Getting Started with Kubeflow

Skyler Thomas, Chief Architect for AI/ML Infrastructure
@skythomas
High Level Goals

• Understand the components of Kubeflow....
• Be able to install and Launch Kubeflow
• Walk through various Job Types
• Understand some of the
• Answer your questions...
• WARNING: I am opinionated about Kubeflow and other subjects. Hopefully, this will make our session more fun!
Agenda and Introductions

- Excited to be here!
- Kubeflow Overview
- Brief Intro to Kubernetes
  - The Kubernetes Operator Pattern
- Installing Kubernetes and Kubeflow
  - KSonnet - RIP
  - Kustomize
  - Create a Kubernetes environment
- Jupyter Notebooks
- Pipelines (Argo)
- Training
  - Tensorflow Jobs
  - PyTorch Jobs
  - Caffe Jobs *Time Permitting
  - Hyperparameters and Katib Jobs
- Serving *Time Permitting
  - Tensorflow Serving
  - SeldonCore
- Questions...
Machine Learning Landscape 2013-2014

- Deep Learning was wildly popular due to the recent ImageNet results and Hinton’s recent papers
- No great notebooks existed for data exploration.
  - The notebook frameworks at the time made embedding libraries tricky.
  - Python version woes
  - etc
- Google just donated TensorFlow to community
- Each framework complex to install and get running
  - Apache Mahout, Spark, TensorFlow, Torch
- Friction meant ability to experiment was limited and that was deeply discouraging.
- Forced to own both the complex environmental engineering and the data science.
The 2015-2017 Landscape

- Exploration and training on a single machine now a good experience.
- Maturing of notebook technologies like Zeppelin and Jupyter
- All frameworks including TensorFlow and PyTorch have gotten much better and richer. Creating training jobs now simpler. However, frameworks still painful to deploy.
- Spark grew up in 2.0. Tons of new features. But, was still limited by Yarn or custom rolled environments
- Many components became containerized but orchestration is still a challenge
Where are we now? And what is about to change?

- Containers are now ubiquitous
- Kubernetes became an insanely powerful orchestration engine
- Spark now has native support for Kubernetes
- Kubeflow is about to make deployment of ML environments on Kubernetes simple
- In summary, things are now orders of magnitude nicer for an IT department wanting to support a company’s data scientists.
What makes ML/AI still so challenging?

- Data Scientists are the worst IT customers.
  - Diversified tooling and framework - Important to support many choices
  - Must experiment constantly with new paradigms and engines - Must make experimentation and tool deployment easier.
- Practitioners aren’t savvy on the data logistics - Need to help them deploy end-to-end (E2E) workflows and get models into production.
- 90% of a DataScientist’s time is spent on logistics!!!
- How can I run my training job close to the data I need?
Kubernetes and Data Scientists - Kubeflow

● Most enterprises want to move away from dedicated environments. This means that your data scientists will have to share resources.
  ○ No dedicated Yarn Clusters
  ○ Non toy Kubernetes environments

● If data scientists are sharing with business critical infrastructure, important to ensure they do not consume all resources

● Each data scientist will want to run their own favorite training framework. (Lisa likes PyTorch and Mary likes Tensorflow) Heterogeneity is only way to keep your data scientists happy.

● Run jobs in dedicated namespaces with security, real resource quotas. This ensures your data scientists are good citizens.

● Adds complexity. But this is where technology vendors will play initially with Kubeflow
What is this magical Kubeflow?

- A repository for ML/AI tools. Repository is getting more curated as we move to 1.0. Less of a dumping ground
- A set of common patterns for how to run distributed ML training on Kubernetes
- A common set of tools that work together over full ML lifecycle
Machine Learning lifecycle with Kubeflow

Data Exploration & Visualization

Training Job Coordination (ex. Pipelines)

Model Training

Model Serving

Model Monitoring

Ingress (REST, RPC, etc)

Router

A

B

 Promethues

/code

/training-data

/notbooks

/models
KubeFlow provides a set of components that run natively on Kubernetes
Kubeflow Pods

- Notebooks
  - JupyterHub
  - Jupyter Notebooks
- Training Operators
  - Tensorflow
  - PyTorch
  - Caffe2
  - MPI
  - MXNet
  - Chainer
  - XGBoost
Kubeflow Pods

• Model Serving
  – TensorFlow Serving
  – Istio
  – Seldon
  – TensorFlow Batch

• Misc
  – Ambassador
  – Katib Containers
  – ML Pipeline Containers
  – Argo
  – Minio
  – MySQL
  – Admission Webhook
  – Spartakus
Kubeflow Pods

