



Introduction to TensorFlow

Machine Learning on Google Cloud Platform

Josh Cogan

Learn how to...

Train a model on Google Cloud

Learn how to...

Train a model on Google Cloud

Monitor model training

Learn how to...

Train a model on Google Cloud

Monitor model training

Deploy a trained model as a
microservice



Why Cloud ML Engine?

Josh Cogan

We will use distributed TensorFlow on Cloud ML Engine

Run TF at scale

tf.estimator

tf.layers, tf.losses, tf.metrics

Core TensorFlow (Python)

Core TensorFlow (C++)

CPU

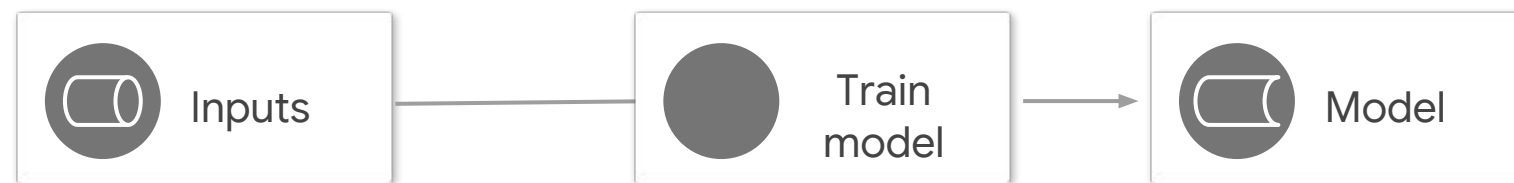
GPU

TPU

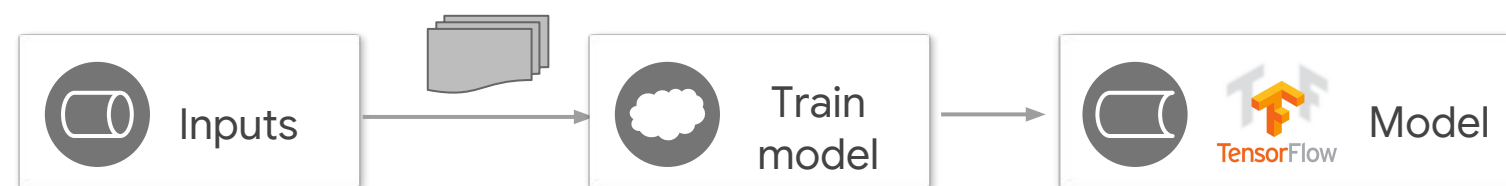
Android

Cloud ML Engine

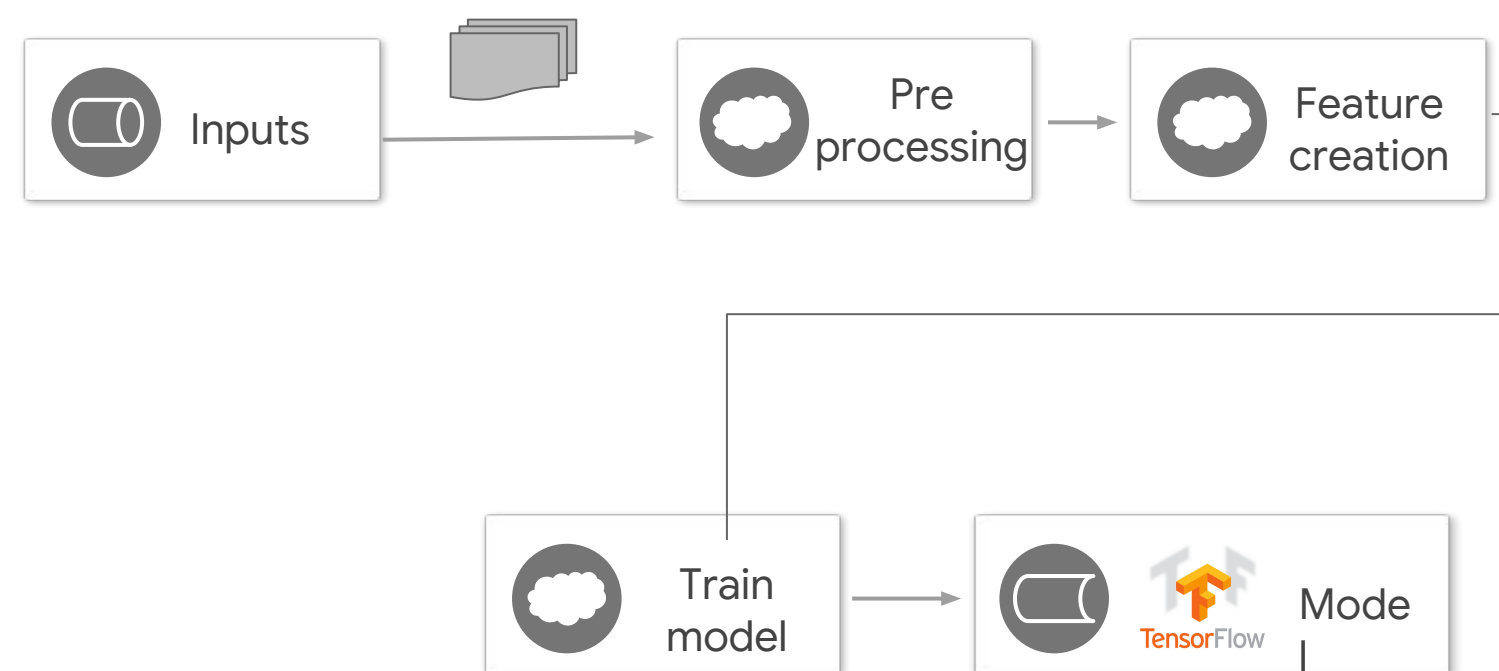
Many machine learning
frameworks can handle
toy problems



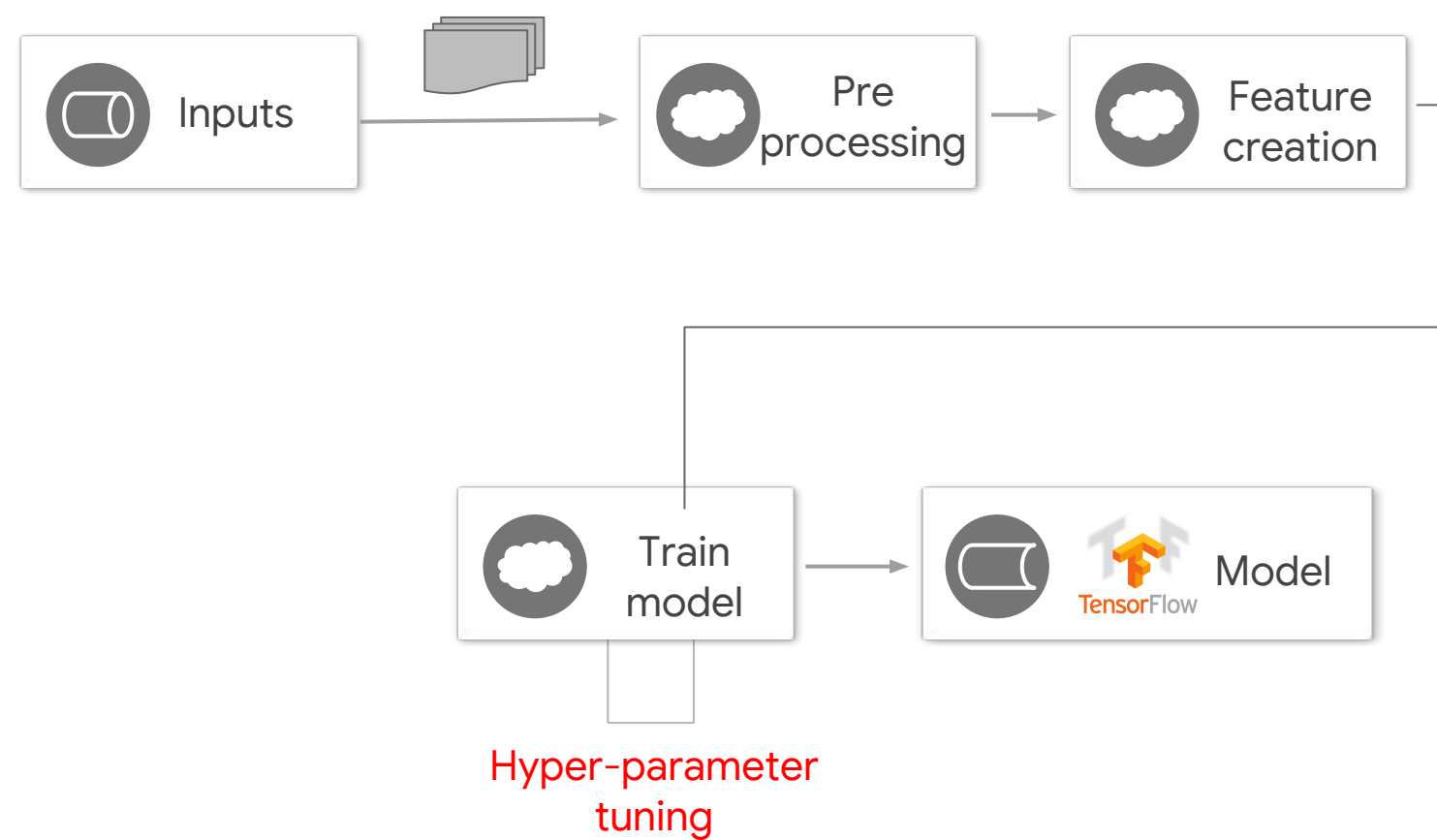
As your data size increases,
batching and distribution
become important



