Running ML/DL Workloads Using Red Hat OpenShift Container Platform v3.11

Accelerate your ML/DL Projects Platform using Kubeflow and Nvidia GPUs

September 2019

H17913.1

White Paper

Abstract

This white paper describes how to deploy Kubeflow v0.5 on Red Hat OpenShift Container Platform and provides recommendations for achieving optimal performance for ML/DL workloads using the latest NVIDIA Tesla GPUs.

Dell EMC Solutions

Copyright

The information in this publication is provided as is. Dell Inc. makes no representations or warranties of any kind with respect to the information in this publication, and specifically disclaims implied warranties of merchantability or fitness for a particular purpose.

Use, copying, and distribution of any software described in this publication requires an applicable software license.

Copyright © 2019 Dell Inc. or its subsidiaries. All Rights Reserved. Dell Technologies, Dell, EMC, Dell EMC and other trademarks are trademarks of Dell Inc. or its subsidiaries. Intel, the Intel logo, the Intel Inside logo and Xeon are trademarks of Intel Corporation in the U.S. and/or other countries. Other trademarks may be trademarks of their respective owners. Published in the USA September 2019 White Paper H17913.1.

Dell Inc. believes the information in this document is accurate as of its publication date. The information is subject to change without notice.

Contents

Executive summary

Business case

Deep learning (DL) has demonstrated success in many application domains, including computer vision, speech recognition, and natural language processing. Despite the widespread adoption of DL, model development, training, and management at scale still pose significant engineering challenges. Enterprises are investing in custom infrastructure platforms to support their Artificial Intelligence (AI) use cases and the computing needs of their data science teams, often using ad hoc hardware implementations that are outside mainstream data center systems infrastructure. The ability to integrate production-grade, experimental AI technologies in well-defined platforms facilitates wider adoption.

Kubeflow

Kubeflow is an open-source Kubernetes-native platform for Machine Learning (ML) workloads that enables enterprises to accelerate their ML/DL projects on Kubernetes. Kubeflow is a composable, scalable, portable ML stack that includes components and contributions from a variety of sources and organizations. Its differentiation is using automation to integrate ML tools so that they work together to create a cohesive pipeline and make it easy to deploy ML application lifecycle at scale. For more information, see Kubeflow: The Machine Learning Toolkit for Kubernetes.

Solution overview

Kubeflow requires a Kubernetes environment such as Google Kubernetes Engine or Red Hat OpenShift Container Platform. Dell EMC and Red Hat offer a proven platform design that provides accelerated delivery of stateless and stateful cloud-native applications using enterprise-grade Kubernetes container orchestration. Dell EMC uses this enterprise-ready platform as the foundation for building a robust, high-performance ML/DL platform that supports various lifecycle stages of an AI project: model development using Jupyter Notebooks, rapid iteration and testing using TensorFlow, training DL models using graphics processing units (GPUs), and prediction using developed models.

Dell EMC provides validated design guidance to help customers rapidly implement OpenShift Container Platform on Dell EMC infrastructure. For more information, see the *Dell EMC Ready Architecture for Red Hat OpenShift Container Platform v3.11 Architecture Guide*. This document assists you in making OpenShift infrastructure design decisions and in selecting server configurations to handle your application workloads.

Running Kubeflow on OpenShift offers several advantages in an ML/DL context:

- Running ML/DL workloads in the same environment as the rest of the company's application reduces IT complexity.
- Using Kubernetes as the underlying platform makes it easier for an ML/DL engineer to develop a model locally using a development system such as a laptop before deploying the application to a production Kubernetes environment.

Document purpose

This white paper describes how to deploy Kubeflow v0.5 on Red Hat OpenShift Container Platform v3.11 using Nvidia Tesla GPUs to achieve a high-performance AI environment in which ML/DL scientists can work without having to build a complete platform from scratch.

Audience

This white paper is for IT machine learning specialists, administrators, and decision makers who intend to to build an ML platform using on-premises infrastructure. Familiarity with ML processes and OpenShift technology is desirable but not essential.

We value your feedback

Dell EMC and the authors of this document welcome your feedback on the solution and the solution documentation. Contact the Dell EMC Solutions team by email or provide your comments by completing our documentation survey. Alternatively, contact the Dell EMC OpenShift team at openshift@dell.com.

