Machine Learning Using the Dell EMC Ready Architecture for Red Hat OpenShift Container Platform

Streamline your Machine Learning Projects on OpenShift Container Platform v3.11 using Kubeflow

December 2020

H17870.3

White Paper

Abstract

This white paper describes how to deploy Kubeflow v0.5 on Red Hat OpenShift Container Platform using the Dell EMC Ready Architecture and provides recommendations for achieving optimal performance using the latest Intel Xeon Scalable processors.

Dell Technologies Solutions

Copyright

Contents

Executive summary 4

Solution technology 6

Machine learning with Kubeflow 8

Installing and deploying Kubeflow 9

Using Kubeflow 11

Summary 17

References 18

Executive summary

Business case

Enterprises are increasing their investments in infrastructure platforms to support Artificial Intelligence (AI) use cases and the computing needs of their data science teams. Machine Learning (ML) and Deep Learning (DL) are AI techniques that have demonstrated success across every industry vertical, including manufacturing, healthcare, retail, and cloud services.

Kubeflow, a Kubernetes-native platform for ML workloads for enterprises, was released as an open-source project in December 2017. Kubeflow makes it easier to develop, deploy, and manage ML applications. For more information, see Kubeflow: The Machine Learning Toolkit for Kubernetes.

Kubeflow requires a Kubernetes environment such as Google Kubernetes Engine or Red Hat OpenShift. Running Kubeflow on OpenShift offers several advantages in an ML context:

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

Red Hat OpenShift Container Platform

Dell Technologies 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 Technologies provides tested and validated design guidance to help customers rapidly implement OpenShift Container Platform on Dell Technologies infrastructure. For more information, see the *Dell EMC Ready Architecture for Red Hat OpenShift Container Platform v3.11 Architecture Guide*.

Dell Technologies uses this enterprise-ready platform as the foundation for building a robust, high-performance ML/DL platform that supports the various life cycle stages of an AI project: model development that uses Jupyter notebooks, experimentation and testing using TensorFlow, and deployment of ML models.

Document scope

This white paper describes how to install and deploy Kubeflow v0.5 on OpenShift Container Platform and provides an example of running an ML training job on the OpenShift platform using a Kubeflow TFJob to run distributed training.

Audience

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

Authors
Dell EMC: Ramesh Radhakrishnan, John Terpstra
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 technology

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, control-plane, infrastructure, application, and storage.

- **Bastion node**—The bastion node serves as the main deployment and management server for the OpenShift cluster.
- **Control-plane nodes**—The control-plane nodes perform control and management 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. The Dell EMC PowerEdge R740 server is used for storage and the PowerEdge R640 server is used for the remaining node types.
- **Application nodes**—The application nodes run containerized workloads. They contain a single binary of OpenShift node components and are used by control-plane nodes to schedule and control containers. Kubeflow uses the application node resources for running ML/DL jobs.

