BEAM SUMMIT

Large Scale Data processing with TFX

Olu Akinlaja
Doit International
https://www.doit.com/

Agenda

- Background and Introduction
  - Motivation
- Why TFX?
  - TFX Components and Apache Beam
- How does using TFX apply to Apache Beam?
  - Is choosing Apache Beam executor necessary?
  - Dataflow Runner and its advantages
- Going deeper into the use case
- Questions

BACKGROUND AND INTRODUCTION

- Introducing myself a little more
  - Data Engineer with the DoIT team.
  - Big fan of Apache Beam
  - Interests in Machine Learning Ops
    - Machine Learning Frameworks and libraries including Tensorflow Extended.

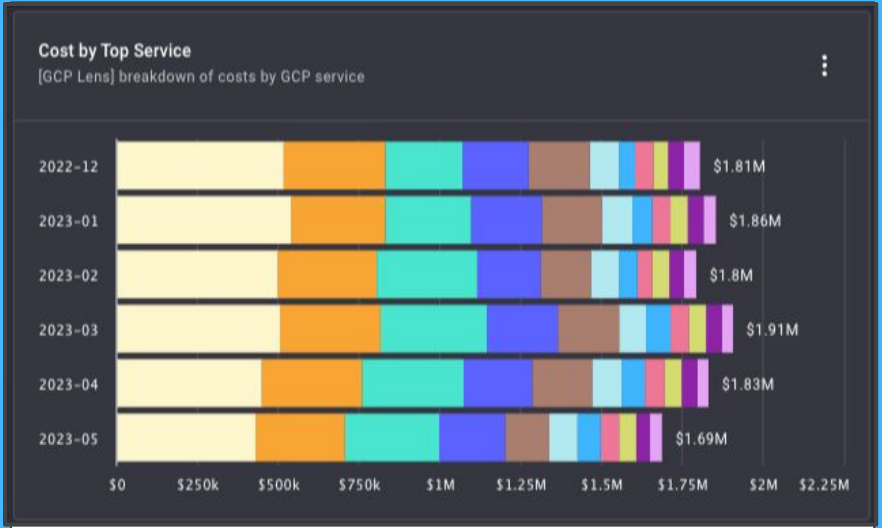
Nonetheless, I do not consider myself an ML expert.

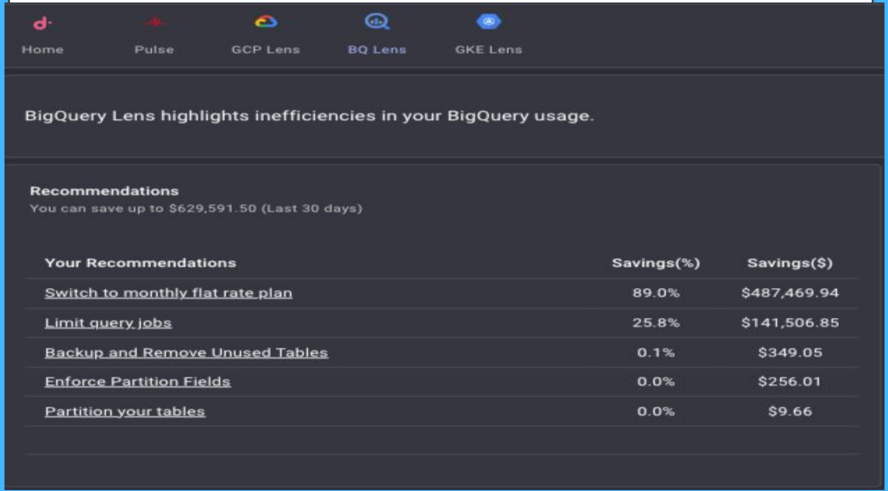
- DoIT is a Global Organization that is a Google Cloud Partner
  - Offers teams to leverage and harness the benefits of Public Clouds
    - Provide technology and cloud expertise to help reduce cloud costs and boost engineering productivity
    - Cloud support is offered at zero cost to Customers.

BACKGROUND AND INTRODUCTION

- DoIT is a Global Organization that is a Google Cloud Partner
  - Offers teams to leverage and harness the benefits of Public Clouds
    - Provide technology and cloud expertise to help reduce cloud costs and boost engineering productivity
    - Cloud support is offered at zero cost to Customers.
    - Introducing the doit Console

