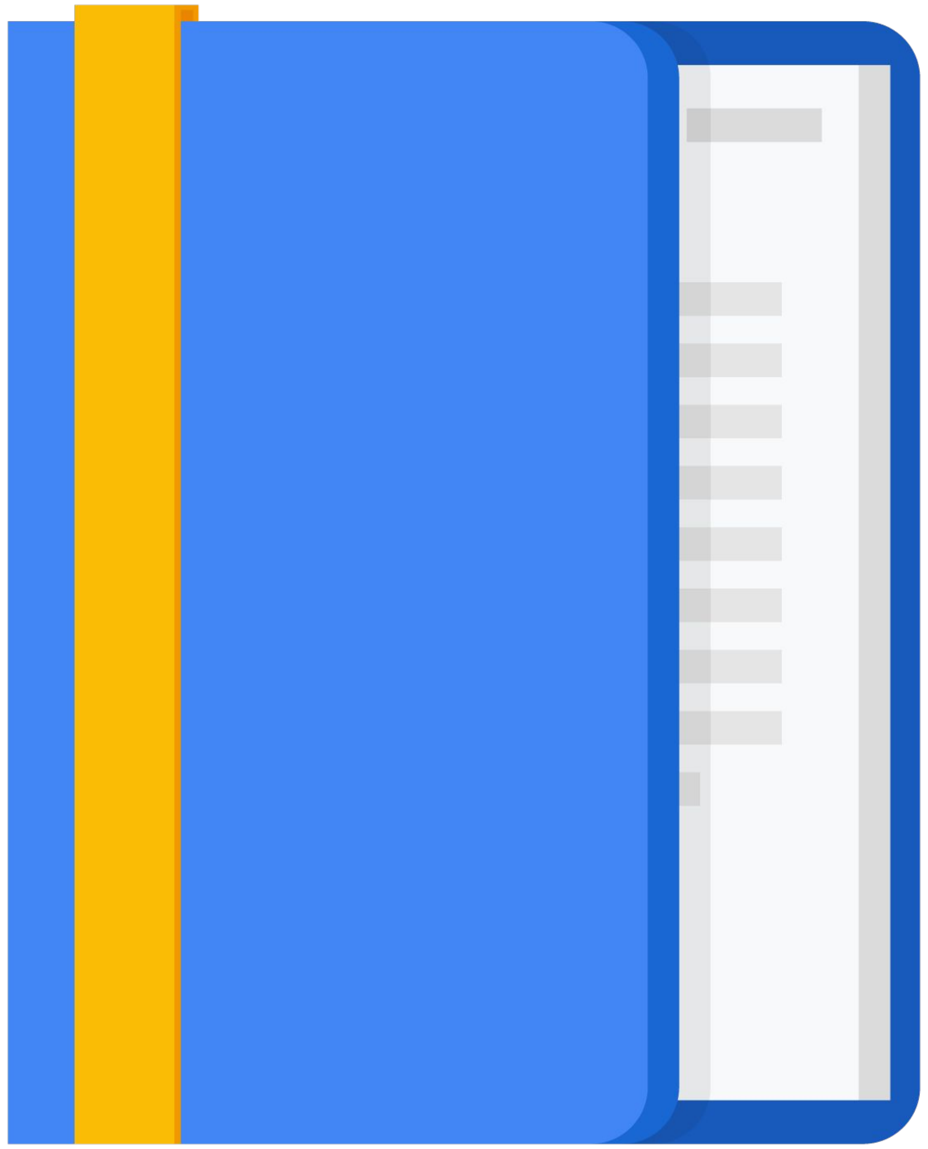
TFX Pipelines

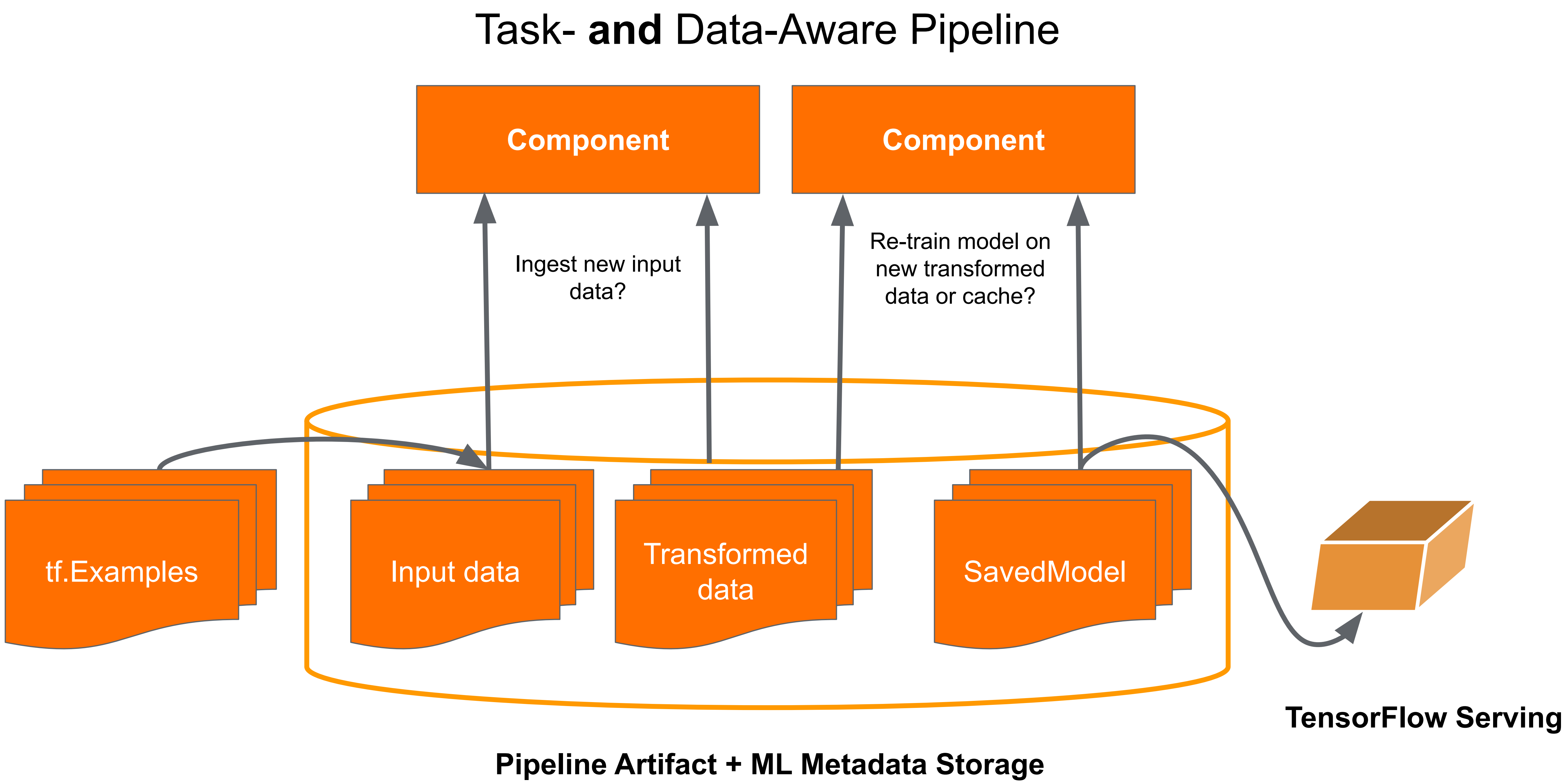
Agenda

TFX orchestrators

TFX pipelines on Vertex AI



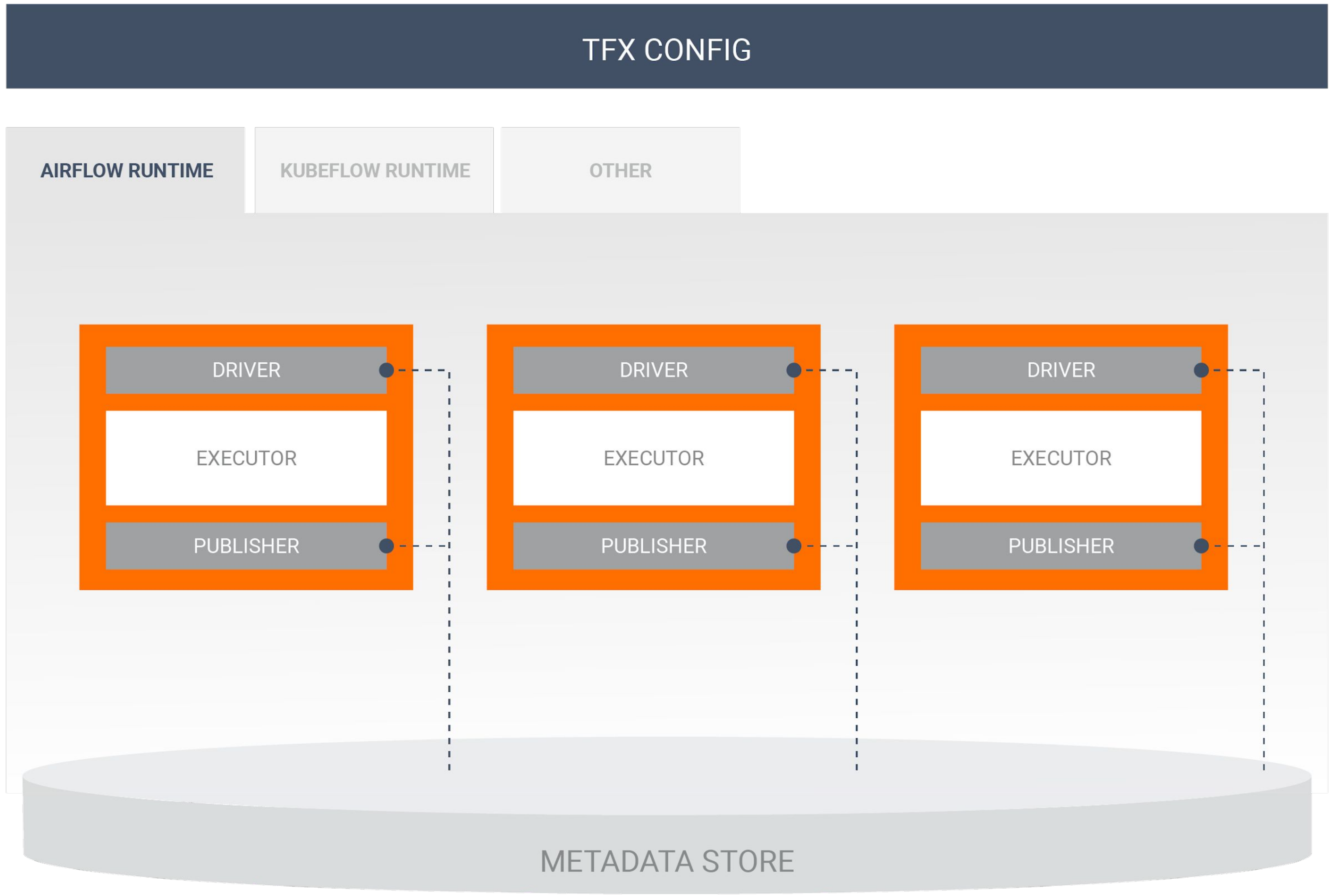
Why orchestrate your ML workflows?



