Notes to Presenter:

- These slides are intended for teaching a live TFX workshop
- The source materials for the workshop are available in the tfx_airflow directory in this repo:

  https://github.com/tensorflow/workshops

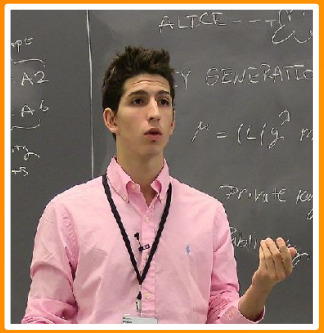
- There are delays during the workshop when you are waiting for things to complete. Plan to fill that time with discussions about the ML tasks and/or TFX components which are relevant to the current stage of the workshop
- For questions, comments, or feedback please feel free to reach out to robertcrowe@google.com
- Thank you for teaching a TFX workshop!

TensorFlow

TFX Workshop

Developing Production ML Pipelines

Robert Crowe
TensorFlow Developer Advocate
@robert_crowe



Pedram Pejman
Technical Program Manager
@unclepeddy

Step 0: Preliminaries

Link to slide download

Download these slides:

https://goo.gle/tfx-workshop-slides

Prerequisites

In the first tutorial we will develop a TFX pipeline in Google Colab:

`https://goo.gle/tfx-london`

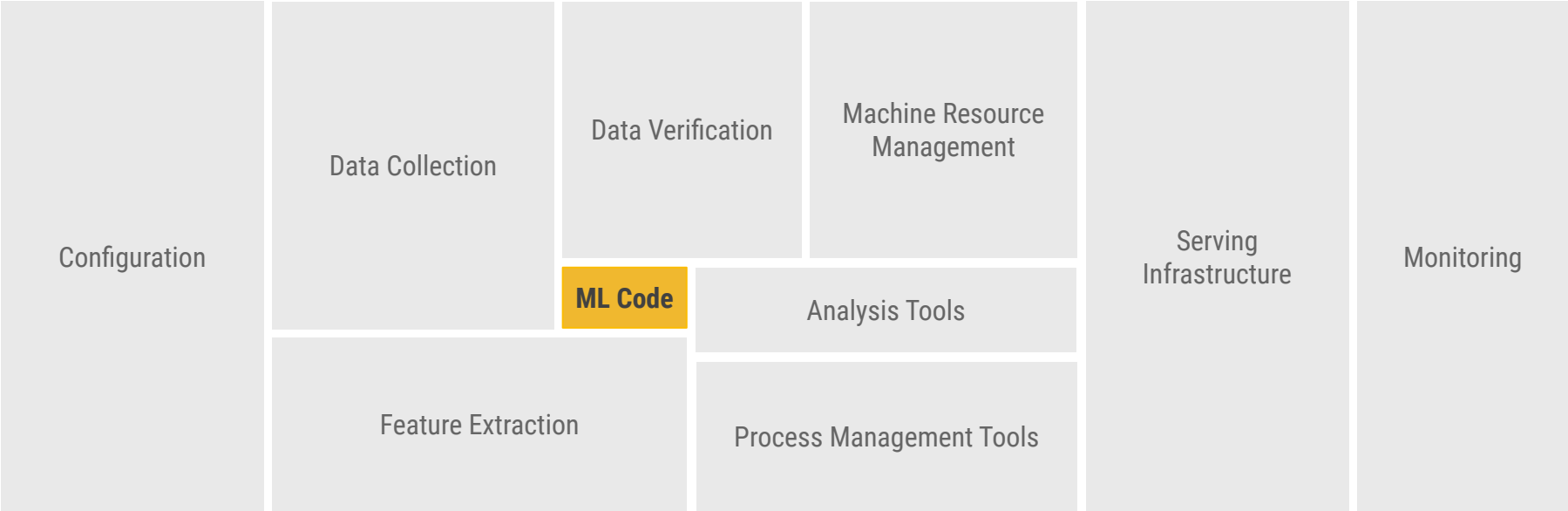
What are we doing here?

What does it all mean?

In addition to training an amazing model ...

Modeling Code

… a production solution requires so much more



What we're doing

We're learning how to create an ML pipeline using TFX

- TFX pipelines are appropriate when:
  - Datasets are large, or may someday get large
  - Training/serving consistency is important
  - Version management for inference is important
- Google uses TFX pipelines for everything from single-node to large-scale ML training and inference

What we're doing

We're following a typical ML development process

- Understanding our data
- Feature engineering
- Training
- Analyze model performance
- Lather, rinse, repeat
- Ready to deploy to production

TFX End-to-End Example

Chicago Taxi Cab Dataset

TFX End-to-End Example

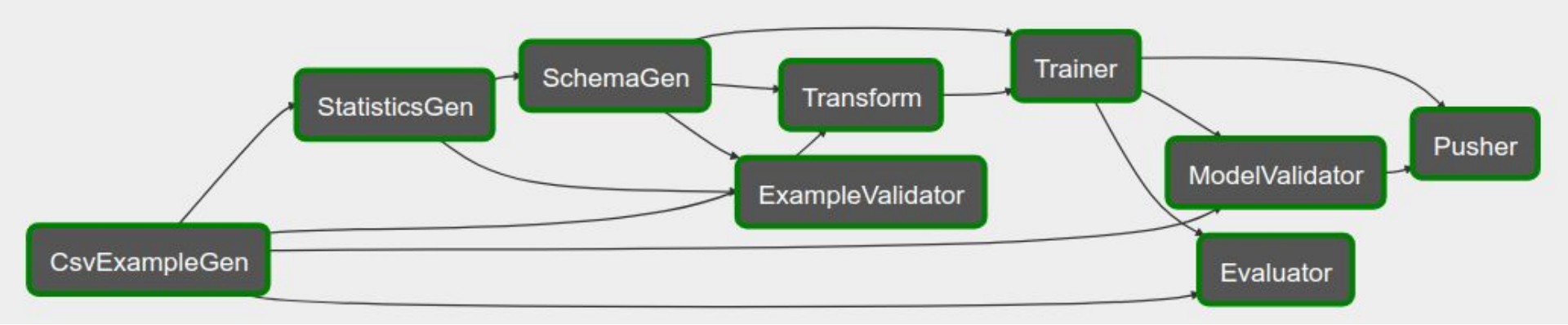
Chicago Taxi Cab Dataset

Features

Categorical Features	Bucket Features	Vocab Features	Dense Float Features
trip_start_hour	pickup_latitude	payment_type	trip_miles
trip_start_day	pickup_longitude	company	fare
trip_start_month	dropoff_latitude		trip_seconds
pickup/dropoff_census_tract	dropoff_longitude		
pickup/dropoff_community_area			

Label = tips > (fare * 20%)

TFX End-to-End Example



Step 1: Setup

Step 2: Dive into our data

It’s all about the data

Data Exploration & Cleanup

The first task in any data science or ML project is to understand and clean the data

- Understand the data types for each feature
- Look for anomalies and missing values
- Understand the distributions for each feature

Components in this step

ExampleGen

- Converts input data to tf.Example

StatisticsGen

- Uses TensorFlow Data Validation (TFDV) to create descriptive statistics for dataset and features

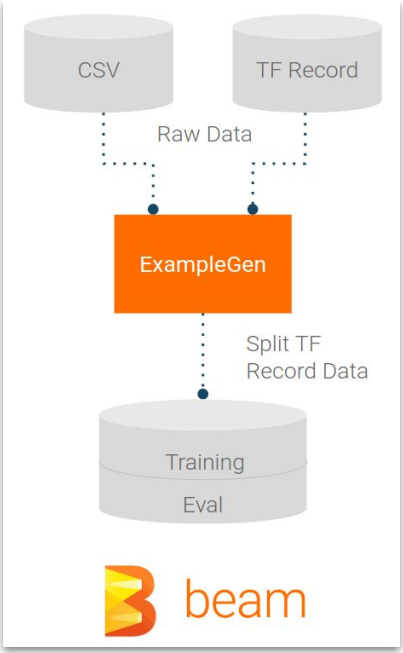
SchemaGen

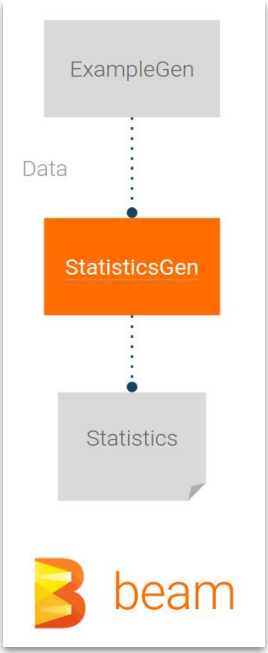
- Uses TensorFlow Data Validation (TFDV) to infer a schema for the dataset

ExampleValidator

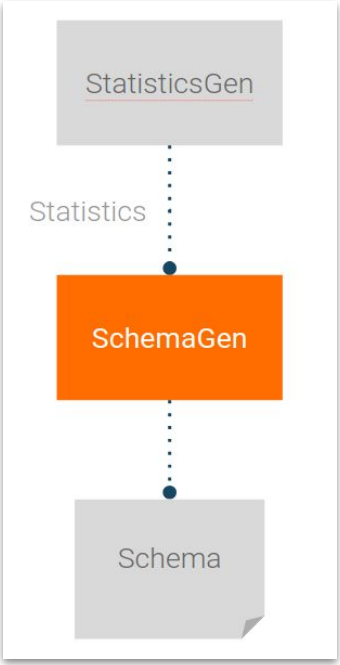
- Uses TensorFlow Data Validation (TFDV) to look for anomalies and missing values

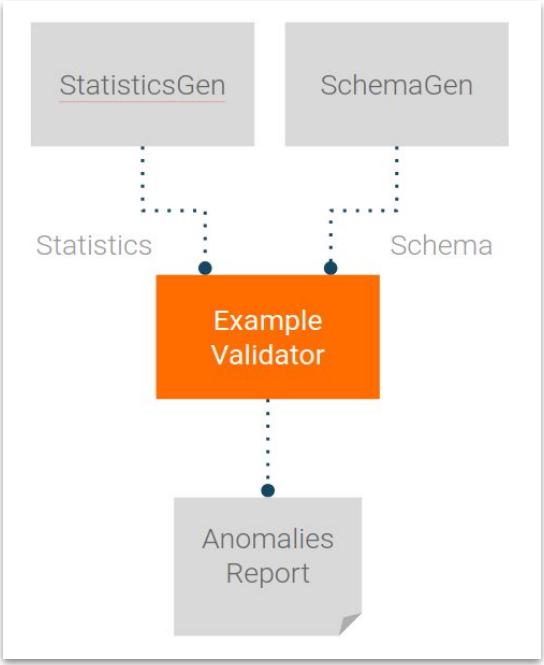
Components in this step





Components in this step





Sort by
Feature order

☐ Reverse order

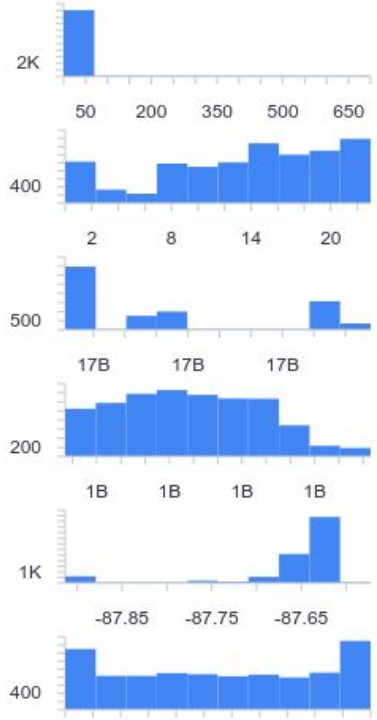
Feature search (regex enabled)

Features: ☑ int(5) ☑ float(5) ☑ variable-length floats(5) ☑ string(1) ☑ fixed-length strings(1) ☑ variable-length strings(1)