doit Console





Target task

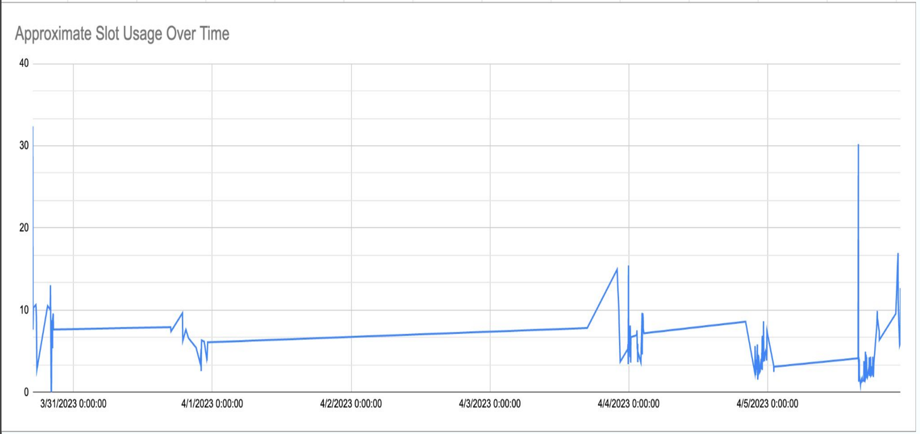
- Recent PSA from the Google Cloud team about the changes to the the pricing and Compute Model of BigQuery
  - Introduction of new BigQuery Editions
  - flat-rate pricing and Flex slot purchased to be disabled for all users
- These changes are set to apply to both the Compute and Storage in BigQuery.
- The changes would apply starting July 5th
- https://www.doit.com/bigquery-editions-and-what-you-need-to-know/

DoiT pricing recommendation

1	**Make a copy of this to start editing, also note it is per-project so you might need to make a copy per project.**
2	
3	doit **BigQuery Editions Pricing Analysis Tool**
4	This analysis tool will consume your BigQuery usage patterns (both compute and storage) and then generate an analysis of that data to give an estimate of spend with Google's new pricing models that go into effect on July 5, 2023.
5	
6	Important Notes
7	Note this tool is doing an estimation of the BigQuery usage for a project with known historical values from your BigQuery usage, there may be additional factors, charges, or changes that are yet to come that are not included in the output of this tool. Thus this tool should only be used as general guidance and not as an absolute source of truth for charges with these new billing models.
8	Note due to limitations with Sheets this can only query a single project at a time and not a whole organization. Making copies of this and running against separate projects is the easiest method to do this for an entire organization
9	
10	IAM Notes
11	This tool will be running queries against the INFORMATION_SCHEMA views inside of a project. It is recommended that the user that has opened up this Sheet have at least the BigQuery Data Editor role for the project that will be used for this analysis
12	
13	How to Use This Tool
14	After running the below instructions numerous calculations on your usage are performed to calculate estimates on BigQuery compute and storage costs for multiple scenarios with Editions, Compressed Storage, and On-Demand pricing models. These estimates can be used to help determine what pricing may look like after July 5, 2023 (or before if opted into starting early) when this pricing goes into effect.
15	
16	Instructions
17	The instructions below are a bit tedious, but unfortunately there is not an easy way to pull data from BigQuery into a format that is usable by both technical BigQuery users and the non-technical users that would need to consume this data to assist in making decisions based upon it. So a balance had to be made which was a Google Sheet showing this data and due to limitations in Sheets and BigQuery integration there are more steps than myself as the author would have liked and if you are able to discover a way to automate this without using App Scripts (that requires more permissions and many organizations restrict its use) please let me know and I will implement it on here.
18	
19	1. Navigate to the BQ Compute Data sheet at the bottom of this page
20	2. Click the "Connection Settings" link in the upper right corner (has a sprocket icon next to it)

+ ☰ Instructions and Guidance ▾ | Compute+Storage ▾ | Compute Calculations ▾ | Storage Calculations ▾ | Slot Usage Chart

Calc slot usage & Compute price



Editions Usage

No Commitments Purchased and Using Compressed Storage (Pay-As-You-Go)

	Current Usage	Existing Flat-Rate Costs w/ Uncompressed Storage	Difference Between Editions and Flat-Rate
Standard Edition	**$172.82**	$2,000.03	**-$1,827.21**
Enterprise Edition	**$259.22**	$2,000.03	**-$1,740.81**
Enterprise Plus Edition	**$432.02**	$2,000.03	**-$1,568.01**

No Commitments Purchased and Using Uncompressed Storage

	Current Usage	Existing Flat-Rate Costs w/ Uncompressed Storage	Difference Between Editions and Flat-Rate
Standard Edition	**$172.83**	$2,000.03	**-$1,827.20**
Enterprise Edition	**$259.23**	$2,000.03	**-$1,740.80**
Enterprise Plus Edition	**$432.03**	$2,000.03	**-$1,568.00**

