TABLE OF CONTENTS

Chapter 1. Introduction ......................................................................................... 1
Chapter 2. Installing Kubernetes Using DeepOps......................................................... 2
Chapter 3. Installing Kubernetes Using Kubeadm.......................................................... 3
  3.1. Before You Begin .......................................................................................... 3
    3.1.1. Master Nodes ......................................................................................... 4
    3.1.2. Worker Nodes ........................................................................................ 4
    3.1.3. Cluster Customization .............................................................................. 5
  3.2. Cluster Management ...................................................................................... 5
    3.2.1. Device Plugin ......................................................................................... 6
    3.2.2. NVIDIA Labels ........................................................................................ 6
    3.2.3. Monitoring NVIDIA GPUs ....................................................................... 6
      3.2.3.1. Setting Up Monitoring ......................................................................... 7
      3.2.3.2. Looking Up Results ............................................................................. 7
    3.2.4. Run GPU Workloads ................................................................................. 8
      3.2.4.1. Look Up GPUs ................................................................................... 8
      3.2.4.2. Run a GPU Workload ......................................................................... 8
      3.2.4.3. Tainting Your Master Node .................................................................... 9
    3.2.5. Driver Containers (Beta) ........................................................................... 9
Appendix A. Kubeadm: Prerequisites ...................................................................... 11
  A.1. DGX and NGC Images .................................................................................. 11
  A.2. Install NVIDIA Container Runtime for Docker 2.0 ........................................ 11
    A.2.1. Uninstall Old Versions ............................................................................. 12
    A.2.2. Set Up the Repository ............................................................................. 12
Appendix B. Kubeadm: Setting Up the CRI Runtime .................................................... 13
  B.1. Docker ..................................................................................................... 13
  B.2. Containerd ................................................................................................ 13
  B.3. CRIO ........................................................................................................ 14
Appendix C. Kubeadm: Troubleshooting ................................................................... 15
  C.1. Package Installation .................................................................................... 16
  C.2. Cluster Initialization .................................................................................... 16
  C.3. Monitoring Issues ....................................................................................... 17
Chapter 1.
INTRODUCTION

Kubernetes is an open-source platform for automating deployment, scaling and managing containerized applications. Kubernetes includes support for GPUs and enhancements to Kubernetes so users can easily configure and use GPU resources for accelerating workloads such as deep learning. This document describes two step-by-step methods for installing upstream Kubernetes with NVIDIA supported components, such as drivers and runtime, and usage with NVIDIA GPUs - a method using DeepOps and a method using Kubeadm.

To set up orchestration and scheduling in your cluster, it is highly recommended that you use DeepOps. DeepOps is a modular collection of ansible scripts which automate the deployment of Kubernetes, Slurm, or a hybrid combination of the two across your nodes. It also installs the necessary GPU drivers, NVIDIA Container Runtime for Docker (nvidia-docker2), and various other dependencies for GPU-accelerated work. Encapsulating best practices for NVIDIA GPUs, it can be customized or run as individual components, as needed.
Kubernetes can be deployed through different mechanisms. Use DeepOps to automate deployment, especially for a cluster of many worker nodes. Use the following procedure to install Kubernetes using DeepOps:

1. Pick a provisioning node to deploy from.
   This is where the DeepOps Ansible scripts run from and is often a development laptop that has a connection to the target cluster. On this provisioning node, clone the DeepOps repository with the following command.
   
   ```bash
   $ git clone https://github.com/NVIDIA/deepops.git
   ```

2. Optionally, check out a recent release tag with the following command.
   ```bash
   $ cd deepops
   $ git checkout tags/20.02
   ```
   If you do not explicitly use a release tag, then the latest development code is used, and not an official release.

Chapter 3.
INSTALLING KUBERNETES USING KUBEADM

The method described in this section is an alternative to using DeepOps. If you have deployed using DeepOps, then skip this section.

For a less scripted approach, especially for smaller clusters or where there is a desire to learn the components that make up a Kubernetes cluster, use Kubeadm.