Authors
Dell EMC: Ramesh Radhakrishnan, John Terpstra, Michael Tondee, Aighne Kearney
Red Hat: Jacob Liberman, Pete MacKinnon

Note: The OpenShift Container Platform Info Hub for Ready Solutions space on the Dell EMC Communities website provides links to additional, relevant documentation.

Solution architecture

As shown in Figure 1, the Dell EMC reference architecture for OpenShift Container Platform on Dell EMC infrastructure uses five node types: bastion, master, infrastructure, application, and storage.

- **Bastion node**—The bastion node serves as the main deployment and management server for the OpenShift cluster.
- **Master nodes**—The master nodes perform control functions for the entire cluster environment. These nodes are responsible for the creation, scheduling, and management of all objects specific to OpenShift, including the API, controller management, and scheduler capabilities.
- **Infrastructure nodes**—The infrastructure nodes execute a range of control plane services, including the OpenShift Container registry, the HAProxy router, and the Heketi service.
- **Storage nodes**—The storage nodes provide persistent storage for the environment. Kubernetes storage classes can create persistent volumes manually or automatically. This solution uses the Dell EMC PowerEdge R740 server for storage and the PowerEdge R640 server for the remaining node types.
- **Application nodes**—The application nodes run containerized workloads. The nodes contain a single binary of OpenShift node components and are used by master nodes to schedule and control containers. Kubeflow uses the application node resources for execution of ML/DL jobs, which are among the most computationally intensive workloads in the enterprise data center.