Editions Estimates with Commitments and Compressed Storage

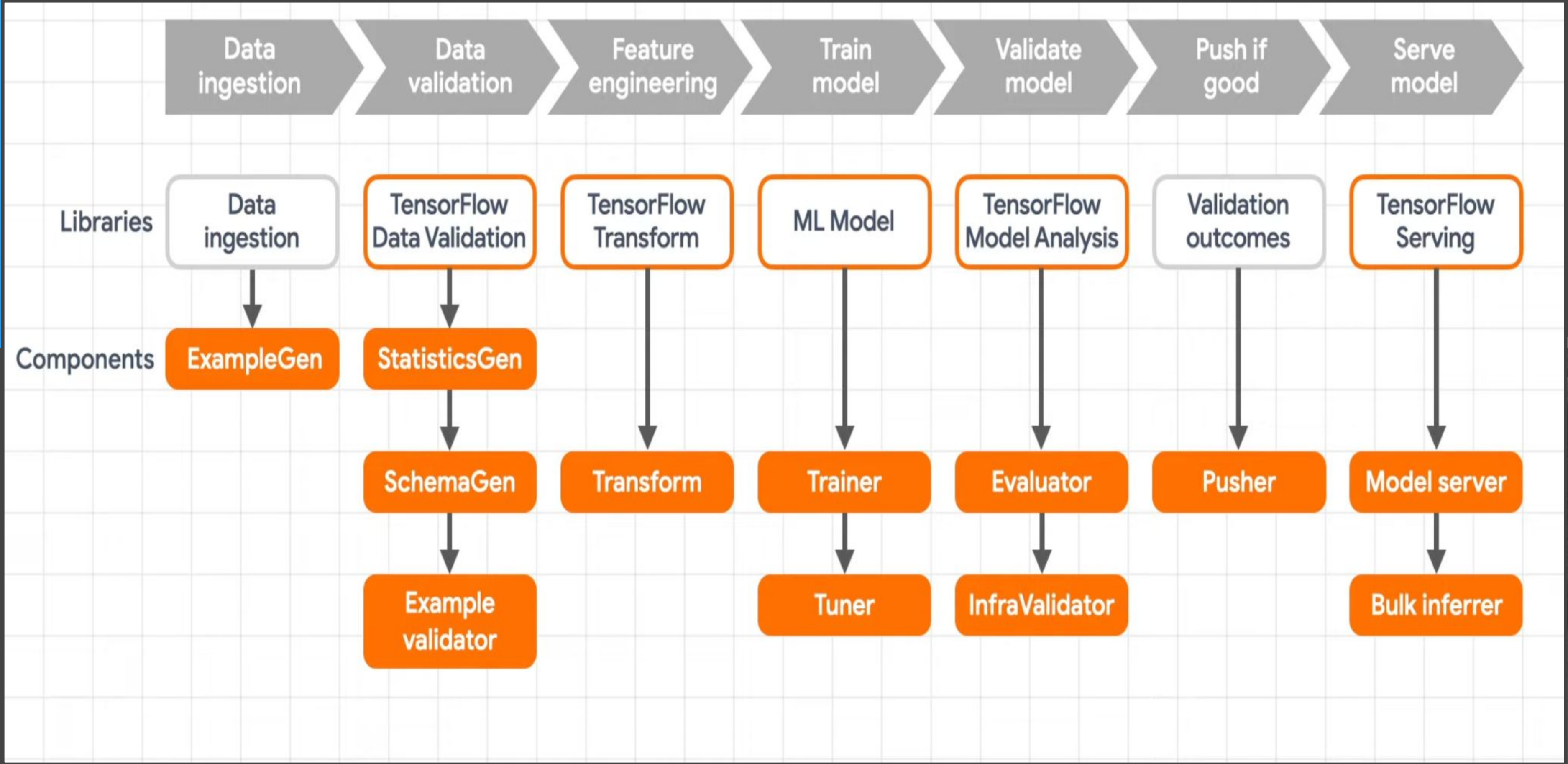
	Current Usage	Existing Flat-Rate Costs w/ Uncompressed Storage	Difference Between Editions and Flat-Rate
Enterprise Edition 1 Year			
1 Year Commitment Baseline Count	0		
1 Year Commitment Baseline	$0.00		
Autoscaling Slots	$259.20		
	$259.22	$2,000.03	**-$1,740.81**
Enterprise Edition 3 Year			
3 Year Commitment Baseline Count	0		
3 Year Commitment Baseline	$0.00		
Autoscaling Slots	$259.20		

+ ☰ Instructions and Guidance ▾ | Compute+Storage ▾ | Compute Calculations ▾ | Storage Calculations ▾

