@bright92 ~]# helm install postgres-operator \
    /cm/shared/apps/kubernetes-postgresql-operator/current/helm/postgres-operator-*.tgz
```

If additional tuning is required then tunable values can be set with a command line similar to the following::

```
[root@bright92 ~]# helm install postgres-operator \
    --values tunables.yaml \
    /cm/shared/apps/kubernetes-postgresql-operator/current/helm/postgres-operator-*.tgz
```

The man page for the operator shows the possible options.

Installed operators can be listed by using the CLI option `--list-operators`:

```
[root@bright92 ~]# cm-kubernetes-setup --list-operators
...
#### stage: kubernetes: Display Available Operators
OPERATOR________________________________: api_available___________________________
cm-jupyter-kernel-operator               : true
cm-kubernetes-postgresql-operator        : true
cm-kubernetes-spark-operator             : false
...
```

6.2 The Jupyter Kernel Operator

6.2.1 Installing The Jupyter Kernel Operator

The Kubernetes Jupyter Kernel Operator can be installed as a part of the `cm-kubernetes-setup` procedure (section 4.2.5), which eventually leads to the selection screen displayed in figure 6.1.

The Kubernetes Jupyter Kernel Operator can alternatively be installed later on using the OS package manager and Helm:

```
[root@bright92 ~]# yum install cm-jupyter-kernel-operator -y
[root@bright92 ~]# helm install cm-jupyter-kernel-operator \
    /cm/shared/apps/jupyter-kernel-operator/current/helm/cm-jupyter-kernel-operator-*.tgz
```

The Jupyter Kernel Operator can be removed with:

Example

```
[root@bright92 ~]# helm uninstall cm-jupyter-kernel-operator
```

It is recommended to enable the PodSecurityPolicy (PSP, section 4.10.2), for the cluster before allowing a user, for example `alice`, to create resources in the Kubernetes cluster.

Example

```
[root@bright92 ~]# cm-kubernetes-setup --psp
[root@bright92 ~]# cm-kubernetes-setup --add-user alice --operators cm-jupyter-kernel-operator
```

The Kubernetes Jupyter Kernel Operator Helm chart creates a CRD that can be used in the Kubernetes API. To check the availability of the CRD, the following command can be run:

Example

```
[root@bright92 ~]# module load kubernetes
[root@bright92 ~]# kubectl get crd | grep jupyterkernels
cmjupyterkernels.apps.brightcomputing.com                                  2022-11-07T09:49:48Z
cmkubernetesoperatorpermissionsjupyterkernels.charts.brightcomputing.com   2022-11-07T09:18:32Z
```

6.2.2 Architecture Overview

The Kubernetes Jupyter Kernel Operator has two main components:

- the operator itself
- the sidecar. This is attached to every user-defined kernel pod, and communicates with Jupyter Enterprise Gateway, acting as a proxy for the kernel process.

The following is an overview of the kernel setup and pod lifecycle when the user runs the Kubernetes Jupyter Kernel Operator:

1. User initiates creating kernel in JupyterLab.
2. JupyterLab delegates this task to Jupyter Enterprise Gateway (JEG).
3. JEG opens a service TCP/IP socket and creates a CRD in Kubernetes specifying this port.
4. KubeApi notifies Jupyter Kernel Operator about the newly created CRD.
5. Jupyter Kernel Operator creates services, configmaps, secrets.
6. Jupyter Kernel Operator creates pod to run Jupyter kernel based on the specification. The sidecar is added to the kernel pod during this step.

7. The sidecar waits for the connection file created by the kernel. Alternatively, it relies on the connection file created by the operator (if requested), as not all kernels create a connection file.

8. The sidecar runs a proxy to forward kernel communications to JEG (stdin, shell, iopub, etc).

9. The sidecar notifies JEG about connection parameters and handles kernel communications.

10. If JEG disappears, or if communication drops, then the sidecar stops. This causes the kernel operator to get a notification via the KubeApi service.

11. The Kubernetes Jupyter Kernel Operator removes the unneeded pod, service, configmap and secrets. It also tries to gather stdout and std error of the kernel pod for debug purposes.

The pod created in step 6 is heavily customized by the kernel operator. For security reasons, running a process inside the pod must be carried out as an unprivileged user.

For the convenience of the Jupyter user, the UID/GID of the process inside the pod should match the UID/GID of the Jupyter user. If that is not the case, then the files created in the container are inaccessible for the Jupyter user.

To achieve matching UID/GIDs, the operator dynamically creates `/etc/passwd` and `/etc/group` files inside the pod and populates them with the data from corresponding templates. At the same time the operator can create a kernel communication file, if requested—some kernels rely on that.

6.2.3 Running Jupyter Kernel Using The Operator

An example of a basic YAML definition for the CMJupyterKernel is:

```
[alice@bright92 ~]$ cat cmjk.yaml
---
apiVersion: apps.brightcomputing.com/v1
kind: CMJupyterKernel
metadata:
  name: cmjk-test
  namespace: alice-restricted
spec:
  username: alice
  uid: 1001
  gid: 1001
  kernel_id: testtesttest
  homedir: /home/alice
  pod:
    volumes:
    - name: homedir
      hostPath:
        path: /home/alice
        type: DirectoryOrCreate
    containers:
    - name: kernel
      image: jupyter/datascience-notebook
      command:
      - "python"
      args:
      - "-m"
      - "ipykernel_launcher"
      - "-f"
      - "/var/tmp/kernel-parm.json"
      workingDir: /home/alice
      securityContext:
        allowPrivilegeEscalation: false
        privileged: false
        runAsNonRoot: true
        runAsUser: 1001
        runAsGroup: 1001
      volumeMounts:
      - name: homedir
        mountPath: /home/alice
```

This can be submitted, but the operator removes it in approximately 1 minute:

```
[alice@bright92 ~]$ module load kubernetes
[alice@bright92 ~]$ kubectl apply -f cmjk.yaml
```

The logs of the operator can be checked for debug purposes:

```
[root@bright92 ~]# module load kubernetes
[root@bright92 ~]# kubectl logs \
    -n cm-jupyter-kernel-operator-system \
    -l control-plane=controller-manager \
    --tail -1 \
    -c manager
```

```
...
2022-02-14T18:26:01.005Z INFO controllers.CMJupyterKernel Container is
stopped. Logs are below "cmjupyterkernel": "alice-restricted/cmjk-test"
...
2022-02-14T18:26:01.190Z INFO controllers.CMJupyterKernel cmjksidecar:
2022/02/14 18:26:00 Timeout receiving pings from server for 60 sec.
Shutting down. "cmjupyterkernel": "alice-restricted/cmjk-test"
...
```

This indicates that the sidecar was stopped because there was no connection from Jupyter Enterprise Gateway to the kernel. This is expected, since the kernel has been run manually, and not using Jupyter. After the sidecar shutdown, the `kube-api` server notifies the operator, which, in turn, removes objects such as CMJupyterKernel, pods, and services.

6.2.4 Jupyter Kernel Operator Tunables

Table 6.1: *Tunable Options for Jupyter Kernel Operators*

Option	Description
`kernel_id`^{R}	Kernel identifier (random UUID) given by Jupyter server
`username`^{O}	Name of the user
`uid`^{R}, `gid`^{R}, `homedir`^{O}, `usershell`^{O}	UID, GID, home directory and default shell of the user
`image_os_flavor`^{O}	Defines template of `/etc/passwd` and `/etc/group` files, where `uid`, `gid`, `homedir`, and `usershell` will be added. Could be one of `ubuntu1604`, `ubuntu1804`, `ubuntu2004`, `ubuntu2404`, `rhel7`, `centos7`, `rhel8`, `centos8`, `sles12`, `sles15`.
`etc_passwd`^{O}, `etc_group`^{O}	Custom content of the `/etc/passwd` or `/etc/group`, if necessary.
`sidecar_command`^{O}, `sidecar_args`^{O}	Commands and arguments to run the sidecar. By default empty. Most of the arguments for the sidecar are passed via environment variables (section 6.2.5).

...continues

...continued

Option	Description
kernel_connection_file_path^O	Where to expect to find kernel connection file. Default: /var/tmp/kernel-parm.json
create_connection_file^O	Does the operator need to create and populate kernel connection file before the pod starts? Default: false
spark_pod_template_path^O, spark_pod_template^O	Options to store or override the Spark executor template
pod^R	Kubernetes Pod definition
service^O	Kubernetes Service definition

Legend:

O: Optional

R: Required

6.2.5 Sidecar Arguments And Environment Variables

Sidecar Arguments

A timeout can be set as an argument for the sidecar.

- `--timeout`: Defines how long, in seconds, that the sidecar waits for the Jupyter Enterprise Gateway proxy to connect before shutdown. Default: `60`

Environment Variables

The following environment variables can be used by the sidecar:

Table 6.2: Environment Variables For The Sidecar

Environment Variable	Description
CMJK_CONNECTION_FILE	Path to find a connection file. The sidecar uses the file to establish a connection to the kernel and to pass data between Jupyter Enterprise Gateway and the kernel. Default: /var/tmp/kernel-parm.json.
CMJK_KERNEL_ID	Unique identifier of the kernel. Usually the UUID in table 6.1.

...continues

...continued

Environment Variable	Description
CMJK_SHELL_PORT	Proxy port to open to forward shell communication. Default: 5001.
CMJK_IOPUB_PORT	Proxy port to open to forward iopub communication. Default: 5002.
CMJK_STDIN_PORT	Proxy port to open to forward stdin communication. Default: 5003.
CMJK_CONTROL_PORT	Proxy port to open to forward control communication. Default: 5004.
CMJK_HB_PORT	Proxy port to open to forward heartbeat communication. Default: 5005.
CMJK_COMM_PORT	Proxy port to open to forward comm communication. Default: 5006.

6.2.6 Running Spark-based Kernels In Jupyter Kernel Operator

Jupyter integration for the cluster manager provides a kernel template (`jupyter-eg-kernel-k8s-cmjkop-py-spark`) and a sample container image (`brightcomputing/jupyter-kernel-sample:k8s-spark-py39-2.0.0`) to run Jupyter kernels in a Spark environment. The image can be altered or created from scratch based on the scripts provided in `/cm/shared/examples/jupyter/kubernetes-kernel-image-spark-py39/`.

The Jupyter kernel is not run directly. Instead, the kernel process is run and controlled by the `spark-submit` executable inside the container.

Jupyter Kernel Operator alters the provided image based on the CRD definition.

Spark-specific tunables are `spark_pod_template_path` and `spark_pod_template`. The operator creates a file inside of the Spark driver Pod and puts the content of `spark_pod_template` in it. After that, `spark-submit` uses this file, via the `--spark.kubernetes.executor.podTemplateFile` configuration option, to create executor pods.

6.2.7 Example: Creating An R kernel

The Jupyter Kernel Operator can be used out-of-the-box to support more kernels.

For example, an R kernel can be added.

The official `jupyter/r-notebook` R image can be taken from the Jupyter project, from `https://jupyter-docker-stacks.readthedocs.io/en/latest/using/selecting.html#jupyter-r-notebook`.

The default entry point cannot be used as that would start Jupyter notebook, while the aim for this section is to use the kernel only.

Some exploratory investigation should reveal the command to start the kernel:

The pod can be run interactively in a Jupyter notebook terminal by a user:

```
kubectl run -i --tty testnotebook --image=jupyter/r-notebook --restart=Never -- bash
```

The kernel specifications can then be investigated:

```
jupyter-kernelspec list
Available kernels:
  ir         /opt/conda/share/jupyter/kernels/ir
  python3    /opt/conda/share/jupyter/kernels/python3
```

The `ir` kernel is what is of interest here. The command line that is used to start the kernel can be found:

```
cat /opt/conda/share/jupyter/kernels/ir/kernel.json
"argv": ["R", "--slave", "-e", "IRkernel::main()", "--args", "connection_file"],
 "display_name":"R",
 "language":"R"
```

Based on this information the Jupyter Kernel Operator CRD can be created for the user:

```
cat cmjk-ir.yaml
---
apiVersion: apps.brightcomputing.com/v1
kind: CMJupyterKernel
metadata:
  name: cmjk-test
  namespace: alice-restricted
spec:
  username: alice
  uid: 1001
  gid: 1001
  kernel_id: testtesttest
  homedir: /home/alice
  create_connection_file: true  # R kernel expects connection file be created
  pod:
    volumes:
    - name: homedir
      hostPath:
        path: /home/alice
        type: DirectoryOrCreate
    containers:
    - name: kernel
      image: jupyter/r-notebook  # image
      command:
      - "R"
      args:
      - "--slave"
      - "-e"
      - "IRkernel::main()"
      - "--args"
      - "/var/tmp/kernel-parm.json"  # we have static connection file
      workingDir: /home/alice
      securityContext:
        allowPrivilegeEscalation: false
        privileged: false
        runAsNonRoot: true
        runAsUser: 1001
        runAsGroup: 1001
      volumeMounts:
      - name: homedir
        mountPath: /home/alice
```

There are several changes from the previous (section 6.2.3) YAML, and from the IR command line:

- `create_connection_file: true`

  If this is not specified then the kernel complains with the following message during startup:

`kernel: cannot open file '/var/tmp/kernel-parm.json': No such file or directory`

This means that the kernel expected this file to be created before the start.

- `image: jupyter/r-notebook`

  Another image needs to be used.

- `args: ... "/var/tmp/kernel-parm.json"`

  The spec file has a fixed path and name, instead of `"connection_file"` as in `kernel.json` earlier

The resulting `cmjk-ir.yaml` file can be submitted to Kubernetes, but it will be removed by the operator after one minute, as it is not being started from the Jupyter Enterprise Gateway.

The next step is to create a kernel template. The Python kernel can be used as a reference:

```
cd /cm/shared/apps/jupyter/current
cd lib/python*/site-packages/cm_jupyter_kernel_creator/kerneltemplates
cp -pr jupyter-eg-kernel-k8s-cmjkop-py jupyter-eg-kernel-k8s-cmjkop-r
```

The files `meta.yaml`, `kernel.json`, and `templates/cmjk.yaml.j2` need to be changed in order to able to provide the correct image and command:

```
vim jupyter-eg-kernel-k8s-cmjkop-r/meta.yaml
vim jupyter-eg-kernel-k8s-cmjkop-r/kernel.json.j2
vim templates/cmjk.yaml.j2
```

The changes that are applied should look similar to the following:

```
diff -u jupyter-eg-kernel-k8s-cmjkop-py/kernel.json.j2 jupyter-eg-kernel-k8s-cmjkop-r/kernel.json.j2
--- jupyter-eg-kernel-k8s-cmjkop-py/kernel.json.j2 2022-01-25 21:13:52.000000000 +0100
+++ jupyter-eg-kernel-k8s-cmjkop-r/kernel.json.j2 2022-02-16 12:22:23.610382929 +0100
@@ -15,8 +15,8 @@
     }
   },
   "argv": [
-    "python",
-    "-m", "ipykernel_launcher",
-    "-f", "/var/tmp/kernel-parm.json"
+    "R",
+    "--slave", "-e", "IRkernel::main()",
+    "--args", "/var/tmp/kernel-parm.json"
   ]
 }

diff -u jupyter-eg-kernel-k8s-cmjkop-py/meta.yaml jupyter-eg-kernel-k8s-cmjkop-r/meta.yaml
--- jupyter-eg-kernel-k8s-cmjkop-py/meta.yaml 2022-01-25 21:13:52.000000000 +0100
+++ jupyter-eg-kernel-k8s-cmjkop-r/meta.yaml 2022-02-16 12:20:57.500974886 +0100
@@ -1,5 +1,5 @@
 ---
-display_name: "Python on Kubernetes Operator"
+display_name: "R on Kubernetes Operator"
 features: "k8s-jupyter-operator-enabled"
 parameters:
   display_name:
```

```
@@ -7,7 +7,7 @@
     definition:
       getter: shell
       exec:
-      - echo "Python on Kubernetes Operator $(date +%y%m%d%H%M%S)"
+      - echo "R on Kubernetes Operator $(date +%y%m%d%H%M%S)"
       display_name: "Display name of the kernel"
   k8s_env_module:
     type: str
@@ -20,9 +20,9 @@
     definition:
       getter: static
       default:
-      - "jupyter/datascience-notebook"
+      - "jupyter/r-notebook"
       values:
-      - "jupyter/datascience-notebook"
+      - "jupyter/r-notebook"
       display_name: "Image to run"
     limits:
       max_len: 1