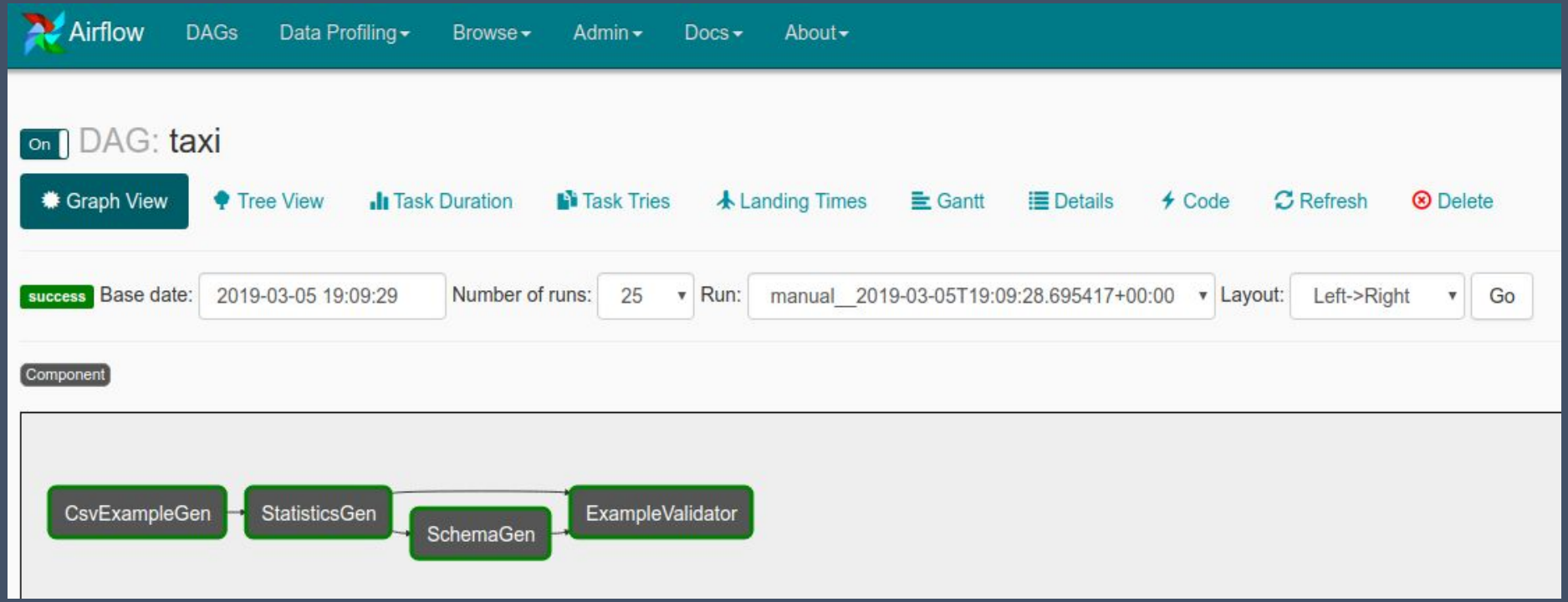
Numeric Features (15)

Chart to show
Standard
☐ log ☐ expand

	count	missing	mean	std dev	zeros	min	median	max
fare	10.1k	0%	11.73	12.18	0.17%	0	7.85	700.07
trip_start_hour	10.1k	0%	13.62	6.59	3.97%	0	15	23
dropoff_census_tract	10.1k	0%	17.0B	330k	0%	17.0B	17.0B	17.0B
trip_start_timestamp	10.1k	0%	1.41B	29.0M	0%	1.36B	1.41B	1.48B
pickup_longitude	10.1k	0%	-87.66	0.07	0%	-87.91	-87.63	-87.57
trip_start_month	10.1k	0%	6.61	3.4	0%	1	7	12



This is what the DAG would look like in Apache Airflow



Step 3: Feature engineering

Squeezing the most from our data

Step 3: Feature Engineering

We can increase the predictive quality of our data and/or reduce dimensionality with feature engineering

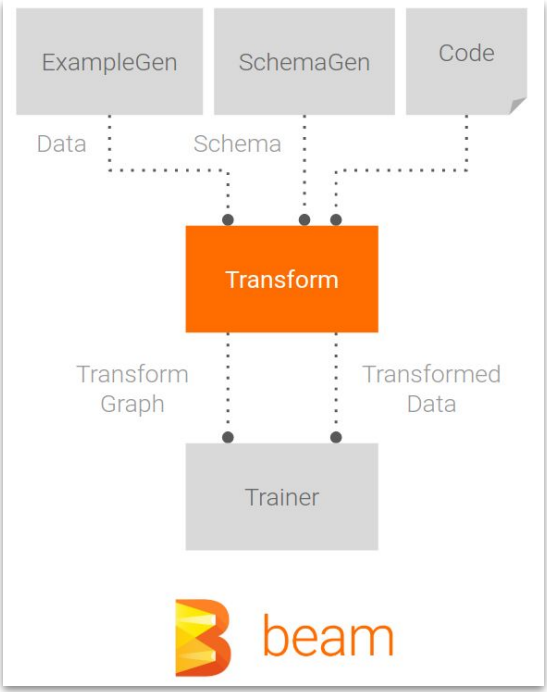
- Feature crosses
- Vocabularies
- Embeddings
- PCA
- Categorical encoding

Write Once - The resulting transforms will be consistent between training and serving

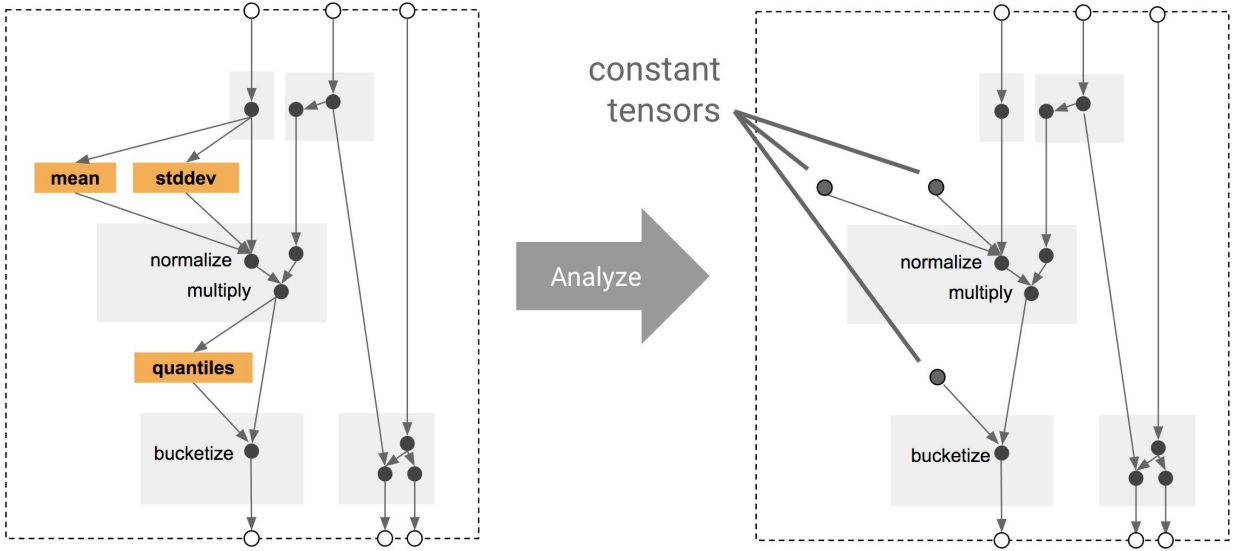
Components in this step

Transform

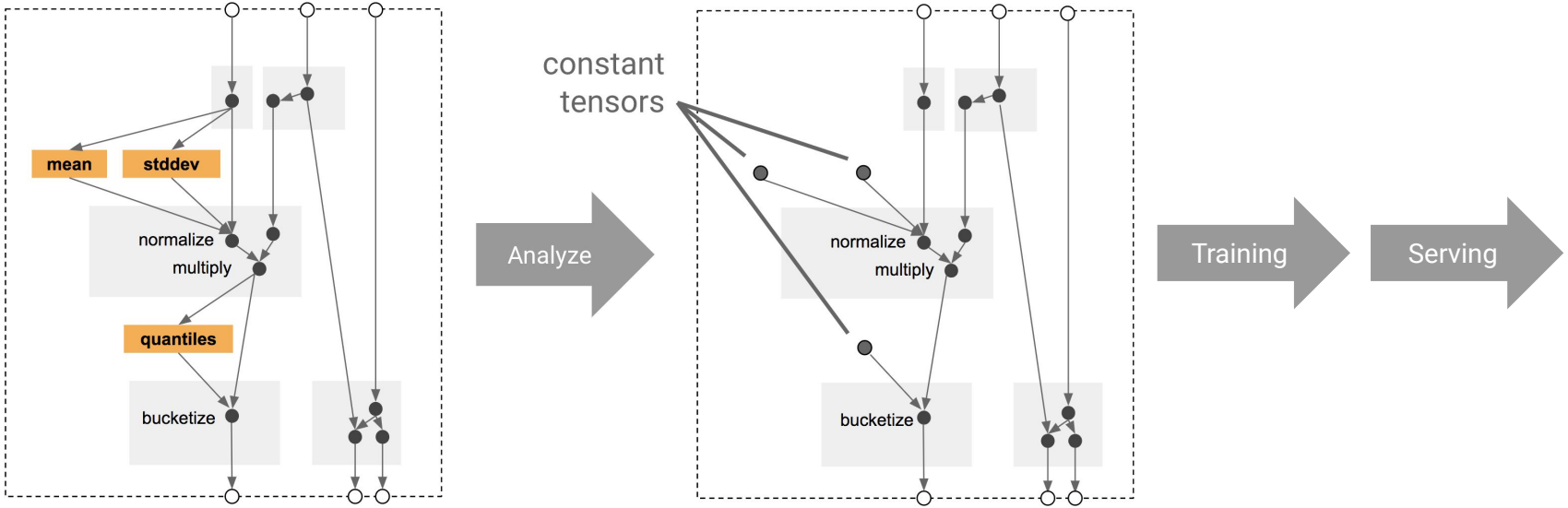
- Uses TensorFlow Transform (TFT) to perform data transformations



TensorFlow Transform



TensorFlow Transform



Step 4: Training a Model

Step 5: Training a Model

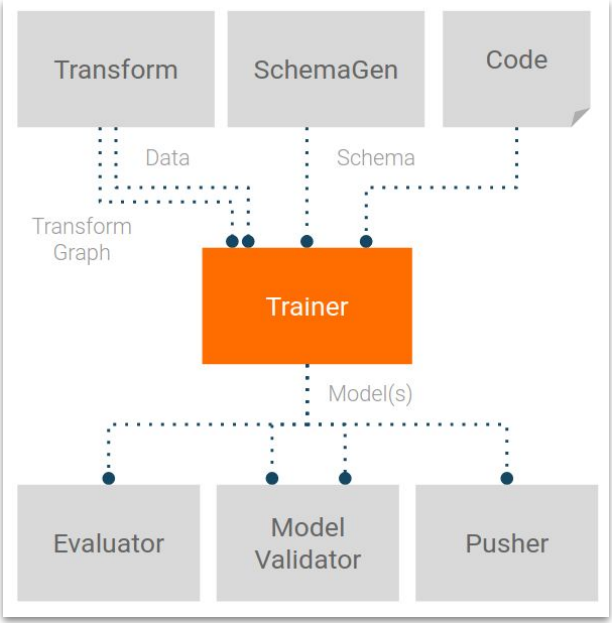
Train a TensorFlow model with our nice, clean, transformed data

- Include the transformations from step 4 so that they are applied consistently
- Save the results as a SavedModel for production
- Visualize and explore the training process using TensorBoard
- Also save an EvalSavedModel for analysis of model performance

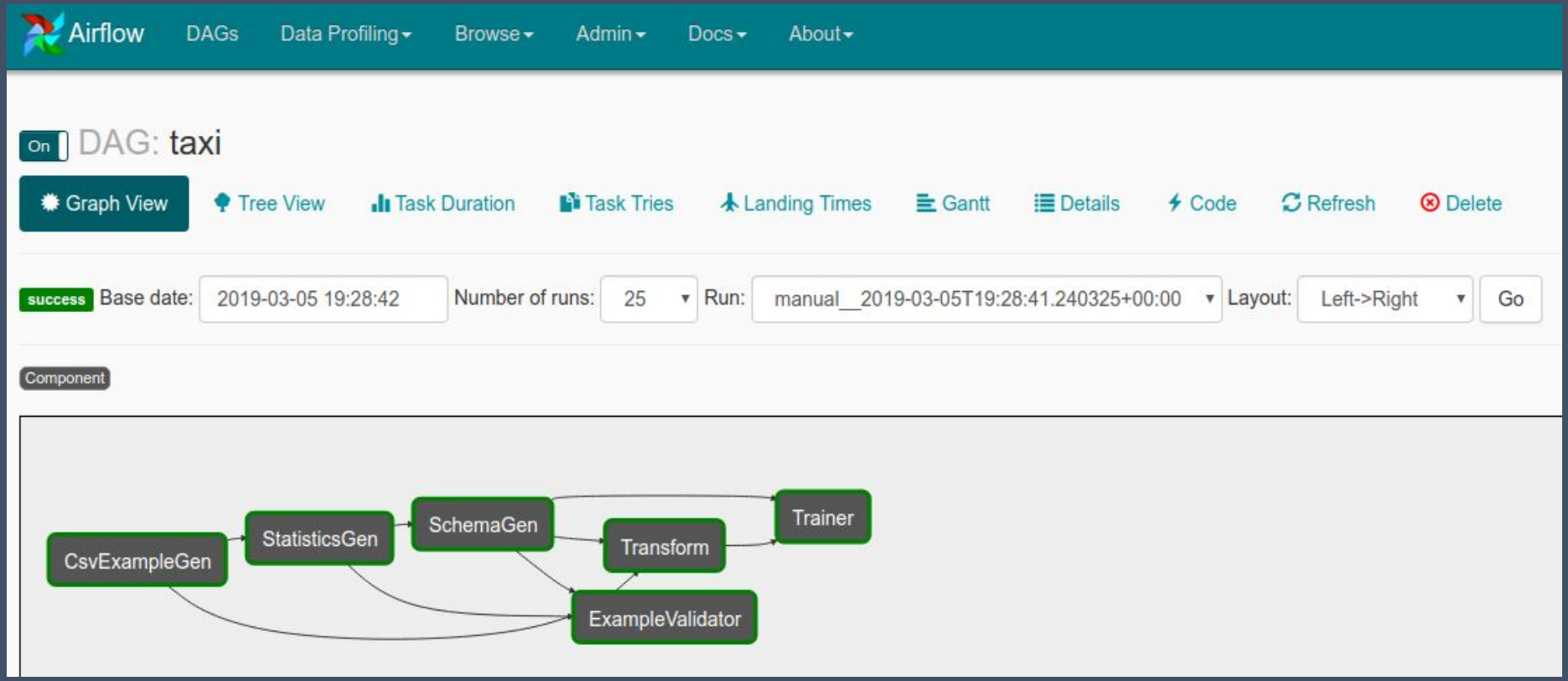
Components in this step

Trainer

- Orchestrates the training of a TensorFlow model



This is what the DAG would look like in Apache Airflow



TensorBoard SCALARS GRAPHS DISTRIBUTIONS HISTOGRAMS PROJECTOR

Search nodes. Regexes supported.

