Introduction to TFX pipelines

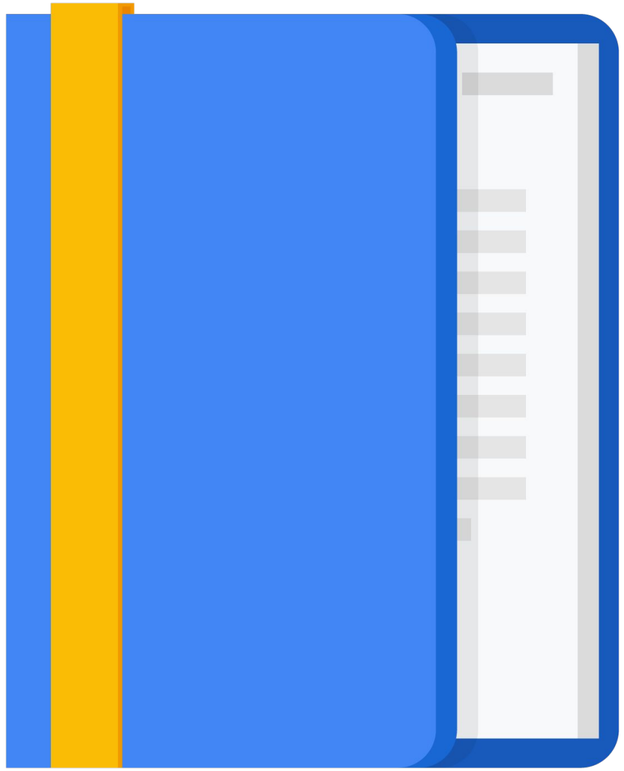
Doug Kelly
ML Solutions Engineer, Google Cloud

Agenda

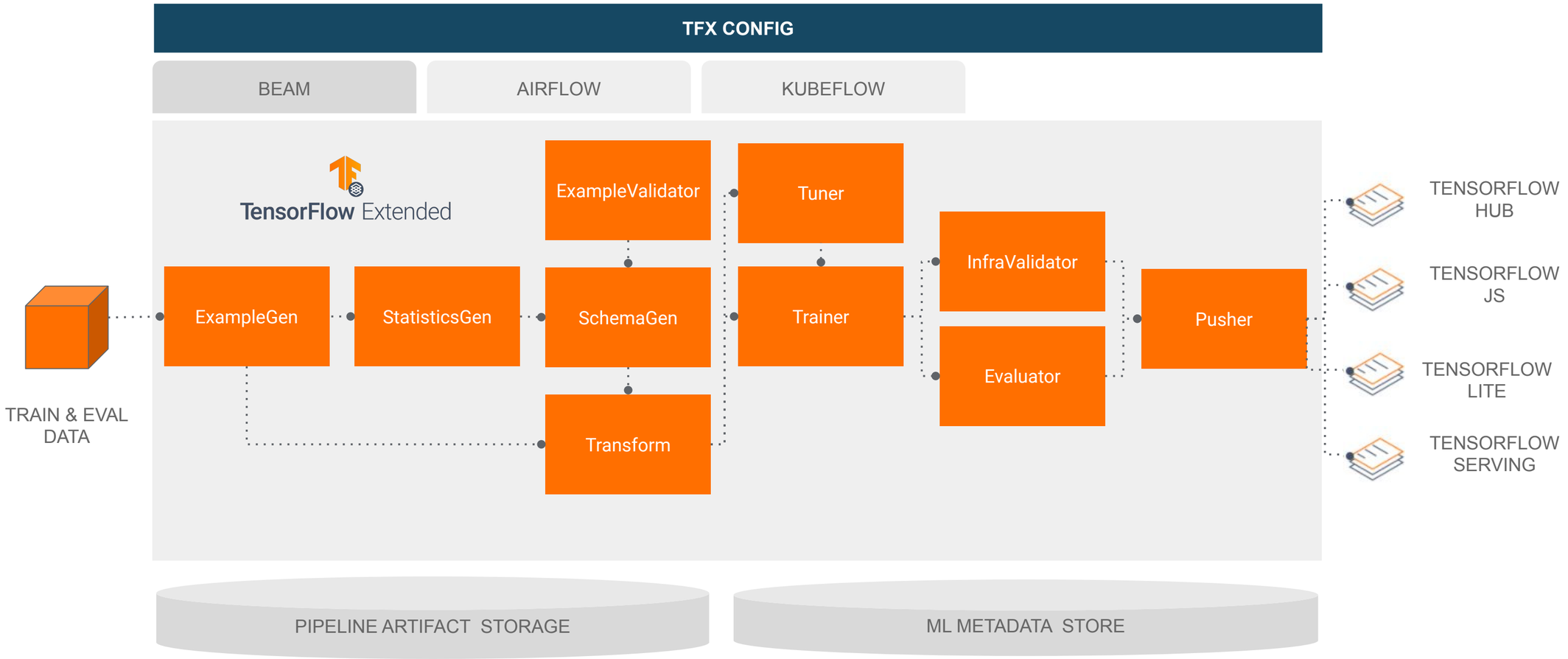
TensorFlow Extended (TFX)

TFX standard components

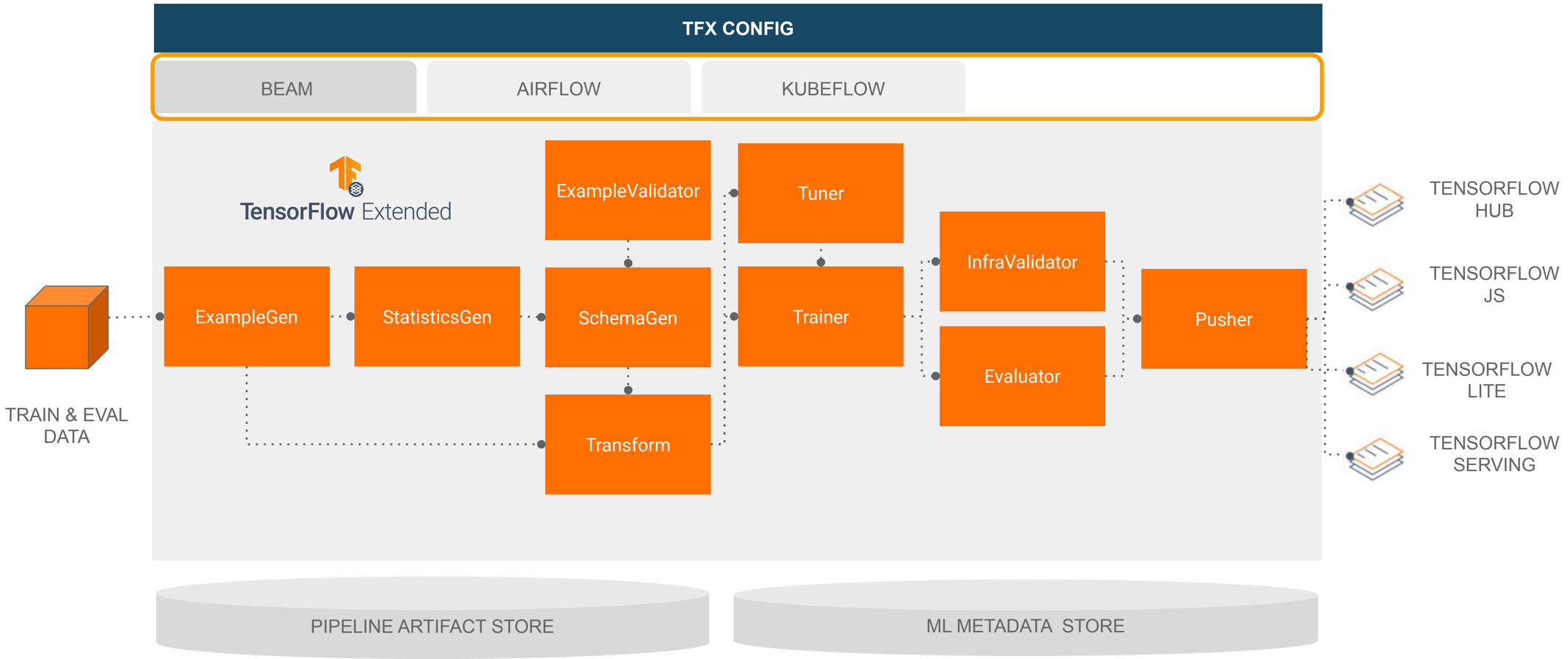
TFX libraries



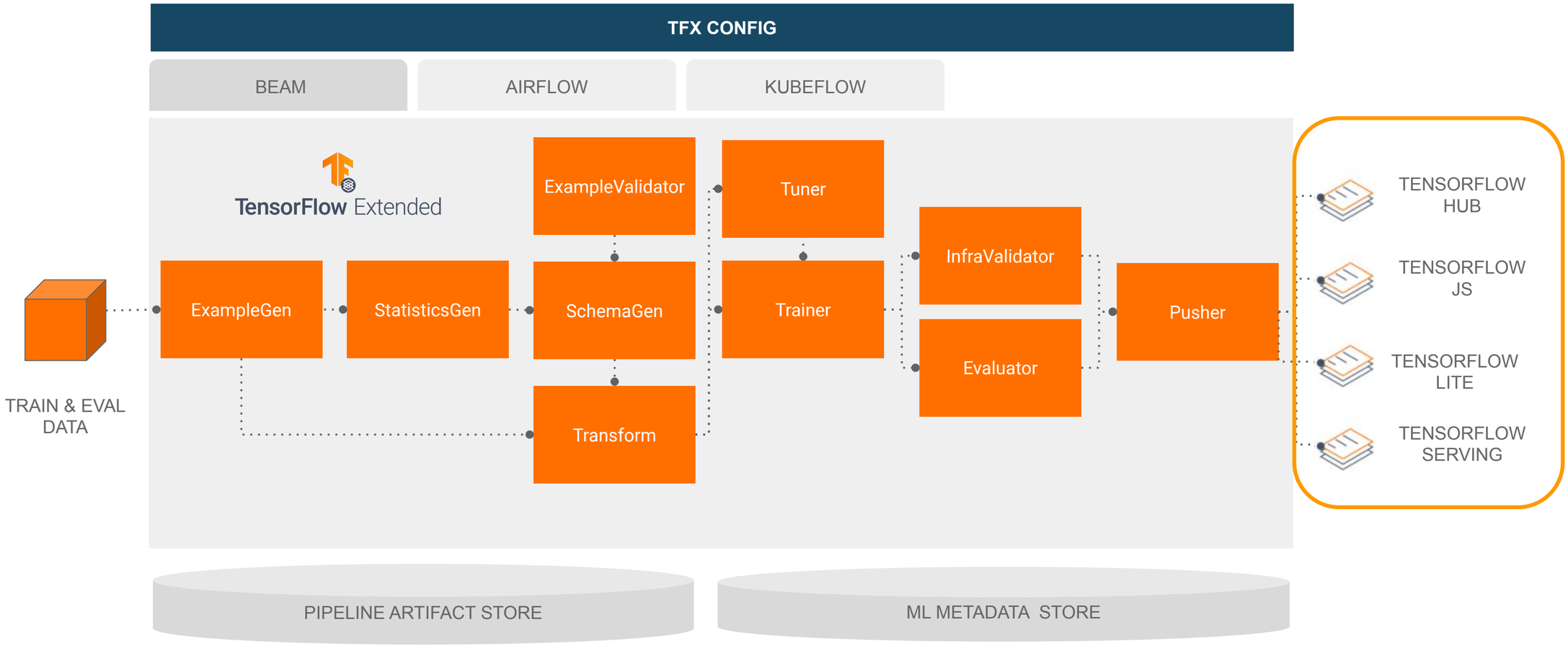
TFX: Google's production machine learning platform



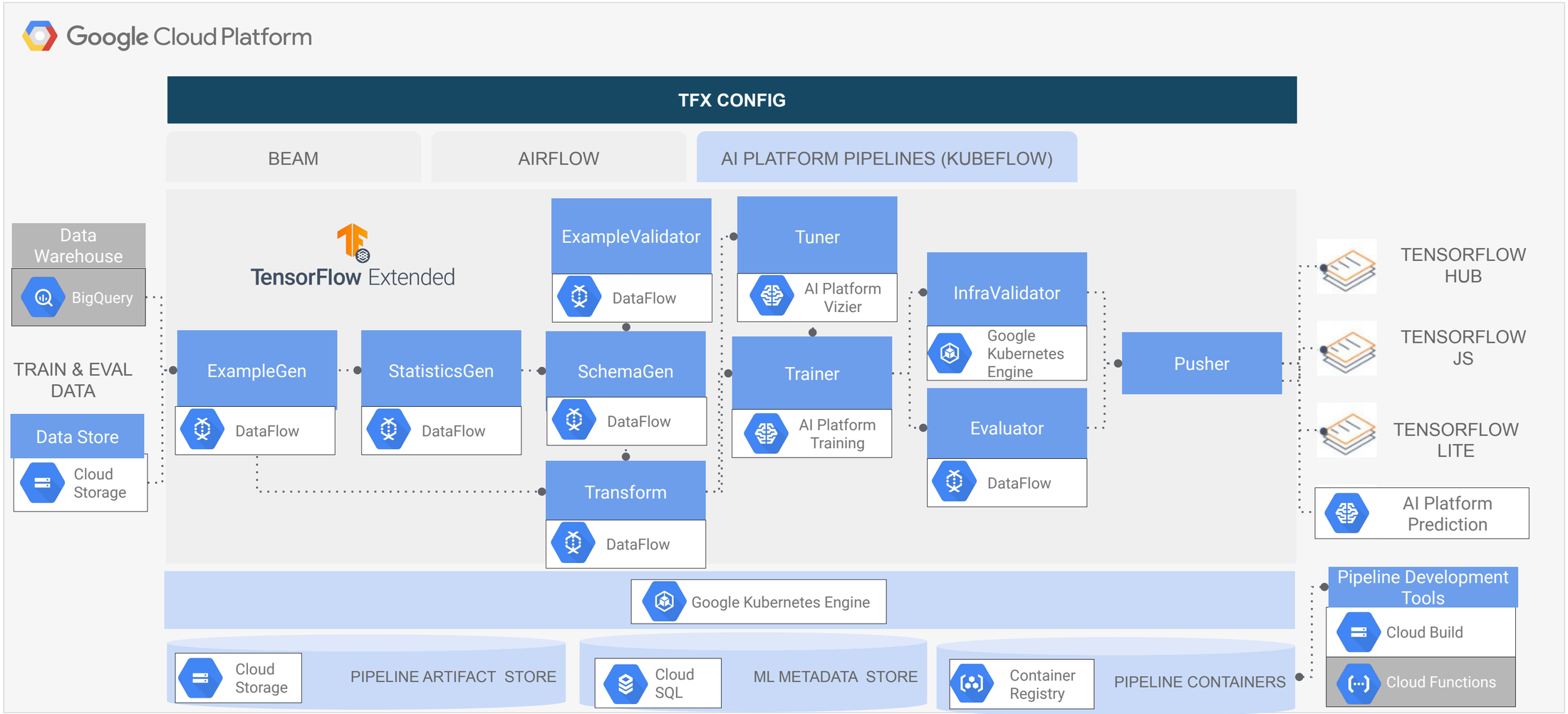
TFX: Google's production machine learning platform



TFX: Google's production machine learning platform



TFX: Google's production machine learning platform



TFX powers Alphabet's most important bets and products

Alphabet Subsidiaries

Google DeepMind verily

LOON Jigsaw

Google Products

Google Cloud AI Products

TFX powered


AutoML Natural Language


AutoML Tables


Recommendations AI

TFX integrations


AI Platform


Dataflow


BigQuery

TFX powers Alphabet's most important bets and products

Alphabet Subsidiaries

Google DeepMind verily

LOON Jigsaw

Google Products

Google Cloud AI Products

TFX powered


AutoML Natural Language


AutoML Tables


Recommendations AI

TFX integrations


AI Platform


Dataflow


BigQuery

TFX powers Alphabet's most important bets and products

Alphabet Subsidiaries

Google Products

Google Cloud AI Products

TFX powered

AutoML Natural Language

AutoML Tables

Recommendations AI

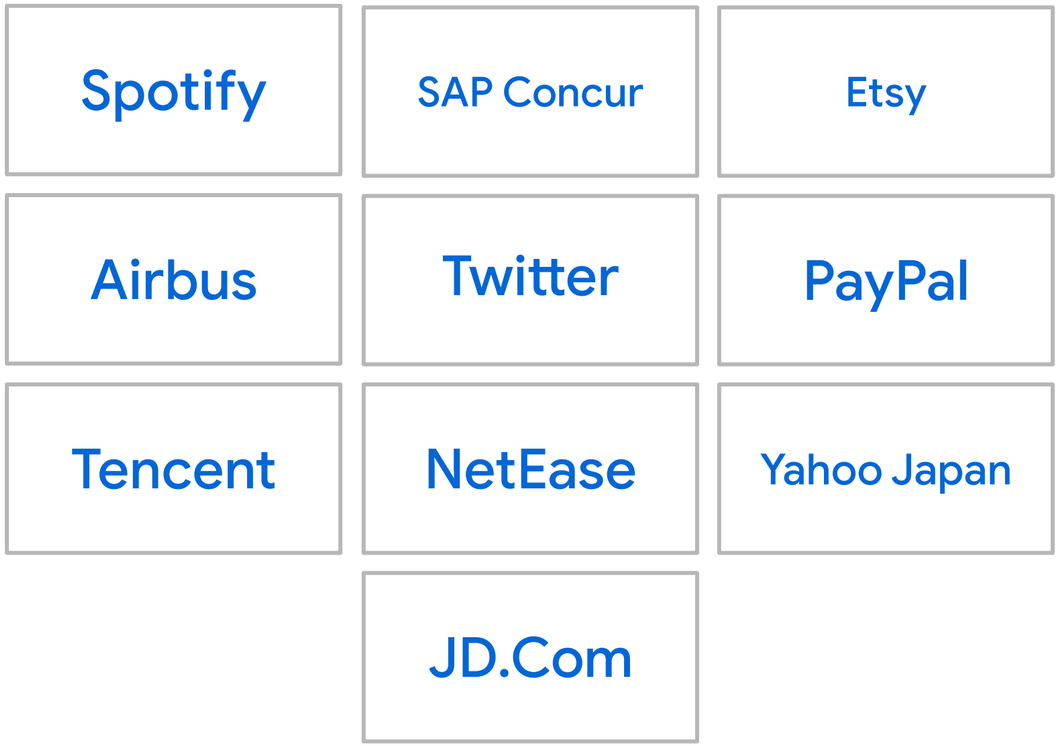
TFX integrations

AI Platform

Dataflow

BigQuery

OSS TFX enables global ML use cases



OSS TFX enables global ML use cases

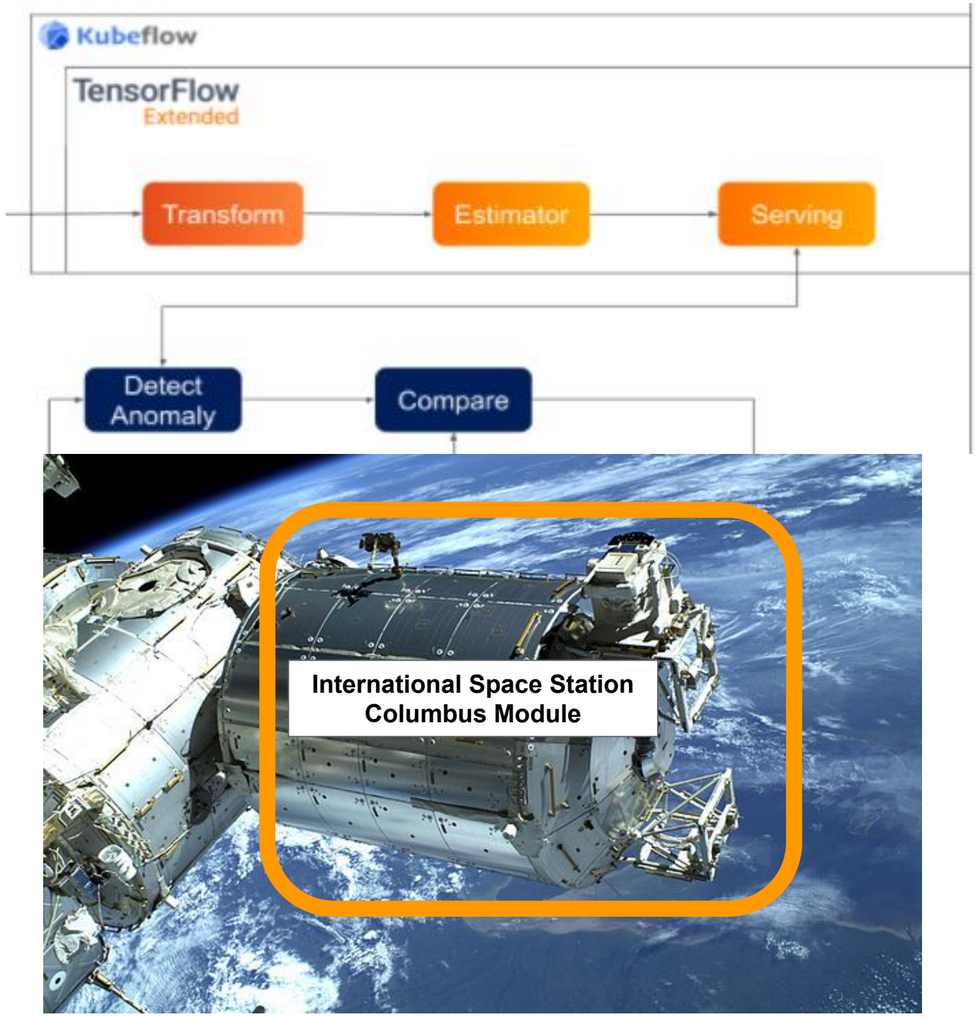
“...we have re-tooled our machine learning platform to use TensorFlow. This yielded significant productivity gains while positioning ourselves to take advantage of the latest industry research.”

Ranking Tweets with TensorFlow - Twitter

https://goo.gle/tf-twitter-rank

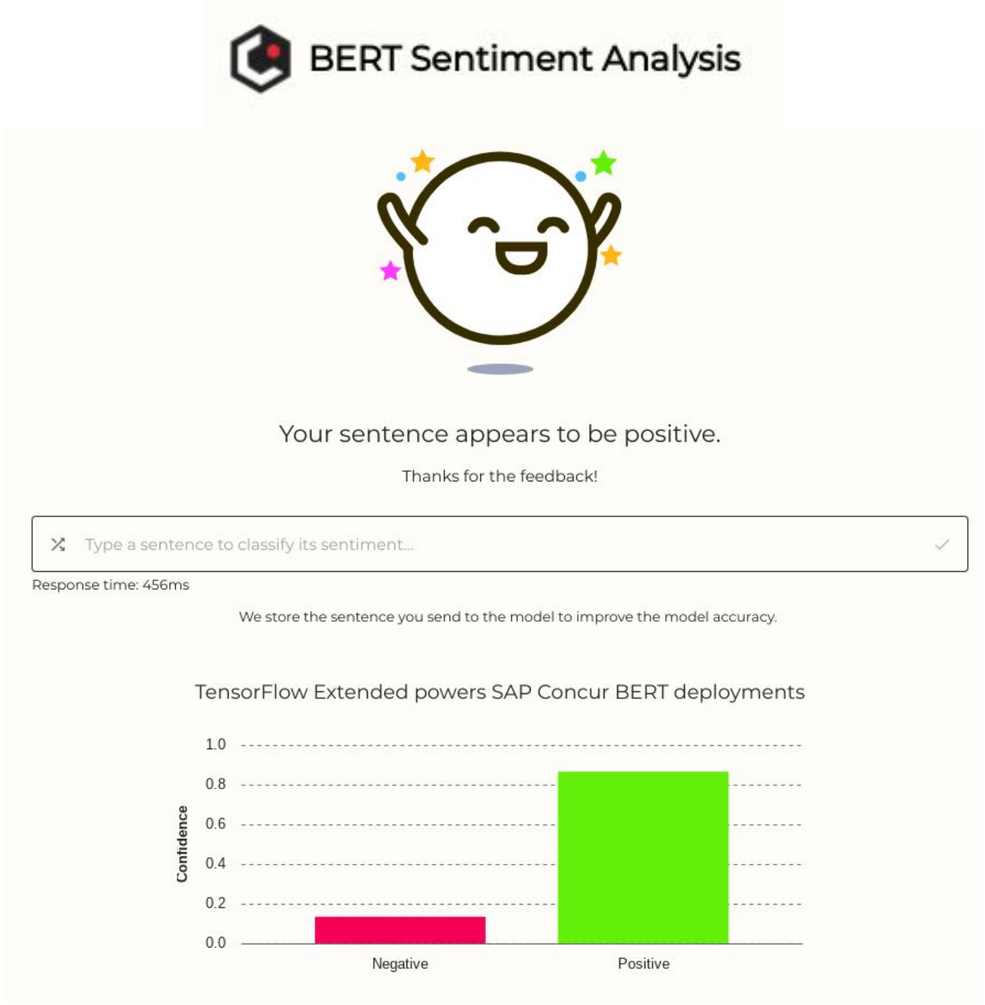
Spotify	SAP Concur	Etsy
Airbus	Twitter	PayPal
Tencent	NetEase	Yahoo Japan
	JD.Com	

OSS TFX enables global ML use cases



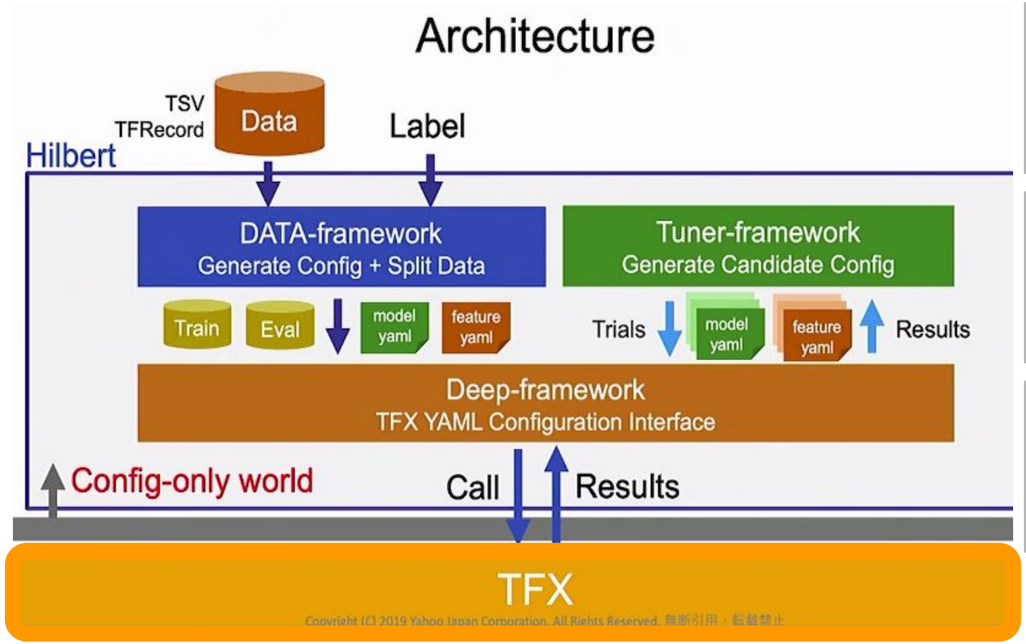
Spotify	SAP Concur	Etsy
Airbus	Twitter	PayPal
Tencent	NetEase	Yahoo Japan
	JD.Com	

OSS TFX enables global ML use cases



Spotify	SAP Concur	Etsy
Airbus	Twitter	PayPal
Tencent	NetEase	Yahoo Japan
	JD.Com	

OSS TFX enables global ML use cases



Spotify	SAP Concur	Etsy
Airbus	Twitter	PayPal
Tencent	NetEase	Yahoo Japan
	JD.Com	

TFX's Lineage

Sibyl

(2007 - 2019) goo.gle/sibyl-video

User-focused, production ML, massive scale, (somewhat) flexible

Previously, one of the most widely used E2E ML platforms at Google.

TFX

(2016 - Present) tensorflow.org/tfx

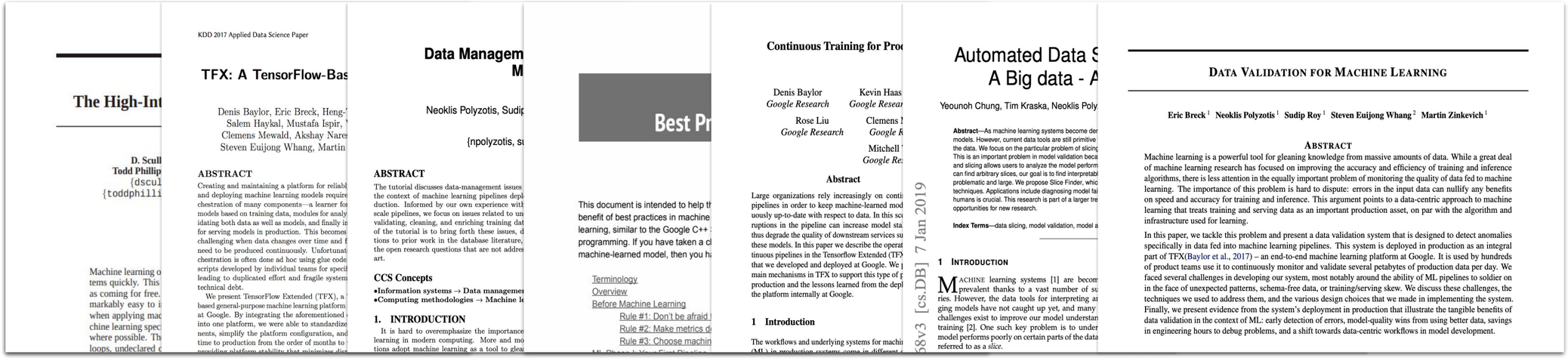
User-focused, production ML, massive scale, **flexible, modular, portable**

The most widely used E2E ML platform at **Alphabet** (including Google).