diff -u jupyter-eg-kernel-k8s-cmjkop-py/templates/cmjk.yaml.j2\
jupyter-eg-kernel-k8s-cmjkop-r/templates/cmjk.yaml.j2
--- jupyter-eg-kernel-k8s-cmjkop-py/templates/cmjk.yaml.j2 2022-01-25 21:13:52.000000000 +0100
+++ jupyter-eg-kernel-k8s-cmjkop-r/templates/cmjk.yaml.j2 2022-02-16 12:24:25.375373991 +0100
@@ -9,6 +9,7 @@
   gid:  gid
   kernel_id:  kernel_id
   homedir:  homedir
+  create_connection_file: true
   pod:
     volumes:
     - name: homedir
```

After instantiating a kernel spec from the template, the R kernel is ready to use:

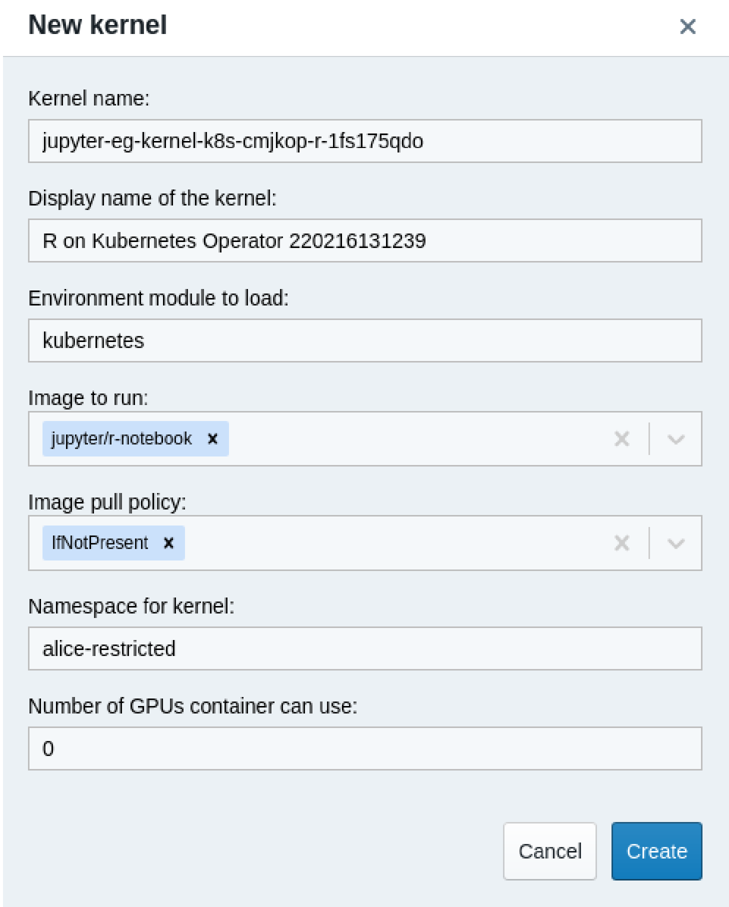

Figure 6.3: Jupyter Kernel Creator. Creating the IR kernel spec

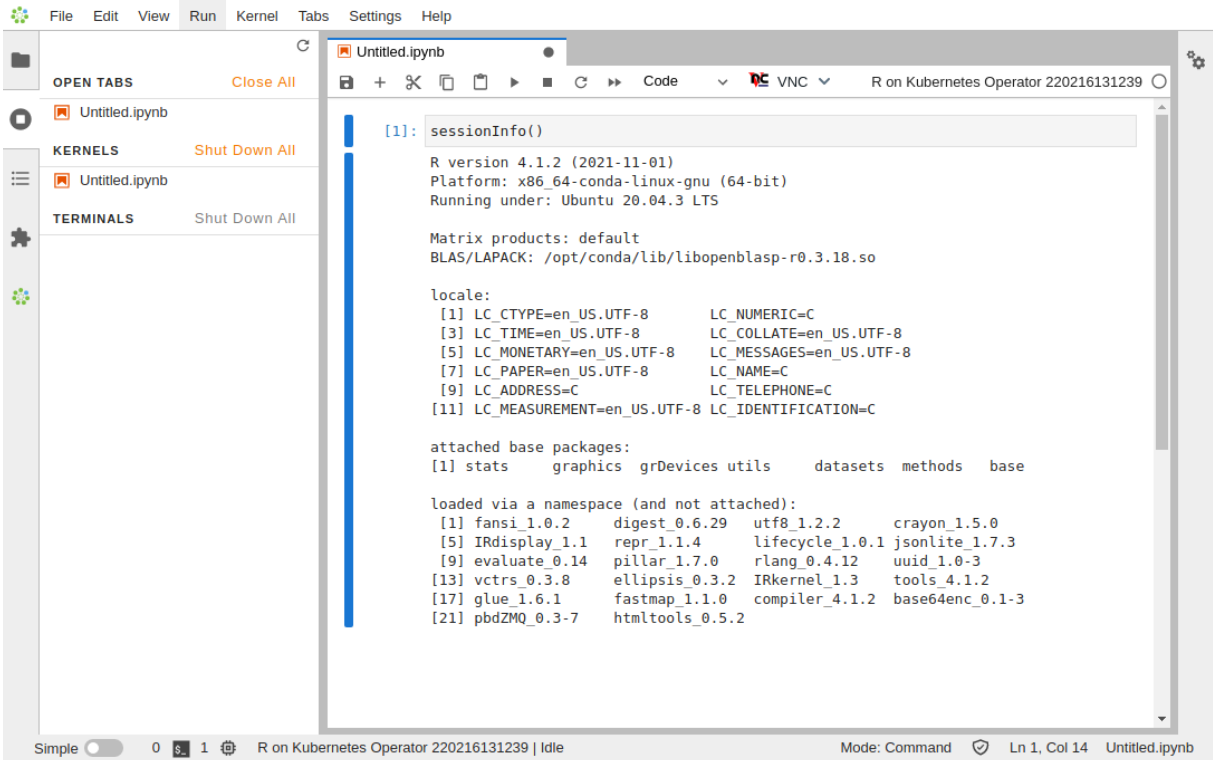

Figure 6.4: JupyterLab. Running the IR kernel

6.3 The NVIDIA GPU Operator

6.3.1 Installing The NVIDIA GPU Operator

The NVIDIA GPU operator can be installed as a part of the installation session by the `cm-kubernetes-setup` wizard (section 4.2.5). During the setup session, a checkbox can be checkmarked to install and enable the GPU operator (figure 6.1). Nodes that run DGX OS are also supported by the wizard.

The NVIDIA GPU operator can also be deployed on an existing Bright Kubernetes cluster, as described next.

6.3.2 Installing The NVIDIA GPU Operator On An Existing Kubernetes Cluster

If the NVIDIA GPU Operator (`https://docs.nvidia.com/datacenter/cloud-native/gpu-operator`) is to be installed within an existing Bright Kubernetes cluster, then it must always be deployed using Helm.

Prerequisites: If the existing cluster uses the NVIDIA device plugin add-on, even if configured by NVIDIA Bright Cluster Manager, then it may be necessary to disable the add-on. This add-on is now deprecated, and will be removed in a future release.

```
[cluster->kubernetes[default]->appgroups[system]->applications[nvidia]]% set enabled no
[cluster->kubernetes*[default*]->appgroups*[system*]->applications*[nvidia*]]% commit
```

One of the prerequisites for the preceding add-on is that it uses labels to identify the nodes to be managed by the add-on. These labels are unnecessary for the GPU operator, and may be removed:

```
[cluster->kubernetes[default]->labelsets]% remove nvidia
[cluster->kubernetes*[default*]->labelsets*]% commit
```

Installing The NVIDIA GPU Operator A knowledge base article that describes how to prepare software images, and how to deploy the NVIDIA GPU Operator using Helm, can be found at:

`https://kb.brightcomputing.com/knowledge-base/the-nvidia-gpu-operator-with-kubernetes-on-a-bright-cluster/`

The article also covers how to deploy the Prometheus Operator Stack, and the Prometheus Adapter for monitoring GPU usage. Deploying these is optional.

Validation methods are described for each step of the deployment.

- For `containerd`, Helm installation is carried out by the root user with the following options:

```
helm install --wait -n gpu-operator --create-namespace \
 --version v1.10.1 \
 --set driver.enabled=false \
 --set operator.defaultRuntime=containerd \
 --set toolkit.enabled=true \
 --set toolkit.env[0].name=CONTAINERD_CONFIG \
 --set toolkit.env[0].value=/cm/local/apps/containerd/var/etc/conf.d/nvidia-cri.toml \
 gpu-operator nvidia/gpu-operator
```

- For `docker`, Helm installation is carried out by the root user with the following options:

```
helm install --wait -n gpu-operator --create-namespace \
 --version v1.10.1 \
 --set driver.enabled=false \
 --set operator.defaultRuntime=docker \
 --set toolkit.enabled=true \
 gpu-operator nvidia/gpu-operator
```

NVIDIA GPU Operator containerd configuration: The operator provides the toolkit binaries and containerd configuration (`nvidia-cri.toml`) on each host where a GPU is auto-detected via a host-mount.

The flag that enables this is `--set toolkit.enabled=true`. The path for the configuration file should be set to: `/cm/local/apps/containerd/var/etc/conf.d/nvidia-cri.toml`, which is where Bright's `cm-containerd` package expects to find it.

The operator provides a similar configuration functionality for the CUDA drivers. However this is not used in the cluster manager, and it is disabled with the `-set driver.enabled=false` flag. This is because Bright supports CUDA on more Linux distributions and kernel versions than the NVIDIA GPU Operator does. CUDA drivers are therefore expected to already be present on the relevant nodes that have GPUs.

NVIDIA GPU Operator Docker configuration: This is only relevant for older Kubernetes deployments that are deployed on top of Docker or Bright Docker.

Default paths are used, so nothing particularly special has to be done for the operator to deploy properly.

6.3.3 Removing The NVIDIA GPU Operator

The NVIDIA GPU Operator can be found in the `gpu-operator` namespace inside Helm and Kubernetes.

```
root@bright92 ~# helm list -n gpu-operator
NAME            NAMESPACE     REVISION ... STATUS     CHART                   APP VERSION
gpu-operator    gpu-operator  1        ... deployed   gpu-operator-v1.10.1    v1.10.1
```

A `helm uninstall gpu-operator` command can be used to uninstall the operator.

6.3.4 Validating The NVIDIA GPU Operator

A pragmatic way to validate the NVIDIA GPU Operator is to check if the validator pods can be run. A `Running` status for the pods that are to have a GPU on them can be seen with:

Example

```
root@bright92 ~# kubectl get pod -n gpu-operator -l app=nvidia-operator-validator -o wide
NAME                              READY  STATUS   ... IP              NODE     NOMINATED NODE  READINESS GATES
nvidia-operator-validator-2qvz6   1/1    Running  ... 172.29.152.172  node001  <none>          <none>
nvidia-operator-validator-xkwwv   1/1    Running  ... 172.29.112.154  node002  <none>          <none>
```

The preceding shows succesfully running pods. The log output should show `all validations are successful`:

Example

```
root@bright92 ~# kubectl logs -n gpu-operator -l app=nvidia-operator-validator -c nvidia-operator-validator
all validations are successful
all validations are successful
```

6.3.5 Validating The NVIDIA GPU Operator In Detail

The set of pods associated with the NVIDIA GPU Operator Pods can be examined in more detail. The following shows outputs from a GPU operator deployment that is working correctly:

Example

