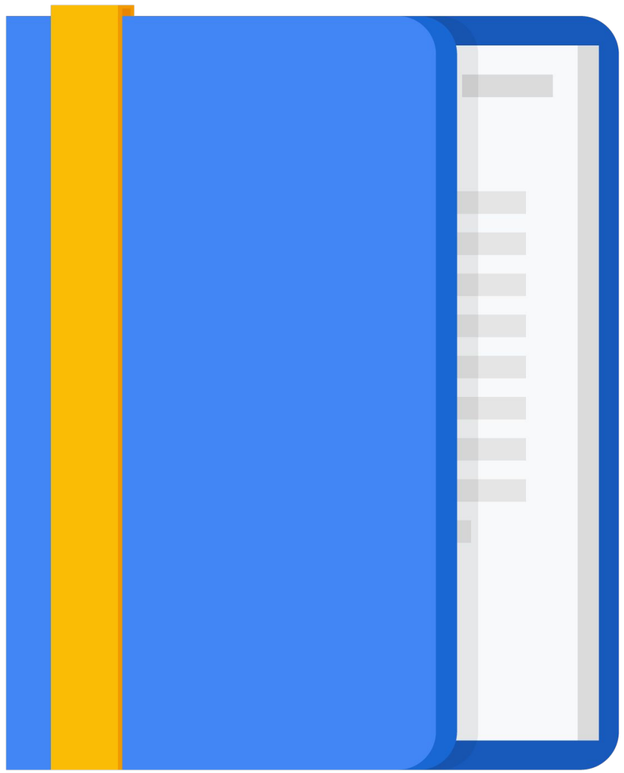
TFX pipeline orchestration and workflows on Google Cloud

Doug Kelly
ML Solutions Engineer, Google Cloud

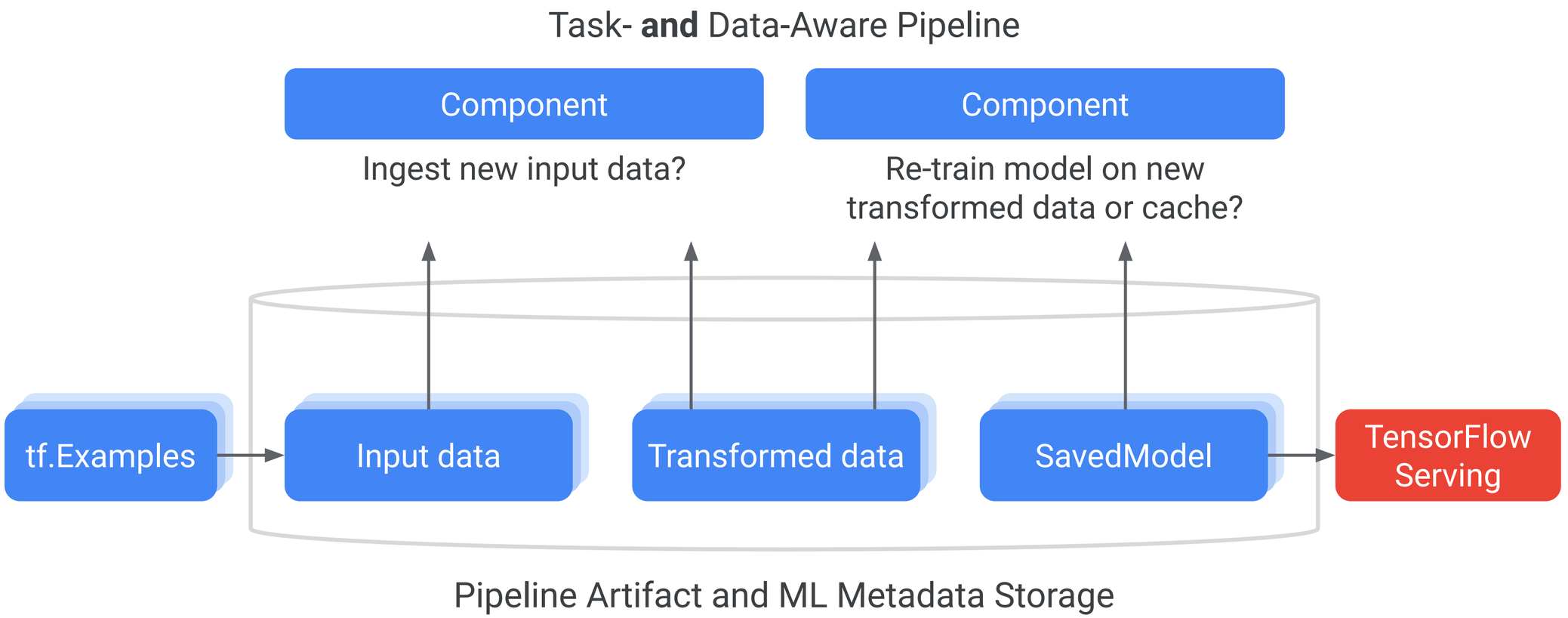
Agenda

TFX orchestrators

TFX on Cloud AI Platform



Why orchestrate your ML workflows?