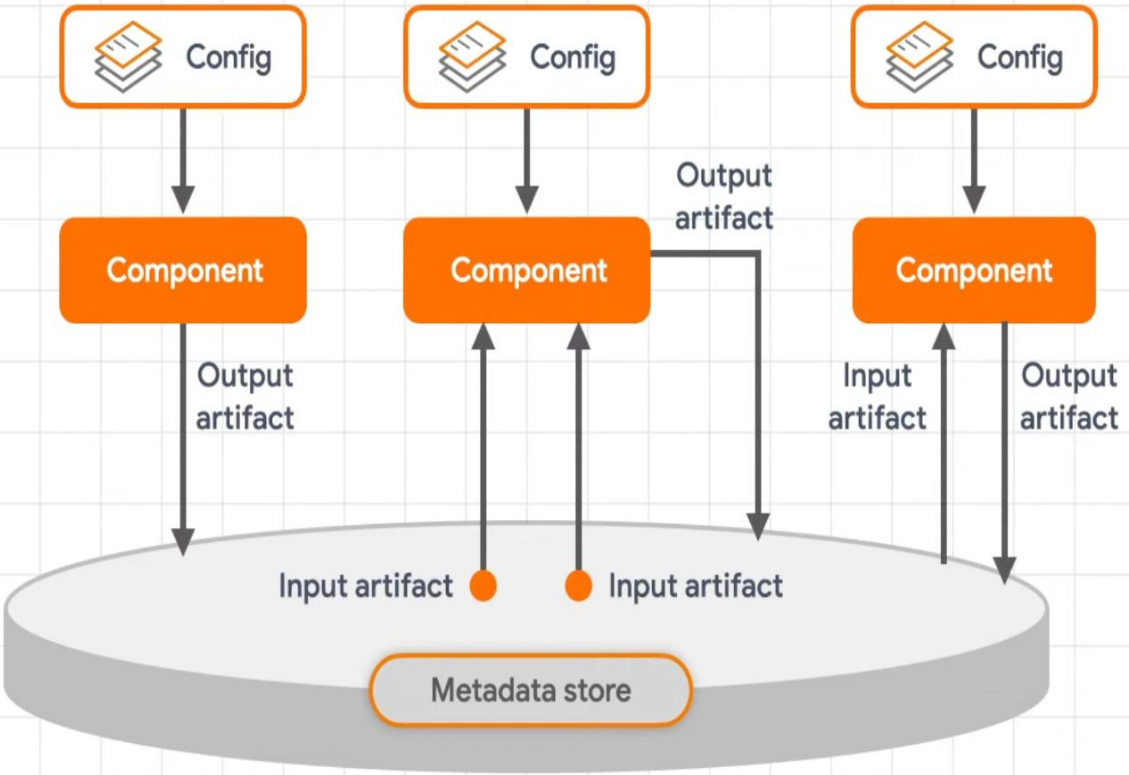
What is TFX and Why TFX

- TFX is a Google-production-scale machine learning (ML) platform, which provides a toolkit that is based on TensorFlow for building ML pipelines.
- Tensorflow Extended (TFX) is designed to build end-to-end machine learning pipelines.
  - Data Ingestion
  - Data preparation
    - Data Exploration
    - Data transformation
    - Feature Engineering
  - Data Segregation
  - Model training
  - Model evaluation
  - Model deployment and Monitoring

TFX Production Components



TFX Components



TFX Components and Apache Beam

- TFX Components use Apache Beam for distributed pipeline processing
  - Internally translated into an Apache Beam pipeline
  - creates a Directly acyclic graph of Computation, which is then sent to the Executor[or Runner, eg Dataflow Runner]
  - Executor then spins up workers to handle the work
  - The results are then sent back to the TFX Components

Apache Beam

- unified programming model to execute data processing pipelines, including ETL, batch and stream processing
- Building batch and Streaming Pipelines in the Language of Choice
  - Providing various language-specific SDK
    - Java SDK
    - Python SDK
    - Go SDK
- Allows execution of built pipelines to be run on different execution environments
  - Apache Spark
  - Flink
  - Dataflow Runner
  - Direct Runner

Apache Beam Runner: Dataflow

- Fully Managed Environment
- There is a Monitoring UI
- Easy Scalability
- Configuration

More Details about my use-case

- Dataset is stored in BigQuery.
  - The data are pulled generally from various BigQuery Datasets on different projects, estimated across the different BigQuery editions.

```
SELECT
job_id,
statement_type,
EXTRACT (DATE
FROM
creation_time) AS EXECUTION_DATE,
EXTRACT (HOUR
FROM
creation_time) AS EXECUTION_HOUR,
EXTRACT (MINUTE
FROM
creation_time) AS EXECUTION_MIN,
user_email AS USER,
project_id as PROJECT,
start_time,
end_time,
reservation_id,
total_slot_ms,
total_bytes_processed,
SAFE_DIVIDE(total_slot_ms, TIMESTAMP_DIFF(end_time,start_time,MILLISECOND)) SLOT_USAGE,
TIMESTAMP_DIFF(end_time,start_time,MILLISECOND) / 1000 TOTAL_DURATION_IN_MS,
(total_bytes_processed)/1024/1024/1024 TOTAL_PROCESSED_GB,
(case
when reservation_id is null then total_bytes_billed else 0
end)/1024/1024/1024 TOTAL_BILLED_GB,
cache_hit
FROM
`region-us.INFORMATION_SCHEMA.JOBS_BY_PROJECT`
WHERE
state='DONE'
AND statement_type in ('SELECT', 'MERGE','CREATE_TABLE_AS_SELECT')
AND
creation_time BETWEEN TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 90 DAY)
AND CURRENT_TIMESTAMP()
```

More Details about my use-case

- Consideration for the needed data was based on the information on the GCP documentation about the BigQuery editions.
  - Each of the BigQuery Editions are billed per slotHour