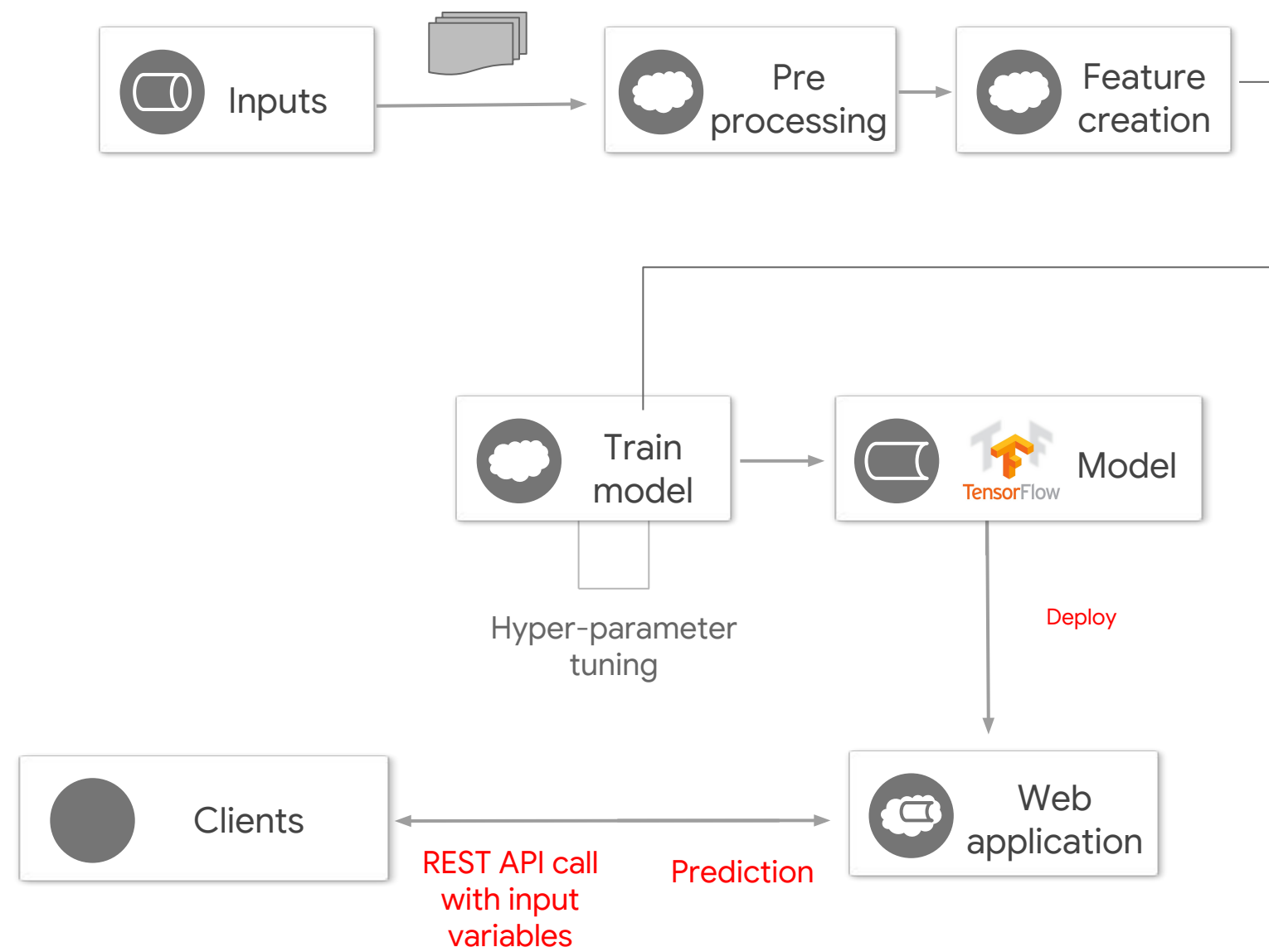
Input necessary transformations



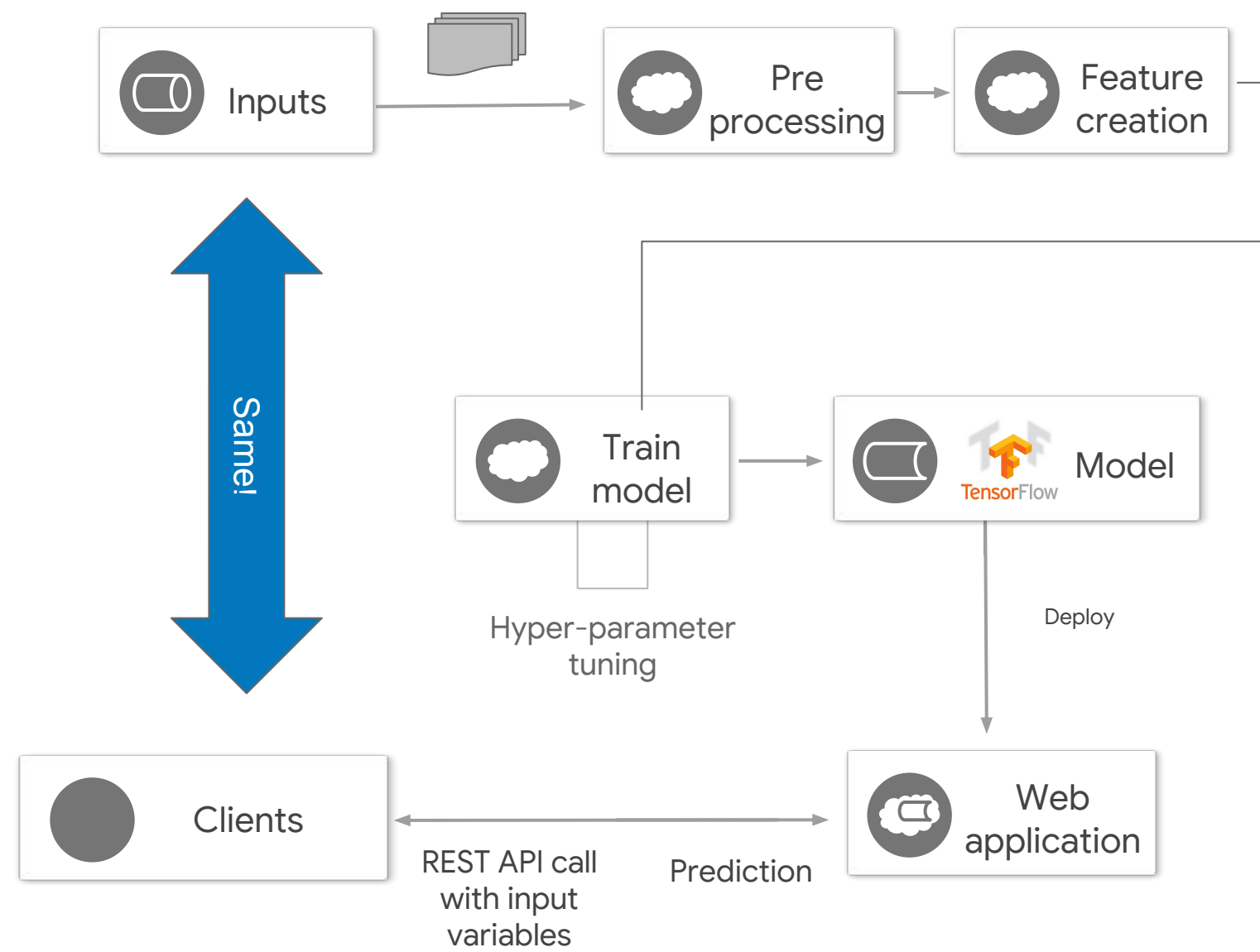
Hyperparameter tuning might be nice



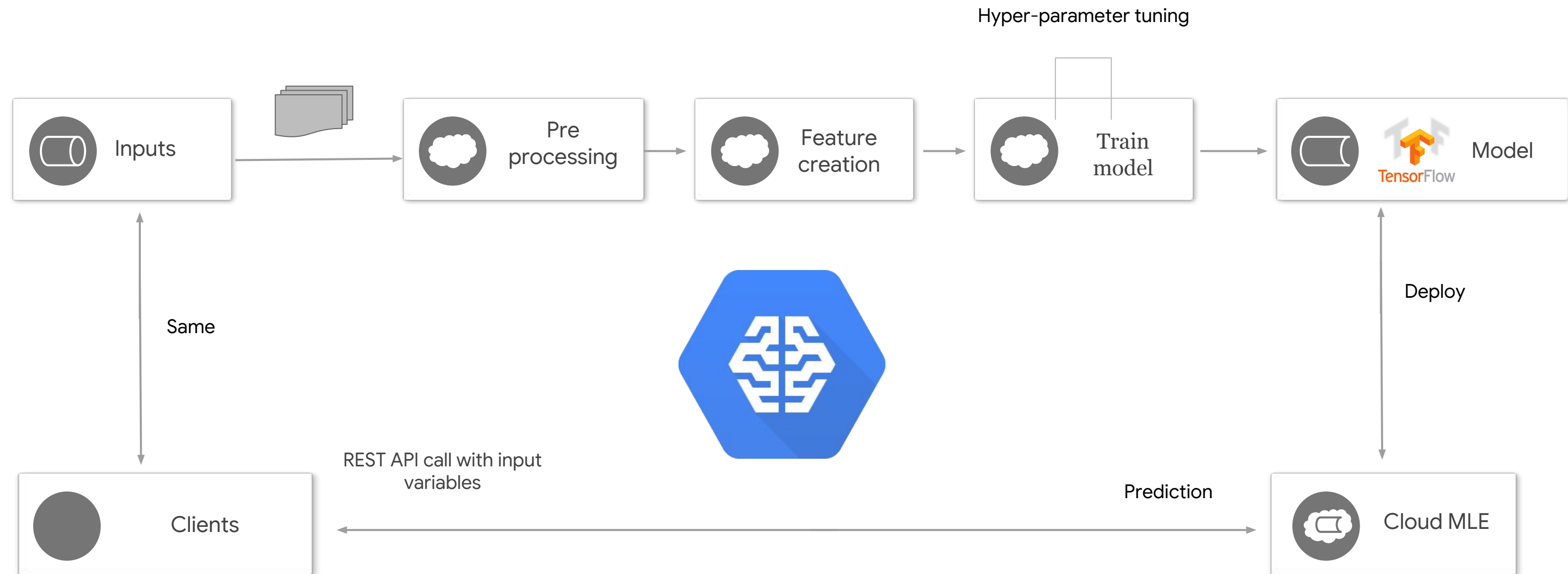
Need to autoscale prediction code



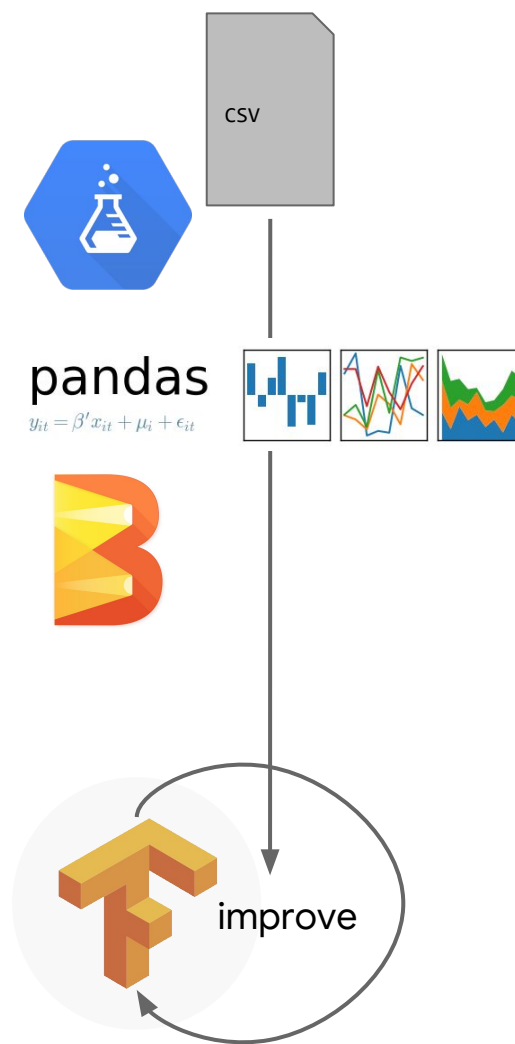
Who does the preprocessing?



Cloud Machine Learning Engine - repeatable, scalable, tuned



In Datalab, start locally on sampled dataset



Google Cloud Datalab tfclassic (unsaved changes)

Notebook Add Code Add Markdown Delete Move Up Move Down Run Clear Reset Session Widgets Navigation Help

20. WORKING WITH LOW-LEVEL TENSORFLOW

This notebook is Lab2b of CPB 102, Google's course on Machine Learning using Cloud ML.

In this notebook, we will work with relatively low-level TensorFlow functions to implement a linear regression model. We will use this notebook to demonstrate early stopping -- a technique whereby training is stopped once the error on the validation dataset starts to increase.

```
import datalab.bigquery as bq
import tensorflow as tf
import pandas as pd
import numpy as np
import shutil
```

Code to read data and compute error is the same as Lab2a.

```
def read_dataset(filename):
    return pd.read_csv(filename, header=None, names=['pickuplon', 'pickuplat', 'dropofflon', 'dropofflat', 'passengers', 'fare_amount'])

df_train = read_dataset('../lab1a/taxi-train.csv')
df_valid = read_dataset('../lab1a/taxi-valid.csv')
df_test = read_dataset('../lab1a/taxi-test.csv')
df_train[:5]
```

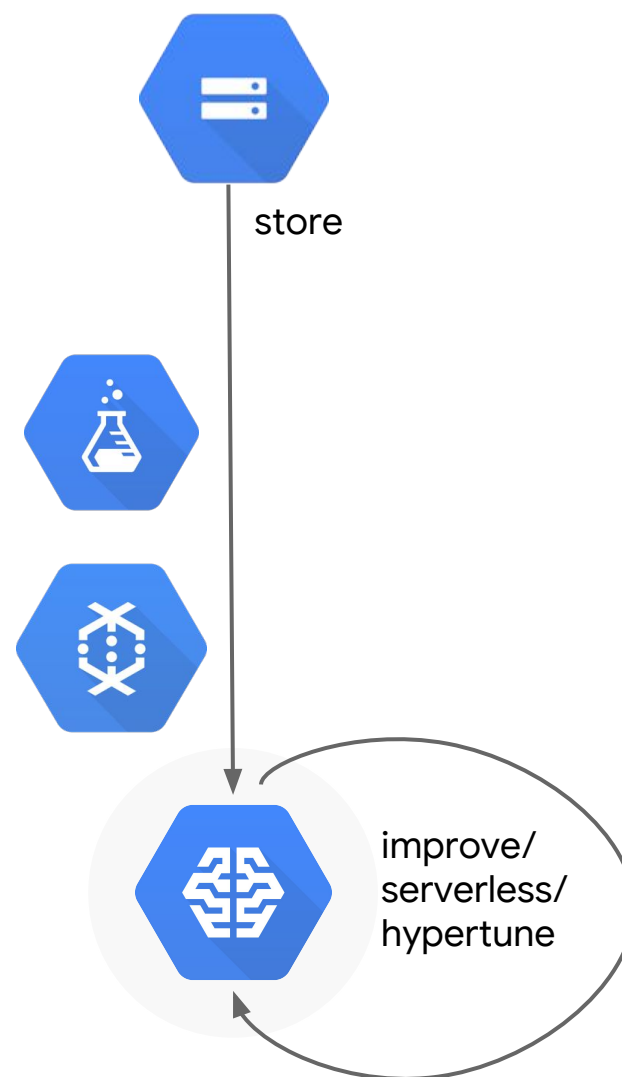
```
FEATURE_COLS = np.arange(0,5)
TARGET_COL = 'fare_amount'
```

```
def compute_rmse(actual, predicted):
    return np.sqrt(np.mean((actual-predicted)**2))

def print_rmse(model):
    print "Train RMSE = {}".format(compute_rmse(df_train[TARGET_COL], model.predict(df_train.iloc[:,FEATURE_COLS].values)))
    print "Valid RMSE = {}".format(compute_rmse(df_valid[TARGET_COL], model.predict(df_valid.iloc[:,FEATURE_COLS].values)))
```

Linear Regression

Then, scale it out to GCP using serverless technology



Google Cloud Datalab cloudml (autosaved)

Notebook + Add Code Add Markdown Delete Move Up Move Down Run Clear Reset Session Widgets Navigation Help

Training on cloud

In order to train on the cloud, we have to copy the model and data to our bucket on Google Cloud Storage (GCS).

```
%bash
rm -rf taxifare.tar.gz taxi_trained
tar cvfz taxifare.tar.gz taxifare
gsutil cp taxifare.tar.gz gs://$BUCKET/taxifare/source/taxifare.tar.gz