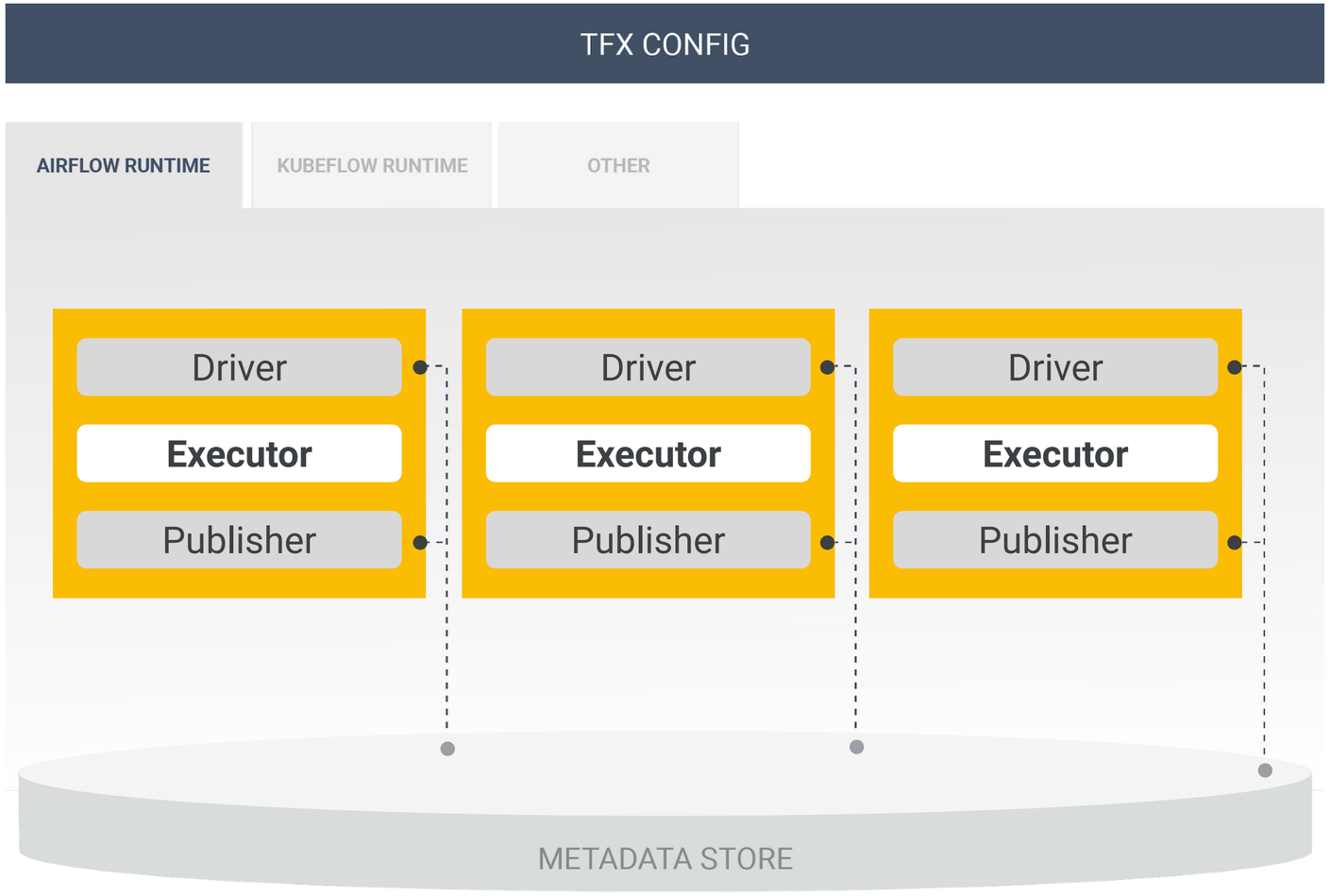


TFX Orchestration in a Notebook

```
context = InteractiveContext()

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

TFX pipelines are portable across Orchestrators

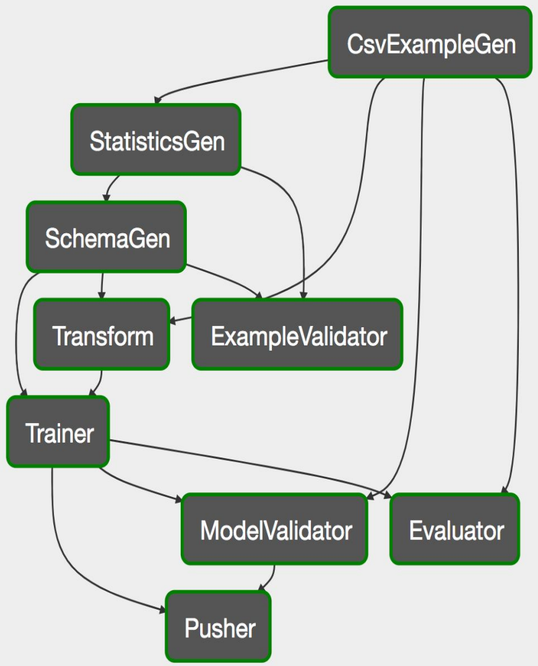
Flexible runtimes run components in sequential order using orchestration systems such as Airflow, Kubeflow, or Beam.



TFX pipelines currently support 3 orchestrators

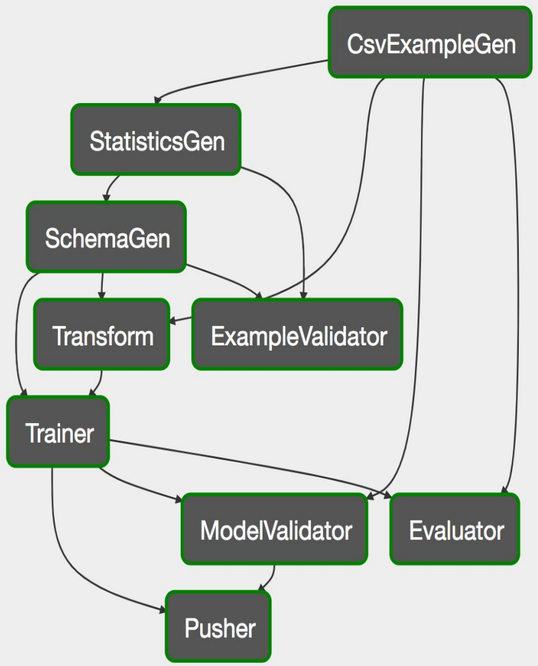
TFX pipelines currently support 3 orchestrators

Apache Airflow

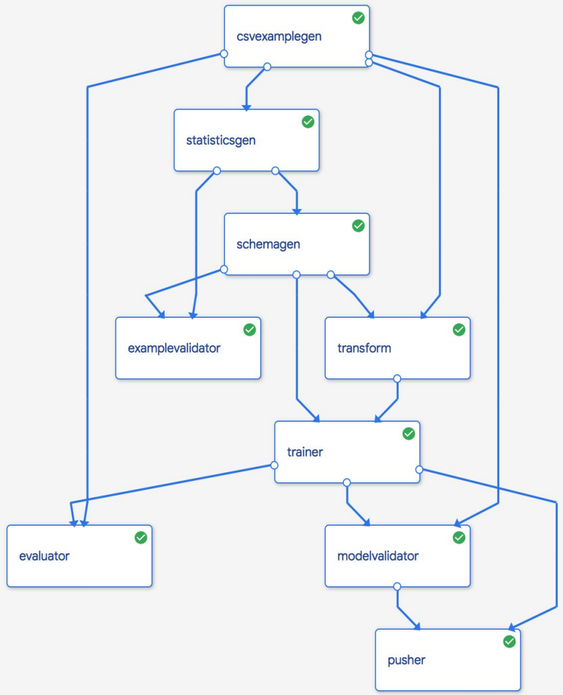


TFX pipelines currently support 3 orchestrators

Apache Airflow

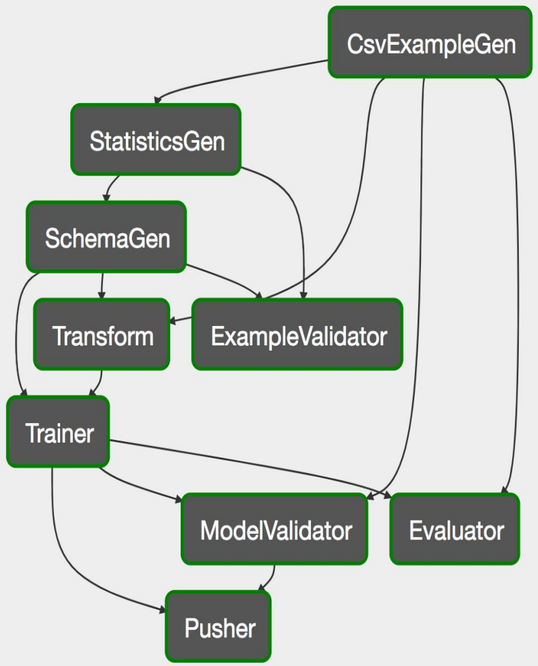


Kubeflow Pipelines

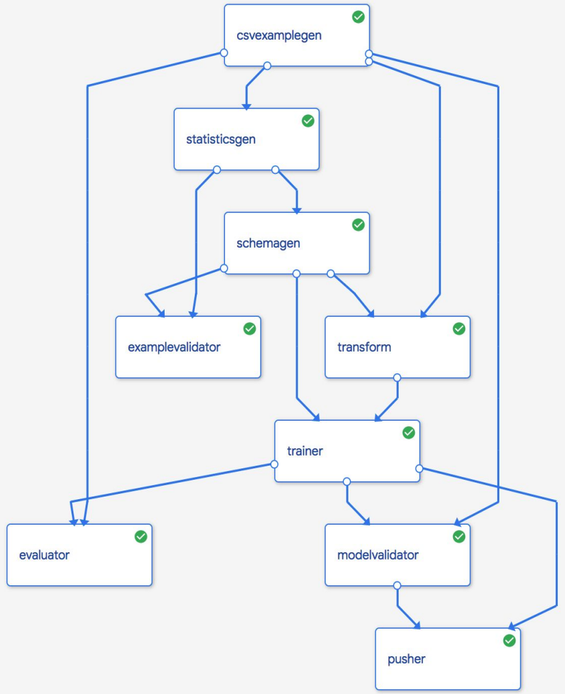


TFX pipelines currently support 3 orchestrators

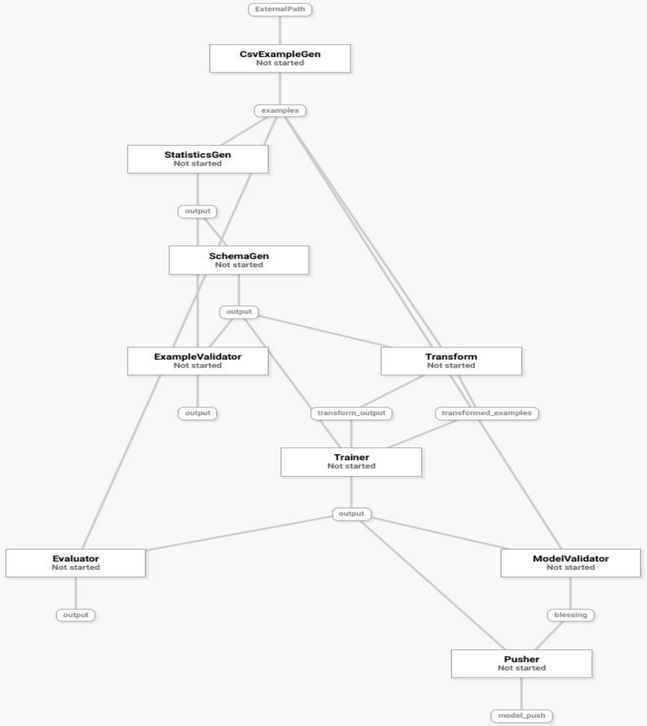
Apache Airflow



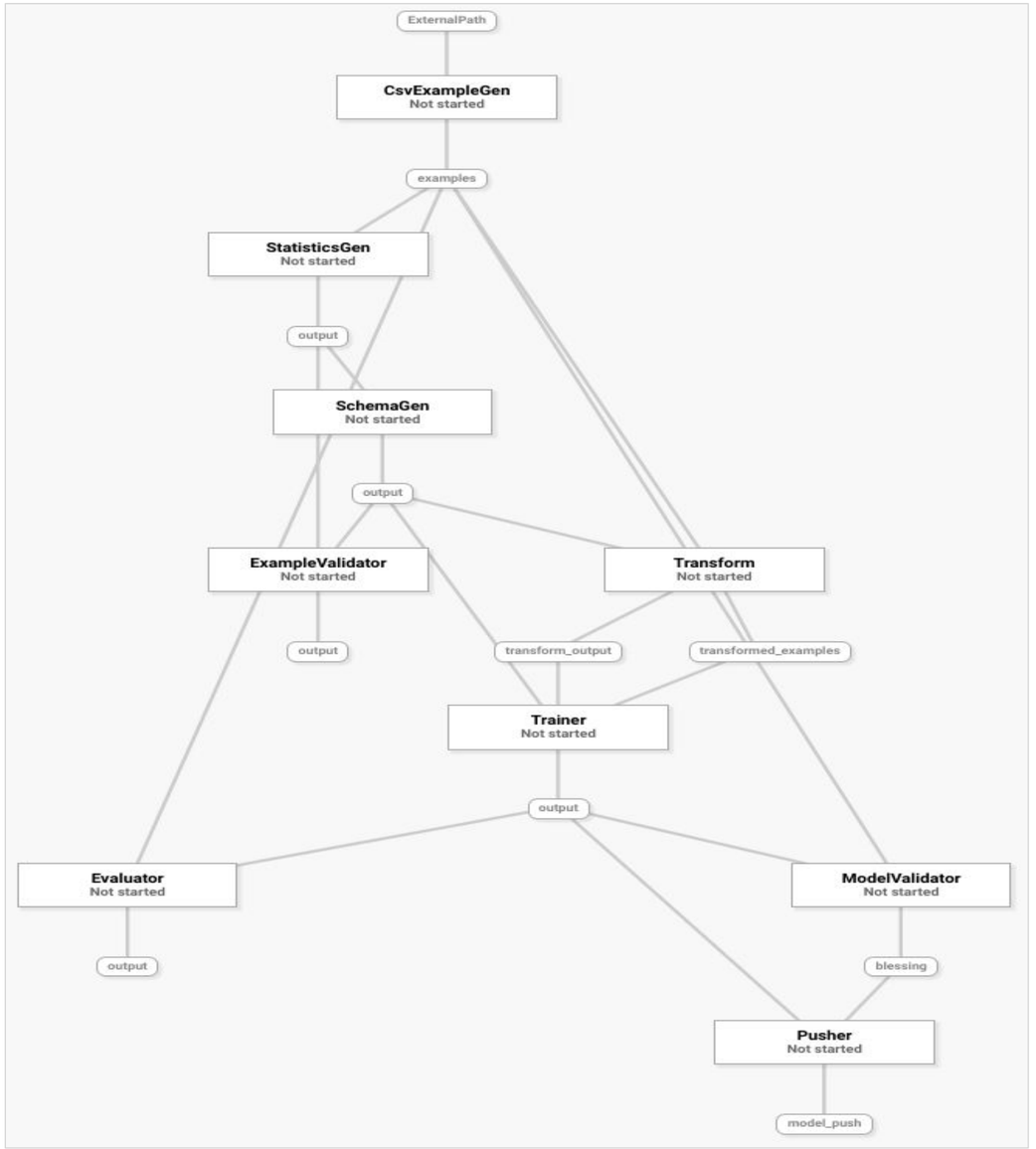
Kubeflow Pipelines



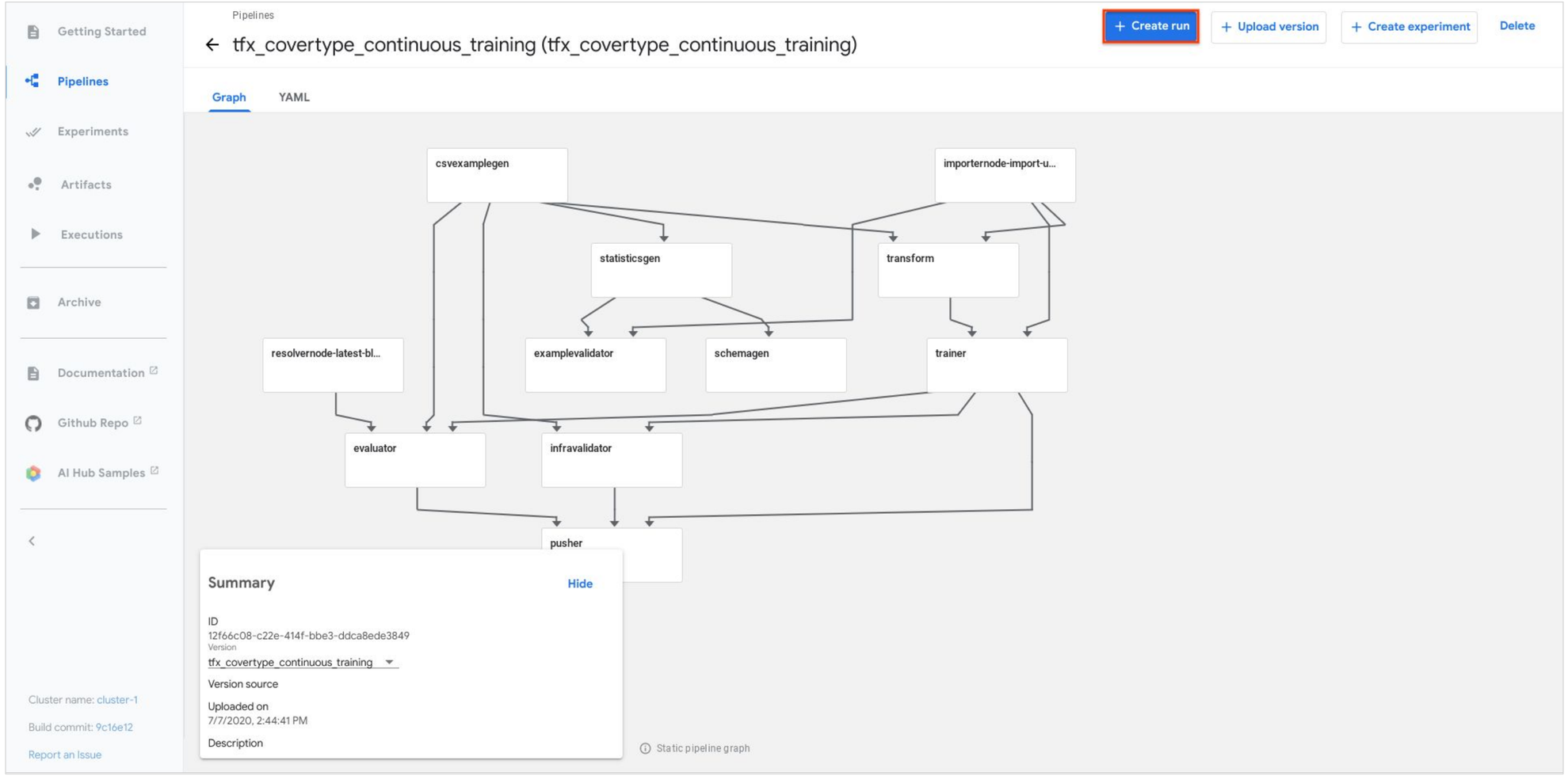
Apache Beam



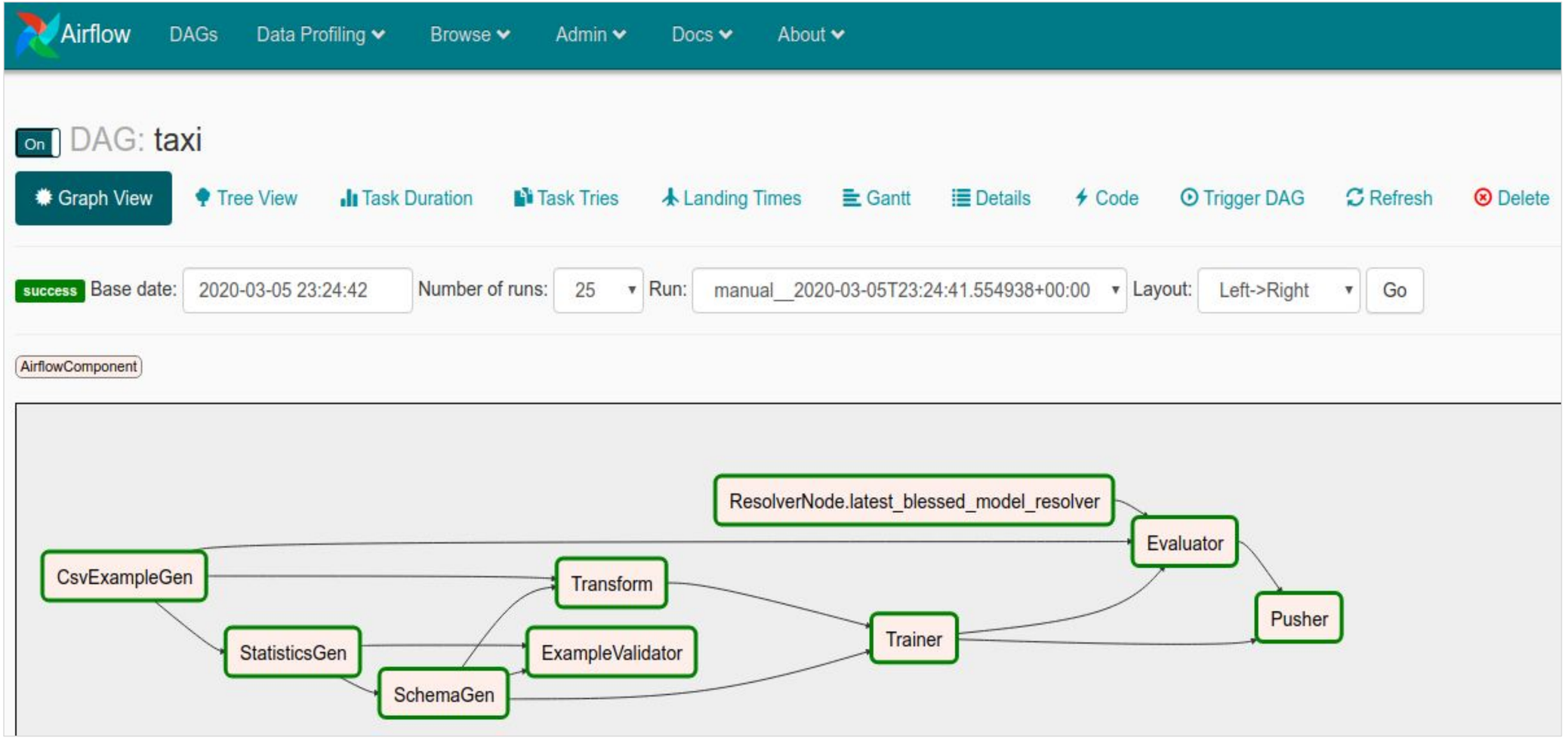
TFX on Apache Beam Orchestrator



TFX on Kubeflow Pipelines



TFX on Airflow



TFX command line interface (CLI): simplified task-based pipeline operations

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

TFX command line interface (CLI): simplified task-based pipeline operations

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

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

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 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.

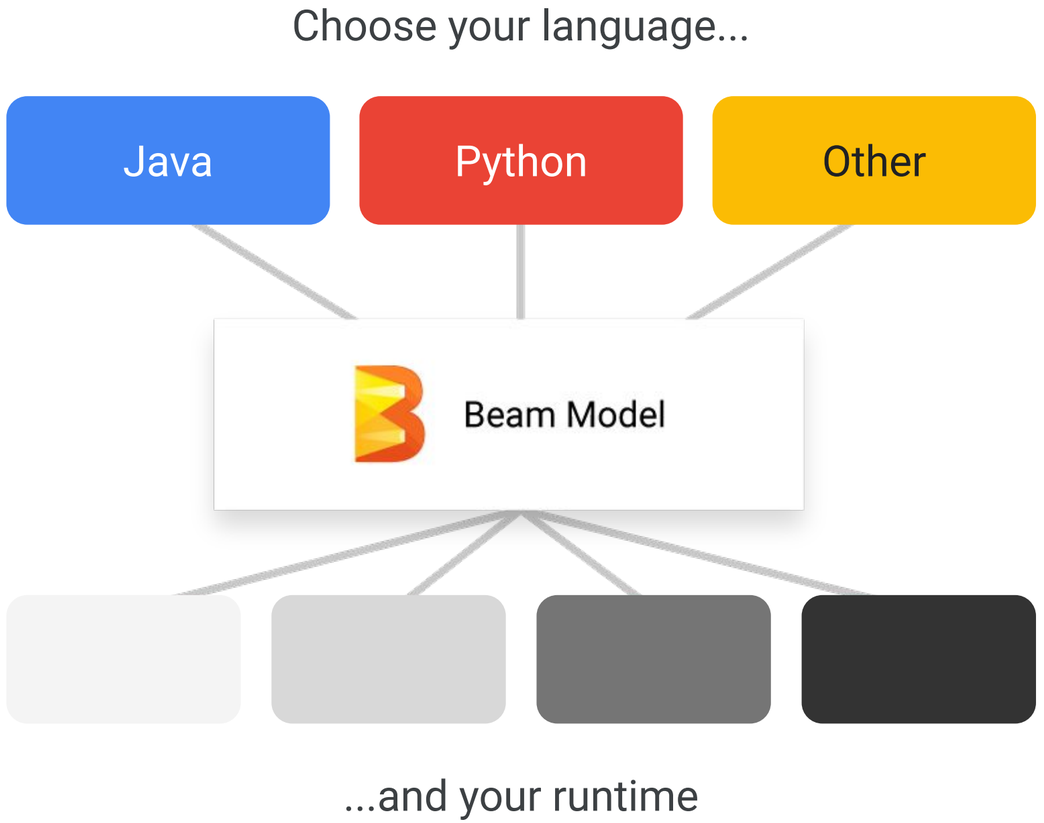
TFX command line interface (CLI): simplified task-based pipeline operations

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

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

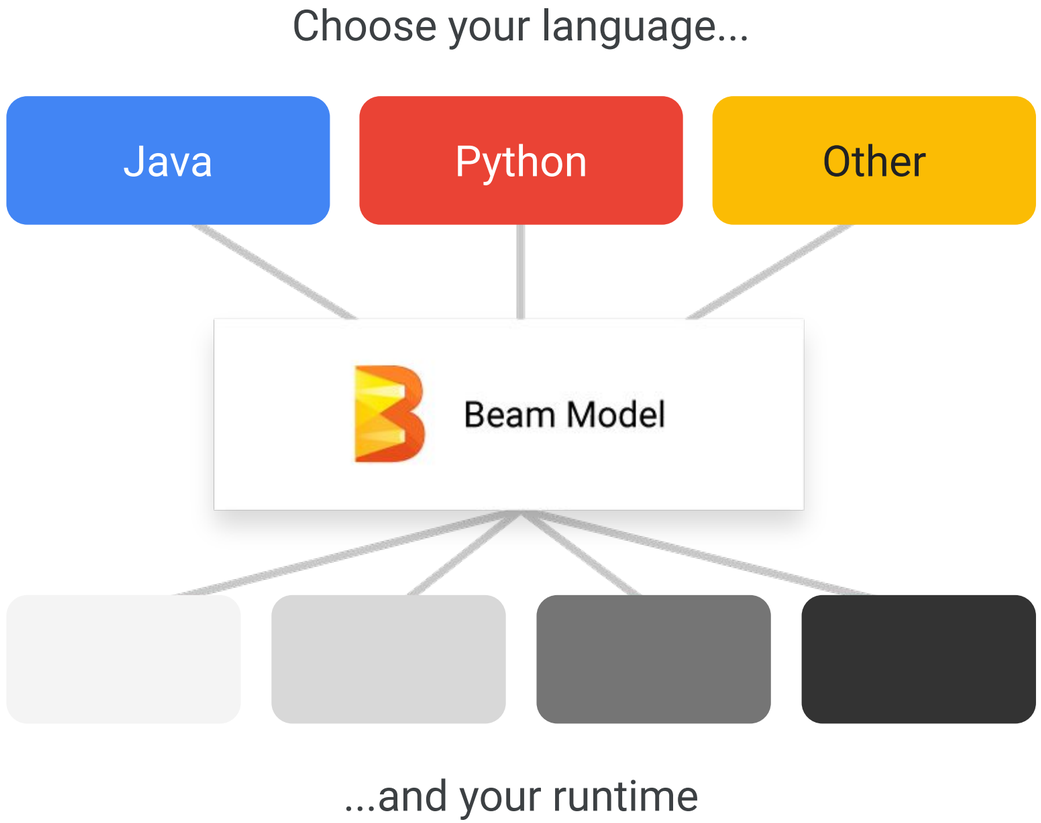
- tfx pipeline - Create and manage TFX pipelines.
- tfx run - Create and manage runs of TFX pipelines on various orchestration platforms.
- tfx template - Experimental commands for listing and copying TFX pipeline templates.

Apache Beam is a key data processing abstraction for TFX



Apache Beam is a key data processing abstraction for TFX

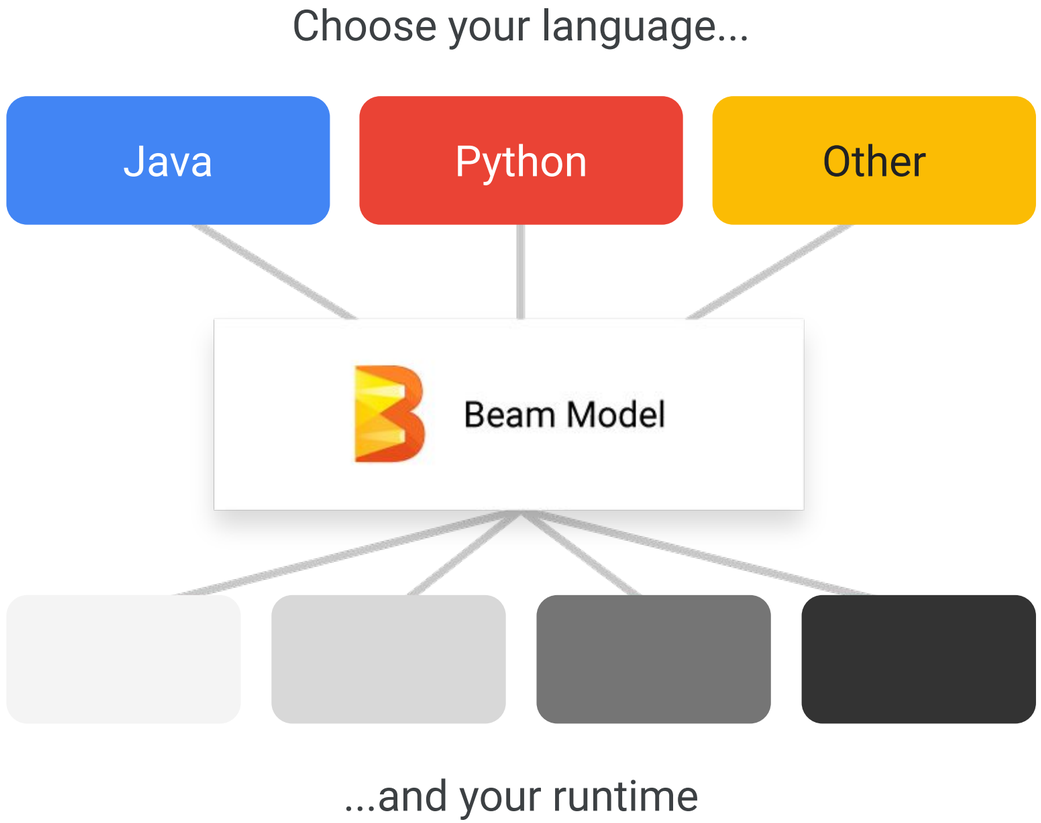
Unified: same programming model for batch and stream



Apache Beam is a key data processing abstraction for TFX

Unified: Programming model for batch and stream

Portable: Provide a choice of execution environments

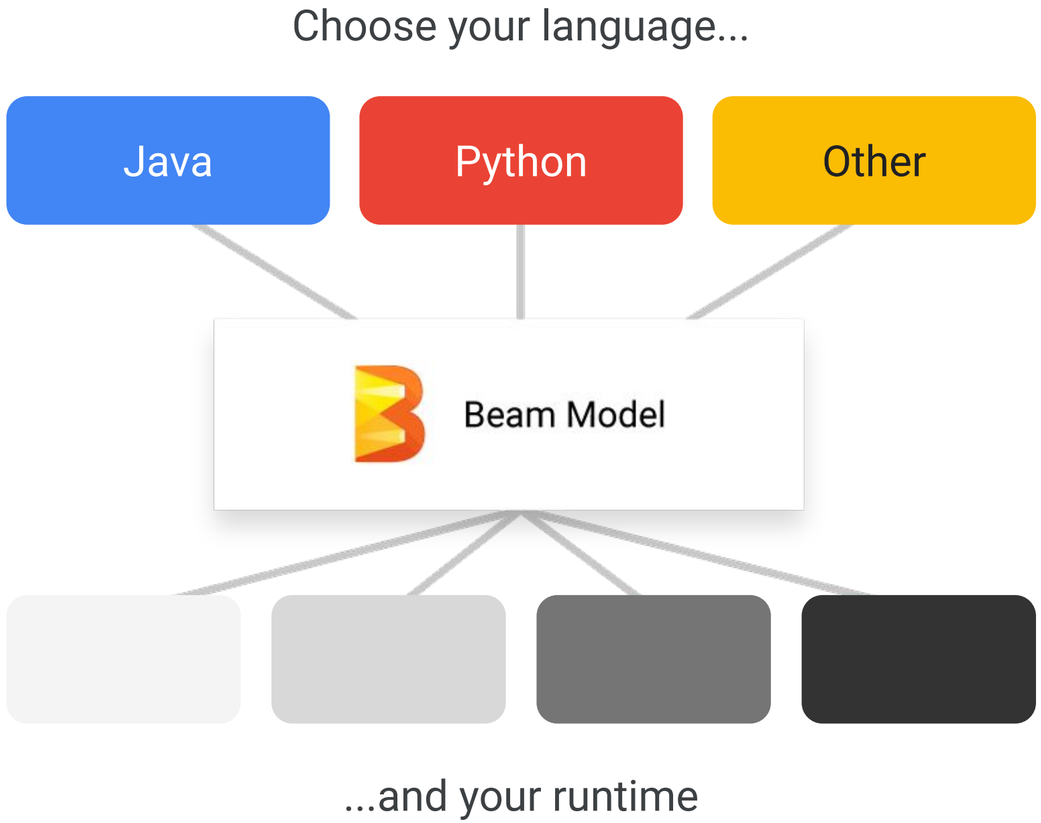


Apache Beam is a key data processing abstraction for TFX

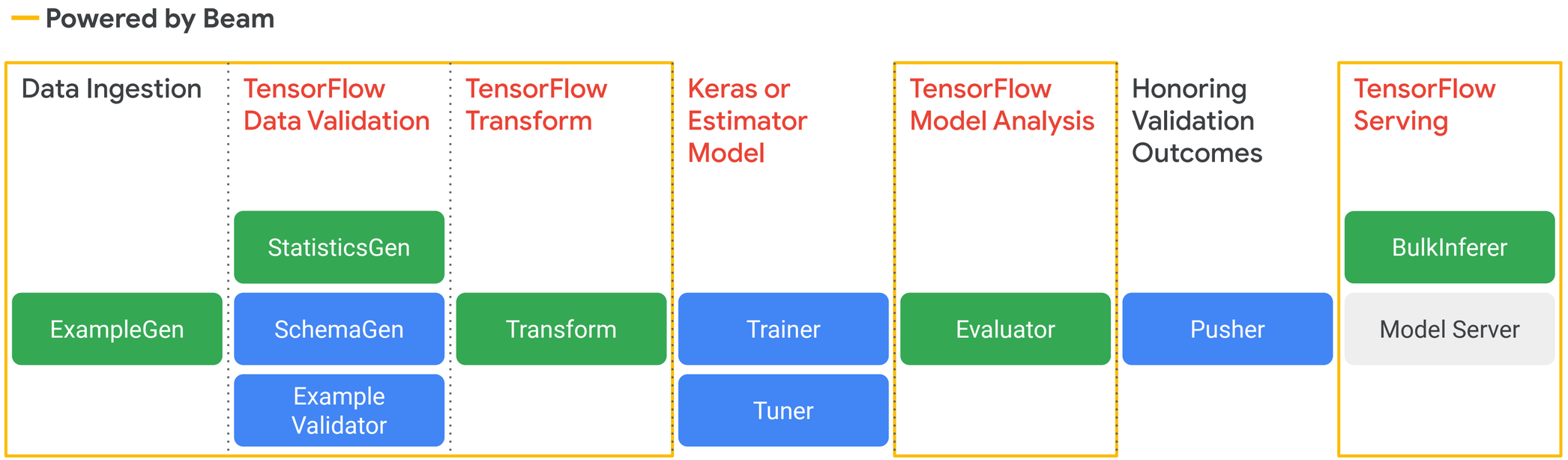
Unified: Programming model for batch and stream

Portable: Provide a choice of execution environments

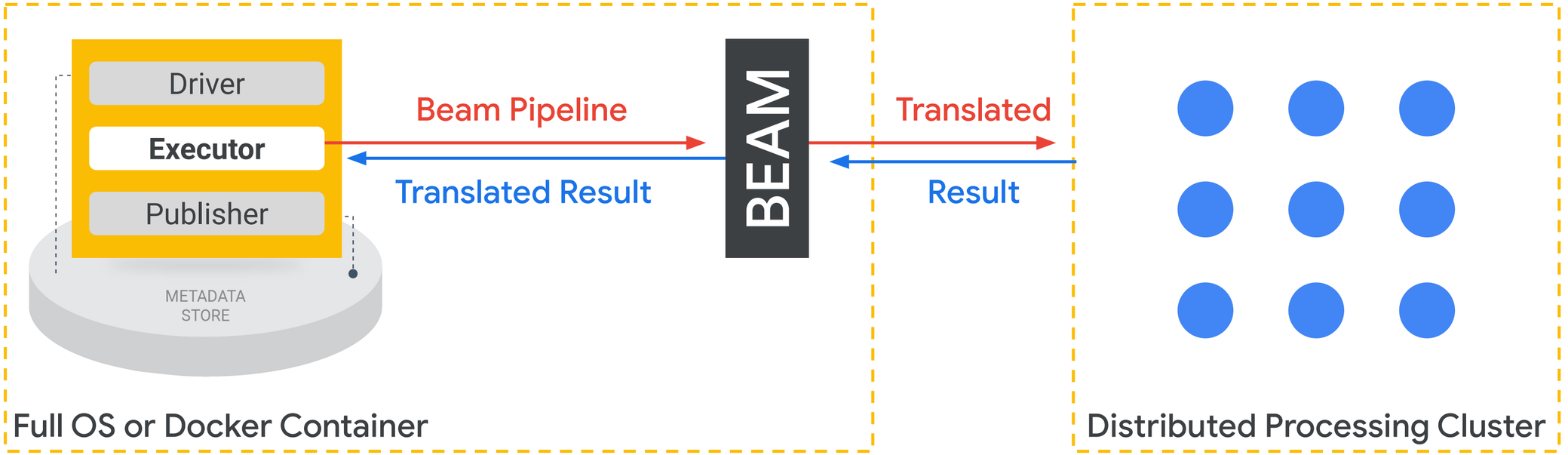
Extensible: Write and share new SDKs, IO connectors, and transforms



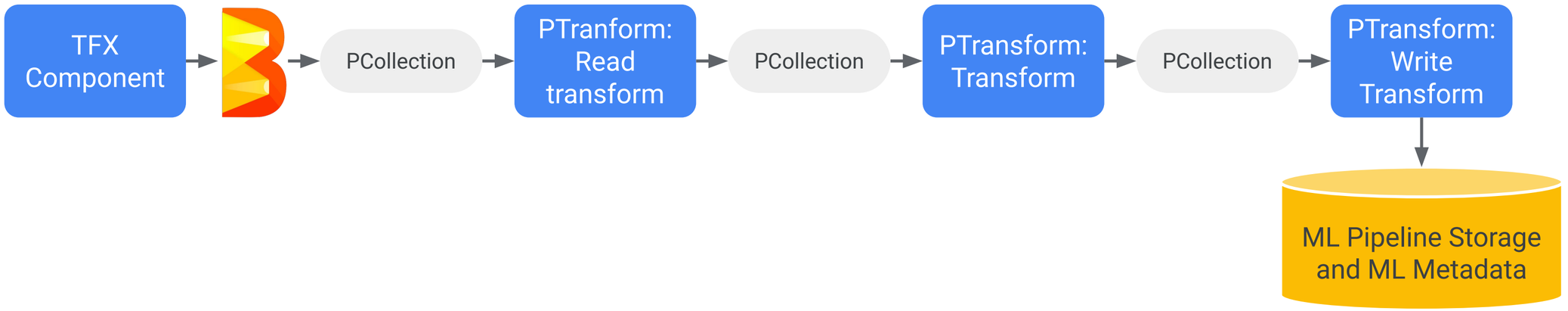
Recall: Apache Beam scales TFX component libraries



How TFX components use Beam

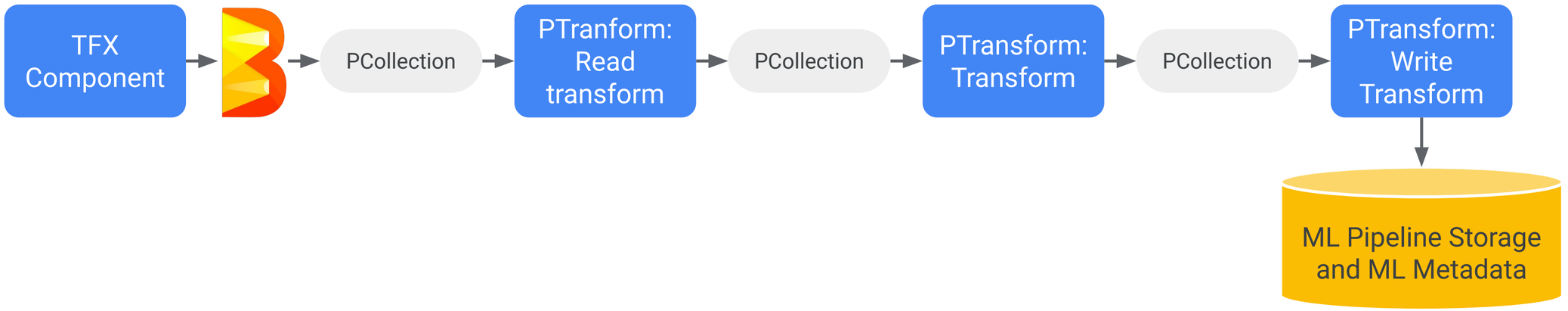


Beam primitives: Pipelines, PCollections, PTransforms, Runners



Beam primitives: Pipelines, PCollections, PTransforms, Runners

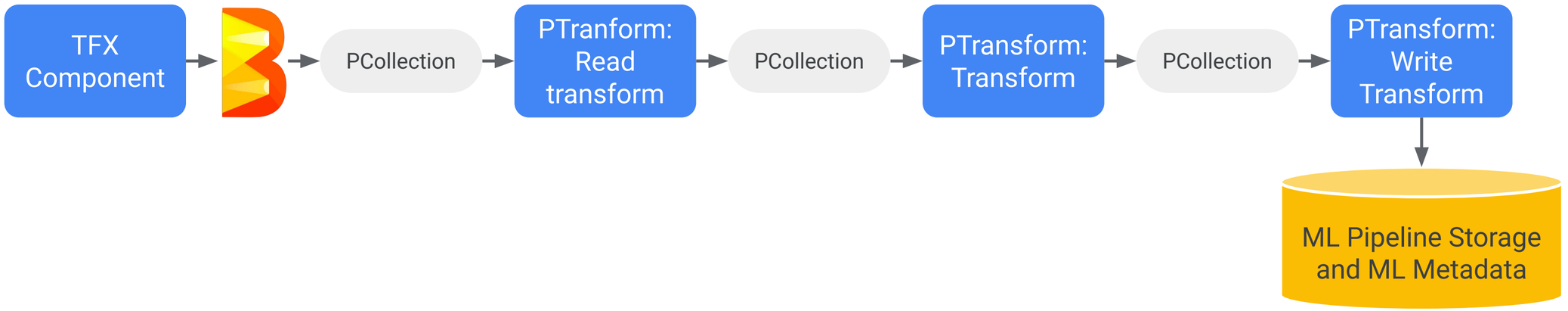
Pipeline: Entire set of operations being performed, including reading input, applying transformations, writing output, and the execution engine to be used.



Beam primitives: Pipelines, PCollections, PTransforms, Runners

Pipeline: Entire set of operations being performed, including reading input, applying transformations, writing output, and the execution engine to be used.

PCollection: Represents an unordered set of data, e.g., for input and output.

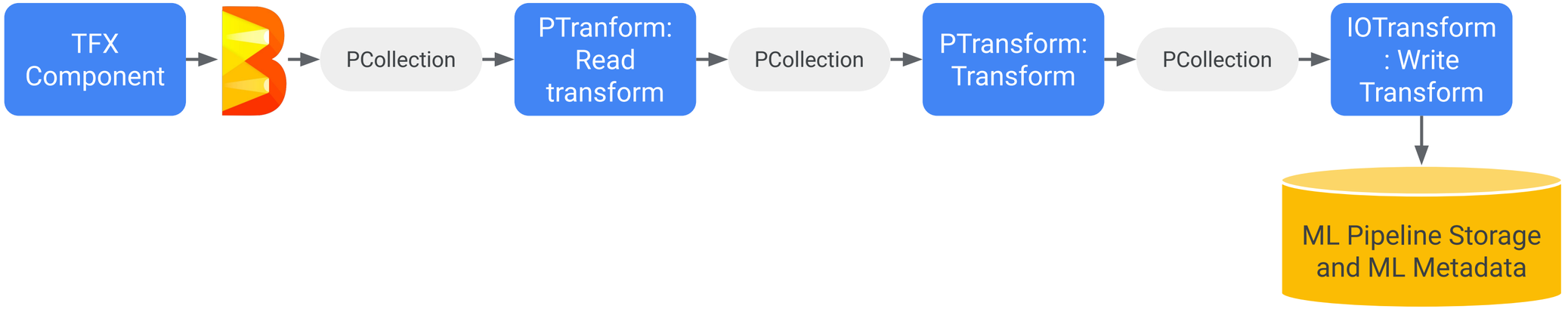


Beam primitives: Pipelines, PCollections, PTransforms, Runners

Pipeline: Entire set of operations being performed, including reading input, applying transformations, writing output, and the execution engine to be used.

PCollection: Represents an unordered set of data, e.g., for input and output.

PTransform: data processing operation that operates over 1:many PCollections. ParDO is the core parallel processing transform.



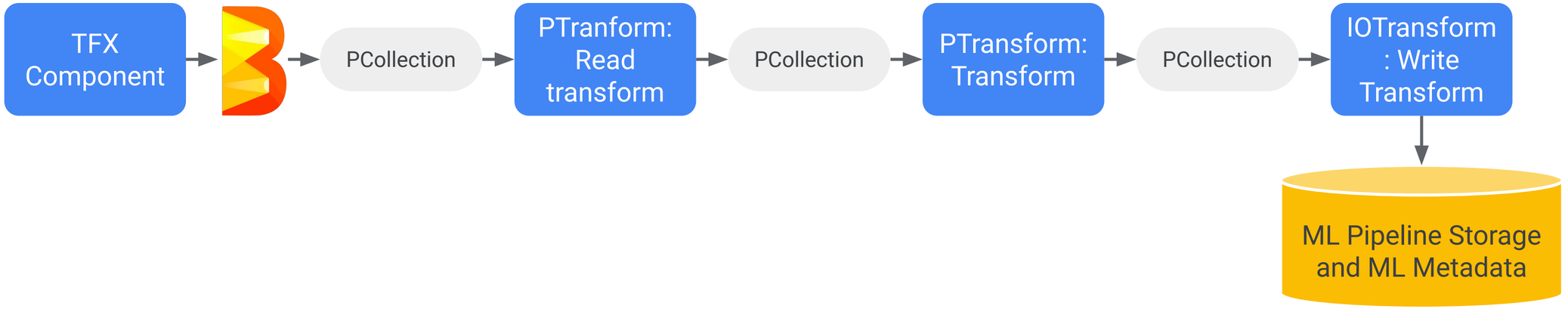
Beam primitives: Pipelines, PCollections, PTransforms, Runners

Pipeline: Entire set of operations being performed, including reading input, applying transformations, writing output, and the execution engine to be used.

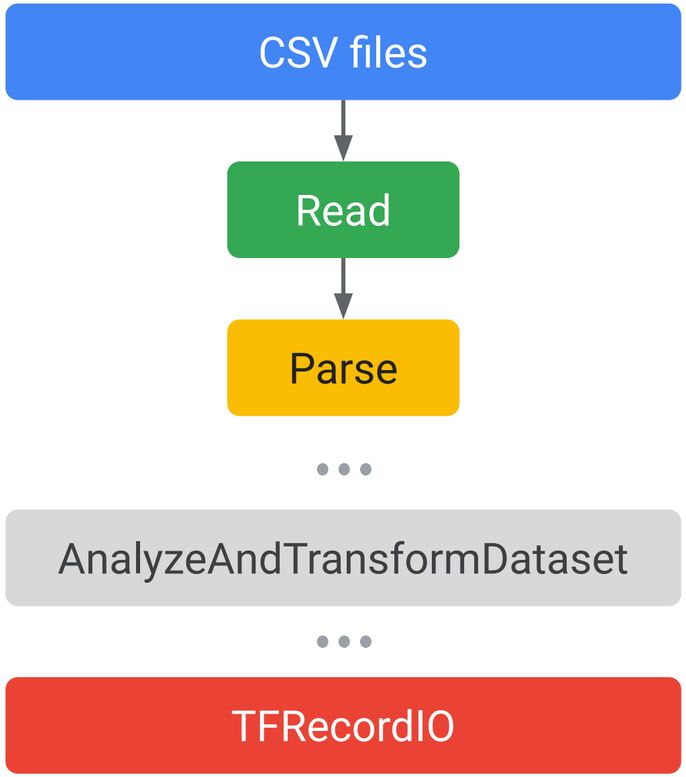
PCollection: Represents an unordered set of data, e.g., for input and output.

PTransform: data processing operation that operates over 1:many PCollections. ParDO is the core parallel processing transform.

Runner: execute and translate pipelines to massively parallel big-data processing systems.



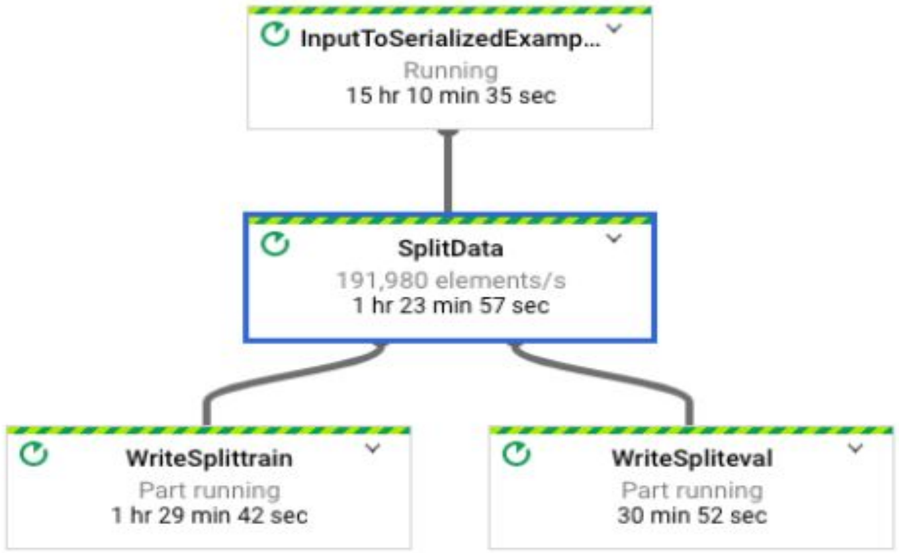
TFX data processing with Apache Beam



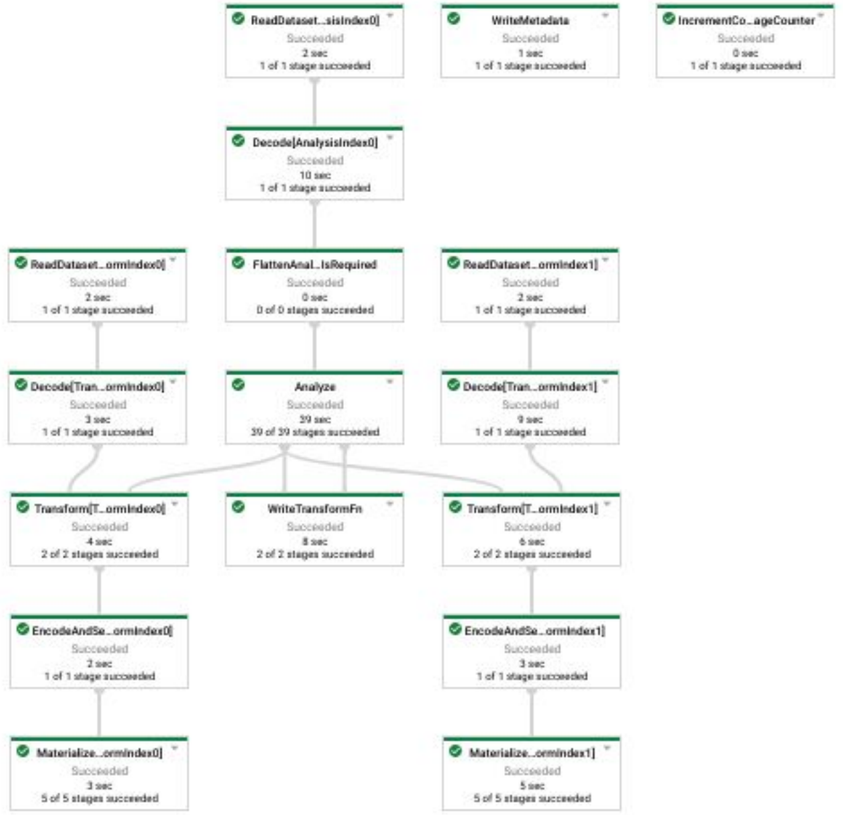