Fit to Screen

Download PNG

Run (2) model_11/serving_model_dir

Session runs (0)

Upload Choose File

Trace inputs

Color Structure

Device

XLA Cluster

Compute time

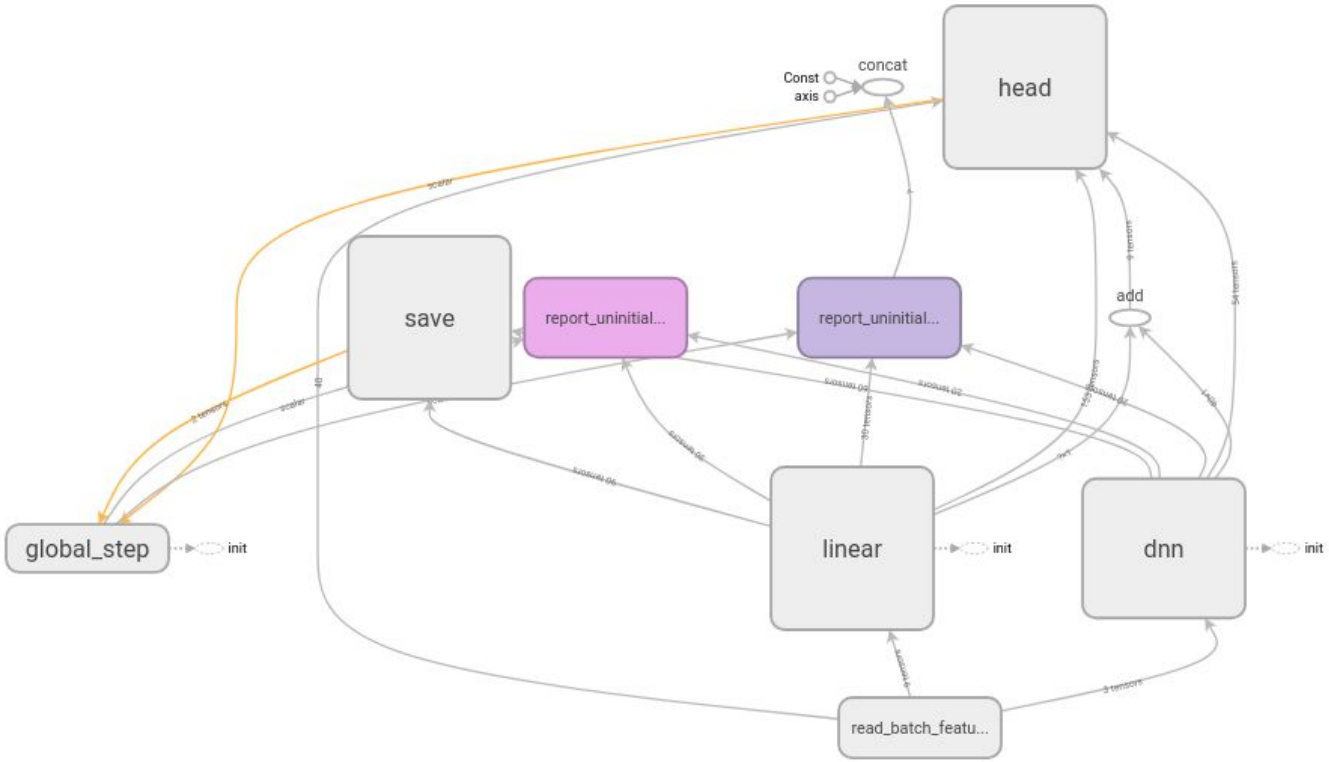
Memory

TPU Compatibility

colors same substructure

unique substructure

Main Graph



Step 5: Analyzing Model Results

Dig deeper

Step 5: Analyzing Model Results

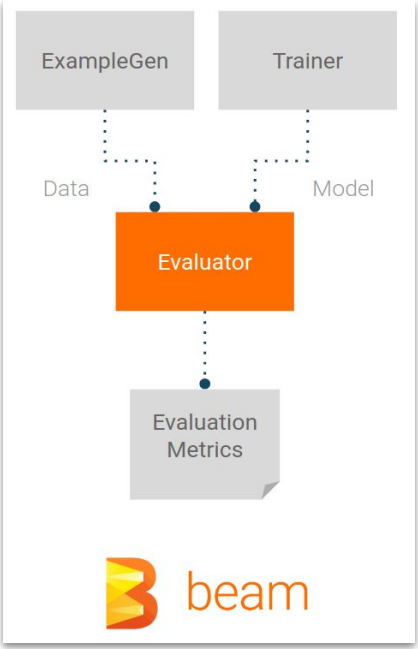
Understanding more than just the top level metrics

- Users experience model performance for their queries only
- Poor performance on slices of data can be hidden by top level metrics
- Model fairness is important
- Often key subsets of users or data are very important, and may be small
    - Performance in critical but unusual conditions
    - Performance for key audiences such as influencers

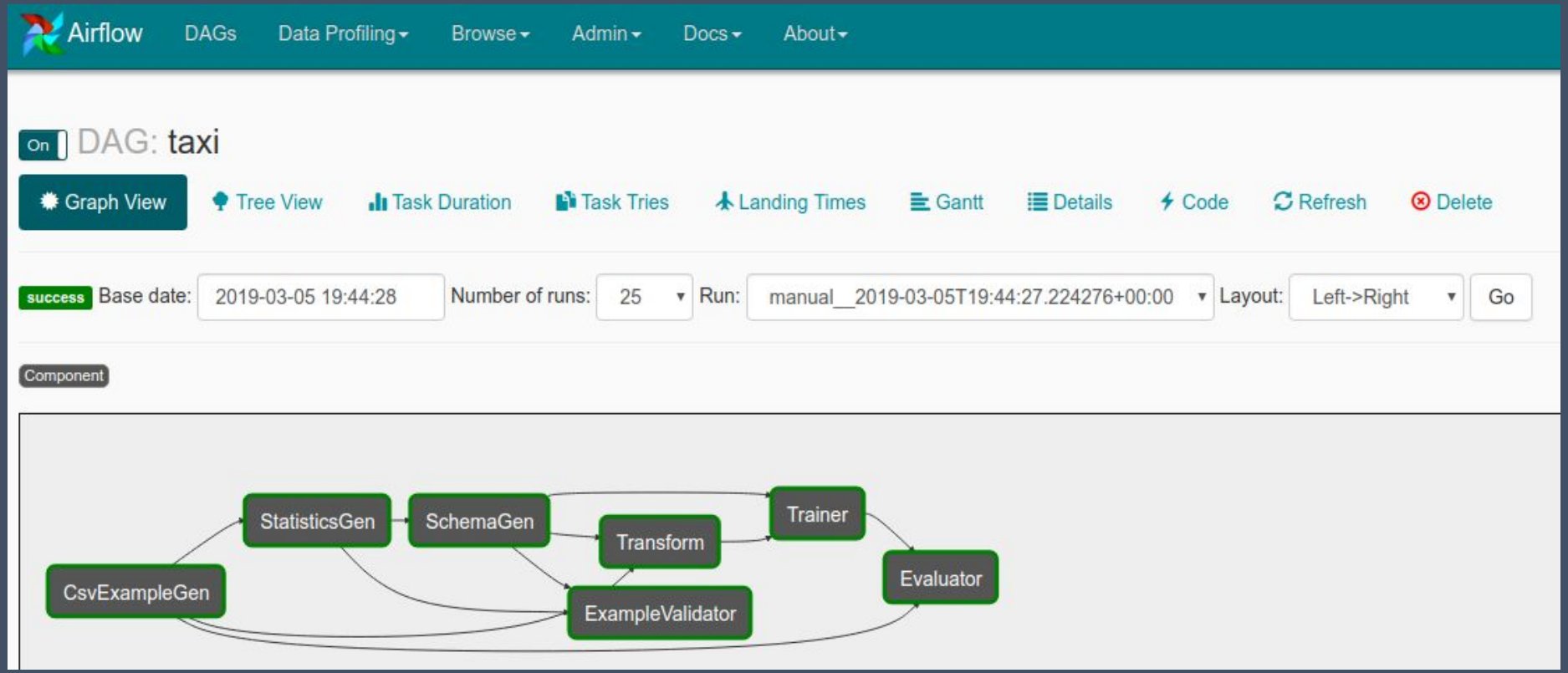
Components

Evaluator

- Uses TensorFlow Model Analysis to perform deep analysis of the performance of the model that we trained



This is what the DAG would look like in Apache Airflow



Examples (Weighted) Threshold

0

Visualization

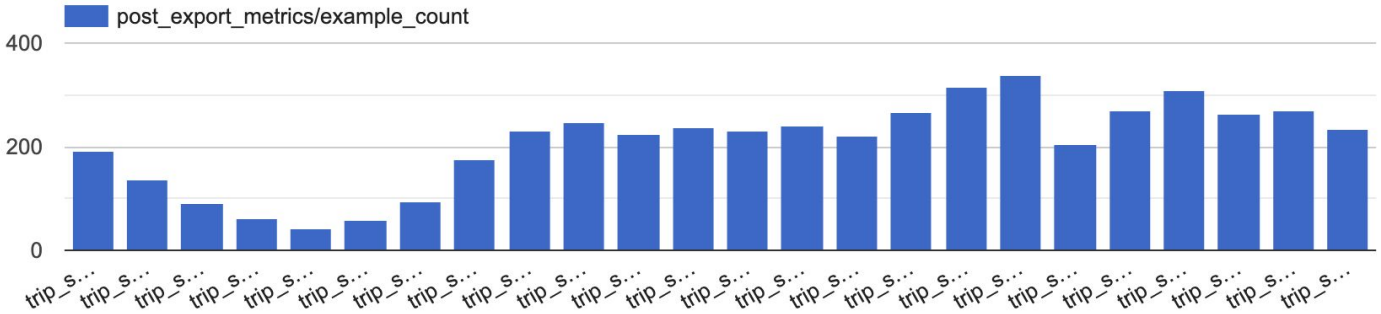
Slices Overview

Show

post_export_metrics/example_count

Sort by

Slice



feature	accuracy	accuracy_baseline	auc	auc_precision_recall	average_loss
trip_start_hour:2	0.82609	0.81884	0.95133	0.71668	0.30771
trip_start_hour:7	0.83158	0.82105	0.95211	0.67722	0.29082
trip_start_hour:12	0.78059	0.75527	0.96740	0.82035	0.32535

Step 6: Production Readiness

Validating model readiness and managing production readiness

Step 6: Production Readiness

If the new model is ready, make it so

- If we're replacing a model that is currently in production, first make sure that the new one is better
  - ModelValidator tells Pusher if the model is OK
- Pusher deploys SavedModels to well-known locations

Step 6: Production Readiness

Deployment targets receive new models from well-known locations

- TensorFlow Serving
- TensorFlow Lite
- TensorFlow JS
- TensorFlow Hub

We will only showcase deployment to Tensorflow Serving today

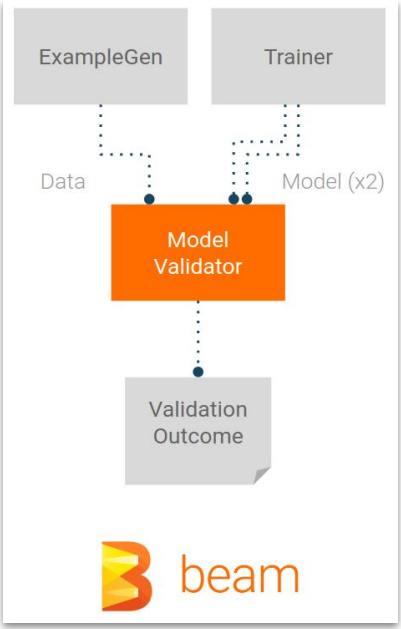
Components

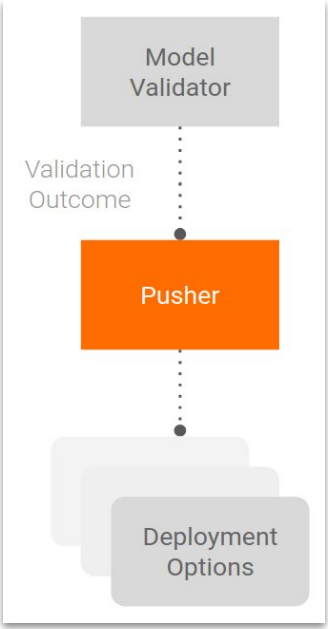
ModelValidator

- Compares multiple versions of the trained model to make sure that the new version meets requirements