TFX Orchestration in a Notebook

```
context = InteractiveContext()

component = MyComponent(...)
context.run(component)
context.show(component.outputs['my_output'])
```

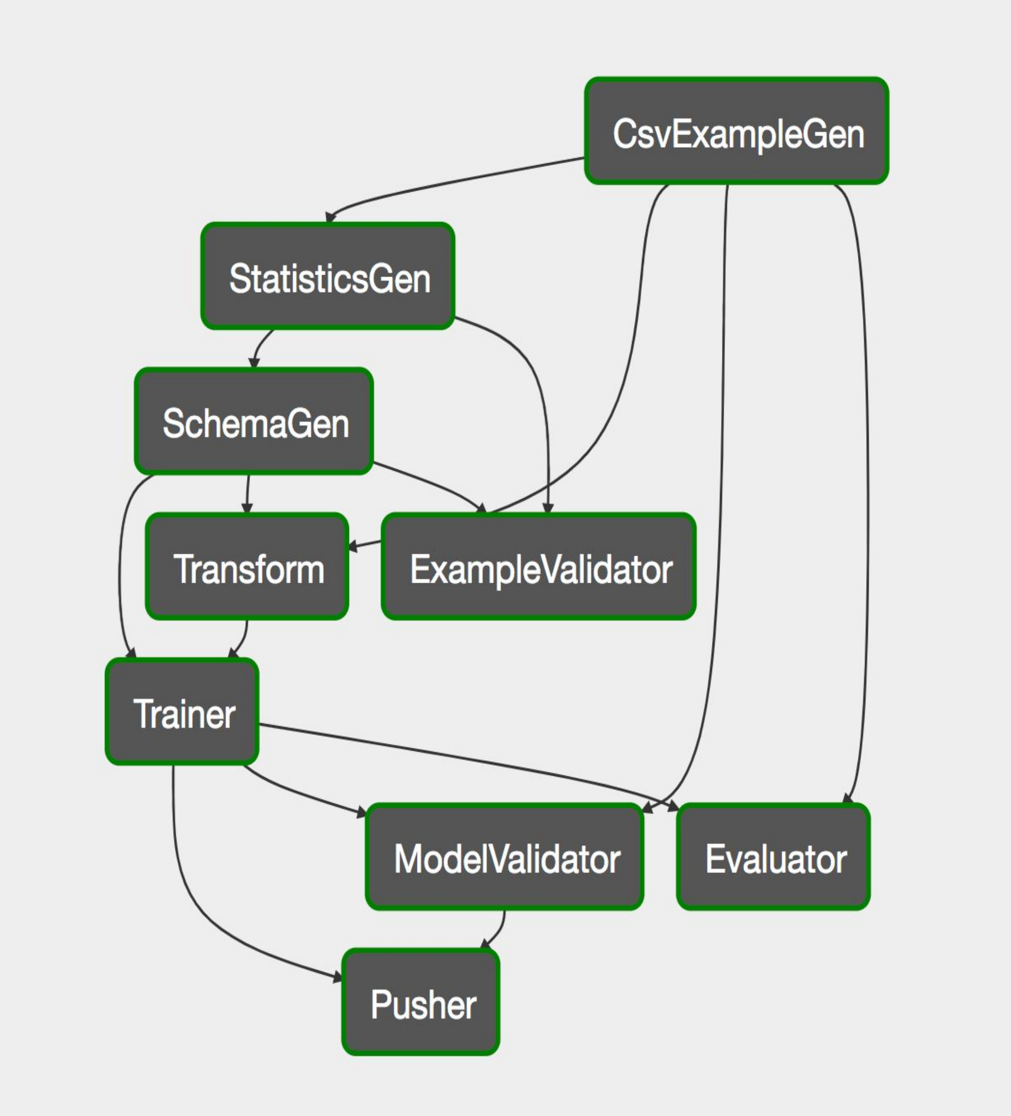
TFX pipelines are portable across Orchestrators



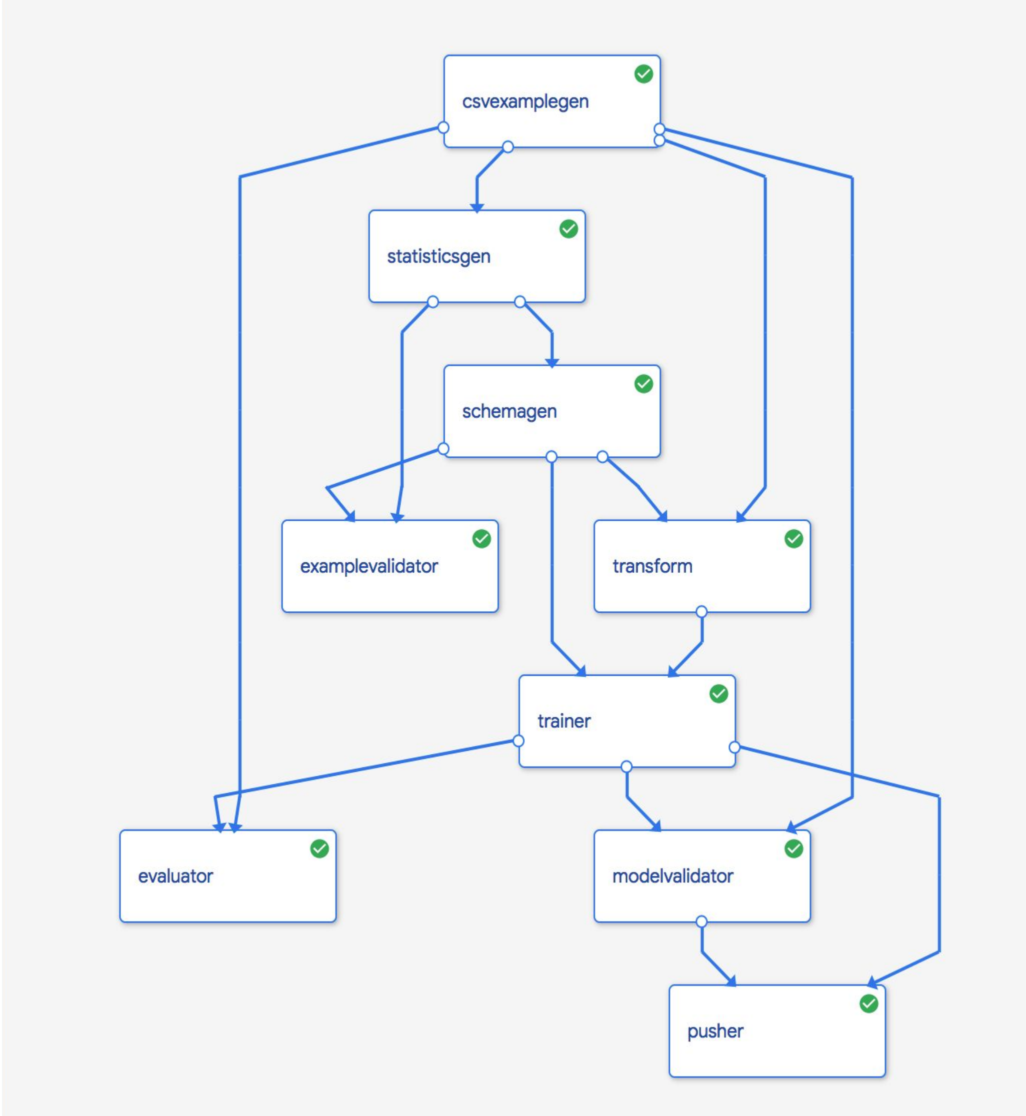
Flexible runtimes run components in sequential order using orchestration systems such as Airflow, Kubeflow, or Beam