```
raw_data = (
    p
    | 'ReadTrainData' >> textio.ReadFromText(train_data_file)
    | ...
    | 'DecodeTrainData' >> beam.Map(converter.decode))
    | ...
    | tft_beam.AnalyzeAndTransformDataset(preprocessing_fn)
    | ...
    | "WriteTrainData" >> tfrecordio.WriteToTFRecord(
```

TFX data processing with Apache Beam

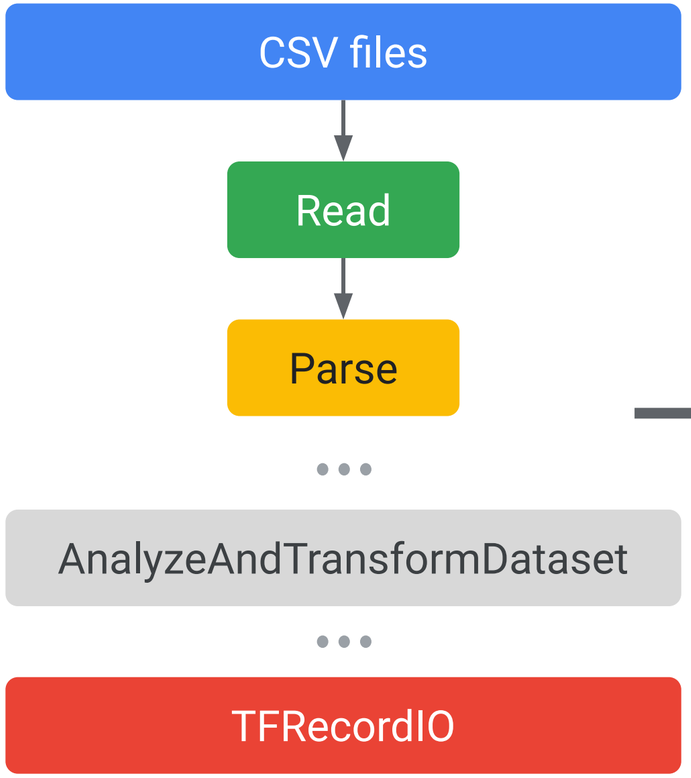
ExampleGen as a Beam pipeline

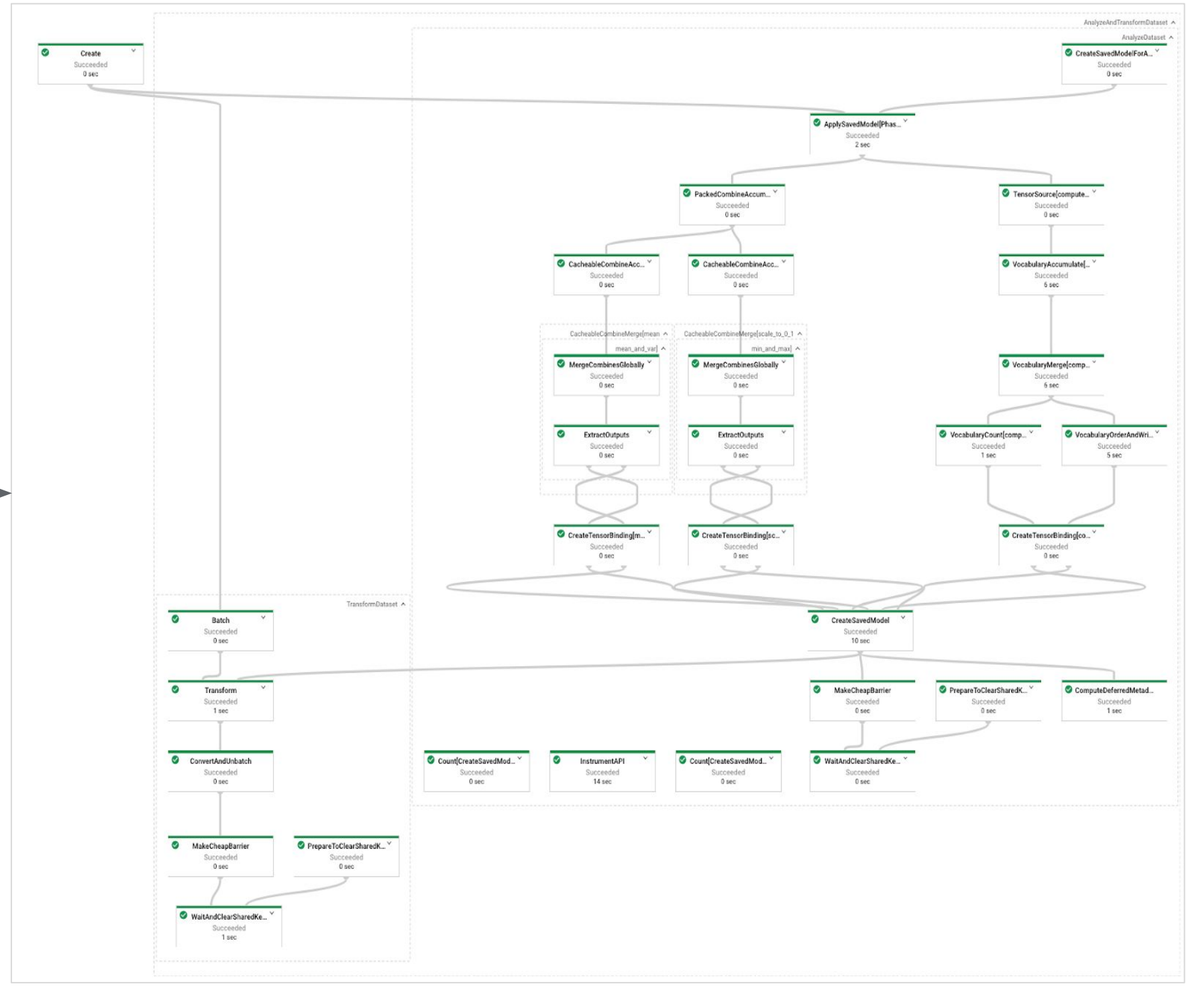


Transform as a Beam pipeline



TFX data processing with Apache Beam





TFX data processing with Apache Beam

tfx_bsl: Beam as a data processor