E2E OSS ML platform **on-premise** and on **Google Cloud.**

TFX has Google's ML best practices built in

Machine Learning: The High Interest Credit Card of Technical Debt. NeurIPS (2015).
TFX: A TensorFlow-Based Production-Scale Machine Learning Platform. KDD (2017).
Data Management Challenges in Production Machine Learning. SIGMOD (2017).
Rules of Machine Learning: Best Practices for ML Engineering. Google AI Web (2017).
Continuous Training for Production ML in the TFX Platform. OpML (2019).
Slice Finder: Automated Data Slicing for Model Validation. ICDE (2019).
Data Validation for Machine Learning. SysML (2019).

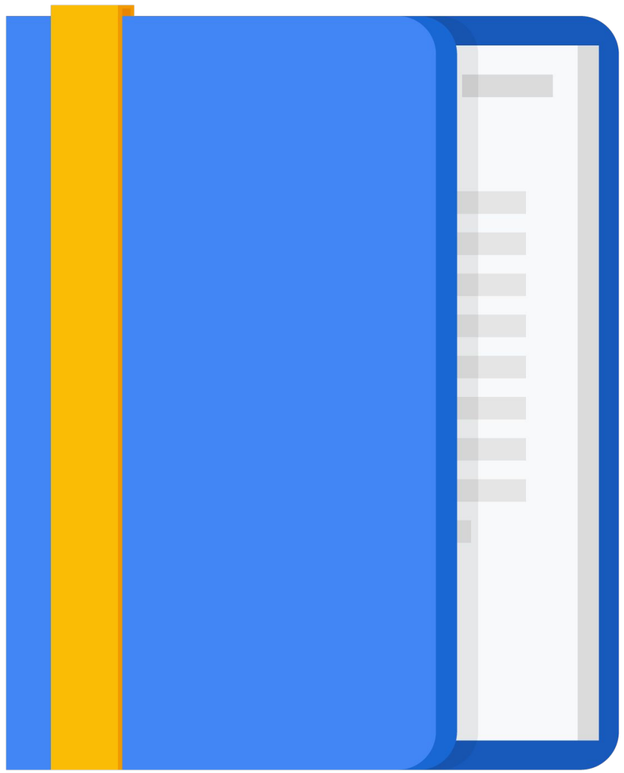


Agenda

TensorFlow Extended (TFX)

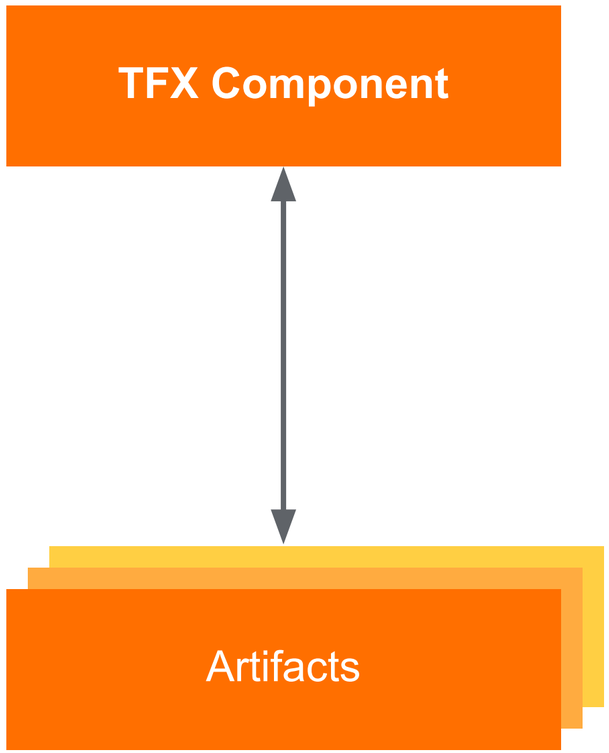
TFX standard components

TFX libraries



A TFX **component** implements a machine learning task

TFX **components** produce and consume **artifacts**



TFX **components** have 5 elements

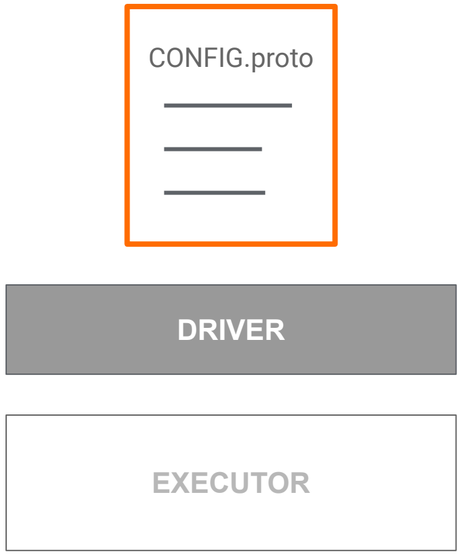
Component specification: A configuration protocol buffer defines how components communicate with each other via input and output artifact channels and runtime parameters.

TFX **components** have 5 elements

Component specification: A configuration protobuf defines how components communicate with each other via input and output artifact channels and runtime parameters.

Component driver: A driver coordinates job execution.

TFX **components** have 5 elements

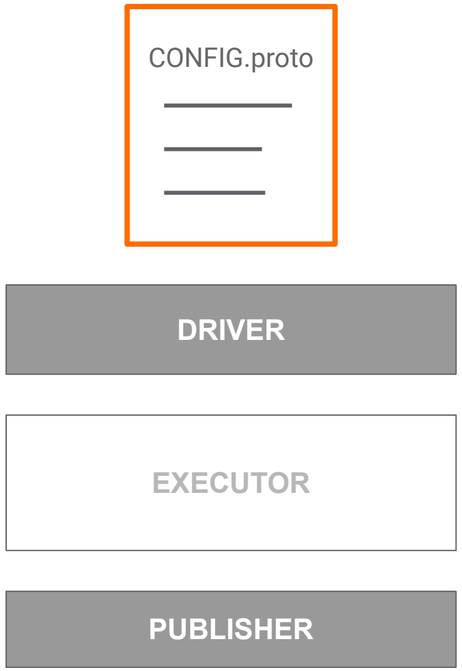


Component specification: A configuration protobuf defines how components communicate with each other via input and output artifact channels and runtime parameters.

Component driver: A driver coordinates job execution.

Component executor: Code to perform ML workflow step such as data preprocessing or TensorFlow model training.

TFX **components** produce and consume **artifacts**



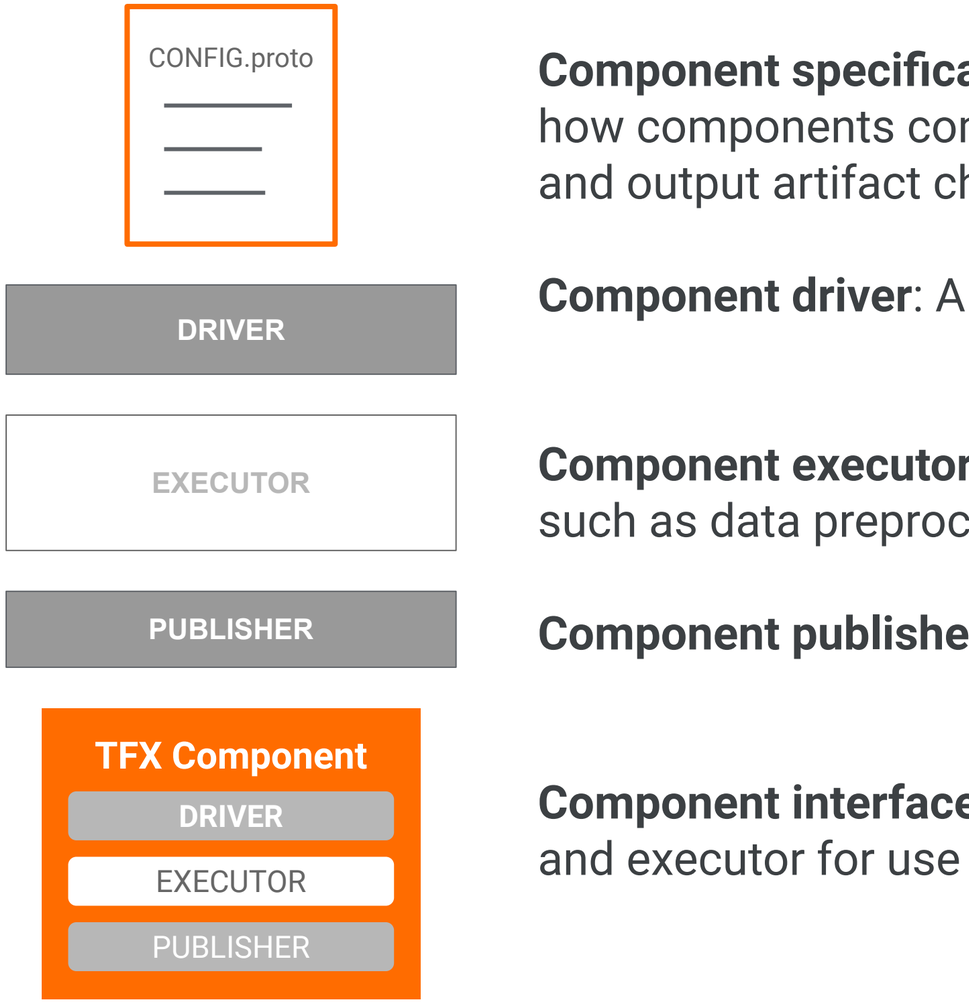
Component specification: A configuration protobuf defines how components communicate with each other via input and output artifact channels and runtime parameters.

Component driver: A driver coordinates job execution.

Component executor: Code to perform ML workflow step such as data preprocessing or TensorFlow model training.

Component publisher: Updates ML Metadata store.

TFX **components** produce and consume **artifacts**



Component specification: A configuration protobuf defines how components communicate with each other via input and output artifact channels and runtime parameters.

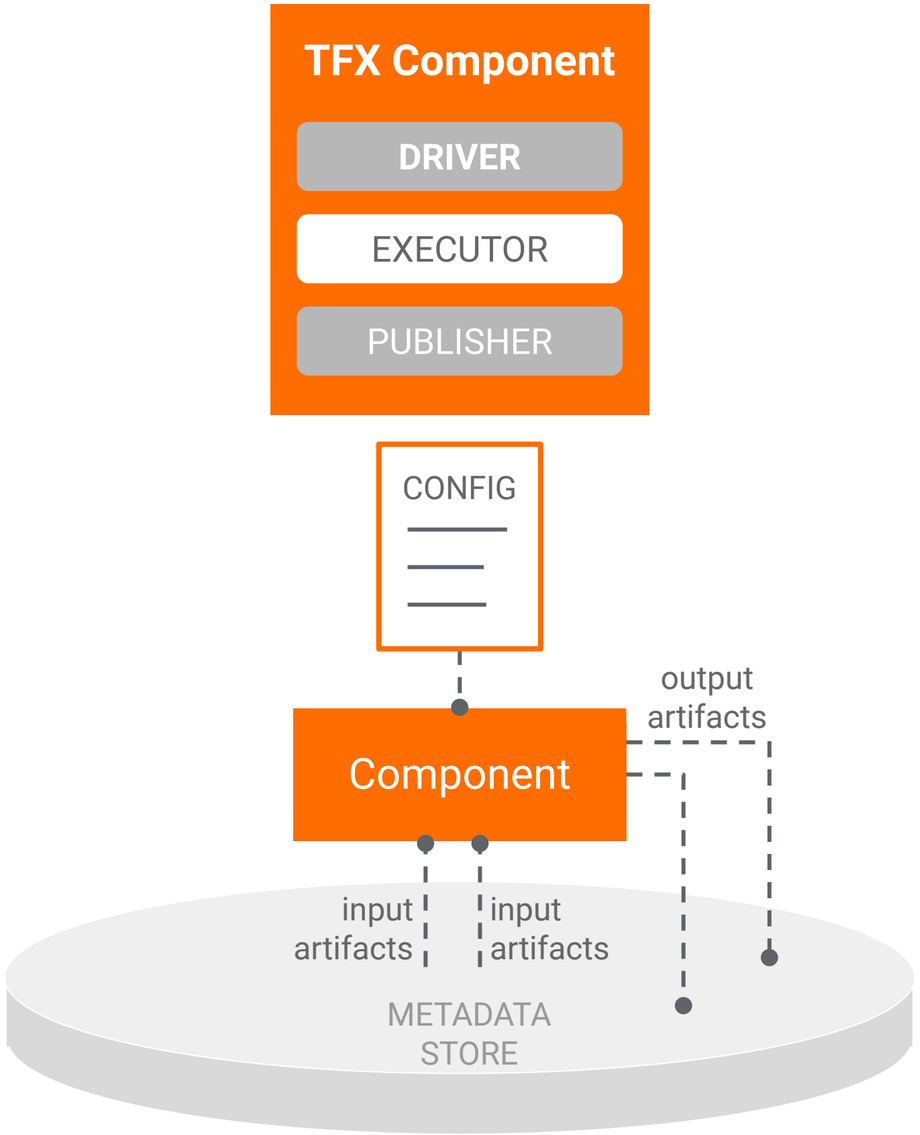
Component driver: A driver coordinates job execution.

Component executor: Code to perform ML workflow step such as data preprocessing or TensorFlow model training.

Component publisher: Updates ML Metadata store.

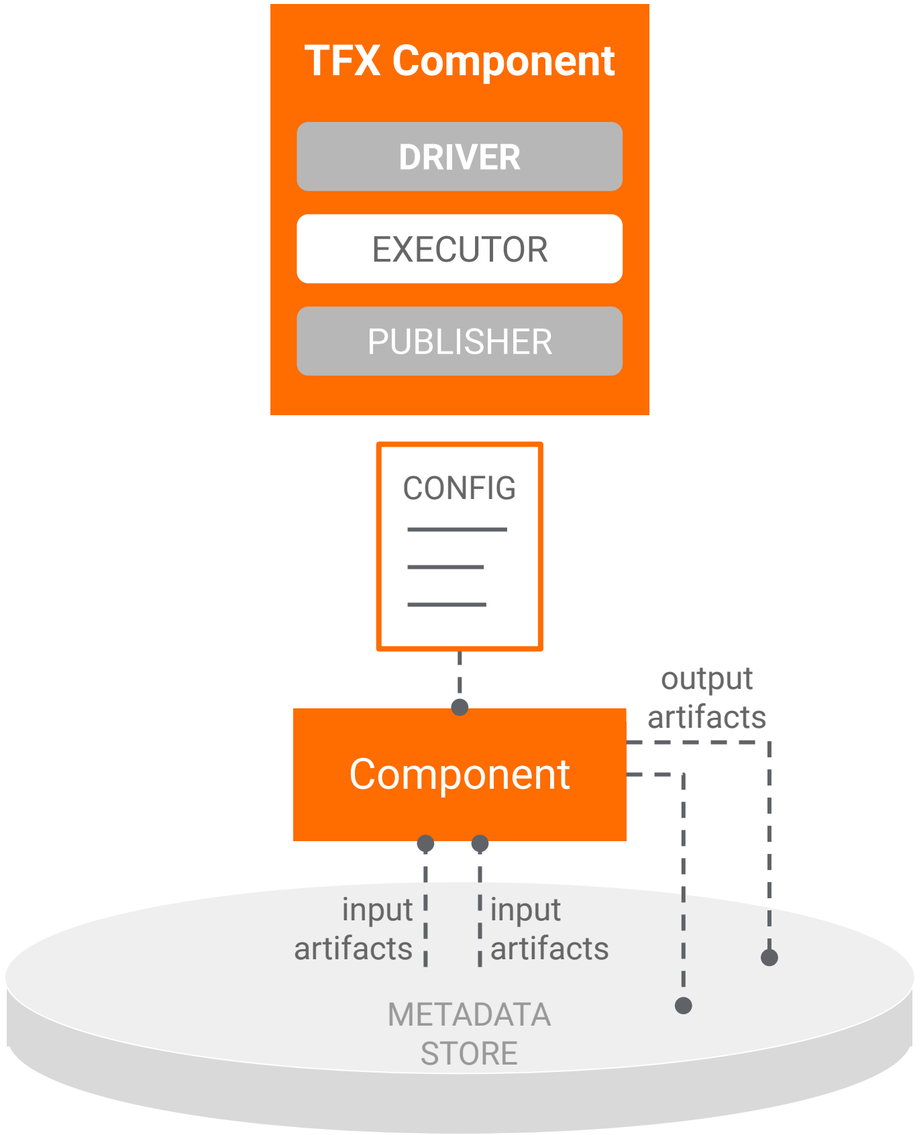
Component interface: packages component specification and executor for use in pipeline.

TFX **components** at runtime



TFX **components** at runtime

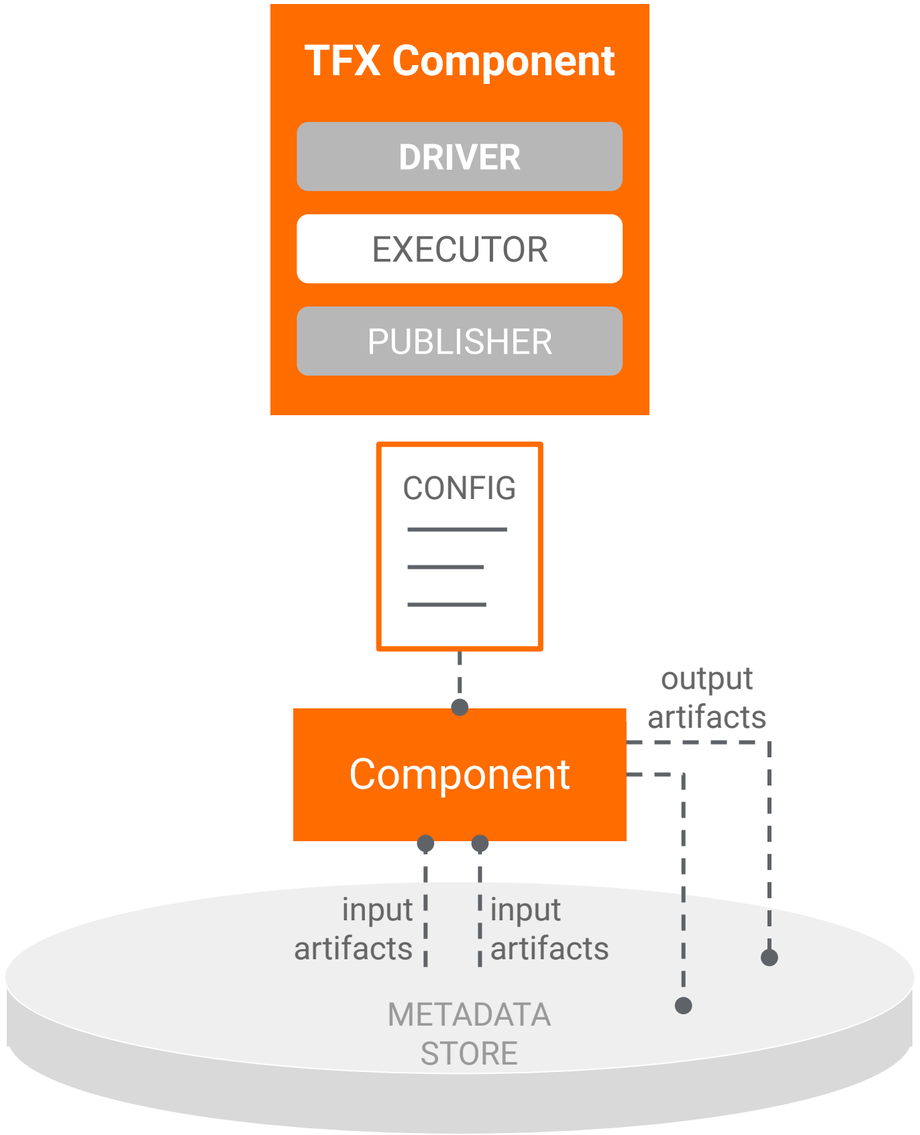
1. **Driver** reads the component specification for parameters and artifacts and retrieves input artifacts from the metadata store for the component.



TFX **components** at runtime

1. **Driver** reads the component specification for parameters and artifacts and retrieves input artifacts from the metadata store for the component.

2. **Executor** performs computation on artifacts.

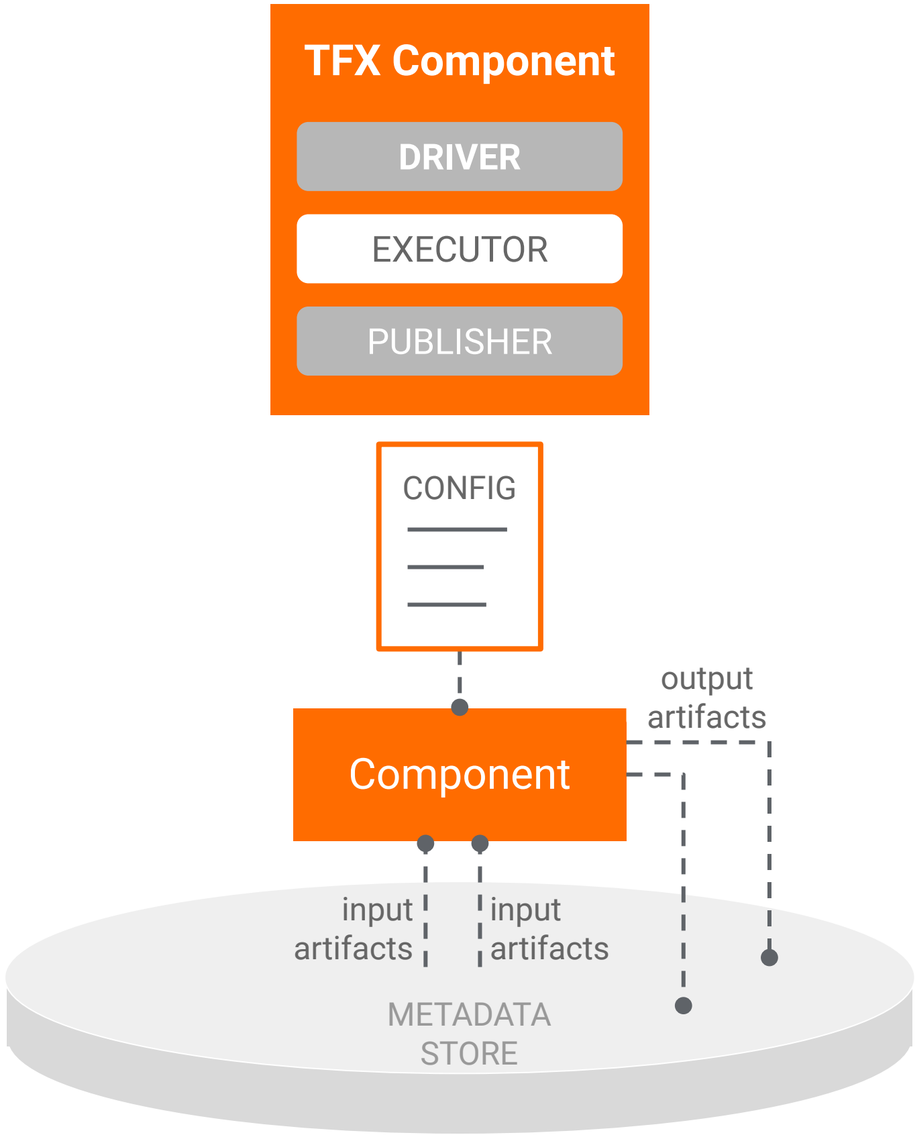


TFX **components** at runtime

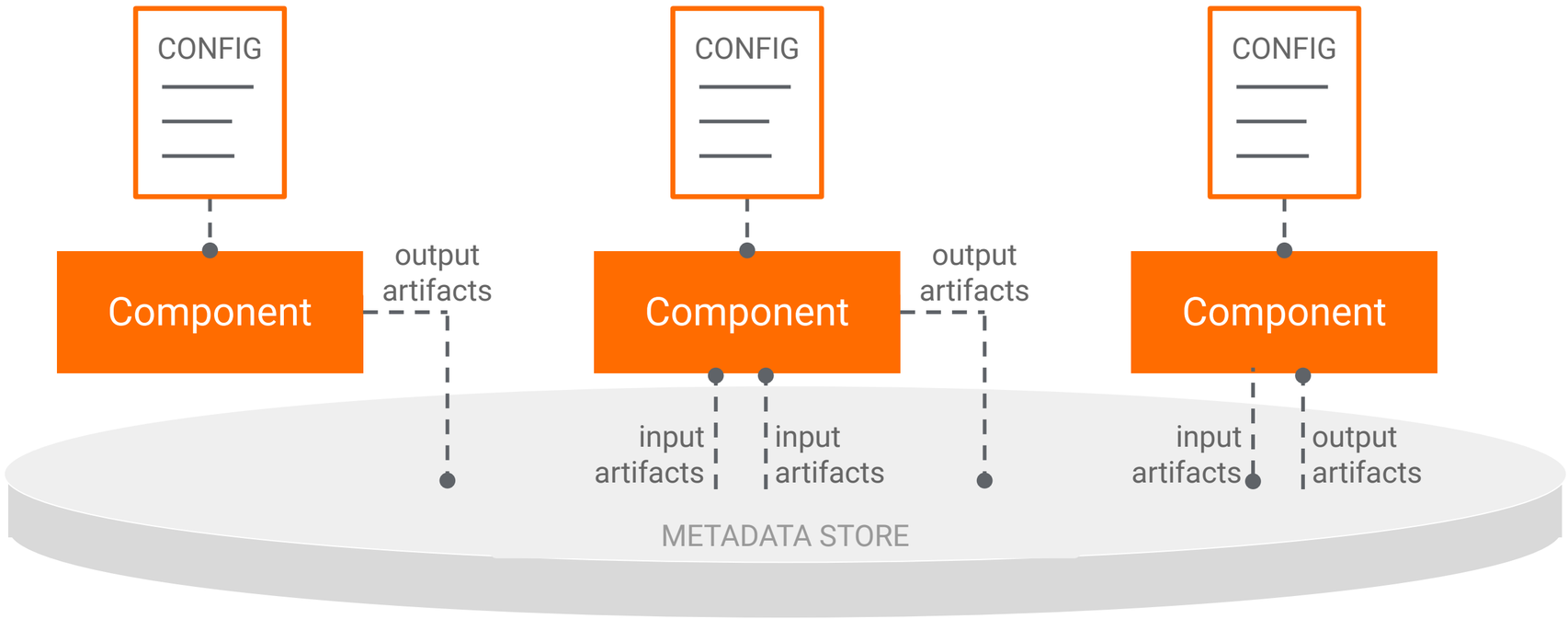
1. **Driver** reads the component specification for parameters and artifacts and retrieves input artifacts from the metadata store for the component.

2. **Executor** performs computation on artifacts.

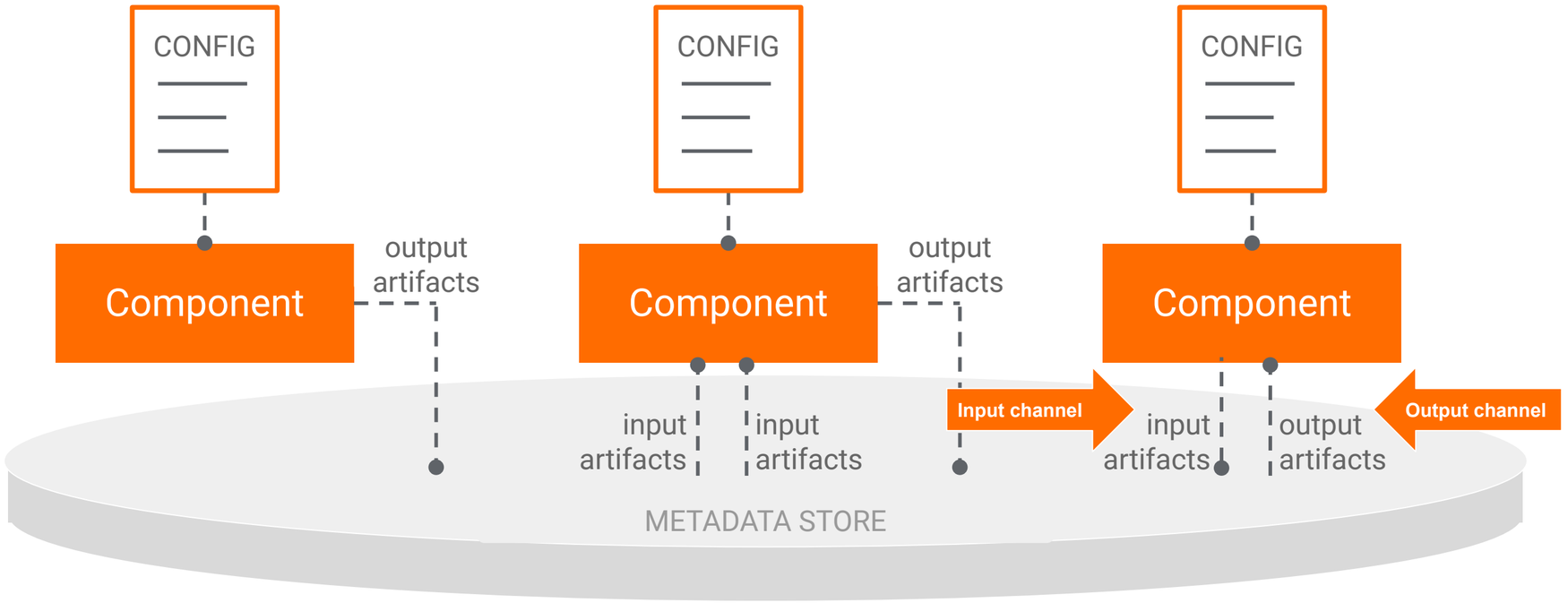
3. **Publisher** uses the component specification and executor results to store the component's output artifacts in the metadata store.