TFX pipelines currently support 3 orchestrators

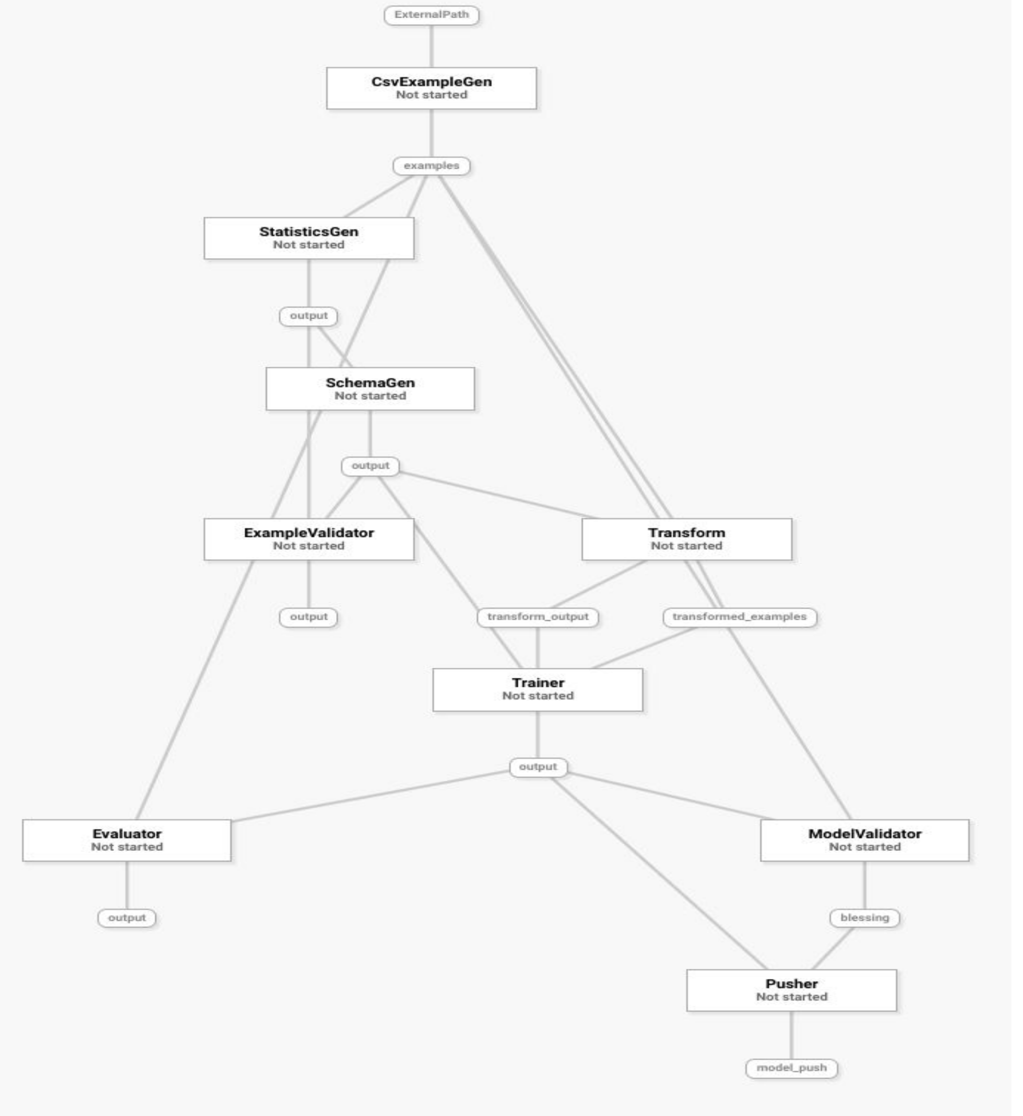
Apache Airflow



Kubeflow Pipelines



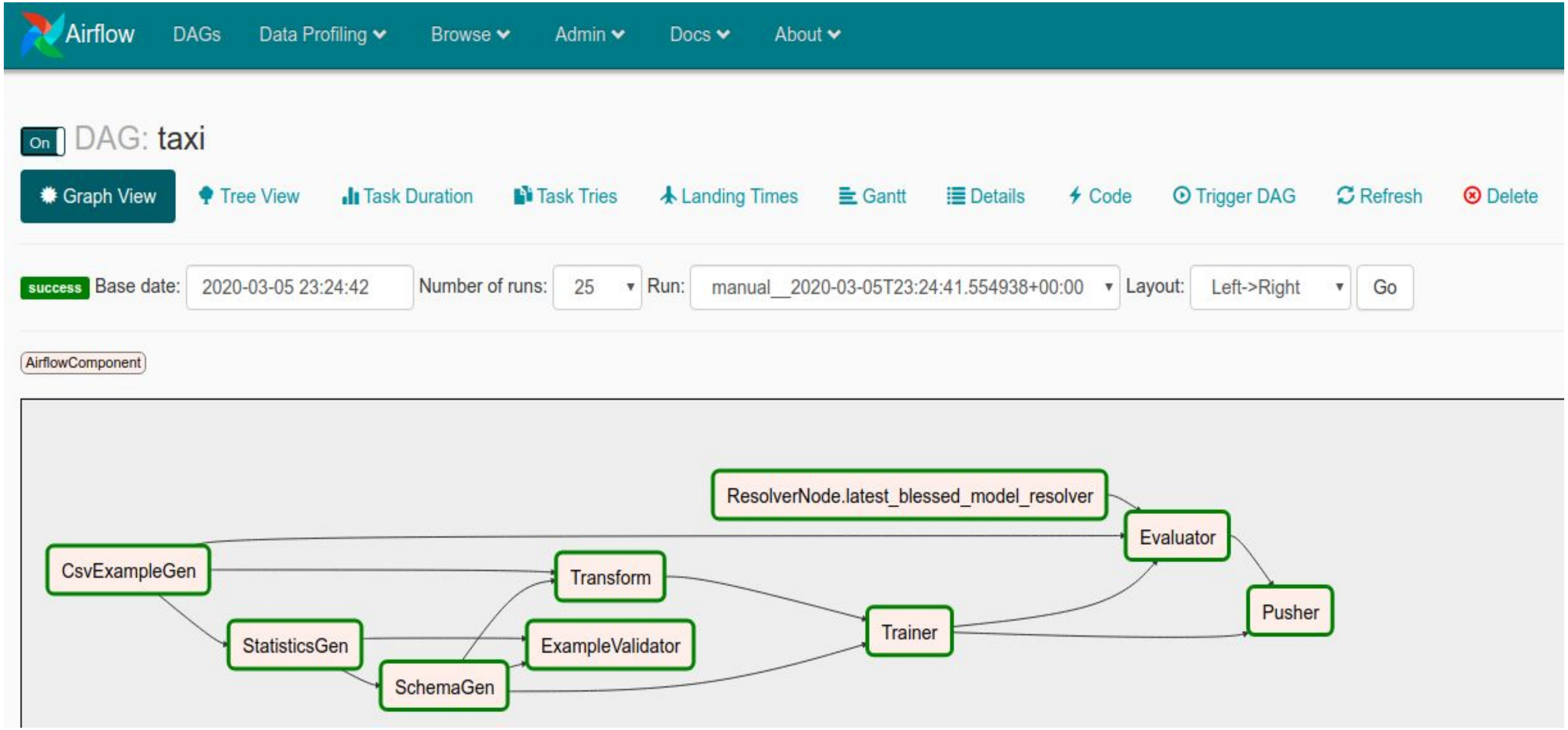
Apache Beam



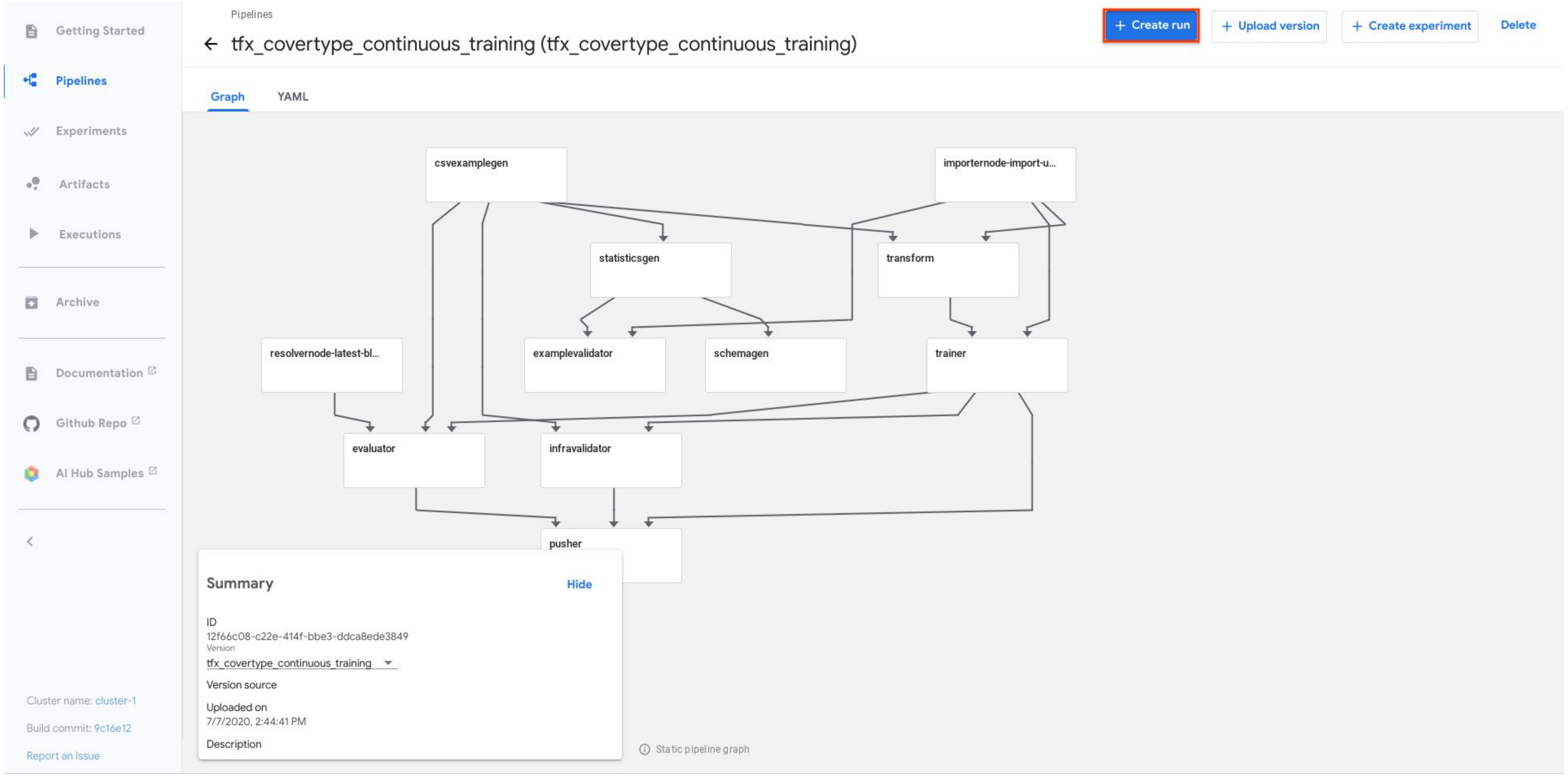
DagRunner

Google Cloud

TFX on Airflow

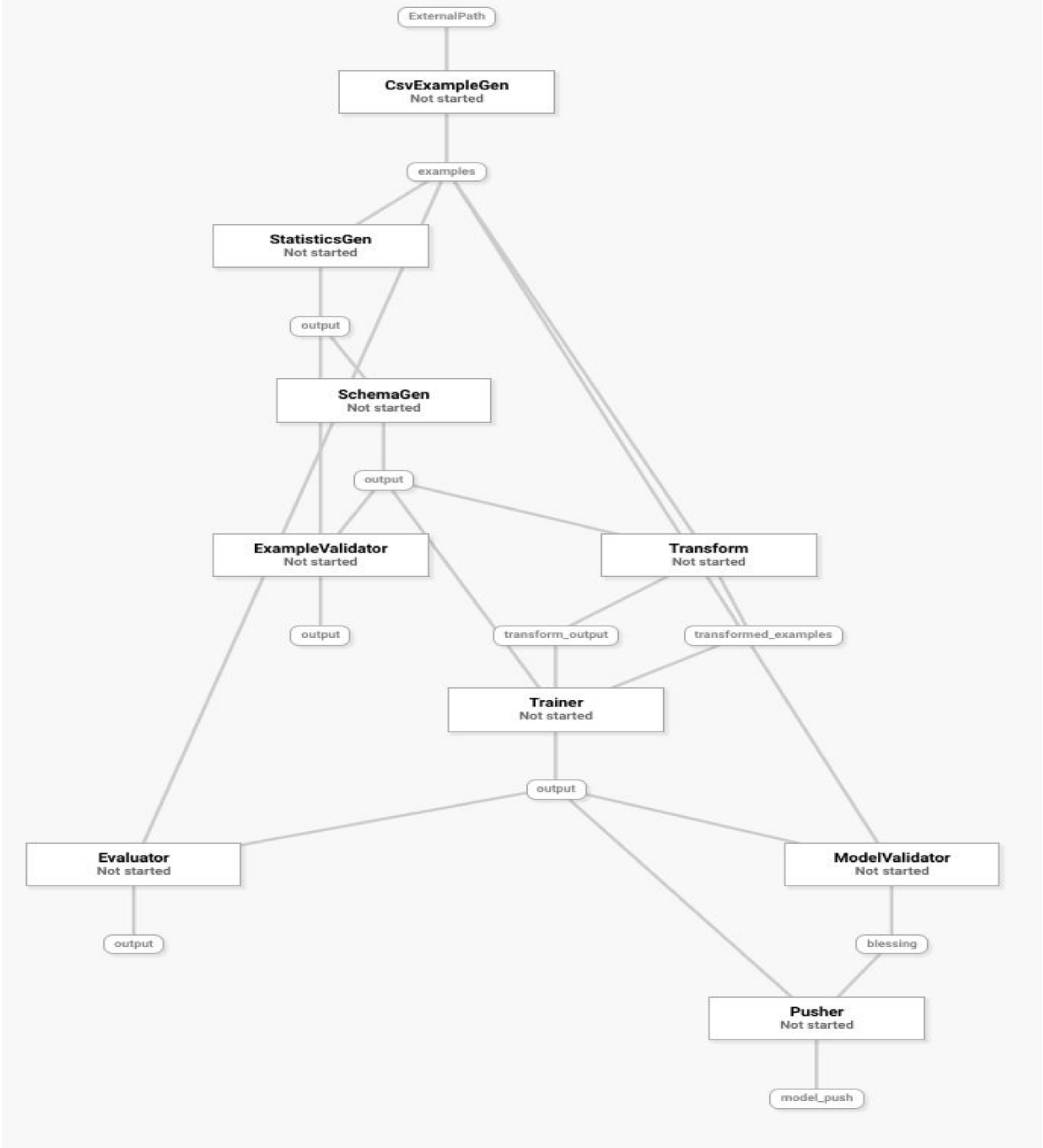


TFX on Kubeflow Pipelines