More Details about my use-case

- My Pipeline implementation was done on a notebook in a Google Cloud VertexAI user-managed Instance
  - Google Cloud Vertex Pipelines helps to easily automate, monitor, and govern ML systems by orchestrating your ML workflow in a serverless manner
  - It integrates easily with BigQuery and Dataflow
  - Also, no need to set metadata_connection_config, which is normally used to locate ML Metadata database. However, as Vertex Pipelines uses a managed metadata service – Hence, there is no need to specify this parameter.
- TFX requires python up to 3.9
  - https://github.com/tensorflow/tfx/issues/5897
  - Short term fix is to revert to Python 3.8.3
    - iPyKernel – IPython Kernel for Jupyter

More Details about my use-case

❖ Specify paths to the pipeline artifacts, python module, etc

```
GOOGLE_CLOUD_PROJECT = '[redacted]                    # <--- ENTER THIS
GOOGLE_CLOUD_PROJECT_NUMBER = '[redacted]  # <--- ENTER THIS
GOOGLE_CLOUD_REGION = 'us-west1'            # <--- ENTER THIS
GCS_BUCKET_NAME = '[redacted]                    # <--- ENTER THIS

if not (GOOGLE_CLOUD_PROJECT and  GOOGLE_CLOUD_PROJECT_NUMBER and GOOGLE_CLOUD_REGION and GCS_BUCKET_NAME):
    from absl import logging
    logging.error('Please set all required parameters.')

PIPELINE_NAME = 'bigquery-editions'

# Path to various pipeline artifact.
PIPELINE_ROOT = 'gs://{}/pipeline_root/{}'.format(
    GCS_BUCKET_NAME, PIPELINE_NAME)

# Paths for users' Python module.
MODULE_ROOT = 'gs://{}/pipeline_module/{}'.format(
    GCS_BUCKET_NAME, PIPELINE_NAME)

# Paths for users' data.
DATA_ROOT = 'gs://{}/data/{}'.format(GCS_BUCKET_NAME, PIPELINE_NAME)

# This is the path where your model will be pushed for serving.
SERVING_MODEL_DIR = 'gs://{}/serving_model/{}'.format(
    GCS_BUCKET_NAME, PIPELINE_NAME)

print('PIPELINE_ROOT: {}'.format(PIPELINE_ROOT))

QUERY = "SELECT * FROM `s[redacted]"

_trainer_module_file = 'SummitTFX.py'
```

More Details about my use-case

- With the TFX libraries, it was easy to use the ExampleGen to pull the data from BigQuery.

```
from typing import List, Optional
import tensorflow as tf
from tfx import v1 as tfx

def _create_pipeline(pipeline_name: str, pipeline_root: str, query: str,
                     module_file: str, serving_model_dir: str,
                     beam_pipeline_args: Optional[List[str]],
                     ) -> tfx.dsl.Pipeline:
  """Creates a TFX pipeline using BigQuery."""

  # NEW: Query data in BigQuery as a data source.
  example_gen = tfx.extensions.google_cloud_big_query.BigQueryExampleGen(query=query)
```

- Similarly, it was easy to use other libraries.
  - Used the tfx components: statisticsGen, SchemaGen and Anomaly detections

More Details about my use-case

```python
# NEW: Computes statistics over data for visualization and schema generation.
statistics_gen = tfx.components.StatisticsGen(
    examples=example_gen.outputs['examples'])

# generate schema
schema_gen= tfx.components.SchemaGen(statistics=statistics_gen.outputs['statistics'])

# Identify anomalies
validator = tfx.components.ExampleValidator(statistics=statistics_gen.outputs['statistics'],
                                            schema=schema_gen.outputs['schema'])
```

More Details about my use-case

```
[21]: import os
import tensorflow as tf
import tensorflow_data_validation as tfdv
STATS_URI = '[redacted]-20230605023331/StatisticsGen_-7946473999146418176/statist
directories = tf.io.gfile.glob(os.path.join(STATS_URI, 'Split-*'))
names = map(os.path.basename, directories)
splits = {name: os.path.join(directory, 'FeatureStats.pb') for name, directory in zip(names, directories)}

print(splits)

lhs_split = 'Split-train'
rhs_split = 'Split-eval'

tfdv.visualize_statistics(
    lhs_statistics=tfdv.load_stats_binary(splits[lhs_split]),
    lhs_name=lhs_split,
    rhs_statistics=tfdv.load_stats_binary(splits[rhs_split]),
    rhs_name=rhs_split
)
```

```
{'Split-eval': [redacted]20230605023331/StatisticsGen_-7946473999146418176/stat
istics/Split-eval/FeatureStats.pb', 'Split-train': '[redacted]-20230605023331/Sta
tisticsGen_-7946473999146418176/statistics/Split-train/FeatureStats.pb'}
```

Sort by
Feature order ☐ Reverse order Feature search (regex enabled)

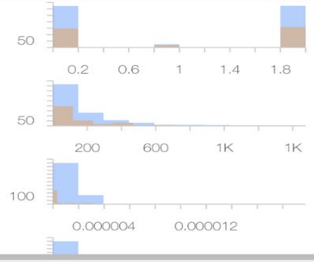
Features: ☑ int(4) ☑ float(9)

