DATA+AI SUMMIT 2022

Quick to production with the best of **Spark** and **Tensorflow** on **Databricks**

Ronny Mathew
Manager, Data Science
Rue Gilt Groupe

ORGANIZED BY databricks

What we will cover

- Using the best of both Spark and Tensorflow ecosystems
- Leverage recent advancements in Tensorflow 2 and tf extension libraries for Spark
- **Quick Distributed Deep Learning** on GPU and CPU spark clusters
- Using MLFlow **Model Registry** as a **central model storage**
- Unlocking the power of realtime inference using **Spark Streaming** and **MLFlow Serving**

Who is this talk for

- **Small-medium** full stack Data science teams looking for a **minimal** approach
- Primarily working or exploring **spark** and/or **tensorflow** based pipelines
- Skip the added complexity of MLOps code, but would like the flexibility to use one later
- Quickly move from **prototype** to **production** stage
- Enable **rapid prototyping** and **experimentations**

Why do we need distributed DL

Deep learning at Scale

- Feed a ton of features as input
- Deep learning models can capture more complex features and interactions
- Clickstream datasets are huge making them difficult to train on a single instance
- Recommenders and others are difficult to train on a subsample of the data
- Speed up training using multiple nodes, each with multiple GPUs/TPUs/CPUs

Unlocks Opportunities like

The world is our oyster

- Deep learning based recommenders and personalization models
- Cutting edge custom language models (BERT and more) in Spark pipelines
- Product Tagging and Catalog Management with NER, text and image models
- Visual models using Autoencoders and CNN models
- Online session-based recommendations w/ and w/out Reinforcement Learning

About Us

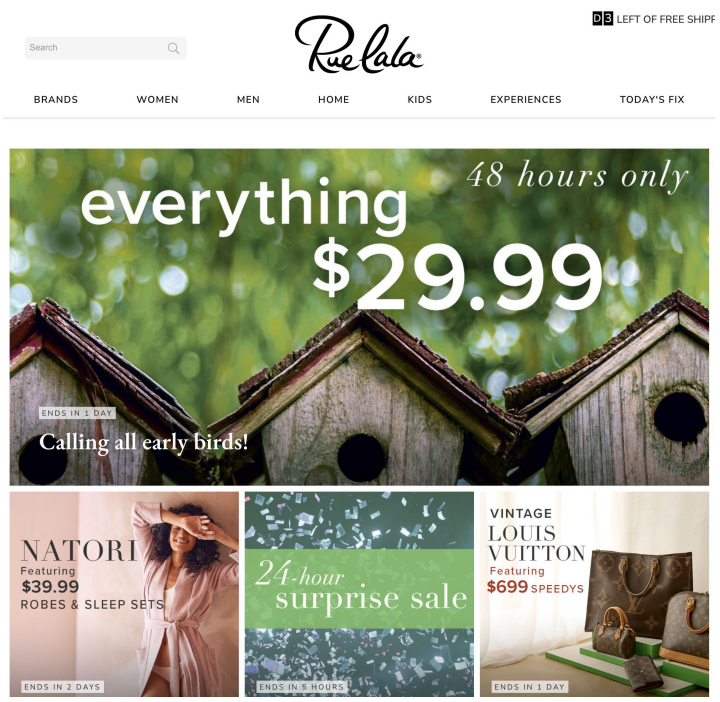
Who are we

RUE GILT GROUPE

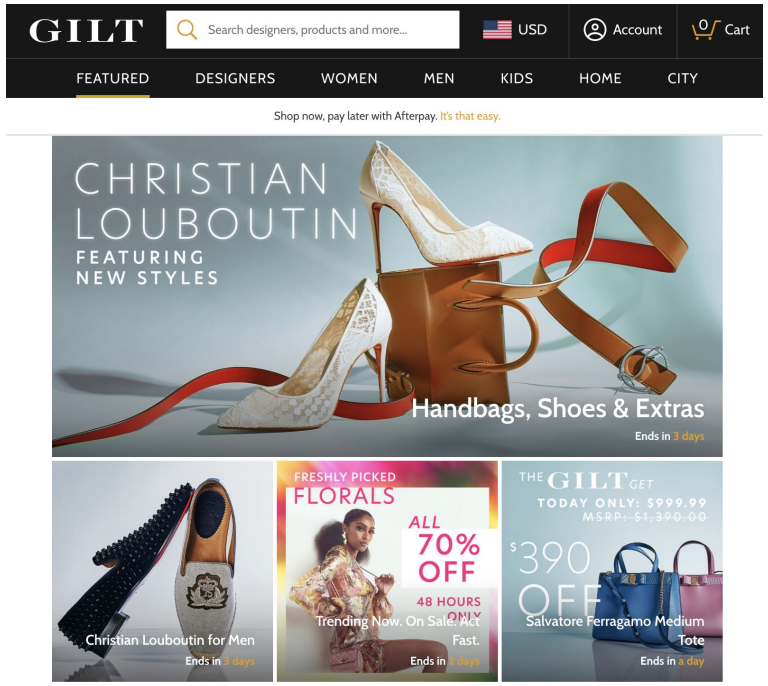
Rue Gilt Groupe is a fashion technology company based in Boston.