Google Cloud

TFX on Apache Beam Orchestrator



Google Cloud

Apache Beam is a key data processing abstraction for TFX

Unified

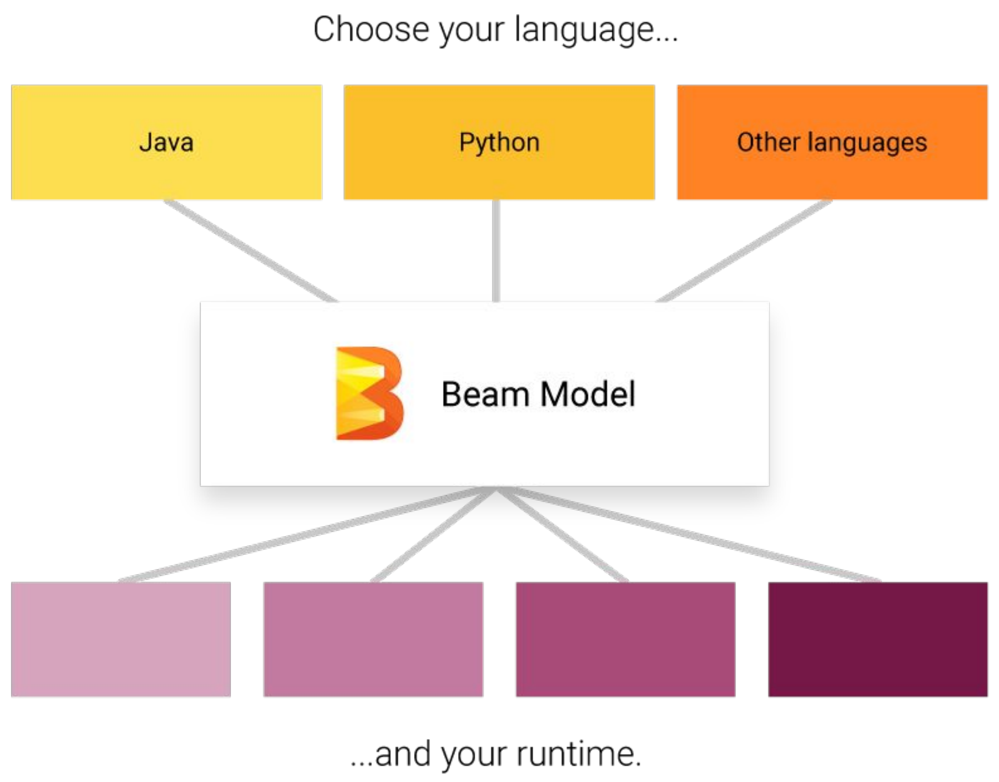
Programming model for **batch** and **stream**

Portable

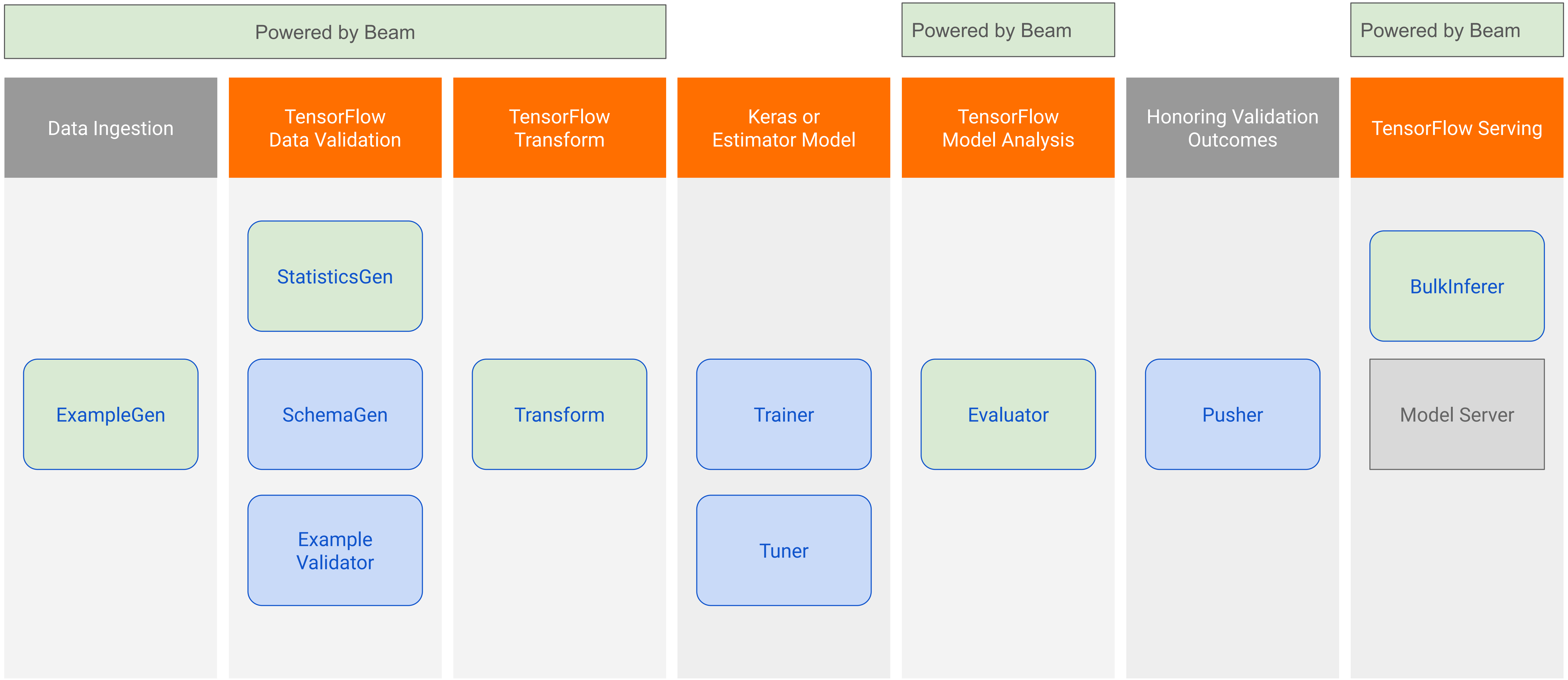
Provide a **choice** of execution **environments**