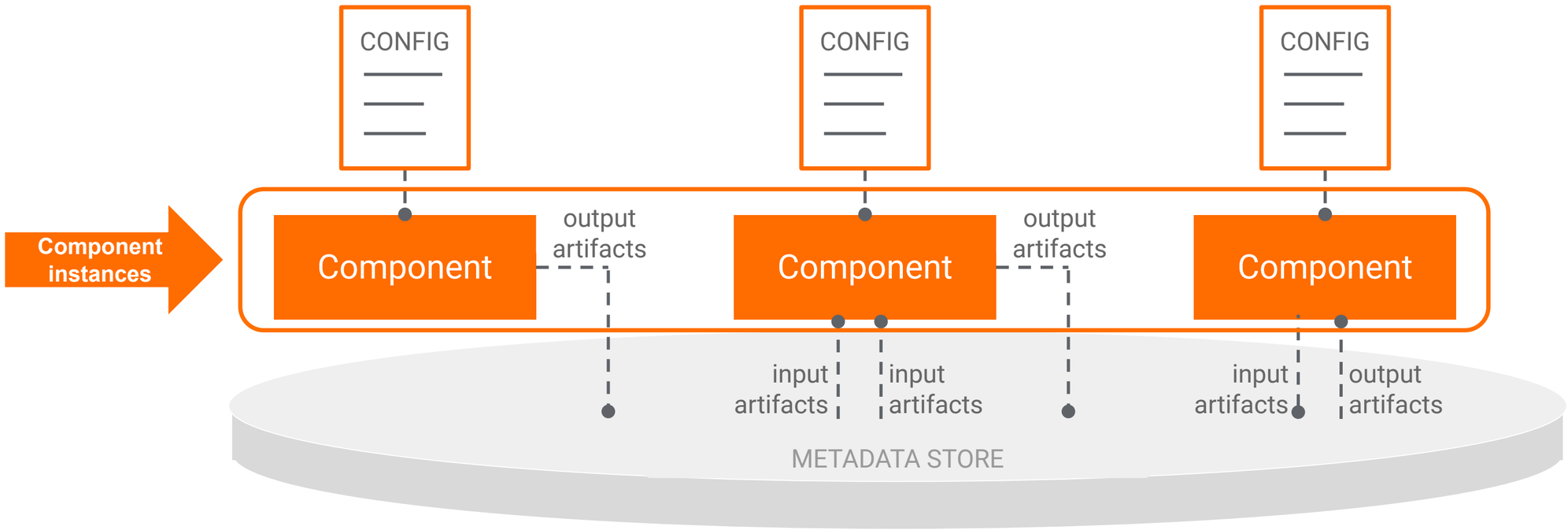
A TFX **pipeline** is a sequence of **components** connected by **channels** in a directed acyclic graph (DAG) of **artifact** dependencies



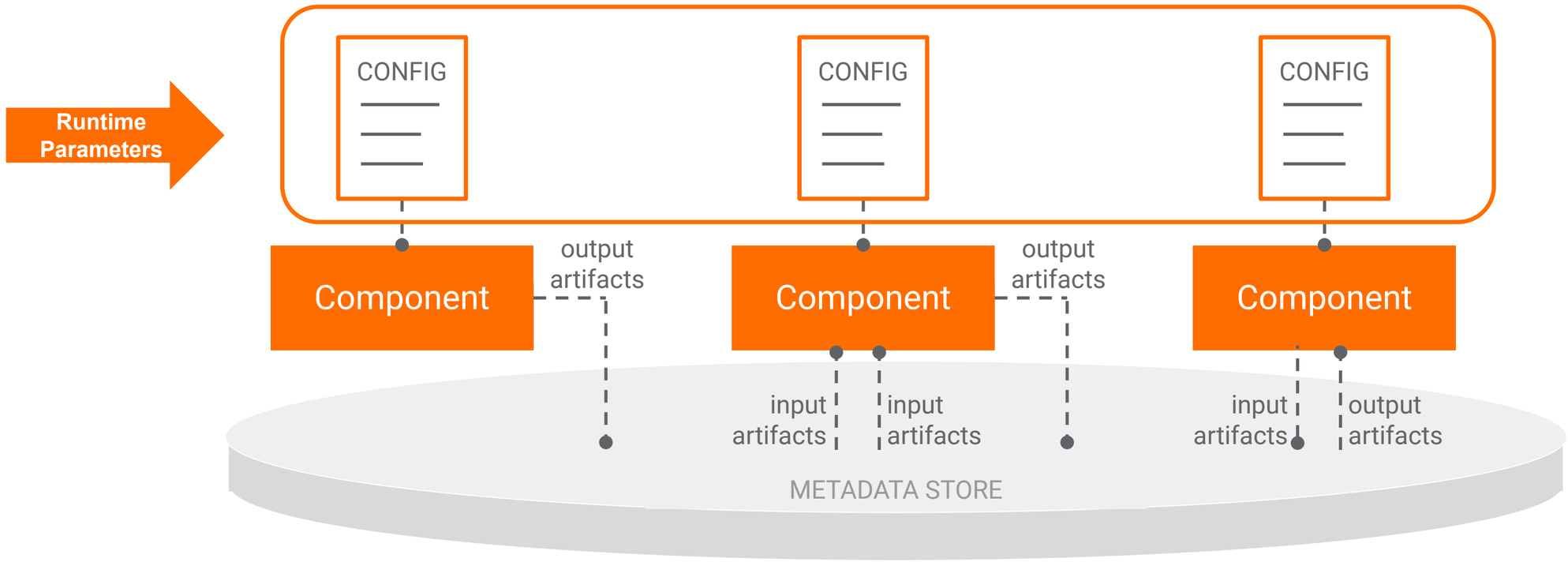
A TFX **pipeline** is a sequence of **components** connected by **channels** in a directed acyclic graph (DAG) of **artifact** dependencies



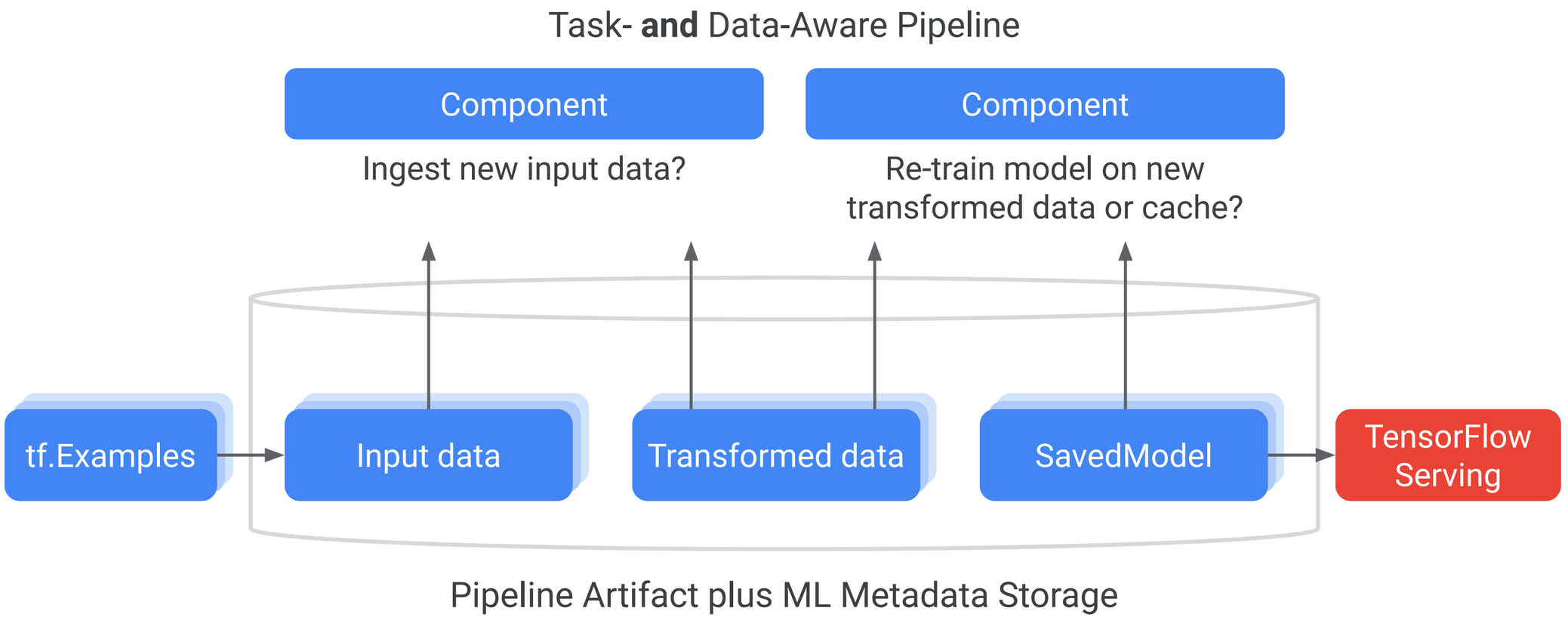
A TFX **pipeline** is a sequence of **components** connected by **channels** in a directed acyclic graph (DAG) of **artifact** dependencies



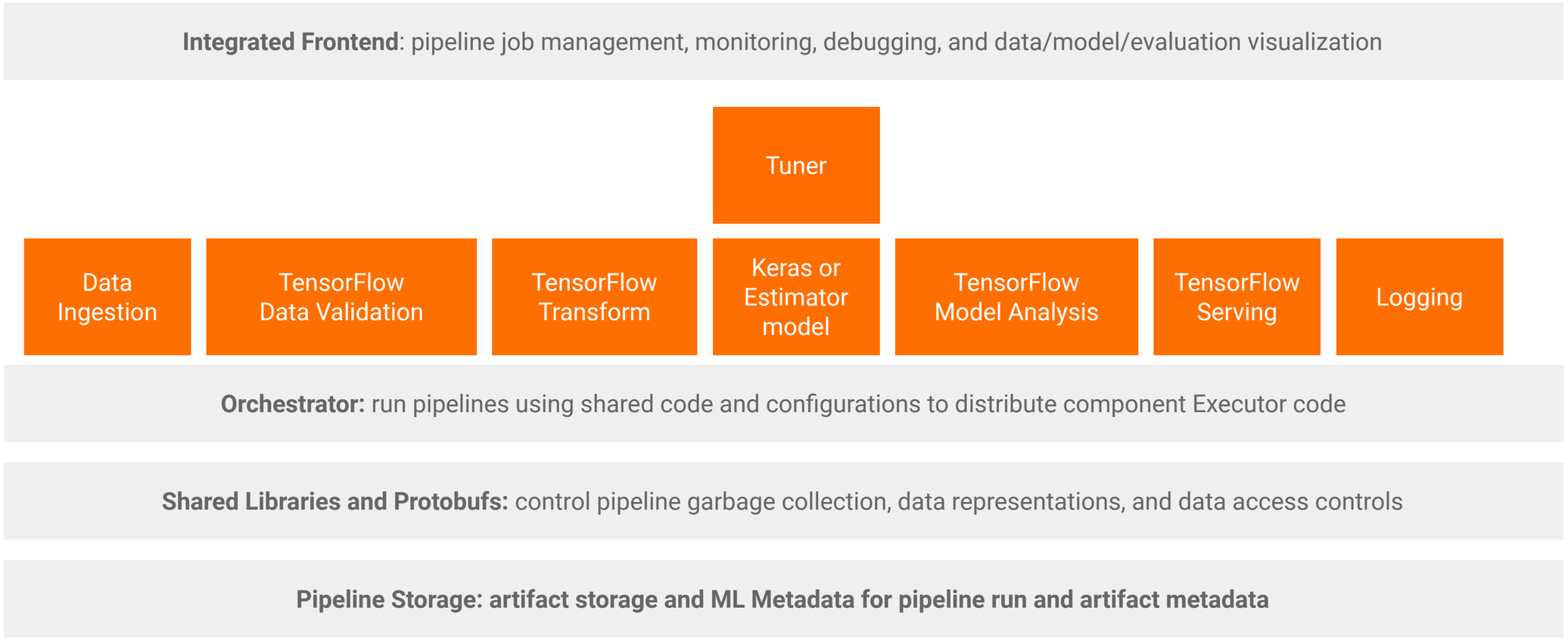
A TFX **pipeline** is a sequence of **components** connected by **channels** in a directed acyclic graph (DAG) of **artifact** dependencies



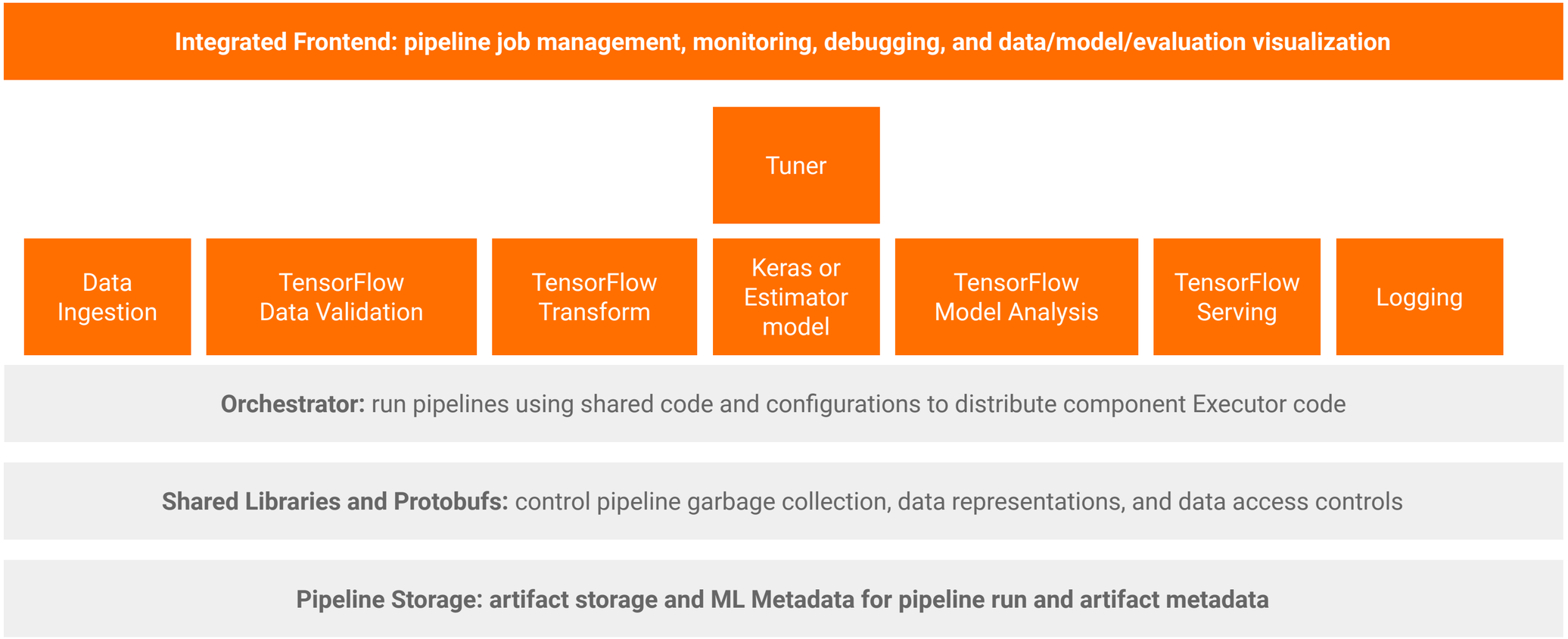
TFX **pipelines** use **ML Metadata** for artifact management and an **orchestrator** for sequential component execution



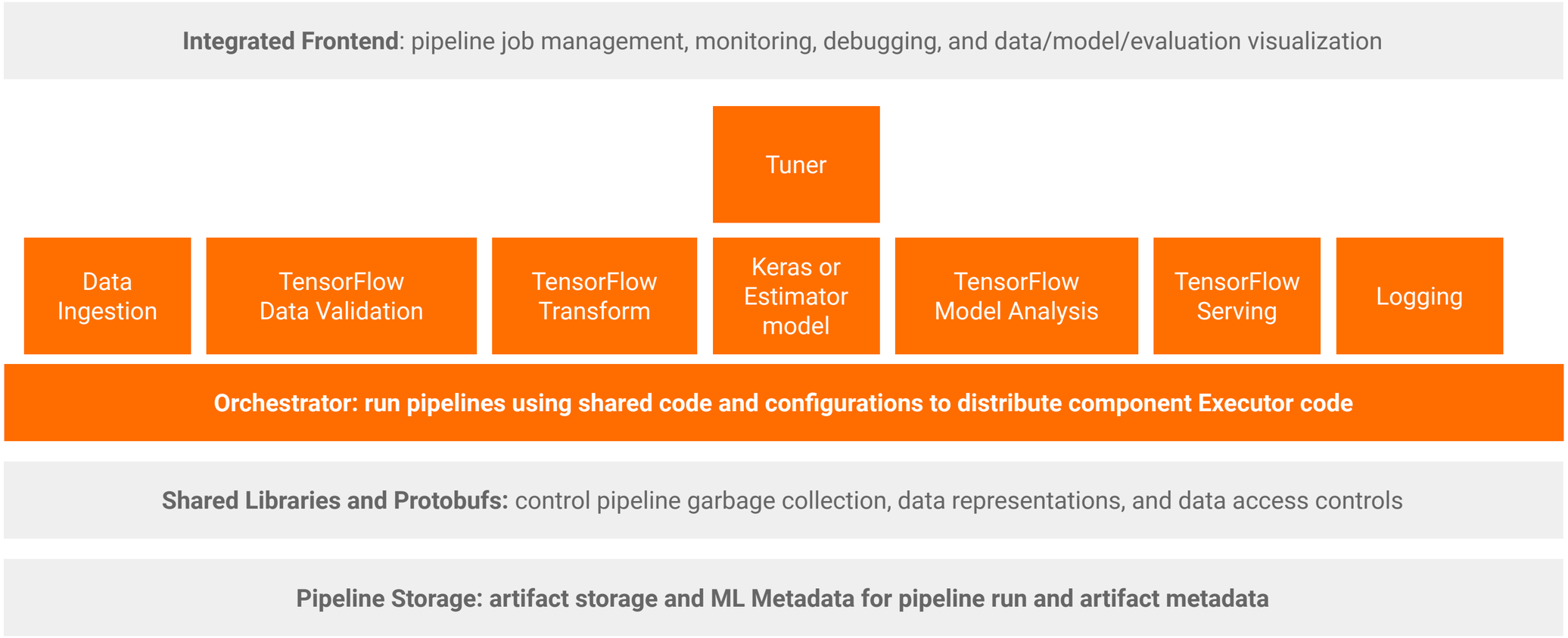
Horizontal layers coordinate pipeline components



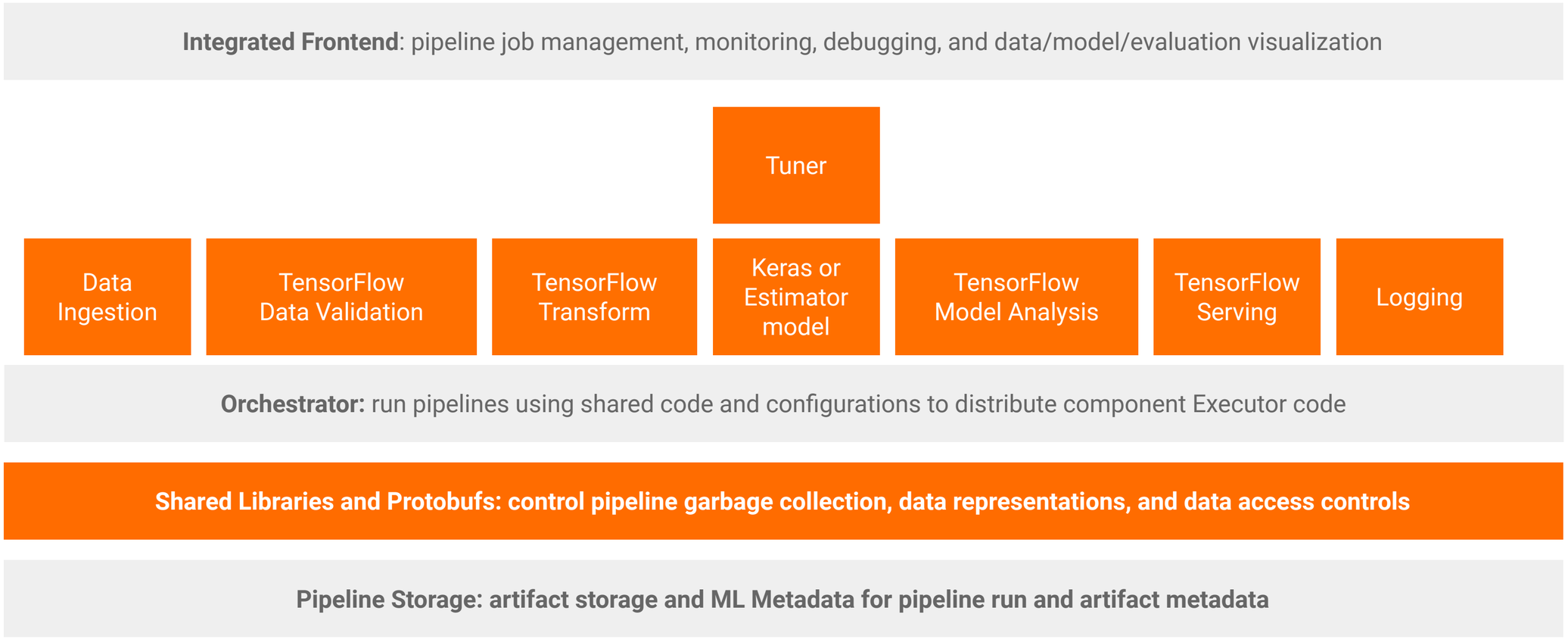
Horizontal layers coordinate pipeline components



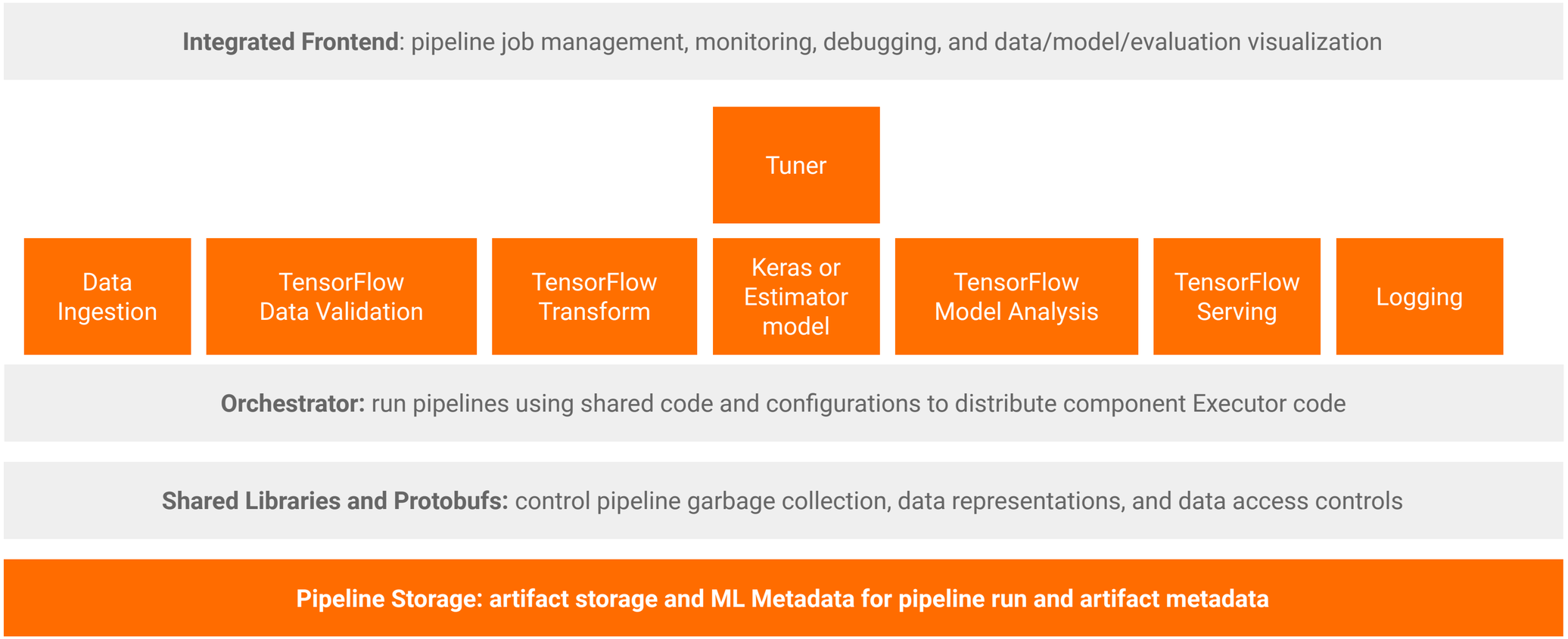
Horizontal layers coordinate pipeline components



Horizontal layers coordinate pipeline components

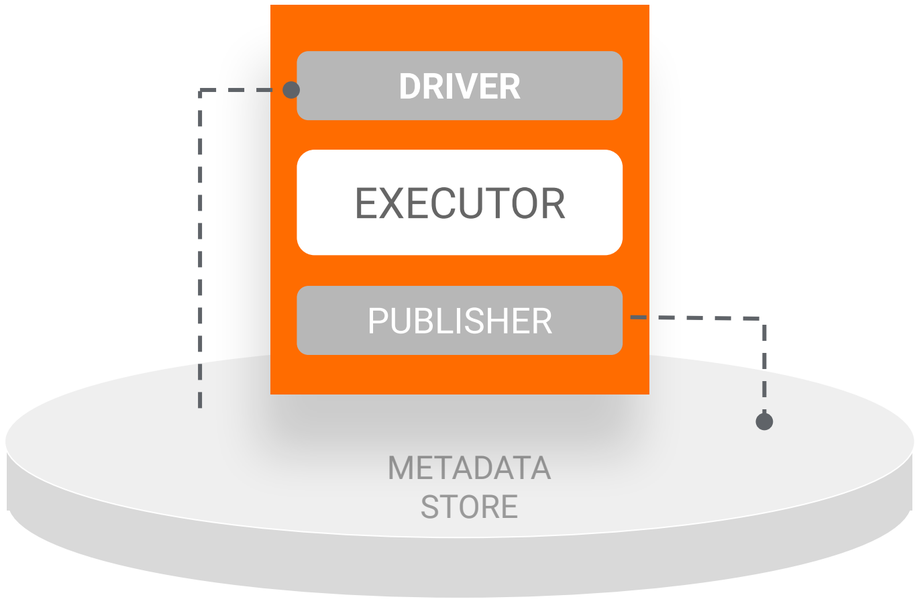


Horizontal layers coordinate pipeline components

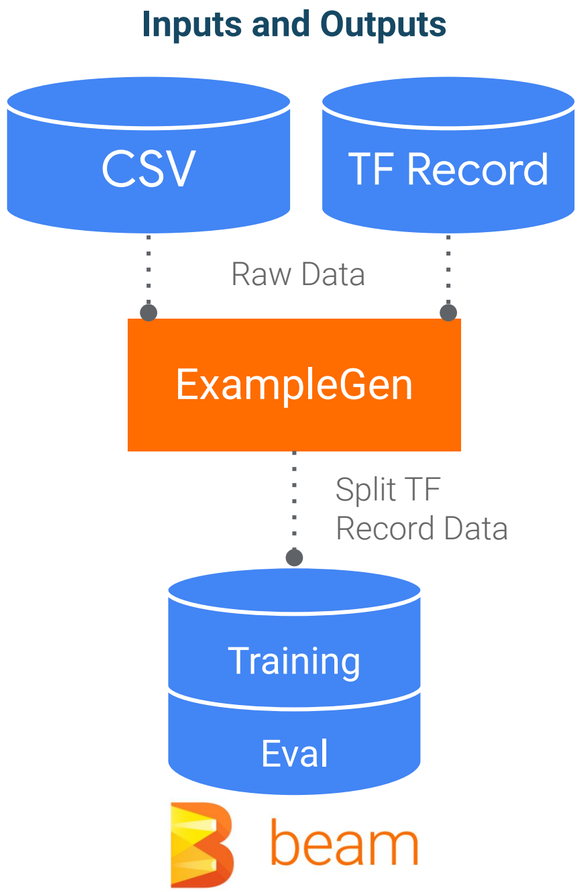


TFX standard components

Compose ML pipelines faster with reusable components pre-configured with production best practices.