```
root@bright92 ~# helm list -n gpu-operator
NAME          NAMESPACE     ... STATUS     CHART                  APP VERSION
gpu-operator  gpu-operator  ... deployed  gpu-operator-v1.10.1  v1.10.1

root@bright92 ~# kubectl get all -n gpu-operator -o wide
NAME                                                             READY  STATUS     RESTARTS     ...  NODE
pod/gpu-feature-discovery-gk892                                  1/1    Running    0            ...  node001
pod/gpu-feature-discovery-rmkvj                                  1/1    Running    0            ...  node002
pod/gpu-operator-798c6ddc97-lmclm                                1/1    Running    0            ...  bright92
pod/gpu-operator-node-feature-discovery-master-6c65c99969-cjlpq  1/1    Running    0            ...  bright92
pod/gpu-operator-node-feature-discovery-worker-cgxzl             1/1    Running    0            ...  node002
pod/gpu-operator-node-feature-discovery-worker-ds5mb             1/1    Running    0            ...  bright92
pod/gpu-operator-node-feature-discovery-worker-jf65c             1/1    Running    0            ...  node001
pod/nvidia-container-toolkit-daemonset-ffbk7                     1/1    Running    1 (46m ago)...  node002
pod/nvidia-container-toolkit-daemonset-lqfkq                     1/1    Running    0            ...  node001
pod/nvidia-cuda-validator-pxs9b                                  0/1    Completed  0            ...  node001
pod/nvidia-cuda-validator-v7gfz                                  0/1    Completed  0            ...  node002
pod/nvidia-dcgm-exporter-bxjrv                                   1/1    Running    0            ...  node001
pod/nvidia-dcgm-exporter-ql19z                                   1/1    Running    0            ...  node002
pod/nvidia-device-plugin-daemonset-698hd                         1/1    Running    0            ...  node001
pod/nvidia-device-plugin-daemonset-xd4kj                         1/1    Running    0            ...  node002
pod/nvidia-device-plugin-validator-5crlc                         0/1    Completed  0            ...  node001
pod/nvidia-device-plugin-validator-wh27x                         0/1    Completed  0            ...  node002
pod/nvidia-operator-validator-2qvz6                              1/1    Running    0            ...  node001
pod/nvidia-operator-validator-xkwwv                              1/1    Running    0            ...  node002
...
```

On this particular example cluster, there are two compute nodes with GPUs, and there is one control-plane node without a GPU:

```
root@bright92 ~# kubectl get nodes
NAME        STATUS   ROLES                  AGE    VERSION
node001     Ready    worker                 3h2m   v1.24.0
node002     Ready    worker                 3h2m   v1.24.0
bright92    Ready    control-plane,master   3h2m   v1.24.0
```

Feature discovery pods: *Node Feature Discovery* (NFD, `https://intel.github.io/kubernetes-docs/nfd/index.html`) is an add-on that is initiated after the operator is installed. A master pod collects discovery information from the worker pods, and schedules more pods in case GPUs have been detected.

In the preceding GPU operator output,

- the master pod is running on `node001` with the name:
  `gpu-operator-node-feature-discovery-master-6c65c99969-wtzcx`
- the worker pods run on each node. For example, the worker pod for `node002` is:
  `gpu-operator-node-feature-discovery-worker-z4skv`

The output for the pods is not very verbose by default, but if more pods under the `nvidia-` namespace are scheduled on a node, besides the `gpu-operator-node-feature-discovery-*` pods, then that means that NFD has detected one or more GPUs.

For example, a GPU discovered on node001 results in a scheduling of the following pods on that node:

- container toolkit
- device plugin
- validator

Container toolkit pods: For nodes that have GPUs, the NVIDIA container toolkit installation pods are started. Pod logs show exactly what is being installed.

One of the requirements for the NVIDIA container toolkit installation pods is that the driver has to be in working order, or the `init` container `driver-validation` will fail. The following is the log from a successful installation:

Example

```
root@bright92 ~# kubectl logs -f -n gpu-operator nvidia-container-toolkit-daemonset-ffbk7
Defaulted container "nvidia-container-toolkit-ctr" out of: nvidia-container-toolkit-ctr, driver-validation (init)
...
time="2022-12-06T14:31:36Z" level=info msg="Installing toolkit"
time="2022-12-06T14:31:36Z" level=info msg="Parsing arguments: [/usr/local/nvidia/toolkit]"
time="2022-12-06T14:31:36Z" level=info msg="Successfully parsed arguments"
time="2022-12-06T14:31:36Z" level=info msg="Installing NVIDIA container toolkit to '/usr/local/nvidia/toolkit'"
...
time="2022-12-06T14:31:36Z" level=info msg="Installing NVIDIA container toolkit config
                               '/usr/local/nvidia/toolkit/.config/nvidia-container-runtime/config.toml'"
time="2022-12-06T14:31:36Z" level=info msg="Setting up runtime"
time="2022-12-06T14:31:36Z" level=info msg="Starting 'setup' for containerd"
time="2022-12-06T14:31:36Z" level=info msg="Parsing arguments: [/usr/local/nvidia/toolkit]"
time="2022-12-06T14:31:36Z" level=info msg="Successfully parsed arguments"
time="2022-12-06T14:31:36Z" level=info msg="Loading config: /runtime/config-dir/nvidia-cri.toml"
...
```

Device plugin pods: The device plugin pods are started up next. These have the toolkit as a requirement. If the toolkit is not in working order, then the init container toolkit-validation fails. The following is the log from a successful startup:

Example

```
root@bright92 ~# kubectl logs -f -n gpu-operator          nvidia-device-plugin-daemonset-698hd
Defaulted container "nvidia-device-plugin-ctr" out of: nvidia-device-plugin-ctr, toolkit-validation (init)
2022/12/06 14:32:20 Loading NVML
2022/12/06 14:32:20 Starting FS watcher.
2022/12/06 14:32:20 Starting OS watcher.
2022/12/06 14:32:20 Retreiving plugins.
2022/12/06 14:32:20 No MIG devices found. Falling back to mig.strategy=&
2022/12/06 14:32:20 Starting GRPC server for 'nvidia.com/gpu'
2022/12/06 14:32:20 Starting to serve 'nvidia.com/gpu' on /var/lib/kubelet/device-plugins/nvidia-gpu.sock
2022/12/06 14:32:20 Registered device plugin for 'nvidia.com/gpu' with Kubelet
```

The pod log output suggests that the GPU is now registered with the Kubelet as a resource. This can be checked by querying the Node resource:

Example

```
root@bright92 ~# kubectl describe node node001 | grep nvidia
nvidia.com/cuda.driver.major=520
nvidia.com/cuda.driver.minor=61
nvidia.com/cuda.driver.rev=05
nvidia.com/cuda.runtime.major=11
nvidia.com/cuda.runtime.minor=8
nvidia.com/gfd.timestamp=1670337142
nvidia.com/gpu.compute.major=7
nvidia.com/gpu.compute.minor=0
nvidia.com/gpu.count=1
nvidia.com/gpu.deploy.container-toolkit=true
nvidia.com/gpu.deploy.dcgm=true
nvidia.com/gpu.deploy.dcgm-exporter=true
nvidia.com/gpu.deploy.device-plugin=true
nvidia.com/gpu.deploy.driver=true
nvidia.com/gpu.deploy.gpu-feature-discovery=true
nvidia.com/gpu.deploy.node-status-exporter=true
nvidia.com/gpu.deploy.operator-validator=true
nvidia.com/gpu.family=volta
nvidia.com/gpu.machine=OpenStack-Nova
nvidia.com/gpu.memory=32768
nvidia.com/gpu.present=true
nvidia.com/gpu.product=Tesla-V100-SXM3-32GB
nvidia.com/mig.strategy=single
nvidia.com/run.ai-swap.enabled=false
nvidia.com/gpu:     1
nvidia.com/gpu:     1
...
```

Validator pods: If anything goes wrong with either the driver, toolkit, CUDA, or the plugin, then validator pods are a good place to start looking.

If all goes well, the main container outputs all validations are successful:

Example

```
root@bright92 ~# kubectl logs -f -n gpu-operator nvidia-operator-validator-2qvz6
Defaulted container "nvidia-operator-validator" out of: nvidia-operator-validator, driver-validation (init),
toolkit-validation (init), cuda-validation (init), plugin-validation (init)
all validations are successful
```

It is possible for an init container to fail. The output for the container should then be checked.

The following shows output from successful init containers:

```
root@bright92 ~# kubectl logs -f -n gpu-operator nvidia-operator-validator-2qvz6 -c driver-validation
running command chroot with args [/run/nvidia/driver nvidia-smi]
Tue Dec  6 15:32:14 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 520.61.05    Driver Version: 520.61.05    CUDA Version: 11.8     |
|-------------------------------+----------------------+----------------------+
| 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 V100-SXM3...  On   | 00000000:00:06.0 Off |                    0 |
| N/A   32C    P0    46W / 350W |      2MiB / 32768MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

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

root@bright92 ~# kubectl logs -f -n gpu-operator nvidia-operator-validator-2qvz6 -c toolkit-validation
Tue Dec  6 14:32:16 2022
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 520.61.05    Driver Version: 520.61.05    CUDA Version: 11.8     |
|-------------------------------+----------------------+----------------------+
| 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 V100-SXM3...  On   | 00000000:00:06.0 Off |                    0 |
| N/A   32C    P0    46W / 350W |      2MiB / 32768MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

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

root@bright92 ~# kubectl logs -f -n gpu-operator nvidia-operator-validator-2qvz6 -c cuda-validation
time="2022-12-06T14:32:17Z" level=info msg="pod nvidia-cuda-validator-pxs9b is curently in Pending phase"
time="2022-12-06T14:32:22Z" level=info msg="pod nvidia-cuda-validator-pxs9b is curently in Pending phase"
```