Extensible

Write and share new **SDK's**, **IO connectors** and **transforms**

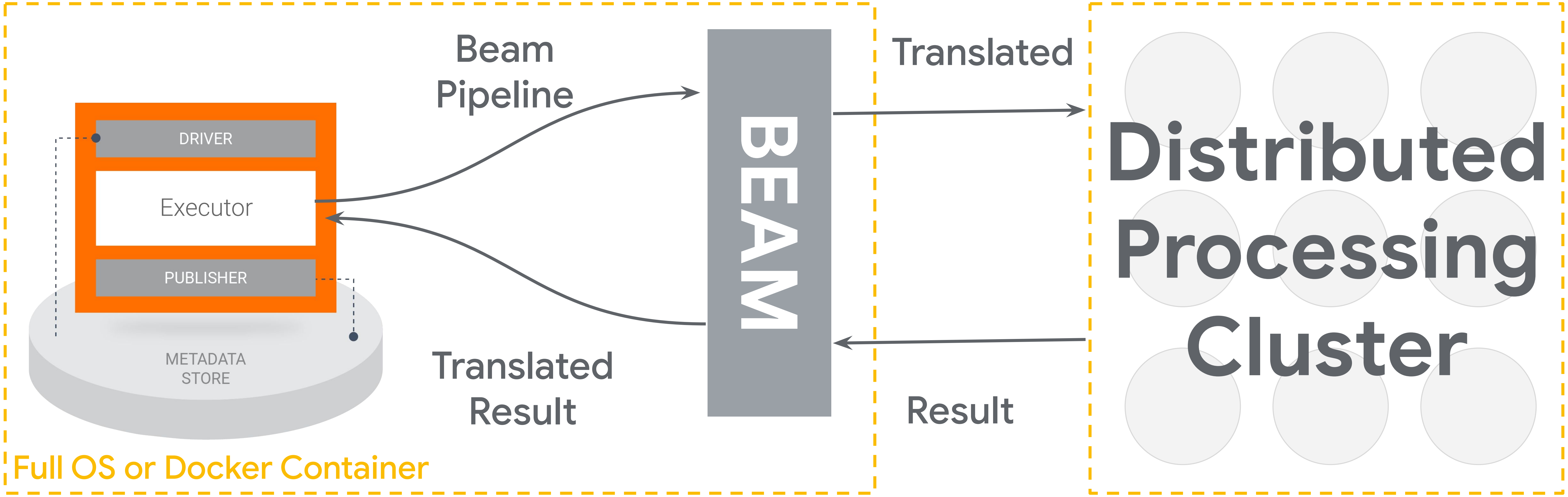


Recall: Apache Beam scales TFX component libraries



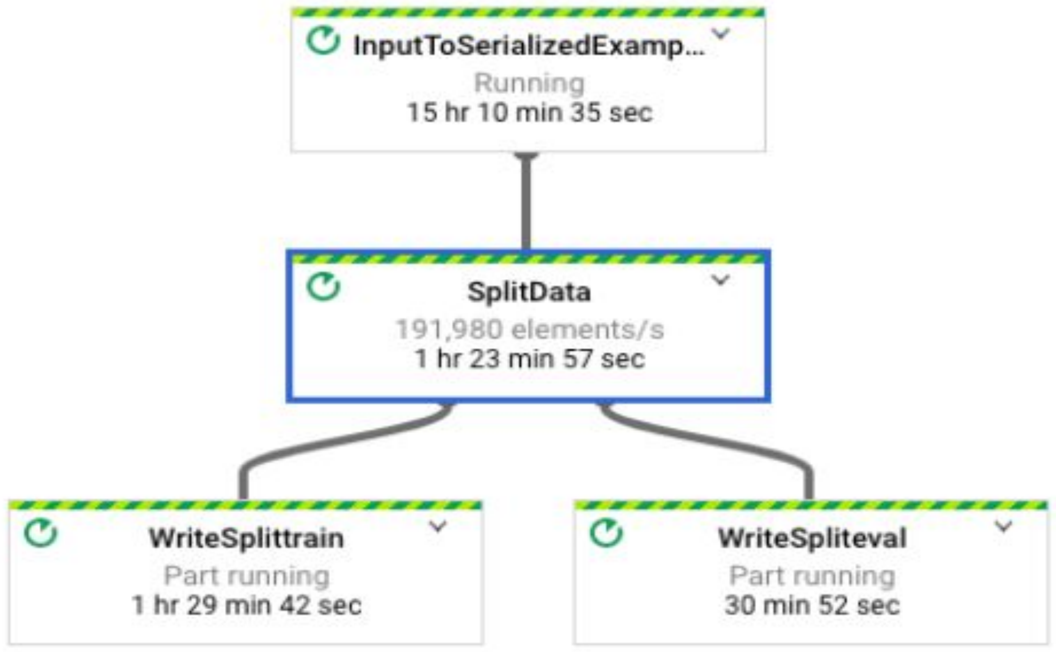
Google Cloud

How TFX Components Use Beam Orchestrator

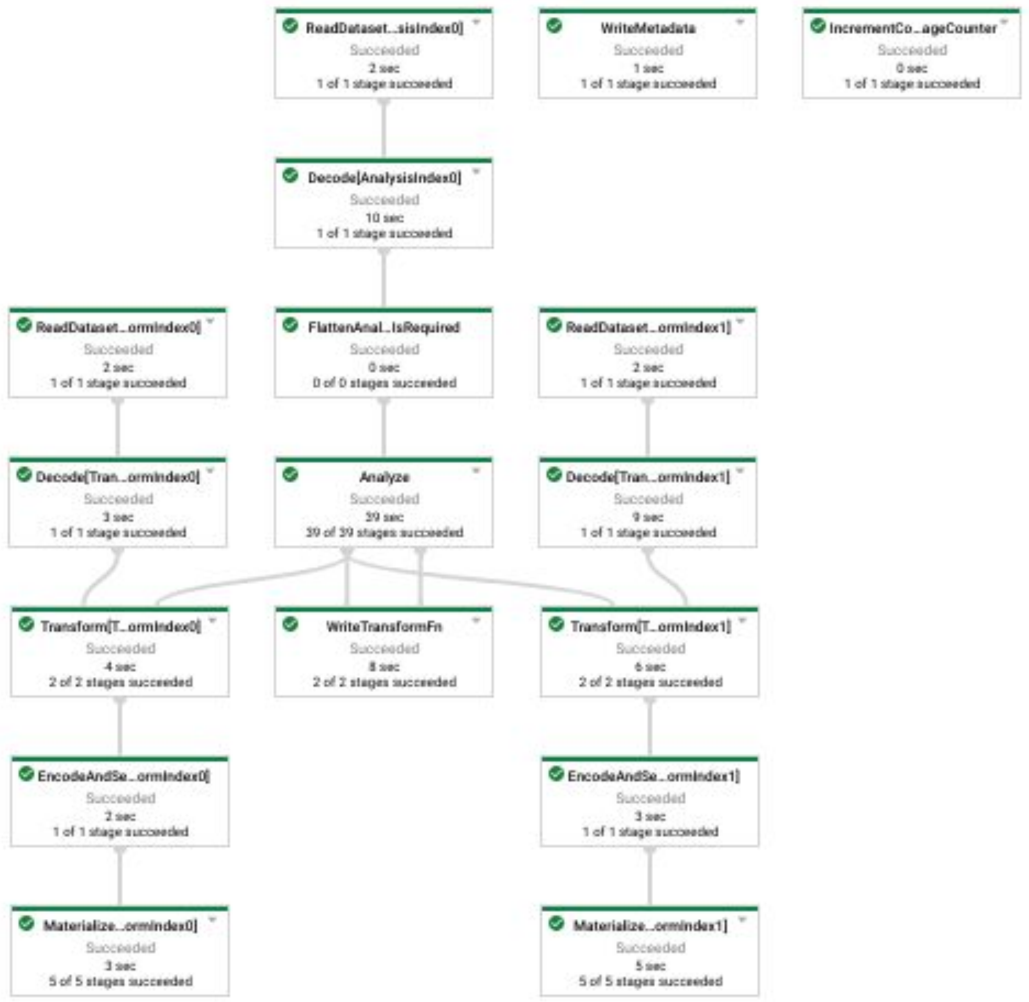


TFX data processing with Apache Beam

ExampleGen as a Beam pipeline



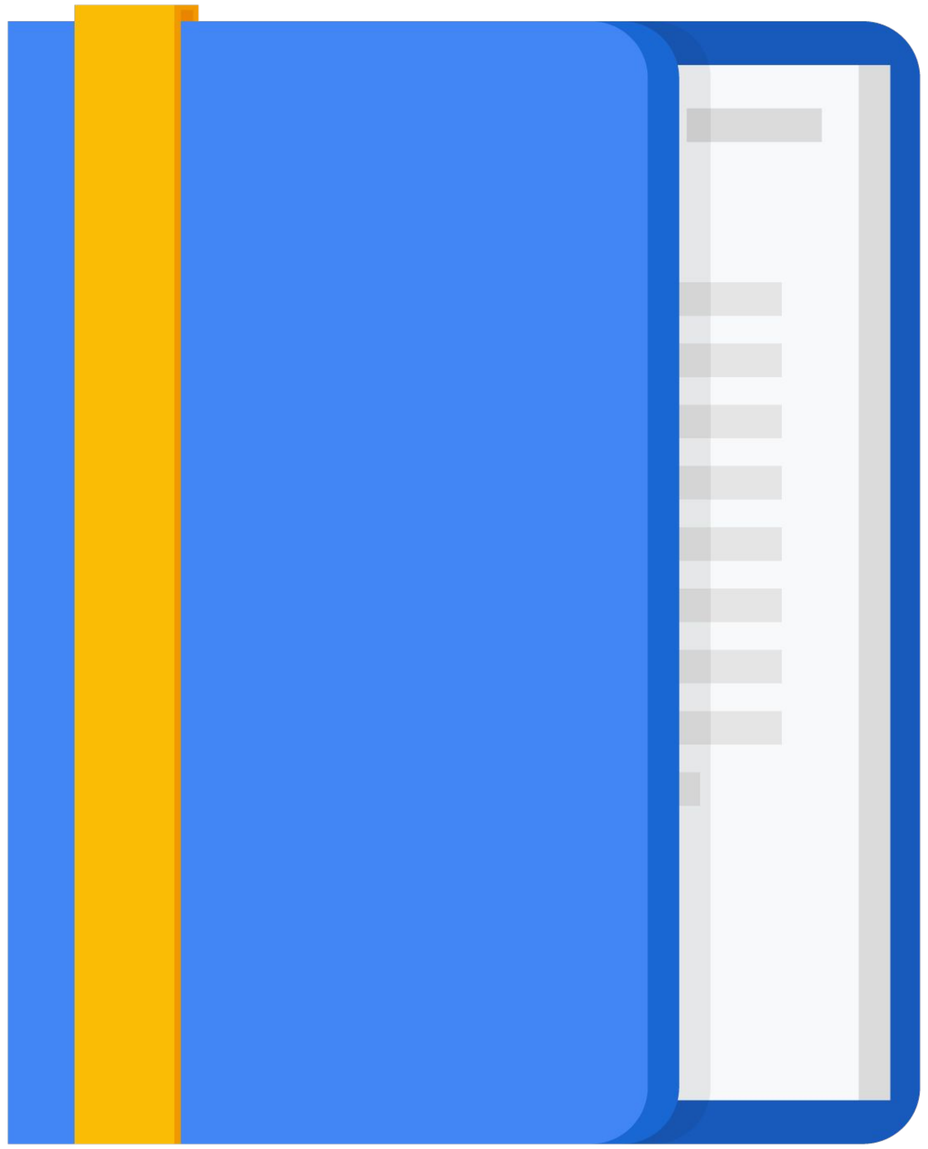
Transform as a Beam pipeline



Agenda

TFX orchestrators

TFX pipelines on Cloud AI Platform