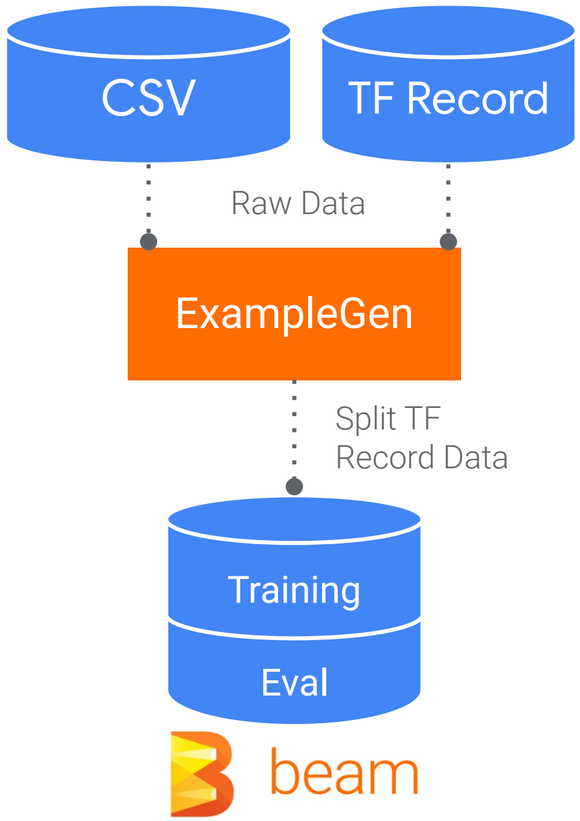


Component: **ExampleGen**



Component: **ExampleGen**

Inputs and Outputs

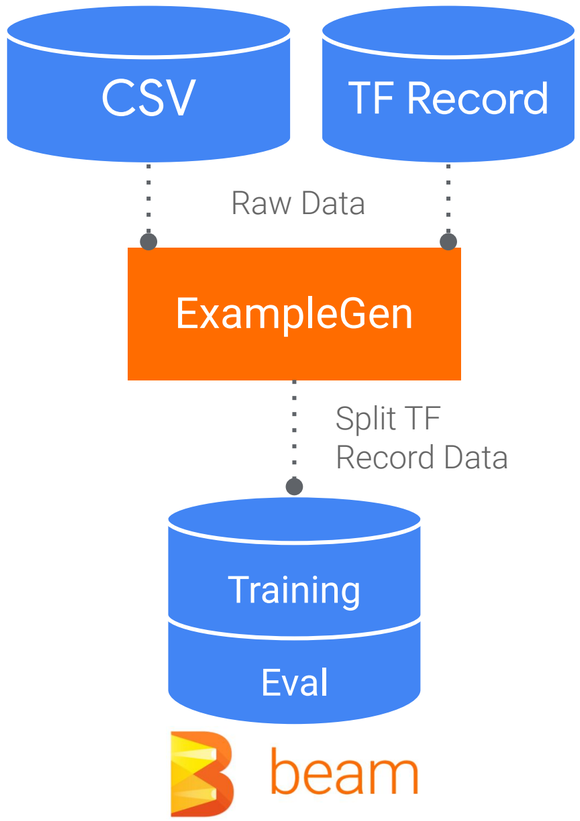


Configuration

```
output_config = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=4),
        example_gen_pb2.SplitConfig.Split(name='dev', hash_buckets=1)
    ]))

example_gen = tfx.components.CsvExampleGen(
    instance_name='Data_Extraction_Spliting',
    input=external_input(DATA_ROOT),
    output_config=output_config
)
```

Component: **ExampleGen**

Inputs and Outputs



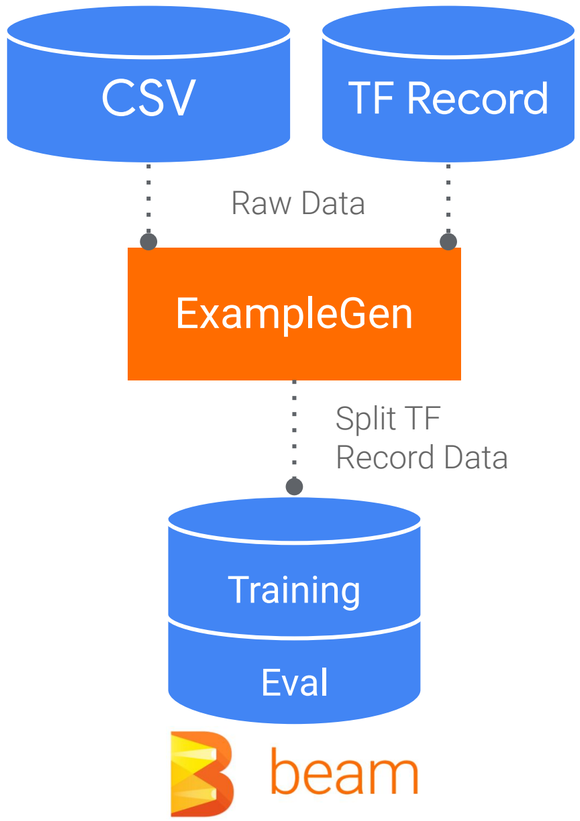
Configuration

```
output_config = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=4),
        example_gen_pb2.SplitConfig.Split(name='dev', hash_buckets=1)
    ]))

example_gen = tfx.components.CsvExampleGen(
    instance_name='Data_Extraction_Spliting',
    input=external_input(DATA_ROOT),
    output_config=output_config
)
```

ML project lifecycle benefits

Component: **ExampleGen**

Inputs and Outputs



Configuration

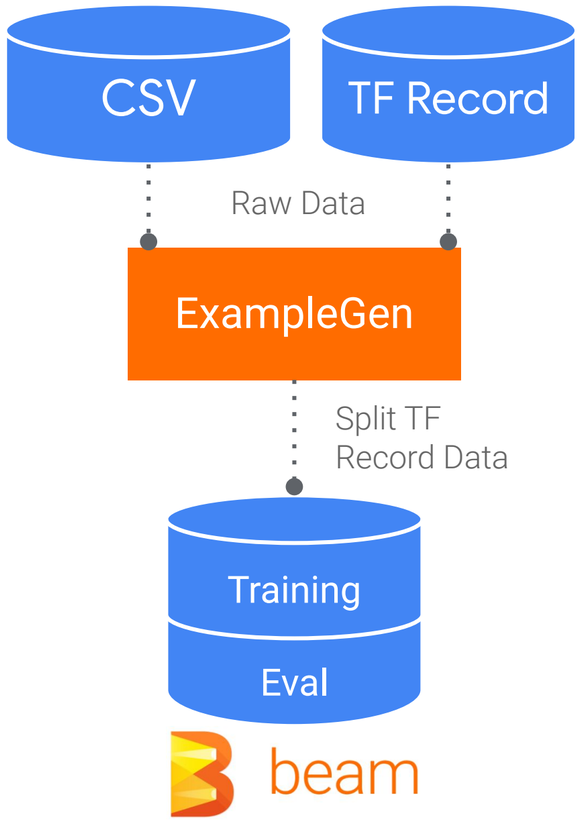
```
output_config = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=4),
        example_gen_pb2.SplitConfig.Split(name='dev', hash_buckets=1)
    ]))

example_gen = tfx.components.CsvExampleGen(
    instance_name='Data_Extraction_Spliting',
    input=external_input(DATA_ROOT),
    output_config=output_config
)
```

ML project lifecycle benefits

- Configurable and reproducible data partitioning and shuffling

Component: **ExampleGen**

Inputs and Outputs



Configuration

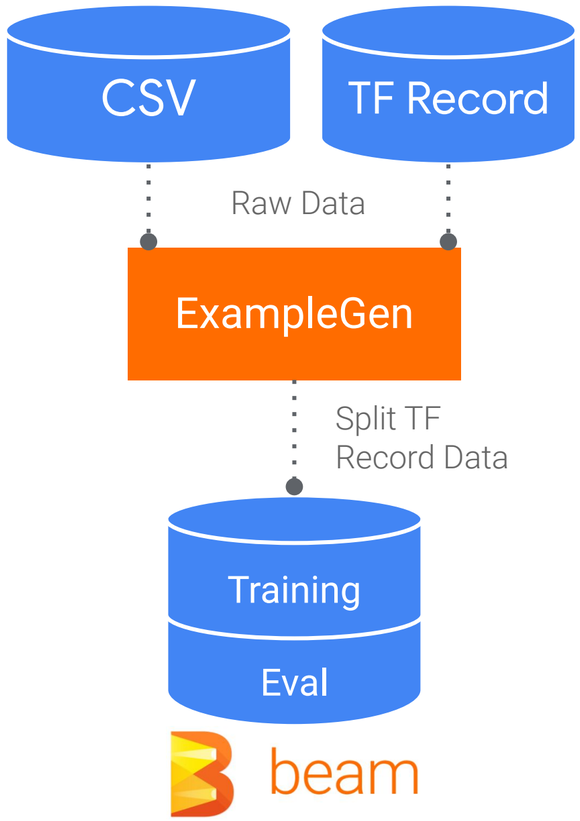
```
output_config = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=4),
        example_gen_pb2.SplitConfig.Split(name='dev', hash_buckets=1)
    ]))

example_gen = tfx.components.CsvExampleGen(
    instance_name='Data_Extraction_Spliting',
    input=external_input(DATA_ROOT),
    output_config=output_config
)
```

ML project lifecycle benefits

- Configurable and reproducible data partitioning and shuffling
- External CSV, Avro, Parquet, BigQuery, and TFRecord ingestion

Component: **ExampleGen**

Inputs and Outputs



Configuration

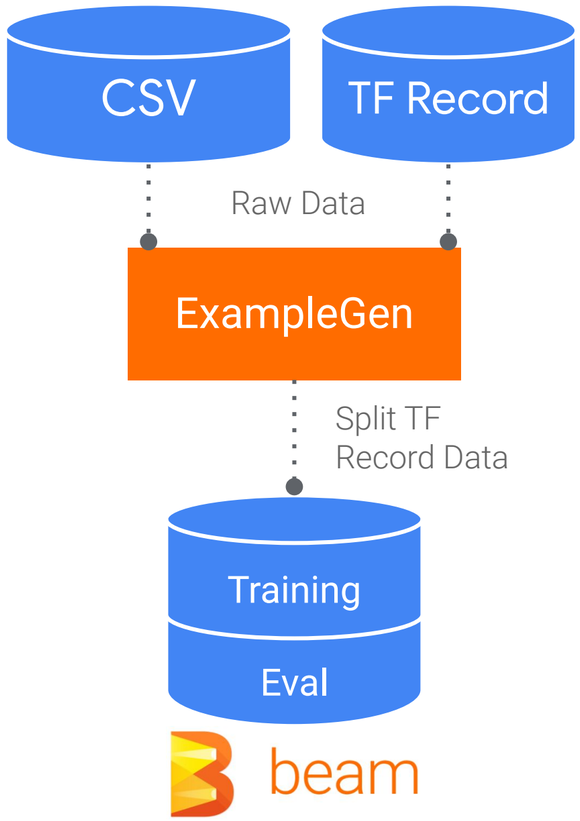
```
output_config = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=4),
        example_gen_pb2.SplitConfig.Split(name='dev', hash_buckets=1)
    ]))

example_gen = tfx.components.CsvExampleGen(
    instance_name='Data_Extraction_Spliting',
    input=external_input(DATA_ROOT),
    output_config=output_config
)
```

ML project lifecycle benefits

- Configurable and reproducible data partitioning and shuffling
- External CSV, Avro, Parquet, BigQuery, and TFRecord ingestion
- Apache Beam supported for scalable data ingestion

Component: **ExampleGen**

Inputs and Outputs



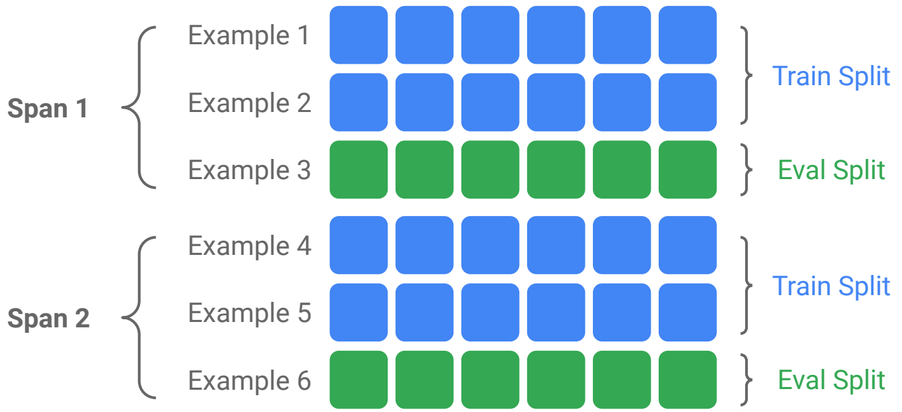
Configuration

```
output_config = example_gen_pb2.Output(
    split_config=example_gen_pb2.SplitConfig(splits=[
        example_gen_pb2.SplitConfig.Split(name='train', hash_buckets=4),
        example_gen_pb2.SplitConfig.Split(name='dev', hash_buckets=1)
    ]))

example_gen = tfx.components.CsvExampleGen(
    instance_name='Data_Extraction_Spliting',
    input=external_input(DATA_ROOT),
    output_config=output_config
)
```

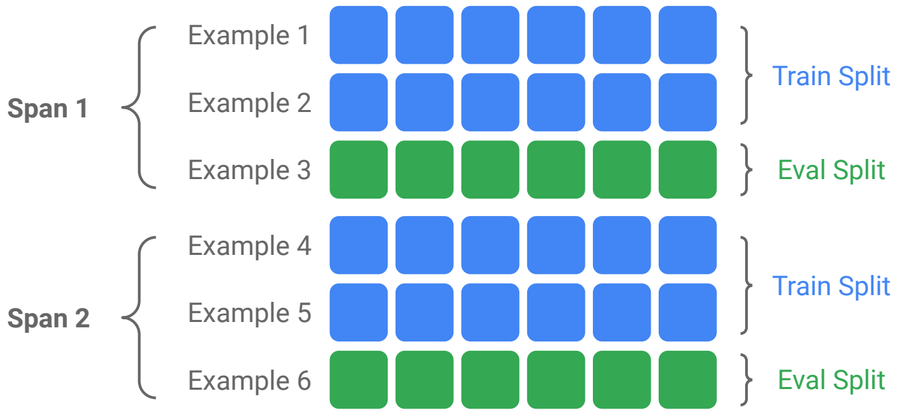
ML project lifecycle benefits

- Configurable and reproducible data partitioning and shuffling
- External CSV, Avro, Parquet, BigQuery, and TFRecord ingestion
- Apache Beam supported for scalable data ingestion
- Customizable to new input data formats and ingestion methods

ExampleGen supports MLOps data management capabilities

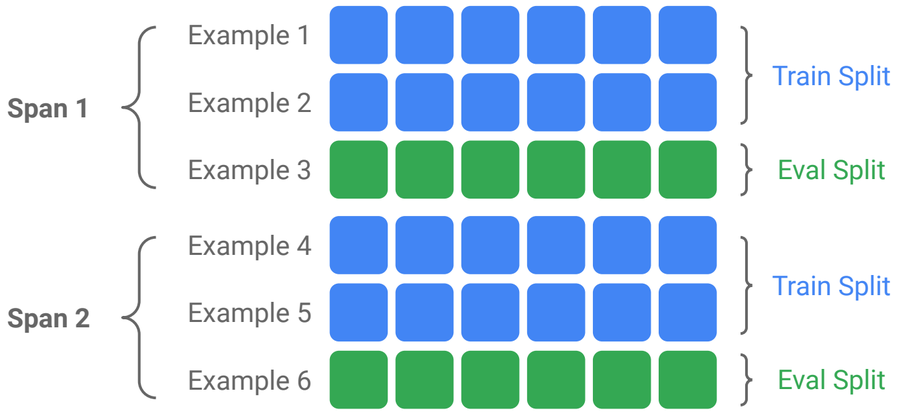


ExampleGen supports MLOps data management capabilities




```
from  tfx.proto import example_gen_pb2
from  tfx.proto import range_config_pb2

input = example_gen_pb2.Input(splits=[
                example_gen_pb2.Input.Split(name='train',
                                            pattern='span-{SPAN:2}/train/*'),
                example_gen_pb2.Input.Split(name='eval',
                                            pattern='span-{SPAN:2}/eval/*')])

range = range_config_pb2.RangeConfig(
                static_range=range_config_pb2.StaticRange(
                        start_span_number=1, end_span_number=1))

examples = csv_input(input_dir)
example_gen = CsvExampleGen(input=examples, input_config=input,
                            range_config=range)
```

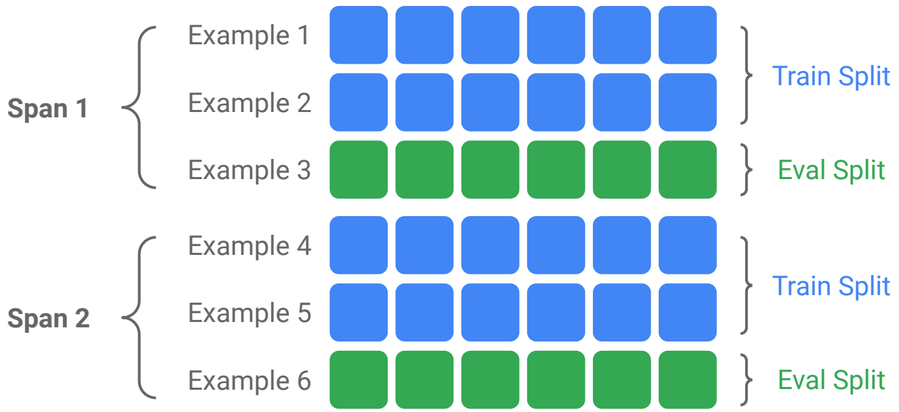
ExampleGen supports MLOps data management capabilities




```
from  tfx.proto import example_gen_pb2
from  tfx.proto import range_config_pb2

input = example_gen_pb2.Input(splits=[
                example_gen_pb2.Input.Split(name='train',
                                            pattern='span-{SPAN:2}/train/*'),
                example_gen_pb2.Input.Split(name='eval',
                                            pattern='span-{SPAN:2}/eval/*')])

range = range_config_pb2.RangeConfig(
                static_range=range_config_pb2.StaticRange(
                        start_span_number=1, end_span_number=1))

examples = csv_input(input_dir)
example_gen = CsvExampleGen(input=examples, input_config=input,
                            range_config=range)
```

Customize data partitioning, spans, versioning, and raw data ingestion method through ExampleGen Input and Output Configs.

ExampleGen supports MLOps data management capabilities

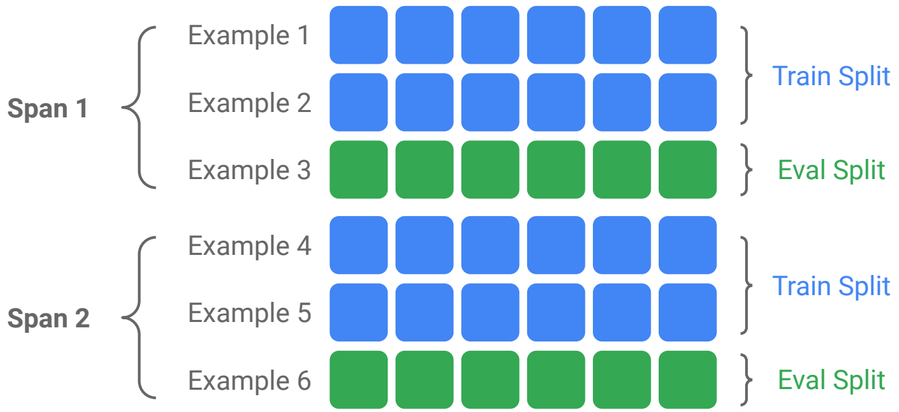



```
from  tfx.proto import example_gen_pb2
from  tfx.proto import range_config_pb2

input = example_gen_pb2.Input(splits=[
                example_gen_pb2.Input.Split(name='train',
                                            pattern='span-{SPAN:2}/train/*'),
                example_gen_pb2.Input.Split(name='eval',
                                            pattern='span-{SPAN:2}/eval/*')])

range = range_config_pb2.RangeConfig(
                static_range=range_config_pb2.StaticRange(
                        start_span_number=1, end_span_number=1))

examples = csv_input(input_dir)
example_gen = CsvExampleGen(input=examples, input_config=input,
                            range_config=range)
```

Customize data partitioning, spans, versioning, and raw data ingestion method through ExampleGen Input and Output Configs.

Customize pipeline data spans with the Range Config.

ExampleGen supports MLOps data management capabilities




```
from  tfx.proto import example_gen_pb2
from  tfx.proto import range_config_pb2

input = example_gen_pb2.Input(splits=[
                example_gen_pb2.Input.Split(name='train',
                                            pattern='span-{SPAN:2}/train/*'),
                example_gen_pb2.Input.Split(name='eval',
                                            pattern='span-{SPAN:2}/eval/*')])

range = range_config_pb2.RangeConfig(
                static_range=range_config_pb2.StaticRange(
                        start_span_number=1, end_span_number=1))

examples = csv_input(input_dir)
example_gen = CsvExampleGen(input=examples, input_config=input,
                            range_config=range)
```

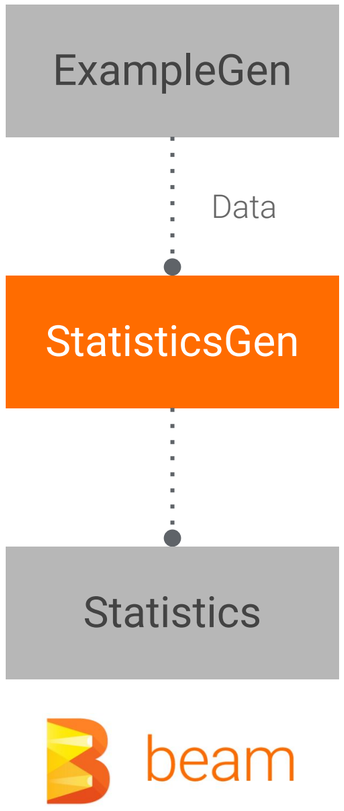
Customize data partitioning, spans, versioning, and raw data ingestion method through ExampleGen Input and Output Configs.

Customize pipeline data spans with the Range Config.

Support for external CSV, Avro, Parquet, BigQuery, and TFRecords.

Component: **StatisticsGen**

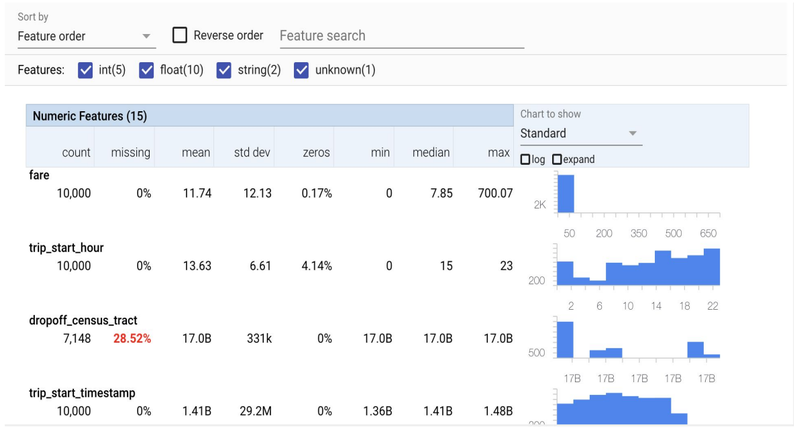
Inputs and Outputs



Configuration

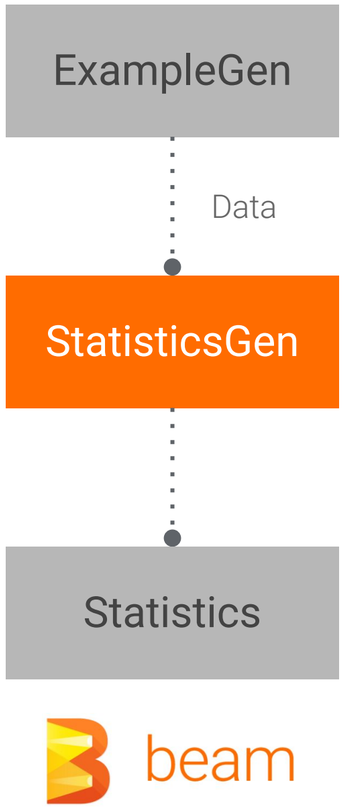
```
statistics_gen = tfx.components.StatisticsGen(
    instance_name='Statistics_Generation',
    examples=example_gen.outputs['examples'])
```

TFDV Feature Statistics Visualization



Component: **StatisticsGen**

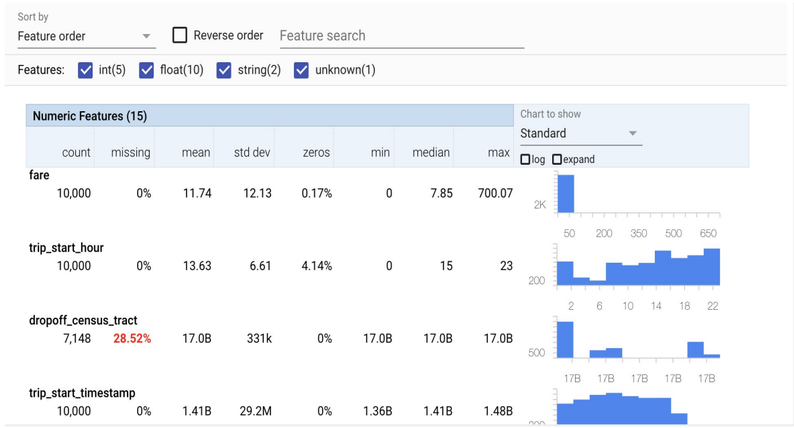
Inputs and Outputs



Configuration

```
statistics_gen = tfx.components.StatisticsGen(
    instance_name='Statistics_Generation',
    examples=example_gen.outputs['examples'])
```

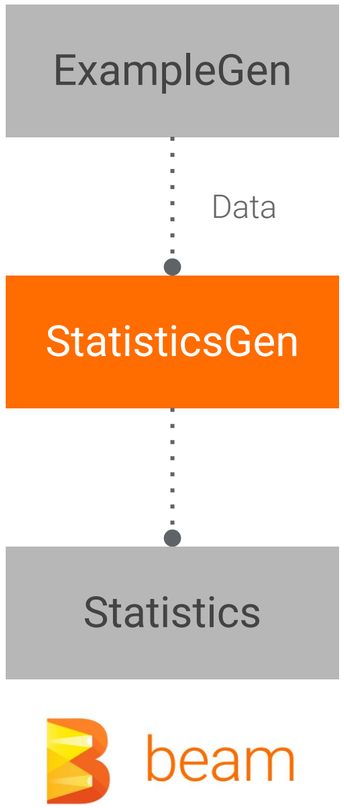
TFDV Feature Statistics Visualization



ML project lifecycle benefits

Component: **StatisticsGen**

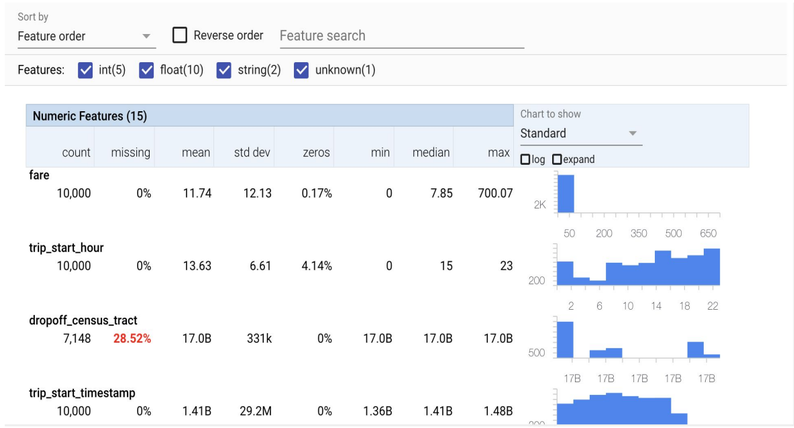
Inputs and Outputs



Configuration

```
statistics_gen = tfx.components.StatisticsGen(
    instance_name='Statistics_Generation',
    examples=example_gen.outputs['examples'])
```