Our **50M+ members** get exclusive access to private sales on must have brands from an ever changing catalog of **3M+ products**

ruelala.com



gilt.com

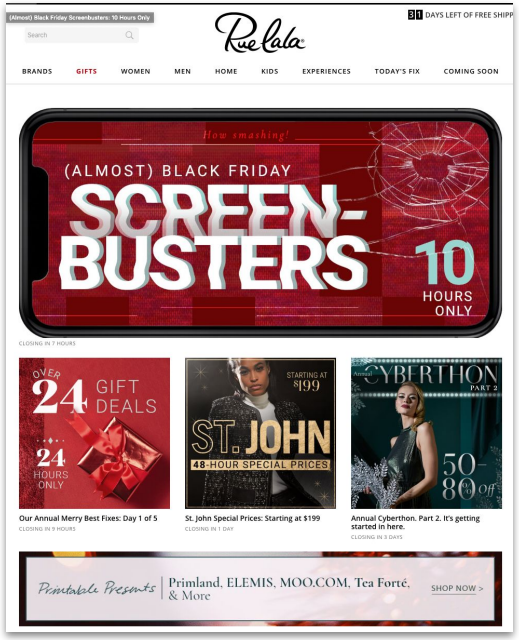


Data Science at Rue Gilt Groupe (RGG)

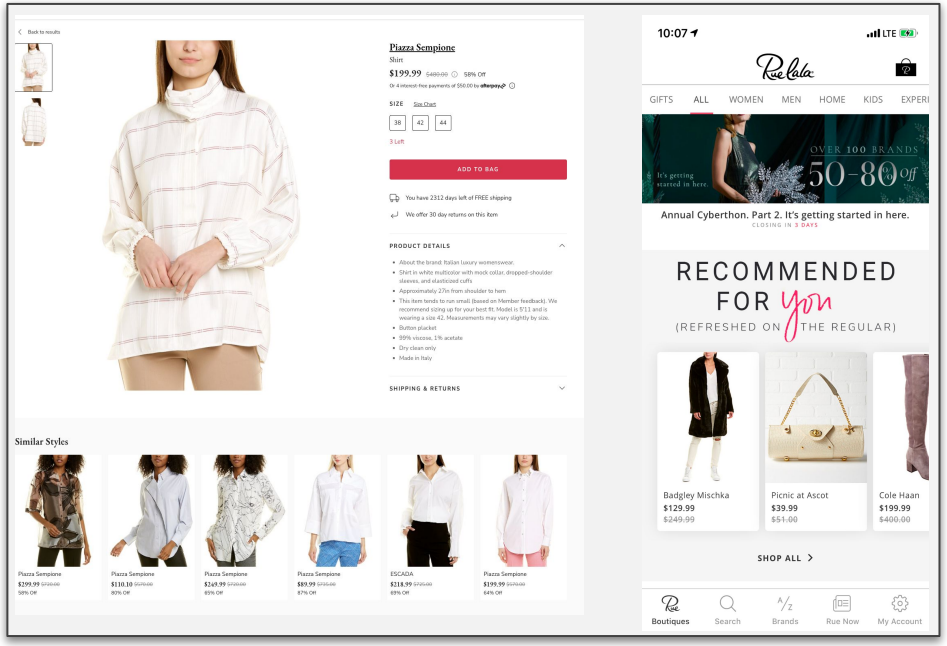
We are a close-knit group of passionate data scientists who thrive on collaboration and innovation

We work on various projects including **NLP, CV, time-series Forecasting, Recommenders**

We use ML to **personalize** the site and email experiences of our members among others projects