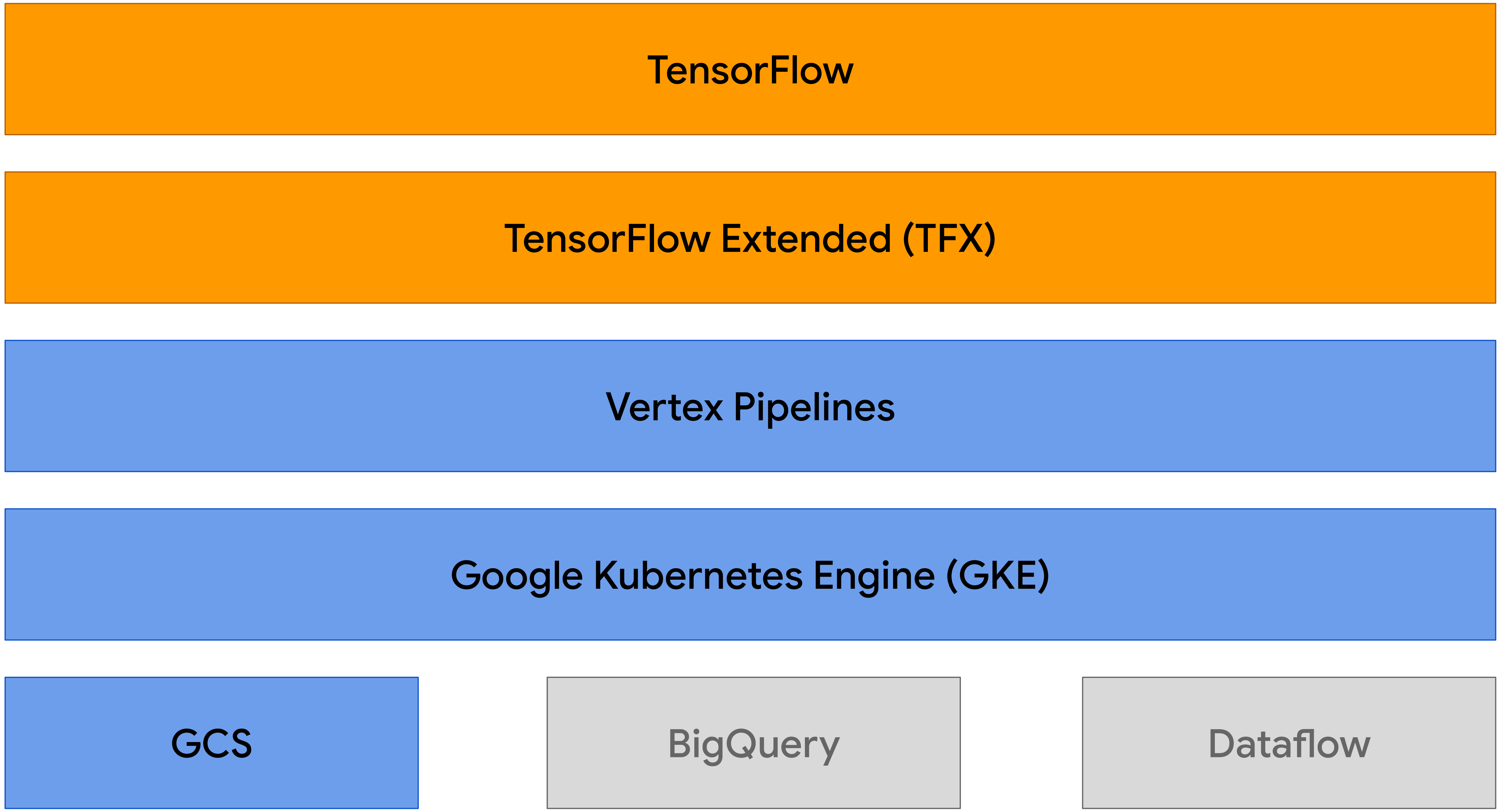
TFX Command Line Interface (CLI): simplified task-based pipeline operations

```
tfx command-group command flags
```

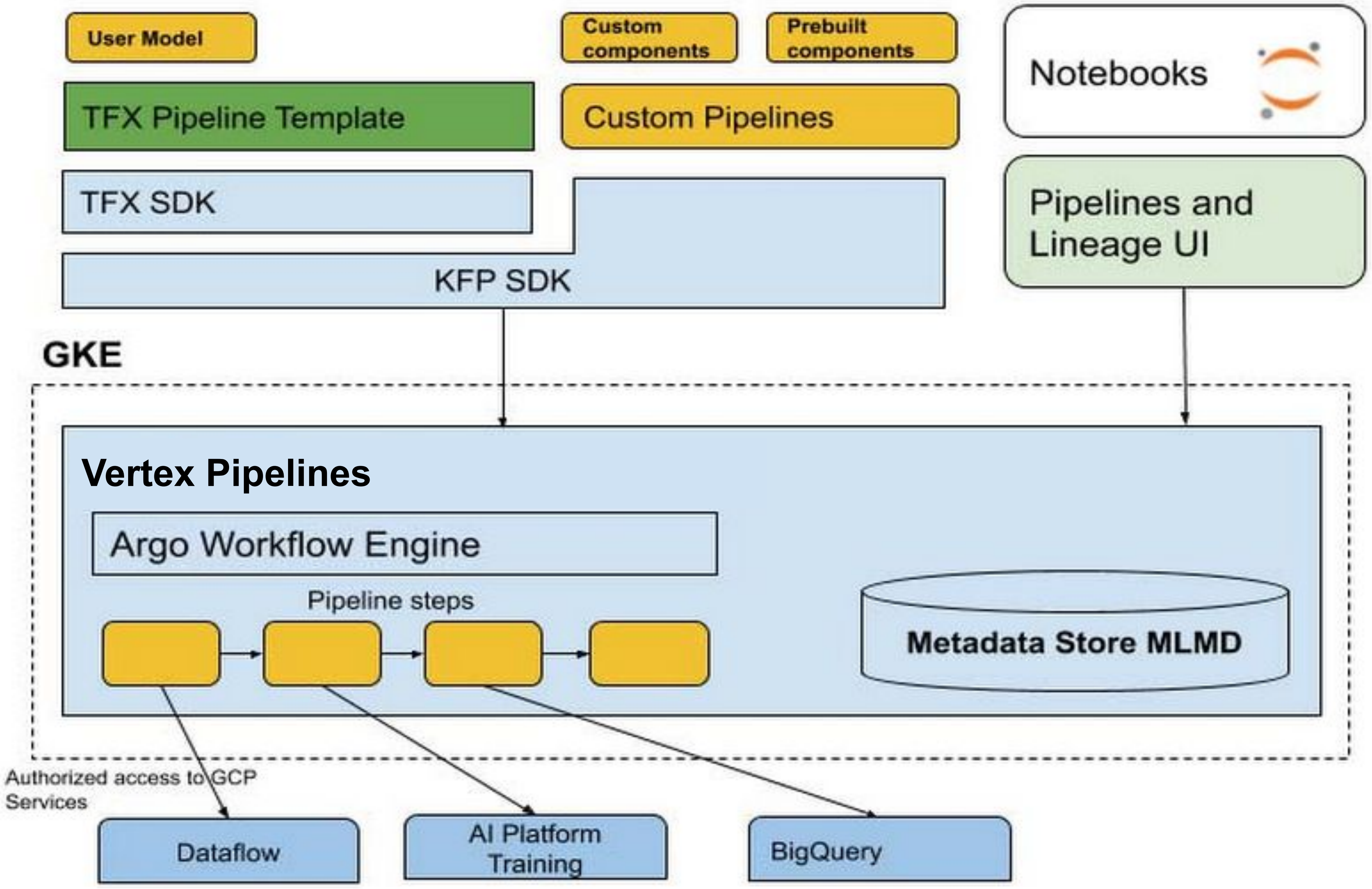
The following **command-group** options are currently supported:

- tfx pipeline - Create and manage TFX pipelines.
- tfx run - Create and manage runs of TFX pipelines on various orchestration platforms.
- tfx template - Experimental commands for listing and copying TFX pipeline templates.