gsutil cp ../lab1a/*.csv gs://$BUCKET/taxifare/input/
gsutil -m rm -r -f gs://$BUCKET/taxifare/taxi_preproc
gsutil -m rm -r -f gs://$BUCKET/taxifare/taxi_trained
```

Running...

When you run your preprocessor, you have to change the input and output to be on GCS.

Using DirectPipelineRunner runs Dataflow locally, but the inputs & outputs are on the cloud. Using BlockingDataflowPipelineRunner will use Cloud Dataflow (and take much longer because of the overhead involved for such a small dataset). To see the status of your BlockingDataflowPipelineRunner job, visit <https://console.cloud.google.com/dataflow>

```
# imports
import apache_beam as beam
import google.cloud.ml as ml
import google.cloud.ml.dataflow.io.tfrecordio as tfrecordio
import google.cloud.ml.io as io
import os

# Change as needed
#RUNNER = 'DirectPipelineRunner' #
RUNNER = 'BlockingDataflowPipelineRunner'

# defines
feature_set = TaxifareFeatures()
OUTPUT_DIR = 'gs://{0}/taxifare/taxi_preproc'.format(BUCKET)
```



Train a model

Josh Cogan

Training your model with Cloud Machine Learning Engine

- 1 Use TensorFlow to create computation graph and training application
- 2 Package your trainer application
- 3 Configure and start a Cloud ML Engine job

Create task.py to parse command-line parameters and send along to train_and_evaluate

```
task.py
parser.add_argument(
    '--train_data_paths', required=True)
parser.add_argument(
    '--train_steps', ...
```

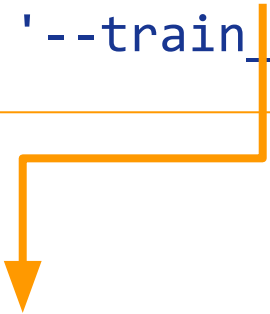
Create task.py to parse command-line parameters and send along to train_and_evaluate

model.py

```
def train_and_evaluate(args):
    estimator = tf.estimator.DNNRegressor(
        model_dir=args['output_dir'],
        feature_columns=feature_cols,
        hidden_units=args['hidden_units'])
    train_spec=tf.estimator.TrainSpec(
        input_fn=read_dataset(args['train_data_paths'],
                               batch_size=args['train_batch_size'],
                               mode=tf.contrib.learn.ModeKeys.TRAIN),
        max_steps=args['train_steps'])
    exporter = tf.estimator.LatestExporter('exporter', serving_input_fn)
    eval_spec=tf.estimator.EvalSpec(...)
    tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```

task.py

```
parser.add_argument(
    '--train_data_paths', required=True)
parser.add_argument(
    '--train_steps', ...)
```




Create task.py to parse command-line parameters and send along to train_and_evaluate

model.py