- UI’s
  - Central Dashboard
  - Argo UI
  - Tensorboard
What is Kubernetes and how does it help us with machine learning workloads?
Philosophy of Pets vs Cattle

- long lived
- name them
- care for them

- ephemeral
- number them
- veterinarians are too expensive
Traditional Distributed Training Architecture (Spark, Tensorflow, etc.)
VM vs Containers

VM

- app
- libs
- os
- hypervisor
- hardware

Container

- app
- libs
- os
- hardware
Container Isolation

- **cgroups**
  - cpu
  - memory
  - network
  - etc.

- **Linux namespaces**
  - pids
  - mnts
  - etc.

- **Chroot (filesystem)**
Container Images
Container Images

![Diagram of Container Images]

- File
- Read-only Layer
- File
- Read-only Layer
Container Images

Writable Layer

Read-only Layer

File

Read-only Layer

File

File
Container = image + isolation

cgroups
- cpu
- memory
- network
- etc.

namespaces
- pids
- mnts
- etc.

File  File  File  Container Image chroot
Pod Isolation

- Pods are the atomic unit of scheduling in K8S
  - Everything in a pod gets scheduled to the same hardware
- A Pod consists of:
  - 1 or more containers
  - YAML file with metadata about the how to run those containers
- Each pod has its own IP address
- Critical to understand the nature of the work of a data scientist. They are called scientists for a reason. The do NOVEL things. A cookie cutter approach will not work for them. This means IT will NOT like their requirements
- Running containers with random python libraries and other executable code
- Better Pod Isolation with projects like gVisor is going to be really helpful for these workloads.
Kubernetes Namespaces & Resource Quotas

- Critical to understand how Kubernetes solves the challenge of multiple users sharing
- Groups a set of resources
  - Allow you to run multiple versions of pods without collisions
  - Provides a way to secure pods
- Namespaces provide overall resource constraints for a set of pods
- Divide your users into projects/tenants/departments via namespaces. Give them BUDGETED resources
- Take advantage of ALL the resource quotas available (CPU, Memory, Ephemeral Storage, Storage, GPU’s, Extended Resources, Number of Pods, PV’s, etc.
- Storage and Compute Scale Separately
- Nodes can be heterogeneous (schedule compute on GPU’s)
KubeFlow workloads are easily deployed into Kubernetes namespaces for security and compliance with resource quotas.
Kubernetes API Server

The API server turns YAML into pods and plans.
Pod YAML

https://github.com/kubernetes/examples/blob/master/staging/podsecuritypolicy/rbac/pod.yaml

```yaml
1 apiVersion: v1
2 kind: Pod
3 metadata:
4   name: nginx
5   labels:
6     name: nginx
7 spec:
8   containers:
9     - name: nginx
10    image: nginx
11   ports:
12     - containerPort: 80
```
The problem of state

- Traditionally treating the container model has one huge problem: State
- Containers can’t move if they have local state
- Data science and ML applications are the WORST applications to think about making stateless
  - We need training data
  - We are creating models
  - We have shared notebooks
- Don’t give up!
- Kubernetes is now MUCH better handling stateful applications
  - StatefulSets and DaemonSets
  - Persistent Volume and PersistentVolumeClaims
  - CSI and FlexVolume plugins
Data in Kubernetes

- Persistent Volumes (PV’s) are the main way to move state out of containers in Kubernetes. (Other methods are possible)
- A pod will reverence a PersistentVolumeClaim in its YAML to indicate it wants mount storage into its containers.
- Kubernetes will dynamically bind a PVC to a PersistentVolume (PV) at runtime
- PV’s use a CSI driver to reference a real filesystem for mounting into the pod.
  - The filesystem must be POSIX. Sorry, no HDFS
  - PV drivers exist for MapR FS (part of the MapR Data Fabric for Kubernetes) and others
  - CSI is GA and supports K8S 1.13+
Kubeflow Aware Storage

- The Hadoop world has been focused on data locality and HDFS.
- The Kubeflow world is different. Uses tons of POSIX and S3/ObjectStorage API’s.
- BOTH are incredibly important. They need to be integrated into K8S. We shouldn’t just do NFS and call it a day.
- If our workloads are going to move across the network, we need to think about how to move data across architectural or geographical divides
- Copying data REALLY sucks…(MapR plug)
- Read/Write access to your datasets is a requirement even if you must copy
- Data security and isolation at the user, team and application levels is big
Exploration/Notebook Phase