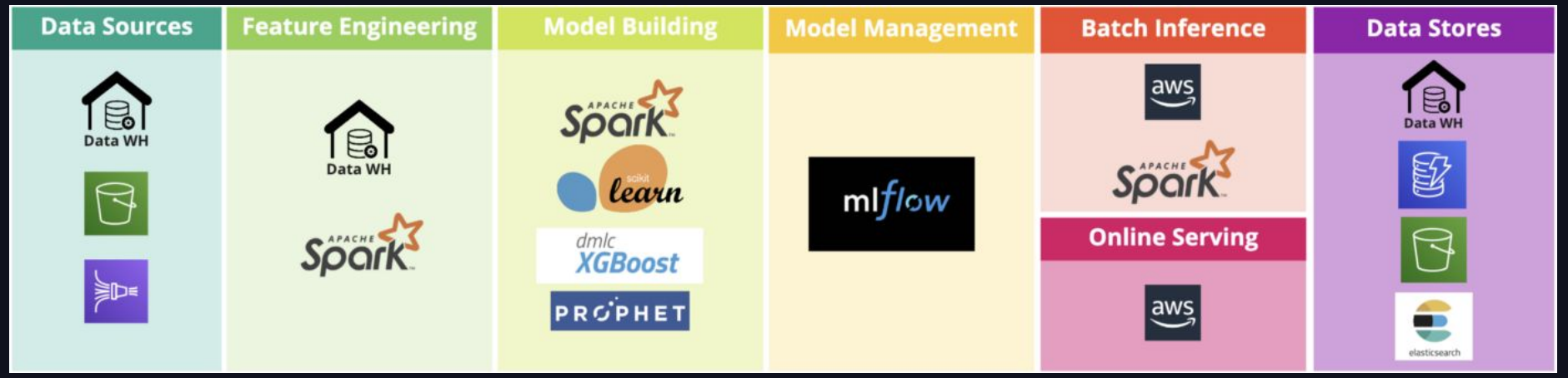
Sorting main pages



Product recs

Our previous ML stack

Spark ML and traditional ML focused



- Leverage Databricks to run our big data pipelines no Spark
- AWS for running container based ML services
- Use other ML packages like sklearn, xgBoost, openCV, prophet

Developing next-gen AI Models

Pushing the tech forward

★ **Our Requirements**

- Need to expand beyond **Spark ML** and leverage state-of-the-art **deep learning** models
- Lets Data Scientists and ML Engineers **focus more on research** and **actual ML tasks**

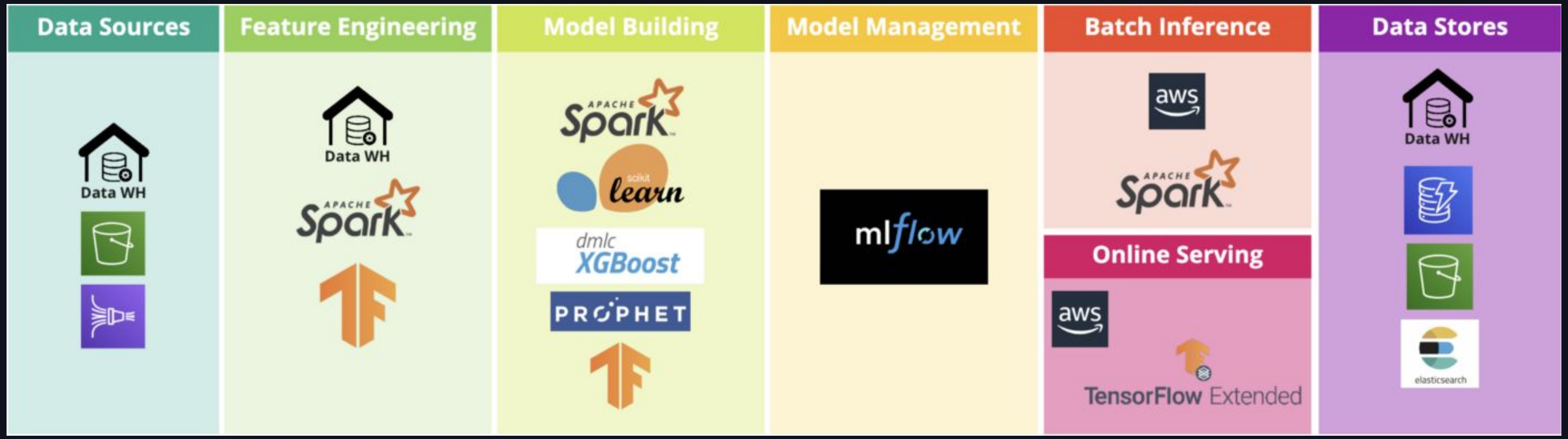
★ **What we were about to accomplish**

- Develop a **GRU based personalization model** prototype for **our new members**
- Converted the prototype to a ready to production model in **less than 2 weeks**
- Proved our new members ❤️ their deep learning recs with **>25% lift in CTR**!