☑ Split-train ☑ Split-eval

Numeric Features (13)

	count	missing	mean	std dev	zeros	min	median	max
Label___Edition_Type								
	674	0%	1	0.98	47.92%	0	1	2
	324	0%	1.04	0.97	45.37%	0	1	2
SLOT_USAGE								
	667	1.04%	151.85	223.2	0%	0.03	55.33	1,475.72
	322	0.62%	146.48	216.58	0%	0.04	8.56	1,167.79
SlotHR								
	674	0%	884n	0	9.5%	0	204n	0
	324	0%	792n	0	9.88%	0	55.9n	0
StandardNoCommit								

Chart to show
Standard
☐ log ☐ expand ☐ percentages



More Details about my use-case

```
[22]: import os
import tensorflow_data_validation as tfdv
SCHEMA_URI = '[redacted]20230605023331/SchemaGen_1276898037708357632/schema"
schema = tfdv.load_schema_text(os.path.join(SCHEMA_URI, 'schema.pbtxt'))
tfdv.display_schema(schema)
```

Feature name	Type	Presence	Valency	Domain
'Label___Edition_Type'	INT	required		-
'SLOT_USAGE'	FLOAT	optional	single	-
'SlotHR'	FLOAT	required		-
'StandardNoCommit'	FLOAT	required		-
'TOTAL_BILLED_GB'	FLOAT	required		-
'TOTAL_DURATION_IN_MS'	FLOAT	required		-
'enterprisePrice1YRCommit'	FLOAT	required		-
'enterprisePrice3YRCommit'	FLOAT	required		-
'enterprisePriceNoCommit'	FLOAT	required		-
'onDemand'	FLOAT	required		-
'reservation_id'	INT	required		-
'total_bytes_processed___'	INT	required		-
'total_slot_ms'	INT	optional	single	-

```
[23]: import os
import tensorflow_data_validation as tfdv
ANOMALIES_URI = "[redacted]-20230605023331/ExampleValidator_7041505560742592512/
anomalies = tfdv.load_anomalies_text(ANOMALIES_URI)
tfdv.display_anomalies(anomalies)
```

No anomalies found.

```
[ ]:
```

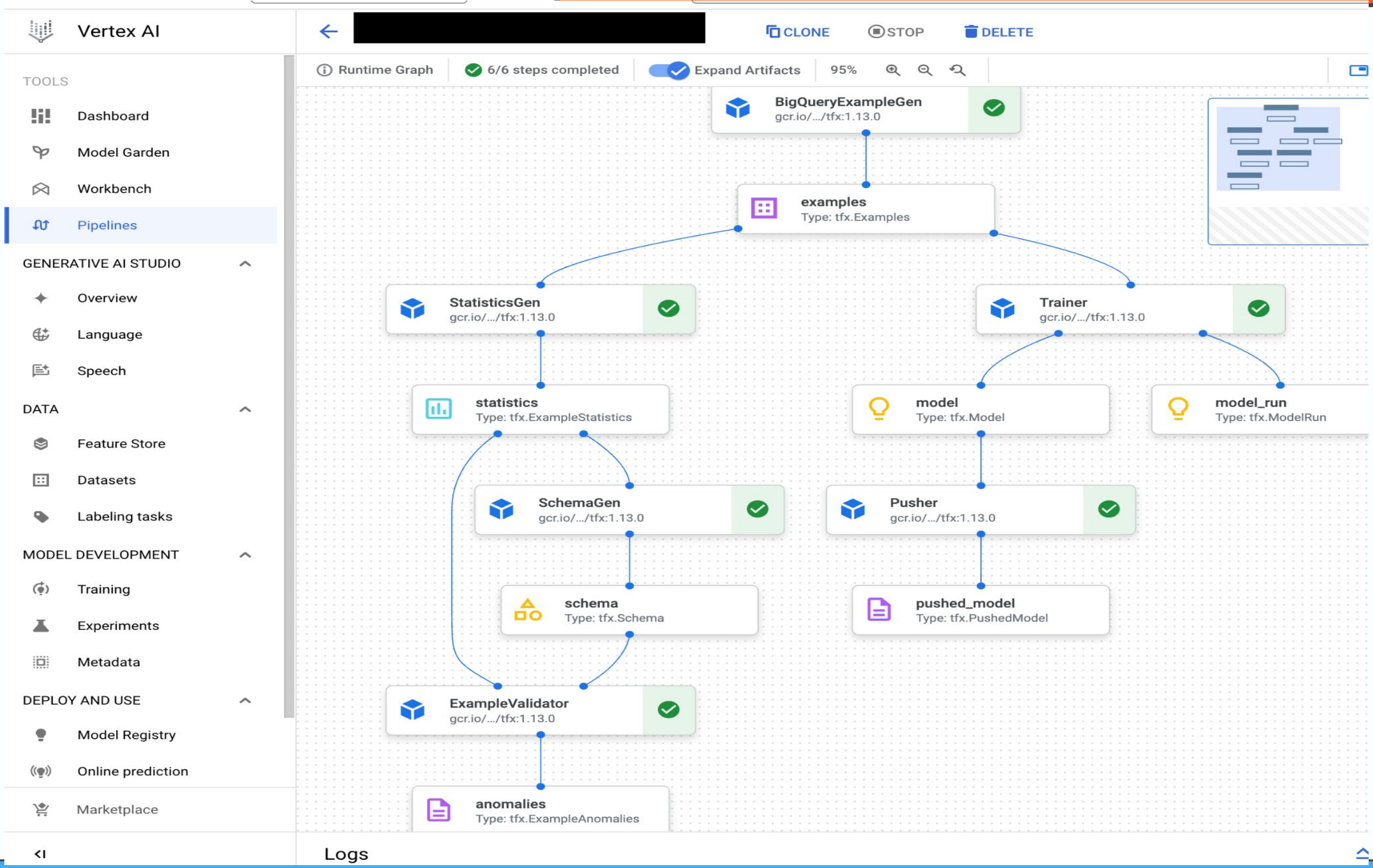
More Details about my use-case

```python
# Uses user-provided Python function that trains a model.
trainer = tfx.components.Trainer(
    module_file=module_file,
    examples=example_gen.outputs['examples'],
    train_args=tfx.proto.TrainArgs(num_steps=100),
    eval_args=tfx.proto.EvalArgs(num_steps=5))

# Pushes the model to a file destination.
pusher = tfx.components.Pusher(
    model=trainer.outputs['model'],
    push_destination=tfx.proto.PushDestination(
        filesystem=tfx.proto.PushDestination.Filesystem(
            base_directory=serving_model_dir)))

components = [
    example_gen,
    statistics_gen,
    schema_gen,
    validator,
    trainer,
    pusher,
]
```