A Kubernetes cluster is composed of master nodes and worker nodes. The master nodes run the control plane components of Kubernetes which allows your cluster to function properly. These components include the API Server (front-end to the kubectl CLI), etcd (stores the cluster state) and others.

Use CPU-only (GPU-free) master nodes, which run the control plane components: Scheduler, API-server, and Controller Manager. Control plane components can have some impact on your CPU intensive tasks and conversely, CPU or HDD/SSD intensive components can have an impact on your control plane components.

3.1. Before You Begin

Before proceeding to install the components, check that all Kubernetes prerequisites have been satisfied. These prerequisites include:

- Check network adapters and required ports
- Disable swap on the nodes so that kubelet can work correctly
- Install dependencies such as Docker. To install Docker on Ubuntu, follow the official instructions provided by Docker.
- The worker nodes are provisioned with the NVIDIA driver.
- Ensure that NVIDIA Container Runtime for Docker 2.0 is installed on the GPU worker nodes. Whether you are setting up a single node GPU cluster for development purposes, or you want to run jobs on the master nodes as well, you must install the NVIDIA Container Runtime for Docker.

Instructions for how to set up the NVIDIA Container Runtime for Docker, Containerd, and CRI-O are provided in Kubeadm: Setting Up the CRI Runtime.
3.1.1. Master Nodes

1. Install the required components on your master node by updating the apt package index and installing the packages with the following commands:

   $ apt-get update & amp; apt-get install -y apt-transport-https
   $ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
   $ cat <<EOF &gt;/etc/apt/sources.list.d/kubernetes.list
   deb https://apt.kubernetes.io/ $(lsb_release -c) main
   EOF
   $ sudo chmod 644 /etc/apt/sources.list.d/kubernetes.list
   $ apt-get update
   $ apt-get install -y kubelet kubeadm kubectl
   $ apt-mark hold kubelet kubeadm kubectl

2. Start your cluster with the following commands. Save the token and the hash of the CA certificate as part of kubeadm init for joining worker nodes to the cluster later. This will take a few minutes.

   $ sudo kubeadm init --pod-network-cidr=10.244.0.0/16
   ...
   $ mkdir -p "$HOME_DIR/.kube"
   $ cp /etc/kubernetes/admin.conf "$HOME_DIR/.kube/config"
   $ chown -R "$uid -u":$uid "$HOME_DIR/.kube"

3. Check that all the control plane components are running on the master node with the following command:

   $ kubectl get nodes
   NAME     STATUS    ROLES     AGE       VERSION
   127.0.0.1 Unhealthy <none> 10m       v1.13.2

3.1.2. Worker Nodes

1. Install the required components on your worker node by updating the apt package index and installing the packages with the following commands:

   $ apt-get update & amp; apt-get install -y apt-transport-https
   $ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -
   $ cat &lt;&lt;EOF &gt;/etc/apt/sources.list.d/kubernetes.list
   deb https://apt.kubernetes.io/ kubernetes-xenial main
   EOF
   $ sudo chmod 644 /etc/apt/sources.list.d/kubernetes.list
   $ apt-get update
   $ apt-get install -y kubelet kubeadm kubectl
   $ apt-mark hold kubelet kubeadm kubectl

2. Before starting your cluster, retrieve the token and CA certificate hash you recorded from when “kubeadm init” was run on the master node. Alternatively, to retrieve the token, use the following command:

   $ sudo kubeadm token create --print-join-command

3. Join the worker node to the cluster with a command similar to the following. (The token shown below is an example that will not work for your installation.)
Installing Kubernetes Using Kubeadm

3.1.3. Cluster Customization

After starting up your cluster, NVIDIA provides a script, post-install.sh, which automatically executes a few post-install steps, to setup your cluster with the following basic utilities:

- The Flannel network plugin
- The Helm package manager for Kubernetes
- The NVIDIA Device Plugin, allowing you to enable GPU support
- The NVIDIA Monitoring Stack

Obtain and run post-install.sh with the following commands:

```
$ curl https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/examples/post-install.sh
$ chmod 700 post-install.sh
$ ./post-install.sh
```

The three commands can be invoked as one command as well:

```
$ curl https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/examples/post-install.sh | bash
```

Helm helps you manage and deploy Kubernetes applications. Helm charts, packages of pre-configured Kubernetes resources, allow you to define, install, and upgrade complex Kubernetes applications. NVIDIA applications, for example, are deployed through Helm charts.