```
with beam.Pipeline() as p:
     _ = (p
          |
beam.io.ReadFromText('gs://my-bucket/samples.json')
          | beam.Map(convert_json_to_tf_example)
          | RunInference(
              model_spec_pb2.InferenceEndpoint(
                  model_endpoint_spec=

model_spec_pb2.AIPlatformPredictionModelSpec(
                        ...,
                        model_name='my-model-name',

version_name='my-model-version'))))
```

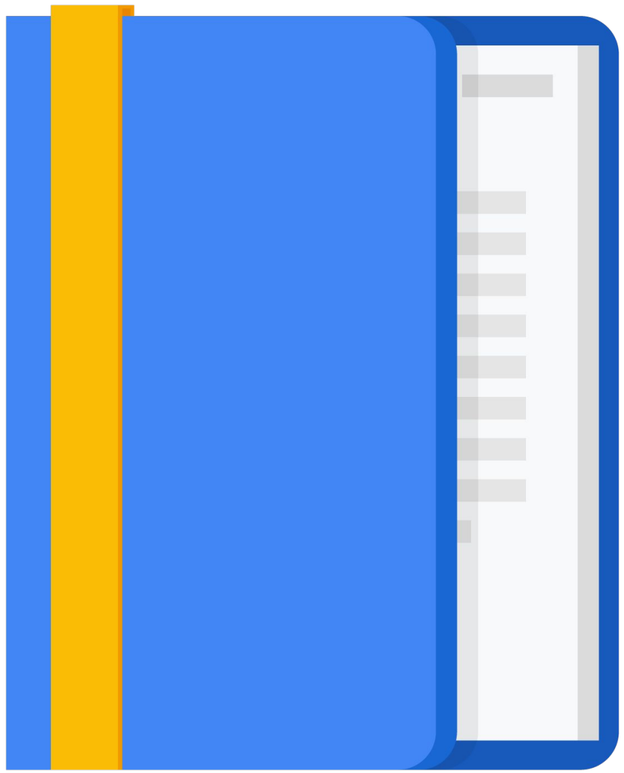
Beam as a data processor and orchestrator

```
...

def run():
  """Define a beam pipeline."""

  BeamDagRunner().run(
      pipeline.create_pipeline(
          pipeline_name=configs.PIPELINE_NAME,
          pipeline_root=PIPELINE_ROOT,
          data_path=DATA_PATH,
          query=configs.BIG_QUERY_QUERY,
          preprocessing_fn=configs.PREPROCESSING_FN,
          run_fn=configs.RUN_FN,
train_args=trainer_pb2.TrainArgs(num_steps=configs.TRAIN_NUM_STEPS),

eval_args=trainer_pb2.EvalArgs(num_steps=configs.EVAL_NUM_STEPS),
          eval_accuracy_threshold=configs.EVAL_ACCURACY_THRESHOLD,
          serving_model_dir=SERVING_MODEL_DIR,
          # BIG_QUERY_WITH_DIRECT_RUNNER_BEAM_PIPELINE_ARGS,

metadata_connection_config=metadata.sqlite_metadata_connection_config(
              METADATA_PATH)))