```
def train_and_evaluate(args):
    estimator = tf.estimator.DNNRegressor(
        model_dir=args['output_dir'],
        feature_columns=feature_cols,
        hidden_units=args['hidden_units'])
    train_spec=tf.estimator.TrainSpec(
        input_fn=read_dataset(args['train_data_paths'],
                               batch_size=args['train_batch_size'],
                               mode=tf.contrib.learn.ModeKeys.TRAIN),
        max_steps=args['train_steps'])
    exporter = tf.estimator.LatestExporter('exporter', serving_input_fn)
    eval_spec=tf.estimator.EvalSpec(...)
    tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```

task.py

```
parser.add_argument(
    '--train_data_paths', required=True)
parser.add_argument(
    '--train_steps', ...)
```




Create task.py to parse command-line parameters and send along to train_and_evaluate

model.py

```
def train_and_evaluate(args):
    estimator = tf.estimator.DNNRegressor(
        model_dir=args['output_dir'],
        feature_columns=feature_cols,
        hidden_units=args['hidden_units'])
    train_spec=tf.estimator.TrainSpec(
        input_fn=read_dataset(args['train_data_paths'],
                               batch_size=args['train_batch_size'],
                               mode=tf.contrib.learn.ModeKeys.TRAIN),
        max_steps=args['train_steps'])
    exporter = tf.estimator.LatestExporter('exporter', serving_input_fn)
    eval_spec=tf.estimator.EvalSpec(...)
    tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)
```

task.py

```
parser.add_argument(
    '--train_data_paths', required=True)
parser.add_argument(
    '--train_steps', ...)
```



The model.py contains the ML model in TensorFlow (Estimator API)

	Example of the code in model.py (see previous chapter)
Training and evaluation input functions	<pre>CSV_COLUMNS = ... def read_dataset(filename, mode, batch_size=512): ...</pre>
Feature columns	<pre>INPUT_COLUMNS = [tf.feature_column.numeric_column('pickuplon'),</pre>
Feature engineering	<pre>def add_more_features(feats): # will be covered in next course; for now, just a no-op return feats</pre>
Serving input function	<pre>def serving_input_fn(): ... return tf.estimator.export.ServingInputReceiver(features, feature_pholders)</pre>
Train and evaluate loop	<pre>def train_and_evaluate(args): ... tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)</pre>

The model.py contains the ML model in TensorFlow (Estimator API)

	Example of the code in model.py (see previous chapter)
Training and evaluation input functions	<pre>CSV_COLUMNS = ... def read_dataset(filename, mode, batch_size=512): ...</pre>
Feature columns	<pre>INPUT_COLUMNS = [tf.feature_column.numeric_column('pickuplon'),</pre>
Feature engineering	<pre>def add_more_features(feats): # will be covered in next course; for now, just a no-op return feats</pre>
Serving input function	<pre>def serving_input_fn(): ... return tf.estimator.export.ServingInputReceiver(features, feature_pholders)</pre>
Train and evaluate loop	<pre>def train_and_evaluate(args): ... tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)</pre>

The model.py contains the ML model in TensorFlow (Estimator API)

	Example of the code in model.py (see previous chapter)
Training and evaluation input functions	<pre>CSV_COLUMNS = ... def read_dataset(filename, mode, batch_size=512): ...</pre>
Feature columns	<pre>INPUT_COLUMNS = [tf.feature_column.numeric_column('pickuplon'),</pre>
Feature engineering	<pre>def add_more_features(feats): # will be covered in next course; for now, just a no-op return feats</pre>
Serving input function	<pre>def serving_input_fn(): ... return tf.estimator.export.ServingInputReceiver(features, feature_pholders)</pre>
Train and evaluate loop	<pre>def train_and_evaluate(args): ... tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)</pre>

The model.py contains the ML model in TensorFlow (Estimator API)

	Example of the code in model.py (see previous chapter)
Training and evaluation input functions	<pre>CSV_COLUMNS = ... def read_dataset(filename, mode, batch_size=512): ...</pre>
Feature columns	<pre>INPUT_COLUMNS = [tf.feature_column.numeric_column('pickuplon'),</pre>
Feature engineering	<pre>def add_more_features(feats): # will be covered in next course; for now, just a no-op return feats</pre>
Serving input function	<pre>def serving_input_fn(): ... return tf.estimator.export.ServingInputReceiver(features, feature_pholders)</pre>
Train and evaluate loop	<pre>def train_and_evaluate(args): ... tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)</pre>

The model.py contains the ML model in TensorFlow (Estimator API)

	Example of the code in model.py (see previous chapter)
Training and evaluation input functions	<pre>CSV_COLUMNS = ... def read_dataset(filename, mode, batch_size=512): ...</pre>
Feature columns	<pre>INPUT_COLUMNS = [tf.feature_column.numeric_column('pickuplon'),</pre>
Feature engineering	<pre>def add_more_features(feats): # will be covered in next course; for now, just a no-op return feats</pre>
Serving input function	<pre>def serving_input_fn(): ... return tf.estimator.export.ServingInputReceiver(features, feature_pholders)</pre>
Train and evaluate loop	<pre>def train_and_evaluate(args): ... tf.estimator.train_and_evaluate(estimator, train_spec, eval_spec)</pre>

Package up TensorFlow model as Python package

```
taxifare/  
taxifare/PKG-INFO  
taxifare/setup.cfg  
taxifare/setup.py  
taxifare/trainer/  
taxifare/trainer/__init__.py  
taxifare/trainer/task.py  
taxifare/trainer/model.py
```

Python modules
need to contain
an `__init__.py` in
every folder

Verify that the model works as a Python package

```
export PYTHONPATH=${PYTHONPATH}:/somedir/taxifare
python -m trainer.task \
  --train_data_paths="/somedir/datasets/*train*" \
  --eval_data_paths="/somedir/datasets/*valid*" \
  --output_dir="/somedir/output" \
  --train_steps=100 --job-dir=/tmp
```

Verify that the model works as a Python package

```
export PYTHONPATH=${PYTHONPATH}:/somedir/taxifare
python -m trainer.task \
  --train_data_paths="/somedir/datasets/*train*" \
  --eval_data_paths="/somedir/datasets/*valid*" \
  --output_dir="/somedir/output" \
  --train_steps=100 --job-dir=/tmp
```

Then use the gcloud command to submit the training job,
either locally or to cloud

```
gcloud ml-engine local train \  
  --module-name=trainer.task \  
  --package-path=/somedir/taxifare/trainer \  
  -- \  
  --train_data_paths etc.  
REST as before
```

```
gcloud ml-engine jobs submit training $JOBNAME \  
  --region=$REGION \  
  --module-name=trainer.task \  
  --job-dir=$OUTDIR --staging-bucket=gs://$BUCKET \  
  --scale-tier=BASIC \  
REST as before
```

Then use the gcloud command to submit the training job,
either locally or to cloud

```
gcloud ml-engine local train \  
  --module-name=trainer.task \  
  --package-path=/somedir/taxifare/trainer \  
  -- \  
  --train_data_paths etc.  
REST as before
```

```
gcloud ml-engine jobs submit training $JOBNAME \  
  --region=$REGION \  
  --module-name=trainer.task \  
  --job-dir=$OUTDIR --staging-bucket=gs://$BUCKET \  
  --scale-tier=BASIC \  
REST as before
```

Then use the gcloud command to submit the training job,
either locally or to cloud

```
gcloud ml-engine local train \  
  --module-name=trainer.task \  
  --package-path=/somedir/taxifare/trainer \  
  -- \  
  --train_data_paths etc.  
REST as before
```

```
gcloud ml-engine jobs submit training $JOBNAME \  
  --region=$REGION \  
  --module-name=trainer.task \  
  --job-dir=$OUTDIR --staging-bucket=gs://$BUCKET \  
  --scale-tier=BASIC \  
REST as before
```


Then use the gcloud command to submit the training job,
either locally or to cloud

```
gcloud ml-engine local train \  
  --module-name=trainer.task \  
  --package-path=/somedir/taxifare/trainer \  
  -- \  
  --train_data_paths etc.  
REST as before
```

```
gcloud ml-engine jobs submit training $JOBNAME \  
  --region=$REGION \  
  --module-name=trainer.task \  
  --job-dir=$OUTDIR --staging-bucket=gs://$BUCKET \  
  --scale-tier=BASIC \  
REST as before
```