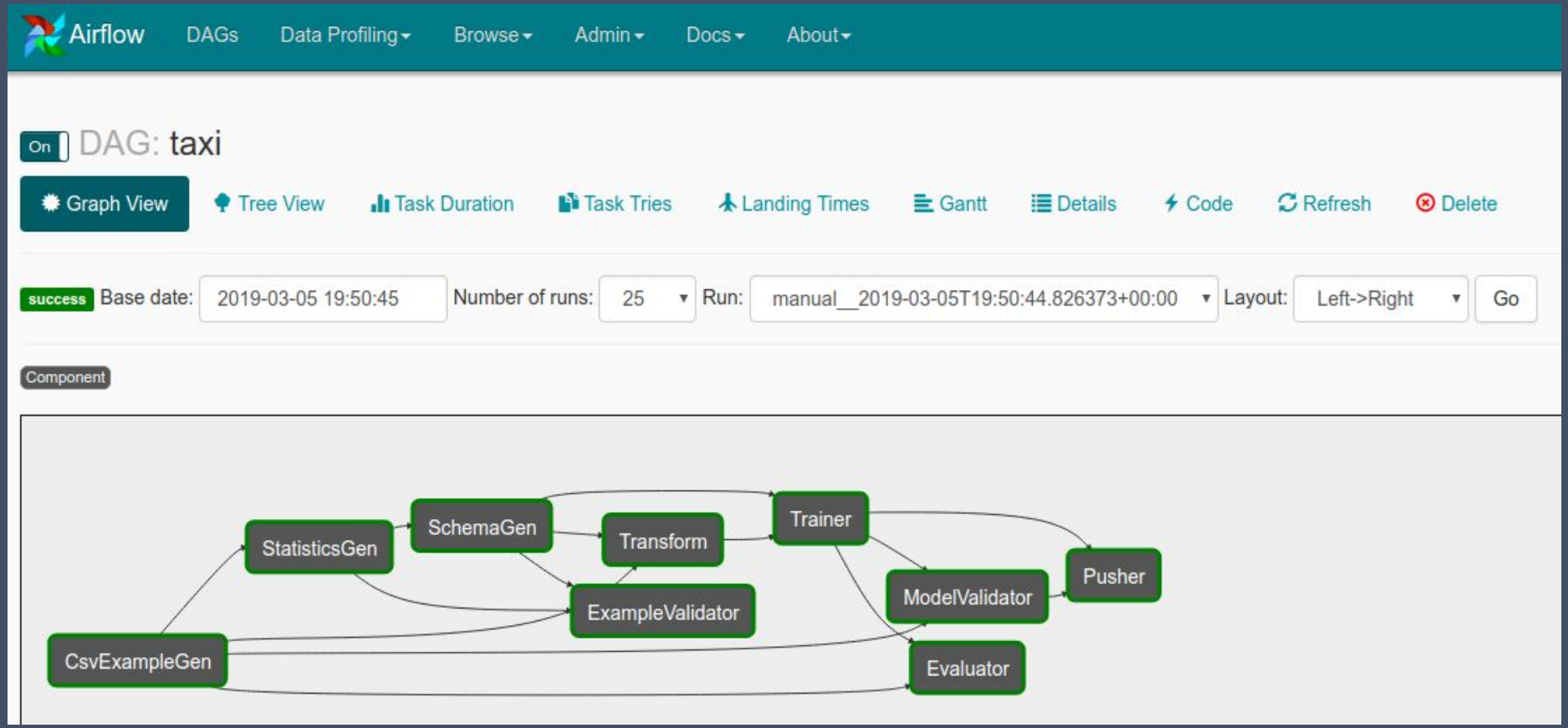
Pusher

- If the model passes ModelValidator, Pusher deploys the SavedModel to a well-known location





This is what the DAG would look like in Apache Airflow



Step 7: Deploy to TF Serving and Query the model

Deploying Tensorflow Serving, instructing it to serve our model and sending example requests

TensorFlow Serving

- Multi-tenant, highly optimized serving system for TF models
- Provides a gRPC or REST API for models with support for versioning
- Low latency and high throughput with inter-request batching
- Used for years at Google supporting millions of QPS

Step 7: Deploy to TF Serving and query the live model

- The pusher has pushed the SavedModel to the filesystem of the container
- The filesystem of the container is mapped via a volume to the host
- We will launch TF Serving in its own container, with the directory containing the SavedModel available mapped in
- We will use a Jupyter notebook to query the model
- This will work because both containers will be sharing the host's network interface (so localhost:8501 is the same in both containers)

Next Steps

You have now deployed your model to Tensorflow Serving and queried it. Using the SavedModel file under the airflow/saved_models/taxi directory, you can now deploy your model to any of the TensorFlow deployment targets, including:

- TensorFlow Lite, for including your model in an Android or iOS native mobile application, or in a Raspberry Pi, IoT, or microcontroller application.
- TensorFlow.js, for running your model in a web browser or Node.JS application.

More Advanced Examples

The examples presented here are really only meant to get you started. For more advanced examples see:

`https://www.tensorflow.org/tfx/tutorials`

- TensorFlow Data Validation Colab
- TensorFlow Transform Colab
- TensorBoard Tutorial
- TFMA Chicago Taxi Tutorial

More Information

Website: https://www.tensorflow.org/tfx

TFX Repo: https://github.com/tensorflow/tfx

MLMD Repo: https://github.com/google/ml-metadata

TFX Discussion Group: https://goo.gle/2OGjw78

TFX YouTube: https://goo.gle/2xVkwt4

Docker-based Tutorial

Prerequisites

- Docker
- Git
- Google Chrome
- Minimum 3GB free disk space

If possible, attendees should pull this Docker image before class:

```
docker pull gcr.io/tfx-oss-public/tfx-workshop:latest
```

Windows

If you're using Windows:

- Must use Windows 10 Pro or better (Not home version)
- Use the **Git Bash** tool (aka MINGW64) which comes with Git for Windows to run the start_container.sh script
- More information and tips in windows_notes.md

Step 1: Setup

Step 0: Pull Docker Container

- Adjust the Docker memory and CPU in Preferences > Advanced
  - Recommended Minimum: 8GB memory, 4 CPUs

```
# If haven't already done the docker pull, do it now
    docker pull gcr.io/tfx-oss-public/tfx-workshop:latest
```

Step 1: Start Workshop in Container

```
# Clone TensorFlow Workshops and start Docker container
    git clone https://github.com/tensorflow/workshops.git
    cd workshops/tfx_airflow
    source start_container.sh
# Wait for the prompt 'root@<hex id>:/#' and enter:
    source setup_demo.sh
```

- Wait for the prompt to create a password that you will use for Jupyter (or just hit return for no password)

Exercise and Solution

The workshop includes an exercise and the solution, in case you get stuck

- Directory: `workshops/tfx_airflow/workshop/airflow/dags`

Exercise:

- taxi_pipeline.py
- taxi_utils.py
- taxi DAG

Solution:

- taxi_pipeline_solution.py
- taxi_utils_solution.py
- taxi_solution DAG

Performing Each Step

- All the pipeline code is included in the files under the **workshop/airflow/dags** directory on your system
  - That directory is mounted in the container, so changes you make on your system are reflected inside the container also
- **You do not need to edit files inside the Docker container**

Performing Each Step

Remove comments in Python source code in **taxi_pipeline.py** and **taxi_utils.py**

- Steps 3-7 is commented out and marked with inline comments. The inline comments identify which step the line of code applies to. For example, the code for step 3 is marked with the comment `# Step 3`.

Code falls into 3 regions:

- imports
- The DAG configuration
- The list returned from the create_pipeline() call
- The supporting code in taxi_utils.py

Performing Each Step

As you go through the tutorial you will …

- Uncomment the lines of code that apply to the tutorial step that you're currently working on
- Uncomment the lines of code will add the code for that step, and update your pipeline

We strongly encourage you to review the code that you're uncommenting

Step 2:
Bring up initial pipeline skeleton

TFX “Hello World” - Start Airflow

Step 2, cont: Bring up initial pipeline skeleton - Start Airflow

On your host, open a browser and go to `http://127.0.0.1:8080`

Enable the taxi DAG

Trigger the taxi DAG

Troubleshooting

If you have any issues with loading the Airflow console in your web browser, then you may have another application running on **port 8080**. That's the default port for Airflow, but you can change it to any other user port that's not being used.

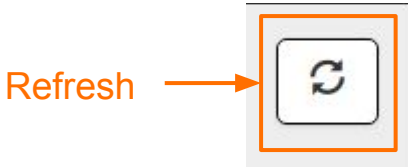
For example, to run Airflow on port 7070 you could run:

```
(tfx_env) airflow webserver -p 7070
```

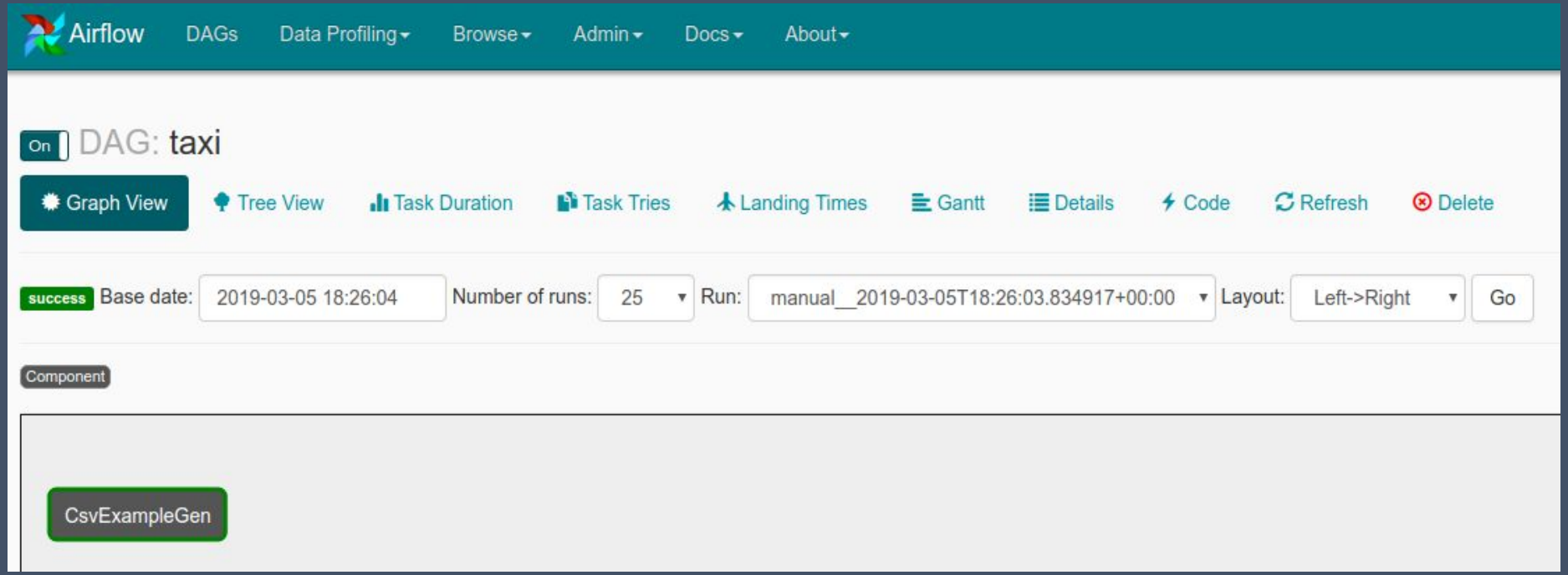
You will need to add the port in docker-compose.yaml and restart the container.