Flannel is a network plugin for Kubernetes that sets up a layer 3 network fabric. Flannel runs a small, single binary agent called flanneld on each host, and is responsible for allocating a subnet lease to each host out of a larger, preconfigured address space.

In Kubernetes, clusters require a pod network add-on installed. Flannel is chosen for installation by post-install.sh for multiple reasons:

- It is recommended by Kubernetes
- It is used in production clusters
- It integrates well with the CRI-O runtime

To check the health of your cluster, run the following command on the master node and make sure your GPU worker nodes appear and their states transition to Healthy. That may take a few minutes for the status to change.

```
$ kubectl describe nodes
```

3.2. Cluster Management

Kubernetes offers a number of features that cluster admins can leverage in order to better manage NVIDIA GPUs:
Driver Containers allow you to manage NVIDIA drivers.

NVIDIA Device plugin allows you to expose the GPU resource to the Kubernetes API.

Labels allow you to identify GPU nodes and attributes to steer workloads accordingly.

GPU Monitoring Agent allows you to extract valuable metrics from your GPU and associate them with running pods.

The following sections describe how these features can be used.

### 3.2.1. Device Plugin

Starting in version 1.8, Kubernetes provides a device plugin framework for vendors to advertise their resources to the kubelet without changing Kubernetes core code. Instead of writing custom Kubernetes code, vendors can implement a device plugin that can be deployed manually or as a DaemonSet.

The NVIDIA device plugin for Kubernetes is a DaemonSet that allows you to automatically:

- Expose the number of GPUs on each nodes of your cluster
- Keep track of the health of your GPUs
- Run GPU enabled containers in your Kubernetes cluster.

If you have setup your nodes as presented in the above sections, you only need to deploy a DaemonSet. The GPU information will show up on your node fairly quickly.

Run the following command to create the device plugin and watch the GPU informations being exposed inside your cluster:

```
kubectl create -f https://raw.githubusercontent.com/NVIDIA/k8s-device-plugin/1.0.0-beta/nvidia-device-plugin.yml
```

### 3.2.2. NVIDIA Labels

NVIDIA exposes a standard set of labels for steering your workloads to different nodes. In the future, these labels will allow you to set family quotas per GPU (e.g: This namespace has 10 V100 and 10 P100).

If you describe the nodes can see a number of labels:

```yaml
...
nvidia.com/node: "GPU"
gpu.nvidia.com/family: "volta"
gpu.nvidia.com/memory: "16GB"
...
```

### 3.2.3. Monitoring NVIDIA GPUs

An integrated monitoring stack is provided for monitoring GPUs in Kubernetes. The stack uses the NVIDIA Datacenter GPU Manager (DCGM), Prometheus (using Prometheus Operator), and Grafana for visualizing the various metrics.
3.2.3.1. Setting Up Monitoring

1. Install the monitoring charts (This is done automatically if you run post-install.sh after installation):

   ```
   $ helm repo update
   $ helm install gpu-helm-charts/prometheus-operator --name prometheus-operator --namespace monitoring
   $ helm install gpu-helm-charts/kube-prometheus --name kube-prometheus --namespace monitoring
   ```

2. Label the GPU nodes and ensure that the label has been applied with the following commands:

   ```
   $ kubectl label nodes <gpu-node-name> hardware-type=NVIDIAGPU
   $ kubectl get nodes --show-labels
   ```

3. Check the status of the pods. Note that it may take a few minutes for the components to initialize and start running.

   ```
   $ kubectl get pods -n monitoring
   NAME                                                   READY     STATUS    AGE
   alertmanager-kube-prometheus-0                         2/2       Running   1h
   kube-prometheus-exporter-kube-state-5c67546677-52sgg   2/2       Running   1h
   kube-prometheus-exporter-node-4568d                    2/2       Running   1h
   kube-prometheus-grafana-694956469c-xwk6x               2/2       Running   1h
   prometheus-kube-prometheus-0                           2/2       Running   1h
   prometheus-operator-789d76f7b9-t5qqz                    1/1       Running   1h
   ```