Scale Tier Options

BASIC

Scale Tier Options

BASIC

STANDARD

Scale Tier Options

BASIC

STANDARD

BASIC_GPU

Scale Tier Options

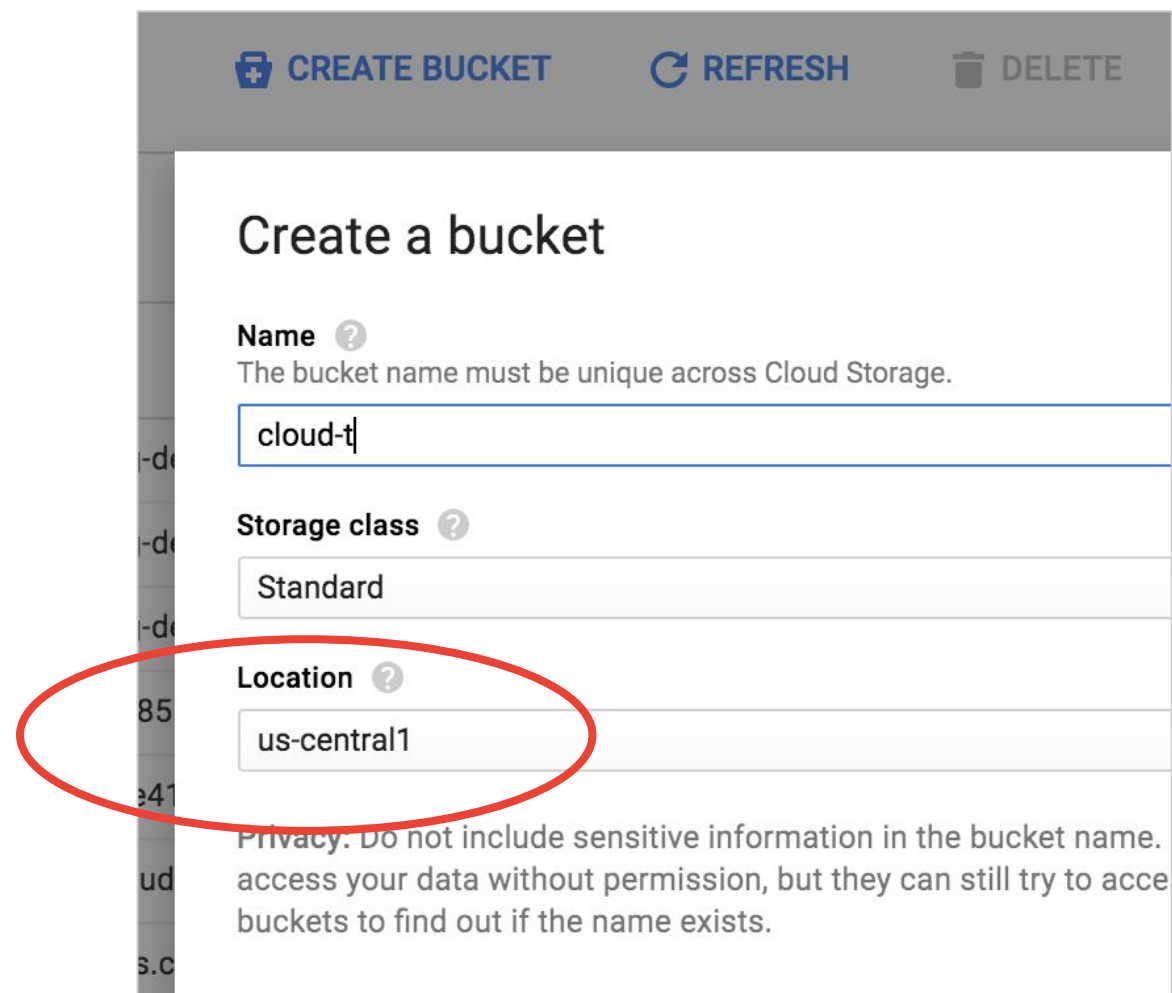
BASIC

STANDARD

BASIC_GPU

BASIC_TPU

Tip: Use single-region bucket for ML



The screenshot shows the 'Create a bucket' dialog in the Google Cloud console. At the top, there are three buttons: 'CREATE BUCKET' (with a plus icon), 'REFRESH' (with a circular arrow icon), and 'DELETE' (with a trash icon). The main title is 'Create a bucket'. Below this, there are three sections: 'Name', 'Storage class', and 'Location'. The 'Name' section has a text input field containing 'cloud-t' and a note: 'The bucket name must be unique across Cloud Storage.' The 'Storage class' section has a dropdown menu with 'Standard' selected. The 'Location' section has a dropdown menu with 'us-central1' selected, which is circled in red. At the bottom, there is a 'Privacy' note: 'Do not include sensitive information in the bucket name. access your data without permission, but they can still try to access buckets to find out if the name exists.'

CREATE BUCKET **REFRESH** **DELETE**

Create a bucket

Name ?
The bucket name must be unique across Cloud Storage.

cloud-t

Storage class ?
Standard

Location ?
us-central1

Privacy. Do not include sensitive information in the bucket name. access your data without permission, but they can still try to access buckets to find out if the name exists.



Monitoring and Deploying a Trained Model

Josh Cogan

Monitor training jobs with gcloud

Get details of current state of job

```
gcloud ml-engine jobs describe job_name
```


Monitor training jobs with gcloud

Get details of current state of job

```
gcloud ml-engine jobs describe job_name
```

Get latest logs from job

```
gcloud ml-engine jobs stream-jobs job_name
```

Monitor training jobs with gcloud

Get details of current state of job

```
gcloud ml-engine jobs describe job_name
```

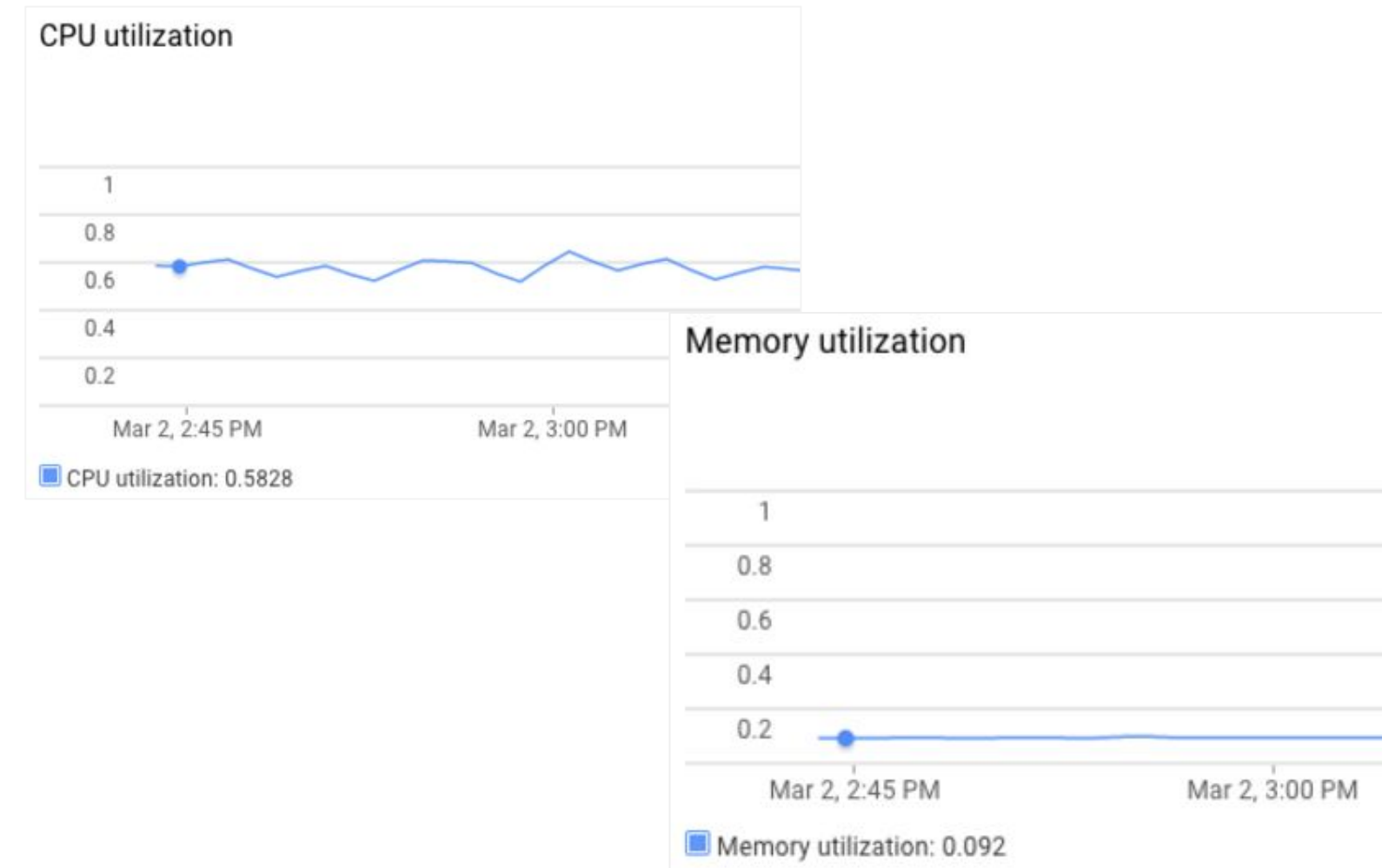
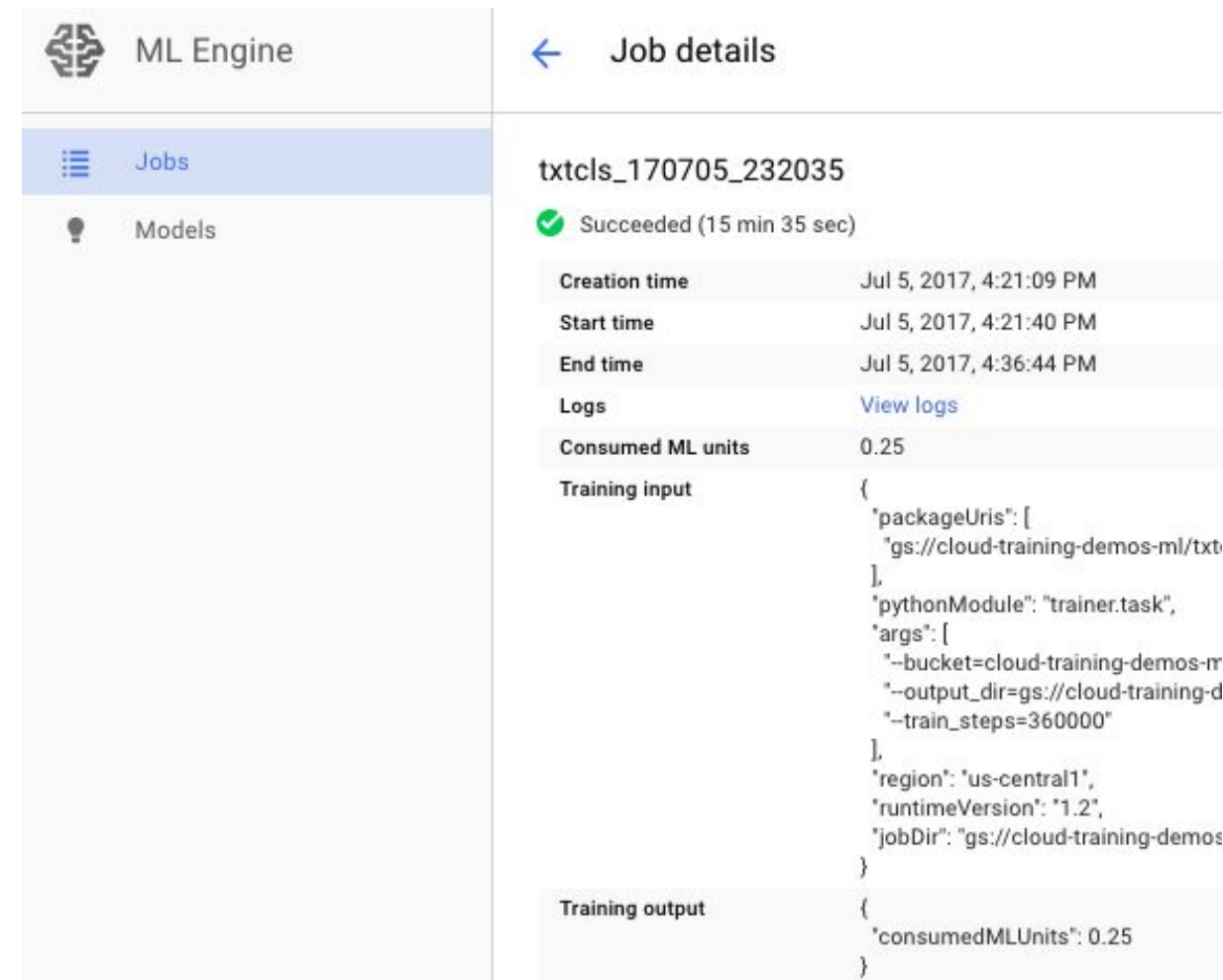
Get latest logs from job

```
gcloud ml-engine jobs stream-jobs job_name
```