if __name__ == '__main__':
  logging.set_verbosity(logging.INFO)
  run()
```

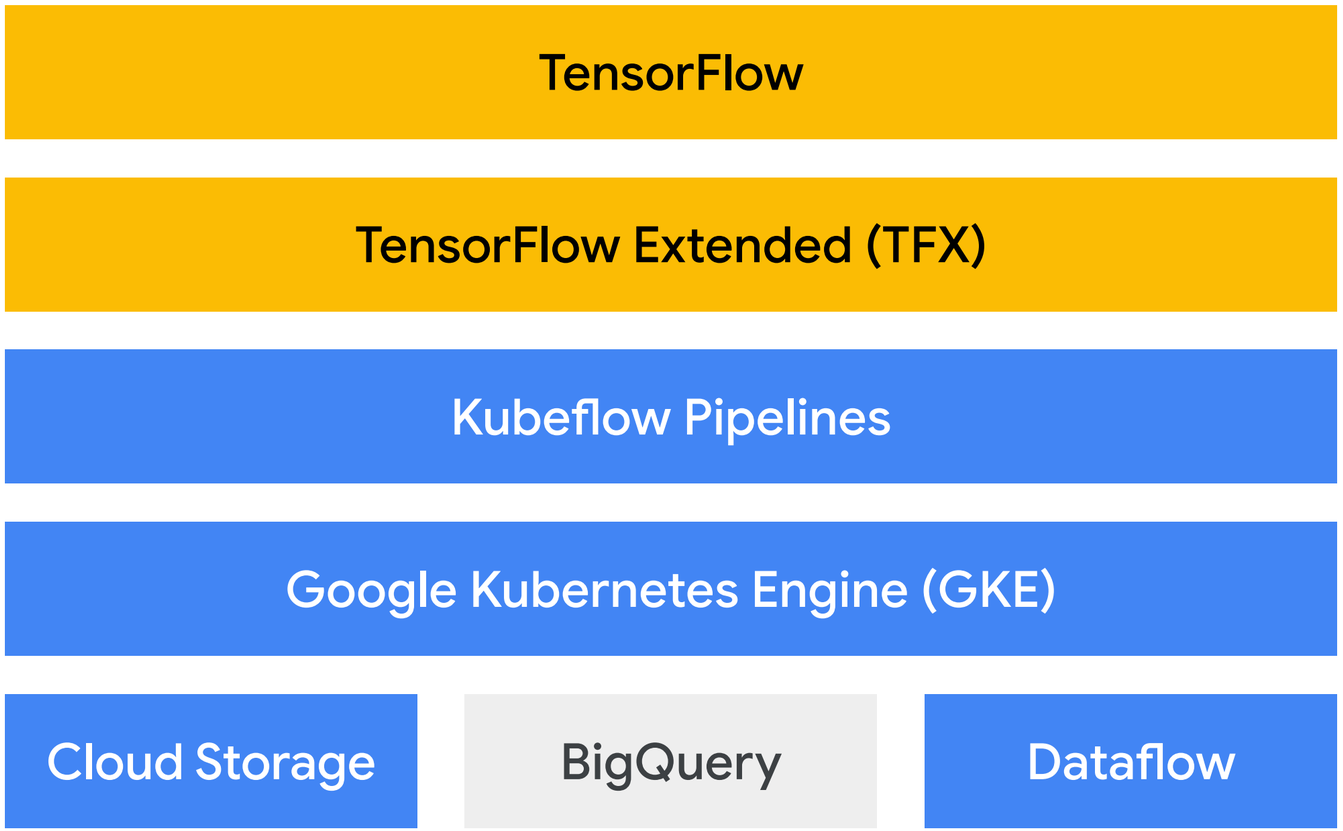
Agenda

TFX orchestrators

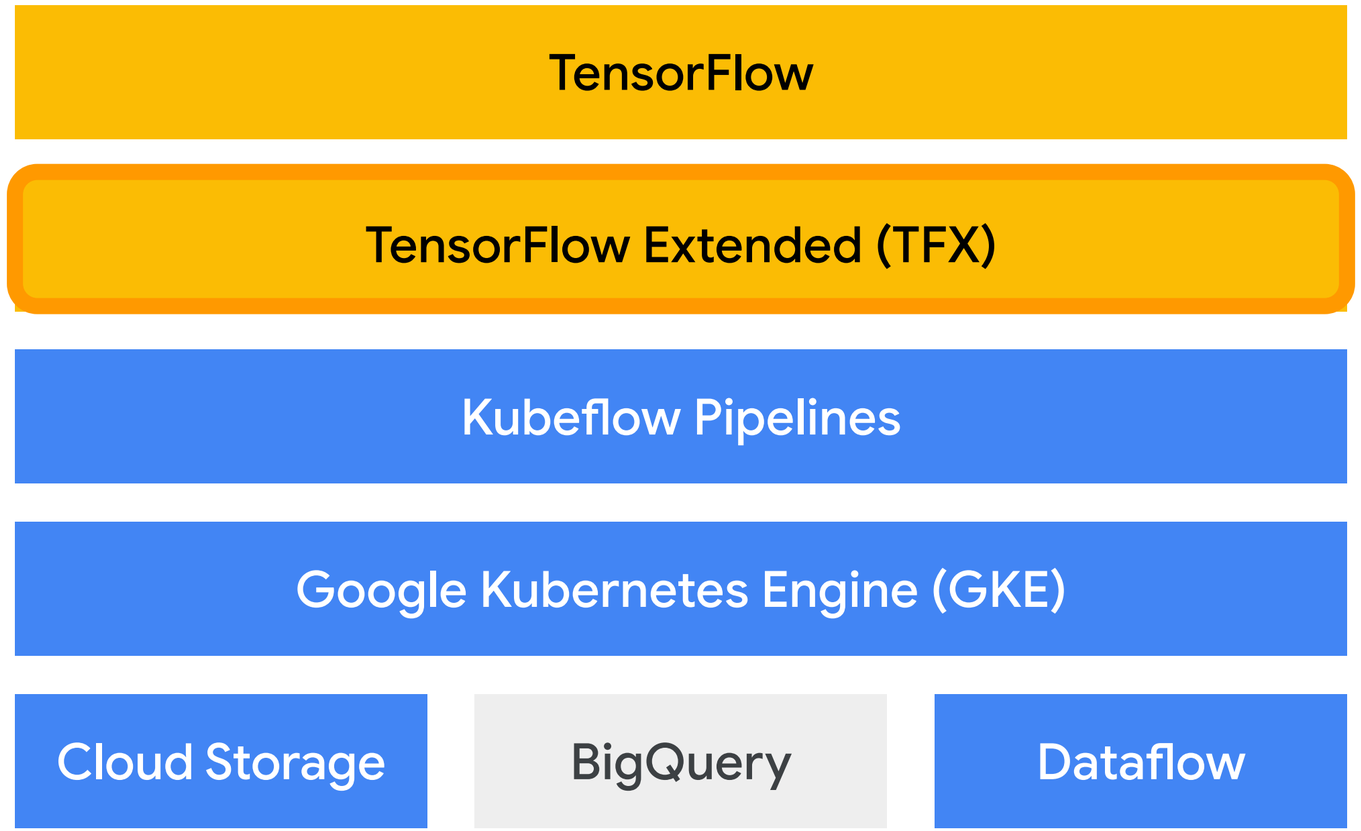
TFX on Cloud AI Platform



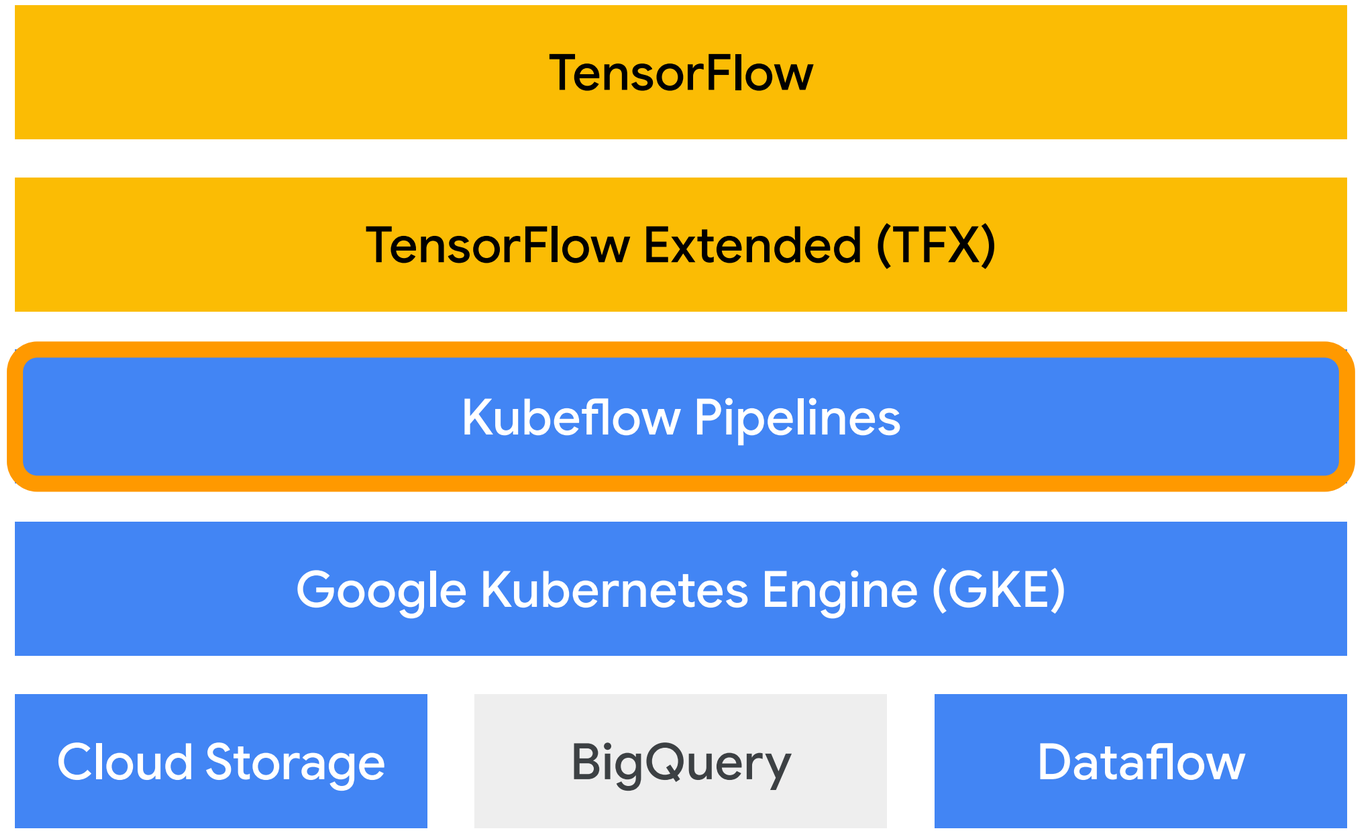
High-level architecture of TFX on Google Cloud



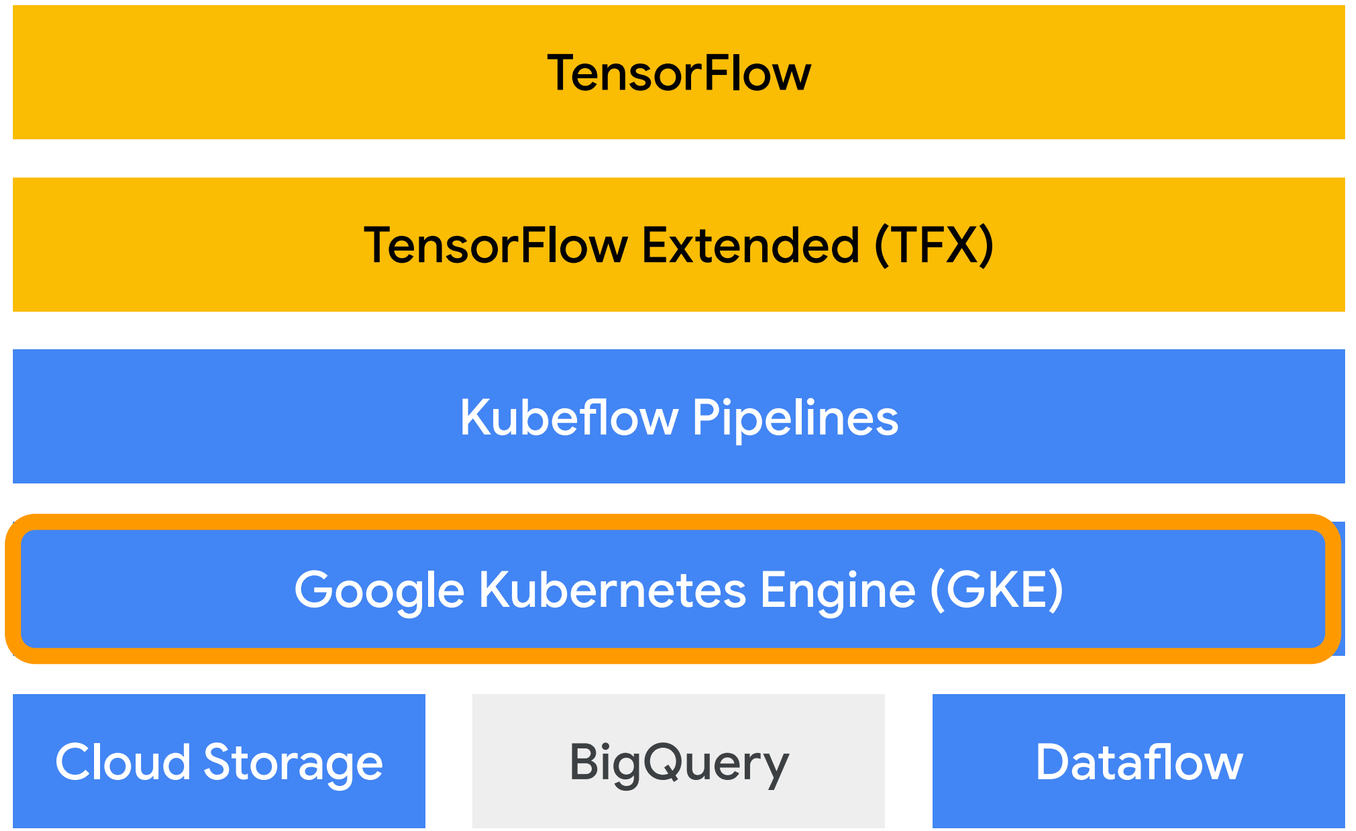
High-level architecture of TFX on Google Cloud



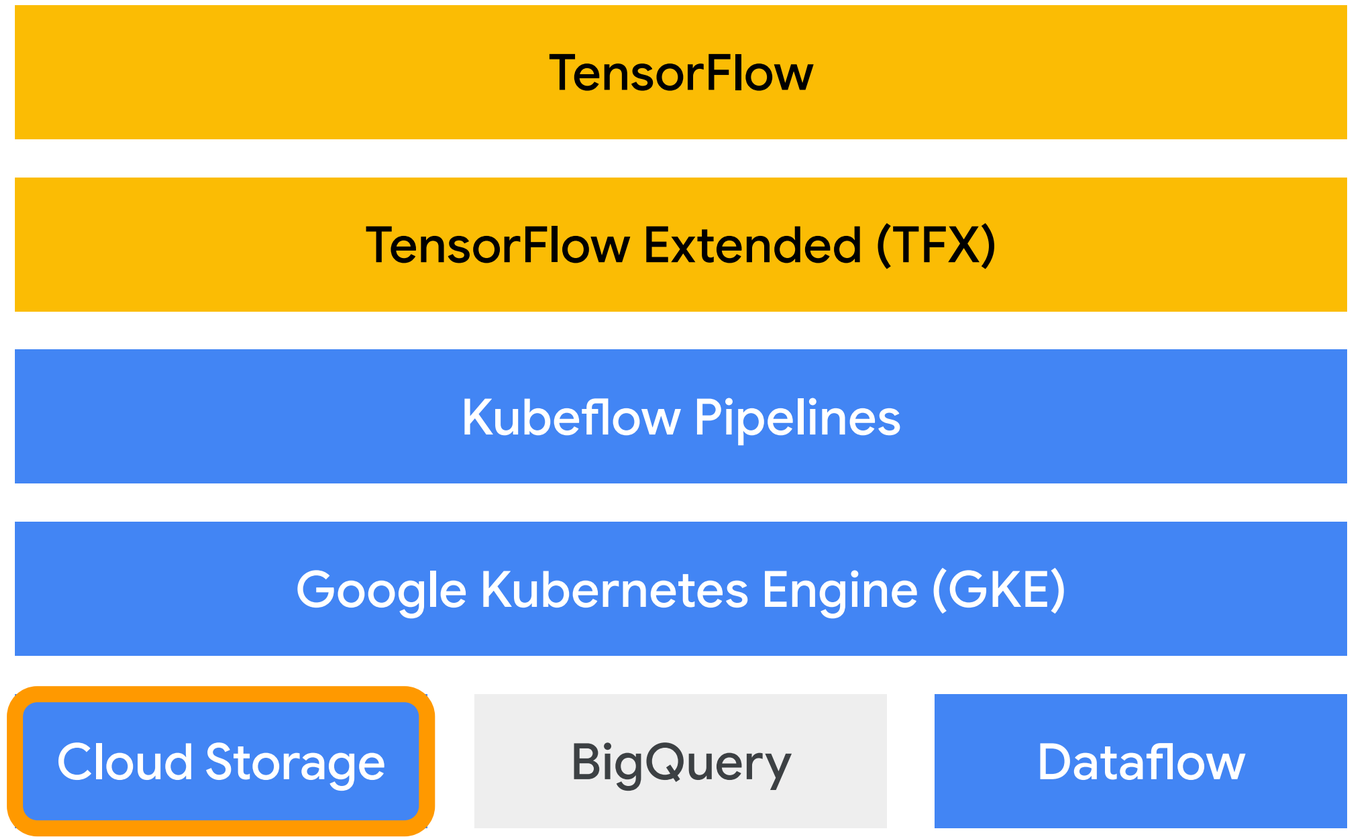
High-level architecture of TFX on Google Cloud



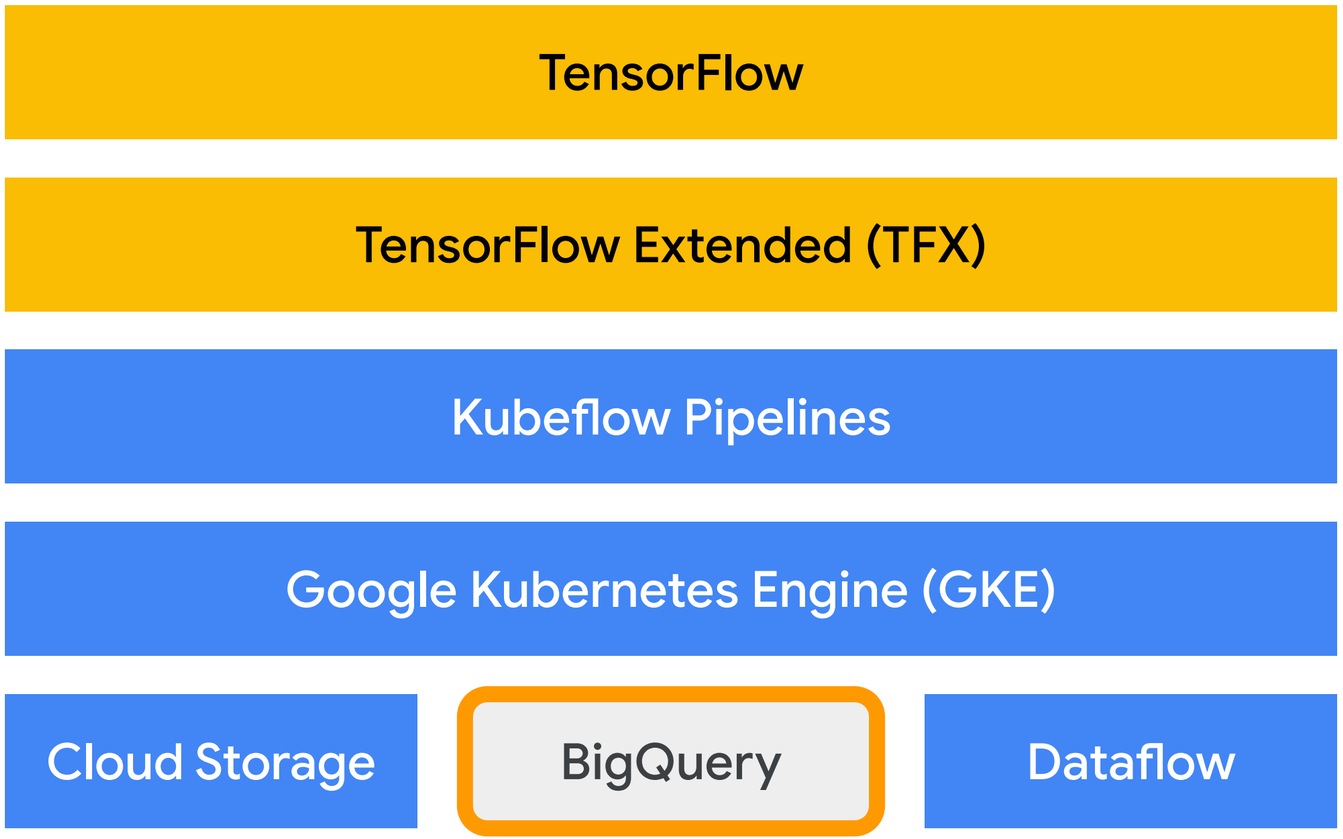
High-level architecture of TFX on Google Cloud



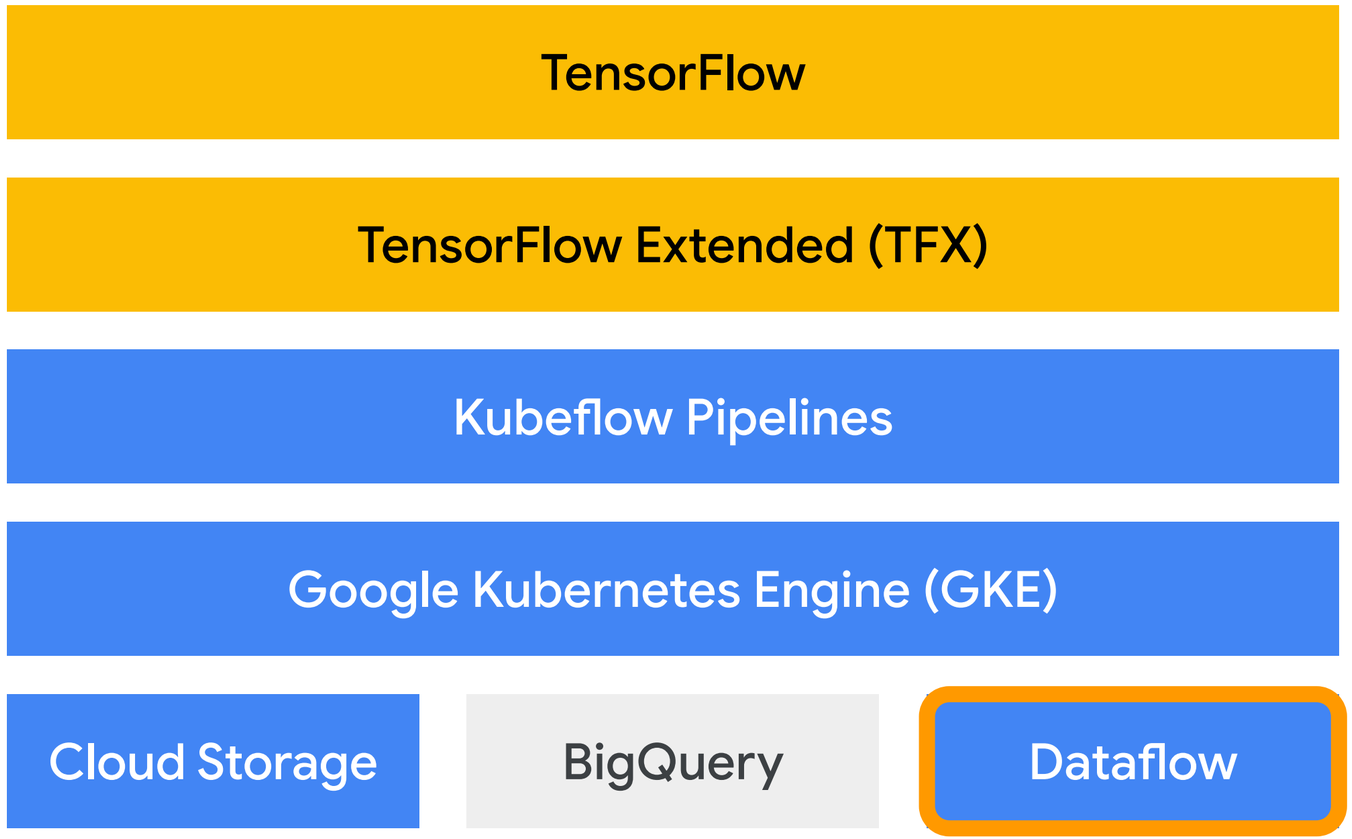
High-level architecture of TFX on Google Cloud



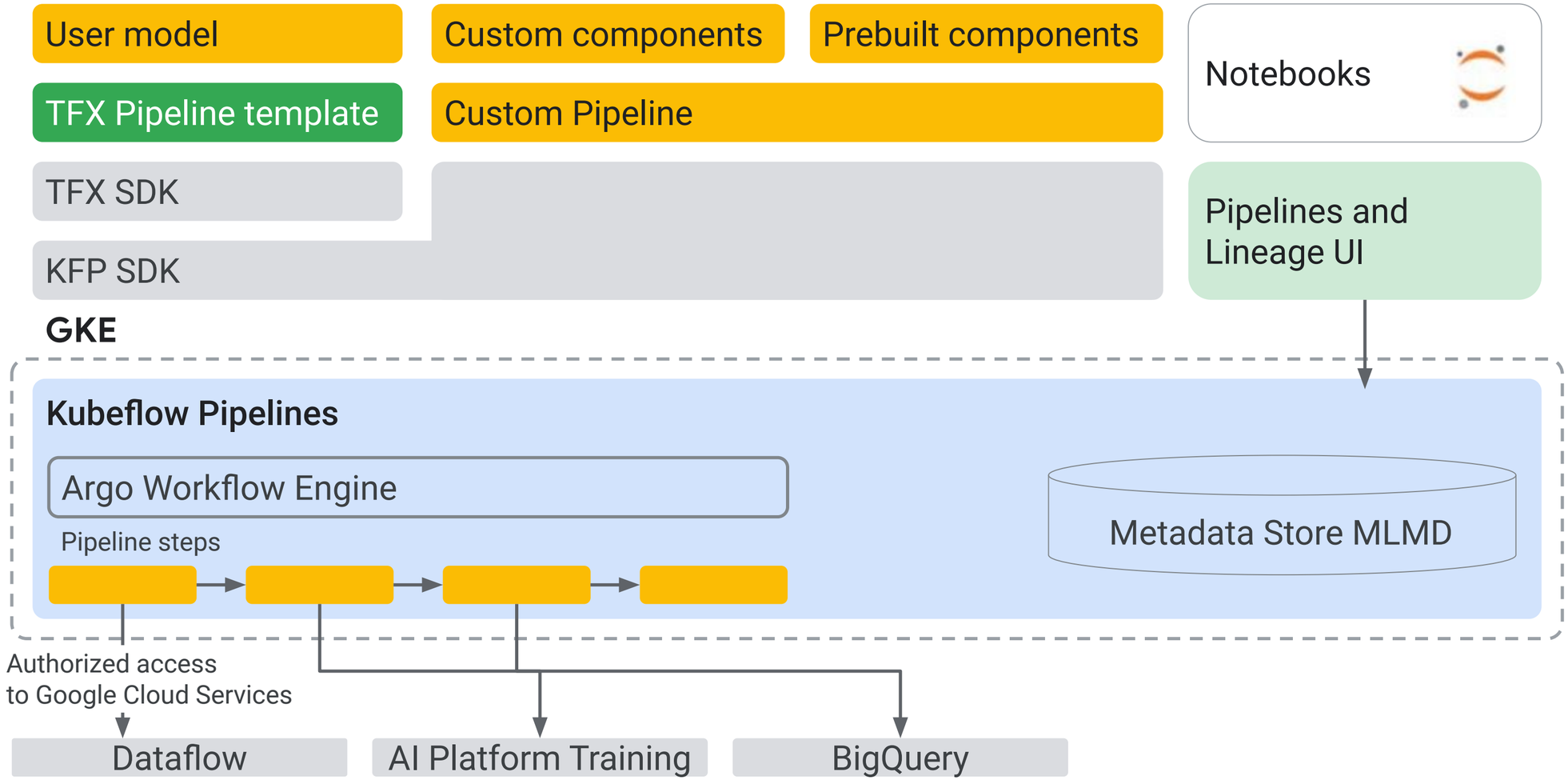
High-level architecture of TFX on Google Cloud



High-level architecture of TFX on Google Cloud



Details view: TFX pipelines run on Google Cloud



TFX DSL to KFP DSL to run on Google Cloud–hosted KFP Pipelines

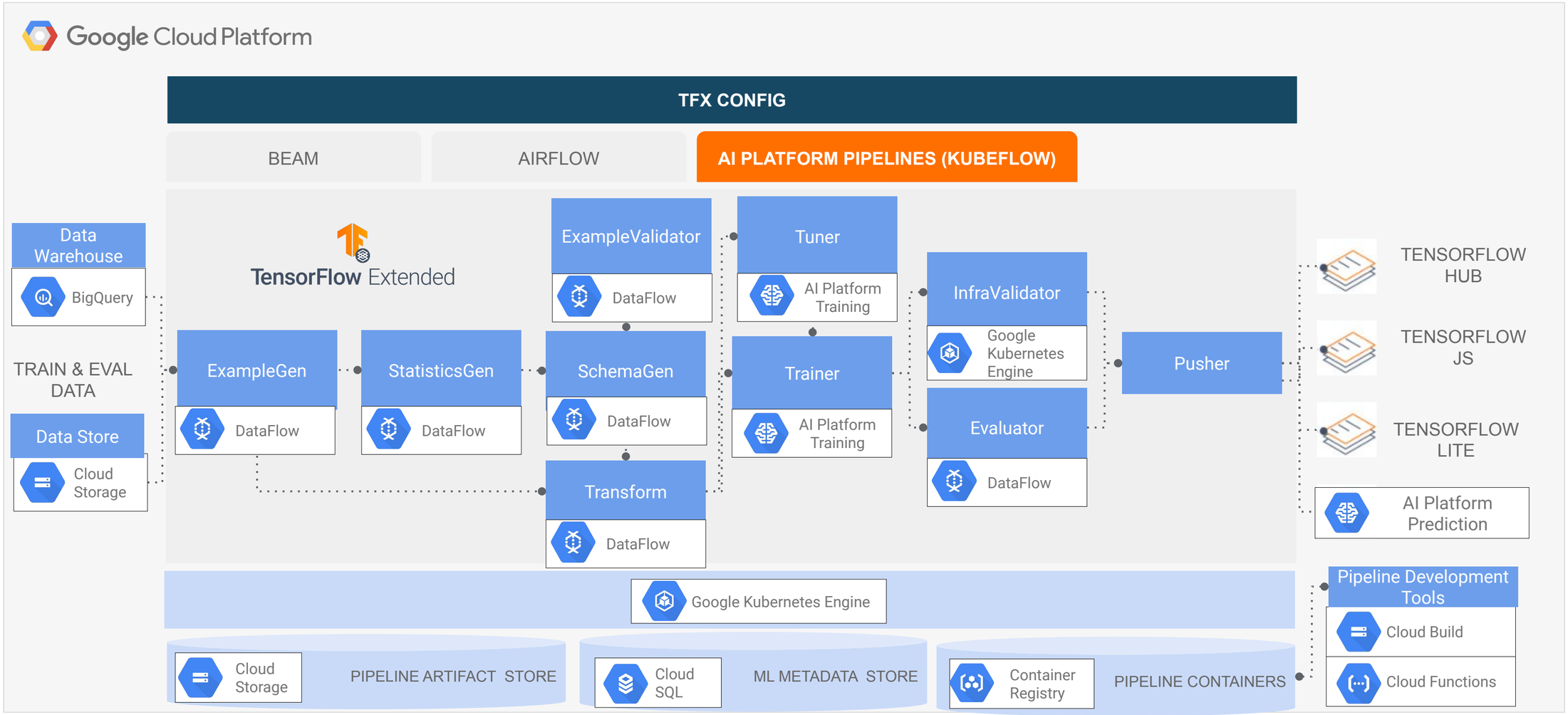
Define a pipeline with TFX