3.2.3.2. Looking Up Results

1. To view the Grafana dashboard in a browser, forward the port with the following commands:

   ```
   $ export GRAFANA_POD=$(kubectl get pods -n monitoring | grep grafana | tr -s ' ' | cut -f 1 -d ' ')
   $ kubectl -n monitoring port-forward $GRAFANA_POD 3000
   ```

2. Open a browser window and type http://localhost:3000 to view the Nodes Dashboard in Grafana. To view GPU metrics in Grafana, create a docker secret to authenticate against nvcr.io and run the resnet sample (provided in the examples directory).

   ```
   $ kubectl create secret docker-registry nvcr.dgxkey --docker-server=nvcr.io --docker-username='$oauthtoken' --docker-password=<NGC API Key> --docker-email=<email>
   $ kubectl create -f /etc/kubeadm/examples/nbody.yml
   ```

GPU metrics are displayed on the Nodes dashboard of Grafana:
3.2.4. Run GPU Workloads

3.2.4.1. Look Up GPUs

GPU plugins are deployed when you run post-install.sh. It may take 1-2 minutes for GPUs to be enabled on your cluster (i.e. for Kubernetes to download and run containers). Once this is done, run the following command to see GPUs listed in the resource section:

```
$ kubectl describe nodes | grep -B 3 gpu
```

Capacity:
- cpu: 8
- memory: 32879772Ki
- nvidia.com/gpu: 2

Allocatable:
- cpu: 8
- memory: 32777372Ki
- nvidia.com/gpu: 2

3.2.4.2. Run a GPU Workload

1. Make sure GPU support is properly set up by running a simple CUDA container. See this web site for samples: https://github.com/NVIDIA/k8s-device-plugin/tree/examples.
2. Start the CUDA sample workload with the following command:

```
$ kubectl create -f https://github.com/NVIDIA/k8s-device-plugin/blob/examples/workloads/deployment.yml
```
3. When the pod is running, execute the nvidia-smi command inside the container with the following command:

```
$ kubectl exec -it gpu-pod nvidia-smi
```

+----------------------------------------------------------------------------+
| NVIDIA-SMI 384.125 Driver Version: 384.125 |  
| --------------------------+---------+---------------- |  
| GPU Name Persistence-M| Bus-Id | Disp.A | Volatile Uncorr. Memory-Usage | GPU-Util Compute M. |  
|----------------------|---------|---------+----------------------- | |  
| 0 Tesla V100-SXM2... On | 00000000:00:1E.0 Off |  
| N/A 34C P0 20W / 300W | 10MiB / 16152MiB | 0% Default |  
+----------------------------------------------------------------------------+
3.2.4.3. Tainting Your Master Node

This step is optional. If you are setting up a multi-node cluster and you don’t want jobs to run on the master node (as they may impact the performance of your control plane), set the master Kubernetes to deny pods that can not run on the master node:

$ kubectl taint nodes <MasterNodeName> node-role.kubernetes.io/master

All nodes are untainted by default.

3.2.5. Driver Containers (Beta)

Driver containers are a beta feature that allows for containerized provisioning of the NVIDIA driver. This provides several benefits over a standard driver installation:

- Ease of deployment
- Fast installation
- Reproducibility

Do not use driver containers on systems with the NVIDIA driver pre-installed, such as DGX.

Driver Containers require special configuration, as the `ipmi_msghandler` kernel module needs to be loaded and the `nvidia-container-runtime` needs to be configured to look for the driver libraries in a specific location.

Run the following commands on the node before launching the driver container.

$ sudo sed -i 's/^#root/root/' /etc/nvidia-container-runtime/config.toml
$ sudo modprobe ipmi_msghandler

Run the following commands in your cluster to deploy the driver container:

Caution Make sure to replace `<DRIVER_VERSION>` in the commands below with the desired driver version.