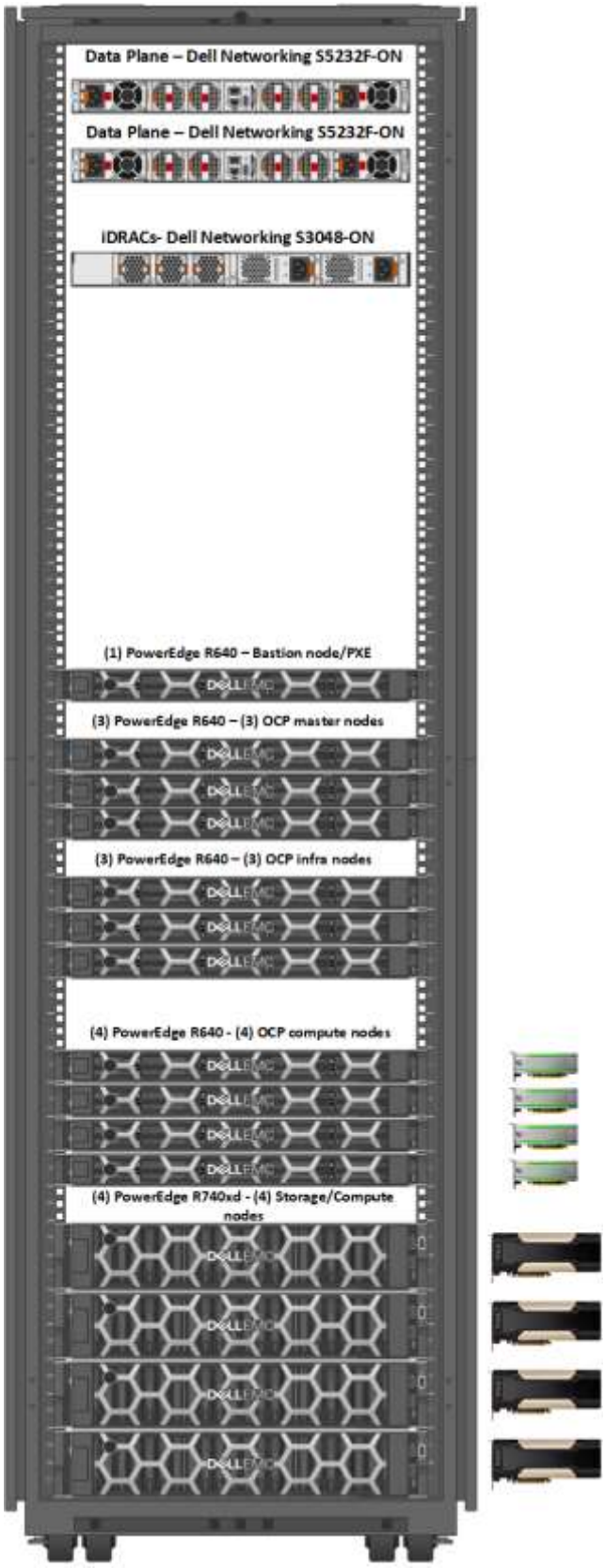

Figure 1. Rack diagram

Application and storage node configuration

Dell EMC engineers configured the application nodes with Nvidia Tesla T4 GPU and the storage nodes with Nvidia Tesla V100 GPU for accelerated computation of complex ML/DL workloads. The Nvidia T4 GPU is based on the new Turing architecture and packaged in an energy-efficient 70-watt, small PCIe form factor. We installed a single Nvidia Tesla T4 GPU in each application node. The Nvidia Tesla T4 GPU is optimized for mainstream computing environments, including D/L training and inference, and features multi-precision Turing Tensor Cores and new RT Cores to deliver up to 65 teraFLOPs of mixed-precision compute power for accelerating ML/DL workloads

The storage nodes were operated in hyperconverged mode. Each storage node was installed with a single Nvidia Tesla V100 GPU. The NVIDIA Tesla V100 GPU accelerators offer up to 112 teraFLOPs of mixed-precision compute capability in a single GPU, enabling data scientists, researchers, and engineers to tackle new challenges.

Machine learning workflow

Dozens or hundreds of iterations are produced as the models are tuned and new datasets are incorporated. Using automation to manage, build, and maintain the stages in a complex ML life cycle reduces the number of steps that must be performed manually, accelerating the ML process and minimizing mistakes.

A typical ML workflow includes the following steps: data acquisition, data analysis, data preparation, data validation, model building, model training, model validation, training at scale, model inference, and monitoring. The following diagram shows an example:

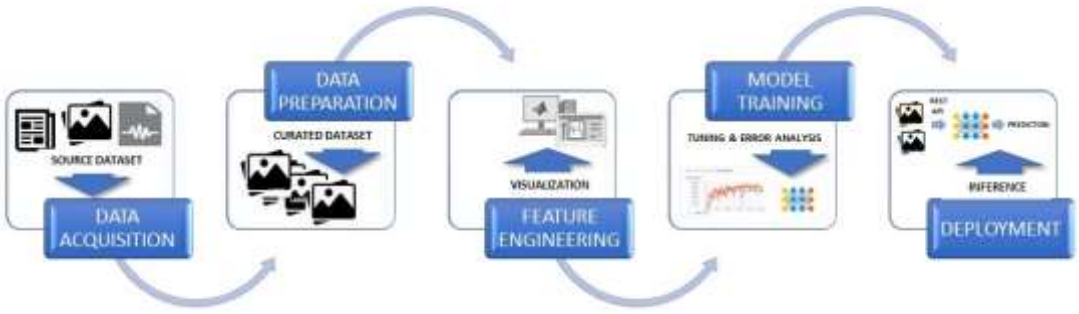

Figure 2. Machine learning workflow

Kubeflow supports the different lifecycle stages of an ML project, integrating commonly used ML tools such as TensorFlow and Jupyter notebooks into a single platform.

GPU accelerated TensorFlow training using TFJobs

Introduction

Model training is the most computationally intensive part of ML/DL. Kubeflow uses TFJobs, a Kubernetes custom resource, to run TensorFlow training jobs in an automated fashion and enable data scientists to monitor job progress by viewing the results. Nvidia GPUs are used for accelerating the training of neural network models and training.

Execution time can also be reduced by running TensorFlow distributed training, which takes advantage of the compute capability of multiple GPUs to work on the same neural network training. Multiple components play a role to enable distributed training: worker nodes, where the computation (model training) takes place, and Parameter Server (PS), which is responsible for storing the parameters needed by the individual workers.

Kubeflow provides a YAML representation for TFJobs. For more information, see TensorFlow Training (TFJob).