```
time="2022-12-06T14:32:27Z" level=info msg="pod nvidia-cuda-validator-pxs9b is curently in Pending phase"
time="2022-12-06T14:32:32Z" level=info msg="pod nvidia-cuda-validator-pxs9b have run successfully"

root@bright92 ~# kubectl logs -f -n gpu-operator nvidia-operator-validator-2qvz6 -c plugin-validation
time="2022-12-06T14:32:33Z" level=info msg="pod nvidia-device-plugin-validator-5crlc is curently in Pending phase"
time="2022-12-06T14:32:38Z" level=info msg="pod nvidia-device-plugin-validator-5crlc is curently in Pending phase"
time="2022-12-06T14:32:43Z" level=info msg="pod nvidia-device-plugin-validator-5crlc have run successfully"
```

This also explains where the pods earlier on came from, the ones marked with status `Completed`. They are used as part of certain validation steps.

Which `init` container prints out error messages should indicate where the problem lies—either with the CUDA drivers, or the toolkit, and so on. If the driver or toolkit is not validating correctly, then it may result in a lot of pods stuck in a `Pending` or an `Init` stage. Looking at what `init` container is associated with the stuck pod helps in diagnosing the problem.

DCGM exporter pods: These pods expose metrics endpoints for scraping, and can be considered less critical. They are involved in GPU metrics collection, and can be utilized with, for example, Prometheus Stack Operator, or the Prometheus Adapter, for *horizontal pod autoscaling* based on GPU metrics.

More information on the Prometheus Stack Operator and the Prometheus Adapter Operator can be found at:

`https://kb.brightcomputing.com/knowledge-base/the-nvidia-gpu-operator-with-kubernetes-on-a-bright-cluster`

6.3.6 Running A GPU Workload

A GPU workload can be run with the following configuration:

Example

```
root@bright92 ~# cat << EOF > gpu.yaml
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  restartPolicy: Never
  containers:
  - name: cuda-container
    image: nvidia/cuda:9.2-runtime
    command: ["nvidia-smi"]
    resources:
      limits:
        nvidia.com/gpu: 1
EOF
root@bright92 ~# kubectl create -f gpu.yaml
pod/gpu-pod created
```

On a cluster with GPUs available, this pod should get scheduled, and should not stay stuck in the `Pending` phase.

The preceding example just invokes `nvidia-smi` in the container. The output can be viewed to confirm that it worked:

Example

```
root@bright92 ~# kubectl logs -f gpu-pod
Tue Dec  6 15:08:03 2022
```

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 520.61.05    Driver Version: 520.61.05    CUDA Version: 11.8     |
|-------------------------------+----------------------+----------------------+
| 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 V100-SXM3...  On   | 00000000:00:06.0 Off |                    0 |
| N/A   34C    P0    47W / 350W |      2MiB / 32768MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+

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

6.4 The Run:ai Operator

The Run:ai operator is a GUI application that can make a cluster installer for Run:ai available via the Bright head node landing page (figure 6.5).

The Run:ai documentation documents the cluster installer bundle for NVIDIA DGX at: `https://docs.run.ai/admin/runai-setup/cluster-setup/dgx-bundle/`.

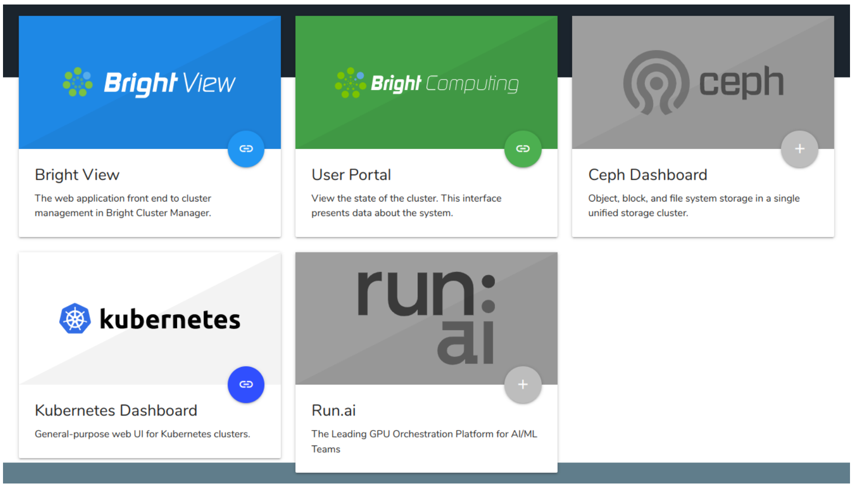

Figure 6.5: Run:ai installer available from the Bright landing page

6.4.1 Installing The Run:ai Operator

The Run:ai operator can be installed as a part of the `cm-kubernetes-setup` procedure (section 4.2.5).

The Helm status can be checked with, for example:

Example

```
root@bright92 ~# helm list -n runai
NAME                 NAMESPACE  ... STATUS    CHART                     APP VERSION
cluster-installer    runai      ... deployed  cluster-installer-2.8.8   0.0.1

root@bright92 ~# kubectl get all -n runai
```

```
NAME                                                 READY   STATUS    RESTARTS   AGE
pod/cluster-installer-deployment-5f4c4cbf4c-82gmx    1/1     Running   0          5m9s

NAME                                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/cluster-installer-service    ClusterIP   10.150.117.247   <none>        8080/TCP   5m9s

NAME                                            READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/cluster-installer-deployment   1/1     1            1           5m9s

NAME                                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/cluster-installer-deployment-5f4c4cbf4c   1         1         1       5m9s
```

6.4.2 Removing The Run:ai Operator

The Run:ai operator can be removed via Helm:

Example

```
[root@bright92 ~]# helm uninstall cluster-installer -n runai
```

Removal of Run:ai access from the Bright head node landing page (figure 6.5) can be carried with a removal of the associated JSON file:

Example

```
root@bright92 ~# ls -al /var/www/html/kubernetes/runai/
total 4
drwxr-xr-x 2 root root  26 Dec  6 12:21 .
drwxr-xr-x 4 root root  51 Dec  1 16:24 ..
-rw-r--r-- 1 root root 317 Dec  6 12:21 default.json
root@bright92 ~# rm -rf /var/www/html/kubernetes/runai/default.json
root@bright92 ~#
```

Each Kubernetes cluster has its own JSON file.

Uninstalling the Kubernetes cluster automatically cleans up everything associated with it.

6.4.3 Completing The Run:ai Installation

Right after `cm-kubernetes-setup` is run, a summary is shown at the end of the installation. This includes URLs that point to Run:ai installer:

Example

```
[root@bright92 ~]# cm-kubernetes-setup
...
#### stage: kubernetes: Print Summary
Installation completed. Pods might still be initializing.

To add users to the cluster use: refer to `cm-kubernetes-setup --help'

To use kubectl load the module file: kubernetes/default/1.24
Common URLs:
- Kubernetes API server: https://rb-runai.openstacklocal:10443
- Kubernetes dashboard: https://dashboard.rb-runai.openstacklocal:30443/
- Kubernetes dashboard: https://0.0.0.0:30443/dashboard/
- Run:ai installer: http://rb-runai.openstacklocal:30080/runai-installer
- Run:ai installer: https://rb-runai.openstacklocal/#runai
## Progress: 100
```

```
Took:      13:18 min.
Progress:  100/100
################### Finished execution for 'Kubernetes Setup', status: completed