TFDV Feature Statistics Visualization

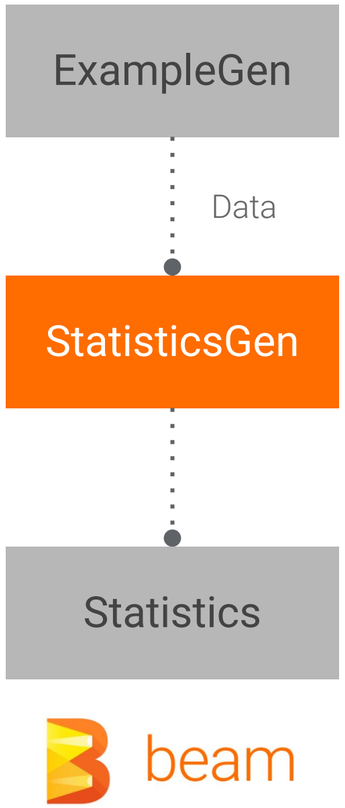


ML project lifecycle benefits

- TensorFlow Data Validation (TFDV) library for calculating feature statistics

Component: **StatisticsGen**

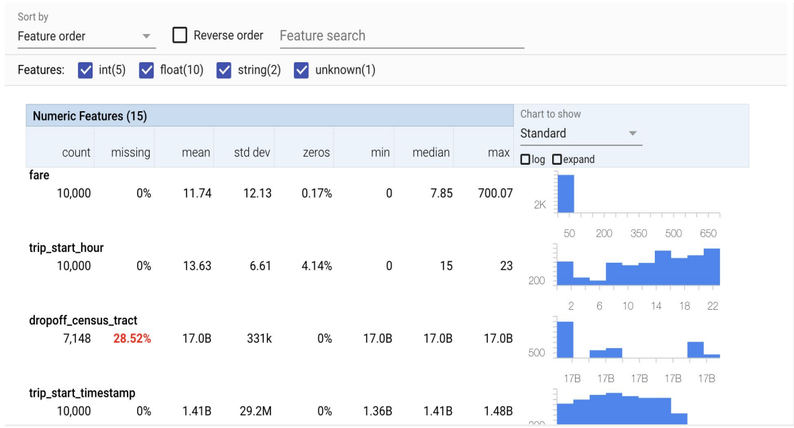
Inputs and Outputs



Configuration

```
statistics_gen = tfx.components.StatisticsGen(
    instance_name='Statistics_Generation',
    examples=example_gen.outputs['examples'])
```

TFDV Feature Statistics Visualization

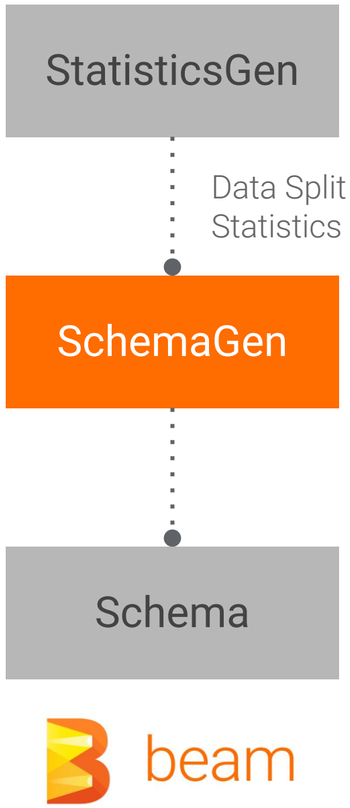


ML project lifecycle benefits

- TensorFlow Data Validation (TFDV) library for calculating feature statistics
- Scalable full-pass dataset feature statistics processing with Apache Beam

Component: **SchemaGen**

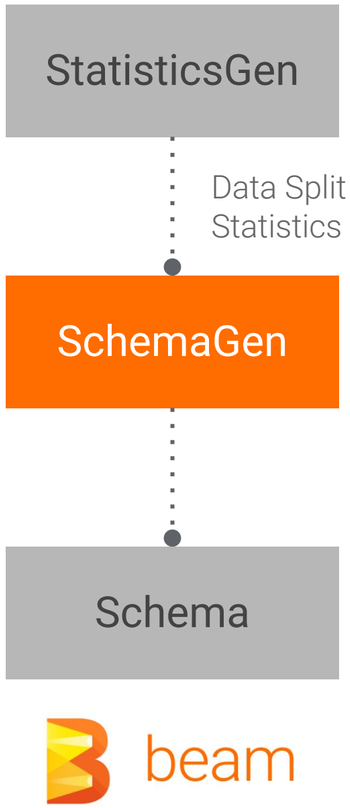
Inputs and Outputs



beam

Component: **SchemaGen**

Inputs and Outputs

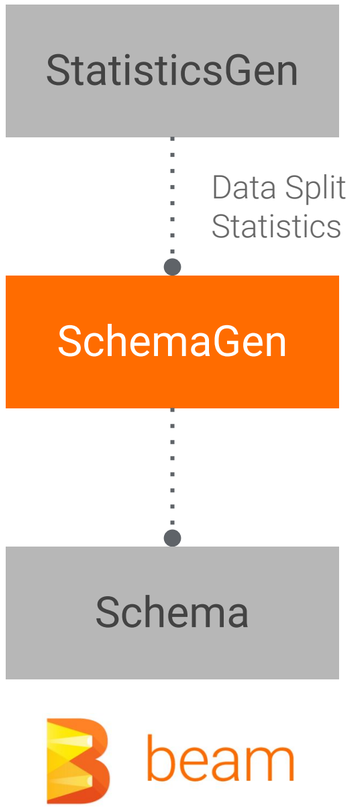


Configuration

```
schema_gen = SchemaGen(
    statistics=statistics_gen.outputs['statistics'],
    infer_feature_shape=False)
```

Component: **SchemaGen**

Inputs and Outputs



beam

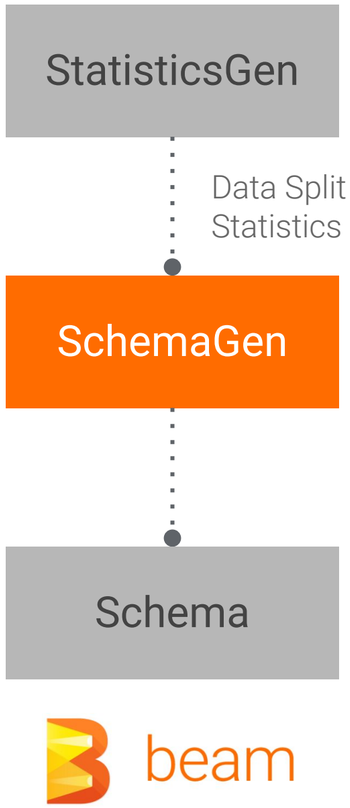
Configuration

```
schema_gen = SchemaGen(
    statistics=statistics_gen.outputs['statistics'],
    infer_feature_shape=False)
```

Feature name	Type	Presence	Valency	Domain
'fare'	FLOAT	required	single	-
'trip_start_hour'	INT	required	single	-
'pickup_census_tract'	BYTES	optional		-
'dropoff_census_tract'	FLOAT	optional	single	-
'company'	STRING	optional	single	'company'

Component: **SchemaGen**

Inputs and Outputs



beam

Configuration

```
schema_gen = SchemaGen(
    statistics=statistics_gen.outputs['statistics'],
    infer_feature_shape=False)
```

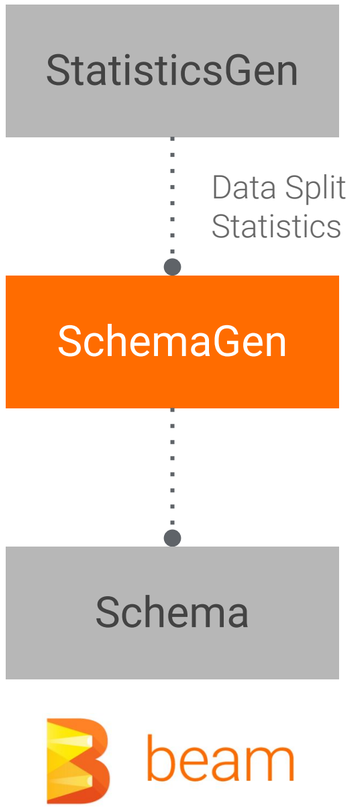
Feature name	Type	Presence	Valency	Domain
'fare'	FLOAT	required	single	-
'trip_start_hour'	INT	required	single	-
'pickup_census_tract'	BYTES	optional		-
'dropoff_census_tract'	FLOAT	optional	single	-
'company'	STRING	optional	single	'company'

ML project lifecycle benefits

- TensorFlow Data Validation (TFDV) creates common data description for pipeline components

Component: **SchemaGen**

Inputs and Outputs



beam

Configuration

```
schema_gen = SchemaGen(
    statistics=statistics_gen.outputs['statistics'],
    infer_feature_shape=False)
```

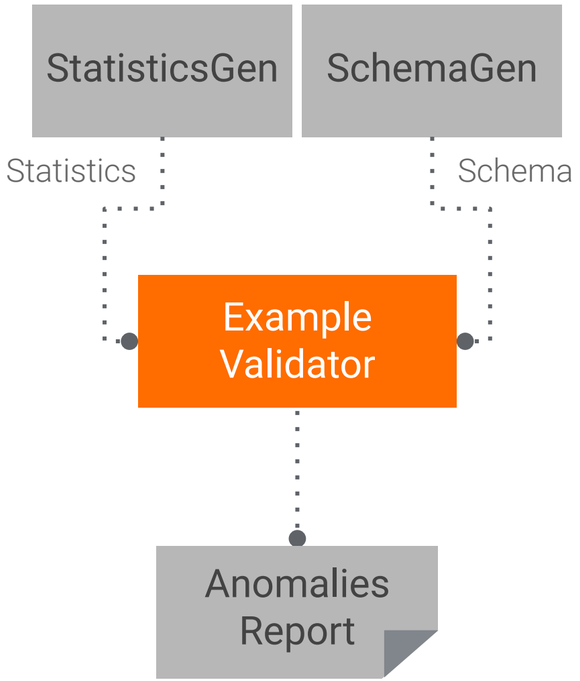
Feature name	Type	Presence	Valency	Domain
'fare'	FLOAT	required	single	-
'trip_start_hour'	INT	required	single	-
'pickup_census_tract'	BYTES	optional		-
'dropoff_census_tract'	FLOAT	optional	single	-
'company'	STRING	optional	single	'company'

ML project lifecycle benefits

- TensorFlow Data Validation (TFDV) creates common data description for pipeline components
- A Schema enables continuous data monitoring and validation during pipeline training

Component: **ExampleValidator**

Inputs and Outputs



Configuration

```
example_validator = ExampleValidator(
    statistics=statistics_gen.outputs['statistics'],
    schema=schema_importer.outputs['result'],
    instance_name="Data_Validation"
)
```

Visualization

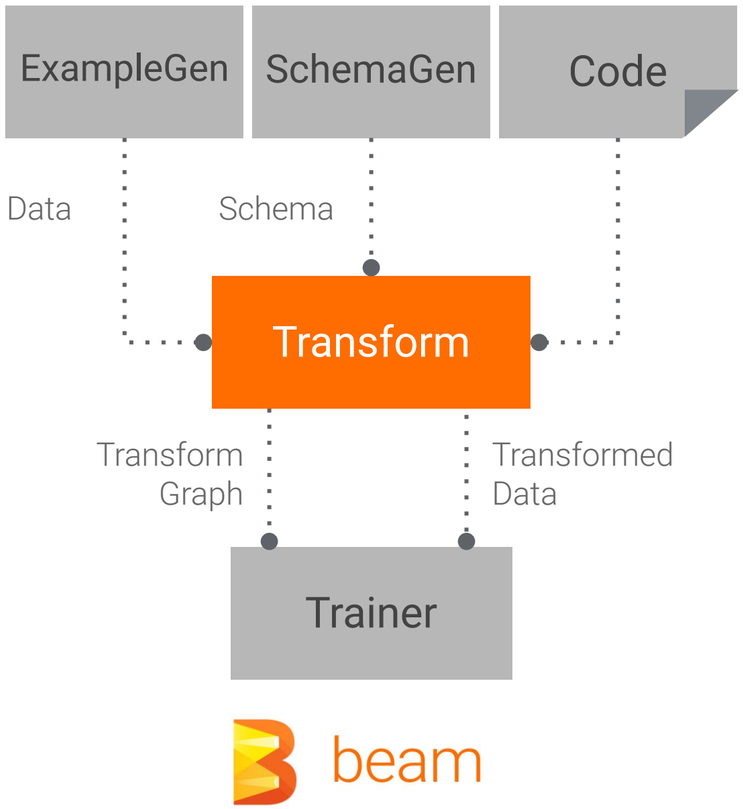
Feature name	Anomaly short description	Anomaly long description
'payment_type'	Unexpected string values	Examples contain values missing from the schema: Prcard (<1%).
'company'	Unexpected string values	Examples contain values missing from the schema: 2092 - 61288 Sbeih company (<1%), 2192 - 73487 Zeymane Corp (<1%), 2192 - Zeymane Corp (<1%), 2823 - 73307 Seung Lee (<1%), 3094 - 24059 G.L.B. Cab Co (<1%), 3319 - CD Cab Co (<1%), 3385 - Eman Cab (<1%), 3897 - 57856 Ilie Malec (<1%), 4053 - 40193 Adwar H. Nikola (<1%), 4197 - Royal Star (<1%), 585 - 88805 Valley Cab Co (<1%), 5874 - Sergey Cab Corp. (<1%), 6057 - 24657 Richard Addo (<1%), 6574 - Babylon Express Inc. (<1%), 6742 - 83735 Tasha ride inc (<1%).

ML project lifecycle benefits

- Data monitoring for detection of anomalies, train-serving skew, and data drift

Component: **Transform**

Inputs and Outputs

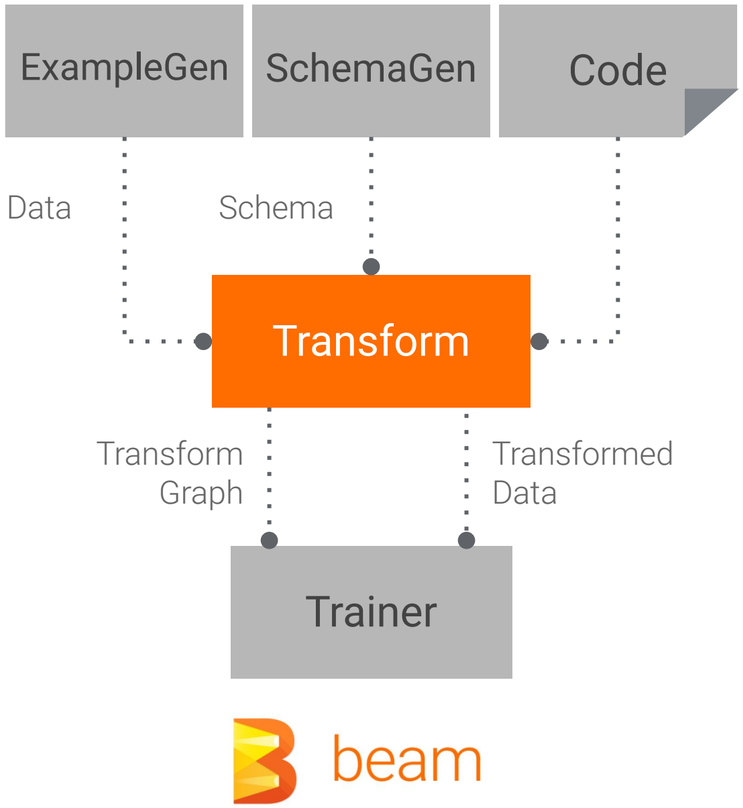


Configuration

```
transform = Transform(
    examples=example_gen.outputs['examples'],
    schema=schema_importer.outputs['result'],
    module_file=TRANSFORM_MODULE)
```

Component: **Transform**

Inputs and Outputs



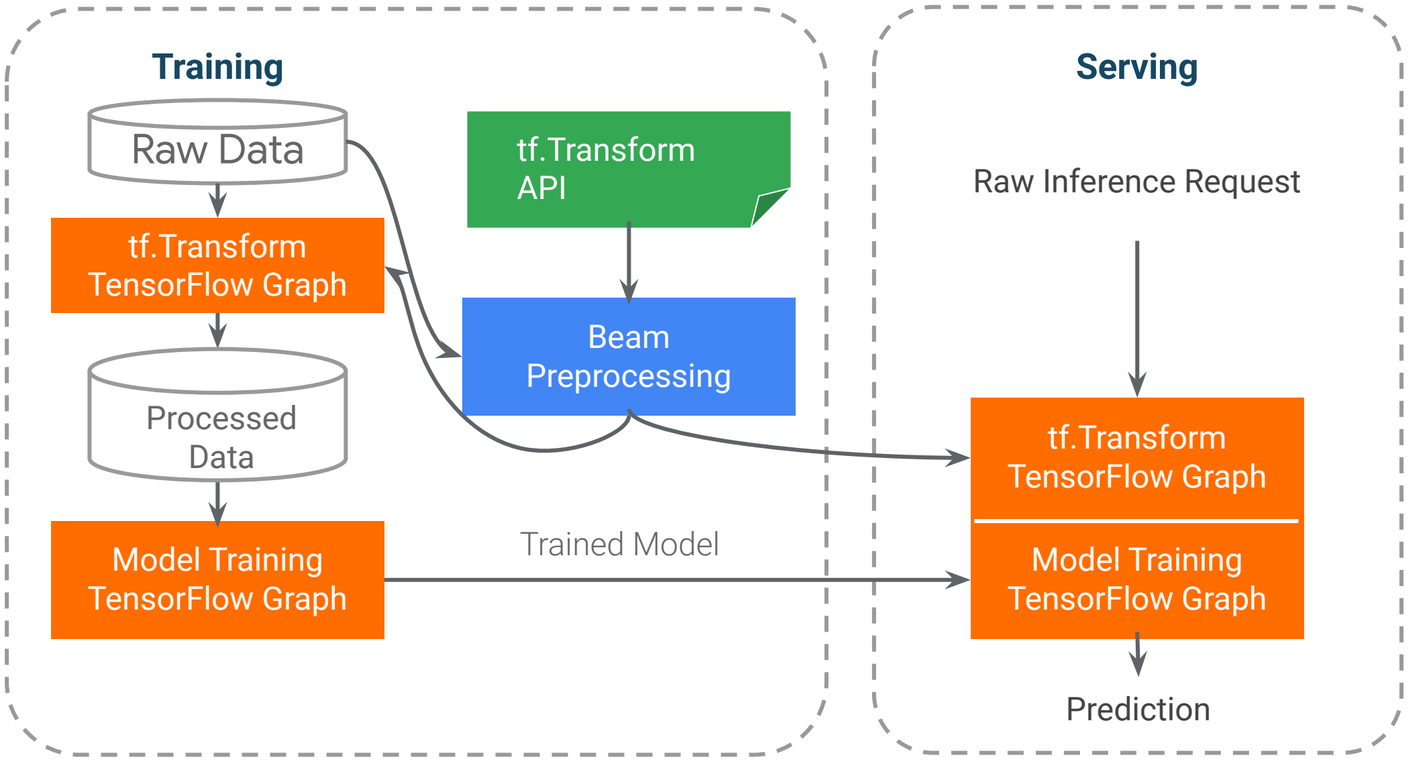
Configuration

```
transform = Transform(
    examples=example_gen.outputs['examples'],
    schema=schema_importer.outputs['result'],
    module_file=TRANSFORM_MODULE)
```

Tensorflow Transform (TFT) library code in pipeline/preprocessing.py

```
def preprocessing_fn(inputs):

  outputs = {}

  # Scale numerical features
  for key in features.NUMERIC_FEATURE_KEYS:
    outputs[features.transformed_name(key)] = tft.scale_to_z_score(
        _fill_in_missing(inputs[key]))

  # Generate vocabularies and maps categorical features
  for key in features.CATEGORICAL_FEATURE_KEYS:
    outputs[features.transformed_name(key)] = tft.compute_and_apply_vocabulary(
        x=_fill_in_missing(inputs[key]), num_oov_buckets=1, vocab_filename=key)

  outputs[features.transformed_name(features.LABEL_KEY)] = _fill_in_missing(
      inputs[features.LABEL_KEY])

  return outputs
```

Component: **Transform**

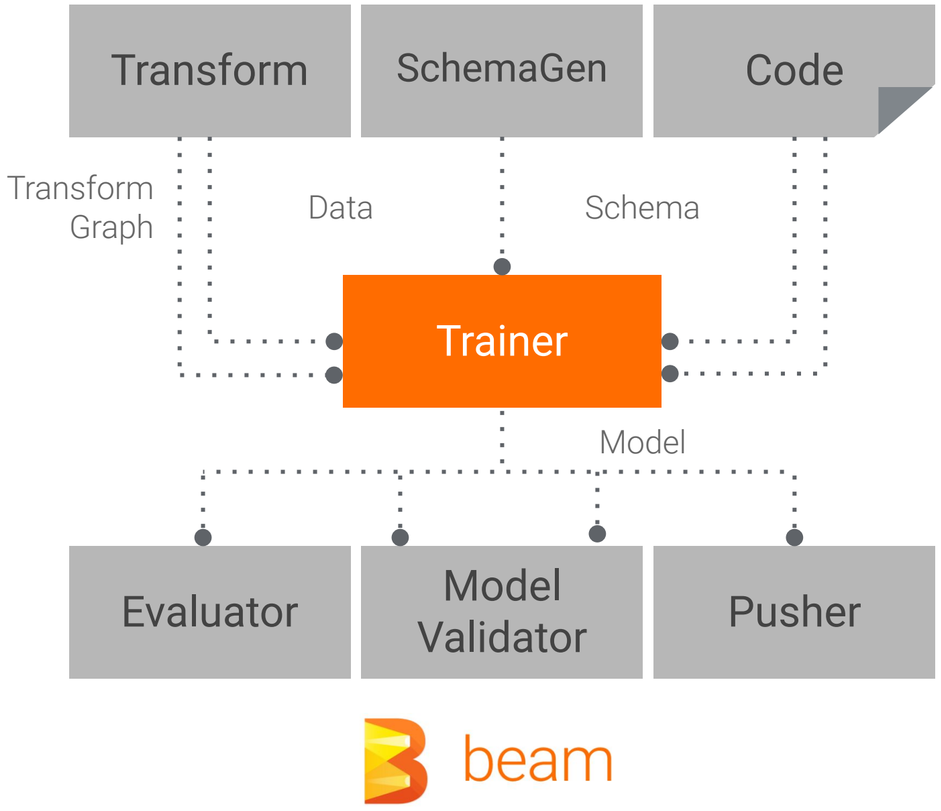


ML project lifecycle benefits

- Generates SavedModel for consistent training and serving feature engineering
- Backed by Apache Beam for scalable distributed data processing to large datasets

Component: **Trainer**

Inputs and Outputs



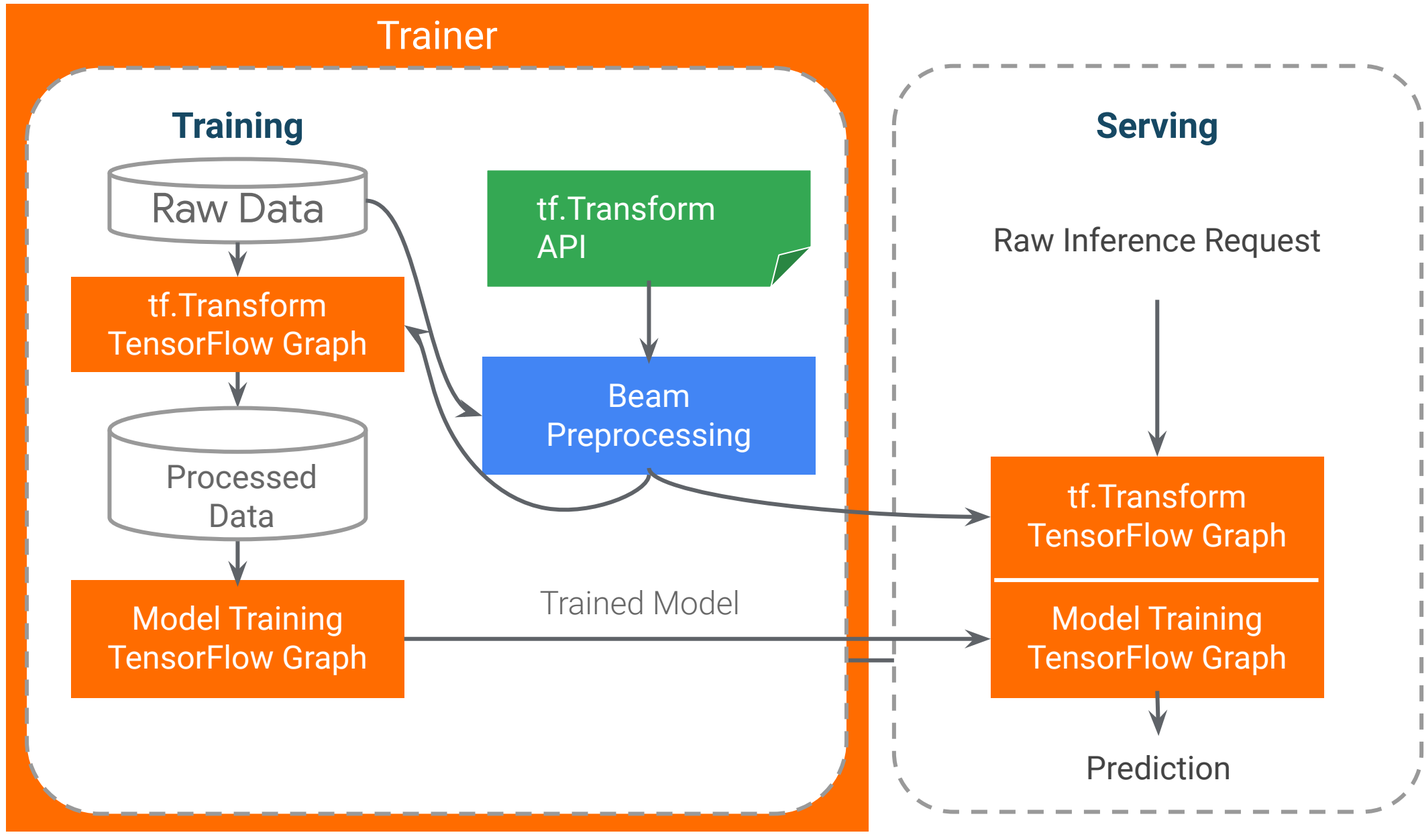
beam

Configuration

```
trainer = Trainer(
custom_executor_spec=executor_spec.ExecutorClassSpec(trainer_executor.Gen
ericExecutor),
    module_file=model.py,
    transformed_examples=transform.outputs["transformed_examples"],
    schema=schema_importer.outputs["result"],
    transform_graph=transform.outputs["transform_graph"],
    train_args=trainer_pb2.TrainArgs(num_steps=5000),
    eval_args=trainer_pb2.EvalArgs(num_steps=1000))
```

Code -> your TensorFlow code in a pipeline/models/model.py

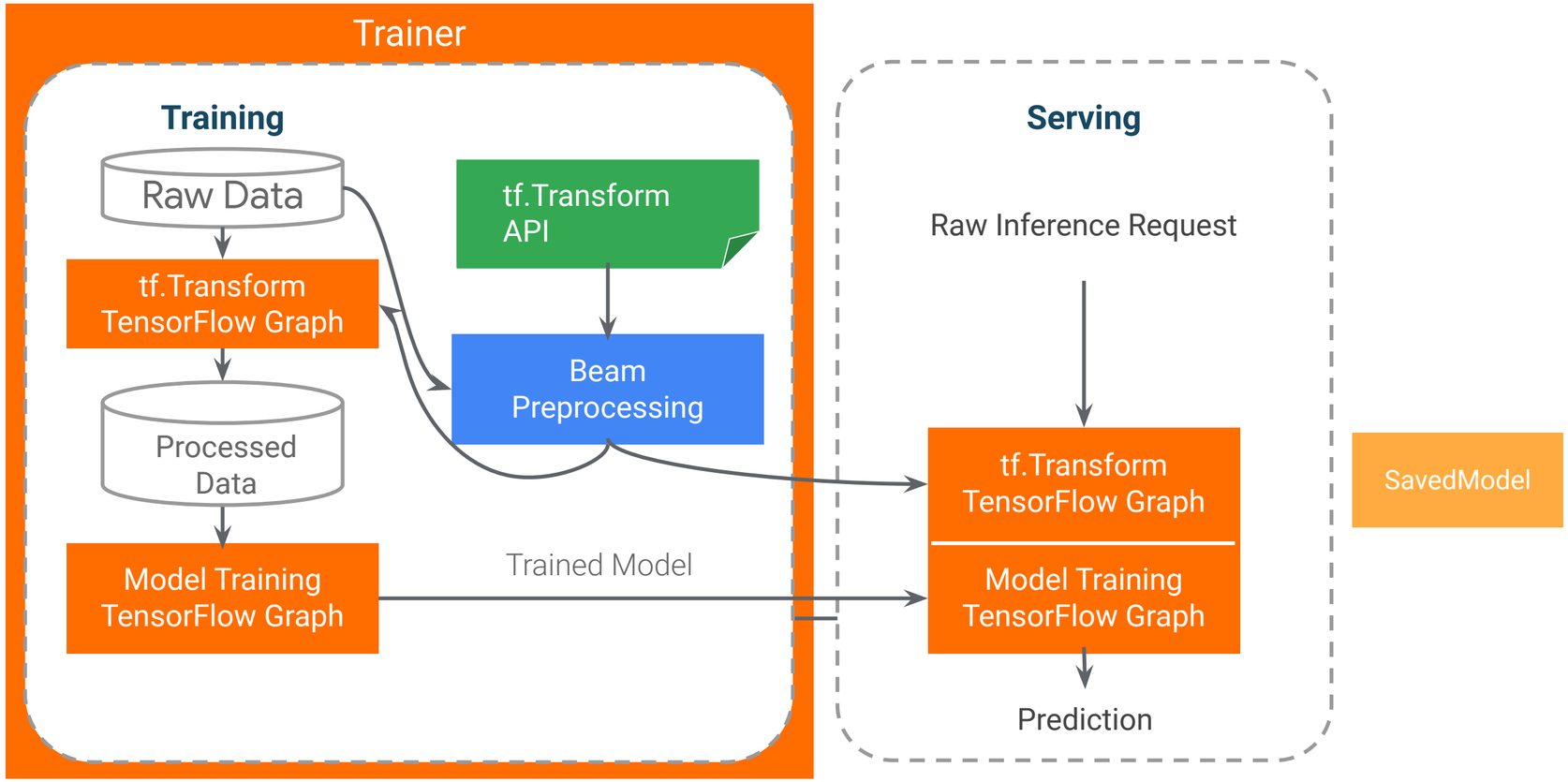
Component: **Trainer**



ML project lifecycle benefits

- Produces standardized TensorFlow SavedModel model artifact for sharing and easier deployment
- Configurable with Generic Trainer for any TensorFlow model API such Estimator, tf.Keras, and TFLite