- Starting point for Data Exploration
- Started via Launcher
- Runs behind Ambassador Proxy
- Started via Jupyter Hub
- Data Analysis Tools
- Typical Volume Types Used
  - Home, Shared Libraries, Training Data, Shared Notebooks, Models
Training Phase

- Multi-machine training can be started via a call from a notebooks
- Or start externally via job
- Many possible Training Types
- Security Mapping
- Volumes
  - Training Code, Shared Libraries, Models, Training Data, Logs
Serving Phase

- Multiple options for serving models..
- Least baked part of Kubeflow
- Most exposed via Ambassador
- Long Term KFServing simplifies story
- Multiple Choices for TensorFlow model serving including:
  - TensorFlow Serving, Tensorflow Batch, Seldon, NVIDIA TensorRT, Istio Integration
- Prepackaged Seldon Model Servers:
  - SciKitLearn, XGBoost, TensorFlow, MLFlow, PyTorch
- Volumes
  - Models, logs
Creating an ML environment the easy way

<table>
<thead>
<tr>
<th>Procure Environment</th>
<th>Create Storage</th>
<th>Deploy Data Science Environments</th>
</tr>
</thead>
<tbody>
<tr>
<td>We create a place to run our workloads.</td>
<td>ML/AI workloads need a some sort storage fabric</td>
<td>Data scientists will need tools and frameworks to do their jobs.</td>
</tr>
<tr>
<td>We will need a set of machines with Kubernetes installed</td>
<td>● Must be available to all compute/jobs in the cluster</td>
<td>Need things like:</td>
</tr>
<tr>
<td>A. In the cloud, it can take just minutes to do a scripted install</td>
<td>● Data movement is bad. (i.e. copying a terabyte of image files for processing on each node is a non-starter)</td>
<td>● Jupyter Notebooks</td>
</tr>
<tr>
<td>B. On-premises install could take a month to get the hardware</td>
<td></td>
<td>● Training Environments (TensorFlow, etc.)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>● Model Serving</td>
</tr>
<tr>
<td></td>
<td></td>
<td>They want the ability to choose their tools.</td>
</tr>
</tbody>
</table>
Kustomize

https://github.com/kubernetes-sigs/kustomize/tree/master/examples
Installation Steps

1. Create a Kubernetes Environment
   a. Cloud (EKS, GKE, AKS)
   b. Bare Metal (Openshift, Vanilla, etc.)
   c. Laptop (MiniKube)
   d. *Linux Appliance (Multipass)

2. Install Kubeflow (Kustomize)

3. Create Resources
Install Multipass

https://multipass.run/#install

Install Multipass

Windows

Download Multipass for Windows

Windows 10 Pro/Enterprise v 1803 or later
or Windows 10 Home/Pro/Enterprise with Virtualbox

macOS

Download Multipass for MacOS

Yosemite 10.10.3 or later, 2010 or newer Mac

Install the Multipass snap

Linux distributions with snap packages support enabled. Get your Linux set up for snaps
Create Kubeflow VM

multipass launch bionic -n kubeflow -m 8G -d 40G -c 4
Setup Kubernetes

Enter VM:
  multipass shell kubeflow
Install Kubernetes Tools in VM:
  git clone https://github.com/canonical-labs/kubernetes-tools
  sudo kubernetes-tools/setup-microk8s.sh
  sudo kubernetes-tools/expose-dashboard.sh
Add microk8s user privileges and logout:
  sudo usermod -a -G microk8s multipass
  exit
Reenter VM:
  multipass shell kubeflow
Check if K8S installed properly

kubectl get ns
Install Volume Provisioner

```bash
kubectl apply -f https://raw.githubusercontent.com/rancher/local-path-provisioner/master/deploy/local-path-storage.yaml
kubectl patch storageclass microk8s-hostpath -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "false"}}}'
kubectl patch storageclass local-path -p '{"metadata": {"annotations": {"storageclass.kubernetes.io/is-default-class": "true"}}}'
```

- **START HERE IF ALREADY HAVE K8S**
- We are cheating here.
- Provisioner we are using is only useful for single node systems.
- Need a better distributed posix file system that can autoprovison on K8S (i.e. MapR FS)
Get KFCTL

```bash
opsys=linux
sudo curl -s https://api.github.com/repos/kubeflow/kubeflow/releases/latest |
  grep browser_download |
  grep $opsys |
  cut -d '"' -f 4 |
  xargs curl -O -L && 
  tar -zvxf kfctl_*_${opsys}.tar.gz
```

