

Machine Learning Using Dell EMC OpenShift Container Platform

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

August 2019

H17870

White Paper

Abstract

Kubeflow enables efficient execution of Machine Learning (ML) workloads. This white paper describes how to deploy Kubeflow v0.5 on Dell EMC OpenShift Container Platform and provides recommendations for achieving optimal performance using the latest Intel Xeon Scalable processors.

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 August 2019 White Paper H17870.

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.....4
- Solution technology.....6
- Machine learning with Kubeflow8
- Installing and deploying Kubeflow.....9
- Using Kubeflow11
- 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 is a composable, scalable, portable ML stack that was originally based on Google's use of TensorFlow on Kubernetes but now includes components and contributions from several sources and organizations. Eventually, most popular ML tools will become a part of Kubeflow. 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 RedHat 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.

Dell EMC OpenShift Container Platform overview

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 provides tested and 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 RedHat OpenShift Container Platform v3.11 Architecture Guide](#).

Dell EMC 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 using Jupyter notebooks, experimentation and testing using TensorFlow, and deployment of ML models.

Document purpose

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 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: Ramesh Radhakrishnan, John Terpstra, Michael Tondee, Aighne Kearney

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, 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. 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 master nodes to schedule and control containers. Kubeflow uses the application node resources for running ML/DL jobs.

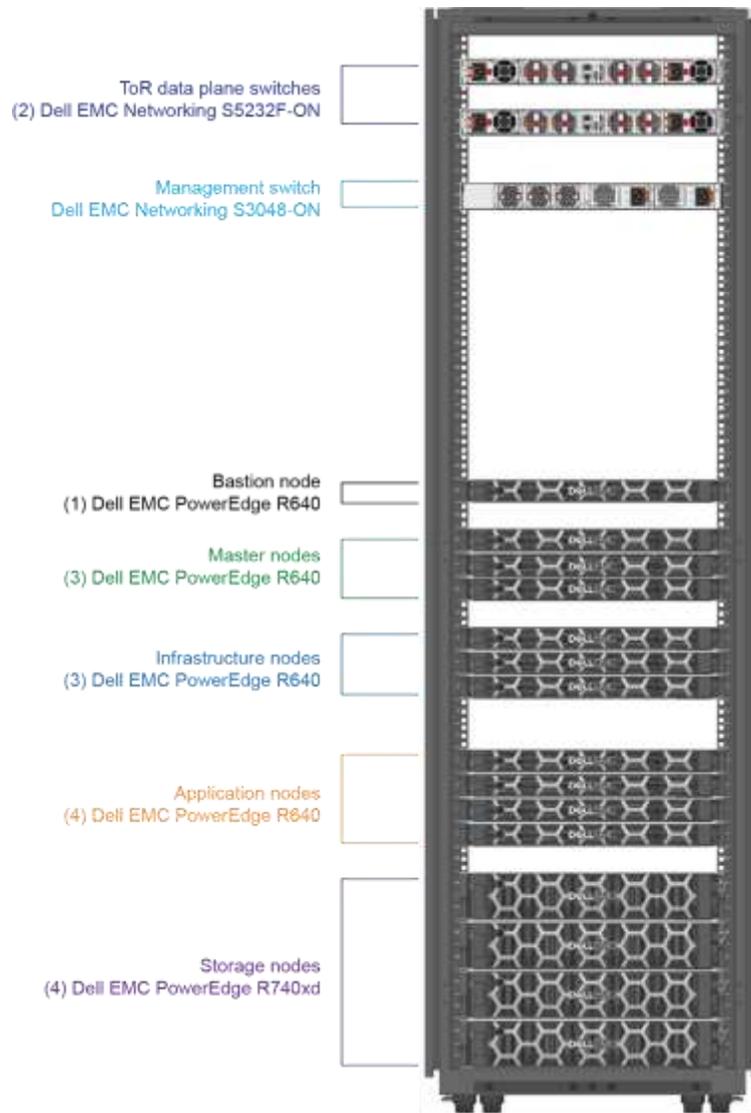


Figure 1. Rack diagram

Recommended configuration for OCP with Kubeflow

Dell EMC recommends the following modifications to OpenShift Container Platform v3.11 to host 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. 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. 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, training at scale, model inference, and monitoring. The following diagram shows an example:



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 provided in the [Dell EMC Ready Architecture for Red Hat OpenShift Container Platform v3.11 Deployment Guide](#). As the deployment guide describes, 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 the following command:

```
$ 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 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 |
| <code>ambassador</code> | <code>ambassador-kubeflow.router.default.svc.cluster.local</code> |
| <code>ambassador</code> | <code>ambassador</code> 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 UI opens, as shown in the following figure:

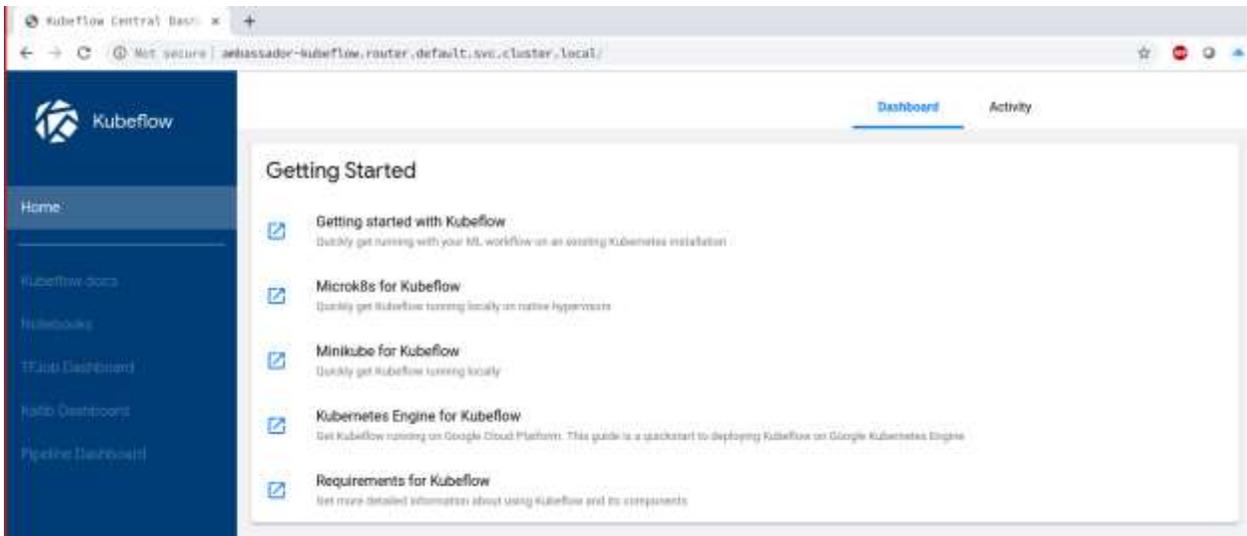


Figure 3. Kubeflow user interface

You can access the Kubeflow components and documentation 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 and how to 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 Figure 4.

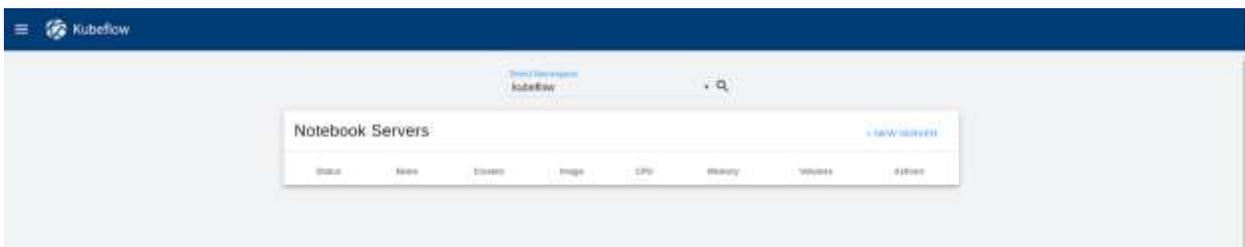


Figure 4. Jupyter Notebook Servers window

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

Figure 5 shows the menu that is now available for configuring the notebook server with various options. The user should at a minimum

- 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, and it can be verified by opening the Notebook section of the Kubeflow dashboard, as shown Figure 6.

New Notebook Server

Name

Specify the name of the Notebook Server and the Namespace it will belong to.

Notebook Server's Name

Namespace

Image

A starter Jupyter Docker Image with a baseline deployment and typical ML packages.

Standard Custom

Image

CPU

Specify the total amount of CPU reserved by your Notebook Server. For CPU-intensive workloads, you can choose more than 1 CPU (e.g. 1.5).

CPU

Memory

Specify the total amount of RAM reserved by your Notebook Server (e.g. 2 - 0G).

Memory

Workspace Volume

Configure the Volume to be mounted as your personal Workspace.

For example, to create an empty Workspace: `New notebook-workspace, 10, /home/jovyan, ReadWriteOnce`

| Type | Name | Size (Gi) | Mount Path | Access Mode |
|------|------|-----------|--------------|---------------|
| New | test | 10 | /home/jovyan | ReadWriteOnce |

Data Volumes

Configure the Volumes to be mounted as your Datasets.

For example, to create an empty Data Volume: `New, data-volume-1, 5, /home/jovyan/data-volume-1, ReadWriteOnce`

+

Extra Resources

Reserve additional resources.

For example, to reserve 2 GPUs: `{\"nvidia.com/gpu\": 2}`

Extra Resources

SPAWN
CANCEL