Filter jobs based on creation time or name

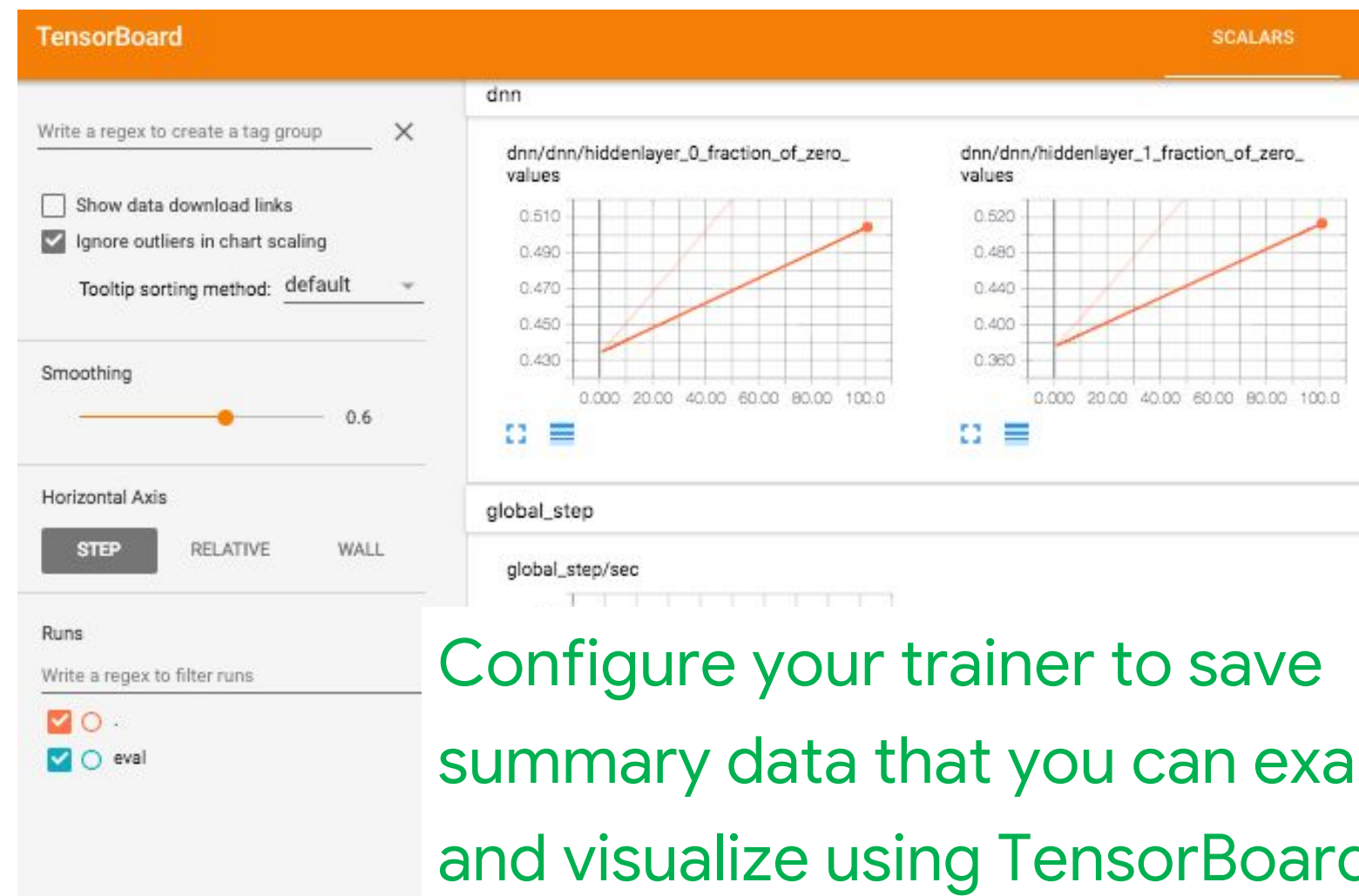
```
gcloud ml-engine jobs list --filter='createTime>2017-01-15T19:00'  
gcloud ml-engine jobs list --filter='jobId:census*' --limit=3
```

Monitor training jobs with GCP console



You can also view CPU and Memory utilization charts for this training job with Stack Driver Monitoring

Monitor training jobs with TensorBoard

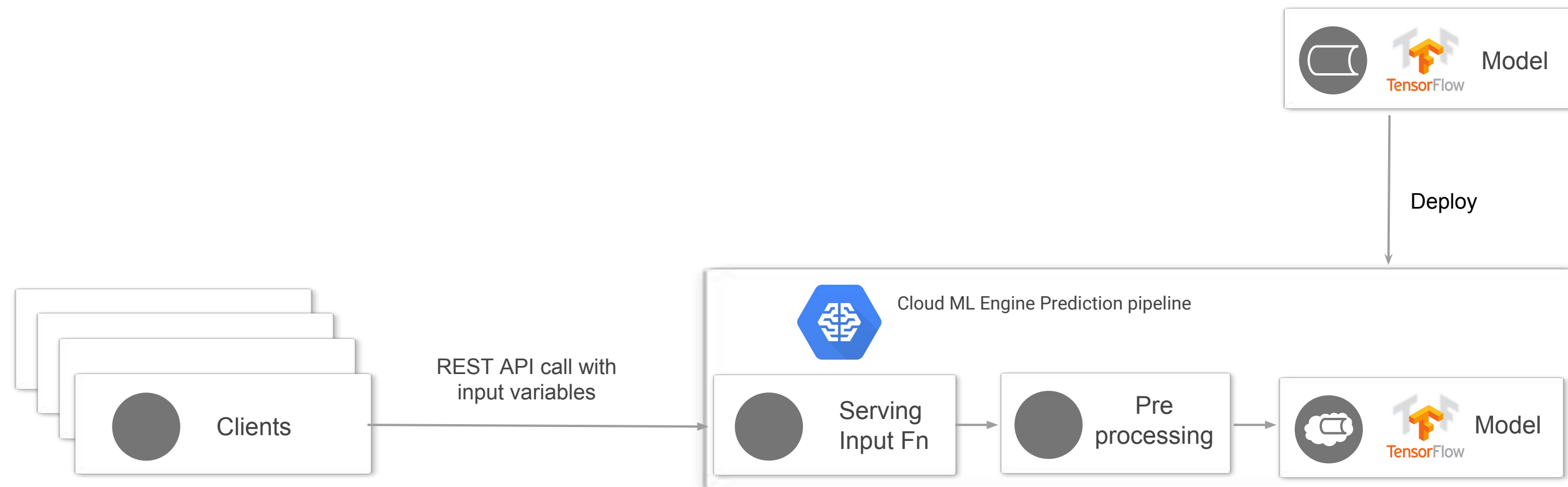




Deploy a trained model and
make predictions

Josh Cogan

Cloud ML Engine makes deploying models and scaling the infrastructure easy



Deploy the saved model to GCP

```
MODEL_NAME="taxifare"
```

Could also be a locally-trained model

```
MODEL_VERSION="v1"
```

```
MODEL_LOCATION="gs://${BUCKET}/taxifare/smallinput/taxi_trained/export/Servo/.../"
```

```
gcloud ml-engine models create ${MODEL_NAME} --regions $REGION
```

```
gcloud ml-engine versions create ${MODEL_VERSION} --model ${MODEL_NAME} --origin
```

```
${MODEL_LOCATION} --runtime-version 1.4
```

Deploy the saved model to GCP

```
MODEL_NAME="taxifare"
```

Could also be a locally-trained model

```
MODEL_VERSION="v1"
```

```
MODEL_LOCATION="gs://${BUCKET}/taxifare/smallinput/taxi_trained/export/Servo/.../"
```

```
gcloud ml-engine models create ${MODEL_NAME} --regions $REGION
```

```
gcloud ml-engine versions create ${MODEL_VERSION} --model ${MODEL_NAME} --origin  
${MODEL_LOCATION} --runtime-version 1.4
```