For bare-metal implementations, use the following commands:

$ export DISTRIBUTION=$(./etc/os-release; echo $ID$VERSION_ID)
$ export DRIVER_VERSION=<DRIVER_VERSION>
$ curl -s -L -o nvidia-driver.yaml.template https://gitlab.com/nvidia/samples/raw/doc-6-6-2019/driver/kubernetes/nvidia-driver.yaml.template
$ envsubst < nvidia-driver.yaml.template > nvidia-driver.yaml
$ kubectl create -f nvidia-driver.yaml
For AWS implementations, use the following commands:

```bash
$ export DISTRIBUTION=$(./etc/os-release; echo $ID$VERSION_ID)-aws
$ export DRIVER_VERSION=<DRIVER_VERSION>
$ curl -s -L -o nvidia-driver.yaml.template https://gitlab.com/nvidia/samples/raw/doc-6-6-2019/driver/kubernetes/nvidia-driver.yaml.template
$ envsubst < nvidia-driver.yaml.template > nvidia-driver.yaml
$ kubectl create -f nvidia-driver.yaml
```
This section details the prerequisites for setting up a Kubernetes node. The prerequisites include:

- The worker nodes must be provisioned with the NVIDIA driver. The recommended way is to use your package manager and install the cuda-drivers package (or equivalent). When no packages are available, use an official runfile from the NVIDIA driver downloads site.
- Ensure that a supported version of Docker is installed before proceeding to install the NVIDIA Container Runtime for Docker (via nvidia-docker2).

A.1. DGX and NGC Images

On DGX systems installed with nvidia-docker version 1.0.1, NVIDIA provides an option to upgrade the existing system environment to the new NVIDIA Container Runtime. Follow the instructions in Upgrading to the NVIDIA Container Runtime for Docker to upgrade your environment. Skip the following section and proceed to installing Kubernetes on the worker nodes.

If you are using the NGC images on AWS or GCP, then you may skip the following section and proceed to installing Kubernetes on the worker nodes.

A.2. Install NVIDIA Container Runtime for Docker 2.0

NVIDIA Container Runtime for Docker (nvidia-docker2) is the supported method for running GPU containers. It is more stable than nvidia-docker 1.0 and is expected to be used with the Device Plugin feature, which is the official method for using GPUs in Kubernetes. Use the procedures in the following sections to install NVIDIA Container Runtime for Docker 2.0.
A.2.1. Uninstall Old Versions

1. List the volumes with the following command:

   ```
   docker volume ls -q -f driver=nvidia-docker | xargs -r -I{} -n1 docker ps -q -a -f volume={} | xargs -r docker rm -f
   ```

2. Remove nvidia-docker-1.0 with the following command:

   ```
   sudo apt-get purge -y nvidia-docker
   ```

A.2.2. Set Up the Repository

1. Add the official GPG keys with the following commands:

   ```
   distribution=$(./etc/os-release;echo $ID$VERSION_ID)
   curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey | sudo apt-key add -
   curl -s -L https://nvidia.github.io/nvidia-docker/$distribution/amd64/nvidia-docker.list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list
   ```

2. Update the apt package index:

   ```
   sudo apt-get update
   ```
NVIDIA GPUs are supported by all of the major Kubernetes runtimes (Docker, Containerd and CRIO). Each of them require some level of configuration for the GPU enablement to be available, however.

**B.1. Docker**

When using Docker with Kubernetes, you must set up the NVIDIA runtime as the default Docker runtime. This allows Kubernetes to inform Docker of the correct GPUs to expose.

As directed in the prerequisite section, make sure that the docker runtime is set to NVIDIA:

```
$ docker info | grep nvidia
Runtimes: nvidia runc
Default Runtime: nvidia

$ cat /etc/docker/daemon.json
{
  "default-runtime": "nvidia",
  "runtimes": {
    "nvidia": {
      "path": "/usr/bin/nvidia-container-runtime",
      "runtimeArgs": []
    }
  }
}
```

**B.2. Containerd**

As with Docker, when using containerd with Kubernetes, you must setup the NVIDIA runtime as the default Docker runtime. This allows kubernetes to inform containerd of the correct GPUs to expose.