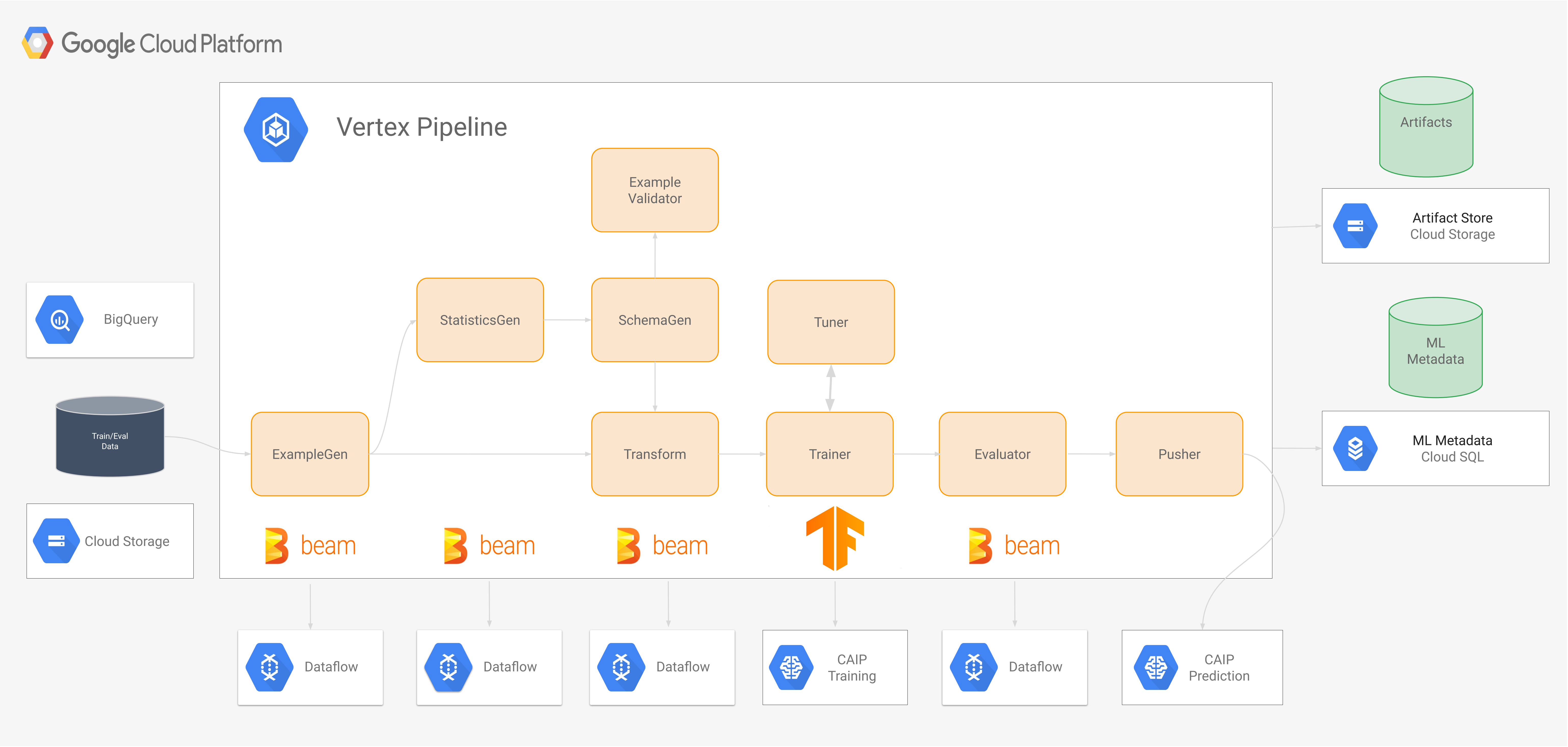
High level architecture of TFX on Google Cloud



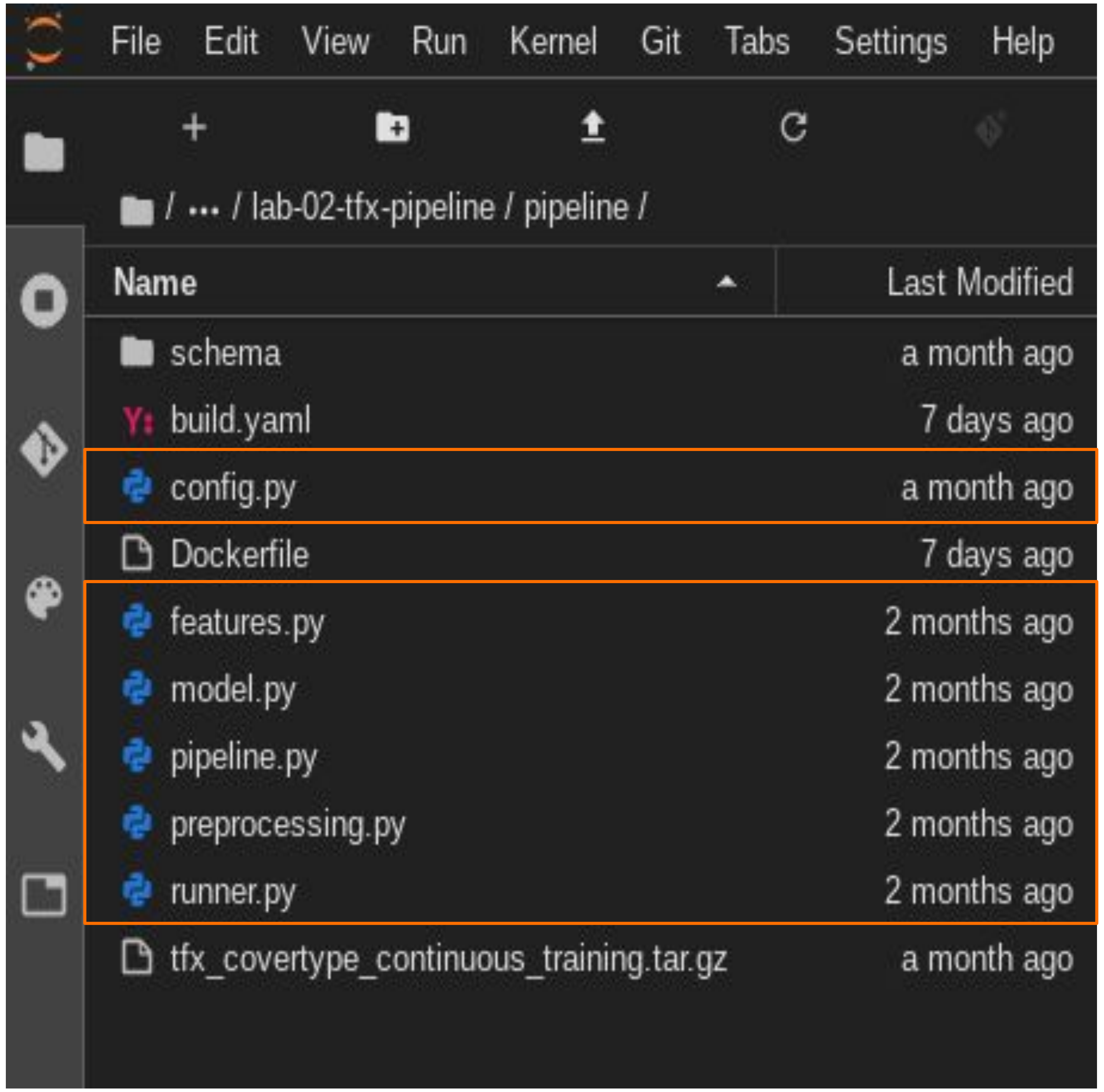
Details view: TFX pipelines run on Google Cloud



TFX integrations with GCP services



How to implement and run a TFX pipeline?



config.py - module configures the default values for the environment specific settings and the default values for the pipeline runtime parameters. The default values can be overwritten at compile time in a set of environment variables.

pipeline.py - module contains the TFX DSL defining the workflow implemented by the pipeline.

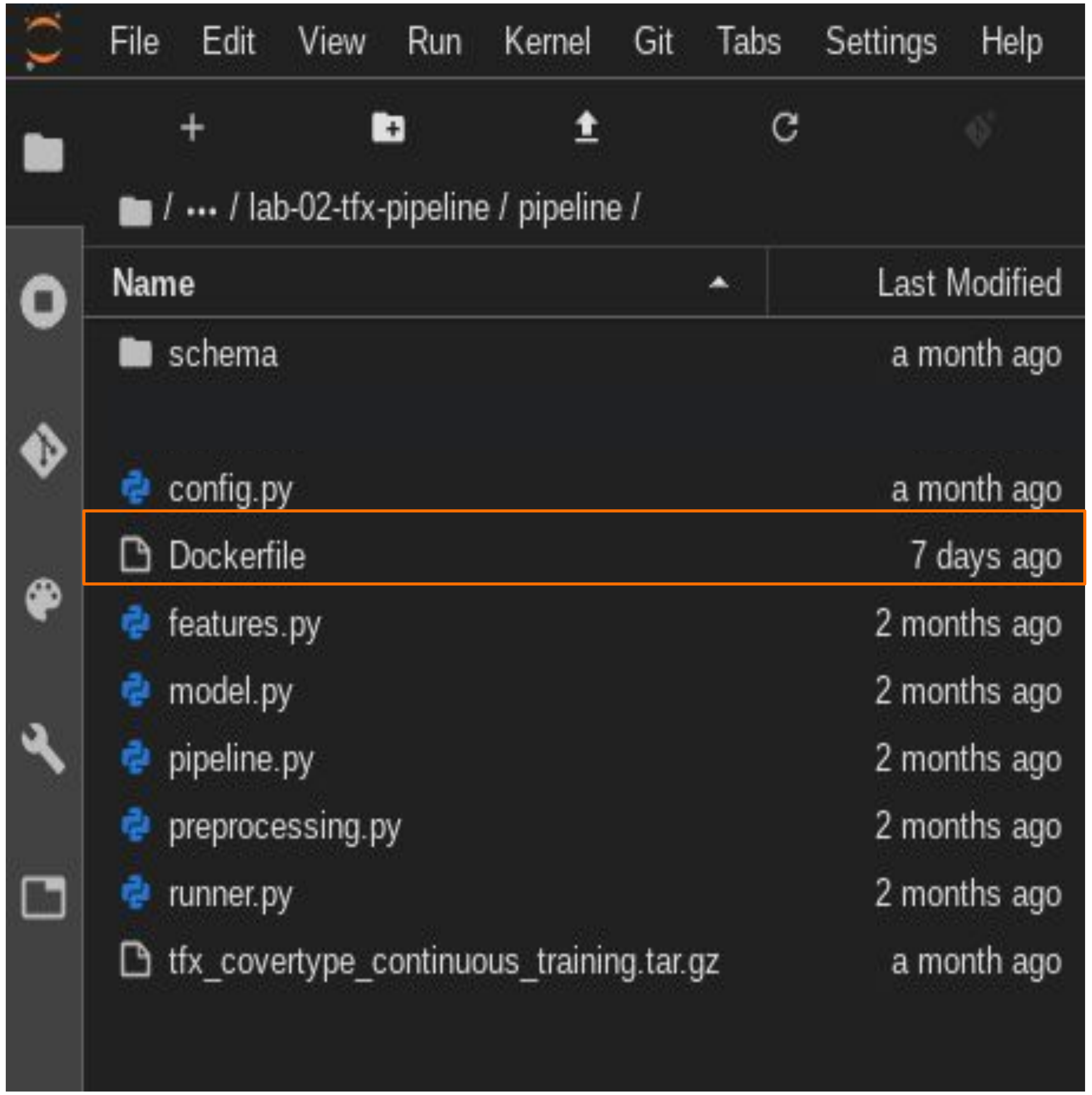
runner.py - module configures and executes KubeflowV2DagRunner. At compile time, the KubeflowV2DagRunner.run() method converts the TFX DSL into the pipeline package in JSON.

model.py - module implements the training logic for the Train component.

features.py - module contains feature definitions common across preprocessing.py and model.py.

preprocessing.py - module implements the data preprocessing logic the Transform component.