Model training example

To show the capabilities of the Kubeflow platform in executing ML/DL training jobs as well as the scaling efficiencies of the OpenShift Container Platform, we ran the TensorFlow CNN benchmark to train the Resnet50 model, as shown in the following figure:

Figure 3. Training example: Resnet50 model

We ran the TensorFlow CNN benchmark using TFJobs, a Kubeflow interface to perform TENSORFLOW training and monitor the training runs. Figure 3 shows the performance of the training jobs using a throughput metric (images/sec). The performance results for the ResNet-50 benchmark is plotted for both NVIDIA Tesla V100 and T4 GPUs. As expected, using more GPUs to train the model results in a higher performance. We used the Horovod library developed by Uber to scale the training job for performing multi-node distributed training. The Tesla V100 GPU job executes approximately 5 times more images/sec than the Tesla T4 GPU, even though their theoretical TFLOPs delta is two times greater.

The Tesla V100 GPU uses the faster HBM2 memory, which has a significant impact on DL training performance. The Tesla V100 GPU model comes at a higher power and price point compared to the Tesla T4. We showcase a flexible environment where users can populate either the Tesla T4, the Tesla V100, or both GPUs on the OpenShift Container Platform and make it available to ML engineers through Kubeflow. The choice and number of GPUs will depend on the workload requirements and price targets for the ML/DL environment.

The following table shows the YAML file we used to deploy the TFJob on four T4 GPUs:

Table 1. YAML file for TFJob deployment

tf_nvidia_cnn.yml

```
apiVersion: kubeflow.org/v1beta2
kind: TFJob
metadata:
  labels:
    experiment: experiment
  name: nvidiatfjob
  namespace: default
spec:
  tfReplicaSpecs:
    Ps:
      nodeSelector:
         nvidia: t4
      replicas: 1
      template:
        metadata:
          creationTimestamp: null
        spec:
          imagePullPolicy: Always
          nodeSelector:
            nvidia: t4
          containers:
          - args:
            - python
            -
/opt/benchmarks/tf_cnn_benchmarks.py
            - --batch_size=256
            - --model=resnet50
            - --num_batches=100
            - --num_gpus=1
            - --
variable_update=horovod
            - --use_fp16=True
            - --xla=True
            image:
nvcr.io/nvidia/tensorflow:19.06-py3
            name: tensorflow
            ports:
            - containerPort: 2222
              name: tfjob-port
            resources:
              limits:
                nvidia.com/gpu: 1
            workingDir:
/home/benchmarks/tf_cnn_benchmarks.py
          restartPolicy: OnFailure
Worker:
      imagePullPolicy: Always
      nodeSelector:
         intelrole: worker
      replicas: 4
      template:
        metadata:
          creationTimestamp: null
        spec:
          nodeSelector:
            nvidia: t4
          containers:
          - args:
            - python
            -
/opt/benchmarks/tf_cnn_benchmarks.py
            - --batch_size=256
            - --model=resnet50
            - --num_batches=100
            - --num_gpus=1
            - --
variable_update=horovod
            - --use_fp16=True
            - --xla=True
            image: nvcr.io/nvidia
/tensorflow:19.06-py3
            name: tensorflow
            ports:
            - containerPort: 2222
              name: tfjob-port
            resources:
              limits:
                nvidia.com/gpu: 1
            workingDir:
/home/benchmarks/tf_cnn_benchmarks.py
          restartPolicy: OnFailure
```

Installing the GPU device plug-in in OpenShift Container Platform 3.11

Overview

This section describes the procedures for enabling containerized GPU workloads in OpenShift:

- Installing the NVIDIA drivers
- Installing the GPU device plug-in

Using GPUs with OpenShift requires that NVIDIA drivers for Red Hat Enterprise Linux are installed on the host. This white paper assumes that GPUs are present in all the storage nodes and application nodes. To prepare the host nodes, it is necessary to install NVIDIA drivers and add the container-runtime hook on each one.

Typographical conventions

Installation instructions use certain typographical conventions to designate commands and screen output. Command syntax is identified by `Courier` font. Screen output is presented in **`bold type`**.

