KUBERNETES ON NVIDIA GPUS

DU-09016-001_v1.0 | November 2018

Installation Guide

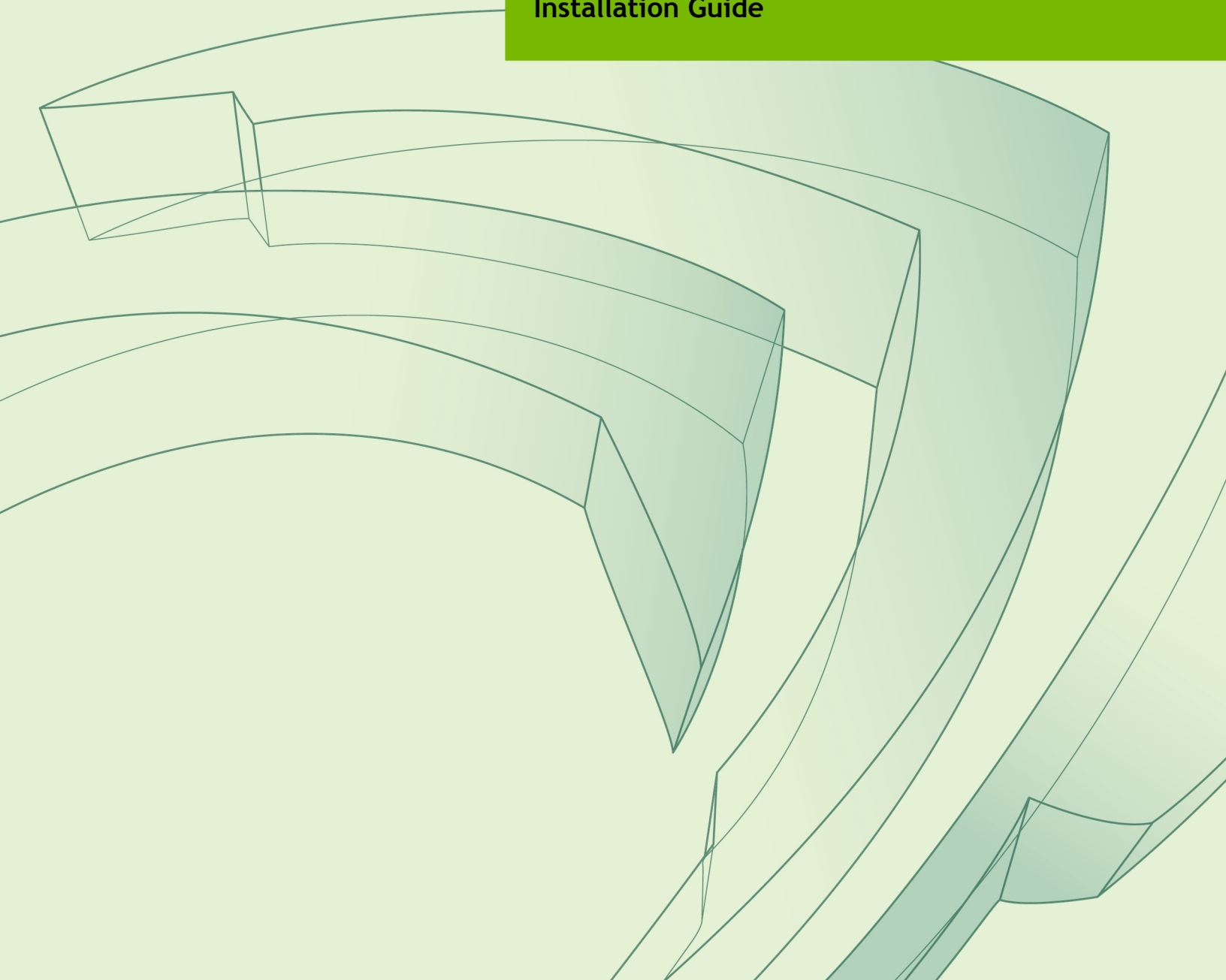


TABLE OF CONTENTS

Chapter 1. INTRODUCTION

Kubernetes is an open-source platform for automating deployment, scaling and managing containerized applications. Kubernetes on NVIDIA GPUs 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 serves as a step-by-step guide to installing Kubernetes and using it with NVIDIA GPUs.

Chapter 2. SUPPORTED PLATFORMS

Releases of Kubernetes up to and including 1.10.8 are supported on the following platforms. Note that there are certain prerequisites that must be satisfied before proceeding to install Kubernetes. These are detailed in the "Before you begin" section.

On-Premises

- DGX-1 Pascal and Volta with OS Server v3.1.6
- DGX-Station with OS Desktop v3.1.6

Cloud

- NVIDIA GPU Cloud virtual machine images available on Amazon EC2 and Google Cloud Platform.