```
def create_pipeline(
    pipeline_name: Text,
    pipeline_root: Text,
    data_root: Text,
    module_file: Text,
    enable_cache: Boolean,
    beam_pipeline_args: Dict[Text],
    serving_model_dri: List[Text]): -> pipeline.Pipeline:

      example_gen = CsvExampleGen(input=external_input(data_root)
      statistics_gen = StatisticsGen(examples=example_gen.outputs['examples'])
      ...
```

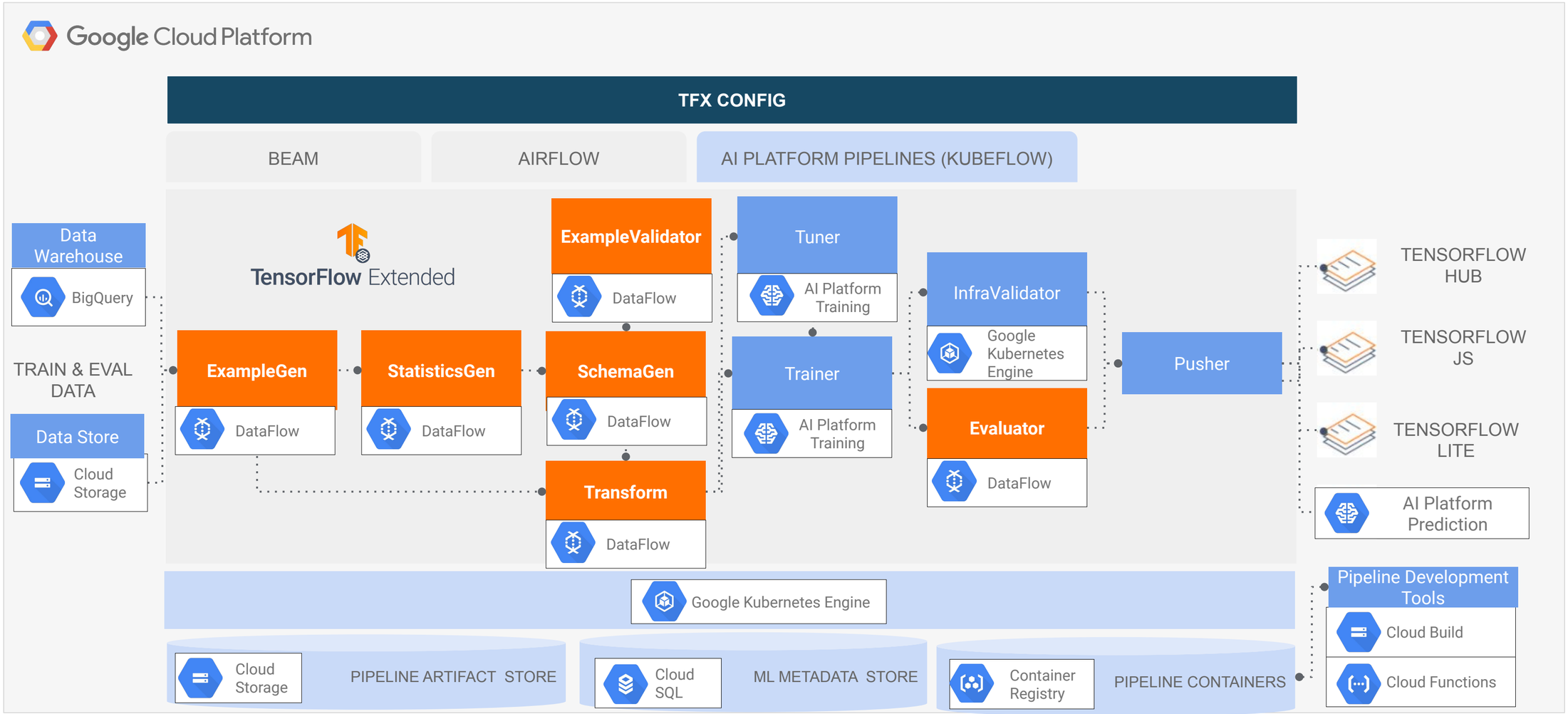
Compile the pipeline to KFP YAML

```
runner_config = kubeflow_dag_runner.KubeflowDagRunner(
    kubeflow_metadata_config=metadata_config,
    tfx_image = 'tensorflow/tfx:0.24'