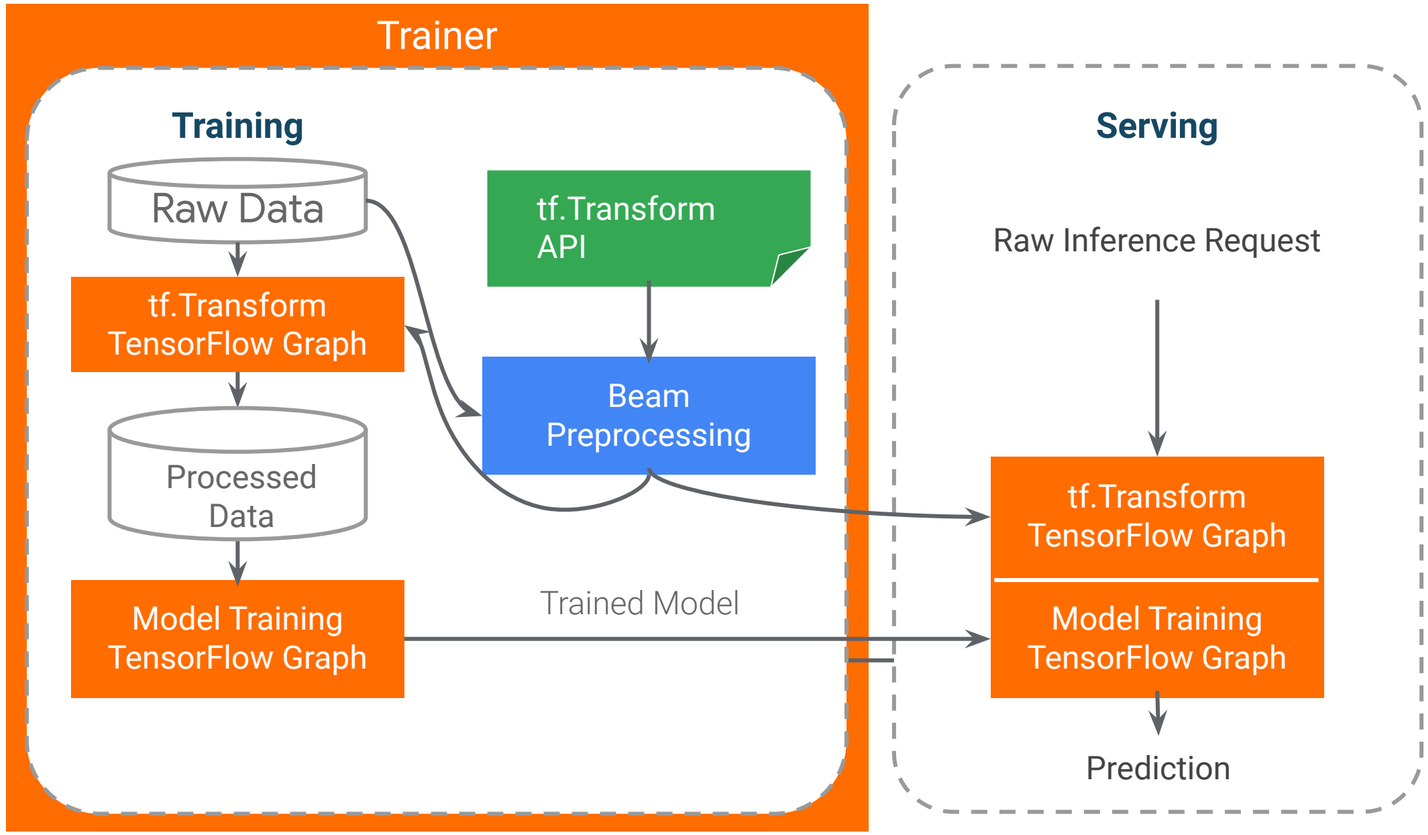
Component: **Trainer**



ML project lifecycle benefits

- Produces standardized TensorFlow SavedModel model artifact for sharing and easier deployment
- Configurable with Generic Trainer for any TensorFlow model API such Estimator, tf.Keras, and TFLite

Component: **Trainer**



Trainer outputs a trained **SavedModel** and **EvalSavedModel**

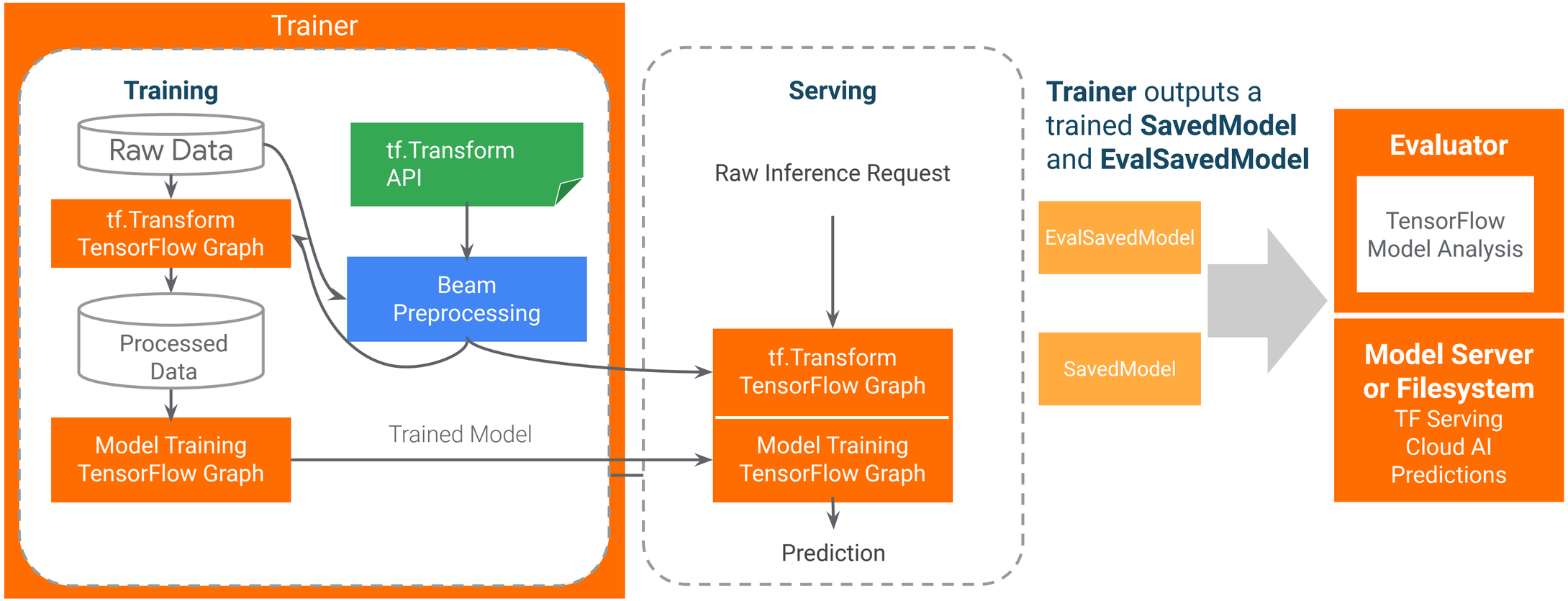
EvalSavedModel

SavedModel

ML project lifecycle benefits

- Produces standardized TensorFlow SavedModel model artifact for sharing and easier deployment
- Configurable with Generic Trainer for any TensorFlow model API such Estimator, tf.Keras, and TFLite

Component: **Trainer**

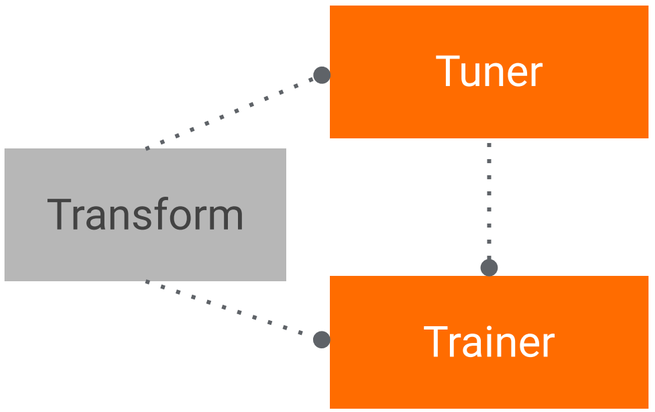


ML project lifecycle benefits

- Produces standardized TensorFlow SavedModel model artifact for sharing and easier deployment
- Configurable with Generic Trainer for any TensorFlow model API such Estimator, tf.Keras, and TFLite

Component: **Tuner**

Tuner works with Trainer

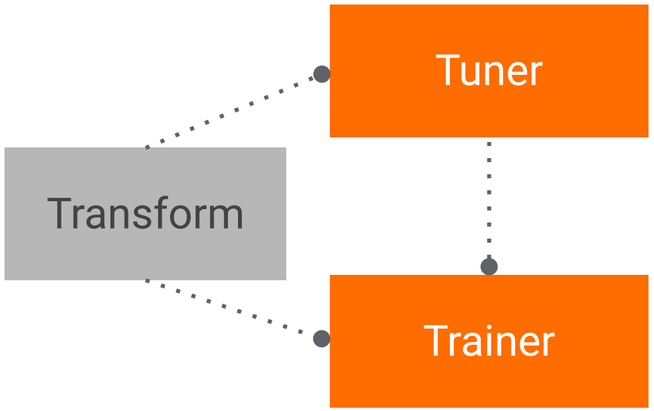


Configuration

```
tuner = Tuner(
    module_file=module_file,  # Contains `tuner_fn`.
    examples=transform.outputs['transformed_examples'],
    transform_graph=transform.outputs['transform_graph'],
    train_args=trainer_pb2.TrainArgs(num_steps=20),
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

Component: **Tuner**

Tuner works with Trainer



Configuration

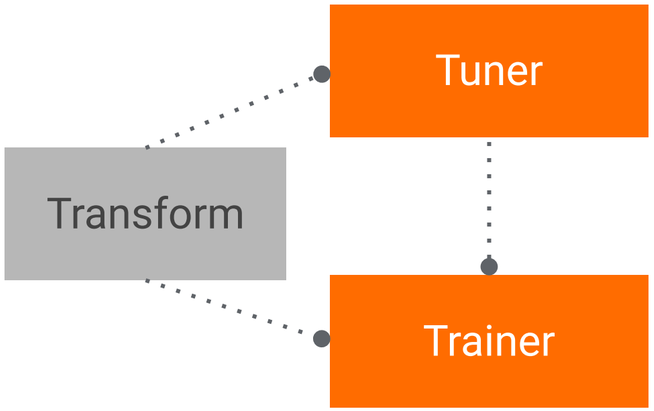
```
tuner = Tuner(
    module_file=module_file,  # Contains `tuner_fn`.
    examples=transform.outputs['transformed_examples'],
    transform_graph=transform.outputs['transform_graph'],
    train_args=trainer_pb2.TrainArgs(num_steps=20),
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

Best hyperparameters added to Trainer in pipeline

```
trainer = Trainer(
    module_file=module_file,  # Contains `run_fn`.
    custom_executor_spec=executor_spec.
        ExecutorClassSpec(GenericExecutor),
    examples=transform.outputs['transformed_examples'],
    transform_graph=transform.outputs['transform_graph'],
    schema=schema_gen.outputs['schema'],
    # This will be passed to `run_fn`.
    hyperparameters=tuner.outputs['best_hyperparameters'],
    train_args=trainer_pb2.TrainArgs(num_steps=100),
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

Component: **Tuner**

Tuner works with Trainer



Configuration

```
tuner = Tuner(
    module_file=module_file,  # Contains `tuner_fn`.
    examples=transform.outputs['transformed_examples'],
    transform_graph=transform.outputs['transform_graph'],
    train_args=trainer_pb2.TrainArgs(num_steps=20),
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

Best hyperparameters added to Trainer in pipeline

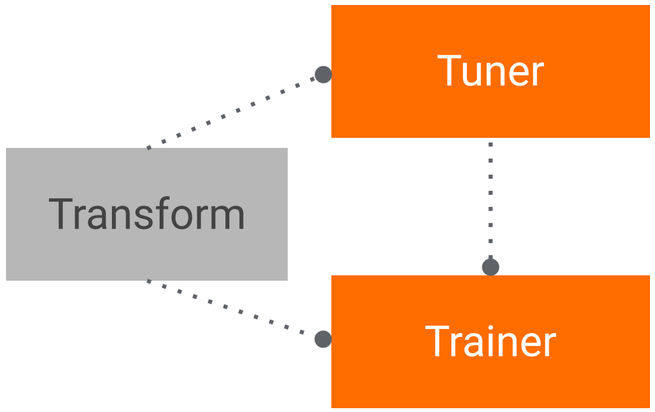
```
trainer = Trainer(
    module_file=module_file,  # Contains `run_fn`.
    custom_executor_spec=executor_spec.
        ExecutorClassSpec(GenericExecutor),
    examples=transform.outputs['transformed_examples'],
    transform_graph=transform.outputs['transform_graph'],
    schema=schema_gen.outputs['schema'],
    # This will be passed to `run_fn`.
    hyperparameters=tuner.outputs['best_hyperparameters'],
    train_args=trainer_pb2.TrainArgs(num_steps=100),
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

ML project lifecycle benefits

- Supports KerasTuner library for hyperparameter tuning and integration with TensorFlow Trainer

Component: **Tuner**

Tuner works with Trainer



Configuration

```
tuner = Tuner(
    module_file=module_file,  # Contains `tuner_fn`.
    examples=transform.outputs['transformed_examples'],
    transform_graph=transform.outputs['transform_graph'],
    train_args=trainer_pb2.TrainArgs(num_steps=20),
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

Best hyperparameters added to Trainer in pipeline

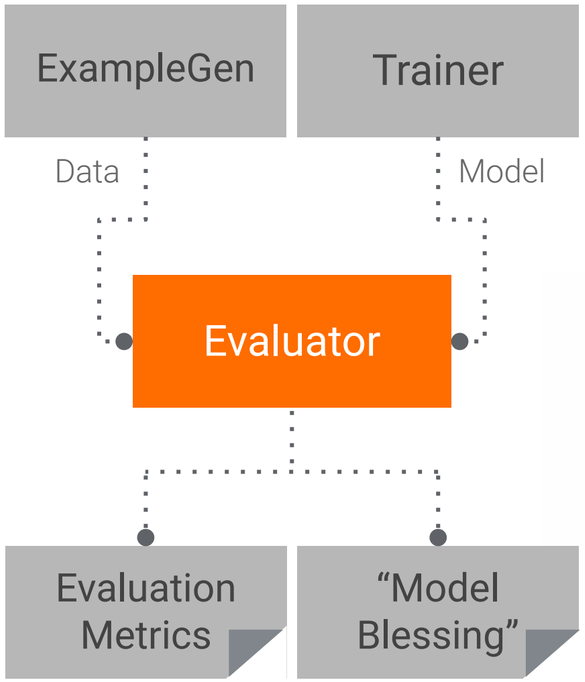
```
trainer = Trainer(
    module_file=module_file,  # Contains `run_fn`.
    custom_executor_spec=executor_spec.
        ExecutorClassSpec(GenericExecutor),
    examples=transform.outputs['transformed_examples'],
    transform_graph=transform.outputs['transform_graph'],
    schema=schema_gen.outputs['schema'],
    # This will be passed to `run_fn`.
    hyperparameters=tuner.outputs['best_hyperparameters'],
    train_args=trainer_pb2.TrainArgs(num_steps=100),
    eval_args=trainer_pb2.EvalArgs(num_steps=5))
```

ML project lifecycle benefits

- Supports KerasTuner library for hyperparameter tuning and integration with TensorFlow Trainer
- Can use Google Cloud AI Platform Optimizer for distributed tuning

Component: **Evaluator**

Inputs and Outputs



beam

Configuration

```
model_analyzer = Evaluator(
    examples=example_gen.outputs.examples,
    model=trainer.outputs.model,
    baseline_model=model_resolver.outputs.model,
    eval_config=eval_config)
```

TensorFlow Model Analysis (TFMA) library for visualizing model evaluation

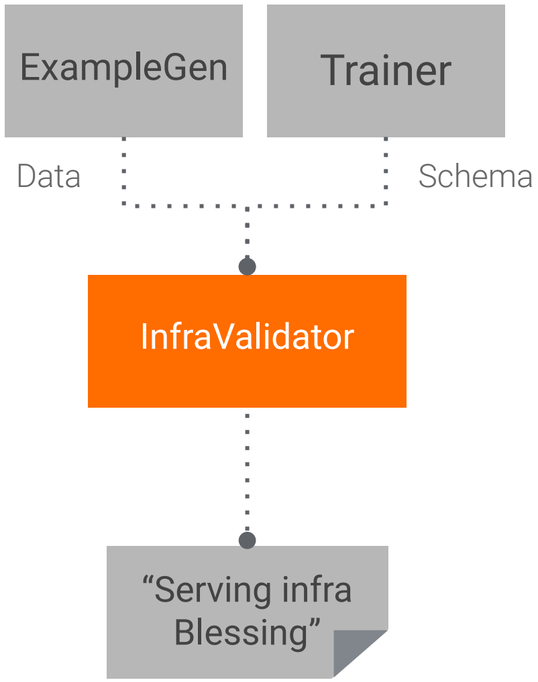
feature	accuracy	accuracy_baseline	auc	auc_precision_recall	average_loss
trip_start_hour:19	0.63582	0.59104	0.64311	0.56092	0.64626
trip_start_hour:14	0.67117	0.65766	0.63793	0.49112	0.61667
trip_start_hour:2	0.66102	0.63559	0.58527	0.47002	0.65236
trip_start_hour:12	0.69643	0.65625	0.68270	0.54122	0.59538
trip_start_hour:0	0.66184	0.66667	0.63773	0.45081	0.61634
trip_start_hour:23	0.65625	0.64844	0.58357	0.43514	0.64315

ML project lifecycle benefits

- Uses TensorFlow Model Analysis (TFMA) and Apache Beam for scalable model evaluation across data splits and slices.
- Validate model performance “good enough” to be pushed to production

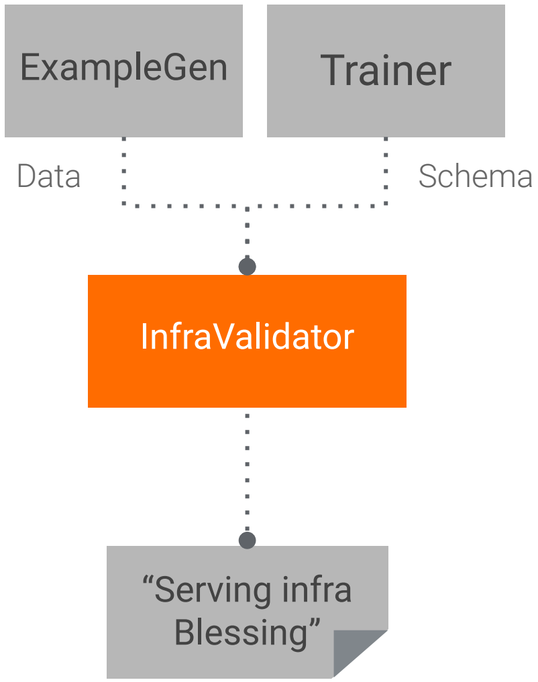
Component: **InfraValidator**

Inputs and Outputs



Component: **InfraValidator**

Inputs and Outputs

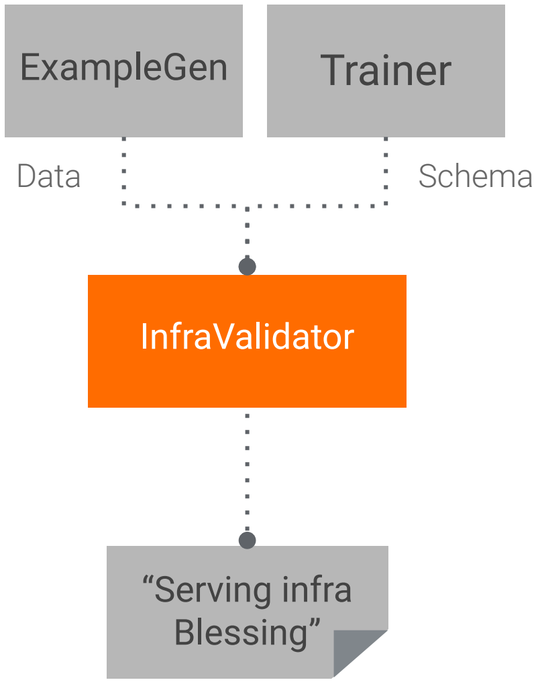


Configuration

```
infra_validator = InfraValidator(
    model=trainer.outputs['model'],
    examples=example_gen.outputs['examples'],
    serving_spec=infra_validator_pb2.ServingSpec(...),
  validation_spec=infra_validator_pb2.ValidationSpec(...),
  request_spec=infra_validator_pb2.RequestSpec(...)
)
```

Component: **InfraValidator**

Inputs and Outputs



Configuration

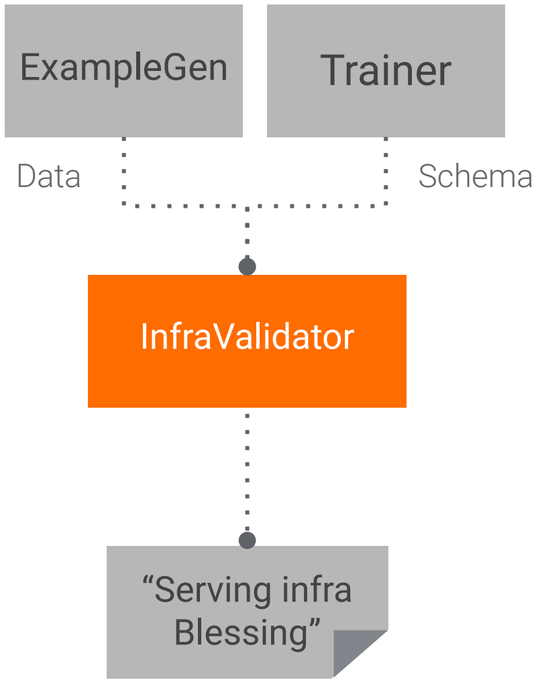
```
infra_validator = InfraValidator(
    model=trainer.outputs['model'],
    examples=example_gen.outputs['examples'],
    serving_spec=infra_validator_pb2.ServingSpec(...),
  validation_spec=infra_validator_pb2.ValidationSpec(...),
  request_spec=infra_validator_pb2.RequestSpec(...)
)
```

Configuration options

ServingSpec: Type of model server and infrastructure to test with, such as TF Serving

Component: **InfraValidator**

Inputs and Outputs



Configuration

```
infra_validator = InfraValidator(
    model=trainer.outputs['model'],
    examples=example_gen.outputs['examples'],
    serving_spec=infra_validator_pb2.ServingSpec(...),
  validation_spec=infra_validator_pb2.ValidationSpec(...),
  request_spec=infra_validator_pb2.RequestSpec(...)
)
```

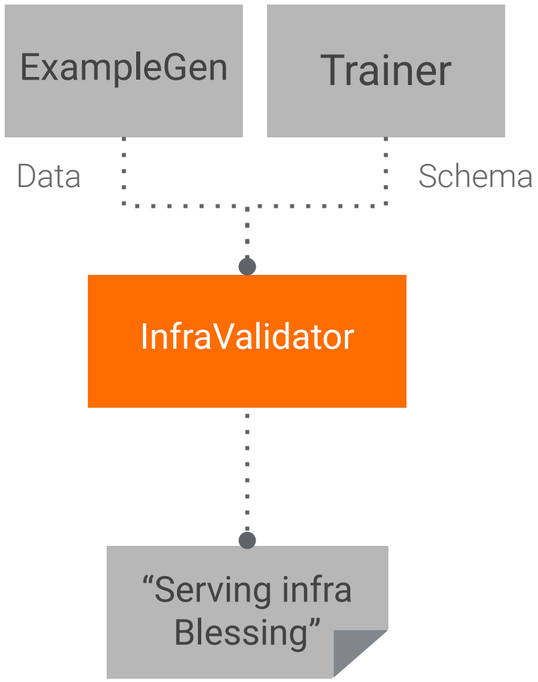
Configuration options

ServingSpec: Type of model server and infrastructure to test with, such as TF Serving

ValidationSpec: Adjusts the infra validation criteria or workflow

Component: **InfraValidator**

Inputs and Outputs



Configuration

```
infra_validator = InfraValidator(
    model=trainer.outputs['model'],
    examples=example_gen.outputs['examples'],
    serving_spec=infra_validator_pb2.ServingSpec(...),
  validation_spec=infra_validator_pb2.ValidationSpec(...),
  request_spec=infra_validator_pb2.RequestSpec(...)
)
```

Configuration options

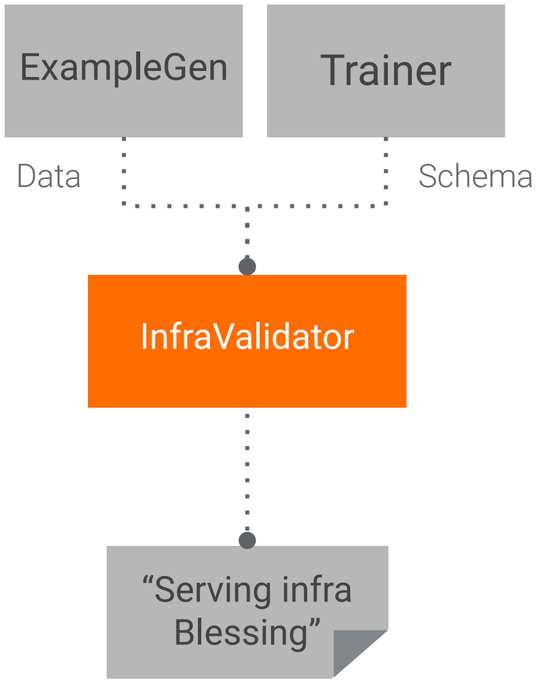
ServingSpec: Type of model server and infrastructure to test with, such as TF Serving

ValidationSpec: Adjusts the infra validation criteria or workflow

RequestSpec: Which model signature, how many examples to test

Component: **InfraValidator**

Inputs and Outputs



Configuration

```
infra_validator = InfraValidator(
    model=trainer.outputs['model'],
    examples=example_gen.outputs['examples'],
    serving_spec=infra_validator_pb2.ServingSpec(...),
  validation_spec=infra_validator_pb2.ValidationSpec(...),
  request_spec=infra_validator_pb2.RequestSpec(...)
)
```

Configuration options

ServingSpec: Type of model server and infrastructure to test with e.g. TF Serving

ValidationSpec: Adjusts the infra validation criteria or workflow.

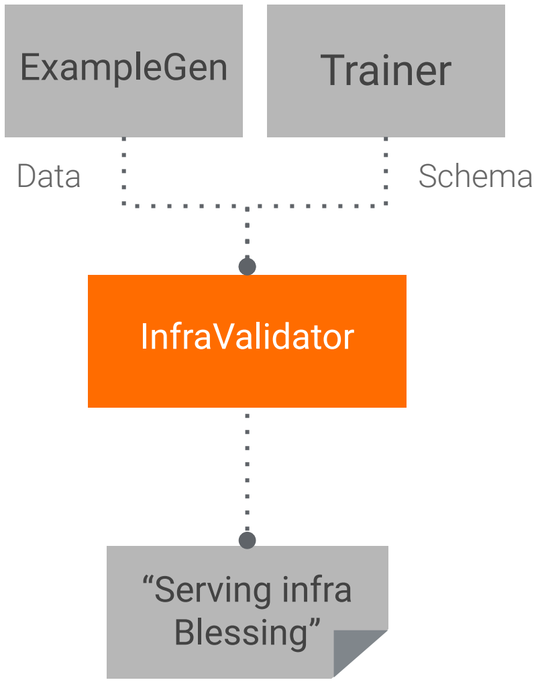
RequestSpec: Which model signature, how many examples to test.