Installing the NVIDIA drivers

NVIDIA drivers are compiled from source. To complete the build process:

1. Install the `kernel-devel` package by running the following command:

   ```
   yum -y install kernel-devel-`uname -r`
   ```

2. The Nvidia-driver package requires the DKMS package. To install the DKMS package from the EPEL repository:

   a. Install the epel repository by running the following command:

   ```
   yum install -y
   https://dl.fedoraproject.org/pub/epel/epel-release-
   latest-7.noarch.rpm
   ```

   b. Locate the newest NVIDIA drivers in the following repository:

   ```
   yum install -y
   https://developer.download.nvidia.com/compute/cuda/repos
   /rhel7/x86_64/cuda-repo-rhel7-10.1.105-1.x86_64.rpm
   ```

3. To install the `nvidia-kmod` package, which includes the NVIDIA kernel modules, run the following command:

   ```
   yum -y install nvidia-driver nvidia-driver-cuda nvidia-
   modprobe
   ```

4. Remove the nouveau kernel module to ensure that the NVIDIA kernel module loads:

   ```
   modprobe -r nouveau
   ```

   Installing the NVIDIA driver package blacklists the driver in the kernel command line `nouveau.modeset=0 rd.driver.blacklist=nouveau video=vesa:off`. This is done so that the nouveau driver is not loaded on

subsequent reboots.

5. Load the NVIDIA and the unified memory kernel modules by running the following command:

```
nvidia-modprobe && nvidia-modprobe -u
```

6. Verify that the installation and the drivers are working by running the following command:

```
nvidia-smi --query-gpu=gpu_name --format=csv,noheader --id=0
| sed -e 's/ /-/g'
```

This command outputs the name of the GPU on the server - `Tesla-V100-SXM2-32GB` in this example. This name can be used to label the node in OpenShift.

Steps 1 to 6 of this procedure show installation of the NVIDIA GPU driver from source. At the time of writing of this paper, NVIDIA and Red Hat announced a technical preview of new packages for GPU drivers for select Red Hat Enterprise Linux versions. These packages eliminate the need to have compilers and a full software development toolchain installed on each system that is running NVIDIA GPUs, simplifying the management experience for the user. To get started with the new packages, follow the instructions in this README.

Add the nvidia-container-runtime-hook

The version of Docker that is shipped by Red Hat includes support for OCI runtime hooks, Therefore, we need to install only the `nvidia-container-runtime-hook` package.

1. Install `libnvidia-container` and the `nvidia-container-runtime` repository by running the following command:

```
curl -s -L https://nvidia.github.io/nvidia-container-
runtime/centos7/nvidia-container-runtime.r
epo | tee /etc/yum.repos.d/nvidia-container-runtime.repo
```

An OCI prestart hook makes NVIDIA libraries and binaries available in a container by bind-mounting them in from the host. The prestart hook is triggered by the presence of certain environment variables in the container: `NVIDIA_DRIVER_CAPABILITES=compute,utility`.

2. Install an OCI prestart hook by running the following command:

```
yum -y install nvidia-container-runtime-hook
```

3. Set the config/activation files for `docker/podman/cri-o` on all the nodes with GPUs (in our installation, the storage and application nodes) by running the following command:

```
cat<<'EOF' >> /usr/share/containers/oci/hooks.d/oci-nvidia-
hook.json
{
"hasbindmounts": true,
```

```
"hook": "/usr/bin/nvidia-container-runtime-hook",
"stage": [ "prestart" ]
}
EOF
```

Add the SELinux policy module

An SELinux policy tailored for running CUDA GPU workloads is required to run NVIDIA containers that are contained and not privileged.

Install the SELinux policy module on all GPU worker nodes by running the following command:

```
wget https://raw.githubusercontent.com/zvonkok/origin-ci-gpu/master/selinux/nvidia-container.pp
semodule -i nvidia-container.pp
```

Check and restore the labels on the node

The new SELinux policy relies on correct labeling of the host. Ensure that the files that are needed have the correct SELinux label by running the following commands:

1. Restorecon all files that the prestart hook will need:

   ```
   nvidia-container-cli -k list | restorecon -v -f -
   ```

2. Restorecon all devices that are accessed:

   ```
   restorecon -Rv /dev
   ```

3. Restorecon all files that the device plug-in will need:

   ```
   restorecon -Rv /var/lib/kubelet
   ```

   The system is now set up to run a GPU-enabled container.