- KFCTL is Kubeflow’s installation tool. It was previously a shell script that integrated with KSONNET
- Today, it works with Kustomize
Setup Kubeflow

export CONFIG="https://raw.githubusercontent.com/kubeflow/kubeflow/v0.6-branch/bootstrap/config/kfctl_k8s_istio.0.6.2.yaml"

./kfctl init mykubeflow --config=${CONFIG} -V
cd mykubeflow
sudo /home/multipass/kfctl generate all -V
kubectl create ns kubeflow-anonymous
sudo /home/multipass/kfctl apply all -V

- KFCTL Steps are:
  - Init - This downloads the Kubeflow repository
  - Generate - This creates the default YAML files for kustomize
  - Apply - This submits the YAML’s to Kubernetes
Post Init Step

```
multipass@kubeflow:~$ ls
kfctl kfctl_v0.6.2_linux.tar.gz kubernetes-tools mykubeflow snap
multipass@kubeflow:~$ cd mykubeflow
multipass@kubeflow:~/mykubeflow$ ls
app.yaml
```
Post Generate Step

multipass@kubeflow:~/mykubeflow$ ls
app.yaml  kustomize
multipass@kubeflow:~/mykubeflow$ cd kustomize
multipass@kubeflow:~/mykubeflow/kustomize$ ls

api-service  centraldashboard  katib-controller  metadata  persistent-agent
application   istio       katib-db       metrics-collector  pipelines-runner
application-crd  istio-crdsl  katib-manager  minio  pipelines-ui
argo             istio-install  katib-ui  mysql  pipelines-viewer
bootstrap       jupyter-web-app  metacontroller  notebook-controller  profiles
                    suggestion
                                pytorch-operator
                                          scheduledworkflow
                                                   tf-job-operator
                                                            webhook
                                                            spartakus
Wait for startup…

- 15 minutes on laptop
- Can be much faster on real environments
Kubernetes Operators

- Operators are just an extremely useful pattern. They are combination of a Kubernetes Custom Resource and a Controller
  - Custom Resource (CR) - Defines a new type of resource that Kubernetes can manage
  - Controller - Examines a resource submitted to Kubernetes and takes action on behalf of the user
- Operators manage at an even higher level of granularity than simple embedded controllers (StatefulSets, Deployments, etc.)
- Kubernetes operators allow you to install, update, and orchestrate containers based on your own syntax and needs
Kubernetes Operators

- Tend to fall into two categories
  - Create a Job that gets distributed across the cluster.
  - Install/Upgrade Product composed of multiple-pieces

- Job Operators
  - TensorFlow*
  - PyTorch*
  - Spark
  - Katib Hyperparameter Training*
  - Chainer Neural Networks*
  - MXNet Deep Learning*
  - MPI Message Passing Interface*
  - Caffe2 Operator*
  - Pachyderm Pipelines*

* Part of Kubeflow
Traditional Tensorflow Training Architecture

- Worker
- Worker
- Worker
- Chief
- Parameter Server
- Client
TensorFlow Job Operator (PyTorch, etc. follow same pattern)

- Single YAML causes an entire TensorFlow cluster to be created
- Common Pattern for ML Training workflows
Example at:
https://github.com/kubeflow/tf-operator/blob/master/examples/v1/dist-mnist/tf_job_mnist.yaml
Tensorflow Training Container

```
# Copyright 2016 The TensorFlow Authors. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.

FROM tensorflow/tensorflow:1.5.0

ADD . /var/tf_dist_mnist
ENTRYPOINT ["python", "/var/tf_dist_mnist/dist_mnist.py"]
```

Example at:
Mount or Build

- Most examples suggest embedding models or training code in a docker file via a build
- This has serious drawbacks…Pipelines or Argo will be working overtime
- An alternative is to build fewer containers and to add the code or model via a volume mount…
```yaml
apiVersion: "kubeflow.org/v1"
kind: "PyTorchJob"
metadata:
  name: "pytorch-dist-mnist-nccl"
spec:
  pytorchReplicaSpecs:
    Master:
      replicas: 1
      restartPolicy: OnFailure
      template:
        spec:
          containers:
            - name: pytorch
              image: gcr.io/<your_project>/pytorch_dist_mnist:latest
              args: ["--backend", "nccl"]
              resources:
                limits:
                  nvidia.com/gpu: 1
    Worker:
      replicas: 1
      restartPolicy: OnFailure
      template:
        spec:
          containers:
            - name: pytorch
              image: gcr.io/<your_project>/pytorch_dist_mnist:latest
              args: ["--backend", "nccl"]
              resources:
```