Kubernetes Setup finished!
```

The Run:ai installer presents a wizard first, which checks for dependencies (figure 6.6).

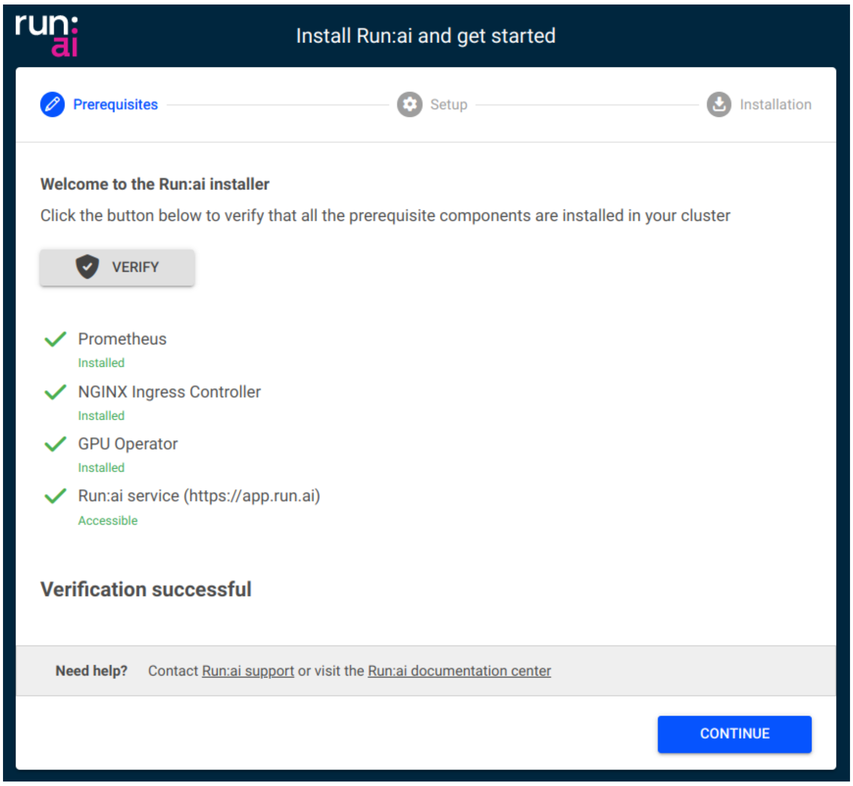

Figure 6.6: Run:ai installer: verification screen

On the next screen, the user is asked for credentials, URLs, and certificates:

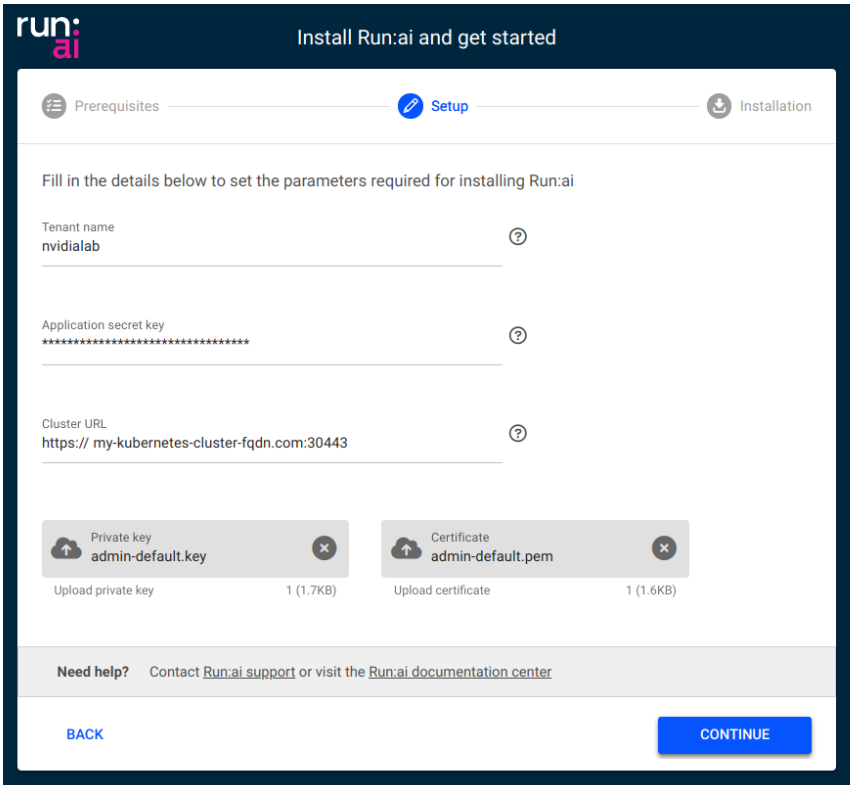

Figure 6.7: Run:ai installer: fields to be filled by the user

- The `Tenant name`, and `Application secret key` are provided by Run:ai.
- The `Cluster URL` has to be the FQDN of the Kubernetes cluster. For example:
  `my-kubernetes-cluster-fqdn.com:30443`
  This is the FQDN that the wizard prompts for in figure 4.3.
- The `Private key` and `Certificate` files are the `admin` key and PEM files from the directory of the user on the cluster. By default:
  - for `root` these are
    * `/root/.kube/admin-default.key`
    * `/root/.kube/admin-default.pem`
  - for a regular user, such as `john`, these are
    * `/home/john/.kube/admin-default.key`
    * `/home/john/.kube/admin-default.pem`

The wizard finalizes the Run:ai setup, and then carries out the installation. A progress meter is displayed during installation:

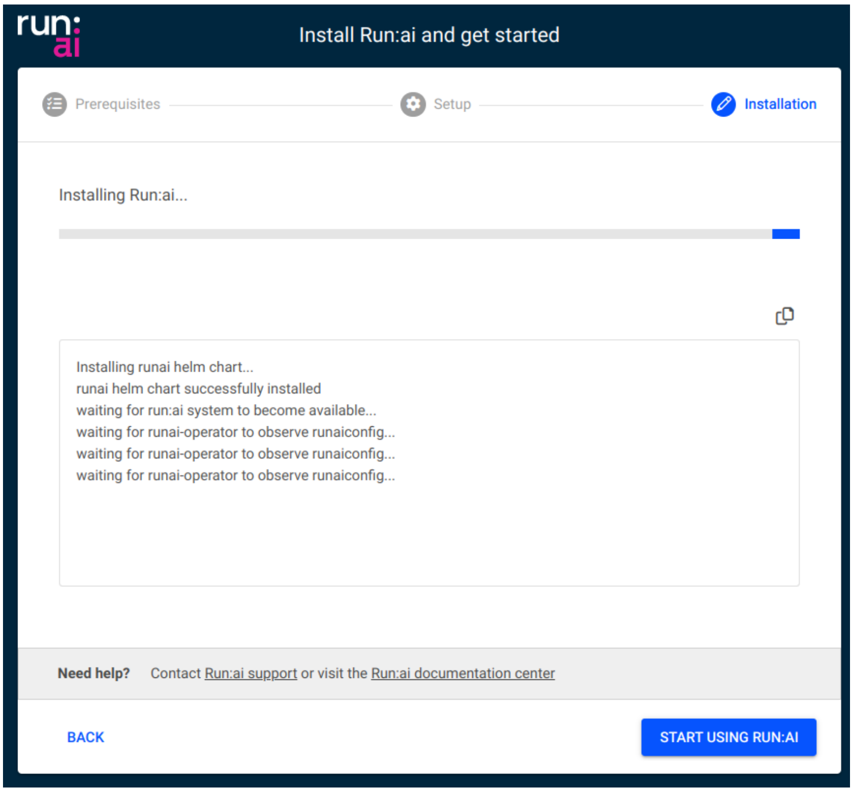

Figure 6.8: Run:ai Installer: installation progress

On completion, the wizard redirects the user to the Run:ai dashboard. The NVIDIA Bright Cluster Manager head node landing page is also updated to point to the Run:ai dashboard, with the "+" sign of figure 6.5 now showing a link icon instead, indicating that Run:ai is now installed:

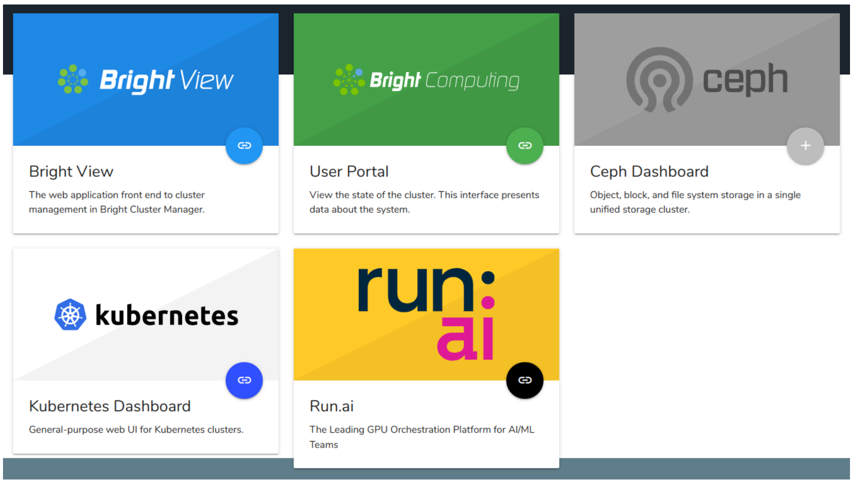

Figure 6.9: Run:ai is now accessible from the Bright head node landing page

6.4.4 Post-installation

The Run:ai documentation includes post-installation steps in the following URL: `https://docs.run.ai/admin/runai-setup/cluster-setup/dgx-bundle/#post-installation`.

The most important aspect is configuring Researcher Access Control in Bright: `https://docs.run.ai/admin/runai-setup/authentication/researcher-authentication/`.

This includes going to `cmsh` and configuring the Kubernetes API server with additional OIDC parameters.

The `runai` binary can be downloaded in various ways, the Run:ai environment has an option where the binary can be downloaded from the cluster itself. The binary can be copied to `/usr/bin` and made executable by the system administrator.

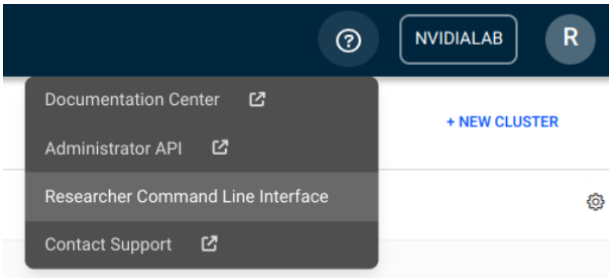

Figure 6.10: `runai` binary download

6.5 Kubernetes Spark Operator

Using the Kubernetes Spark Operator is a simpler alternative to using the `spark-submit` tool for job submission.

6.5.1 Installing The Kubernetes Spark Operator

The Kubernetes Spark Operator can be installed as a part of the `cm-kubernetes-setup` procedure (section 4.2.5), which eventually leads to a display listing the operator packages that may be installed (figure 6.1).

The Kubernetes Spark Operator can alternatively be installed later on using the OS package manager and Helm:

```
[root@bright92 ~]# yum install cm-kubernetes-spark-operator -y
[root@bright92 ~]# helm install cm-kubernetes-spark-operator \
  /cm/shared/apps/kubernetes-spark-operator/current/helm/spark-operator-*.tgz
```

The Kubernetes Spark Operator can be removed with:

Example

```
[root@bright92 ~]# helm uninstall cm-kubernetes-spark-operator
```

The operator installation state can be verified with `--list-operators`:

Example

```
[root@bright92 ~]# cm-kubernetes-setup --list-operators
...
OPERATOR________________________________: api_available___________________________
cm-jupyter-kernel-operator              : 0
cm-kubernetes-postgresql-operator       : 0
cm-kubernetes-spark-operator            : 1
...
```

The Helm status can be checked with, for example:

Example

```
[root@bright92 ~]#  helm list
NAME                            NAMESPACE  ...  STATUS     CHART                  APP VERSION
cm-kubernetes-spark-operator    default    ...  deployed   spark-operator-1.0.8  v1beta2-1.2.0-3.0.0
[root@bright92 ~]#  helm status cm-kubernetes-spark-operator
NAME: cm-kubernetes-spark-operator
LAST DEPLOYED: Mon Sep 19 11:17:21 2022
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
[root@bright92 ~]#
```

The Permission Manager (section 4.16) and PodSecurityPolicy (PSP, section 4.10.2) must both be enabled for the cluster, before allowing a user to create resources in the Kubernetes cluster in their namespace:

Example