Verify SELinux and prestart hook functionality

To verify correct operation of the driver and container enablement, run a `cuda-vector-add container` with Docker or Podman as follows:

```
docker run --user 1000:1000 --security-opt=no-new-privileges
--cap-drop=ALL \
--security-opt label=type:nvidia_container_t \
docker.io/mirrorgooglecontainers/cuda-vector-add:v0.1
Trying to pull repository
docker.io/mirrorgooglecontainers/cuda-vector-add ...
v0.1: Pulling from docker.io/mirrorgooglecontainers/cuda-
vector-add
5d9a20cbabf3: Pull complete
84b2e9f421b6: Pull complete
6f94649104a2: Pull complete
6c16e819a84a: Pull complete
9822cda4c699: Pull complete
1bc138ea32ad: Pull complete
ade909bfe2a5: Pull complete
```

```
e70e5ba470d6: Pull complete
ab71e6b7eb90: Pull complete
925740434ebd: Pull complete
2f93605342b5: Pull complete
fe61ad4992f7: Pull complete
Digest:
sha256:0705cd690bc0abf54c0f0489d82bb846796586e9d087e9a93b579
4576a456aea
Status: Downloaded newer image for
docker.io/mirrorgooglecontainers/cuda-vector-add:v0.1
[Vector addition of 50000 elements]
Copy input data from the host memory to the CUDA device
CUDA kernel launch with 196 blocks of 256 threads
Copy output data from the CUDA device to the host memory
Test PASSED
Done
```

If the test passes, as indicated by the words `Test PASSED`, the drivers, hooks and container runtime are functioning correctly and we you can proceed to configuring OpenShift Container Platform.

Installing the GPU device plug-in

Install the GPU device plug-in after a successful installation of OpenShift 3.11.

Schedule the device plug-in on nodes that include GPUs

Follow these steps:

1. Label the node by running the following command:

   ```
   oc label node <node-with-gpu> openshift.com/gpu-
   accelerator=true
   ```

   The labels are used in the next stage of the installation.

2. To install the device plug-in on the storage nodes, run the following commands:

   ```
   oc label node stor1.r5a.local openshift.com/gpu-
   accelerator=true
   oc label node stor2.r5a.local openshift.com/gpu-
   accelerator=true
   oc label node stor3.r5a.local openshift.com/gpu-
   accelerator=true
   oc label node stor4.r5a.local openshift.com/gpu-
   accelerator=true
   ```

Deploy the NVIDIA device plug-in daemonset

Follow these steps:

1. Clone the following repository, which contains several yaml files for future use, by running:

   ```
   git clone https://github.com/redhat-performance/openshift-
   psap.git
   ```

```
cd openshift-psap/blog/gpu/device-plugin
```

The sample daemonset `device-plugin/nvidia-device-plugin.yml` uses the label you created in Schedule the device plug-in on nodes that include GPUs so that the plugin pods run only where GPU hardware is available.

2. Create the NVIDIA device plug-in daemonset by running the following command:

```
oc create -f nvidia-device-plugin.yml
```

3. Verify that the device plug-in is working correctly by running the following command:

```
oc get pods -n kube-system
NAME READY STATUS RESTARTS AGE
nvidia-device-plugin-daemonset-czzbs   1/1       Running   0
32s
nvidia-device-plugin-daemonset-hz5kr   1/1       Running   0
32s
nvidia-device-plugin-daemonset-w9bxj   1/1       Running   0
32s
nvidia-device-plugin-daemonset-xql7z   1/1       Running   0
32s
```

Four are running because we labeled four storage nodes in the preceding step.

4. Review the logs by running the following command:

```
oc logs nvidia-device-plugin-daemonset- czzbs -n kube-system
2019/07/12 2:19:45 Loading NVML
2019/07/12 2:19:45 Fetching devices.
2019/07/12 2:19:45 Starting FS watcher.
2019/07/12 2:19:45 Starting OS watcher.
2019/07/12 2:19:45 Starting to serve on
/var/lib/kubelet/device-plugins/nvidia.sock
2019/07/12 2:19:45 Registered device plugin with Kubelet
```

The node advertises the `nvidia.com/gpu` extended resource that is in its capacity:

```
oc describe node stor1.r5a.local openshift.com |
egrep'Capacity|Allocatable|gpu'
Capacity:
nvidia.com/gpu: 1
Allocatable:
nvidia.com/gpu: 1
```

Nodes that do not have GPUs installed do not advertise GPU capacity.