Client code can make REST calls

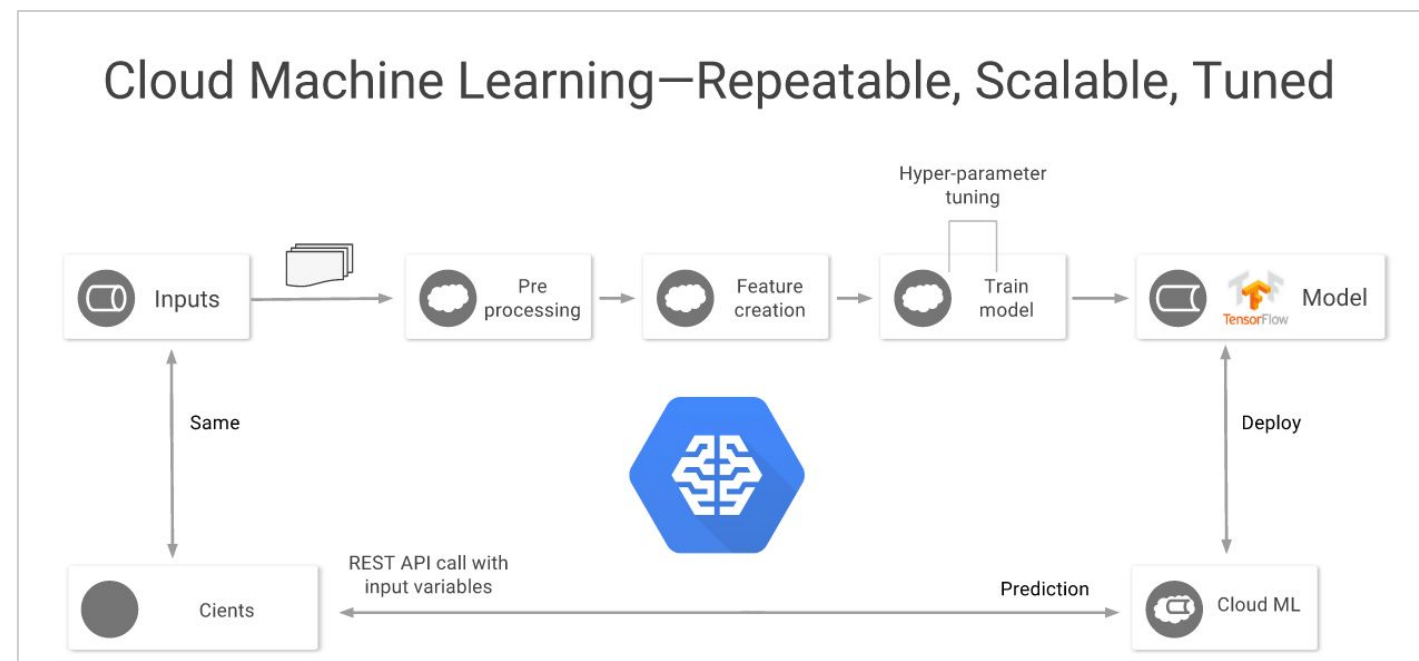
```
credentials = GoogleCredentials.get_application_default()
api = discovery.build('ml', 'v1', credentials=credentials)
request_data = [
    {'pickup_longitude': -73.885262,
     'pickup_latitude': 40.773008,
     'dropoff_longitude': -73.987232,
     'dropoff_latitude': 40.732403,
     'passenger_count': 2}]
parent = 'projects/%s/models/%s/versions/%s' % ('cloud-training-demos', 'taxifare', 'v1')
response = api.projects().predict(body={'instances': request_data},
name=parent).execute()
```