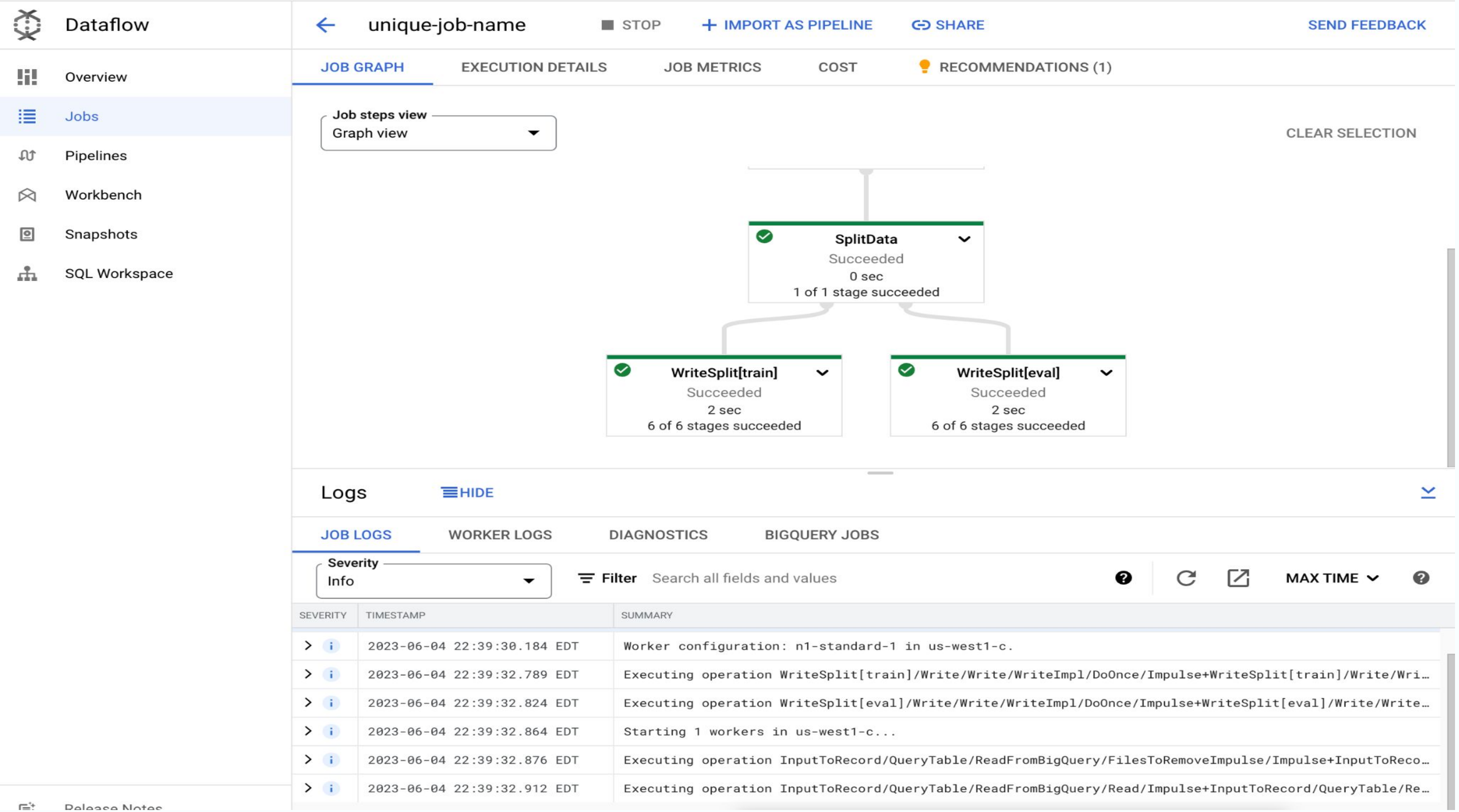
More Details about my use-case

[20]:
```python
# docs_infra: no_execute
import os

# We need to pass some GCP related configs to BigQuery. This is currently done
# using `beam_pipeline_args` parameter.
#DIRECT_RUNNER = [
#    '--project=' + GOOGLE_CLOUD_PROJECT,
#    '--temp_location=' + os.path.join('gs://', GCS_BUCKET_NAME, 'tmp'),
#    ]

DATAFLOW_RUNNER = [
   '--runner=DataflowRunner',
   '--project=' + GOOGLE_CLOUD_PROJECT,
   '--job_name=unique-job-name',
   '--region=us-west1',
   '--temp_location=' + os.path.join('gs://', GCS_BUCKET_NAME, 'tmp'),
   ]

PIPELINE_DEFINITION_FILE = PIPELINE_NAME + '_pipeline.json'

runner = tfx.orchestration.experimental.KubeflowV2DagRunner(
    config=tfx.orchestration.experimental.KubeflowV2DagRunnerConfig(),
    output_filename=PIPELINE_DEFINITION_FILE)
_ = runner.run(
    _create_pipeline(
        pipeline_name=PIPELINE_NAME,
        pipeline_root=PIPELINE_ROOT,
        query=QUERY,
        module_file=os.path.join(MODULE_ROOT, _trainer_module_file),
        serving_model_dir=SERVING_MODEL_DIR,
#         beam_pipeline_args=DIRECT_RUNNER))
        beam_pipeline_args=DATAFLOW_RUNNER))


# docs_infra: no_execute
from google.cloud import aiplatform
from google.cloud.aiplatform import pipeline_jobs
import logging
logging.getLogger().setLevel(logging.INFO)

aiplatform.init(project=GOOGLE_CLOUD_PROJECT, location=GOOGLE_CLOUD_REGION)

job = pipeline_jobs.PipelineJob(template_path=PIPELINE_DEFINITION_FILE,
                                display_name=PIPELINE_NAME)
job.submit()
```

More Details about my use-case



More Details about my use-case



More Details about my use-case

[50]:
```
import tensorflow_model_analysis as tfma

eval_config = tfma.EvalConfig(
    model_specs=[tfma.ModelSpec(label_key='rating')],