Deploy a pod that requires a GPU

Use the `device-plugin/cuda-vector-add.yaml` as a pod description for running the `cuda-vector-add` image in OpenShift. The last line of the file requests one Nvidia GPU from OpenShift. The OpenShift scheduler sees this and schedules the pod to a node

that has a free GPU. After the `pod create` request arrives at a node, the Kubelet coordinates with the device plug-in to start the pod with a GPU resource.

To run a GPU-enabled container on the cluster:

1. Create a project to group the GPU work by running the following command:

   ```
   oc new-project nvidia
   ```

2. Create and start the pod by running the following command:

   ```
   oc create -f cuda-vector-add.yaml
   ```

   The container finishes and outputs the following:

   ```
   oc get pods
   NAME READY STATUS RESTARTS AGE
   cuda-vector-add 0/1 Completed 0 3s
   nvidia-device-plugin-daemonset- czzbs 1/1 Running 0 9m
   ```

3. Review the logs for any errors by running the following command:

   ```
   oc logs cuda-vector-add
   [Vector addition of 50000 elements]
   Copy input data from the host memory to the CUDA device
   CUDA kernel launch with 196 blocks of 256 threads
   Copy output data from the CUDA device to the host memory
   Test PASSED
   Done
   ```

   This output is the same as when we ran the container directly using Podman or Docker.

4. If you see a “permission denied” error, check that you have the correct SELinux label.

Troubleshooting SELinux

The following table shows the labels of the files that are needed for a working GPU container:

Table 2. File labels for a GPU container

File	SELinux label
/dev/nvidia*	xserver_misc_device_t
/usr/bin/nvidia-*	xserver_exec_t
/var/lib/kubelet/*/*	container_file_t

Installing and deploying Kubeflow

Introduction

This section assumes that you have successfully deployed the OpenShift Container platform following the instructions in the *Dell EMC Ready Architecture for Red Hat OpenShift Container Platform v3.11 Deployment Guide*. As described in the deployment guide, you can run an application such as Nginx to confirm that key cluster operations are working as expected before proceeding to install Kubeflow. This validation step is recommended so that you can be assured that the platform on which you are deploying Kubeflow has no major issues.

Deploying Kubeflow v0.5

Kubeflow v0.5 installation is based on `ksonnet`, a configurable, typed, templating system for the Kubernetes application developer. Ksonnet separates Kubernetes object definitions from the cluster destination to simplify and automate deployments.

To deploy Kubeflow v0.5 on OpenShift Container Platform v3.11:

1. Download `ksonnet` and `kfctl` and add them to your path by running the following command:

```
mkdir ~/bin
cd ~/bin
$ wget
https://github.com/kubeflow/kubeflow/releases/download/v0.5.
1/kfctl_v0.5.1_linux.tar.gz
tar -xzf kfctl_v0.5.1_linux.tar.gz
wget
https://github.com/ksonnet/ksonnet/releases/download/v0.13.1
/ks_0.13.1_linux_amd64.tar.gz
tar -xzf ks_0.13.1_linux_amd64.tar.gz
ln -s ks_0.13.1_linux_amd64/ks ks
export PATH=$PATH:~/bin
cd ~
```

2. Export a value for KFAPP by running the following command:

```
export KFAPP=kubeflow
kfctl init ${KFAPP}
cd ${KFAPP}
kfctl generate all -V
```

Note: KFAPP serves as both the name of the directory where the deployment will be stored and the project/namespace where Kubeflow will be installed. The name “kubeflow” is recommended because some of the `ksonnet` parameters are still hard-coded with `kubeflow` as the project name.

3. Deploy Kubeflow by running the following command:

```
kfctl apply all -V
```

4. Add `ambassador`, `default`, and `katib` to the `anyuid` security context by running the following command:

```
oc adm policy add-scc-to-user anyuid -z ambassador
oc adm policy add-scc-to-user anyuid -z default
oc adm policy add-scc-to-user anyuid -z katib-ui
```

Note: Relaxing the Security Context Constraints (SCC) is required to get Kubeflow services up and running, but is not recommended for a production environment. We expect OpenShift Container Platform to add proper SCCs for these users in the future.

5. Set the ambassador service to deploy as ClusterIP (the default is NodePort) by running the following command:

```
ks param set ambassador ambassadorServiceType 'ClusterIP'
oc expose service ambassador
oc get routes
NAME           HOST/PORT
PATH      SERVICES       PORT         TERMINATION    WILDCARD
ambassador     ambassador-
kubeflow.router.default.svc.cluster.local
ambassador     ambassador                 None
```

6. To verify the Kubeflow installation, enter the URL that was exposed by the route, as displayed by the ambassador service.