kubeflow_dag_runner.KubeflowDagRunner(config=runner_config).run(
    _create_pipeline(...))
```

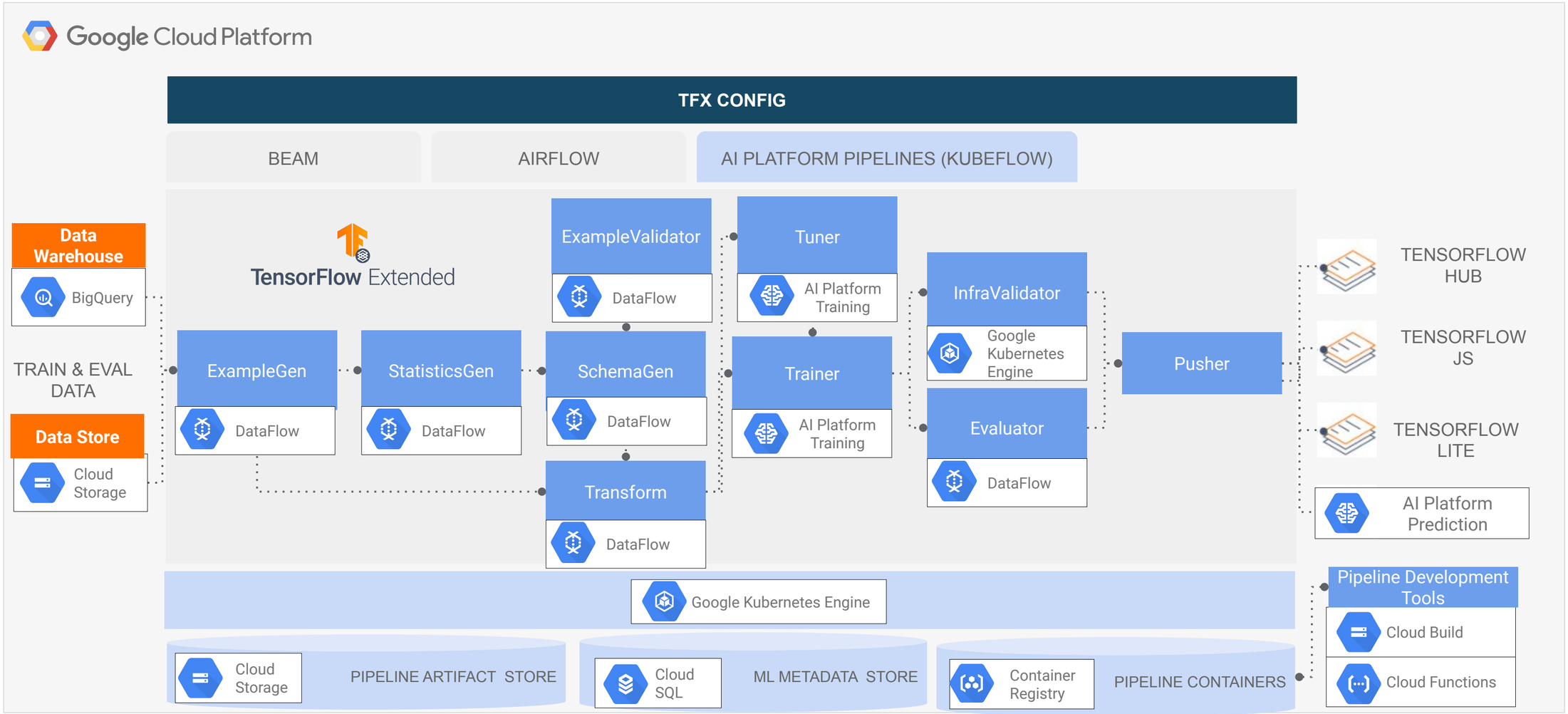
TFX on Google Cloud



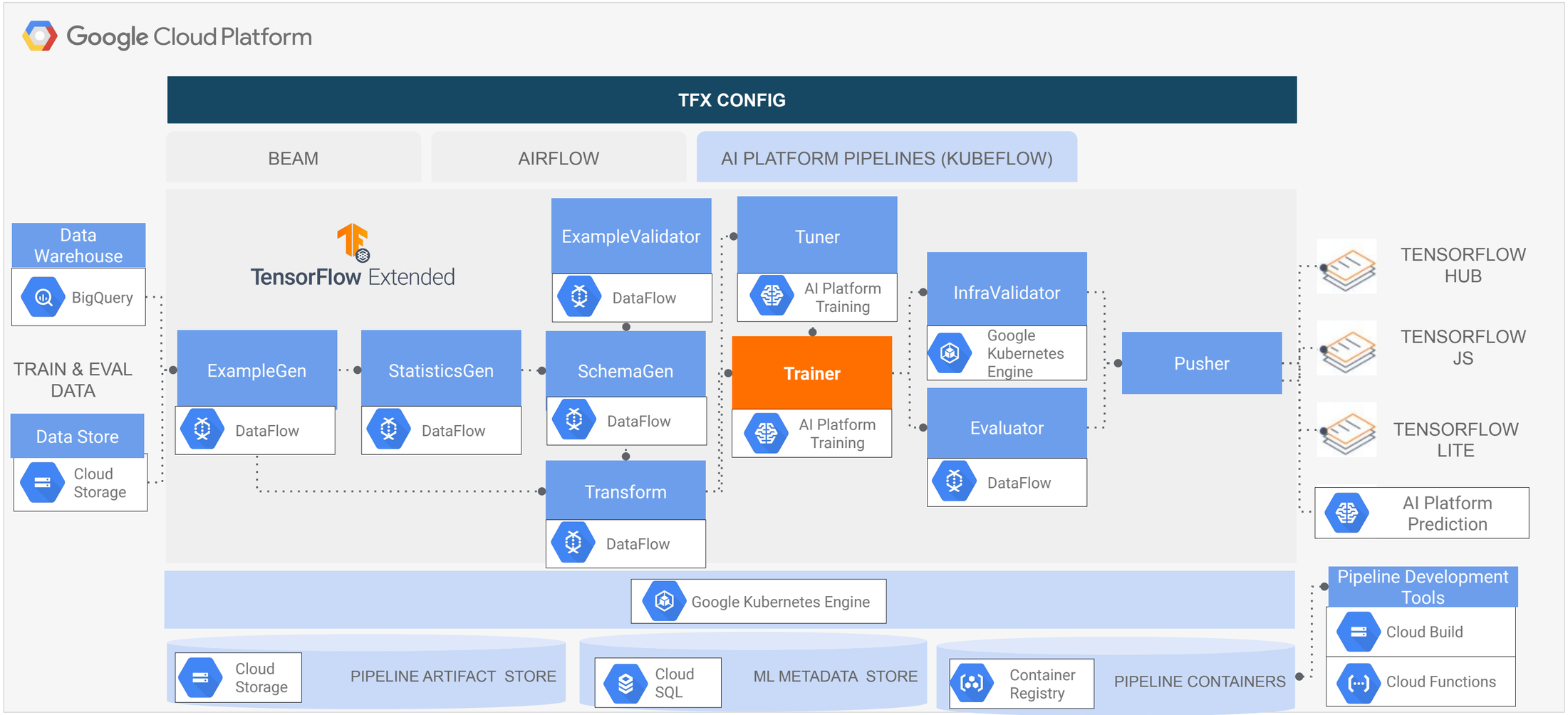
TFX on Google Cloud



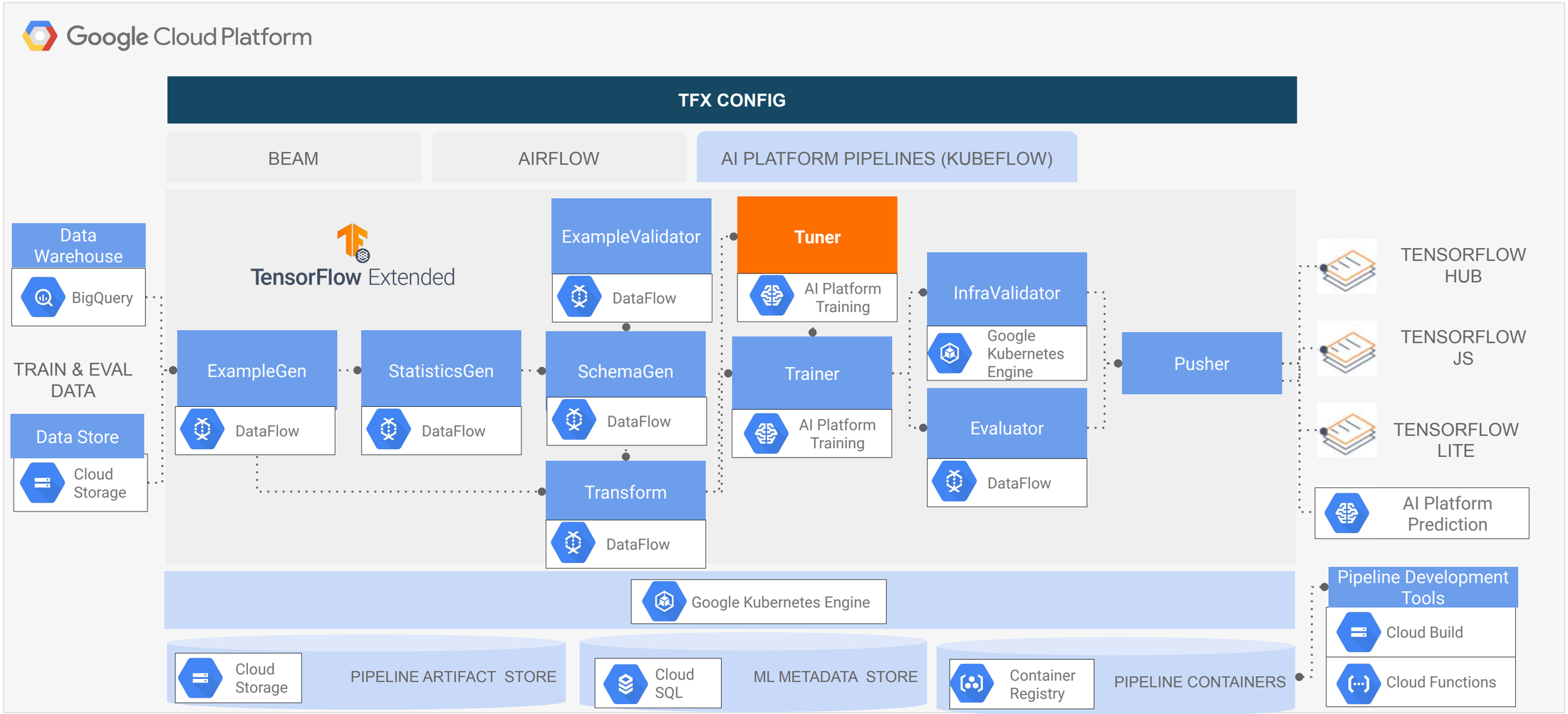
TFX on Google Cloud



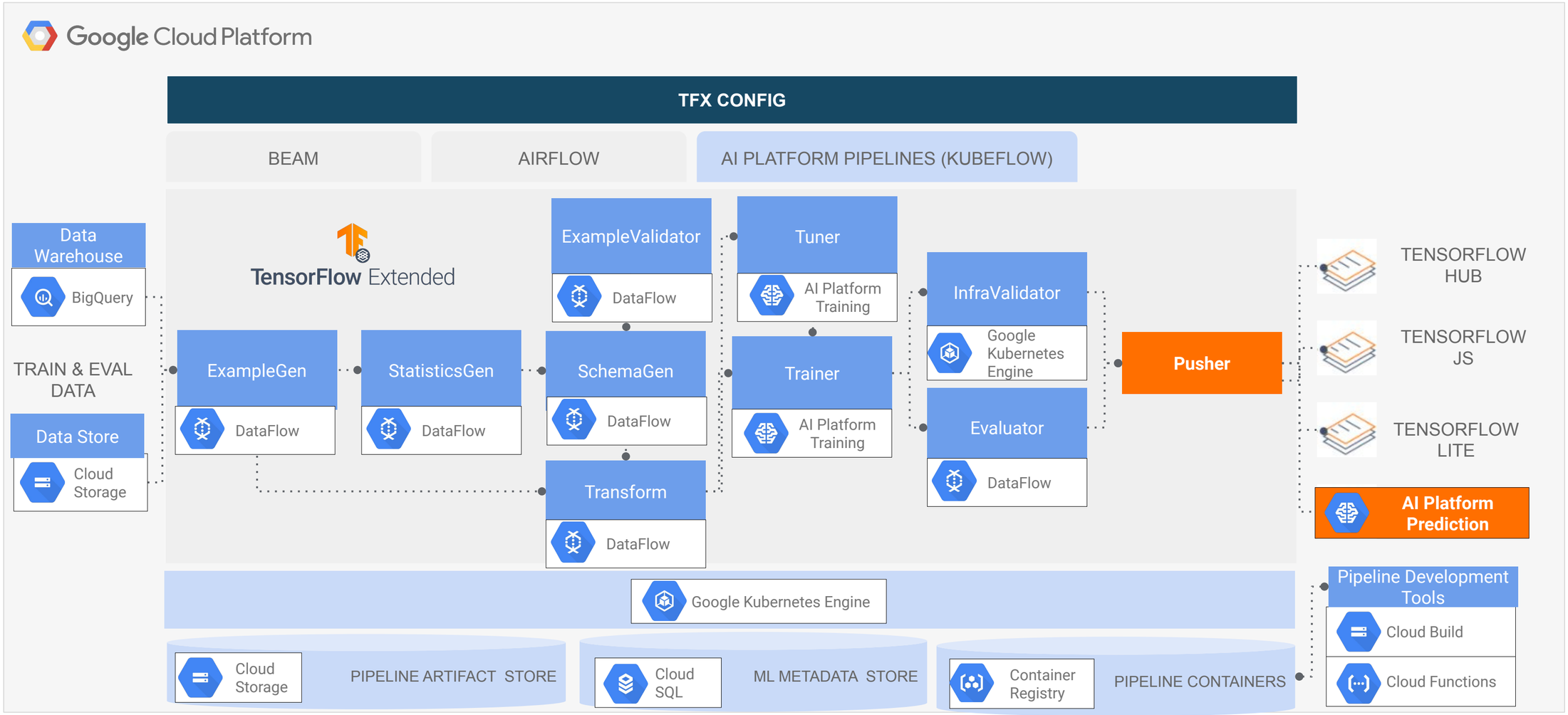
TFX on Google Cloud



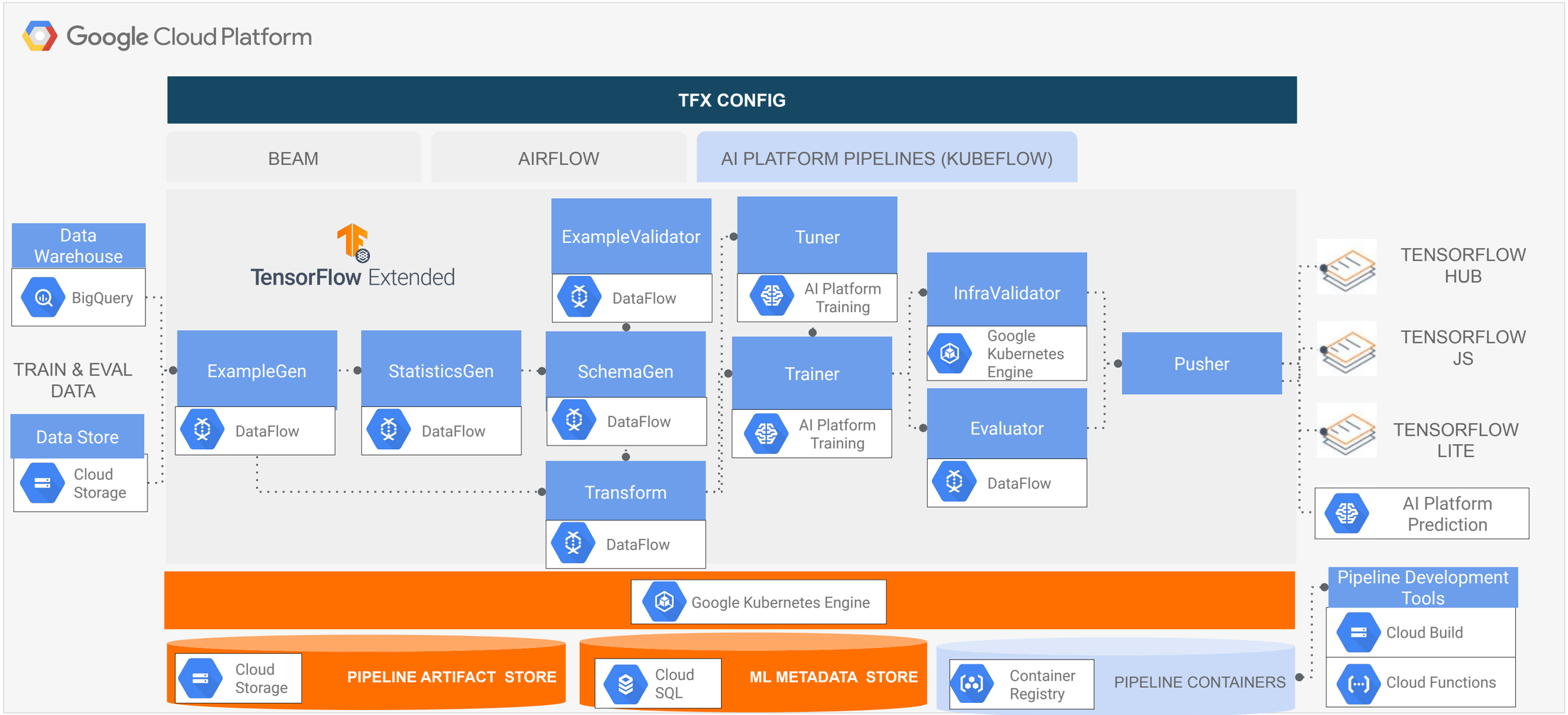
TFX on Google Cloud



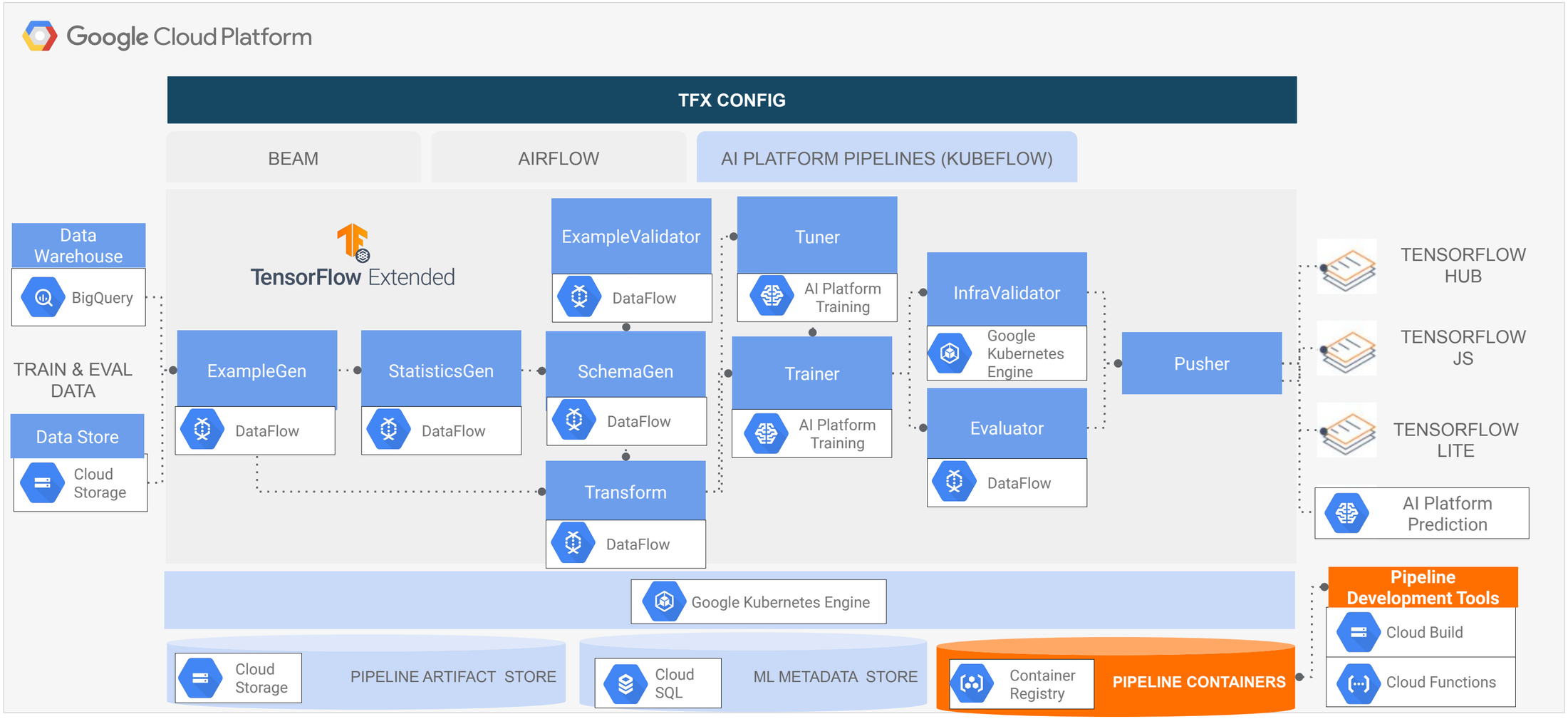
TFX on Google Cloud



TFX on Google Cloud



TFX on Google Cloud



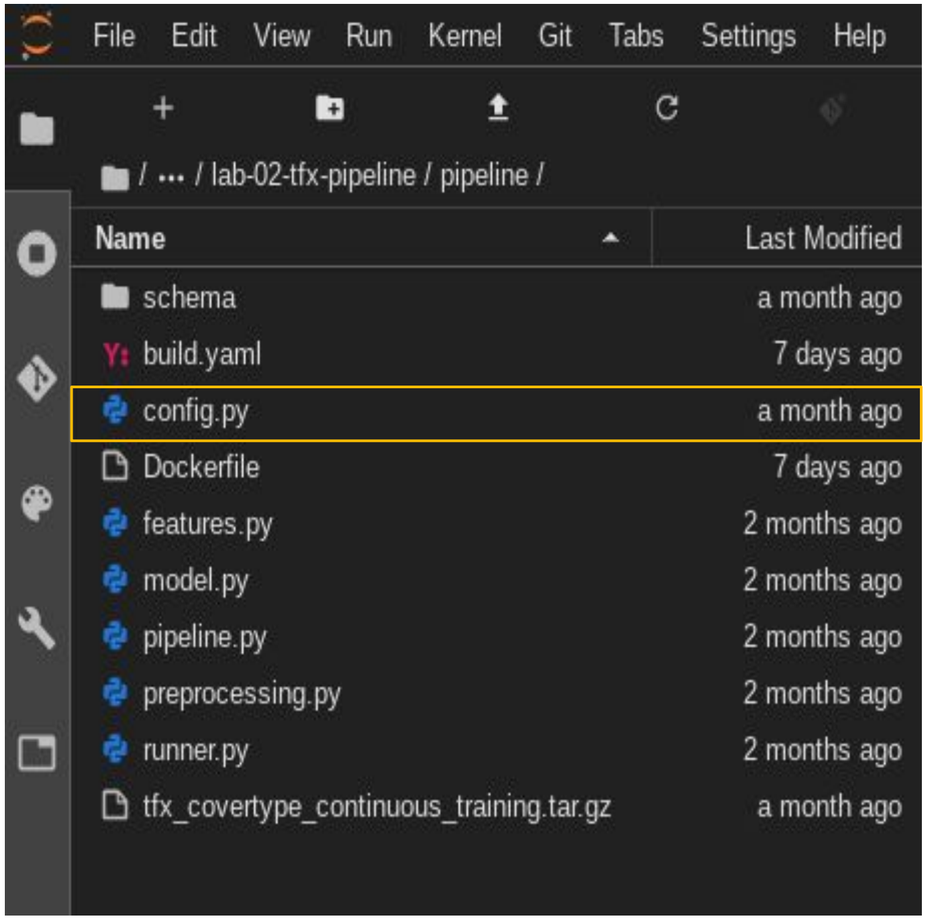
Lab

TFX pipelines on Cloud AI Platform

TFX pipeline design pattern

Modules

config.py: Configures the default values for the environment-specific settings and the default values for the pipeline runtime parameters.

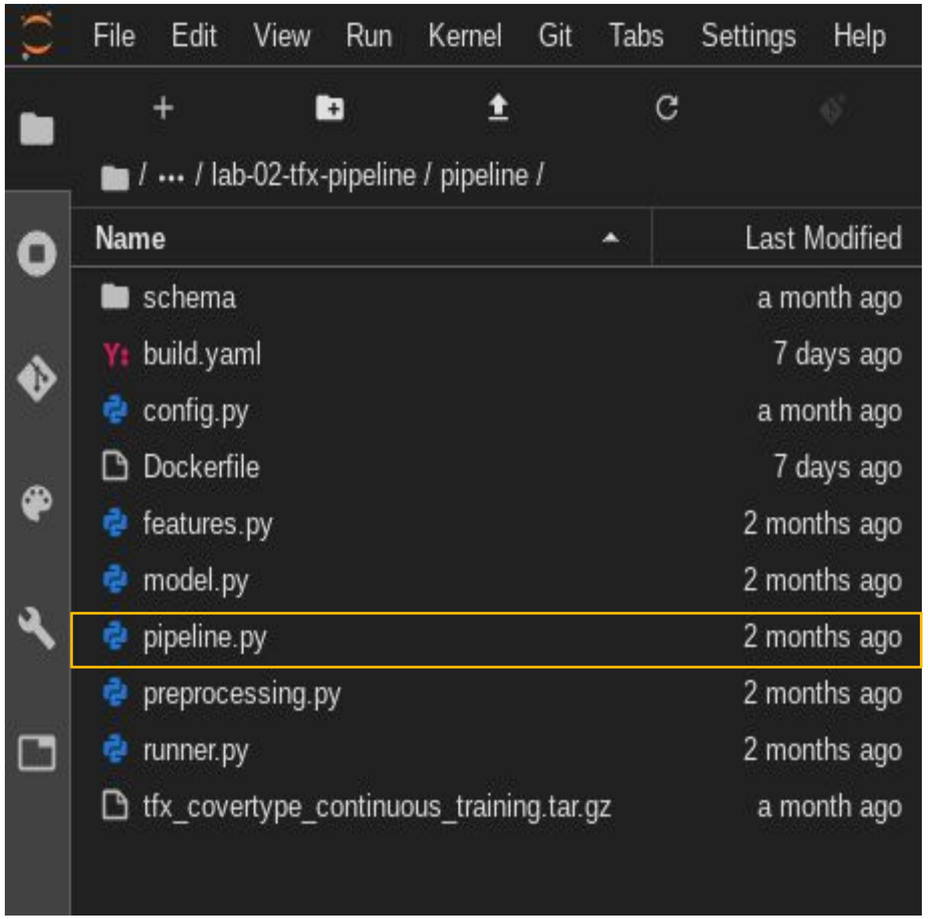


TFX pipeline design pattern

Modules

config.py: Configures the default values for the environment-specific settings and the default values for the pipeline runtime parameters.

Pipeline.py: Contains the TFX DSL that defines the workflow implemented by the pipeline.



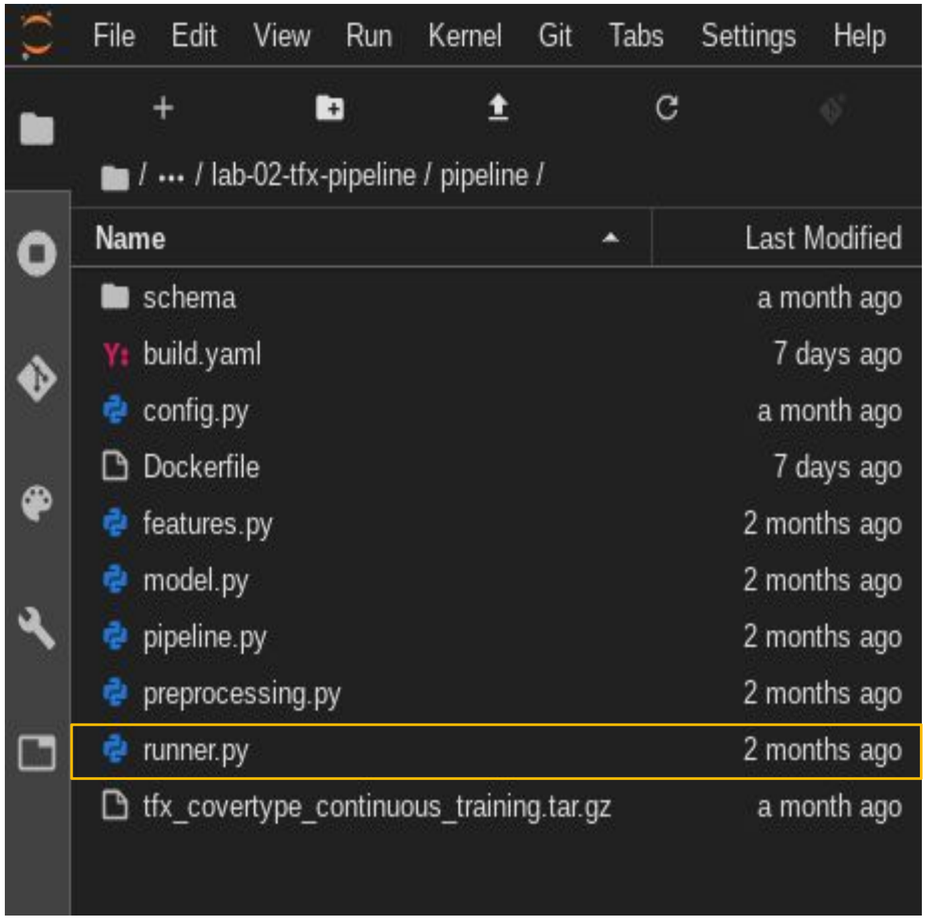
TFX pipeline design pattern

Modules

config.py: Configures the default values for the environment-specific settings and the default values for the pipeline runtime parameters.

pipeline.py: Contains the TFX DSL that defines the workflow implemented by the pipeline.

runner.py: Configures and executes KubeflowDagRunner. At compile time, the KubeflowDagRunner.run() method converts the TFX DSL into the pipeline package in the Kubeflow Argo format.



TFX pipeline design pattern

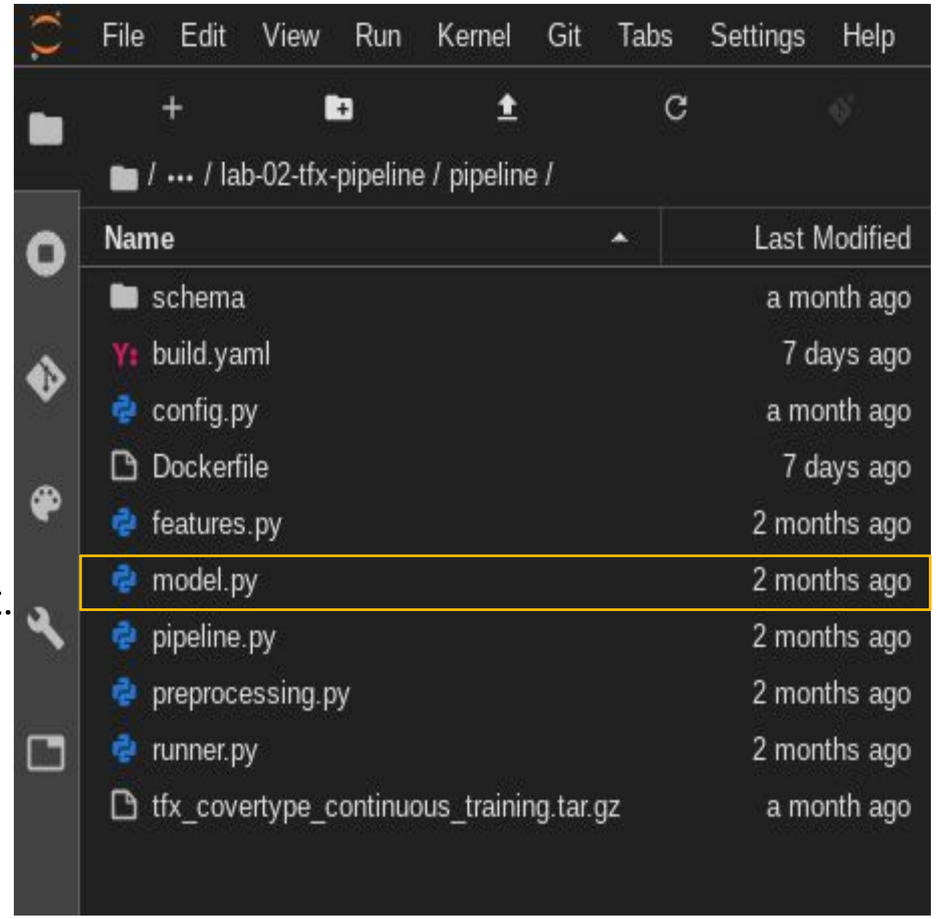
Modules

config.py: Configures the default values for the environment-specific settings and the default values for the pipeline runtime parameters.

pipeline.py: Contains the TFX DSL that defines the workflow implemented by the pipeline.

runner.py: Configures and executes KubeflowDagRunner. At compile time, the KubeflowDagRunner.run() method converts the TFX DSL into the pipeline package in the Kubeflow Argo format.

model.py: Implements the training logic for the Trainer component.



TFX pipeline design pattern

Modules

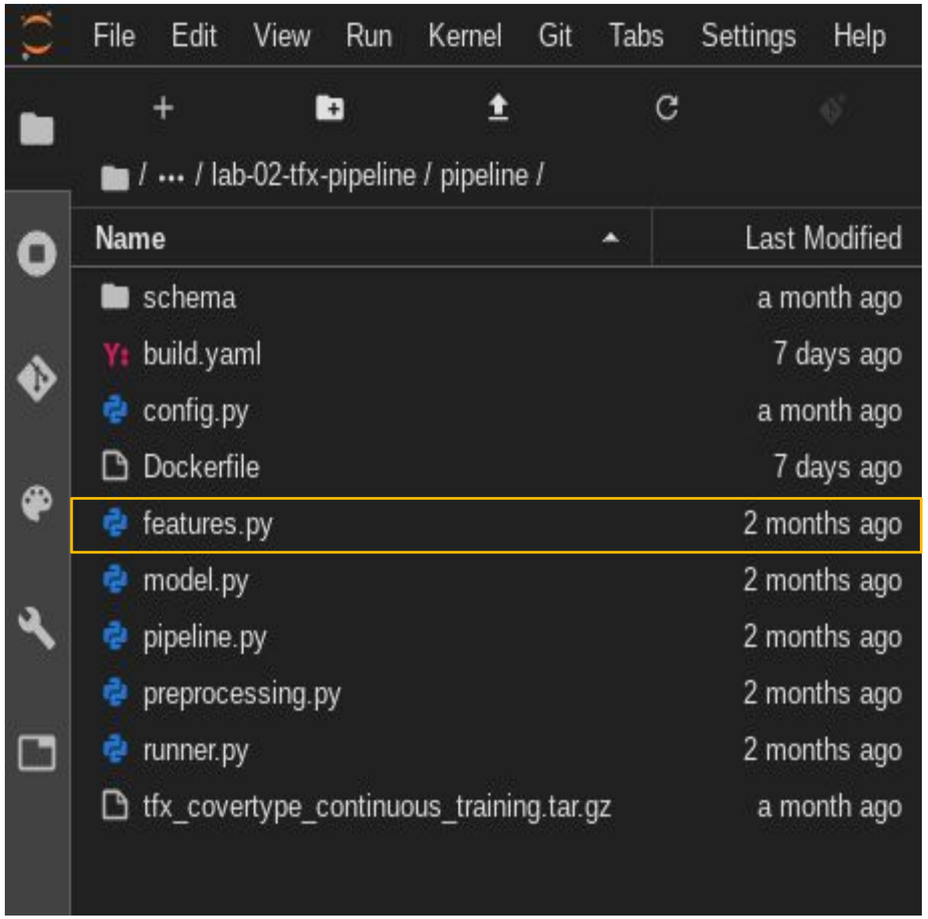
config.py: Configures the default values for the environment-specific settings and the default values for the pipeline runtime parameters.

pipeline.py: Contains the TFX DSL that defines the workflow implemented by the pipeline.

runner.py: Configures and executes KubeflowDagRunner. At compile time, the KubeflowDagRunner.run() method converts the TFX DSL into the pipeline package in the Kubeflow Argo format.

model.py: Implements the training logic for the Train component.

features.py: Contains feature definitions common across preprocessing.py and model.py.



TFX pipeline design pattern

Modules

config.py: Configures the default values for the environment-specific settings and the default values for the pipeline runtime parameters.

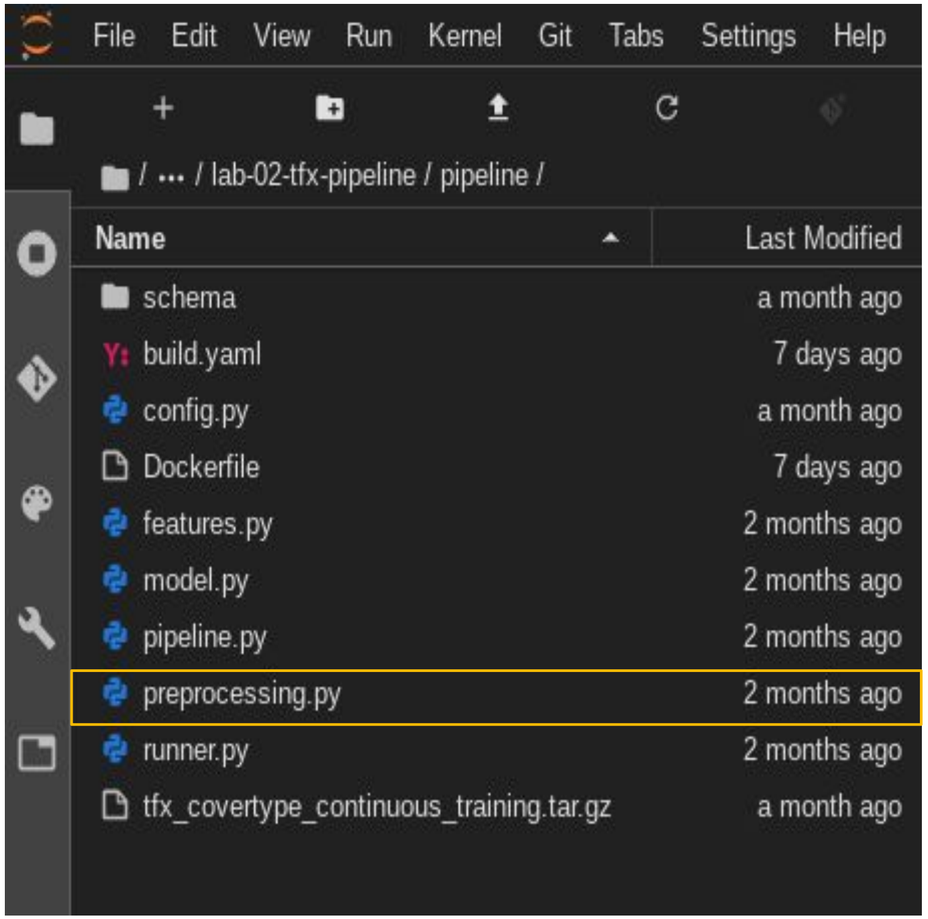
pipeline.py: Contains the TFX DSL that defines the workflow implemented by the pipeline.

runner.py: Configures and executes KubeflowDagRunner. At compile time, the KubeflowDagRunner.run() method converts the TFX DSL into the pipeline package in the Kubeflow Argo format.

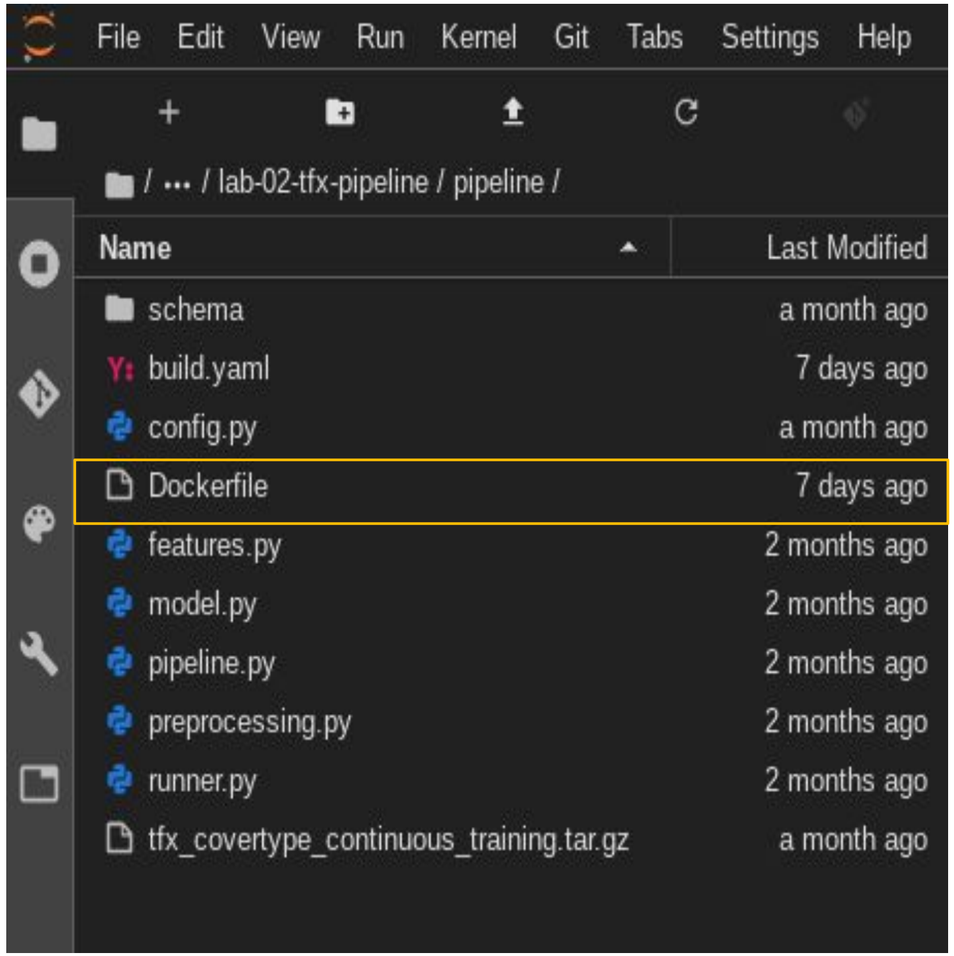
model.py: Implements the training logic for the Train component.

features.py: Contains feature definitions common across preprocessing.py and model.py.

preprocessing.py: Implements the data preprocessing logic for the Transform component.



Package your TFX pipeline as a Docker container



Dockerfile