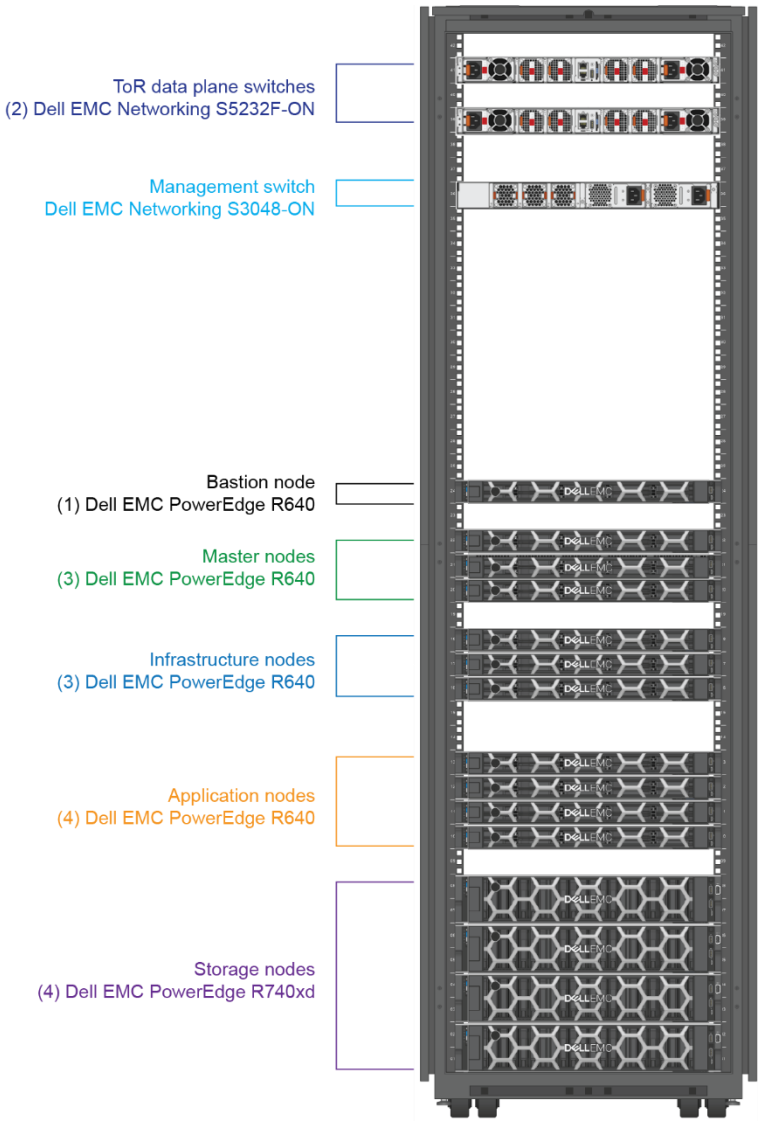

Figure 1. Rack diagram

Configuration for OpenShift Container Platform with Kubeflow

This section describes the modifications to OpenShift Container Platform v3.11 that Dell EMC recommends for ML workloads using Kubeflow.

ML/DL training is one of the most computationally intensive workloads in the enterprise data center. Dell EMC recommends using the latest Intel Xeon Scalable processor family to take advantage of the Intel Deep Learning Boost optimizations that are targeted for complex ML workloads.

The following table shows the recommended configuration for application nodes:

Table 1. Application node configuration

Hardware	Description	SKU
Server	3 x Dell EMC PowerEdge R640	210-AKWU

Hardware	Description	SKU
CPU	2 x Intel Xeon Gold 6248 processor (20 cores, 2.5 GHz, 150W)	338-BRVO
Memory	384 GB (12 x 32 GB 2666MHz DDR4 ECC RDIMM)	370-ADNF
Storage	Capacity Tier: 2 x 1.6 =TB Intel SSD DC P4610 Boot Drive: BOSS controller card + with 2 x M.2 sticks 480 GB	400-BELT 403-BBTJ

Machine learning with Kubeflow

Kubeflow integrates commonly used ML tools such as Tensorflow and Jupyter Notebooks into a single platform. Multiple 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. Kubeflow makes it easier to develop, deploy, and manage portable and scalable ML on Dell EMC platforms and supports the different life cycle stages of an ML project.

ML workflow

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 figure shows a sample workflow:

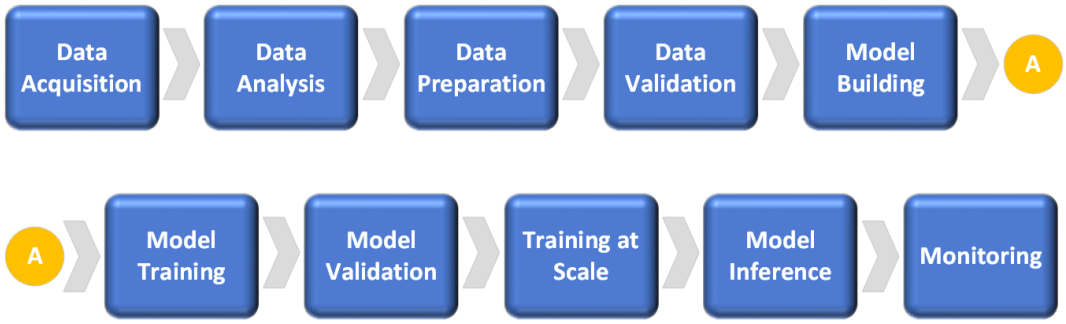

Figure 2. Machine learning workflow

Installing and deploying Kubeflow

Overview

This section describes how to install the Kubeflow ML platform on OpenShift Container Platform and provides steps for launching Jupyter Notebooks in Kubeflow and training at scale using TFJobs.

These procedures assume that you have successfully deployed OpenShift Container Platform following the directions in the *Dell EMC Ready Architecture for Red Hat OpenShift Container Platform v3.11 Deployment Guide*. You can run an application such as Nginx to confirm that key cluster operations are working as expected before proceeding with the Kubeflow installation.