Example at: https://github.com/kubeflow/pytorch-operator/blob/master/examples/mnist/v1/pytorch_job_mnist_nccl.yaml
MXNet Job

- Multi-language Deep Learning Framework
- Apache Project

Example at:
https://github.com/kubeflow/mxnet-operator/blob/master/examples/v1beta1/train/mx_job_dist_gpu.yaml
**MPI Job**

- Message Passing Interface Framework
- Intel

Example at:
https://github.com/kubeflow/mpi-operator/blob/master/examples/mxnet/mxnet-mnist.yaml
Chainer Job

- Python Deep Learning Framework

Example at:
https://github.com/kubeflow/chainer-operator/blob/master/examples/chainerjob-mn.yaml
XGBoost Job

- Gradient Boosting Framework
- Early. Still Incubating

Example at:
https://github.com/kubeflow/community/issues/247
**Caffe2 Job**

- Lightweight Deep Learning Framework
- Joining forces with PyTorch

Example at:
https://github.com/kubeflow/caffe2-operator/blob/master/examples/resnet50_redis.yaml
Start UI Proxy

```
kubectl port-forward -n istio-system svc/istio-ingressgateway 8080:80
```

- Not necessary in the real world… Load balancers in the cloud cooperate with Ambassador service to allow external traffic to communicate with Kubeflow UI’s
* Optional* Install Kubernetes

https://kubernetic.com
* Optional* Kubectl access outside of VM

multipass list

```
Name       State     IPv4            Image
kubeflow   Running  192.168.64.2  Ubuntu 18.04 LTS
```
* Optional* Kubectl access outside of VM

cd .kube

cat config
* Optional* Kubectl access outside of VM

- Copy kubectl config info from vm to kubectl config outside of vm…
  - certificate-authority-data
  - password
- Use ip address from multipass list
- To test:
  kubectl get ns
**Exploration/Notebook Phase**

- Starting point for Data Exploration
- Started via Launcher
- Runs behind Ambassador Proxy
- Started via Jupyter Hub
- Data Analysis Tools
- Typical Volume Types Used
  - Home, Shared Libraries, Training Data, Shared Notebooks, Models
Start Notebook

- 10 minutes on laptop… Faster in real world
# Copyright (c) Jupyter Development Team.
# Distributed under the terms of the Modified BSD License.
ARG BASE_CONTAINER=jupyter/scipy-notebook
FROM $BASE_CONTAINER

LABEL maintainer="Jupyter Project <jupyter@googlegroups.com>"

# Install Tensorflow
RUN conda install --quiet --yes \
    'tensorflow=1.13*' \
    'keras=2.2*' && \
    conda clean --all -f -y && \
    fix-permissions $CONDA_DIR && \
    fix-permissions /home/$NB_USER

More complicated example at:
### Start Notebook

#### Notebook Servers

<table>
<thead>
<tr>
<th>Status</th>
<th>Name</th>
<th>Image</th>
<th>CPU</th>
<th>Memory</th>
<th>Volumes</th>
<th>Connect</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>sky</td>
<td>tensorflow-1.13.1-notebook-cpu:v0.5.0</td>
<td>0.5</td>
<td>1.0Gi</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Start Notebook

Select items to perform actions on them.

The notebook list is empty.
from tensorflow.examples.tutorials.mnist import input_data

mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

import tensorflow as tf

x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)

y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))

train_step = tf.train.GradientDescentOptimizer(0.05).minimize(cross_entropy)

with tf.Session() as sess:
    tf.global_variables_initializer().run()

    for _ in range(1000):
        batch_xs, batch_ys = mnist.train.next_batch(100)
        sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

    correct_prediction = tf.equal(tf.argmax(y, 1), tf.argmax(y_, 1))
    accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))

    print("Accuracy: ", sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
Training Phase

- Multi-machine training can be started via a call from a notebooks
- Or start externally via job
- Many possible Training Types
- Security Mapping
- Volumes
  - Training Code, Shared Libraries, Models, Training Data, Logs
What the heck are pipelines?