Make sure that the containerd runtime is set to the nvidia-container-runtime binary, as shown below.
B.3. CRIO

Follow the steps at https://blog.openshift.com/how-to-use-gpus-with-deviceplugin-in-openshift-3-10/ to use the CRIO (OpenShift 3.10) runtime.
Appendix C.
KUBEADM: TROUBLESHOOTING

Use the following steps to perform basic troubleshooting.

1. Check the logs and verify that the Kubelet is running:

```
$sudo systemctl status kubelet
$sudo journalctl -xeu kubelet
$subectl cluster-info dump
```

2. Check that all the control plane components are running and healthy:

```
$kubectl get pods --all-namespaces
```

<table>
<thead>
<tr>
<th>NAMESPACE</th>
<th>NAME</th>
<th>READY</th>
<th>STATUS</th>
</tr>
</thead>
<tbody>
<tr>
<td>kube-system</td>
<td>etcd-master</td>
<td>1/1</td>
<td>Running</td>
</tr>
<tr>
<td></td>
<td>kube-apiserver-master</td>
<td>1/1</td>
<td>Running</td>
</tr>
<tr>
<td></td>
<td>kube-controller-manager-master</td>
<td>1/1</td>
<td>Running</td>
</tr>
<tr>
<td></td>
<td>kube-dns-55856cb6b6-c9mvz</td>
<td>3/3</td>
<td>Running</td>
</tr>
<tr>
<td></td>
<td>kube-flannel-ds-qddkv</td>
<td>1/1</td>
<td>Running</td>
</tr>
<tr>
<td></td>
<td>kube-proxy-549sh</td>
<td>1/1</td>
<td>Running</td>
</tr>
<tr>
<td></td>
<td>kube-scheduler-master</td>
<td>1/1</td>
<td>Running</td>
</tr>
<tr>
<td></td>
<td>nvidia-device-plugin-daemonset-1.9.10-5m95f</td>
<td>1/1</td>
<td>Running</td>
</tr>
<tr>
<td></td>
<td>tiller-deploy-768dcd4457-j2ffs</td>
<td>1/1</td>
<td>Running</td>
</tr>
<tr>
<td></td>
<td>alertmanager-kube-prometheus-0</td>
<td>2/2</td>
<td>Running</td>
</tr>
<tr>
<td></td>
<td>kube-prometheus-exporter-kube-state-52sgg</td>
<td>2/2</td>
<td>Running</td>
</tr>
<tr>
<td></td>
<td>kube-prometheus-grafana-694956469c-xwk6x</td>
<td>2/2</td>
<td>Running</td>
</tr>
<tr>
<td></td>
<td>prometheus-kube-prometheus-0</td>
<td>2/2</td>
<td>Running</td>
</tr>
<tr>
<td></td>
<td>prometheus-operator-789d76f7b9-t5qqz</td>
<td>1/1</td>
<td>Running</td>
</tr>
</tbody>
</table>

3. Attempt to restart Kubeadm (restarting the kubelet may be required):

```
$sudo kubeadm reset
$sudo systemctl start kubelet
```
C.1. Package Installation

This topic describes common package installation issues and workarounds for those issues.

**apt-update Warnings or Package Installation Errors**

Check if the official GPG keys and repositories have been added correctly.

**To Verify if Kubernetes (kubelet, kubectl, and kubeadm) is Installed Properly**

Verify correct installation with the following commands.

```
$ dpkg -l '*kube*' | grep +nvidia
$ ls /etc/kubeadm
```

C.2. Cluster Initialization

This topic describes common cluster initialization issues and workarounds for those issues.

**If the Initialization of the Cluster (kubeadm init) Fails**

Check status and logs with the following commands:

```
$ sudo systemctl status kubelet
$ sudo journalctl -xeu kubelet
```

Check for commonly encountered network failures during the initialization process:

- Check if port is already busy with `netstat -lptn`.
- Check for time out with `ufw status` (firewall)

Restart Kubeadm with the following command.

```
$ kubeadm reset
```

It might be necessary to start kubelet after restarting Kubeadm.