Deploying Kubeflow v0.5

Kubeflow v0.5 installation is based on `ksonnet`, a configurable, typed, templating system for the Kubernetes application developer. The `ksonnet` system 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:

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

Note: KFAPP serves as the name of the directory where the deployment is stored and the project or namespace where Kubeflow is 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:

```
kfctl apply all -V
```

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

```
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:

```
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 Kubeflow web user interface (UI) opens, as shown in the following figure:

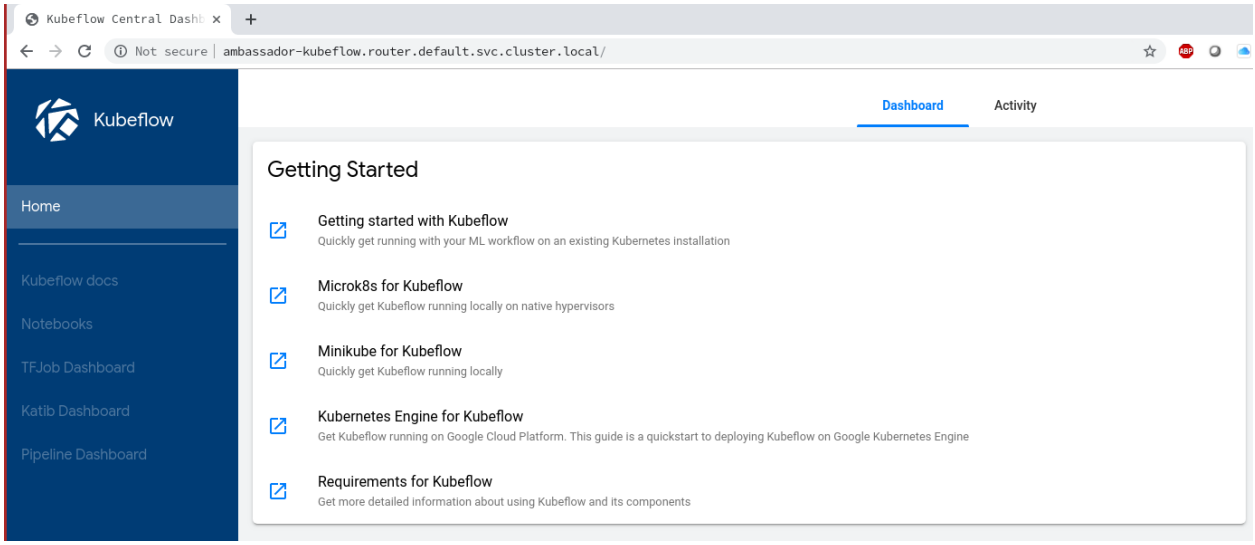

Figure 3. Kubeflow user interface

You can access the Kubeflow components and documentation by using the web UI.

Using Kubeflow

Overview

This section describes how to:

- Launch a Jupyter Notebook using the notebook server after the Kubeflow installation is complete
- Train a TensorFlow model using TFJobs

Launching a Jupyter Notebook

Jupyter is a collaborative tool that data scientists use to develop and execute code, documentation, and visualization in their ML model development process. For more information, see the Jupyter website.

To create and manage Notebook servers in your Kubeflow deployment:

1. In the Kubeflow dashboard, click **Notebooks**.

   A Notebook management window opens, as shown in the following figure:

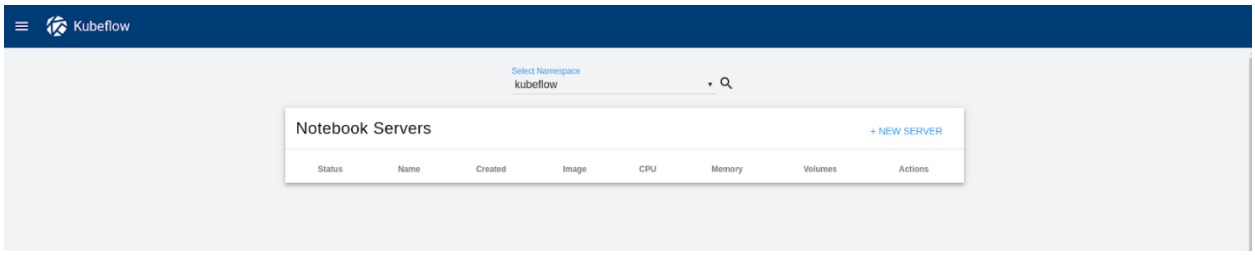

Figure 4. Jupyter Notebook Servers window

2. Click **New Server** in the top right corner of the Notebook Servers pane.

   A menu shows the options that are available for configuring the notebook server:

   - Set a name for the new notebook server.
   - Choose the namespace to which the notebook server will belong.
   - Choose one of the standard TensorFlow notebook server images that come as part of the Kubeflow deployment.