```
FROM tensorflow/tfx:0.25.0
WORKDIR ./pipeline
COPY ./ ./
ENV PYTHONPATH="/pipeline:${PYTHONPATH}"
```

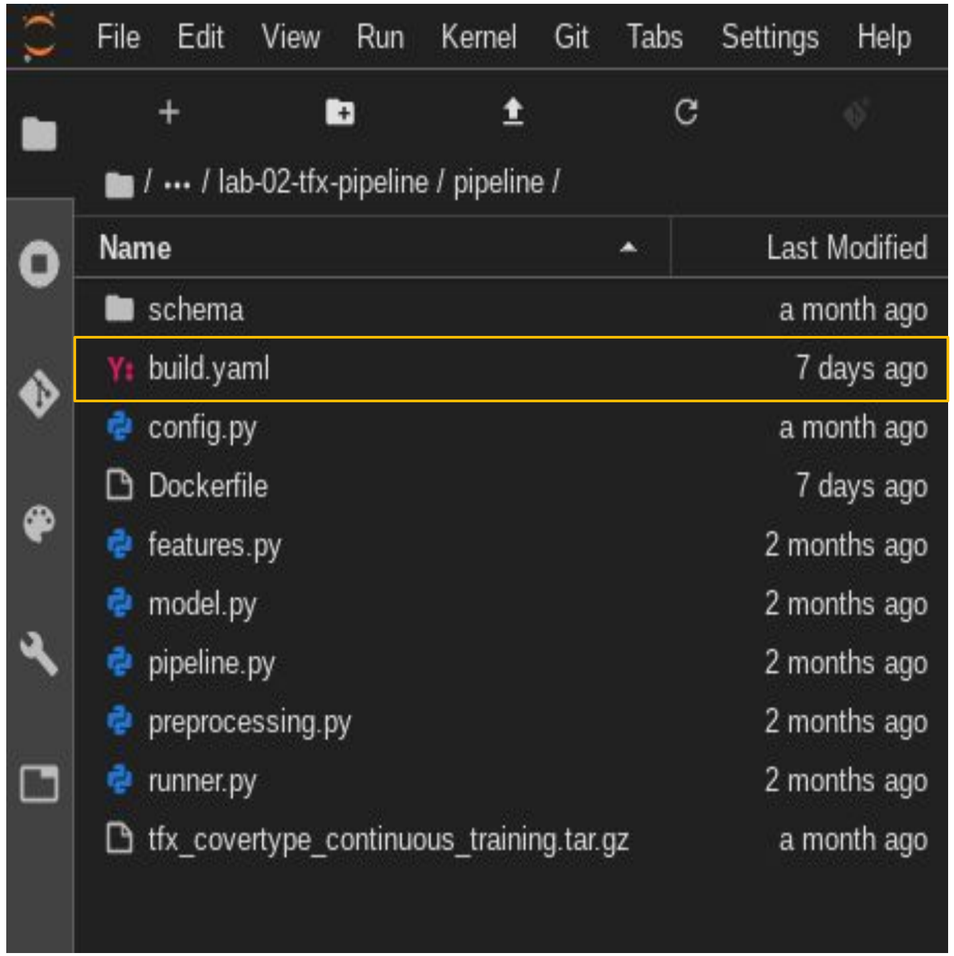
Package your pipeline in a Dockerfile to build with skaffold

build.yaml

```
apiVersion: skaffold/v2beta4
build:
  artifacts:
  - context: .
    docker:
      dockerfile: Dockerfile
    image:
gcr.io/asl-ml-immersion/tfx_covertype_continuous
_training
  tagPolicy:
    envTemplate:
      template: '{{.IMAGE_NAME}}:latest'
kind: Config
```

Publish pipeline to Container Registry

Package your TFX pipeline as a Docker container



Dockerfile

```
FROM tensorflow/tfx:0.25.0
WORKDIR ./pipeline
COPY ./ ./
ENV PYTHONPATH="/pipeline:${PYTHONPATH}"
```

Package your pipeline in a Dockerfile to build with skaffold

build.yaml

```
apiVersion: skaffold/v2beta4
build:
  artifacts:
  - context: .
    docker:
      dockerfile: Dockerfile
    image: 
gcr.io/asl-ml-immersion/tfx_covertype_continuous
_training
  tagPolicy:
    envTemplate:
      template: '{{.IMAGE_NAME}}:latest'
kind: Config
```

Publish pipeline to Container Registry

Compile the TFX pipeline with the TFX CLI

Package the TFX pipeline as a Docker container and publish to Container Registry.

```
PIPELINE_NAME = 'tfx_covertype_continuous_training'
MODEL_NAME = 'tfx_covertype_classifier'

USE_KFP_SA=False
DATA_ROOT_URI = 'gs://workshop-datasets/covertype/small'
CUSTOM_TFX_IMAGE = 'gcr.io/{}/{}'.format(PROJECT_ID, PIPELINE_NAME)
RUNTIME_VERSION = '2.3'
PYTHON_VERSION = '3.7'
```

Define TFX runtime parameters as environment variables.

```
%env PROJECT_ID={PROJECT_ID}
%env KUBEFLOW_TFX_IMAGE={CUSTOM_TFX_IMAGE}
%env ARTIFACT_STORE_URI={ARTIFACT_STORE_URI}
%env DATA_ROOT_URI={DATA_ROOT_URI}
%env GCP_REGION={GCP_REGION}
%env MODEL_NAME={MODEL_NAME}
%env PIPELINE_NAME={PIPELINE_NAME}
%env RUNTIME_VERSION={RUNTIME_VERSION}
%env PYTHON_VERIONS={PYTHON_VERSION}
%env USE_KFP_SA={USE_KFP_SA}
```

Use TFX CLI to compile your pipeline.

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

Compile the TFX pipeline with the TFX CLI

Package the TFX pipeline as a Docker container and publish to Container Registry.

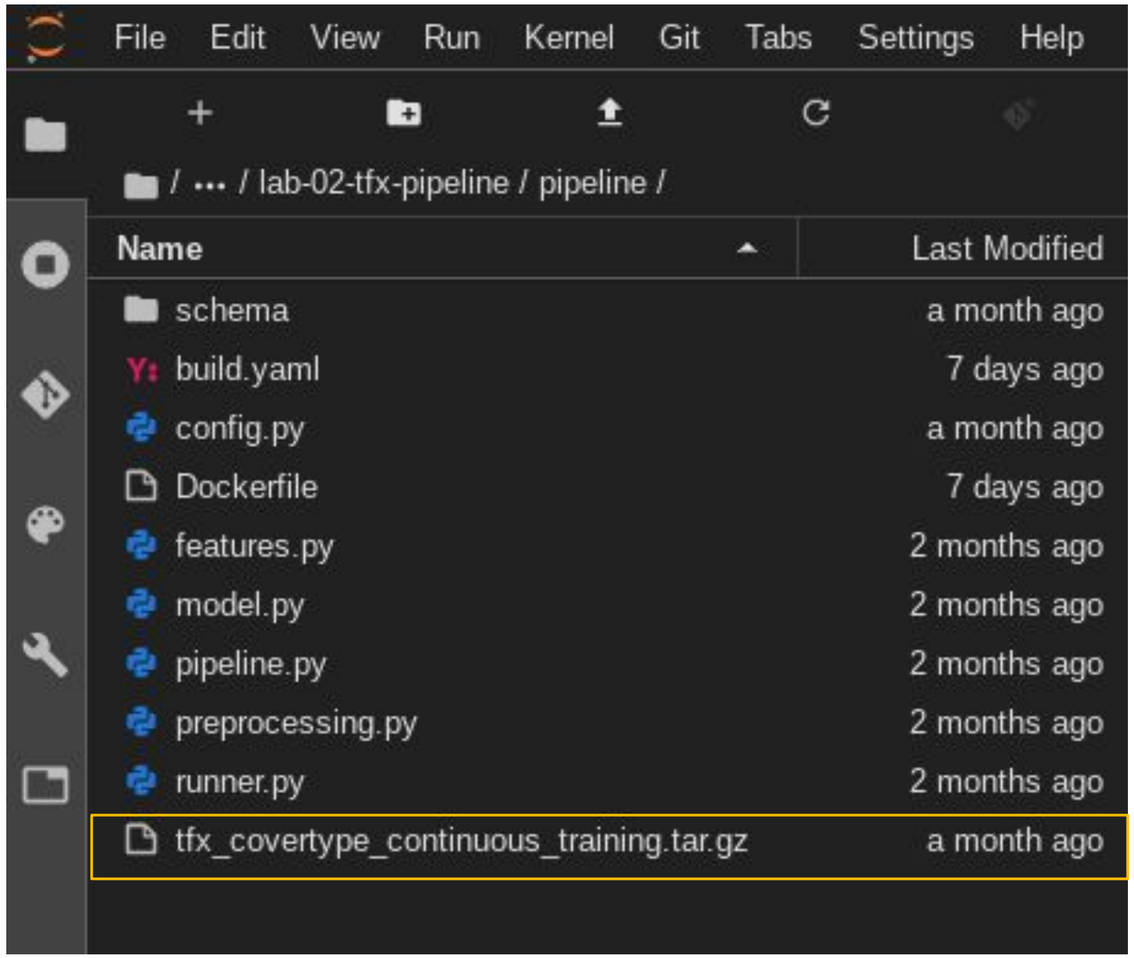
Define TFX runtime parameters as environment variables.

```
PIPELINE_NAME = 'tfx_covertype_continuous_training'
MODEL_NAME = 'tfx_covertype_classifier'

USE_KFP_SA=False
DATA_ROOT_URI = 'gs://workshop-datasets/covertype/small'
CUSTOM_TFX_IMAGE = 'gcr.io/{}/{}'.format(PROJECT_ID, PIPELINE_NAME)
RUNTIME_VERSION = '2.3'
PYTHON_VERSION = '3.7'


%env PROJECT_ID={PROJECT_ID}
%env KUBEFLOW_TFX_IMAGE={CUSTOM_TFX_IMAGE}
%env ARTIFACT_STORE_URI={ARTIFACT_STORE_URI}
%env DATA_ROOT_URI={DATA_ROOT_URI}
%env GCP_REGION={GCP_REGION}
%env MODEL_NAME={MODEL_NAME}
%env PIPELINE_NAME={PIPELINE_NAME}
%env RUNTIME_VERSION={RUNTIME_VERSION}
%env PYTHON_VERIONS={PYTHON_VERSION}
%env USE_KFP_SA={USE_KFP_SA}
```

Use TFX CLI to compile your pipeline.

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

Deploy your pipeline package to Cloud AI Platform




```
!tfx pipeline create  \
--pipeline_path=runner.py \
--endpoint={ENDPOINT} \
--build_target_image={CUSTOM_TFX_IMAGE}
```

Trigger model training on Cloud AI Platform

Create and monitor pipeline runs from Kubeflow Pipelines UI.