ML project lifecycle benefits

- Validate model in model serving environment before pushing to production

Component: **InfraValidator**

Inputs and Outputs



Configuration

```
infra_validator = InfraValidator(
    model=trainer.outputs['model'],
    examples=example_gen.outputs['examples'],
    serving_spec=infra_validator_pb2.ServingSpec(...),
  validation_spec=infra_validator_pb2.ValidationSpec(...),
  request_spec=infra_validator_pb2.RequestSpec(...)
)
```

Configuration options

ServingSpec: Type of model server and infrastructure to test with e.g. TF Serving

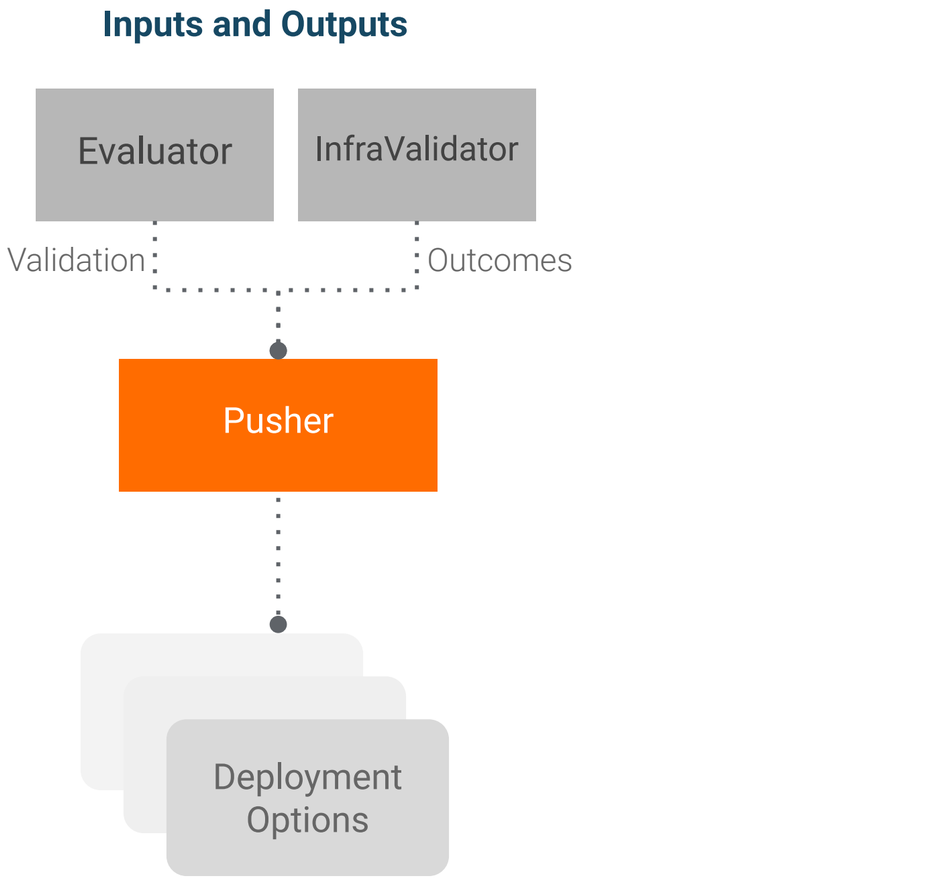
ValidationSpec: Adjusts the infra validation criteria or workflow.

RequestSpec: Which model signature, how many examples to test.

ML project lifecycle benefits

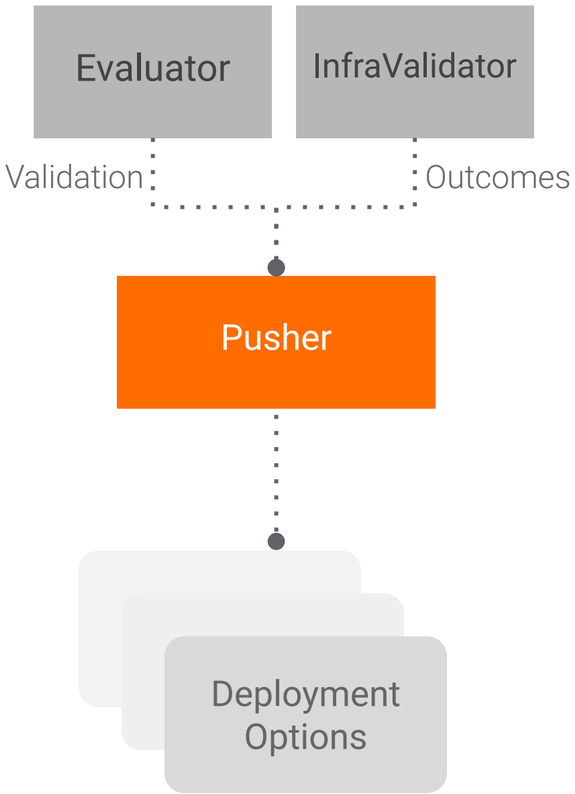
- Validate model in model serving environment before pushing to production
- Configurable to mirror different model serving environments such as Kubernetes and TF Serving

Component: **Pusher**



Component: **Pusher**

Inputs and Outputs

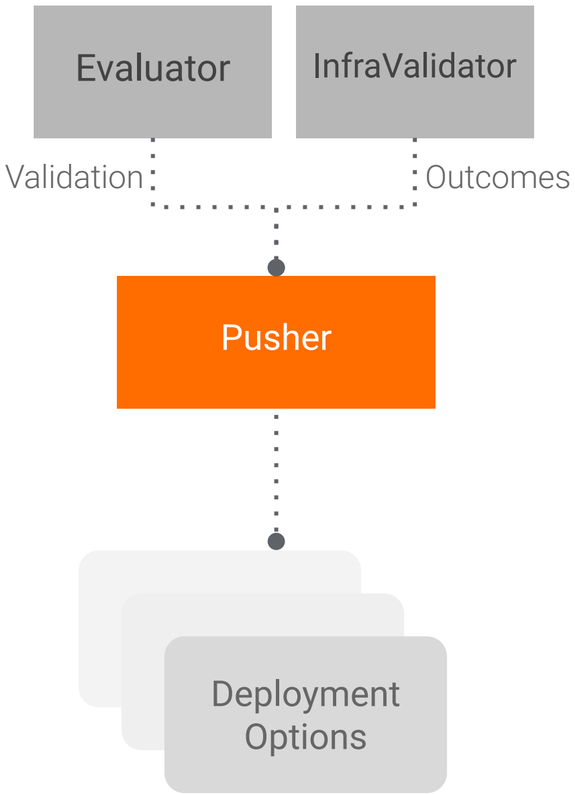


Configuration

```
pusher = Pusher(
    model=trainer.outputs['model'],
    model_blessing=model_analyzer.outputs['blessing'],
    infra_blessing=infra_validator.outputs['blessing'],
    push_destination=pusher_pb2.PushDestination(
        filesystem=pusher_pb2.PushDestination.Filesystem(
            base_directory=SERVING_MODEL_DIR)))
```

Component: **Pusher**

Inputs and Outputs



Configuration

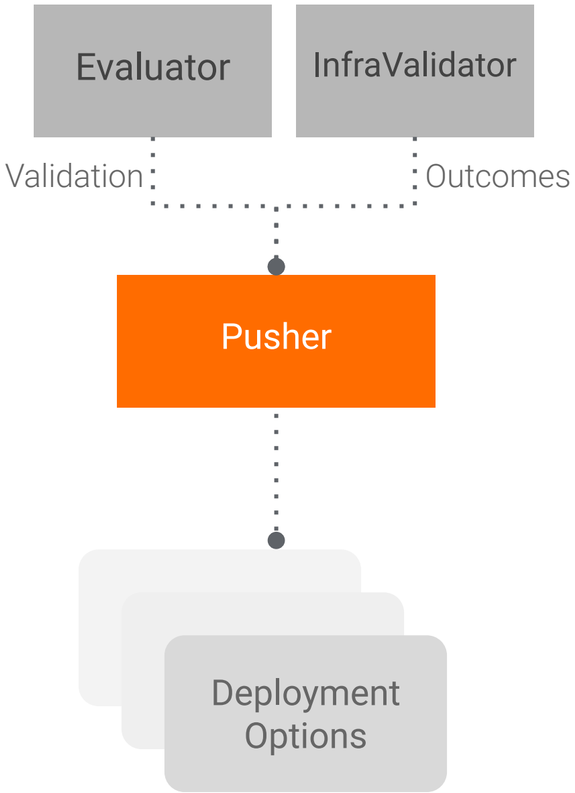
```
pusher = Pusher(
    model=trainer.outputs['model'],
    model_blessing=model_analyzer.outputs['blessing'],
    infra_blessing=infra_validator.outputs['blessing'],
    push_destination=pusher_pb2.PushDestination(
        filesystem=pusher_pb2.PushDestination.Filesystem(
            base_directory=SERVING_MODEL_DIR)))
```

ML project lifecycle benefits

- Validation checks on model performance and serving ability before pushing to production.

Component: **Pusher**

Inputs and Outputs



Configuration

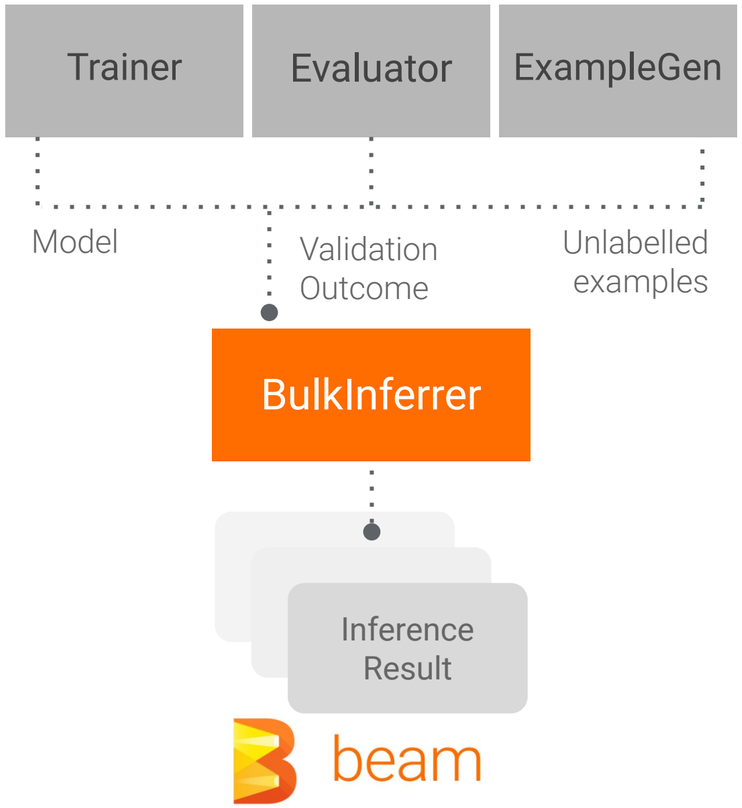
```
pusher = Pusher(
    model=trainer.outputs['model'],
    model_blessing=model_analyzer.outputs['blessing'],
    infra_blessing=infra_validator.outputs['blessing'],
    push_destination=pusher_pb2.PushDestination(
        filesystem=pusher_pb2.PushDestination.Filesystem(
            base_directory=SERVING_MODEL_DIR)))
```

ML project lifecycle benefits

- Validation checks on model performance and serving ability before pushing to production.
- Reusable model export code that is configurable for different push destinations:
  - Filesystem (TensorFlow Lite, TensorFlow JS)
  - Model Server (TensorFlow Serving)

Component: **BulkInferer**

Inputs and Outputs



Configuration

```
bulk_inferrer = BulkInferrer(
    examples=inference_example_gen.outputs['examples'],
    model_export=trainer.outputs['output'],
    model_blessing=evaluator.outputs['blessing'],
    data_spec=bulk_inferrer_pb2.DataSpec(
        example_splits=['unlabelled']),
    model_spec=bulk_inferrer_pb2.ModelSpec())
```

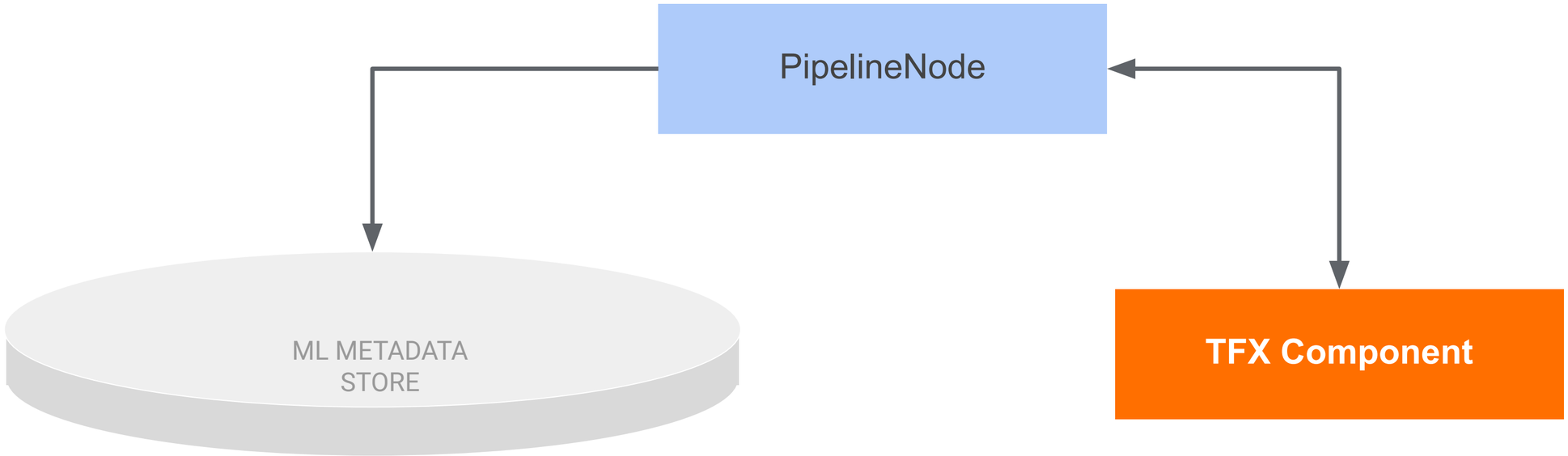
Configuration options

- Block batch inference on a successful model validation.
- Choose the inference examples from example gen's output.
- Choose the signatures and tags of inference model.

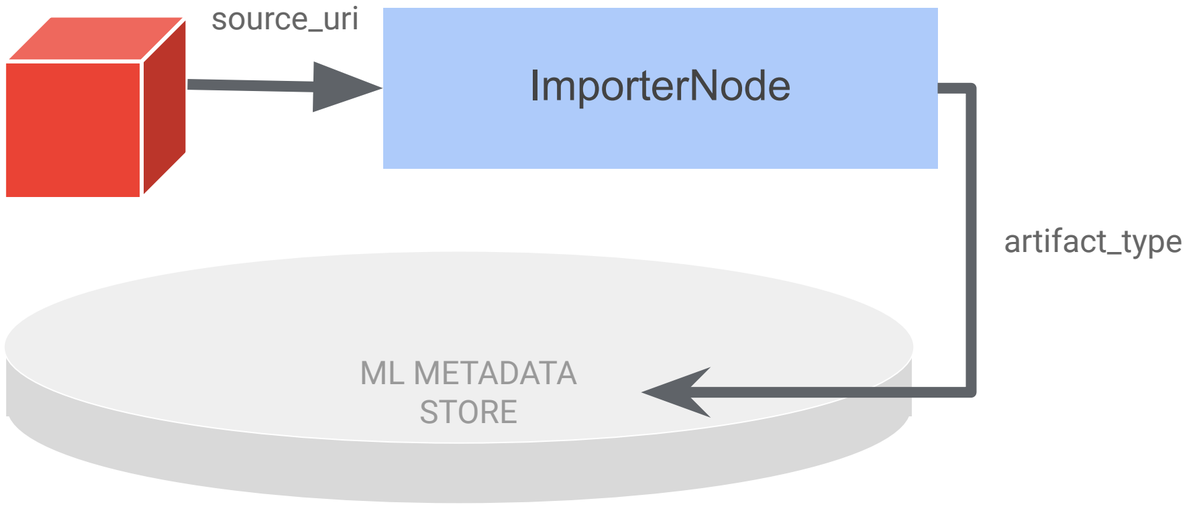
ML project lifecycle benefits

- Task and data-driven inference directly in your TFX pipeline

What are pipeline nodes?

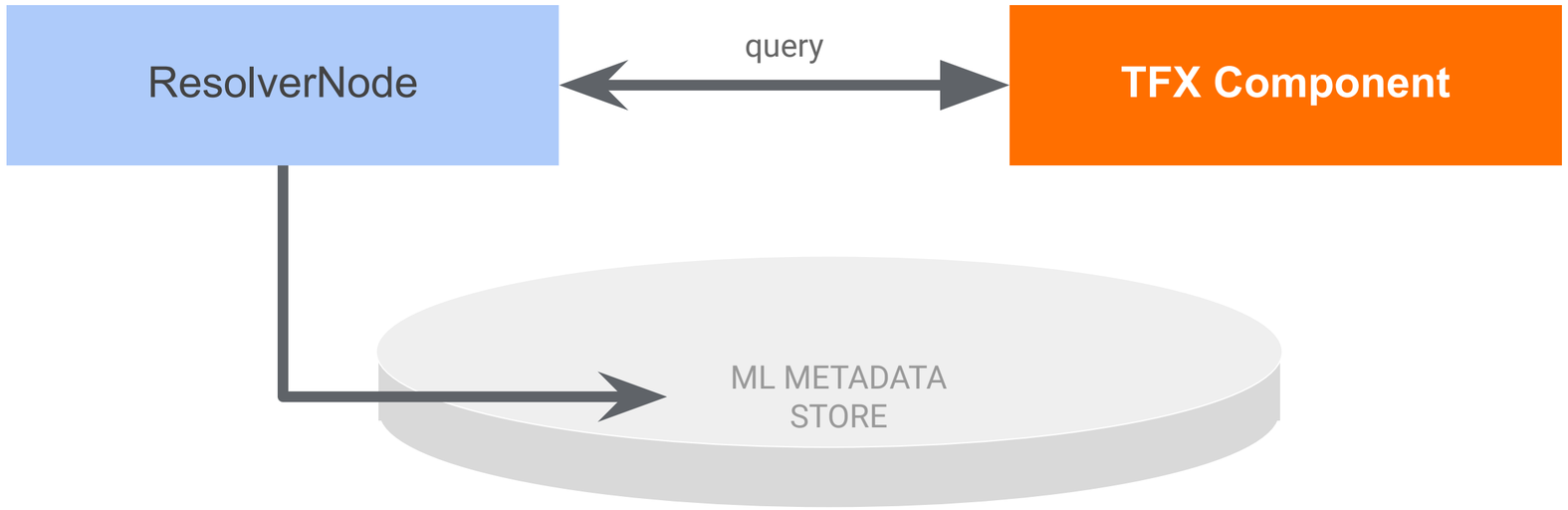


ImporterNode imports an external data object into ML Metadata



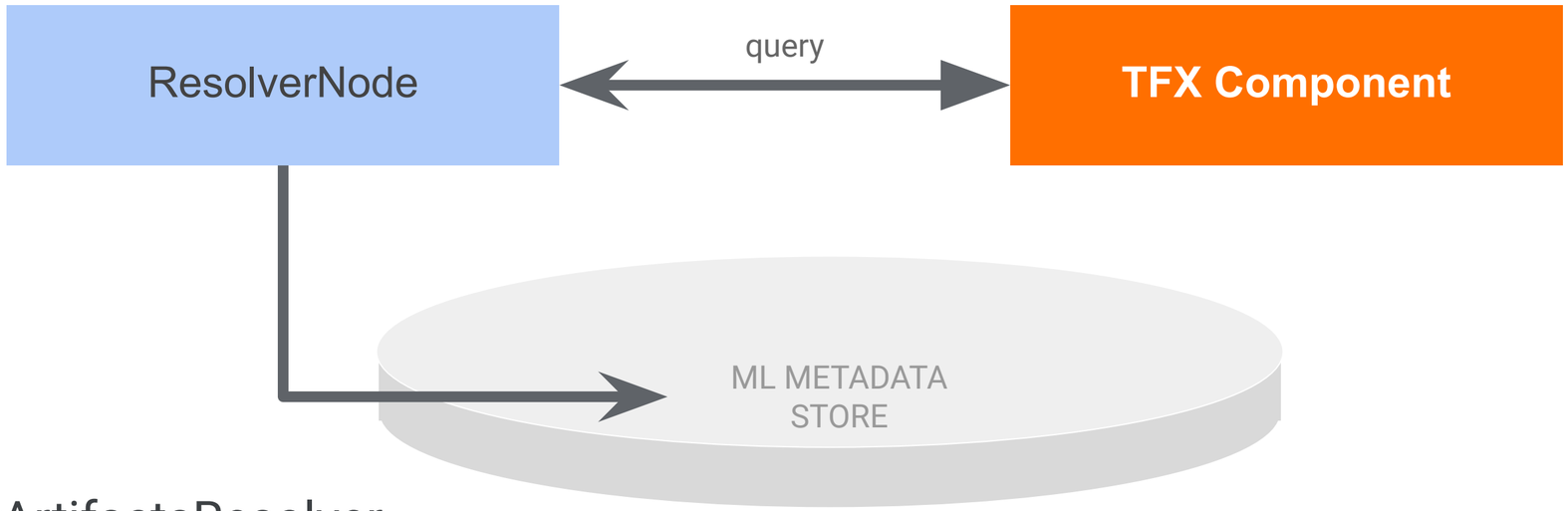

```
importer = ImporterNode(
      instance_name='import_schema',
      source_uri='uri/to/schema'
      artifact_type=standard_artifacts.Schema,
      reimport=False)
```

ResolverNode performs metadata queries



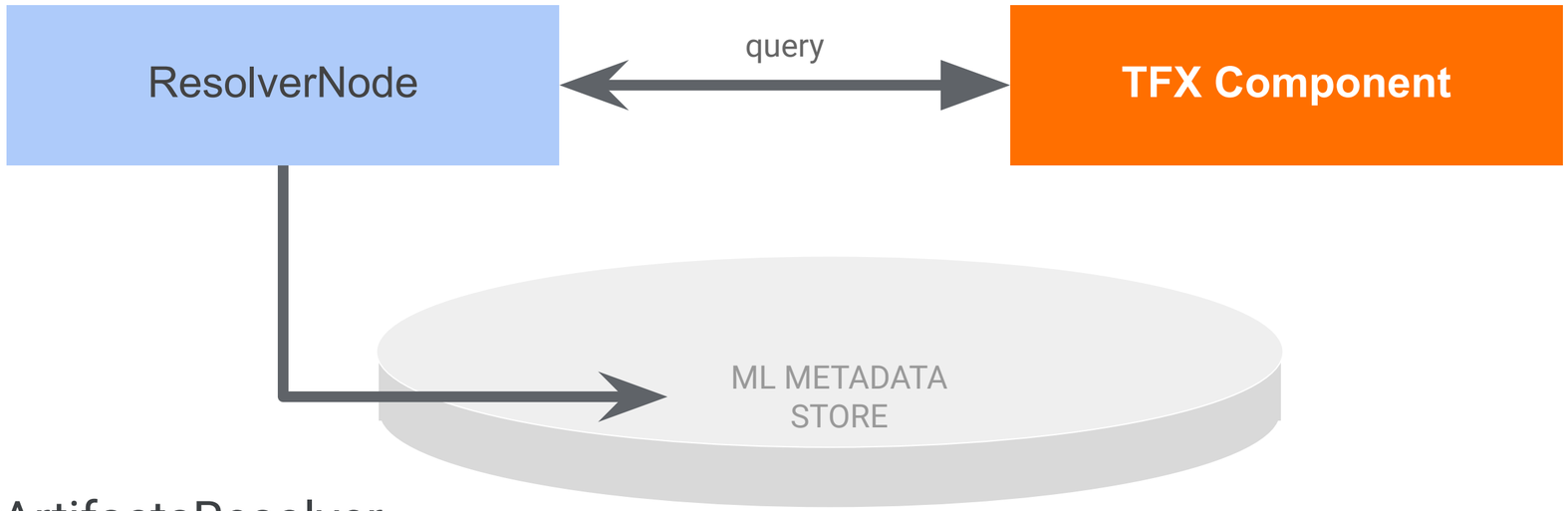

```
latest_five_examples_resolver = ResolverNode(
      instance_name='latest_five_examples_resolver',
      resolver_class=latest_artifacts_resolver.LatestArtifactsResolver, # Class
      resolver_config={'desired_num_of_artifacts' : 5},                 # Query
config
      examples=example_gen.outputs['examples'])                         # Channel
```

Current **ResolverNodes**



- LatestArtifactsResolver

Current **ResolverNodes**



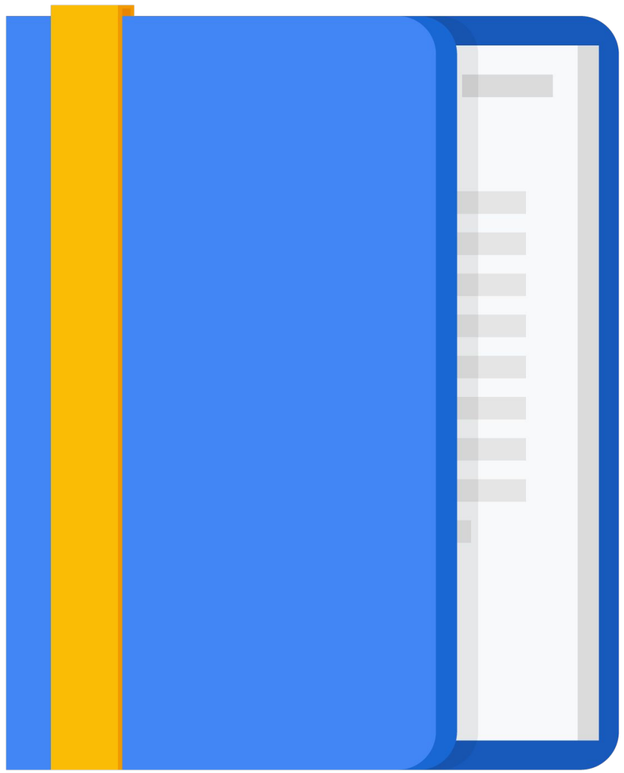
- LatestArtifactsResolver
- LatestBlessedModelResolver

Agenda

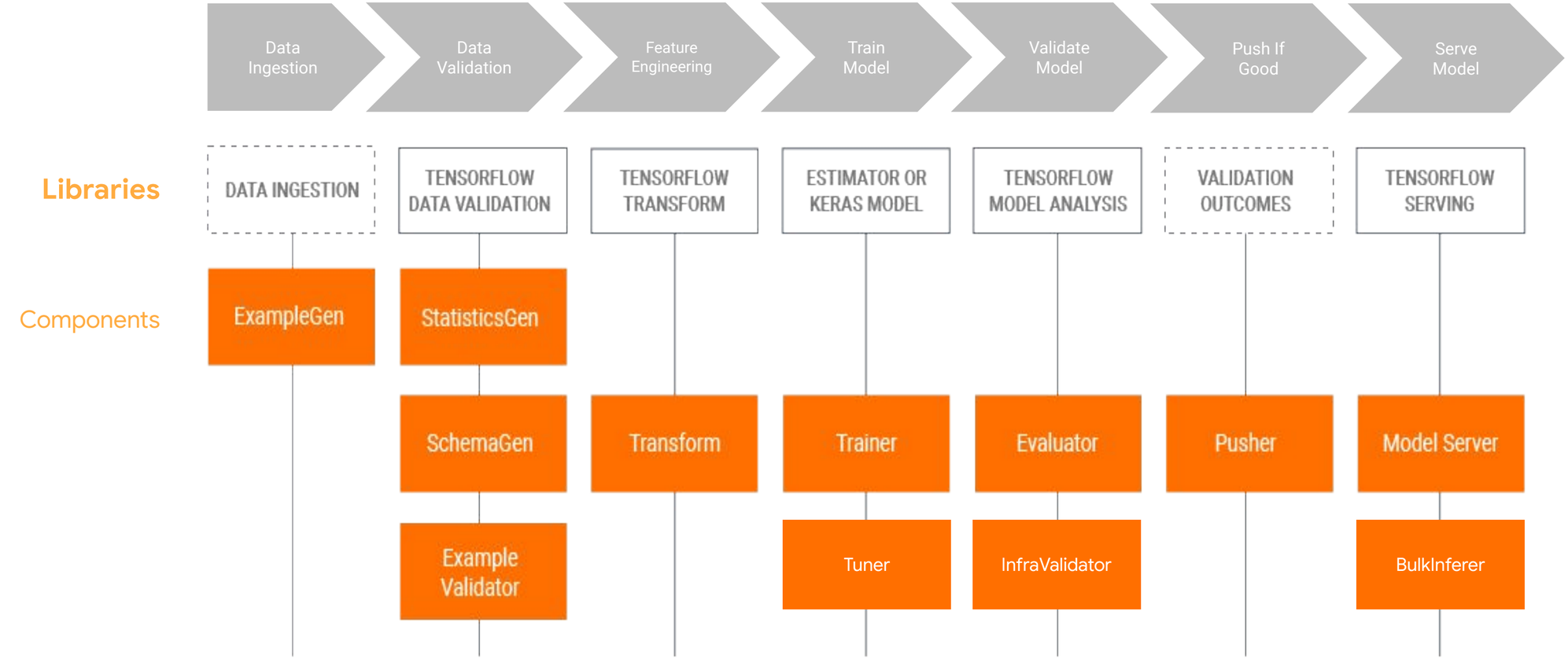
TensorFlow Extended (TFX)