Cluster Topology

- One master CPU or GPU node
- At least one worker GPU node

Before You Begin

- Ensure that NVIDIA drivers are loaded
- Ensure that a supported version of Docker is installed.
- Ensure that NVIDIA Container Runtime for Docker 2.0 is installed.

Chapter 3. INSTALLING KUBERNETES

Kubernetes can be deployed through different mechanisms. NVIDIA recommends using `kubeadm` to deploy Kubernetes.

The master nodes run the control plane components of Kubernetes. These include the API Server (front-end to the kubectl CLI), `etcd` (stores the cluster state) and other components. Master nodes need to be setup with the following three components, of which only the kubelet has been customized with changes from NVIDIA:

- Kubelet
- Kubeadm
- Kubectl

We recommend that your master nodes not be equipped with GPUs and to only run the master components, such as the following:

- Scheduler
- API-server
- Controller Manager

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

- Check network adapters and required ports.
- Disable swap for kubelet to work correctly
- Install dependencies such as the Docker container runtime. To install Docker on Ubuntu, follow the official instructions provided by Docker.
- The worker nodes must be provisioned with the NVIDIA driver.
- Ensure that NVIDIA Container Runtime for Docker 2.0 is installed on the machine
- Run **`ps -ef | grep resolv`** to determine whether Network Manager or systemd-resolved is being used.

If you are setting up a single node GPU cluster for development purposes or you want to run jobs on the master nodes as well, then you must install the NVIDIA Container Runtime for Docker.

3.1. Master Nodes

Install the required components on your master node:

3.1.1. Installing and Running Kubernetes

1. Add the official GPG keys.

```
$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
$ curl -s -L https://nvidia.github.io/kubernetes/gpgkey | sudo apt-key add -
$ curl -s -L https://nvidia.github.io/kubernetes/ubuntu16.04/nvidia-kubernetes.list |\
            sudo tee /etc/apt/sources.list.d/nvidia-kubernetes.list
```

2. Update the package index.

```
$ sudo apt update
```

3. Install packages.

```
$ VERSION=1.10.8+nvidia
$ sudo apt install -y kubectl=${VERSION} kubelet=${VERSION} \
                                    kubeadm=${VERSION} helm=${VERSION}
```

4. Start your cluster.

```
$ sudo kubeadm init --ignore-preflight-errors=all --config /etc/kubeadm/config.yml
```

   You may choose to save the token and the hash of the CA certificate as part of of **`kubeadm init`** to join worker nodes to the cluster later. This will take a few minutes.

5. For NGC VMIs, issue a **`chmod`** command on the newly created file.

```
$ sudo chmod 644 /etc/apt/sources.list.d/nvidia-kubernetes.list
```

3.1.2. Checking Cluster Health

Check that all the control plane components are running on the master node:

```
$ kubectl get all --all-namespaces
NAMESPACE     NAME                                    READY     STATUS
 RESTARTS   AGE
kube-system   etcd-master                               1/1     Running   0
          2m
kube-system   kube-apiserver-master                     1/1     Running   0
          1m
kube-system   kube-controller-manager-master            1/1     Running   0
          1m
kube-system   kube-dns-55856cb6b6-c9mvz                 3/3     Running   0
          2m
kube-system   kube-flannel-ds-qddkv                     1/1     Running   0
          2m
kube-system   kube-proxy-549sh                          1/1     Running   0
          2m
kube-system   kube-scheduler-master                     1/1     Running   0
          1m
```

```
kube-system   nvidia-device-plugin-daemonset-1.9.10-5m95f  1/1     Running   0
         2m
kube-system   tiller-deploy-768dcd4457-j2ffs               1/1     Running   0
         2m
monitoring    alertmanager-kube-prometheus-0               2/2     Running   0
         1m
monitoring    kube-prometheus-exporter-kube-state-52sgg    2/2     Running   0
         1m
monitoring    kube-prometheus-grafana-694956469c-xwk6x     2/2     Running   0
         1m
monitoring    prometheus-kube-prometheus-0                 2/2     Running   0
         1m
monitoring    prometheus-operator-789d76f7b9-t5qqz         1/1     Running   0
         2m
```

3.1.3. DGX Station

On DGX Station (and other Ubuntu 16.04 desktop systems), there is a known issue with Kubernetes 1.9.7 and Ubuntu 16.04 Desktop where the **kube-dns** service will not run. In order to work around this issue, take the following actions, depending on the DNS resolver service you are using. In most cases for Ubuntu 16.04 desktop systems, NetworkManager is the DNS resolver service and the procedure in For NetworkManager applies.

Run **ps -ef | grep resolv** to determine whether Network Manager or systemd-resolved is being used.

3.1.3.1. For NetworkManager

1. Find the active interface.

   ```
   $ route | grep '^default' | grep -o '[^ ]*$'
   ```

   (Alternately, use `ifconfig`.)