3. After you have chosen your options, click **Spawn** at the bottom of the window.

   A pod is deployed using the TensorFlow container image that you specified. To verify the pod, open the Notebook section of the Kubeflow dashboard, as shown in the following figure:

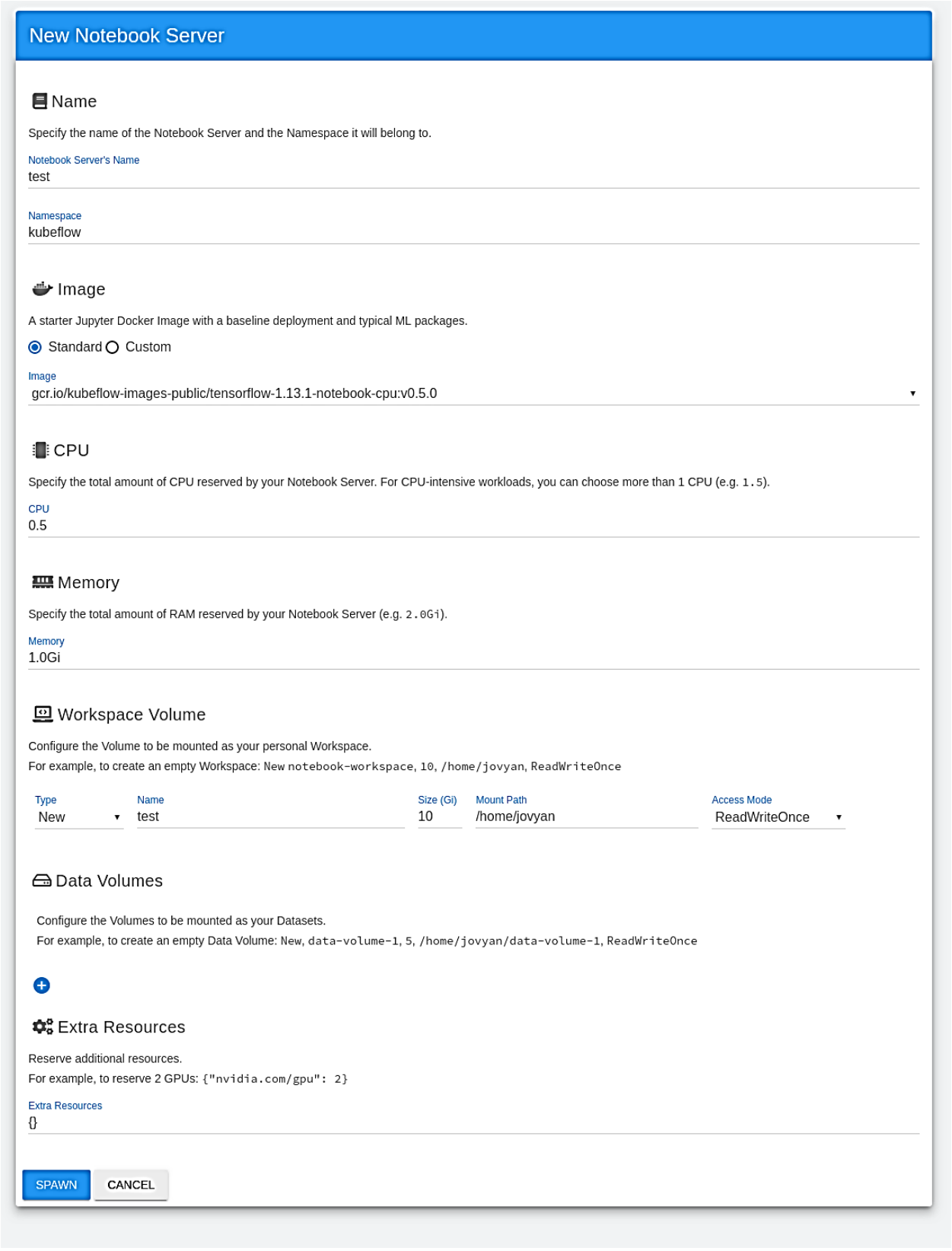

Figure 5. Creating a Jupyter Notebook server

A Notebook server is created using the options you selected, as shown in the following figure:

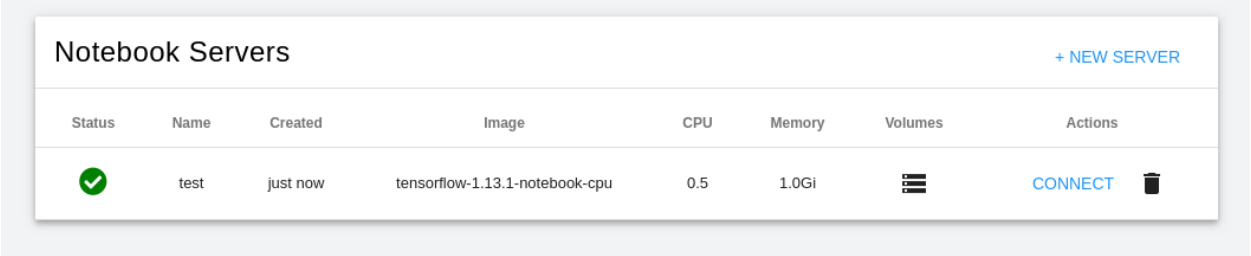

Figure 6. Configured Notebook Server

4. Click **Connect**.

   The Jupyter notebook opens, as shown in the following figure:

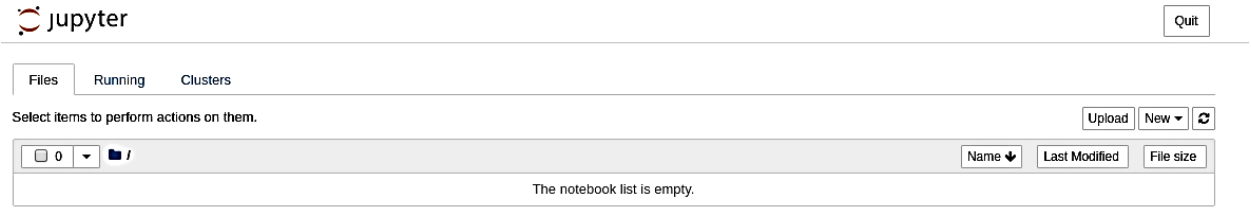

Figure 7. Jupyter notebook

5. To verify that the correct TensorFlow version was installed, click **New**.

   An empty code cell opens in which you can run Python code to display the TensorFlow version that is installed in the notebook. The following figure shows the expected version, v1.13.1:

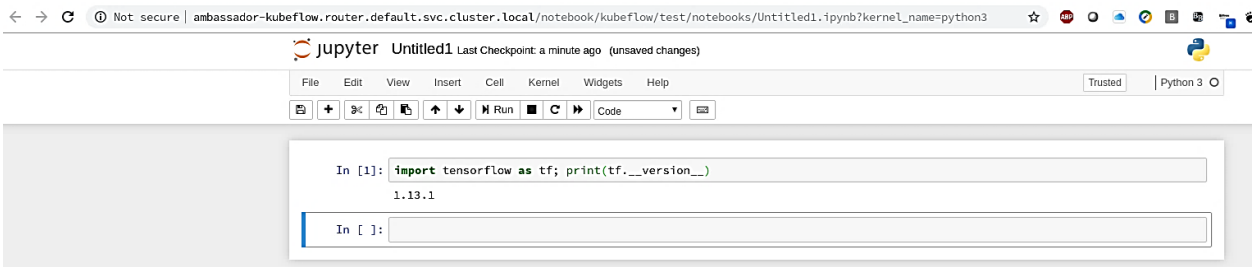

Figure 8. Verifying the TensorFlow installation

Tensorflow training

Another significant state in the ML life cycle is the training of neural network models. This is the most computationally intensive part of ML/DL and therefore data scientists need results as quickly as possible to be productive.

TFJobs

Kubeflow uses a Kubernetes custom resource, TFJobs, to run TensorFlow training jobs in an automated fashion and enable data scientists to monitor job progress. Kubeflow provides a YAML representation for TFJobs. For more information, see TensorFlow Training (TFJob).