```
[root@bright92 ~]# cm-kubernetes-setup --psp
```

The user `alice` can be allowed to use the Spark operator, and allowed to run a process as any UID in the pod:

Example

```
[root@bright92 ~]# cm-kubernetes-setup --add-user alice --operators cm-kubernetes-spark-operator \
--allow-all-uids
```

The Kubernetes Spark operator Helm chart creates a CRD that can be used in the Kubernetes API.

For Alice, the CRD is available and can be used with a Spark operator YAML, to build a Spark application carry out a pi run in the restricted namespace.

6.5.2 Example Spark Operator Run: Calculating Pi

Continuing on with the user `alice` of the preceding section, a YAML file based on the specification at `https://github.com/GoogleCloudPlatform/spark-on-k8s-operator/blob/master/examples/spark-py-pi.yaml` can be used:

Example

```
[root@bright92 ~]# su - alice
[alice@bright92 ~]$ module load kubernetes
[alice@bright92 ~]$ cat <<EOF > pi-spark.yaml
apiVersion: "sparkoperator.k8s.io/v1beta2"
kind: SparkApplication
metadata:
  name: pyspark-pi
spec:
  type: Python
  pythonVersion: "3"
  mode: cluster
  image: "gcr.io/spark-operator/spark-py:v3.1.1"
  imagePullPolicy: Always
  mainApplicationFile: local:///opt/spark/examples/src/main/python/pi.py
  sparkVersion: "3.1.1"
  restartPolicy:
    type: OnFailure
    onFailureRetries: 3
```

```
    onFailureRetryInterval: 10
    onSubmissionFailureRetries: 5
    onSubmissionFailureRetryInterval: 20
  driver:
    cores: 1
    coreLimit: "1200m"
    memory: "512m"
    labels:
      version: 3.1.1
    serviceAccount: spark
  executor:
    cores: 1
    instances: 1
    memory: "512m"
    labels:
      version: 3.1.1
EOF
[alice@bright92 ~]$ kubectl apply -f pi-spark.yaml
sparkapplication.sparkoperator.k8s.io/pyspark-pi created
[alice@bright92 ~]$ kubectl get pods
NAME                READY   STATUS              RESTARTS   AGE
pyspark-pi-driver   0/1     ContainerCreating   0          1s
[alice@bright92 ~]$ kubectl get pods
NAME                READY   STATUS    RESTARTS   AGE
pyspark-pi-driver   1/1     Running   0          3s
[alice@bright92 ~]$ kubectl get sparkapplications
NAME         AGE
pyspark-pi   7s
[alice@bright92 ~]$ kubectl get pods
NAME                                READY   STATUS              RESTARTS   AGE
pyspark-pi-driver                   1/1     Running             0          14s
pythonpi-e768128383a881b3-exec-1    0/1     ContainerCreating   0          0s
[alice@bright92 ~]$ kubectl get pods
NAME                                READY   STATUS        RESTARTS   AGE
pyspark-pi-driver                   0/1     Completed     0          34s
pythonpi-e768128383a881b3-exec-1    0/1     Terminating   0          20s
[alice@bright92 ~]$ kubectl get pods
NAME                READY   STATUS      RESTARTS   AGE
pyspark-pi-driver   0/1     Completed   0          36s
```

Instead of tracking the pod with:
`kubectl get pods`
as in the preceding session, or with the more convenient:
`watch kubectl get pods`
the pod could be tracked with the `-f| --follow` option to stream the driver logs:

Example

```
[alice@bright92 ~]$ kubectl logs pyspark-pi-driver -f
```

To get intended output of the pi run—the calculated value of pi—it is sufficient to grep the log as follows:

Example

```
[alice@bright92 ~]$ kubectl logs pyspark-pi-driver | grep ^Pi
Pi is roughly 3.148800
```

After the pi run has completed, the resources can be removed from the namespace:

```
[alice@bright92 ~]$ kubectl delete -f pi-spark.yaml
sparkapplication.sparkoperator.k8s.io "pyspark-pi" deleted

[alice@bright92 ~]$ kubectl get pods
No resources found in alice-restricted namespace.
[alice@bright92 ~]$ kubectl get sparkapplications
No resources found in alice-restricted namespace.
```

7

Kubernetes On Edge

How edge sites can be configured is described in Chapter 2 of the *Edge Manual*.

If there are Bright Edge sites configured in the cluster, then the Kubernetes setup prompts the user with edge sites that Kubernetes can be deployed on.

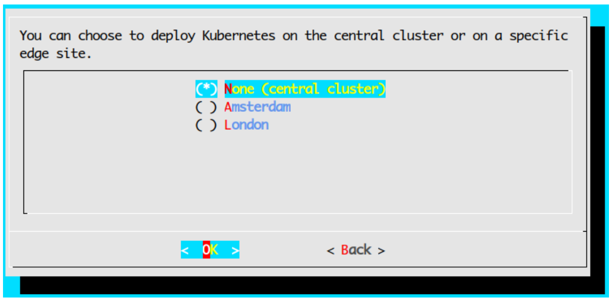

Figure 7.1: `cm-kubernetes-setup` prompting for edge sites.

If an edge site is selected, then the rest of the wizard prompts only for nodes available within that edge site; prompts only for the associated network interfaces; and so on.

7.1 Flags For Edge Installation

Edge directors often lack high-bandwidth connectivity to the central head node, or they often may benefit from coming up as quickly as possible. It can therefore sometimes be useful to skip stages of the setup.

Running `cm-kubernetes-setup --help` displays some additional flags that allow some setup stages, that bring up a cloud director, to be skipped explicitly:

```
cm-kubernetes-setup --help
...
installing Kubernetes clusters:
  Flags for installing or managing Kubernetes clusters