2. Determine the nameservers. For interface, use the active interface listed in the output of the previous command.

   ```
   $ nmcli device show interface | grep IP4.DNS
   ```

   For example:

   ```
   $ nmcli device show enp2s0f0 | grep IP4.DNS
   IP4.DNS[1]:                             192.0.2.0
   IP4.DNS[2]:                             192.0.2.1
   IP4.DNS[3]:                             192.0.2.2
   ```

3. Create the configuration file and submit it to kubernetes with the following commands:

   ```
   $ cat config.yml
   apiVersion: v1
   kind: ConfigMap
   metadata:
     name: kube-dns
     namespace: kube-system
   data:
     upstreamNameservers: | # 3 Nameservers Maximum ["192.0.2.0", "192.0.2.1",
    "192.0.2.0"]
   $ kubectl create -f config.yml
   ```

3.1.3.2. For systemd-resolved

1. Add the following line to `/etc/systemd/system/kubelet.service.d/10-kubeadm.conf`.

```
Environment="KUBELET_RESOLVER_ARGS=--resolv-conf=/run/systemd/resolve/resolv.conf"
```

2. Start kubelet.

```
$ sudo systemctl start kubelet
```

3.2. Worker Nodes

The procedures in this section do not need to be completed if the goal is a single-node cluster.

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 NVIDIA Container Runtime. Follow these instructions 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.

3.2.1. Installing and Running Kubernetes

1. Add the official GPG keys.

```
$ curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo apt-key add -
$ curl -s -L https://nvidia.github.io/kubernetes/gpgkey | sudo apt-key add -
$ curl -s -L https://nvidia.github.io/kubernetes/ubuntu16.04/nvidia-kubernetes.list |\
          sudo tee /etc/apt/sources.list.d/nvidia-kubernetes.list
```

2. Update the package index.

```
$ sudo apt update
```

3. Install packages.

```
$ VERSION=1.10.8+nvidia
$ sudo apt install -y kubectl=${VERSION} kubelet=${VERSION} \
                                kubeadm=${VERSION} helm=${VERSION}
```

4. 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
```

5. Join the worker node to the cluster with a command similar to the following. (The command below is an example that will not work for your installation.)

```
$ sudo kubeadm join --token <token> <master-ip>:<master-port> --discovery-token-ca-cert-hash sha256::<hash> --ignore-preflight-errors=all
```

3.2.2. Check Your Cluster State

Run the following command on the Master Node and make sure your GPU worker nodes appear and their state transitions to Healthy. It may take a few minutes for the status to change.

```
$ kubectl describe nodes
```

3.3. Run GPU Tasks

3.3.1. Enable GPUs

As part of installing Kubernetes on the worker nodes, GPUs are enabled by default. It may take up to a minute or two for GPUs to be enabled on your cluster (i.e. for Kubernetes to download and run containers).

Once Kubernetes has downloaded and run containers, 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
```

3.3.2. Run a GPU Workload

Make sure the GPU support has been properly set up by running a simple CUDA container. We provided one in the artifacts you downloaded (you'll need to have a GPU with at least 8GB). There are also other examples available in the examples directory.

1. Start the CUDA sample workload.

```
$ kubectl create -f /etc/kubeadm/examples/pod.yml
```

2. When the pod is running, you can execute the nvidia-smi command inside the container.

```
$ 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.
 ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute
 M. |
|===============================+======================
+======================|
|   0  Tesla V100-SXM2...  On   | 00000000:00:1E.0 Off |
 0 |
| N/A   34C    P0    20W / 300W |     10MiB / 16152MiB |      0%
 Default |
+-------------------------------+----------------------
+----------------------+

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

Chapter 4.
CLUSTER CUSTOMIZATION

After starting up your cluster, it is setup with a few basic utilities:

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

Edit **`/etc/kubeadm/post-install.sh`** to change some or all of these auto-deployed utilities.

The Flannel Network Plugin

Kubernetes clusters need a pod network addon installed. Flannel is deployed by default, for multiple reasons:

- Recommended by Kubernetes
- Used in production clusters
- Integrates well with the CRI-O runtime

For more information and other networking options refer to: https://kubernetes.io/pod-network.

Helm

Helm is automatically installed and deployed with Kubernetes on NVIDIA GPUs. Helm helps you manage Kubernetes applications, through helm charts it allows you to define, install, and upgrade even the most complex Kubernetes application. Charts are packages of pre-configured Kubernetes resources.

For more information on helm refer to: https://github.com/helm/helm.

Monitoring Stack

An integrated monitoring stack is deployed by default in your cluster to monitor health and get metrics from GPUs in Kubernetes.

The stack is deployed using helm (the charts can be found at /etc/kubeadm/monitoring) and uses the NVIDIA Datacenter GPU Manager (DCGM), Prometheus (using Prometheus Operator), and Grafana for visualizing the various metrics.

You can change some of the configuration in the values file of the chart:

/etc/kubeadm/monitoring/kube-prometheus/charts/exporter-node/values.yml

Tainting the Master Node (Optional)

If you are setting up a multi-node cluster and you do not want jobs to run on the master node (to avoid impact on control plane performance), set the master Kubernetes node to deny pods that can't run on the master node:

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

Chapter 5.
USING DISTRIBUTION-SPECIFIC FEATURES

Kubernetes on NVIDIA GPUs include the following features that are not yet available in the upstream release of Kubernetes:

- GPU attributes exposed in a node
- Scheduling improvements
- GPU monitoring
- Support for the CRI-O runtime preview feature

5.1. Exposing GPU Attributes In a Node

Nodes now expose the attributes of your GPUs. This can be inspected by querying the Kubernetes API at the node endpoint. The GPUs attributes currently advertised are:

- GPU memory
- GPU ECC
- GPU compute capabilities

Inspect GPU attributes in a node with the following command:

```
$ kubectl proxy --port=8000 &
$ curl -s http://localhost:8000/api/v1/nodes | grep -B 7 -A 3 gpu-memory
```

5.2. Scheduling GPUs By Attribute

Pods can now specify device selectors based on the attributes that are advertised on the node. These can be specified at the container level. For example:

```
apiVersion: v1
kind: Pod
metadata:
  name: gpu-pod
spec:
  containers:
    - name: cuda-container
      image: nvidia/cuda:9.0-base
```

```
    command: ["sleep"]
    args: ["100000"]
    computeResourceRequests: ["nvidia-gpu"]
computeResources:
  - name: "nvidia-gpu"
    resources:
      limits:
        nvidia.com/gpu: 1
    affinity:
      required:
        - key: "nvidia.com/gpu-memory"
          operator: "Gt"
          values: ["8000"] # change value to appropriate mem for GPU
```

1. Create the pod and check its status.

```
$ kubectl create -f /etc/kubeadm/examples/pod.yml
```

2. List the pods running in the cluster.

```
$ kubectl get pods
```

3. Run the nvidia-smi command inside the container.

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

5.3. Setting Up Monitoring

To set up monitoring, follow these steps.

1. Label the GPU nodes.

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

2. Ensure that the label has been applied.

```
$ kubectl get nodes --show-labels
```

3. Install the monitoring charts. (This step is performed automatically at the end of installation.)

```
$ helm install /etc/kubeadm/monitoring/prometheus-operator-0.0.15.tgz --name
 prometheus-operator --namespace monitoring
$ helm install /etc/kubeadm/monitoring/kube-prometheus-0.0.43.tgz --name
 kube-prometheus --namespace monitoring
```

4. Check the status of the pods. It may take a few minutes for the components to initalize and start running.

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

```
prometheus-operator-789d76f7b9-t5qqz                    1/1       Running   0
          1h
```

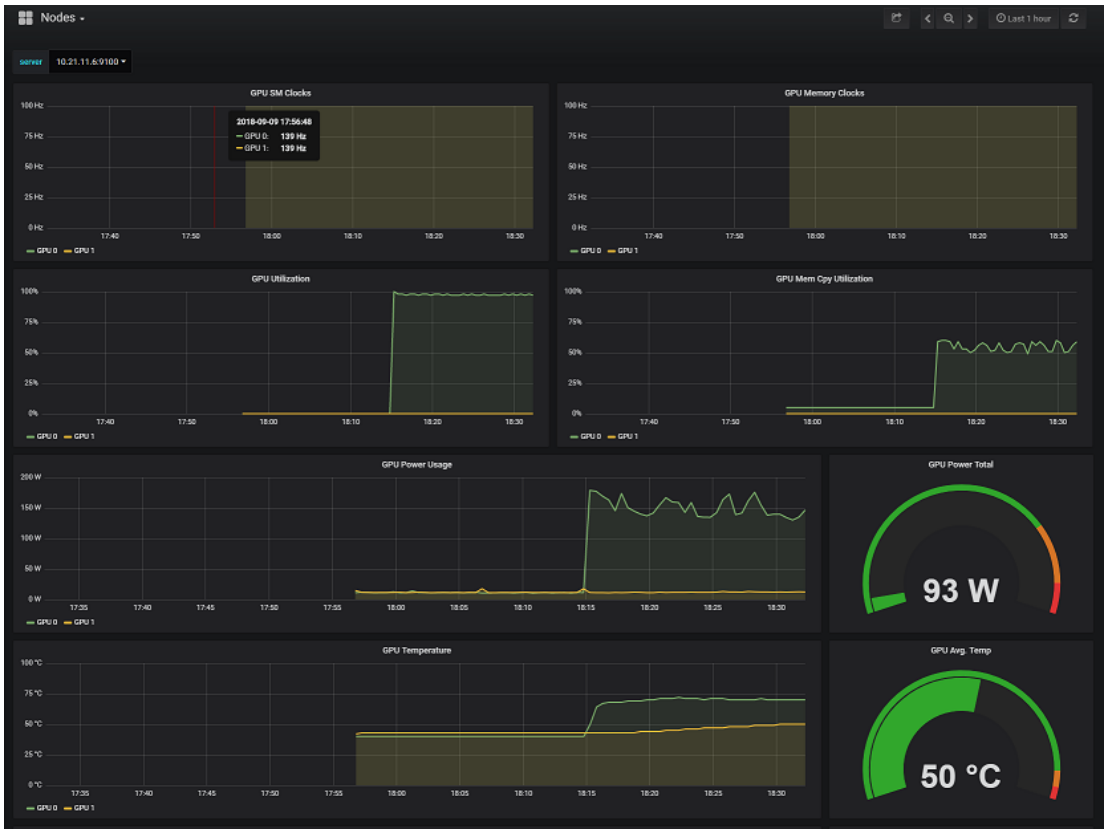
5. Forward the port for Grafana.