Lab

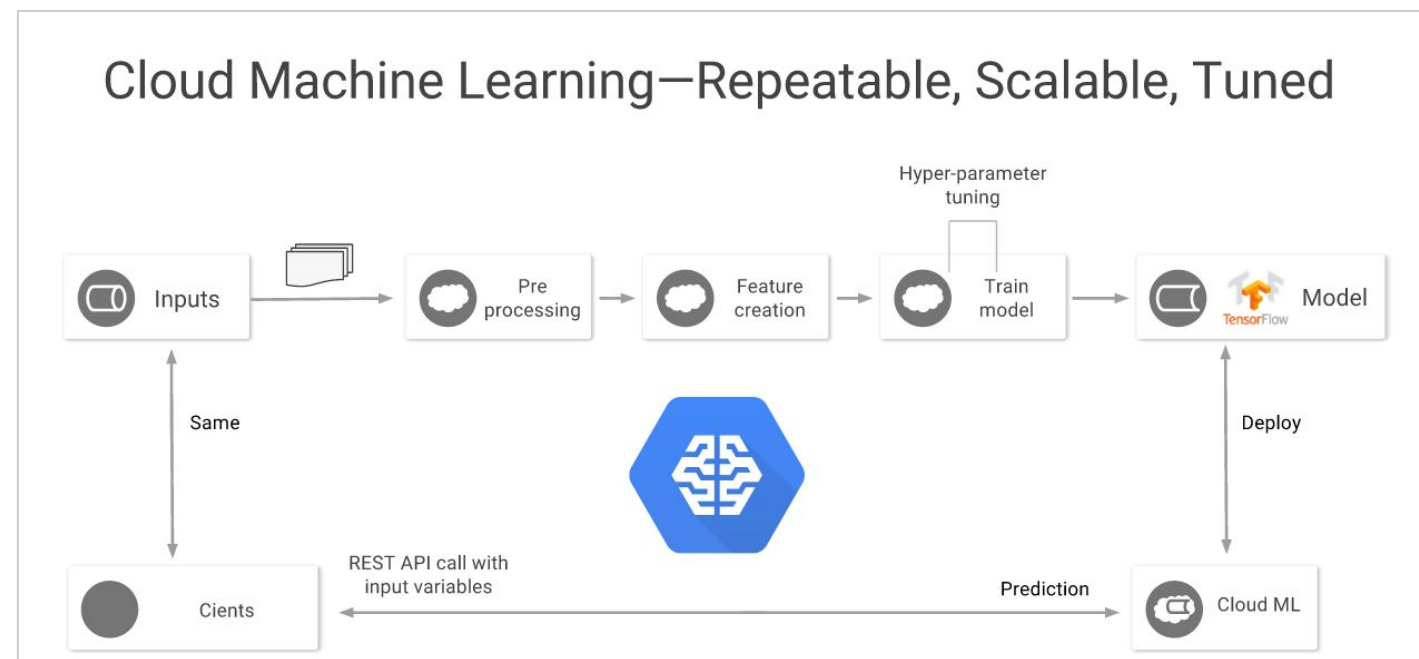
Scaling TensorFlow with Cloud MLE

Josh Cogan

Lab: Scaling TensorFlow with Cloud Machine Learning Engine

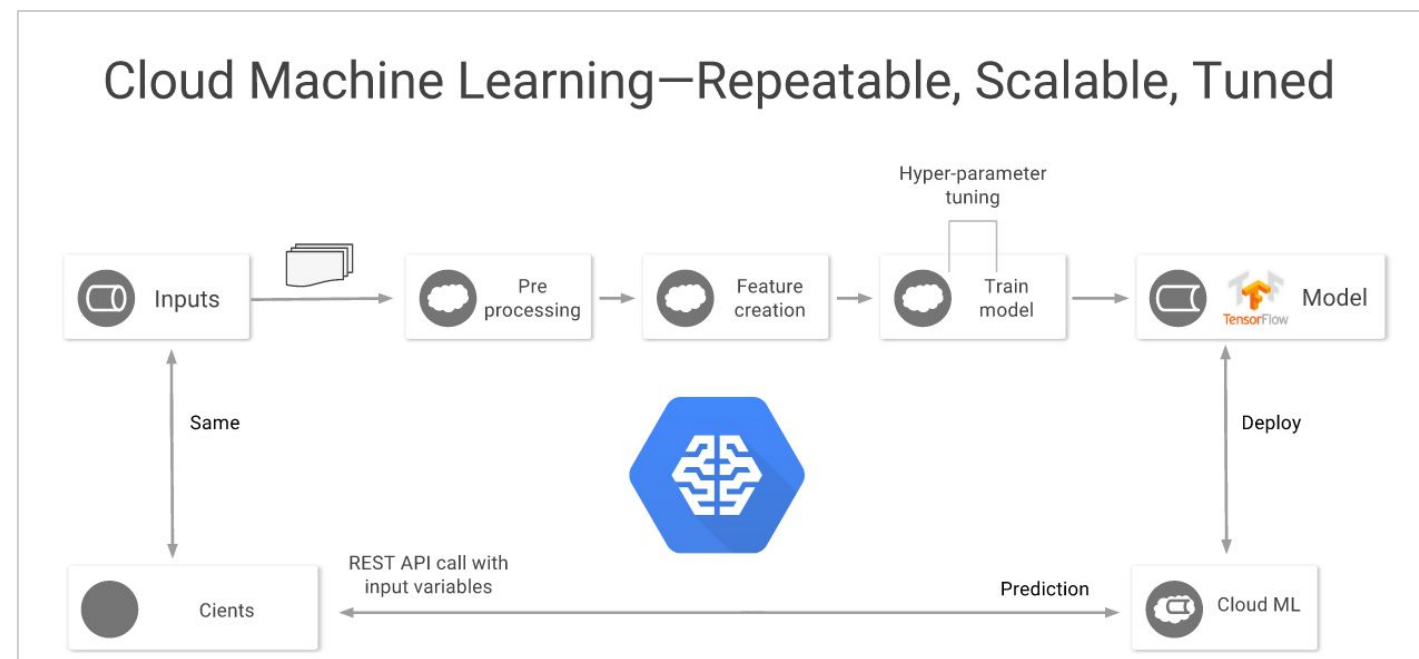


Lab: Scaling TensorFlow with Cloud Machine Learning Engine



Package up TensorFlow model

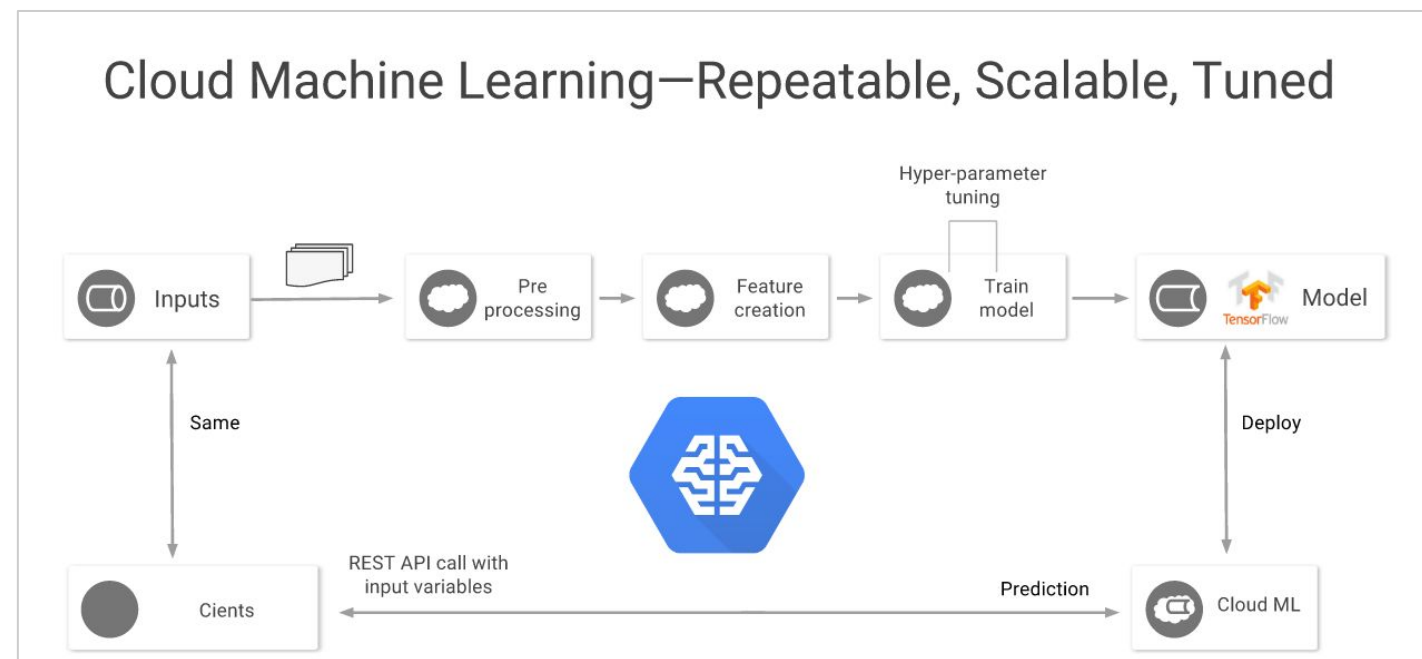
Lab: Scaling TensorFlow with Cloud Machine Learning Engine



Package up TensorFlow model

Run training locally

Lab: Scaling TensorFlow with Cloud Machine Learning Engine

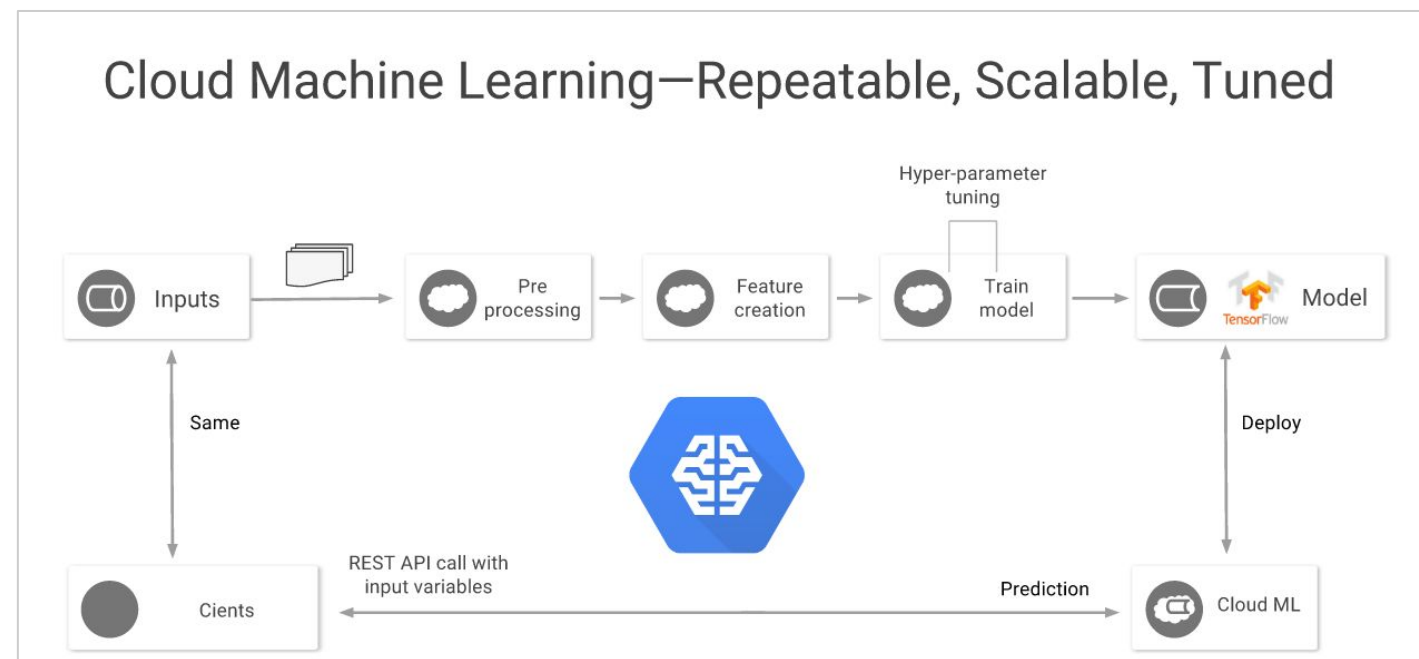


Package up TensorFlow model

Run training locally

Run training on cloud

Lab: Scaling TensorFlow with Cloud Machine Learning Engine



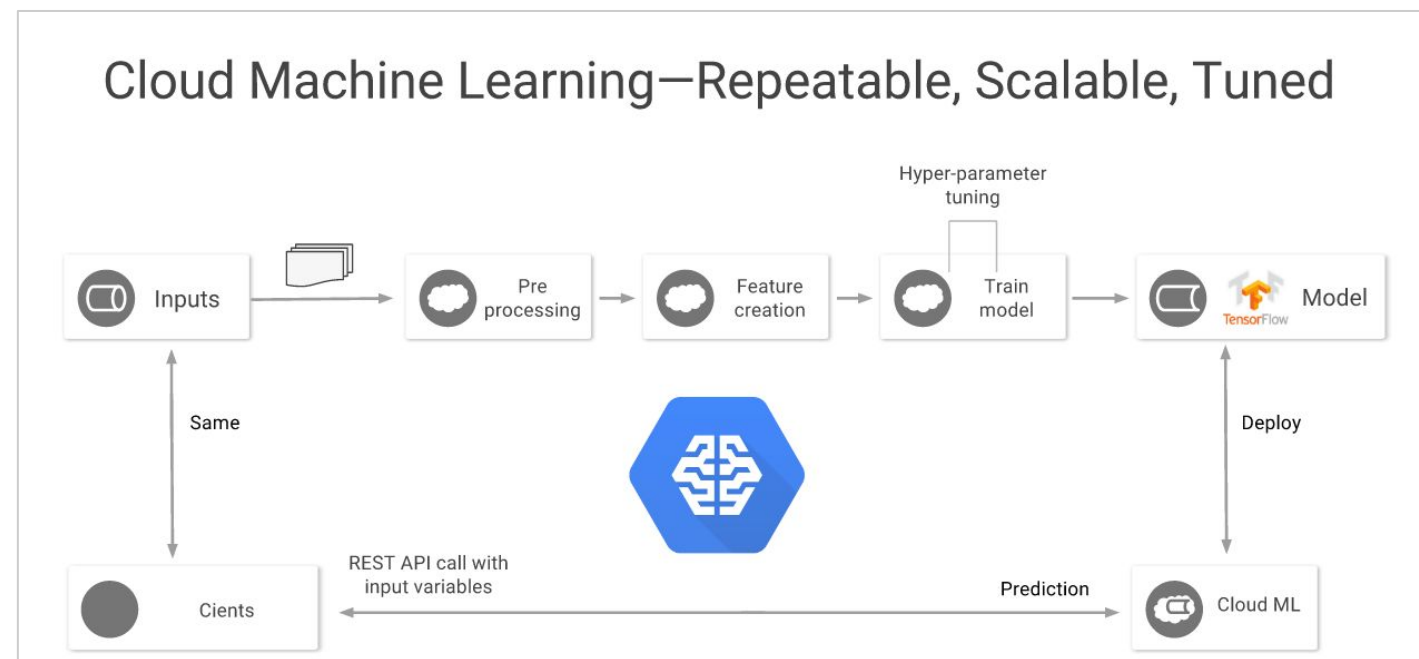
Package up TensorFlow model

Run training locally

Run training on cloud

Deploy model to cloud

Lab: Scaling TensorFlow with Cloud Machine Learning Engine



Package up TensorFlow model

Run training locally

Run training on cloud

Deploy model to cloud

Invoke model to carry out predictions

cloud.google.com