Figure 5. Creating a Jupyter Notebook Server

A notebook server is created using your selected options, as shown in the following figure:

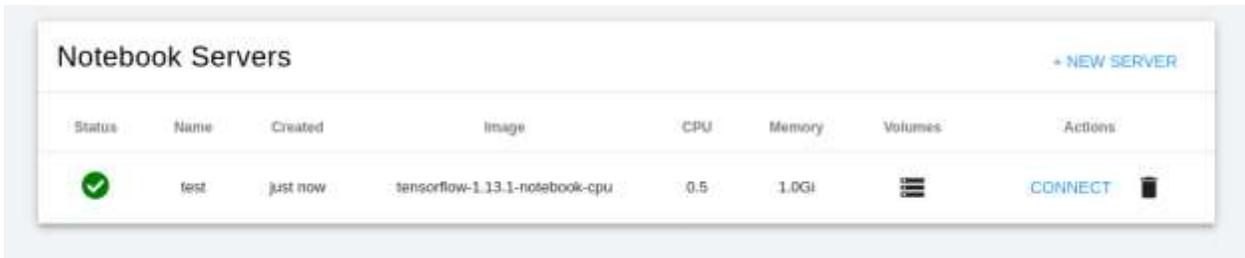


Figure 6. Configured Notebook Server

- Click **Connect**.

The Jupyter notebook opens, as shown in Figure 7.



Figure 7. Jupyter notebook

- 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. Figure 8 shows the expected version, v1.13.1:

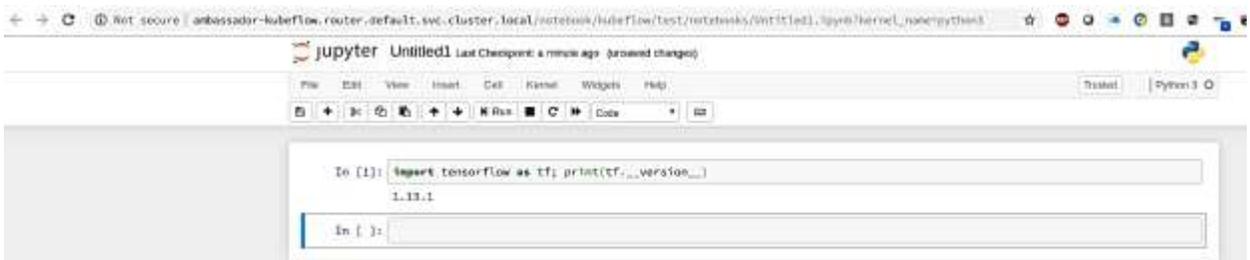


Figure 8. Verifying the TensorFlow installation

TensorFlow training

Another significant state in the ML life cycle is the training of neural network models. Because this is the most computationally intensive part of ML/DL, 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 by viewing the results. Kubeflow provides a YAML representation for TFJobs. For more information, see [TensorFlow Training \(TFJob\)](#).

To run the training jobs efficiently and take advantage of the hardware optimization features that are available in the latest Intel Xeon scalable processor, Dell EMC 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 to 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 capabilities of the latest Intel Xeon Scalable processor in running ML/DL training jobs and the scaling efficiencies of the OpenShift Platform, 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 | | |
|--|--|--|
| <pre> apiVersion: kubeflow.org/v1 beta2 kind: TFJob metadata: labels: experiment: experiment name: inteltfjob namespace: default spec: tfReplicaSpec s: </pre> | <pre> Ps: nodeSelector: intelrole: ps replicas: 1 template: metadata: creationTimestamp: null spec: nodeSelector: intelrole: ps containers: - args: - nohup - unbuffer - python - /opt/benchmarks/tf_cnn_bench marks.py - -- batch_size=256 - -- model=resnet50 </pre> | <pre> Worker: nodeSelector: intelrole: worker replicas: 2 template: metadata: creationTimestamp: null spec: nodeSelector: intelrole: worker containers: - args: - nohup - unbuffer - python - /opt/benchmarks/tf_cnn_bench marks.py - -- batch_size=500 - -- model=resnet50 </pre> |

| | |
|---|---|
| <pre> - -- variable_update=parameter_server - --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_benchmarks.py restartPolicy: OnFailure </pre> | <pre> - -- variable_update=parameter_server - --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_benchmarks.py restartPolicy: OnFailure </pre> |
|---|---|

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 using `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 the following commands:

```
$ 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 the following command:

```
$ ks generate tf-job-simple-v1beta1 ${CNN_JOB_NAME} --
name=${CNN_JOB_NAME}
```
4. Deploy the TFJob to the cluster by running the following command:

```
$ ks apply default -c ${CNN_JOB_NAME}
```
5. To view the logs of the running `tfjob`, run the following command:

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

6. Check that the pods for running the benchmark were launched by running the following command:

```
$ 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:

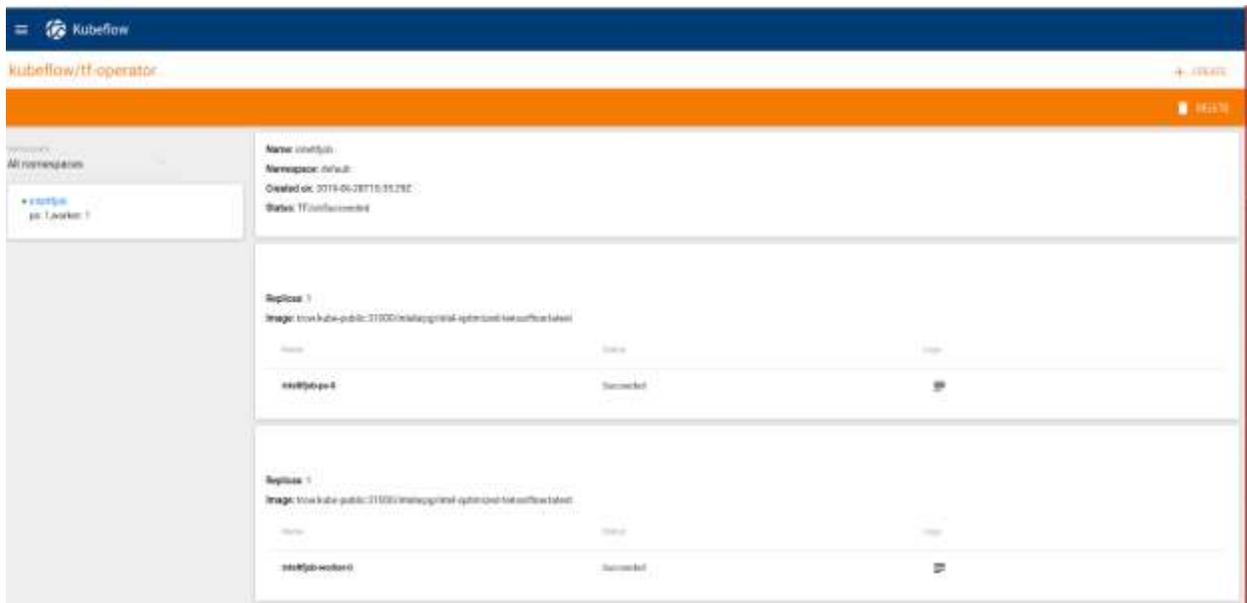


Figure 9. TFJobs dashboard

Comparison of distributed with non-distributed training

Performance comparison

We compared the performance results from executing the TensorFlow benchmark on a single application node (non-distributed training) with execution 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 indicates 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 execute their TensorFlow jobs efficiently on Intel Xeon Scalable processors.

Summary

Kubeflow on OpenShift 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 enable higher data scientist productivity by reducing model training times.

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 RedHat OpenShift Container Platform v3.11 Architecture Guide](#)
- [Dell EMC Ready Architecture for RedHat 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](#)