```
$ kubectl -n monitoring port-forward $(kubectl get pods -n monitoring -
lapp=kube-prometheus-grafana -ojsonpath='{range .items[*]}{.metadata.name}
{"\n"}{end}') 3000 &
```

6. Open a browser window and type `http://localhost:3000` to view the Nodes Dashboard in Grafana. To view the 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/resnet.yml
```

The GPU metrics are reported on the Grafana Nodes dashboard:



If necessary destroy pods in the monitoring namespace with the following command:

```
$ helm del --purge <chart-name>
```

If necessary, uninstall helm from the cluster with the following command:

```
$ helm reset --force
```

5.4. CRI-O Runtime Preview Feature Support

CRI-O is a lightweight container runtime for Kubernetes as an alternative to Docker. It is an implementation of the Kubernetes CRI (Container Runtime Interface) to enable using OCI (Open Container Initiative) compatible runtimes. For more information on CRI-O, visit the CRI-O website.

5.4.1. Install CRI-O

1. Add the repository.

```
$ sudo add-apt-repository -y ppa:alexlarsson/flatpak
```

2. Update the apt package index.

```
$ sudo apt-get update
```

3. Install the dependencies.

```
$ sudo apt-get install -y btrfs-tools git libassuan-dev libdevmapper-dev \
    libglib2.0-dev libc6-dev libgpgme11-dev libgpg-error-dev libseccomp-dev
 \
    libselinux1-dev pkg-config go-md2man runc libostree-dev libapparmor-dev
```

4. Install and setup Golang and setup your PATH environment variables for the current user and the root user.

```
$ wget https://dl.google.com/go/go1.9.3.linux-amd64.tar.gz
$ sudo tar -C /usr/local -xzf go1.9.3.linux-amd64.tar.gz

$ export GOPATH=~/go
$ export PATH=/usr/local/go/bin:/usr/bin:$GOPATH/bin:$PATH

$ echo 'GOPATH=~/go' >> ~/.bashrc
$ echo 'PATH=/usr/local/go/bin:/usr/bin:$GOPATH/bin:$PATH' >> ~/.bashrc

$ echo "GOPATH=$HOME/go" | sudo tee --append /root/.bashrc
$ echo 'PATH=/usr/local/go/bin:/usr/bin:$GOPATH/bin:$PATH' | sudo tee --
append /root/.bashrc
```

5. Install the CRI-O tools.

```
$ git clone https://github.com/projectatomic/skopeo $GOPATH/src/github.com/
projectatomic/skopeo
$ cd $GOPATH/src/github.com/projectatomic/skopeo && make binary-local
```

6. Install and setup CRI-O.

```
$ mkdir -p $GOPATH/src/github.com/kubernetes-incubator && cd $_
$ git clone https://github.com/kubernetes-incubator/cri-tools && cd cri-
tools
$ git checkout v1.0.0-alpha.0 && make
$ sudo cp ~/go/bin/crictl /usr/bin/crictl

$ git clone https://github.com/kubernetes-incubator/cri-o && cd cri-o
$ git checkout release-1.9
$ make install.tools && make && sudo make install
```

7. Set up `runc` container runtime.

```
$ wget https://github.com/opencontainers/runc/releases/download/v1.0.0-rc4/
runc.amd64
$ chmod +x runc.amd64
$ sudo mv runc.amd64 /usr/bin/runc
```

8. Set up the different policies and hooks.

```
$ sudo mkdir -p /etc/containers /etc/crio /etc/cni/net.d/
$ sudo mkdir -p /usr/share/containers/oci/hooks.d /etc/containers/oci/
hooks.d/

$ sudo cp seccomp.json /etc/crio/seccomp.json
$ sudo cp test/policy.json /etc/containers/policy.json
$ sudo cp contrib/cni/* /etc/cni/net.d/
$ sudo cp /etc/kubeadm/crio/nvidia.json /etc/containers/oci/hooks.d/
$ sudo cp /etc/kubeadm/crio/nvidia-crio-hook.sh /usr/bin/nvidia-crio-hook.sh
```