To run the training jobs efficiently by using the hardware optimization features that are available in the latest Intel Xeon scalable processor, Dell Technologies recommends using the Intel Optimized Tensorflow ML Framework. The Optimized TensorFlow ML framework uses Intel MKL-DNN library primitives to take advantage of Intel architecture features such as AVX2 for floating-point matrix multiplication and run Tensorflow training jobs on Intel CPUs.

TensorFlow distributed training uses the compute capability of multiple nodes to work on the same neural network training, reducing execution time. Multiple components play a role to enable distributed training. For example, the Parameter Server (PS) stores the parameters that the individual workers need, while worker nodes are responsible for the computation or model training.

Running a TFJob

To demonstrate the scaling efficiencies of the OpenShift Platform and the capabilities of the latest Intel Xeon Scalable processor in running ML/DL training jobs, we ran the commonly used TensorFlow CNN benchmark to train the ResNet50 model.

To deploy the TFJob, we used the following YAML file:

tf_intel_cnn.yaml

```
apiVersion: kubeflow.org/v1beta2
kind: TFJob
metadata:
  labels:
    experiment: experiment
  name: inteltfjob
  namespace: default
spec:
  tfReplicaSpecs:
    Ps:
      nodeSelector:
        intelrole: ps
      replicas: 1
      template:
        metadata:
          creationTimestamp: null
        spec:
          nodeSelector:
            intelrole: ps
          containers:
          - args:
            - nohup
            - unbuffer
            - python
            - /opt/benchmarks/tf_cnn_benchmarks.py
            - --batch_size=256
            - --model=resnet50
    Worker:
      nodeSelector:
        intelrole: worker
      replicas: 2
      template:
        metadata:
          creationTimestamp: null
        spec:
          nodeSelector:
            intelrole: worker
          containers:
          - args:
            - nohup
            - unbuffer
            - python
            - /opt/benchmarks/tf_cnn_benchmarks.py
            - --batch_size=500
            - --model=resnet50
```

	- -- variable_update=parameter_se rver - --mkl=True - -- num_batches=100 - -- num_inter_threads=2 - -- num_intra_threads=40 - -- data_format=NHWC - -- kmp_blocktime=1 - -- local_parameter_device=cpu image: docker.io/intelaipg/intel- optimized-tensorflow name: tensorflow ports: - containerPort: 2222 name: tfjob- port resources: {} workingDir: /home/benchmarks/tf_cnn_benc hmarks.py restartPolicy: OnFailure	- -- variable_update=parameter_se rver - --mkl=True - -- num_batches=100 - -- num_inter_threads=2 - -- num_intra_threads=40 - -- data_format=NHWC - -- kmp_blocktime=1 - -- local_parameter_device=cpu image: docker.io/intelaipg/intel- optimized-tensorflow name: tensorflow ports: - containerPort: 2222 name: tfjob- port resources: {} workingDir: /home/benchmarks/tf_cnn_benc hmarks.py restartPolicy: OnFailure

Note: The container image specified in the file, `docker.io/intelaipg/intel-optimized-tensorflow`, uses the Intel MKL-DNN library to optimize TensorFlow performance on Intel Xeon processors. Parameters recommended by Intel, such as the use of `NUM_INTER_THREADS` and `NUM_INTRA_THREADS` to specify inter- and intra-task parallelism, are passed as arguments to the training run to take advantage of all available cores on the Xeon CPU and maximize parallelism.

1. To register the YAML file, run the following command:

   ```
   $ oc apply -f tf_intel_cnn.yaml
   ```

2. To run the Tensorflow benchmark by executing the TFJob, set the variables by running:

   ```
   $ export KUBEFLOW_TAG=v0.5.0

   $ export KUBEFLOW_SRC=${HOME}/kubeflow-${KUBEFLOW_TAG}

   $ export KFAPP=kf-app

   $ export CNN_JOB_NAME=inteltfjob
   ```

3. Generate the `ks` component for `tf-job-simple-v1beta1` by running:

   ```
   $ ks generate tf-job-simple-v1beta1 ${CNN_JOB_NAME} --
   name=${CNN_JOB_NAME}
   ```

4. Deploy the TFJob to the cluster by running:

   ```
   $ ks apply default -c ${CNN_JOB_NAME}
   ```

5. To view the logs of the running `tfjob`, run:

```
$ oc get -n kubeflow -o yaml tfjobs ${CNN_JOB_NAME}
NAME         AGE
             Inteltfjob  1m
```