First Run

- In the Airflow DAG view, click on taxi
- Wait for the CsvExampleGen component outline to turn dark green (~1 minutes)
- Use the refresh button on the right or refresh the page



After a few minutes, your DAG should look like this



Step 3: Dive into our data

It’s all about the data

Data Exploration & Cleanup

The first task in any data science or ML project is to understand and clean the data

- Understand the data types for each feature
- Look for anomalies and missing values
- Understand the distributions for each feature

Components in this step

ExampleGen

- Converts input data to tf.Example

StatisticsGen

- Uses TensorFlow Data Validation (TFDV) to create descriptive statistics for dataset and features

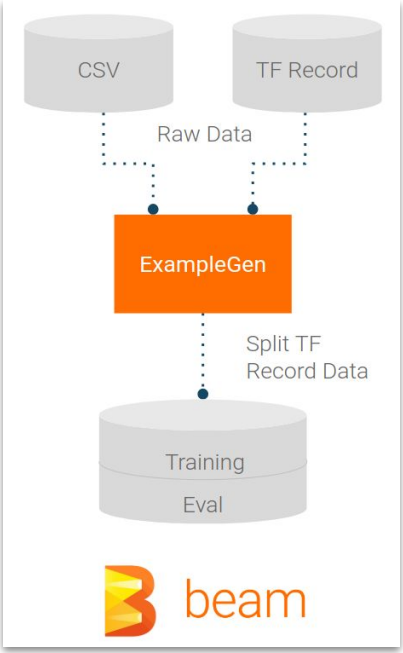
SchemaGen

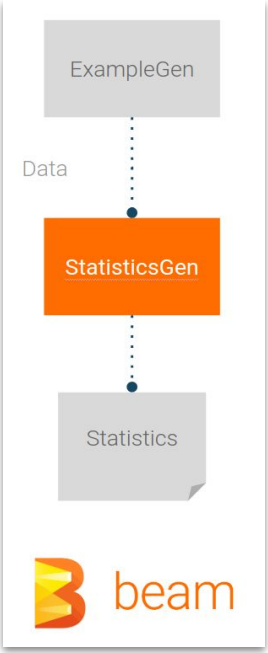
- Uses TensorFlow Data Validation (TFDV) to infer a schema for the dataset

ExampleValidator

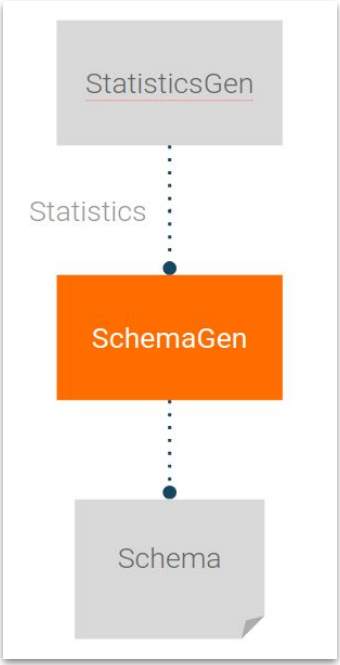
- Uses TensorFlow Data Validation (TFDV) to look for anomalies and missing values

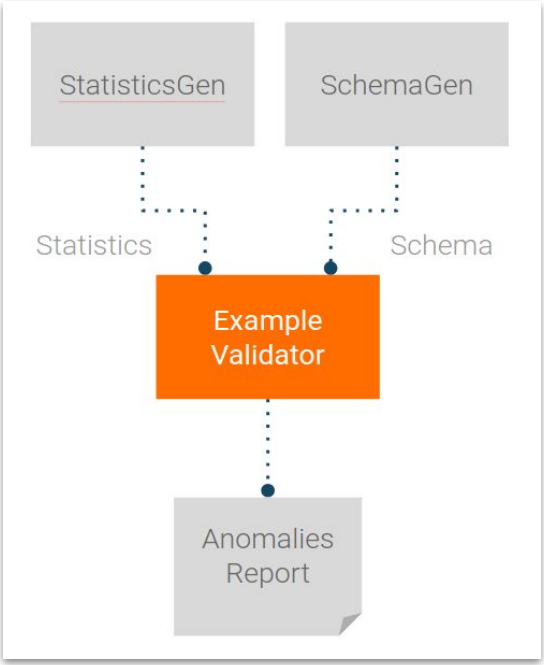
Components in this step





Components in this step





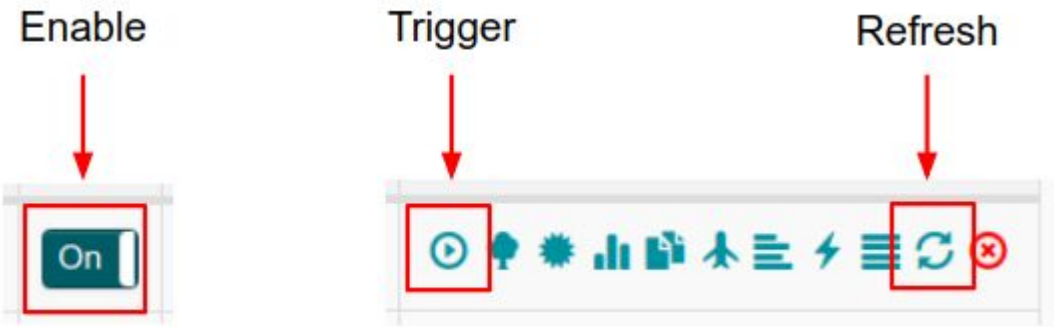
Step 3: Dive into our data

- Search through `taxi_pipeline.py` in `airflow/dags`
  a. Search for “Step 3”
  b. Uncomment the lines marked with Step 3
  c. Take a moment to review the code that you uncommented

Refresh and Trigger

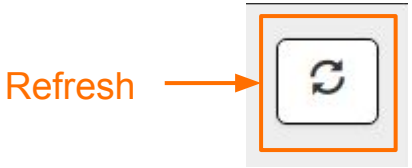
In a browser:

- Return to DAGs list page in Airflow by clicking on "DAGs" link
- Click the refresh button on the right side for the taxi DAG
  - You should see "DAG [taxi] is now fresh as a daisy"
- Trigger taxi

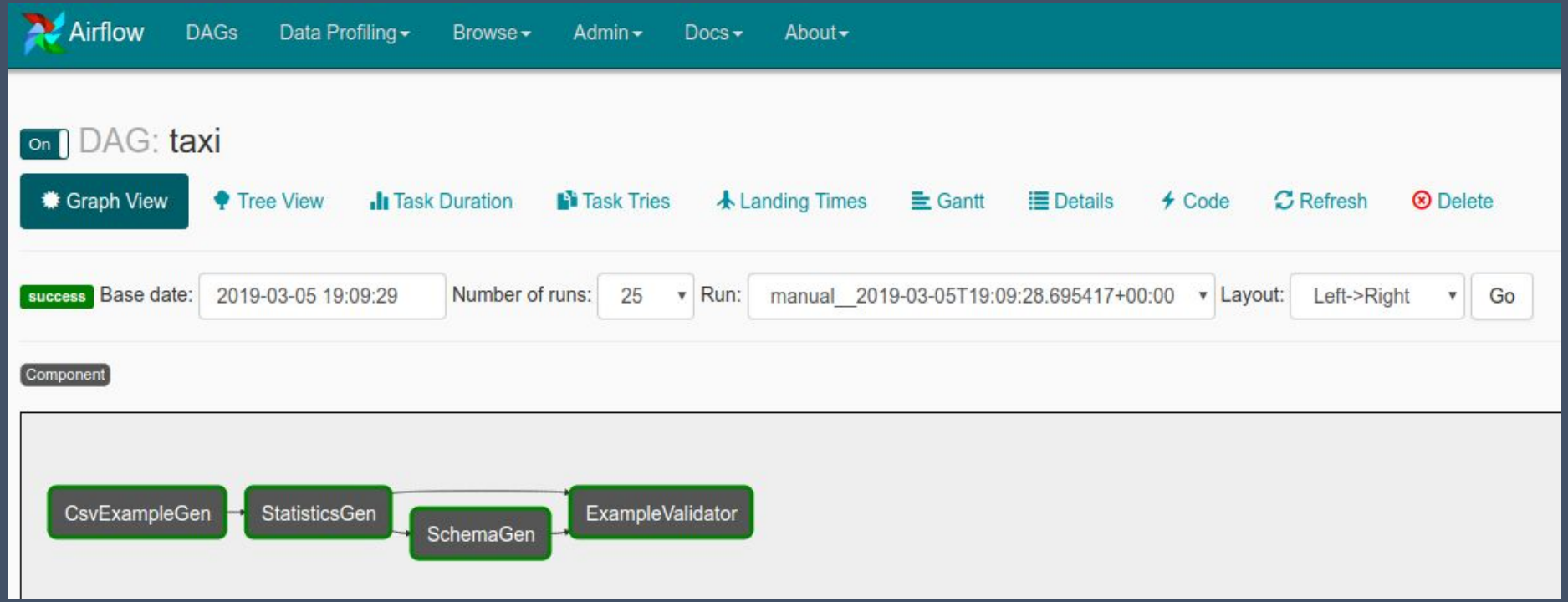


Refresh and Trigger

- In the Airflow DAG view, click on taxi
- Wait for the all components to turn dark green
- Use the refresh button on the right or refresh the page



After a few minutes, your DAG should look like this



Back on Jupyter

- Open step3.ipynb
- Follow the notebook
  - Note that the notebook starts by connecting to the metadata store
  - Artifacts used in the notebook are created by the TFX pipeline that we run in Airflow
  - The notebook accesses the metadata store to read artifacts

Sort by
Feature order

☐ Reverse order

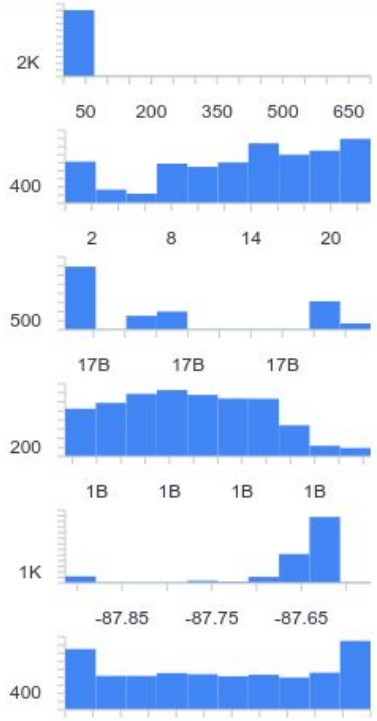
Feature search (regex enabled)

Features: ☑ int(5) ☑ float(5) ☑ variable-length floats(5) ☑ string(1) ☑ fixed-length strings(1) ☑ variable-length strings(1)

Numeric Features (15)

Chart to show
Standard
☐ log ☐ expand

	count	missing	mean	std dev	zeros	min	median	max
fare	10.1k	0%	11.73	12.18	0.17%	0	7.85	700.07
trip_start_hour	10.1k	0%	13.62	6.59	3.97%	0	15	23
dropoff_census_tract	10.1k	0%	17.0B	330k	0%	17.0B	17.0B	17.0B
trip_start_timestamp	10.1k	0%	1.41B	29.0M	0%	1.36B	1.41B	1.48B
pickup_longitude	10.1k	0%	-87.66	0.07	0%	-87.91	-87.63	-87.57
trip_start_month	10.1k	0%	6.61	3.4	0%	1	7	12



Step 4: Feature engineering

Squeezing the most from our data

Step 4: Feature Engineering

We can increase the predictive quality of our data and/or reduce dimensionality with feature engineering

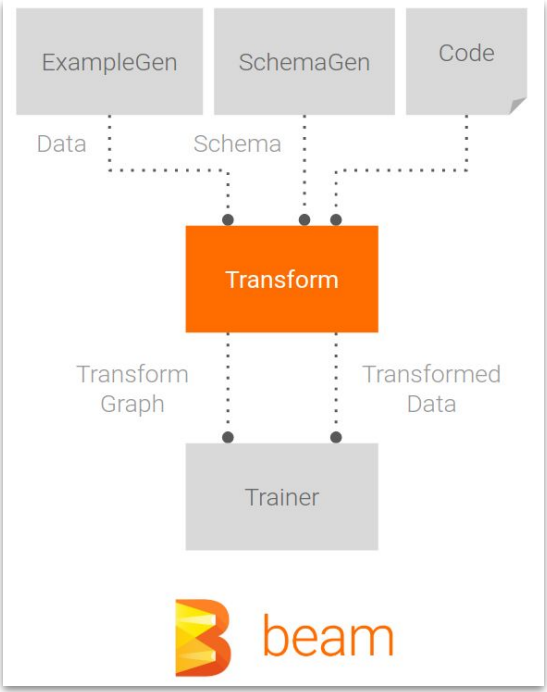
- Feature crosses
- Vocabularies
- Embeddings
- PCA
- Categorical encoding

Write Once - The resulting transforms will be consistent between training and serving