5.4.2. Run the CRI-O Service

1. Create a systemd service for CRI-O.

```
$ sudo sh -c 'echo "[Unit]
Description=OCI-based implementation of Kubernetes Container Runtime
 Interface
Documentation=https://github.com/kubernetes-incubator/cri-o

[Service]
ExecStart=/usr/local/bin/crio --log-level debug
Restart=on-failure
RestartSec=5

[Install]
WantedBy=multi-user.target" > /etc/systemd/system/crio.service'
```

2. Start the systemd service.

```
$ sudo systemctl enable crio
$ sudo systemctl start crio
```

3. Insure that the daemon is running.

```
$ sudo crictl --runtime-endpoint /var/run/crio/crio.sock info
```

Output is similar to the following:

```
{
  "status": {
    "conditions": [
      {
        "type": "RuntimeReady",
        "status": true,
        "reason": "",
        "message": ""
      },
      {
        "type": "NetworkReady",
        "status": true,
        "reason": "",
        "message": ""
      }
    ]
  }
}
```

5.4.3. Configure the Kubelet to Use CRI-O

1. Replace the Kubelet systemd Unit file at /etc/systemd/system/kubelet.service.d/10-kubeadm.conf The new version should have the following contents:

```
[Service]
Wants=docker.socket crio.service

Environment="KUBELET_KUBECONFIG_ARGS=--bootstrap-kubeconfig=/etc/kubernetes/
bootstrap-kubelet.conf --kubeconfig=/etc/kubernetes/kubelet.conf"
Environment="KUBELET_SYSTEM_PODS_ARGS=--pod-manifest-path=/etc/kubernetes/
manifests --allow-privileged=true"
Environment="KUBELET_NETWORK_ARGS=--network-plugin=cni --cni-conf-dir=/etc/
cni/net.d --cni-bin-dir=/opt/cni/bin"
Environment="KUBELET_DNS_ARGS=--cluster-dns=10.96.0.10 --cluster-
domain=cluster.local"
Environment="KUBELET_AUTHZ_ARGS=--authorization-mode=Webhook --client-ca-
file=/etc/kubernetes/pki/ca.crt"
Environment="KUBELET_CADVISOR_ARGS=--cadvisor-port=0"
Environment="KUBELET_CERTIFICATE_ARGS=--rotate-certificates=true --cert-
dir=/var/lib/kubelet/pki"

Environment="KUBELET_CRIO_ARGS=--container-runtime=remote --container-
runtime-endpoint=/var/run/crio/crio.sock --runtime-request-timeout=10m"

ExecStart=
ExecStart=/usr/bin/kubelet $KUBELET_KUBECONFIG_ARGS $KUBELET_EXTRA_ARGS
 $KUBELET_SYSTEM_PODS_ARGS $KUBELET_NETWORK_ARGS $KUBELET_DNS_ARGS
 $KUBELET_AUTHZ_ARGS $KUBELET_CADVISOR_ARGS $KUBELET_CERTIFICATE_ARGS
 $KUBELET_EXTRA_ARGS $KUBELET_CRIO_ARGS
```

2. Reload and restart Kubelet.

```
$ sudo systemctl daemon-reload
$ sudo systemctl restart kubelet
```

3. Join your cluster with the Kubeadm join command.

```
$ sudo kubeadm join --token <token> <master-ip>:<master-port> --discovery-
token-ca-cert-hash sha256::<hash> --skip-preflight-checks
```

4. Alternatively, you can also initialize the cluster using Kubeadm.

```
$ sudo kubeadm init --ignore-preflight-errors=all --config /etc/kubeadm/
config.yml
$ mkdir -p $HOME/.kube
$ sudo cp /etc/kubernetes/admin.conf $HOME/.kube/config
$ sudo chown -R $USER:$USER $HOME/.kube/

$ kubectl apply -f https://raw.githubusercontent.com/coreos/flannel/v0.9.1/
Documentation/kube-flannel.yml
```

5. If using a single node cluster, you may need to untaint the master.

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

6. Check if all the control plane components are running on the master.

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

5.4.4. Run a GPU Task

Make sure the CRI-O runtime is properly set up by running the simple CUDA container provided in the kubeadm package at `/etc/kubeadm/examples` (Requires a GPU with at least 8GB.)

1. Start the CUDA sample workload.

```
$ kubectl create -f /etc/kubeadm/examples/pod.yml
```

2. When the pod is running, execute the `nvidia-smi` command inside the container.

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

If the CRI-O runtime is properly set up, output is similar to the following.

```
+-----------------------------------------------------------------------------
+
| NVIDIA-SMI 384.125                Driver Version: 384.125
  |
|-------------------------------+----------------------
+----------------------+
| 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 V100-SXM2...  On   | 00000000:00:1E.0 Off |
 0 |
| N/A   34C    P0    20W / 300W |     10MiB / 16152MiB |      0%
 Default |
+-------------------------------+----------------------
+----------------------+

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

Chapter 6. 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
$ kubectl cluster-info dump
```

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

```
$ sudo kubeadm reset
$ sudo systemctl start kubelet
$ sudo kubeadm init --ignore-preflight-errors=all --config /etc/kubeadm/
config.yml
```

6.1. 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
```

6.2. 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 (It might be necessary to start kubelet after restarting Kubeadm.)