6. Verify that the pods for running the benchmark were launched by running:

```
$ oc get pods
NAME                          READY     STATUS    RESTARTS
AGE
inteltfjob-ps-0               0/1       Pending   0          4m
inteltfjob-worker-0           1/1       Running   0          4m
inteltfjob-worker-1           1/1       Running   0          4m
```

After the TFJob launches successfully, it is displayed in the TFJobs dashboard, as shown in the following figure:

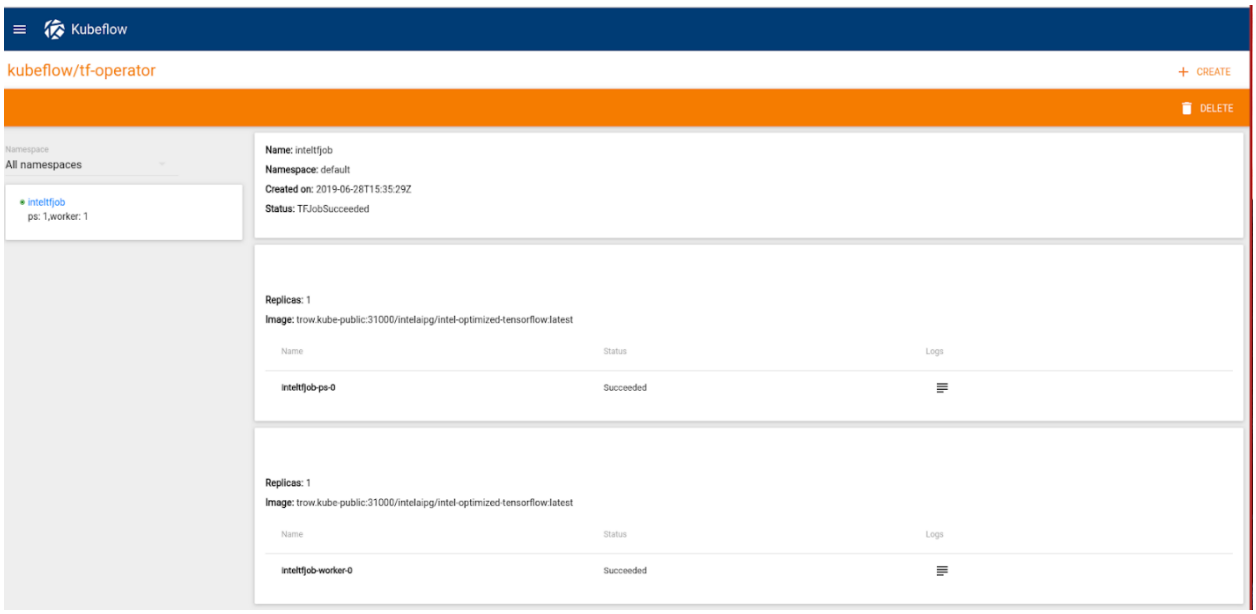

Figure 9. TFJobs dashboard

Distributed versus non-distributed training

Performance comparison

We compared the performance results from running the TensorFlow benchmark on a single application node (non-distributed training) with using training on two application nodes (one PS node and two worker nodes). The following figure shows the results:

Figure 10. Performance scaling for model training using multiple server nodes

The throughput shows that a training job using two application nodes is almost 1.9 times faster than a training job using a single application node. Using the optimized Tensorflow framework distribution from Intel and applying the appropriate parameters when launching a training job enables the ML practitioner to run their Tensorflow jobs efficiently on Intel Xeon Scalable processors.

Summary

Kubeflow enables the efficient running of ML workloads. Kubeflow on OpenShift Container Platform offers several advantages for teams that need an enterprise-ready ML/DL platform:

- Running ML workloads in the same environment as the rest of the organization's applications reduces IT complexity.
- Using Kubernetes as the underlying platform makes it easier for an ML engineer to develop a model locally on their development platform before deploying the application to a production Kubernetes environment such as OpenShift.
- The performance capabilities of the Intel Scalable processors combined with Intel Optimized Tensorflow distribution facilitate higher data scientist productivity by reducing model training times.

References

Dell Technologies documentation

The following Dell Technologies documentation provides additional and relevant information. Access to these documents depends on your login credentials. If you do not have access to a document, contact your Dell Technologies 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

The following documentation provides additional information about the solution.:

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