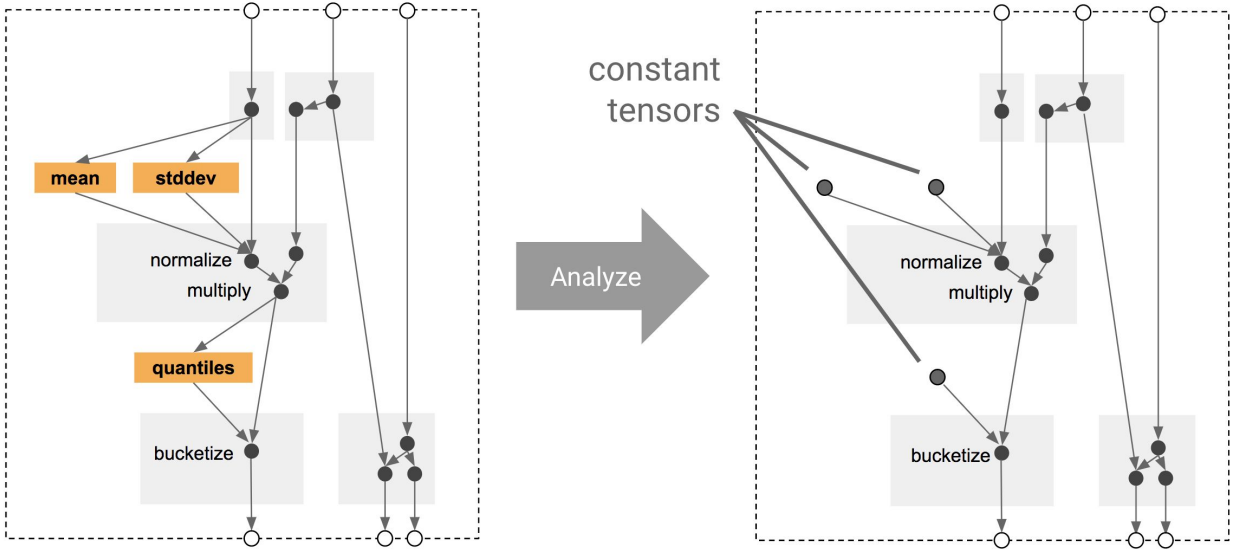
Components in this step

Transform

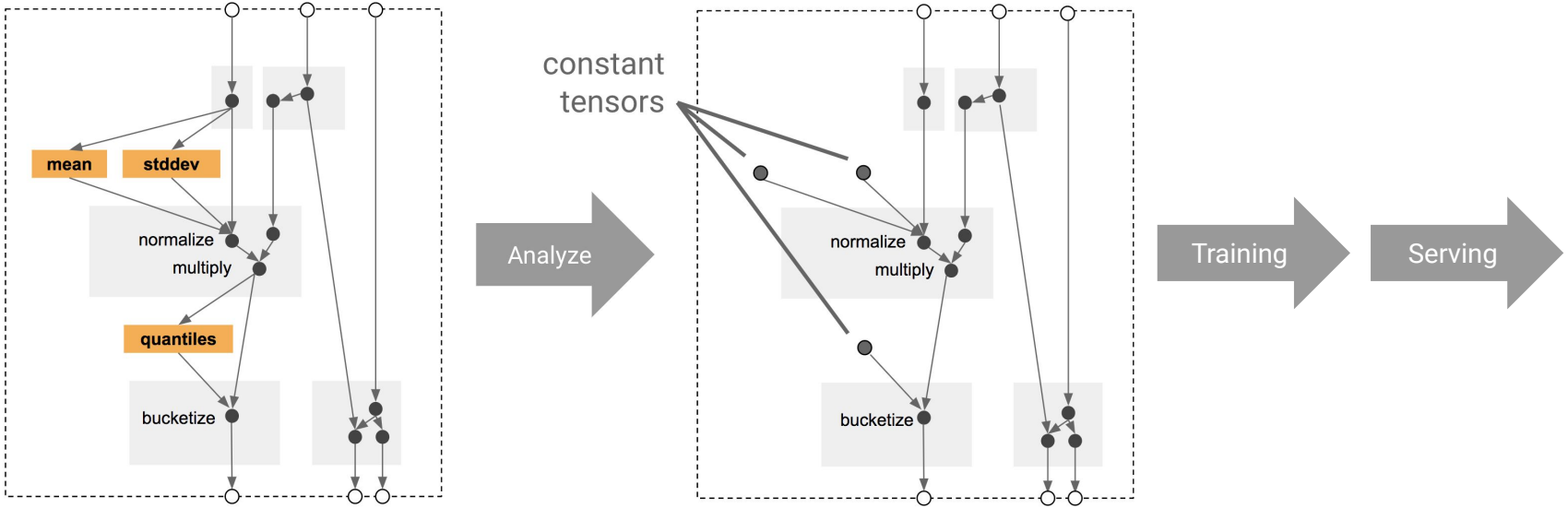
- Uses TensorFlow Transform (TFT) to perform data transformations



TensorFlow Transform



TensorFlow Transform



Step 4: Feature Engineering

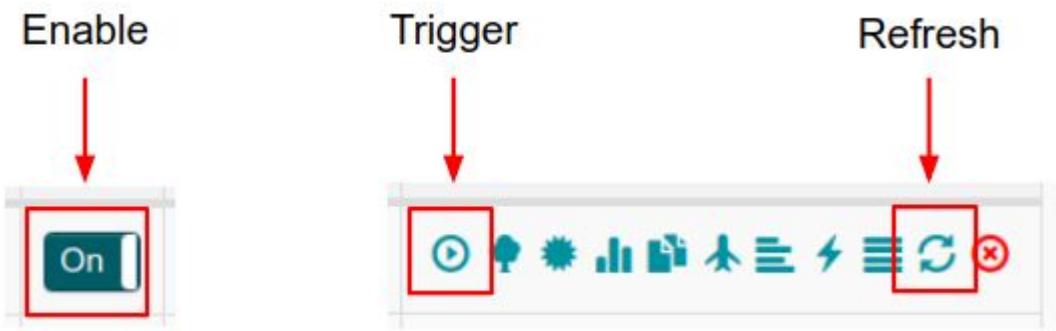
Search through `taxi_pipeline.py` and `taxi_utils.py` in airflow/dags

- Search for “Step 4”
- Uncomment the lines marked with Step 4
- Take a moment to review the code that you uncommented

Refresh and Trigger

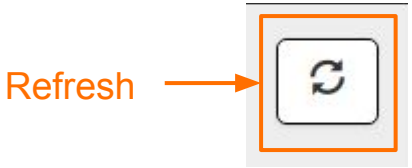
In a browser:

- Return to DAGs list page in Airflow by clicking on "DAGs" link
- Click the refresh button on the right side for the taxi DAG
  - You should see "DAG [taxi] is now fresh as a daisy"
- Trigger taxi

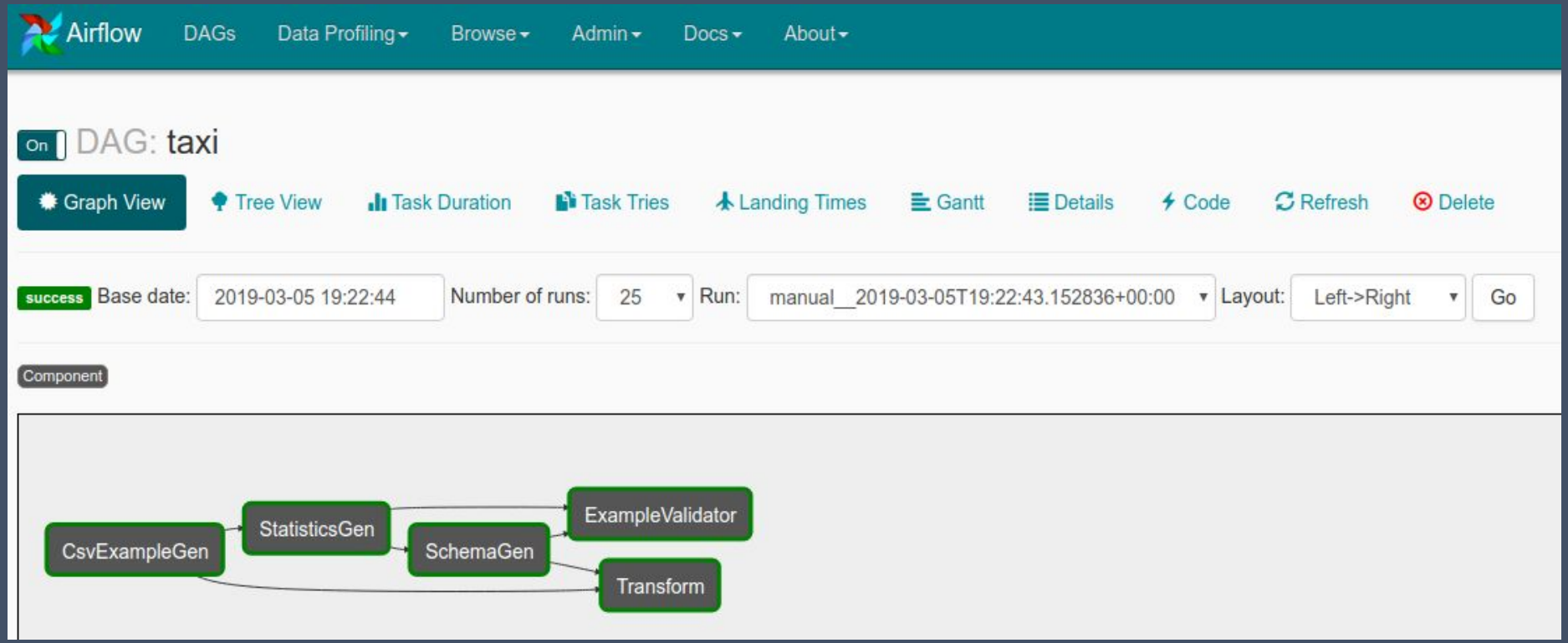


Refresh and Trigger

- In the Airflow DAG view, click on taxi
- Wait for the all components to turn dark green
- Use the refresh button on the right or refresh the page



After a few minutes, your DAG should look like this



Back on Jupyter

- Open step4.ipynb
- Follow the notebook
  - Note that the notebook starts by connecting to the metadata store
  - Artifacts used in the notebook are created by the TFX pipeline that we run in Airflow
  - The notebook accesses the metadata store to read artifacts

Step 5: Training a Model

Step 5: Training a Model

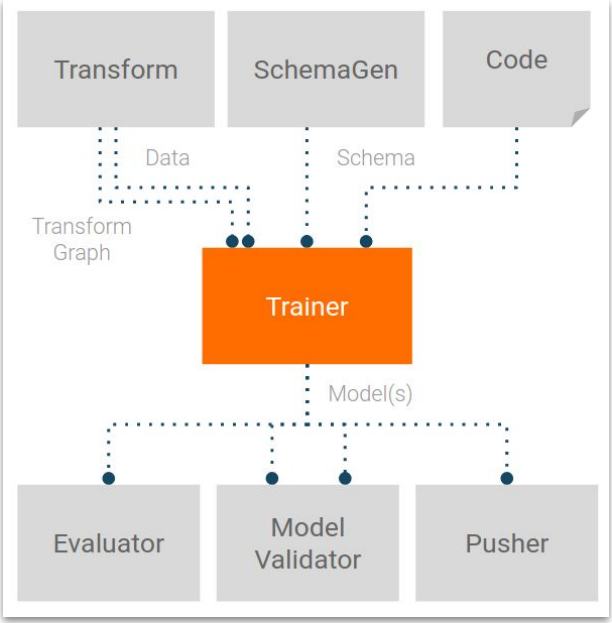
Train a TensorFlow model with our nice, clean, transformed data

- Include the transformations from step 4 so that they are applied consistently
- Save the results as a SavedModel for production
- Visualize and explore the training process using TensorBoard
- Also save an EvalSavedModel for analysis of model performance

Components in this step

Trainer

- Orchestrates the training of a TensorFlow model



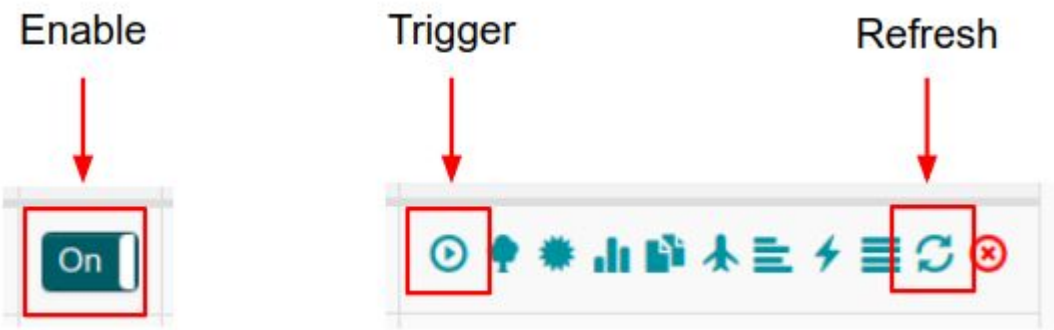
Step 5: Training a Model

- Search through `taxi_pipeline.py` and `taxi_utils.py` in airflow/dags
  a. Search for “Step 5”
  b. Uncomment the lines marked with Step 5
  c. Take a moment to review the code that you uncommented

Refresh and Trigger

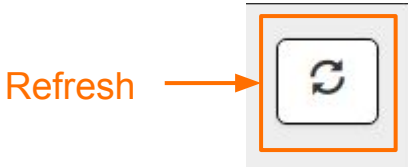
In a browser:

- Return to DAGs list page in Airflow by clicking on "DAGs" link
- Click the refresh button on the right side for the taxi DAG
  - You should see "DAG [taxi] is now fresh as a daisy"
- Trigger taxi