```
$ kubeadm reset
```

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:

```
$ 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:

```
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.

6.3. Monitoring Issues

This section describes common monitoring issues and workarounds.

Common Grafana Errors

Check if the Grafana pod is deployed and running:

```
$ 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:

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

Appendix A. DGX SYSTEMS

This section details the prerequisites for setting up a Kubernetes node. And list the steps to follow in order to correctly setup your nodes.

The prerequisites include:

- The worker nodes must be provisioned with the NVIDIA driver. The recommended way is to use a package manager and install the `cuda-drivers` package (or equivalent). If no packages are available, use an official download from the NVIDIA Driver Downloads website.
- 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 these instructions 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, 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 way to run GPU containers. It is more stable than nvidia-docker 1.0 and required for use of the Device Plugin feature in Kubernetes.

A.2.1. Uninstalling Old Versions

1. List the volumes.

```
$ 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.

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

A.2.2. Setting Up the Repository

1. Add the official GPG keys.

```
$ curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey |sudo apt-key add
 -
$ curl -s -L https://nvidia.github.io/nvidia-docker/ubuntu16.04/amd64/
nvidia-docker.list | sudo tee /etc/apt/sources.list.d/nvidia-docker.list
```

2. Update the package index.

```
$ sudo apt update
```

A.2.3. Install NVIDIA Container Runtime

1. Install the `nvidia-docker2` package.

```
$ sudo apt-get install -y nvidia-docker2
```

2. Restart the Docker daemon.

```
$ sudo pkill -SIGHUP dockerd
```

3. Check that nvidia-docker2 is properly installed.

```
$ sudo docker info | grep nvidia
Runtimes: runc nvidia
```

4. Check that the new runtime is functional.

```
$ docker run -it --rm --runtime nvidia nvidia/cuda nvidia-smi
+-----------------------------------------------------------------------------
+
| NVIDIA-SMI 384.125                Driver Version: 384.125
  |
|-------------------------------+----------------------
+----------------------+
| 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 V100-SXM2...  On   | 00000000:00:1E.0 Off |
 0 |
| N/A   34C    P0    20W / 300W |     10MiB / 16152MiB |      0%
 Default |
+-------------------------------+----------------------
+----------------------+

+-----------------------------------------------------------------------------
+
| Processes:                                                       GPU
 Memory |
|  GPU       PID   Type   Process name                             Usage
  |
```

```
|
=============================================================================|
|  No running processes found
  |
+-----------------------------------------------------------------------------
+
```

Appendix B. UPGRADING THE CLUSTER

B.1. Upgrading the Cluster from 1.9.7 to 1.9.10

Before upgrading using the procedures in this section, make sure your cluster is running version 1.9.7, and make sure swap is disabled.

The **`kubeadm upgrade`** command does not impact workloads, only Kubernetes-internal components. Always backup any application-level state, such as a database an application might depend on (like MySQL or MongoDB) before upgrading.

All containers are restarted after the upgrade, due to changes in container spec hash value.

Only one minor version upgrade is supported. For example, you can only upgrade from 1.8 to 1.9, not from 1.7 to 1.9.

B.1.1. Upgrading the Control Plane

1. Download version 1.9.10 of kubeadm using curl.

```
$ wget https://github.com/nvidia/kubernetes/raw/gh-pages/ubuntu16.04/amd64/
kubeadm_1.9.10%2Bnvidia.bz2
$ tar -xvjf ./kubeadm_1.9.10+nvidia.bz2
$ chmod a+rx ./kubeadm_1.9.10+nvidia
$ sudo mv ./kubeadm_1.9.10+nvidia /usr/bin/kubeadm
```

2. Verify the download of kubeadm works and is the expected version.

```
$ kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"9", GitVersion:"v1.9.10",
GitCommit:"9a668c2cd86617b1fd6097a65e6c08a3ad68939e", GitTreeState:"clean",
BuildDate:"2018-08-27T15:30:21Z", GoVersion:"go1.9.3", Compiler:"gc",
 Platform:"linux/amd64"}
```

3. Upgrade to the new version.

```
$ sudo kubeadm upgrade apply ${VERSION}
[preflight] Running pre-flight checks.
[upgrade] Making sure the cluster is healthy:
```

```
[upgrade/config] Making sure the configuration is correct:
[upgrade/config] Reading configuration from the cluster...
[upgrade/config] FYI: You can look at this config file with 'kubectl -n
 kube-system get cm kubeadm-config -oyaml'
...
[addons] Applied essential addon: kube-dns
[addons] Applied essential addon: kube-proxy
[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.9.10". Enjoy!