   The web interface for Kubeflow appears, as shown in the following figure:

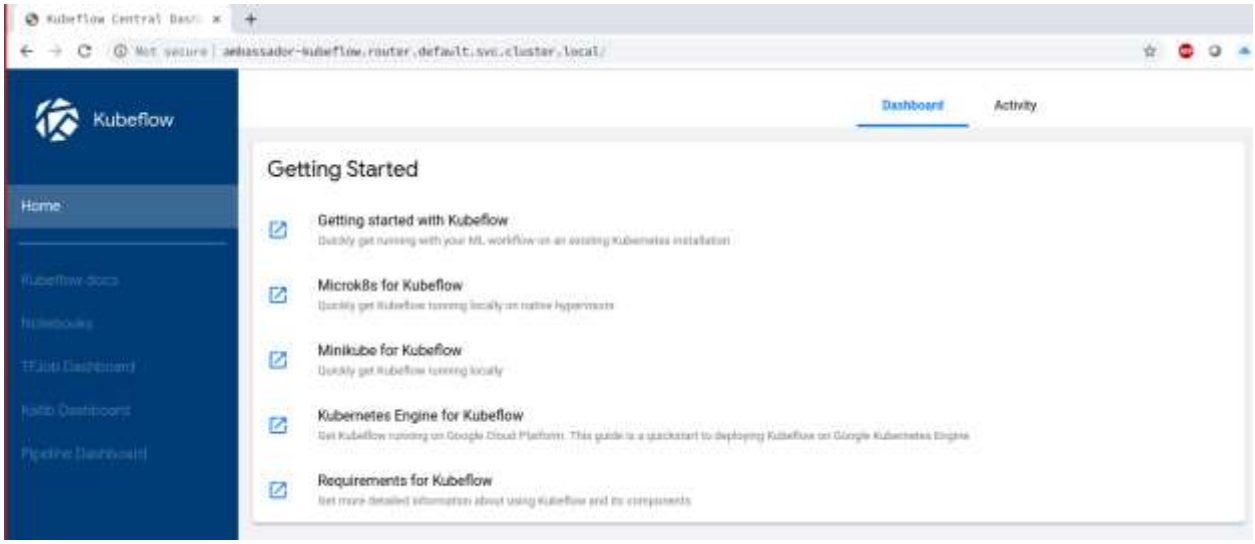

Figure 4. Kubeflow web interface

You can access the different Kubeflow components and documentation through this interface.

Summary

This paper describes how to deploy Nvidia Tesla GPUs for use with Kubeflow on OpenShift Container Platform. By integrating Nvidia GPUs on dedicated application worker nodes, and in converged mode on storage nodes, we have demonstrated flexible configurations that you can use to create a high-performance compute environment that can meet your ML/DL needs. This paper also documents the ability to configure an OpenShift Container Platform with a mix of Nvidia GPUs, thus extending the scope and capability of the ML/DL work profiles that can be processed in a single environment.

The development of a DL model is a computationally intensive operation. In most situations, the full learning process may require training of neural networks with millions of parameters. The learning process can severely tax a nonaccelerated compute platform. Nvidia GPUs are designed to runs 1,000s of threads, exploiting parallelisms that are available in ML/DL workloads and enabling higher levels of productivity for organizations using Kubeflow to develop ML/DL applications. This document demonstrates how Nvidia GPUs can be added to your OpenShift Container Platform and extend its processing capacity for execution of ML/DL workloads using Kubeflow.

References

The following documentation provides additional information about the solution.

Dell EMC documentation

Access to these documents depends on your login credentials. If you do not have access to a document, contact your Dell EMC representative.

- OpenShift Container Platform Info Hub for Ready Solutions
- *Dell EMC Ready Architecture for Red Hat OpenShift Container Platform v3.11 Architecture Guide*
- *Dell EMC Ready Architecture for Red Hat OpenShift Container Platform v3.11 Deployment Guide*

Red Hat documentation

For additional information from Red Hat, see:

- *Red Hat OpenShift Container Platform*
- *Red Hat OpenShift Container Storage*