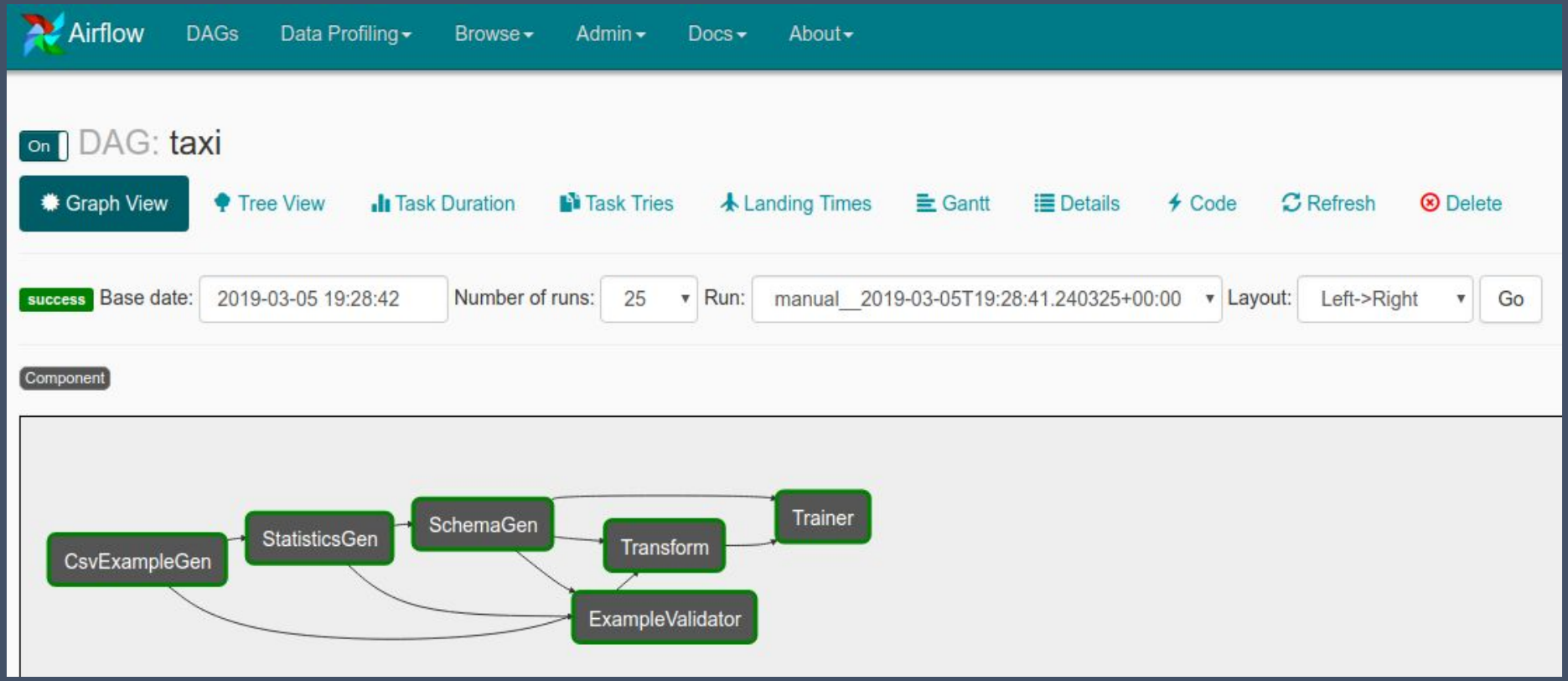


Refresh and Trigger

- In the Airflow DAG view, click on taxi
- Wait for the all components to turn dark green
- Use the refresh button on the right or refresh the page

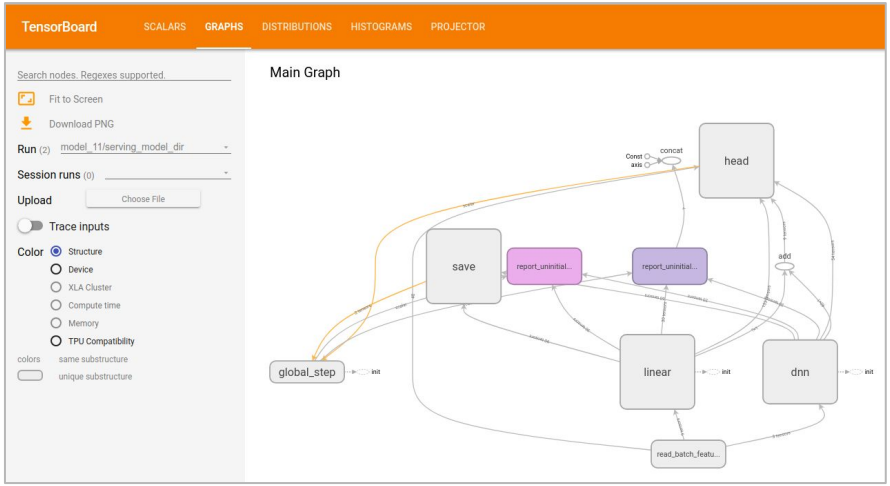


After a few minutes, your DAG should look like this



Back on Jupyter

- Open step5.ipynb
- Follow the notebook
- Go to http://localhost:6006 to open TensorBoard in a browser tab



TensorBoard SCALARS GRAPHS DISTRIBUTIONS HISTOGRAMS PROJECTOR

Search nodes. Regexes supported.

Fit to Screen

Download PNG

Run (2) model_11/serving_model_dir

Session runs (0)

Upload Choose File

Trace inputs

Color Structure

Device

XLA Cluster

Compute time

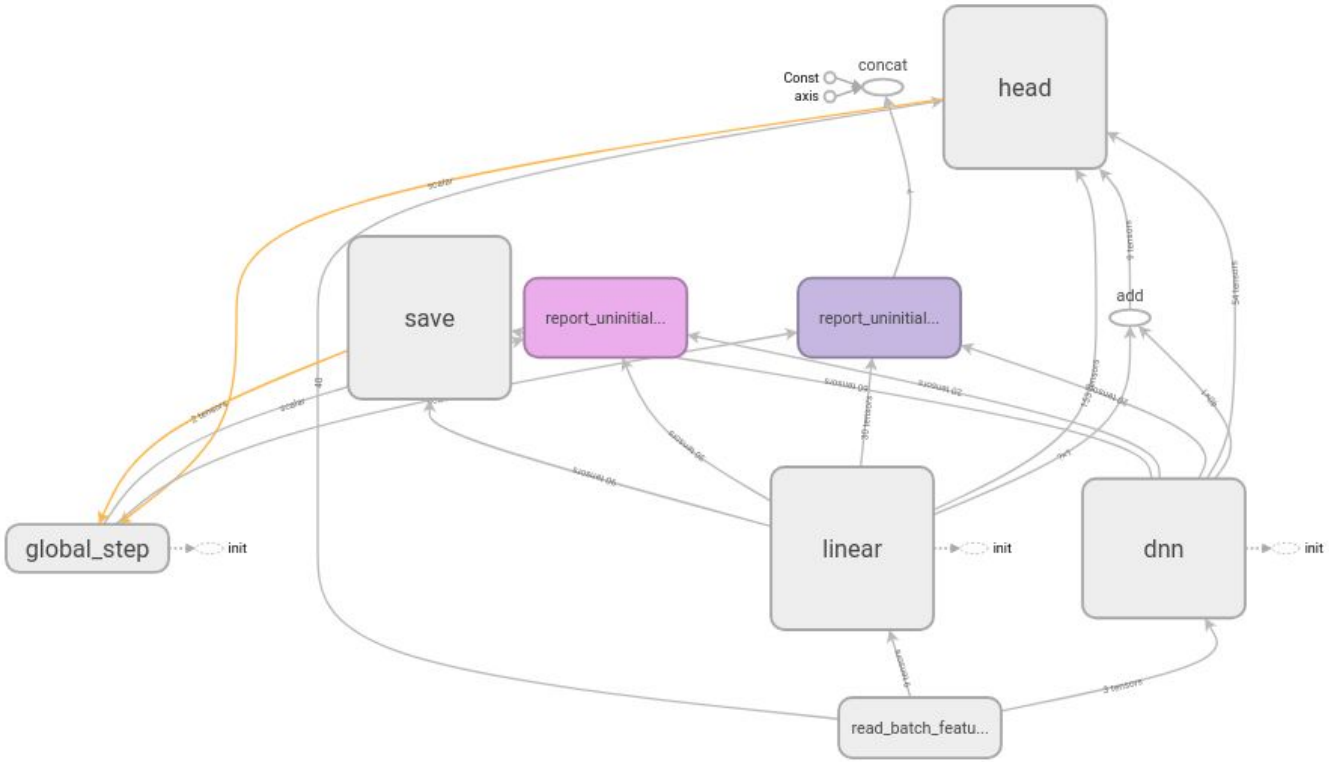
Memory

TPU Compatibility

colors same substructure

unique substructure

Main Graph



Step 6: Analyzing Model Results

Dig deeper

Step 6: Analyzing Model Results

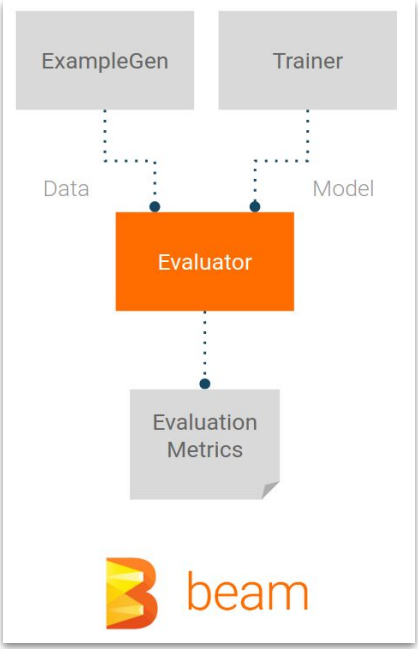
Understanding more than just the top level metrics

- Users experience model performance for their queries only
- Poor performance on slices of data can be hidden by top level metrics
- Model fairness is important
- Often key subsets of users or data are very important, and may be small
    - Performance in critical but unusual conditions
    - Performance for key audiences such as influencers

Components

Evaluator

- Uses TensorFlow Model Analysis to perform deep analysis of the performance of the model that we trained



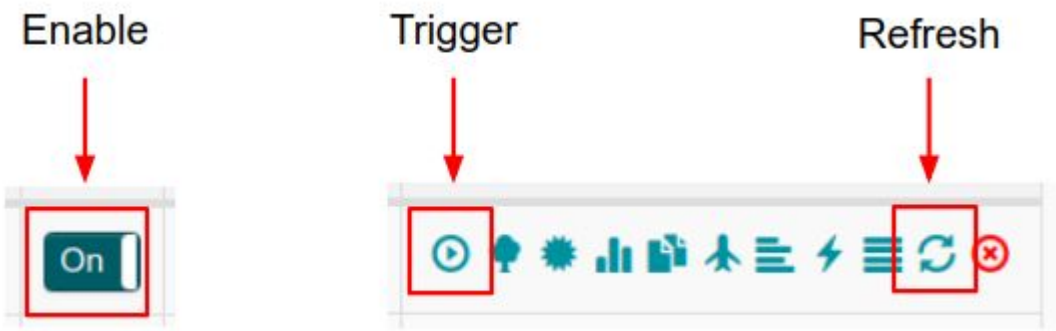
Step 6: Analyzing Model Results

- Search through `taxi_pipeline.py` in airflow/dags
  a. Search for “Step 6”
  b. Uncomment the lines marked with Step 6
  c. Take a moment to review the code that you uncommented

Refresh and Trigger

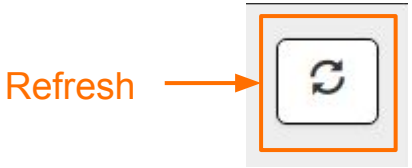
In a browser:

- Return to DAGs list page in Airflow by clicking on "DAGs" link
- Click the refresh button on the right side for the taxi DAG
  - You should see "DAG [taxi] is now fresh as a daisy"
- Trigger taxi

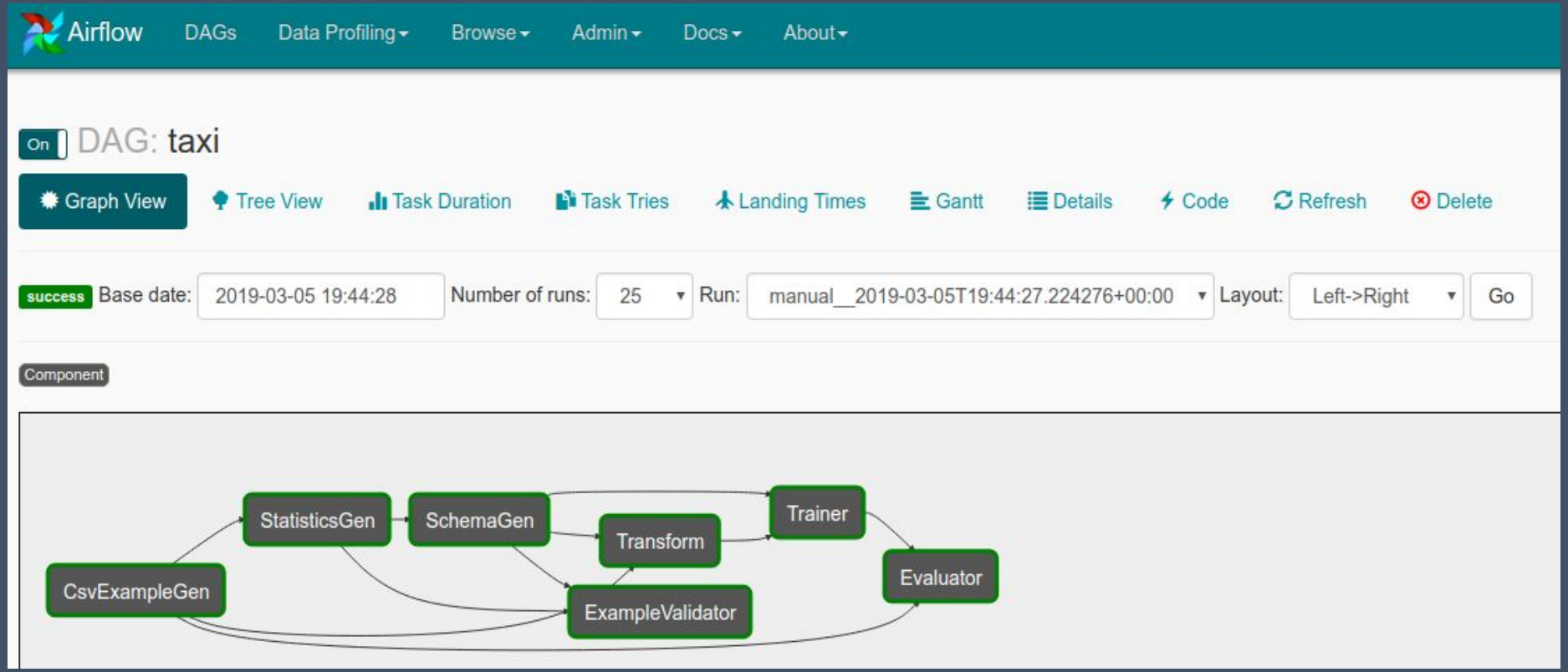


Refresh and Trigger

- In the Airflow DAG view, click on taxi
- Wait for the all components to turn dark green
- Use the refresh button on the right or refresh the page