[upgrade/kubelet] Now that your control plane is upgraded, please proceed
 with upgrading your kubelets in turn.
```

B.1.2. Finalizing the Upgrade

You'll need to log into every node, drain them of existing workloads and apt update and upgrade all the KONG packages. On the master node, remove the old device plugin and create the new device plugin. You can use **`kubectl drain`** to safely evict all of your pods from a node before you perform maintenance on the node (e.g. kernel upgrade, hardware maintenance, etc.) Safe evictions allow pod containers to gracefully terminate. They also respect PodDisruptionBudgets you have specified.

1. Identify the name of the node you wish to drain using **`kubectl get nodes`**, then drain the node.

```
$ kubectl drain $HOST_NAME --ignore-daemonsets
```

2. Update the node.

```
$ sudo apt update
$ $ VERSION=1.9.10
$ sudo apt install -y kubectl=${VERSION}+nvidia \
                      kubelet=${VERSION}+nvidia \
                      kubeadm=${VERSION}+nvidia \
                      helm=${VERSION}+nvidia
```

3. Replace the device plugin 1.9.7 device plugin with the 1.9.10 device plugin. If you intend to have multiple KONG nodes running different versions for a long time, label the nodes and edit the device plugins to use these labels:

```
$ kubectl delete -f /etc/kubeadm/device-plugin/nvidia-1.9.7.yml
$ kubectl create -f /etc/kubeadm/device-plugin/nvidia-${VERSION}.yml
```

4. Mark the node as schedulable with the following command:

```
$ kubectl uncordon $HOST_NAME
```

B.2. Upgrading the Cluster from 1.9.10 to 1.10.8

Before upgrading using the procedures in this section, make sure your cluster is running version 1.9.10, and make sure swap is disabled.

The **kubeadm upgrade** command does not impact workloads, only Kubernetes-internal components. Always backup any application-level state, such as a database an application might depend on (like MySQL or MongoDB) before upgrading.

All containers are restarted after the upgrade, due to changes in container spec hash value.

Only one minor version upgrade is supported. For example, you can only upgrade from 1.9 to 1.10, not from 1.8 to 1.10.

B.2.1. Upgrading the Control Plane

1. Download version 1.10.8 of kubeadm using curl.

```
$ wget https://github.com/nvidia/kubernetes/raw/gh-pages/ubuntu16.04/amd64/
kubeadm_1.10.8%2Bnvidia.bz2
$ tar -xvjf ./kubeadm_1.10.8+nvidia.bz2
$ chmod a+rx ./kubeadm_1.10.8+nvidia
$ sudo mv ./kubeadm_1.10.8+nvidia /usr/bin/kubeadm
```

2. Verify the download of kubeadm works and is the expected version.

```
$ kubeadm version
kubeadm version: &version.Info{Major:"1", Minor:"10", GitVersion:"v1.10.8",
GitCommit:"9a668c2cd86617b1fd6097a65e6c08a3ad68939e", GitTreeState:"clean",
BuildDate:"2018-08-27T15:30:21Z", GoVersion:"go1.9.3", Compiler:"gc",
 Platform:"linux/amd64"}
```

3. Upgrade to the new version.

```
$ sudo kubeadm upgrade apply ${VERSION}
[preflight] Running pre-flight checks.
[upgrade] Making sure the cluster is healthy:
[upgrade/config] Making sure the configuration is correct:
[upgrade/config] Reading configuration from the cluster...
[upgrade/config] FYI: You can look at this config file with 'kubectl -n
 kube-system get cm kubeadm-config -oyaml'
...
[addons] Applied essential addon: kube-dns
[addons] Applied essential addon: kube-proxy
[upgrade/successful] SUCCESS! Your cluster was upgraded to "v1.10.8". Enjoy!

[upgrade/kubelet] Now that your control plane is upgraded, please proceed
 with upgrading your kubelets in turn.
```

B.2.2. Finalizing the Upgrade

You'll need to log into every node, drain them of existing workloads and apt update and upgrade all the KONG packages. On the master node, remove the old device plugin and create the new device plugin. You can use **kubectl drain** to safely evict all of your pods from a node before you perform maintenance on the node (e.g. kernel upgrade, hardware maintenance, etc.) Safe evictions allow pod containers to gracefully terminate. They also respect PodDisruptionBudgets you have specified.

1. Identify the name of the node you wish to drain using **kubectl get nodes**, then drain the node.

```
$ kubectl drain $HOST_NAME --ignore-daemonsets
```

2. Update the node.

```
$ sudo apt update
$ $ VERSION=1.10.8
$ sudo apt install -y kubectl=${VERSION}+nvidia \
                      kubelet=${VERSION}+nvidia \
                      kubeadm=${VERSION}+nvidia \
                      helm=${VERSION}+nvidia
```

3. Mark the node as schedulable with the following command:

```
$ kubectl uncordon $HOST_NAME
```

Appendix C. SUPPORT

For more information, see:

- Kubernetes Tutorials
- Using kubeadm to create a cluster
- Kubernetes

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