- Pipelines are multi-step ML workflows
- Created in Python
- Visual Representation
- Uses Argo behind the scenes for workflow
- Includes Metadata (relational DB) and Artifacts (Files)
- Very new… Not quite baked…
Pipeline UI
Pipeline SDK Architecture
Argo Workflow

- Runs Workflows
- DAG Engine
- Compare to Airflow or Oozie

Example at:
https://github.com/argoproj/argo/blob/master/examples/dag-diamond.yaml

```yaml
apiVersion: argoproj.io/v1alpha1
class: Workflow
metadata:
  generateName: dag-diamond-
spec:
  entrypoint: diamond
templates:
  - name: diamond
tasks:
  - name: A
template: echo
    arguments:
      parameters: [{name: message, value: A}]
  - name: B
dependencies: [A]
template: echo
    arguments:
      parameters: [{name: message, value: B}]
  - name: C
dependencies: [A]
template: echo
    arguments:
      parameters: [{name: message, value: C}]
  - name: D
dependencies: [B, C]
template: echo
    arguments:
      parameters: [{name: message, value: D}]
  - name: echo
    inputs:
      parameters:
        - name: message
    container:
      image: alpine:3.7
      command: [echo, "${{inputs.parameters.message}}"]
```
Katib Job

- Runs Experiments
- Based on Google’s Vizer
- Hyperparameters and constructed parameters

Example at:
https://github.com/kubeflow/katib/blob/master/examples/v1alpha3/tfjob-example.yml
Serving Phase

- Multiple options for serving models..
- Least baked part of Kubeflow
- Most exposed via Ambassador
- Long Term KFServing simplifies story
- Multiple Choices for TensorFlow model serving including:
  - TensorFlow Serving, Tensorflow Batch, Seldon, NVIDIA TensorRT, Istio Integration
- Prepackaged Seldon Model Servers:
  - SciKitLearn, XGBoost, TensorFlow, MLFlow, PyTorch
- Volumes
  - Models, logs
User Workflow - Simple Model Serving

Model lives in container
- Container translates incoming requests into something the model can understand and respond to
- Because each tool
User Workflow - Complex Model Serving

Complex graphs allow for inter-model routing and other advanced ensembles:

- Routers - route API requests to sub-graphs. Examples: AB Tests, Multi-Armed Bandits.
- Combiners - combine the responses from sub-graphs. Examples: ensembles of models
- Transformers - transform request or responses. Example: transform feature requests.
Seldon

- Serve Model Containers

---

Example at:
https://github.com/kubeflow/katib/blob/master/examples/v1alpha3/tfjob-example.yml
Istio
Kubeflow is trying to remove this complexity with the KFServing project.

Not quite baked yet

Handles predictor models created with:

- TensorFlow
- SciKit Learn
- XGBoost
- PyTorch
- ONNX
Fairing

- Very new component
- Python Library
- Build, train, and deploy training jobs remotely
- Pushes to Google’s AI Platform
- Remotely create docker files
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("MNIST_data/", one_hot=True)

import tensorflow as tf
x = tf.placeholder(tf.float32, [None, 784])
W = tf.Variable(tf.zeros([784, 10]))
b = tf.Variable(tf.zeros([10]))
y = tf.nn.softmax(tf.matmul(x, W) + b)
y_ = tf.placeholder(tf.float32, [None, 10])
cross_entropy = tf.reduce_mean(-tf.reduce_sum(y_ * tf.log(y), reduction_indices=[1]))
train_step = tf.train.GradientDescentOptimizer(0.05).minimize(cross_entropy)
sess = tf.InteractiveSession()
tf.global_variables_initializer().run()

for _ in range(1000):
    batch_xs, batch_ys = mnist.train.next_batch(100)
sess.run(train_step, feed_dict={x: batch_xs, y_: batch_ys})

correct_prediction = tf.equal(tf.argmax(y,1), tf.argmax(y_,1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
print("Accuracy: ", sess.run(accuracy, feed_dict={x: mnist.test.images, y_: mnist.test.labels}))
2019 Kubeflow Roadmap

0.6 Highlights
● KSonnet -> Kustomize
● Initial version of Metadata for pipelines
● Pipeline Improvements (Still early IMHO)
● Initial Multi-User

0.7 Highlights
● Notebooks to Beta
● Clean up deployment
● Namespace/RBAC fixes
●

1.0 = 2020
● Many components still Beta
Thank You for your attention!

Twitter: @skythomas
Rate today’s session

Session page on conference website

O’Reilly Events App