After a few minutes, your DAG should look like this



Back on Jupyter

- Open step6.ipynb
- Follow the notebook
  - Note that the notebook starts by connecting to the metadata store
  - Artifacts used in the notebook are created by the TFX pipeline that we run in Airflow
  - The notebook accesses the metadata store to read artifacts

Examples (Weighted) Threshold

0

Visualization

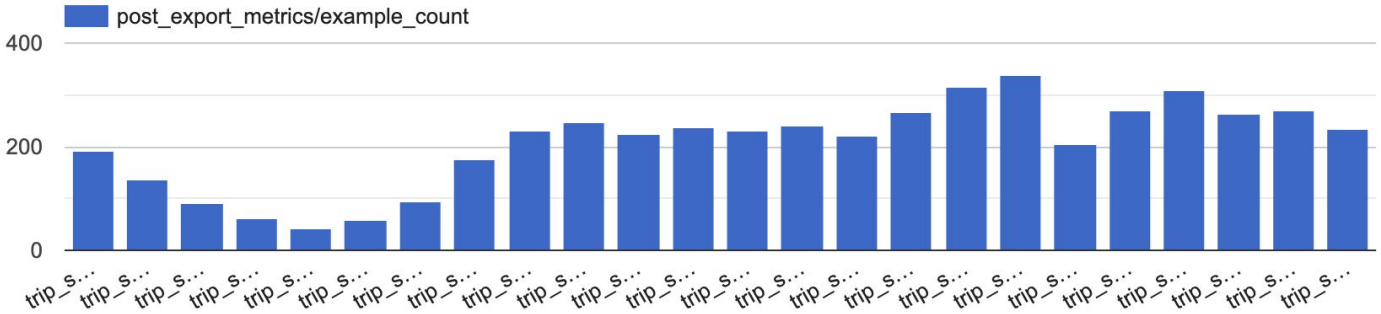
Slices Overview

Show

post_export_metrics/example_count

Sort by

Slice



feature	accuracy	accuracy_baseline	auc	auc_precision_recall	average_loss
trip_start_hour:2	0.82609	0.81884	0.95133	0.71668	0.30771
trip_start_hour:7	0.83158	0.82105	0.95211	0.67722	0.29082
trip_start_hour:12	0.78059	0.75527	0.96740	0.82035	0.32535

TensorFlow

Step 7: Production Readiness

Validating model readiness and managing production readiness

Step 7: Production Readiness

If the new model is ready, make it so

- If we're replacing a model that is currently in production, first make sure that the new one is better
  - ModelValidator tells Pusher if the model is OK
- Pusher deploys SavedModels to well-known locations

Step 7: Production Readiness

Deployment targets receive new models from well-known locations

- TensorFlow Serving
- TensorFlow Lite
- TensorFlow JS
- TensorFlow Hub

We will only showcase deployment to Tensorflow Serving today

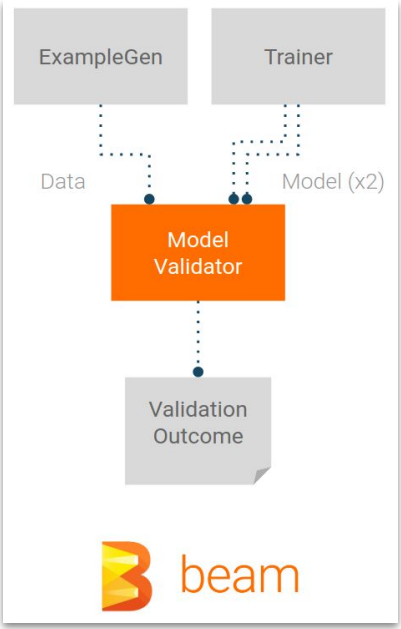
Components

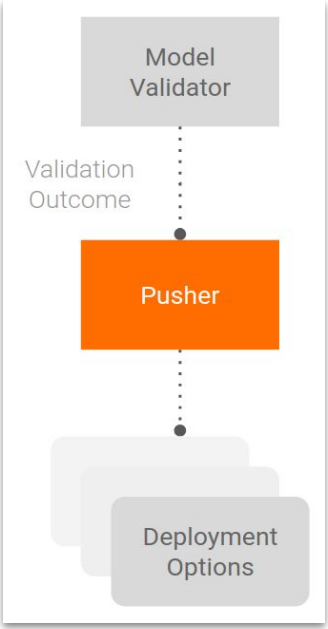
ModelValidator

- Compares multiple versions of the trained model to make sure that the new version meets requirements

Pusher

- If the model passes ModelValidator, Pusher deploys the SavedModel to a well-known location





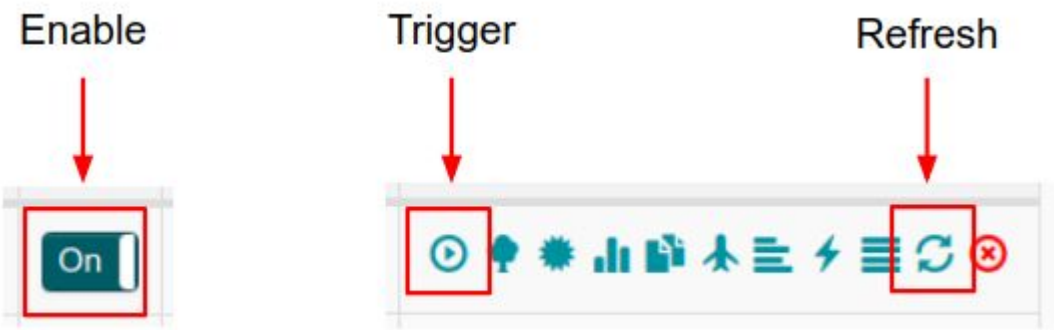
Step 7: Production Readiness

- Search through `taxi_pipeline.py` in airflow/dags
  a. Search for “Step 7”
  b. Uncomment the lines marked with Step 7
  c. Take a moment to review the code that you uncommented

Refresh and Trigger

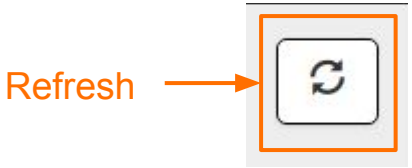
In a browser:

- Return to DAGs list page in Airflow by clicking on "DAGs" link
- Click the refresh button on the right side for the taxi DAG
  - You should see "DAG [taxi] is now fresh as a daisy"
- Trigger taxi

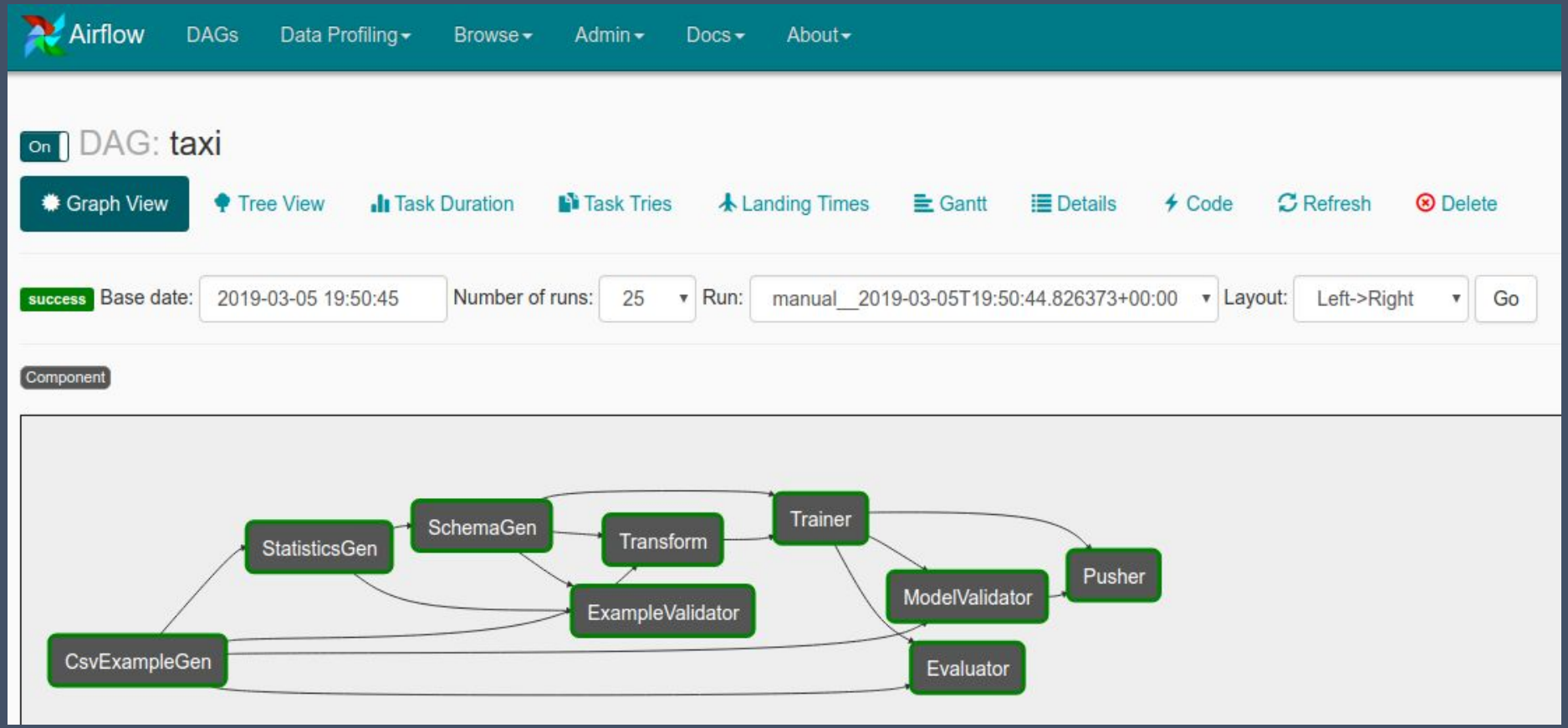


Refresh and Trigger

- In the Airflow DAG view, click on taxi
- Wait for the all components to turn dark green
- Use the refresh button on the right or refresh the page



After a few minutes, your DAG should look like this



Step 8: Deploy to TF Serving and Query the model

Deploying Tensorflow Serving, instructing it to serve our model and sending example requests

TensorFlow Serving

- Multi-tenant, highly optimized serving system for TF models
- Provides a gRPC or REST API for models with support for versioning
- Low latency and high throughput with inter-request batching
- Used for years at Google supporting millions of QPS

Step 8: Deploy to TF Serving and query the live model

- The pusher has pushed the SavedModel to the filesystem of the container
- The filesystem of the container is mapped via a volume to the host
- We will launch TF Serving in its own container, with the directory containing the SavedModel available mapped in
- We will use a Jupyter notebook to query the model
- This will work because both containers will be sharing the host's network interface (so localhost:8501 is the same in both containers)

Step 8: Deploy to TF Serving

- `docker pull tensorflow/serving:nightly`
- `export SAVED_MODEL_DIR=<repo root>/tfx_airflow/workshop/airflow/serving_model`
- `docker run --net=host --rm  -p 8501:8501 -e MODEL_NAME=taxi -v $SAVED_MODEL_DIR:/models -t tensorflow/serving:nightly`

Back on Jupyter

- Open step8.ipynb
- Follow the notebook to query the live model
  - Note that the notebook uses the metadata store to get the schema for our model, which is needed to encode our example requests
  - The notebook simply loads a CSV dataset, encodes it as TensorFlow Examples and sends requests to TF Serving

Cleanup Docker

When you're done:

- Use `docker ps` to get the name of the Serving container
- Use `docker stop <container name>` to stop the Serving container
- Use `docker ps -a` to get the names of both the TFX and Serving containers
- Use `docker rm <TFX container> <Serving container>` to remove the containers

Next Steps

You have now deployed your model to Tensorflow Serving and queried it. Using the SavedModel file under the airflow/saved_models/taxi directory, you can now deploy your model to any of the TensorFlow deployment targets, including:

- TensorFlow Lite, for including your model in an Android or iOS native mobile application, or in a Raspberry Pi, IoT, or microcontroller application.
- TensorFlow.js, for running your model in a web browser or Node.JS application.

More Advanced Examples

The examples presented here are really only meant to get you started. For more advanced examples see:

`https://www.tensorflow.org/tfx/tutorials`

- TensorFlow Data Validation Colab
- TensorFlow Transform Colab
- TensorBoard Tutorial
- TFMA Chicago Taxi Tutorial

More Information

Website: https://www.tensorflow.org/tfx

TFX Repo: https://github.com/tensorflow/tfx

MLMD Repo: https://github.com/google/ml-metadata

TFX Discussion Group: https://goo.gle/2OGjw78

TFX YouTube: https://goo.gle/2xVkwt4