    slicing_specs=[tfma.SlicingSpec()],
    metrics_specs=[
        tfma.MetricsSpec(metrics=[

            tfma.MetricConfig(class_name='ExampleCount'),
            tfma.MetricConfig(class_name='AUC'),
            tfma.MetricConfig(class_name='FalsePositives'),
            tfma.MetricConfig(class_name='TruePositives'),
            tfma.MetricConfig(class_name='FalseNegatives'),
            tfma.MetricConfig(class_name='TrueNegatives'),
            tfma.MetricConfig(class_name='BinaryAccuracy',
                threshold=tfma.MetricThreshold(
                    value_threshold=tfma.GenericValueThreshold(
                        lower_bound={'value':0.5}),
                    change_threshold=tfma.GenericChangeThreshold(
                        direction=tfma.MetricDirection.HIGHER_IS_BETTER,
                        absolute={'value':0.0001})
                    )
            )
        ])
    ]

)

from tfx.components import Evaluator
evaluator = Evaluator(
    examples=example_gen.outputs['examples'],
    model=trainer.outputs['model'],
    baseline_model= model_resolver.outputs['model'],
    eval_config=eval_config)

context.run(evaluator)
```

More Details about my use-case

```
# Visualize the evaluation results
eval_result = evaluator.outputs['evaluation'].get()[0].uri
tfma_result = tfma.load_eval_result(eval_result)
tfma.view.render_slicing_metrics(tfma_result)
```

```
]: # Print validation results
eval_result = evaluator.outputs['evaluation'].get()[0].uri
print(tfma.load_validation_result(eval_result))
```

```
validation_ok: true
validation_details {
  slicing_details {
    slicing_spec {
    }
    num_matching_slices: 1
  }
}
```

Special Thanks

- Steeve Dominique
- Gurkomal Singh Rao
- Jared Burns
- Sayle Matthews
- Asad Khan

NAMES

QUESTIONS?

Maybe some contact info here?
Twitter
Linkedin
Github
Or whatever you want