Package your TFX pipeline as a Docker container



Dockerfile

```
FROM gcr.io/tfx-oss-public/tfx:1.4.0

RUN pip install -U pip

RUN pip install google-cloud-aiplatform==1.7.1 kfp==1.8.1
```

Publish pipeline Container to Container Registry

pipeline.py

```
from tfx.v1.components import CsvExampleGen, SatisticsGen, #etc.
from tfx.orchestration.pipeline import Pipeline


def create_pipeline(...):

   generate_examples = CsvExampleGen(...)
   generate_statistics = StatisticsGen(...)
   ....
   deploy =  Pusher(...)


    return Pipeline(
        pipeline_name=pipeline_name,
        pipeline_root=pipeline_root,
        components=[
           generate_examples, generate_statistics, import_schema, infer_schema, validate_stats, transform,
            train, resolve, analyze, infra_validate, deploy
        ],
```

runner.py

```
from tfx.orchestration.kubeflow.v2 import kubeflow_dag_runner

from pipeline import create_pipeline

[...]

pipeline =   create_pipeline(
      pipeline_name=Config.PIPELINE_NAME,
      pipeline_root=Config.PIPELINE_ROOT,
      data_root_uri=Config.DATA_ROOT_URI,
      train_steps=Config.TRAIN_STEPS,
      eval_steps=Config.EVAL_STEPS,
)


kubeflow_dag_runner.KubeflowV2DagRunner(config=runner_config).run(pipeline)
```

config.py

```
class Config:
    """Sets configuration vars."""

    PROJECT_ID = os.getenv("PROJECT_ID")
    REGION = os.getenv("REGION", "us-central1")
    ARTIFACT_STORE = os.getenv("ARTIFACT_STORE", f"gs://{PROJECT_ID}")
    PIPELINE_NAME = os.getenv("PIPELINE_NAME", "tfxcovertype")

    # etc.
```

Compile the TFX pipeline with the TFX CLI

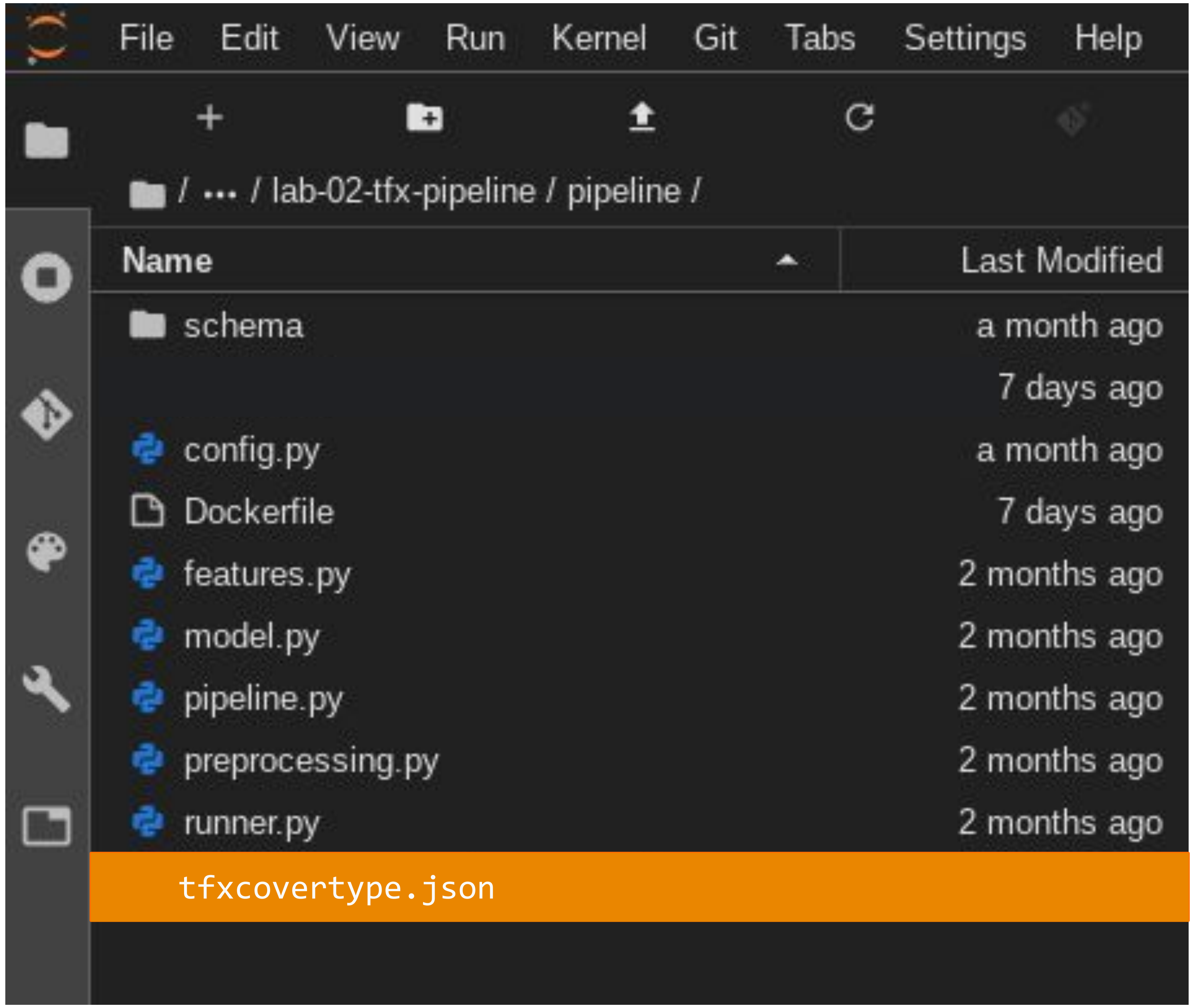
1. Define TFX runtime parameters as environment variables

```
%env PIPELINE_NAME={PIPELINE_NAME}
%env DATA_ROOT_URI={DATA_ROOT_URI}
%env TFX_IMAGE_URI={TFX_IMAGE_URI}
%env PIPELINE_JSON={PIPELINE_JSON}
%env TRAIN_STEPS={TRAIN_STEPS}
%env EVAL_STEPS={EVAL_STEPS}
```

2. Use TFX CLI to compile your pipeline.

```
!tfx pipeline compile --engine vertex --pipeline_path runner.py
```

Deploy your pipeline package to Cloud AI Platform

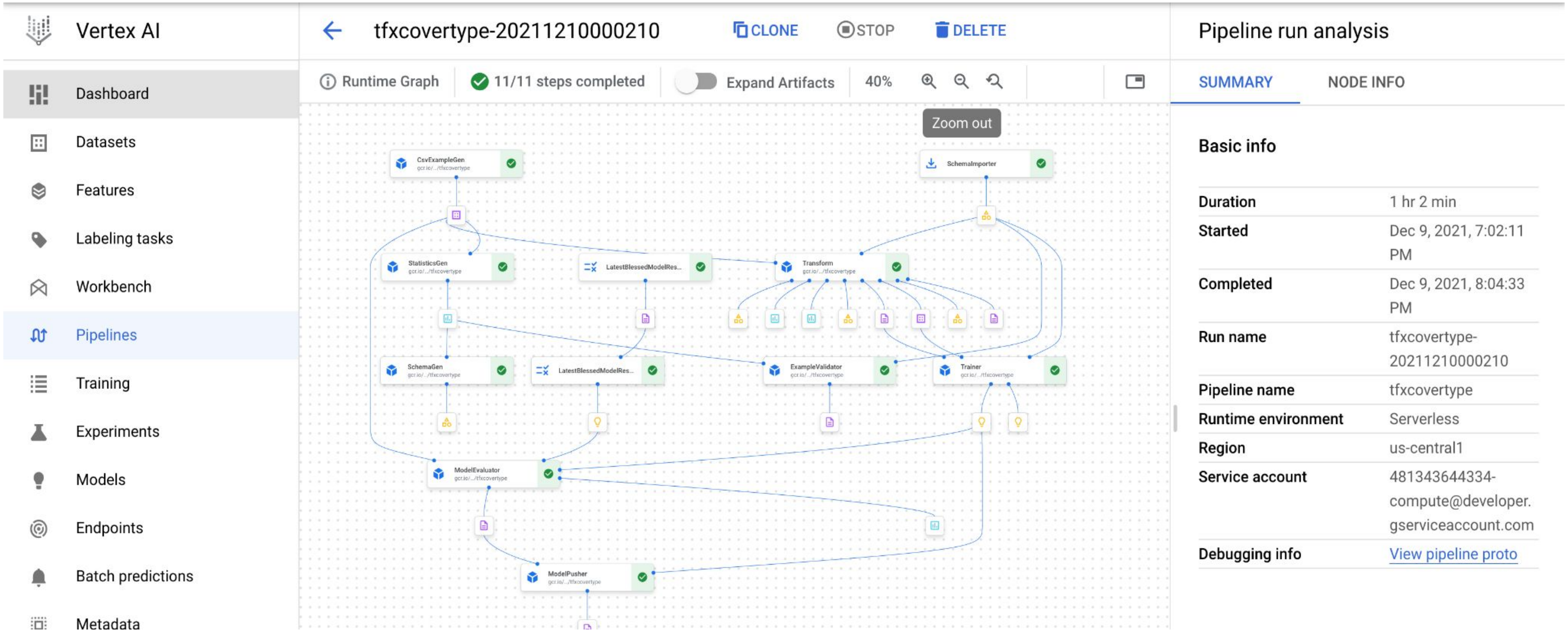


3. Use Vertex SDK to deploy your pipeline.

```
from google.cloud import aiplatform as vertex_ai


vertex_ai.init(project=PROJECT_ID, location=REGION)

pipeline = vertex_ai.PipelineJob(
    display_name="tfxcovertype4",
    template_path=PIPELINE_JSON,
    enable_caching=False,
)

pipeline.run()
```

Trigger model training on Cloud AI Platform

Create and monitor pipeline runs from Vertex Pipelines



Lab

TFX pipelines on Cloud AI Platform

tfx_pipelines/pipeline/labs/tfx_pipeline_vertex.ipynb