Remove the user account to administer the cluster with the following command.

```
$ rmdir $HOME/.kube
```

**Verify User Account Permissions for Cluster Administration ($HOME/.kube)**

Verify permissions of the user account with the following command:

```
$ sudo chown -R $(id -u):$(id -g) $HOME/.kube
```
**Determine If the Pod Status is Other Than “Running”**

Determine pod status with the following commands:

```bash
$ kubectl describe pod POD_NAME
$ kubectl get events --all-namespaces
$ kubectl logs -n NAMESPACE POD_NAME -c CONTAINER
```

**Validation Errors**

When running a GPU pod, you may encounter the following error:

```bash
error: error validating "/etc/kubeadm/pod.yml": error validating data:
ValidationError(Pod.spec.extendedResources[0]): unknown field "resources" in io.k8s.api.core.v1.PodExtendedResource
```

Ensure that you have installed the Kubernetes components (kubectl, kubelet and kubeadm) from NVIDIA and not upstream Kubernetes.

**C.3. Monitoring Issues**

This section describes common monitoring issues and workarounds.

**Common Grafana Errors**

Check if the Grafana pod is deployed and running:

```bash
$ helm list
```

**Port Forwarding Errors**

Check if port 3000 is already busy with the `netstat -lptn` command.

Kill the port-forwarding process with the following commands:

```bash
$ jobs | grep grafana
$ kill %JOB_ID
```
Notice

THE INFORMATION IN THIS GUIDE AND ALL OTHER INFORMATION CONTAINED IN NVIDIA DOCUMENTATION REFERENCED IN THIS GUIDE IS PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE INFORMATION FOR THE PRODUCT, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE. Notwithstanding any damages that customer might incur for any reason whatsoever, NVIDIA's aggregate and cumulative liability towards customer for the product described in this guide shall be limited in accordance with the NVIDIA terms and conditions of sale for the product.

THE NVIDIA PRODUCT DESCRIBED IN THIS GUIDE IS NOT FAULT TOLERANT AND IS NOT DESIGNED, MANUFACTURED OR INTENDED FOR USE IN CONNECTION WITH THE DESIGN, CONSTRUCTION, MAINTENANCE, AND/OR OPERATION OF ANY SYSTEM WHERE THE USE OR A FAILURE OF SUCH SYSTEM COULD RESULT IN A SITUATION THAT THREATENS THE SAFETY OF HUMAN LIFE OR SEVERE PHYSICAL HARM OR PROPERTY DAMAGE (INCLUDING, FOR EXAMPLE, USE IN CONNECTION WITH ANY NUCLEAR, AVIONICS, LIFE SUPPORT OR OTHER LIFE CRITICAL APPLICATION). NVIDIA EXPRESSLY DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY OF FITNESS FOR SUCH HIGH RISK USES. NVIDIA SHALL NOT BE LIABLE TO CUSTOMER OR ANY THIRD PARTY, IN WHOLE OR IN PART, FOR ANY CLAIMS OR DAMAGES ARISING FROM SUCH HIGH RISK USES.

NVIDIA makes no representation or warranty that the product described in this guide will be suitable for any specified use without further testing or modification. Testing of all parameters of each product is not necessarily performed by NVIDIA. It is customer’s sole responsibility to ensure the product is suitable and fit for the application planned by customer and to do the necessary testing for the application in order to avoid a default of the application or the product. Weaknesses in customer’s product designs may affect the quality and reliability of the NVIDIA product and may result in additional or different conditions and/or requirements beyond those contained in this guide. NVIDIA does not accept any liability related to any default, damage, costs or problem which may be based on or attributable to: (i) the use of the NVIDIA product in any manner that is contrary to this guide, or (ii) customer product designs.

Other than the right for customer to use the information in this guide with the product, no other license, either expressed or implied, is hereby granted by NVIDIA under this guide. Reproduction of information in this guide is permissible only if reproduction is approved by NVIDIA in writing, is reproduced without alteration, and is accompanied by all associated conditions, limitations, and notices.

Trademarks

NVIDIA and the NVIDIA logo are trademarks and/or registered trademarks of NVIDIA Corporation in the Unites States and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2020 NVIDIA Corporation. All rights reserved.

www.nvidia.com