TFX standard components

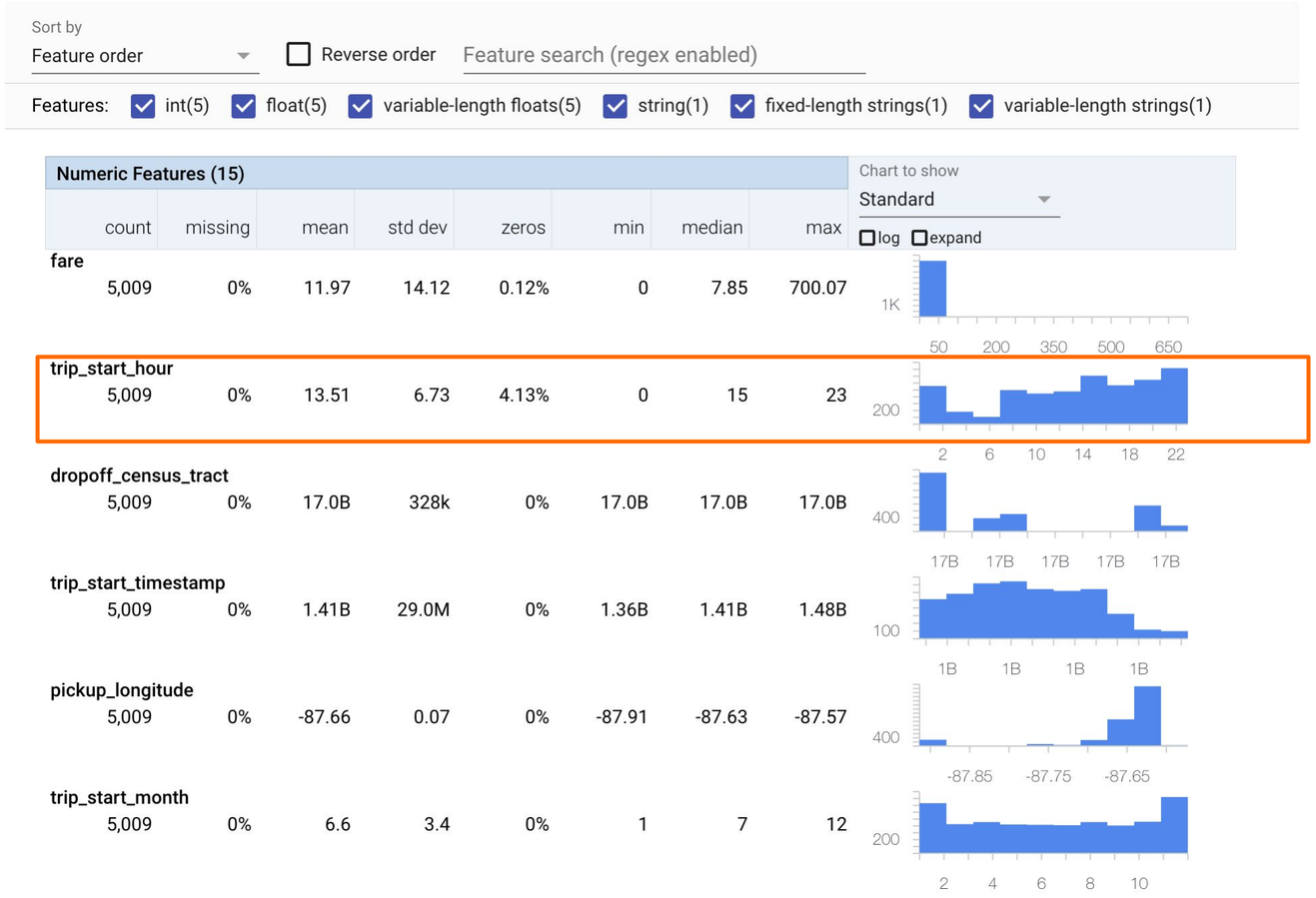
TFX libraries



TFX libraries integrate with each phase of a machine learning pipeline



TensorFlow Data Validation (TFDV) for analyzing and validating data



TensorFlow Transform (TFT) library for preprocessing data and feature engineering with TensorFlow

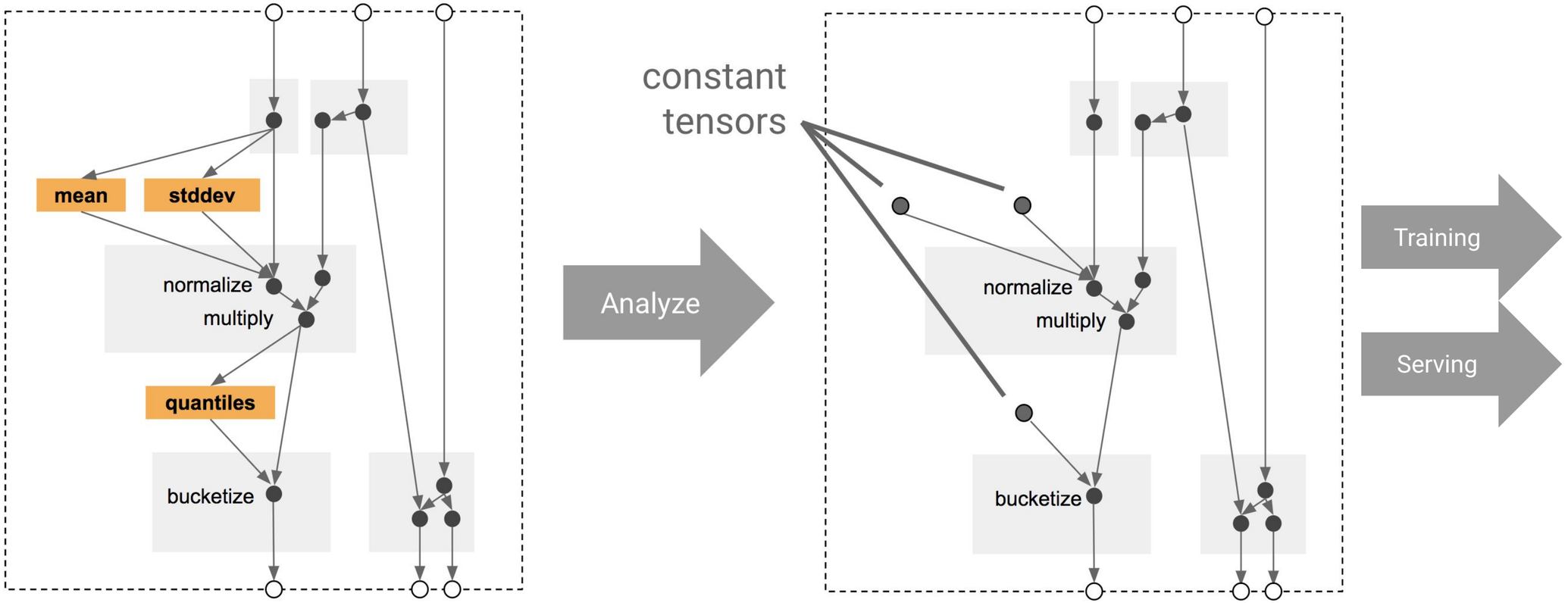
Analyzers

- max
- min
- mean
- sum
- var
- covariance
- quantiles
- size
- vocabulary
- pca

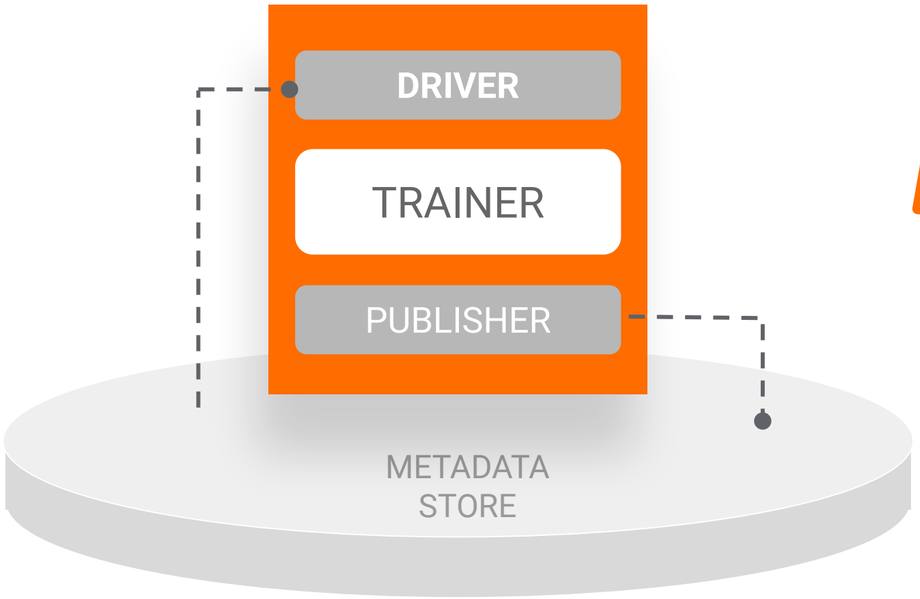
Mappers

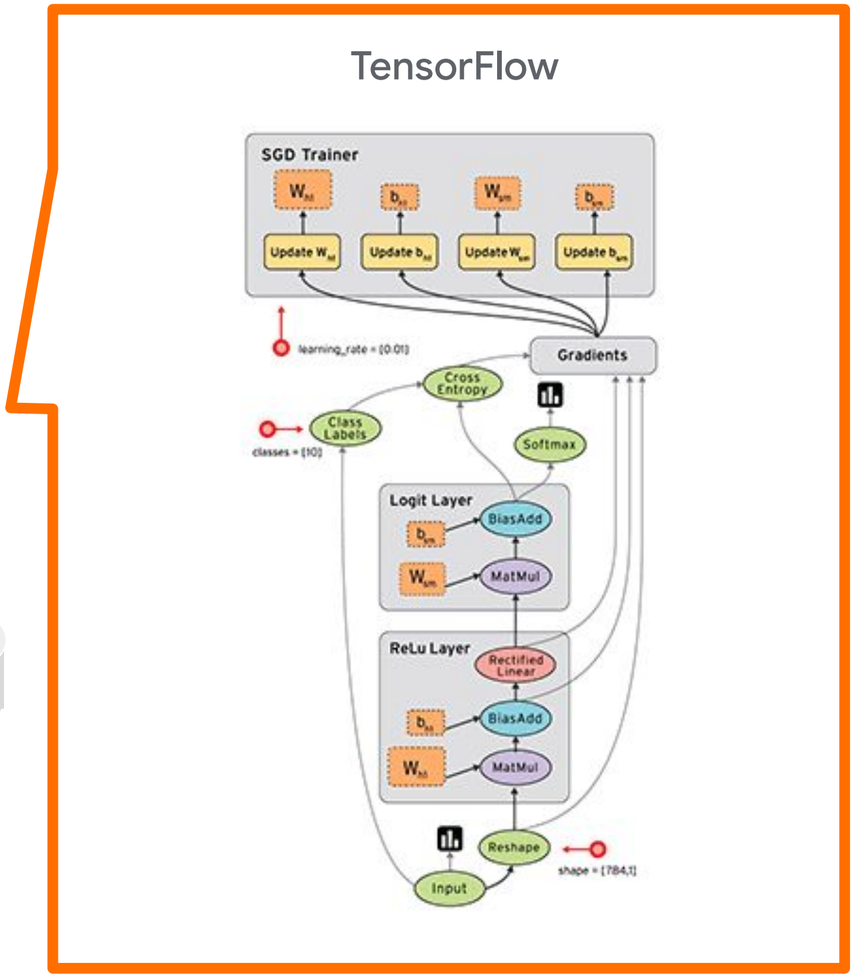
- bucketize
- apply_buckets
- hash_string
- ngram
- scale_0+1
- scale_z_score
- Scale_max_min
- tfidf
- compute_and_apply_vocabulary

TensorFlow Transform (TFT) library for preprocessing data and feature engin with TensorFlow

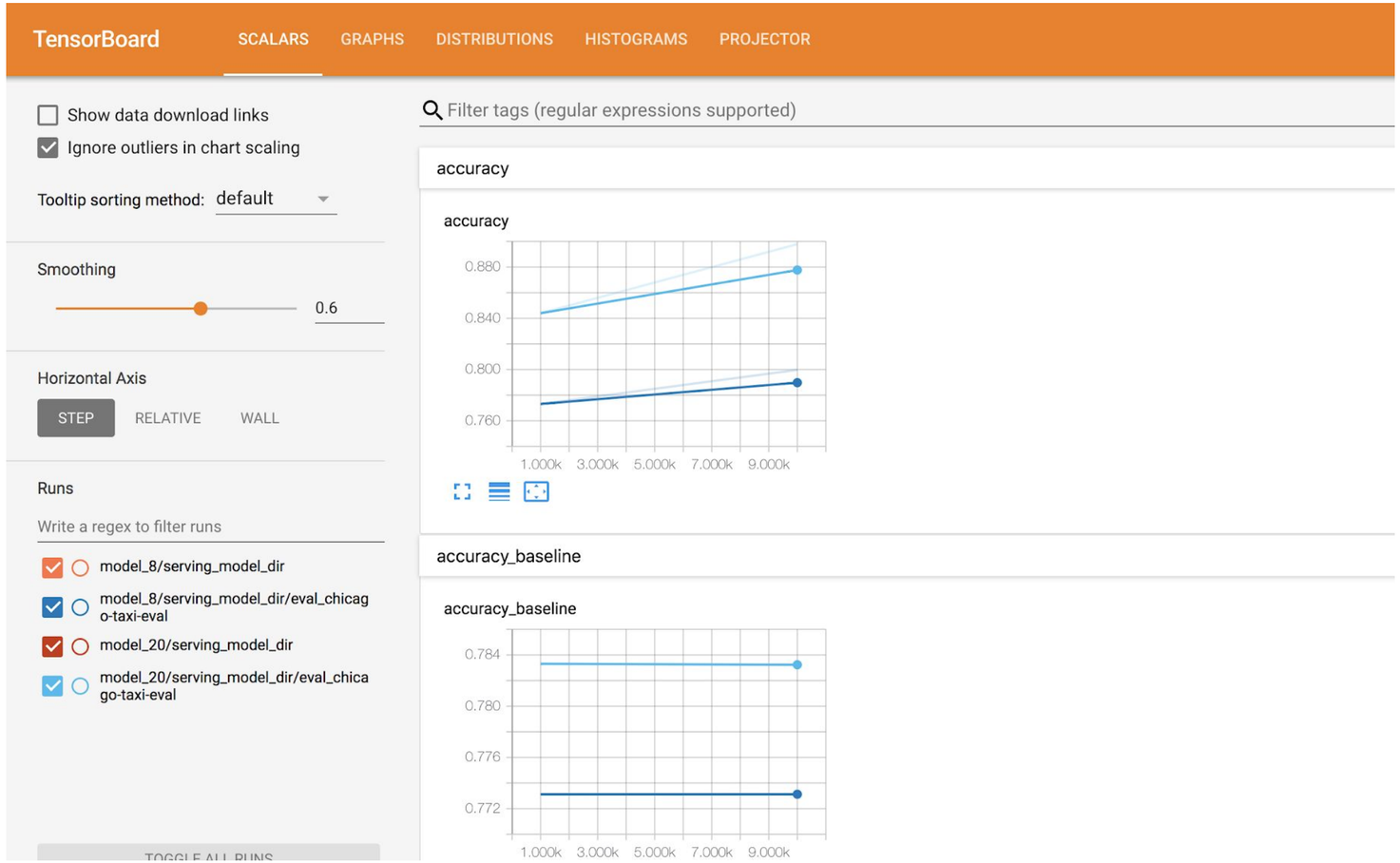


TensorFlow for model training and tuning





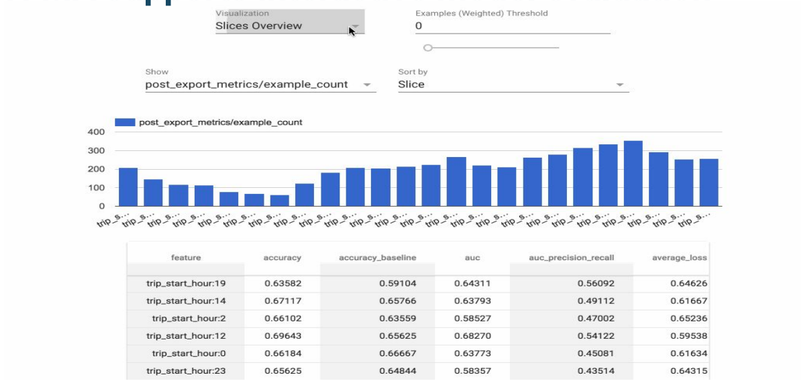
TensorBoard visualization and tooling for ML experimentation integrated with TFX



TensorFlow Model Analysis (TFMA) library for model evaluation

TensorFlow Model Analysis (TFMA) library for model evaluation

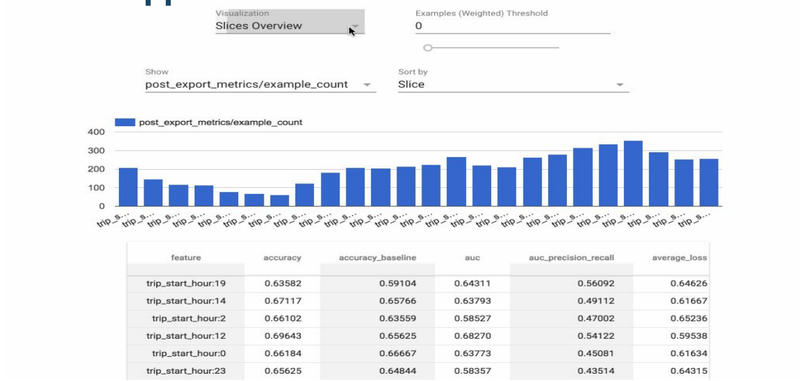
TFMA supports interactive model evaluation



feature	accuracy	accuracy_baseline	auc	auc_precision_recall	average_loss
trip_start_hour:19	0.63582	0.59104	0.64311	0.56092	0.64626
trip_start_hour:14	0.67117	0.65766	0.63793	0.49112	0.61667
trip_start_hour:2	0.66102	0.63559	0.58527	0.47002	0.65236
trip_start_hour:12	0.69643	0.65625	0.68270	0.54122	0.59538
trip_start_hour:0	0.66184	0.66667	0.63773	0.45081	0.61634
trip_start_hour:23	0.65625	0.64844	0.58357	0.43514	0.64315

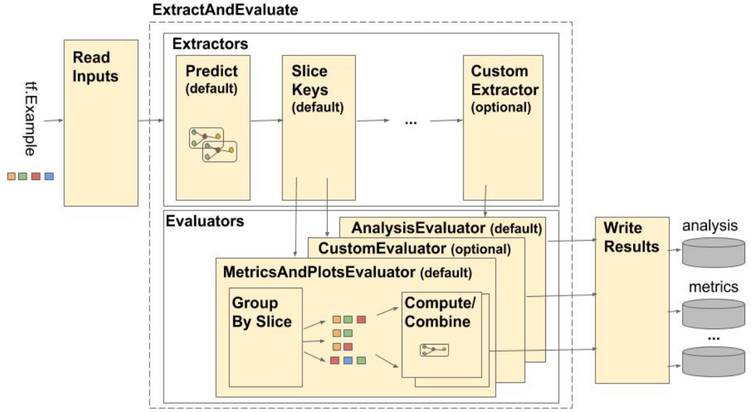
TensorFlow Model Analysis (TFMA) library for model evaluation

TFMA supports interactive model evaluation



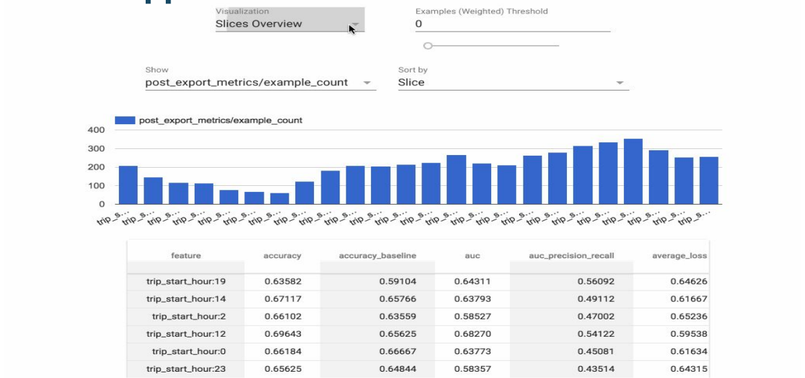
feature	accuracy	accuracy_baseline	auc	auc_precision_recall	average_loss
trip_start_hour:19	0.63582	0.59104	0.64311	0.56092	0.64626
trip_start_hour:14	0.67117	0.65766	0.63793	0.49112	0.61667
trip_start_hour:2	0.66102	0.63559	0.58527	0.47002	0.65236
trip_start_hour:12	0.69643	0.65625	0.68270	0.54122	0.59538
trip_start_hour:0	0.66184	0.66667	0.63773	0.45081	0.61634
trip_start_hour:23	0.65625	0.64844	0.58357	0.43514	0.64315

TFMA scales model evaluation with Apache Beam



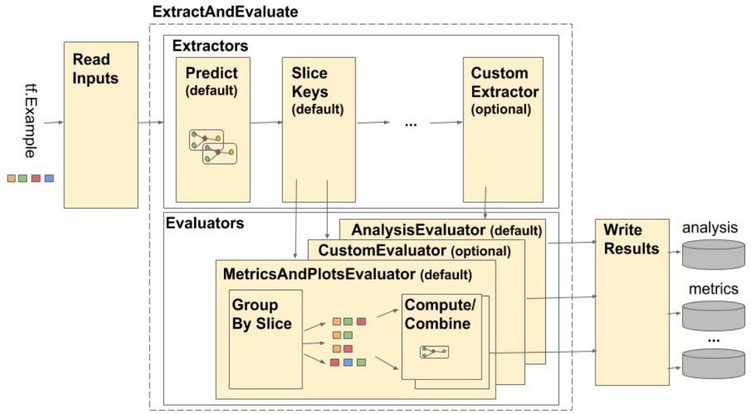
TensorFlow Model Analysis (TFMA) library for model evaluation

TFMA supports interactive model evaluation

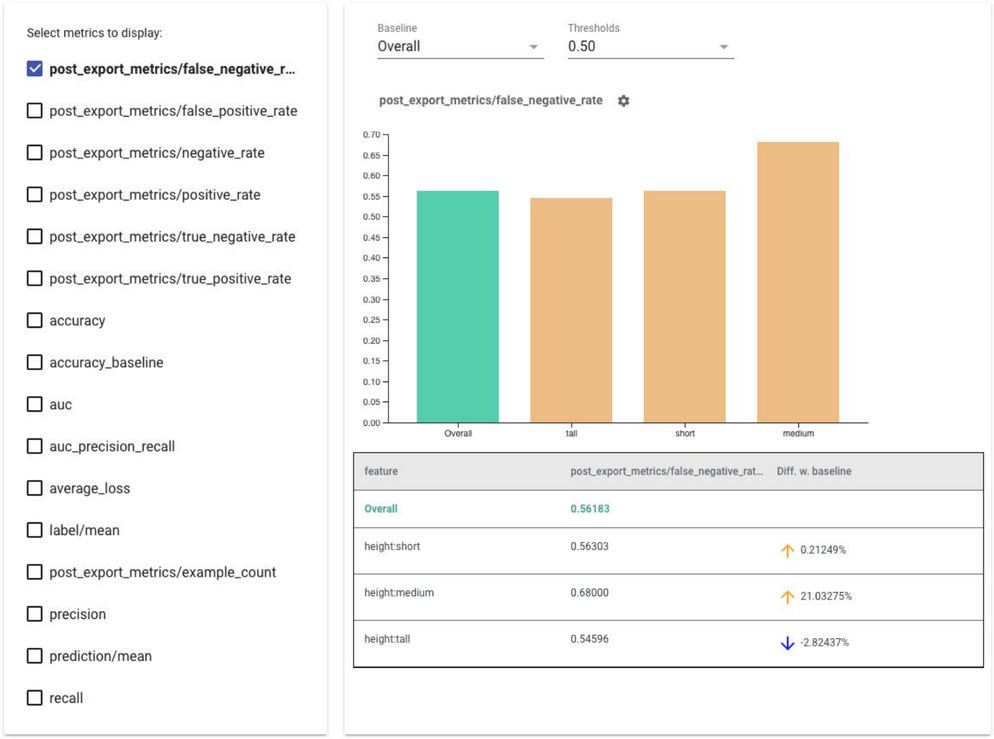


feature	accuracy	accuracy_baseline	auc	auc_precision_recall	average_loss
trip_start_hour:19	0.63582	0.59104	0.64311	0.56092	0.64626
trip_start_hour:14	0.67117	0.65766	0.63793	0.49112	0.61667
trip_start_hour:2	0.66102	0.63559	0.58527	0.47002	0.65236
trip_start_hour:12	0.69643	0.65625	0.68270	0.54122	0.59538
trip_start_hour:0	0.66184	0.66667	0.63773	0.45081	0.61634
trip_start_hour:23	0.65625	0.64844	0.58357	0.43514	0.64315

TFMA scales model evaluation with Apache Beam

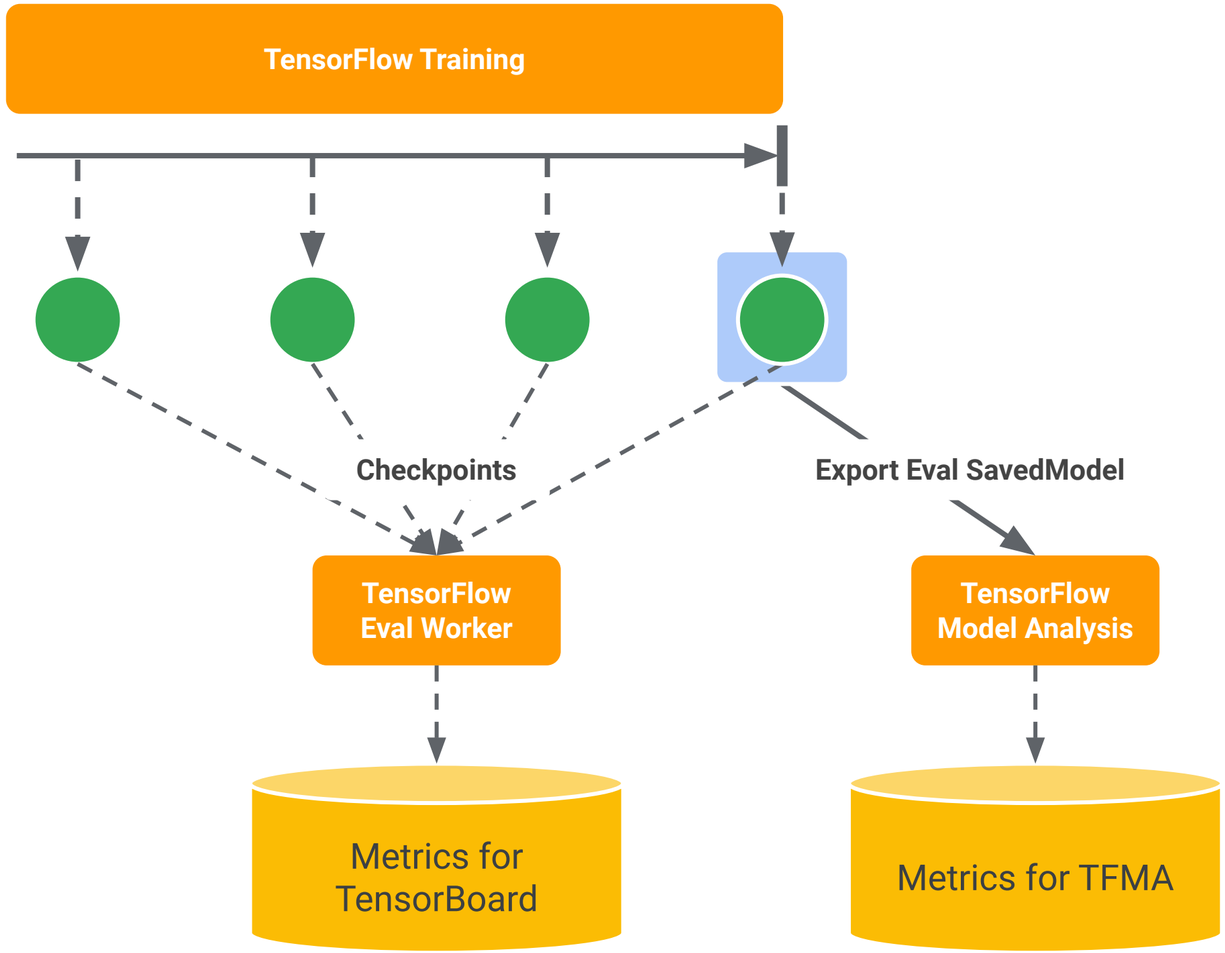


TFMA integrated Fairness Indicators for responsible AI development



feature	post_export_metrics/false_negative_rat...	Diff. w. baseline
Overall	0.56183	
height:short	0.56303	↑ 0.21249%
height:medium	0.68000	↑ 21.03275%
height:tall	0.54596	↓ -2.82437%

How does TFMA compare to TensorBoard?



Lab

TFX Walkthrough