  --skip-package-install
                        Skip the package installation steps. Ignores skip_packages
                        flags in the config.
  --skip-reboot         Skip the reboot steps.
  --skip-image-update   Skip the image update steps.
  --skip-disksetup-changes
                        Never change the disk-setup. Use this flag if you manually
                        configure a partition or device for docker thin pool devices
                        for example.
```

...

7.1.1 Speeding Up Kubernetes Installation To Edge Nodes With The `--skip-*` Flags: Use Cases

Explanations and use cases for these flags are given in the following table:

Flag	**Use case**
`--skip-package-install`	all edge directors share the same software image, and the image is already up to date. So the installer does not need to install packages from that image to the edge director.
`--skip-image-update` and `--skip-reboot`	all edge directors are already provisioned with the up-to-date software image. So the installer does not need to carry out an update from the ISO or head node, and then reboot the edge director.
`--skip-disksetup-changes`	all edge directors already have the correct disk layout. This flag can be set if the disk layout was already configured upfront, in order to avoid full provisioning.

These flags can also be configured in the YAML configuration file of the `cm-kubernetes-setup` wizard.

The flags can be used for scripted installations for quick Kubernetes setups. For a scripted installation of an edge director, preparations can be done beforehand so that all the requirements in the software images that the edge directors use are already installed, the right disk layouts are already configured, and packages are already updated.

All the stages in the flag options can then be skipped for installing onto edge sites. This can make the setup take just a few seconds per Kubernetes deployment.

8 Singularity

The cluster manager provides an application containerization tool called Singularity. Singularity is designed to execute containers as if they are just native applications on a host computer, and to work with HPC. Singularity users can therefore run Singularity containers just as they run any other program on an HPC cluster. The cluster manager provides Singularity version 3.8.4.

8.1 Use Cases

Adding Singularity to the cluster manager brings a stronger integration of containerization with HPC. While Docker and Kubernetes can work within HPC, some drawbacks still prevent the use of HPC resources in the way that HPC users and administrators are used to.

Besides the use of Singularity containers in HPC jobs, Singularity users can create portable images with their applications. Singularity images are files that represent the container filesystem. These images can be copied from one environment (cluster) to another and executed without modification. Thus, when a user creates a container image file, it is up to the user what files, or which RPMs, to install in the image. For example, the user can create an image file that bundles Open MPI with the user's application. This guarantees that the application will be able to run if it requires that MPI implementation, even if no MPI libraries are installed on the execution host or if there is some version incompatibility.

There is no need for a special configuration inside workload managers in order to use Singularity. This is because the containers are designed to be run like any application on the system. Users need just to use the image file as the usual script or binary to be executed in their jobscripts or in a shell. The `singularity` command can also be used to apply special options to the container, when executing the image file in the jobscript or shell.

8.2 Package `cm-singularity`

Singularity is packaged for SLES12, SLES15, Ubuntu 18.04, Ubuntu 20.04, RHEL7/CentOS7, and RHEL8/Rocky Linux 8. It is available from the YUM or Zypper repositories from version 7.3 of NVIDIA Bright Cluster Manager onward, and is distributed as a package called `cm-singularity`. The package should be installed in the software image for each node. The user is able to run a Singularity image only if the Singularity package is installed on the node. In order to allow users to build an image, it makes sense to install the package on the head and login nodes as well. The tool does not provide services that run in the background, so a simple installation of the package is enough to start using it.

Singularity contexts are always run as the user running them. This means that there is no risk in allowing the containers to have access to, and interact with, the file system of the host.

This means that, if an image is created by the root user on a machine, then the files that require root access inside the image, still need to be allowed root permissions on any other machine. Thus, if a user crates an image on a laptop, and adds a file that can be read only by the root user, then when the container is started on another machine by a regular user, that regular user has no access to the root-only readable file inside the container.

While there is no daemon running as root, nor any persistent processes that an attacker may use to escalate privileges, there is a need to run some system calls as root so that the container is encapsulated. For this part of the run flow, there is a single SUID binary called Sexec (Singularity Exec). This is a simple binary that is as small as possible, and which the Singularity developers claim has been audited by multiple security experts.

8.3 MPI Integration

Because of the nature of Singularity, all MPI implementations should work fine inside a Singularity container. The developers of the tool have spent a lot of effort in making Singularity aware of Open MPI, as well as adding a Singularity module into Open MPI so that running at extreme scale is as efficient as possible. However, in some cases, starting an MPI process may not be as optimal as execution outside the container. So, specifically for Open MPI, Singularity provides a special mechanism to handle the execution of MPI processes. It adds all the MPI processes of the same MPI application to the same container on a host. This also reduces the application startup time. The Open MPI daemon `orted` in this case is not added to the running container, which means the overhead of starting up daemons is reduced.

When an Open MPI application that has been packaged to an image is started, the following steps take place:

1. `mpirun` is called;
2. `mpirun` forks and executes `orted`;
3. `orted` initializes the PMI (process management interface);
4. `orted` forks as many times as the number of processes per node requested;
5. the container image is started in each fork (because it is the original command specified in `mpirun` arguments);
6. each container process executes the command (that is, the MPI application) passed inside the given container;
7. each of the MPI process links to the dynamic Open MPI library, which loads shared libraries with `dlopen` system call;
8. Open MPI libraries connect back to the original `orted` process via PMI;
9. all non-shared memory communication then occurs through the PMI, and then passes on to local network interfaces.

Additional information about Singularity usage can be found in Chapter 11 of the *User Manual*. The official web site of the tool is `https://www.sylabs.io/singularity`.

9

OpenShift Container Platform Integration With The Cluster Manager

OpenShift is Red Hat's container manager.

This chapter describes how OpenShift 4.9 can be installed on a cluster running the cluster manager version 9.2. The integration of OpenShift with the cluster manager is deprecated and will be dropped in a future release.

The OpenShift cluster deployed in this chapter is composed of 7 nodes:

- 3 OpenShift master nodes (not managed by Bright)
- 3 RHEL OpenShift compute nodes (managed by Bright and OpenShift)
- 1 RHEL NVIDIA Bright Cluster Manager compute node running a load balancer (managed by Bright).

More OpenShift compute nodes may be added after the initial setup is completed.

The OpenShift master nodes run Red Hat's Core operating system (RHCOS), and are not Bright-managed. The OpenShift compute nodes are managed by both Bright and OpenShift.

The following official Red Hat documentation can be referred to for further details on OpenShift:

- Installing a cluster on bare metal (OpenShift Container Platform 4.9) (`https://docs.openshift.com/container-platform/4.9/installing/installing_bare_metal/installing-bare-metal.html`)
- Adding a RHEL compute machine (OpenShift Container Platform 4.9) (`https://docs.openshift.com/container-platform/4.9/machine_management/user_infra/adding-rhel-compute.html`)

9.1 Prerequisites

A cluster with 7 spare compute nodes can be used to install OpenShift 4.9. The minimal hardware specifications for 6 of the OpenShift nodes are:

Node Type	Storage	Virtual RAM	vCPU	IOPS
3 masters	100 GB	16 GB	4	300
1 bootstrap	100 GB	16 GB	4	300
2 computes	100 GB	8 GB	2	300

Aside from the 6 nodes described above, one extra node is to be used as an L4 (OSI Layer 4, the transport layer) load balancer. Its minimum specifications depend on the expected load of the OpenShift cluster. During the setup process any regular node can handle the load sufficiently.

The head node should have at least 30 GB of spare storage space for the installation.

The supported distribution versions for OpenShift 4.9 use RHEL 7.9 or RHEL 8.4. The Bright cluster is currently limited to running OpenShift on RHEL 7.9.

This session assumes that Kubernetes is not set up.

A prerequisite is that the head node, from where the installation setup is to run, has an active RHEL subscription. A RHEL subscription can be set up by running the following commands:

```
[root@bright92 ~]# subscription-manager register --username=<user_name> --password=<password>
[root@bright92 ~]# subscription-manager refresh
```

9.2 Installation

9.2.1 Head Node Setup

The subscription to `OpenShift Container Platform` should be made. Its details can be searched for by running:

```
[root@bright92 ~]# subscription-manager list --available --matches '*OpenShift Container Platform*'
```

One of the outputs in the listing is the pool ID. Using the pool ID that is obtained from the listing, something similar to the following can then be run:

```
[root@bright92 ~]# openshift_version=4.9
[root@bright92 ~]# pool_id=8a85f99b70d0559c0170d5b5fa6e5c41
[root@bright92 ~]# subscription-manager attach --pool=${pool_id}
[root@bright92 ~]# subscription-manager repos --enable="rhel-7-server-rpms" --enable="rhel-7-server-extras-\
rpms" --enable="rhel-7-server-ansible-2.8-rpms" --enable="rhel-7-server-ose-${openshift_version}-rpms"
[root@bright92 ~]# yum install -y openshift-ansible openshift-clients jq
```

9.2.2 Installing OpenShift

A pull secret file should be picked up from `https://cloud.redhat.com/openshift/install/metal`, by clicking on `Download pull secret`. After placing the file on the head node, the following is run as root on the head node:

```
[root@bright92 ~]# cm-openshift-setup
```

An interactive wizard then walks the cluster administrator through some configuration options for setting up an initial OpenShift cluster.

Some fields in the screens of the wizard are pre-filled with default values.

9.2.3 Installation Steps

When running `cm-openshift-setup`, some of the queries concern the following:

- RHEL Subscription Manager Credentials: setting these credentials is necessary in order to install the necessary packages into the OpenShift compute nodes.
- Pull Secret: The path of the file that was downloaded earlier.
- Selecting which nodes are to be used as bootstrap, master, compute, and load balancer nodes.
- Setting a name for the OpenShift Cluster, and a domain name. Any name valid for a URL (RFC 1035) works.

There are options which are not asked in the wizard, with default values pre-filled:

- `software_image_name`: The name of the software image used by cmdaemon when provisioning new OpenShift compute nodes.
- `category_name`: The name of the category assigned by cmdaemon to OpenShift compute nodes.
- `openshift_version`: Currently set to `4.9.x`. Changing this value is not supported at the time of writing (November 2021).
- `binaries_urls`: The default values can be used initially. In future installations, the administrator may want to host those files at a local server, and provide the path for those files in that server, in order to speed up the process in future installations.
- `ssh_key_path`: The RSA Key used to interact with the nodes. If the key in the path does not yet exist, then a new one is generated, which is recommended. The administrator may provide an existing key instead.
- `openshift_config_dir`: Path to store the OpenShift configurations. The default path should work fine.
- `roles.openshift_load_balancer`: Used to set up the OpenShift load balancer.
- `pxe_template`: Used to populate the PXE boot template, in order to boot RHCOS nodes.

Installation progress can be examined in greater detail in the log file:

Example

```
[root@bright92 ~]# tail -f /var/log/cm-openshift-setup.log
```

Installation can take around 40 to 90 minutes. After installation is complete, the kubeadmin password can be found in the following path:

```
[root@bright92 ~]# cat /etc/openshift/auth/kubeadmin-password
```

9.2.4 Adding New Compute Nodes

After the installation is complete, new OpenShift compute nodes can be added into the OpenShift cluster. There are two ways to do so.

1. One way is interactive, using the wizard again:

   ```
   [root@bright92 ~]# cm-openshift-setup
   ```

   In the main TUI screen of the wizard, the option `Add OpenShift Compute Nodes` should be selected. The dialog that follows then asks for nodes to be selected with:

   - `Select node assigned as the balancer for OpenShift`: This is the balancer node that was chosen by the administrator during the installation step
   - `Select the nodes to be reprovisioned into OpenShift Compute Nodes`: These are the compute nodes.

2. The second way uses a configuration file with the `cm-openshift-setup` wizard. The configuration file can be generated by `cm-openshift-setup` after selecting the `Add OpenShift Compute Nodes` option.

   ```
   [root@bright92 ~]# cm-openshift-setup --add-compute-nodes -c <path to cm-openshift-setup config file>
   ```

   The nodes that are added are moved to a different category and software image.

9.2.5 Validation

All the nodes that were selected during setup can be listed with.

```
[root@bright92 ~]# oc get nodes
```

The load balancer does not appear in the list since it is not managed by OpenShift.

A validation check for the cluster operators is:

```
[root@bright92 ~]# oc get clusteroperators
```

If all values in the column `AVAILABLE` are set to `True`, then the installation has been successful.

9.2.6 Uninstall

The following 3 ways of uninstalling OpenShift are possible if it was installed using the `cm-openshift-setup` wizard.

1. The setup wizard can be run interactively, and the `Uninstall OpenShift` option can be selected:

   ```
   [root@bright92 ~]# cm-openshift-setup
   ```

   [*TUI screen opens up*]

   When uninstalling, a confirmation is requested for the values of the fields `unmanaged_config_name`, `category_name` and `software_image_name`. Those fields are pre-filled with the default values. A change is only needed if these were explicitly changed in the installation configuration file generated by the wizard during the installation process (section 9.2).

2. The setup wizard can be run non-interactively for an uninstall as follows:

   ```
   [root@bright92 ~]# cm-openshift-setup --remove
   ```

   Run like this, the default values are used.

3. The setup wizard can also be run non-interactively for an uninstall while providing a configuration file for the values, using the following format:

   ```
   [root@bright92 ~]# cm-openshift-setup --remove -c <path to cm-openshift-setup configuration file>
   ```