Our new next-gen AI stack

Now with tensorflow

- Successfully integrated Tensorflow into our spark ML pipelines with these new libraries
- Have the flexibility to choose from Spark or Tensorflow ecosystem at any ML stage



Databricks+Spark

Simplifying big data

Databricks Spark

open source big data processing platform

- Write code in multiple programming languages
- Run pipelines on a single-node or clusters
- Work on structured and unstructured data
- Batch and streaming pipelines
- Distributed Machine learning with Spark ML

Data Pipelines in multiple languages

simplified ETL workloads

- Easy-to-use APIs in SQL, Python, Java, R, and/or Scala
- User-defined functions extend spark SQL functionalities

Data engineers

```
# in python
import pyspark.sql.functions as f
@f.udf("string")
def say_hello(name: str) -> str:
    return f"Hello {name}"
sqlContext.udf.register("say_hello", say_hello)
```

Data Analysts/Data Scientists

```
-- use the UDF in sql
SELECT say_hello('Bob')
```

Pandas UDFs

Manipulate spark like pandas

Allows using a pandas UDF to manipulate spark dataframes

Manipulate spark dataframe as pandas DataFrames instead of a per row transformation

```
@f.pandas_udf("string")

def simple_udf(iterator: Iterator[pd.Series]) ->
Iterator[pd.Series]:

       for x in iterator:

              yield pd.Series(list(map(lambda r: r +
       "1", x)))
```

Koalas

`quickly convert pandas to pyspark`

- implementing the pandas DataFrame API on top of Apache Spark
- a single codebase that works both with pandas (tests, smaller datasets) and with Spark
- No learning curve for ppl familiar with pandas APIs

TODO: add example

Spark ML

- Scalable implementations of many common ML algorithms, including
  - Traditional ML models like Regression, Trees, Multilayer Perceptron, Naive Bayes
  - Text models like Bag of words, Word2Vec, TF–IDF, LDA
  - ALS-based Collaborative Filtering and FP-Growth mining
  - Locality Sensitive Hashing and Random Projection for ANN and approx joins
- Libraries are available to integrate sklearn, XGBoost, tensorflow, pytorch and others

Pure Spark pipelines

End to end Spark on Databricks

- Read data from data source
- Data preprocessing and feature engineering on spark
- Write spark dataframes as parquet (or json, csv) to persistent store
- Read and feed data to models using spark
- Store models to MLflow or persistent store
- Build inference pipelines (batch & streaming)

Tensorflow

Simplifying deep learning

Tensorflow

Deep learning

- Machine learning framework maintained and used within Google
- V2 has adopted the core principle of **progressive disclosure of complexity** from Keras

> *You should always be able to get into lower-level workflows in a gradual way. You shouldn't fall off a cliff if the high-level functionality doesn't exactly match your use case. You should be able to gain more control over the small details while retaining a commensurate amount of high-level convenience.*
>
> source

- TFx extends tensorflow beyond model building and to end to end ML pipelines
- A lot of extension libraries that extends Tensorflow capabilities
- Great tutorials and guides on the official website for anyone to get started quickly
  - https://www.tensorflow.org/tutorials
  - https://www.tensorflow.org/guide

TF Data

Loading data into models

TensorFlow

- TF APIs for data loading and manipulation using Tensorflow datasets
- Recommended way to load data into a tensorflow 2.x model
- **TFRecordDataset** is a tf dataset that can load data directly from **tfRecord** files
- Does a lot of great optimizations behind the scenes including prefetching and caching
  - While training on GPU, prefetch uses the CPU to load the next n batches
  - **tf.data.AUTOTUNE** can automatically adapt prefetching (and others) to your cluster

```
dataset = (tf.data.TFRecordDataset(filenames=[list of tfrecord files]))
                .batch(1000)
                .shuffle(10000)
                .prefetch(tf.data.AUTOTUNE))
```

TF Distribute

TensorFlow

- Provides synchronous/asynchronous distributed training strategies on GPU/TPU, etc
- Automatically assigns **default strategy** to every training loop
- **A single-node development code can be converted to a distributed code by adding a few lines initially when we start development**

```
# we can also add more conditions here, eg check for num workers
# add more strategies as needed
if tf.config.list_physical_devices('GPU'):
    my_strategy = tf.distribute.MirroredStrategy()
else:  # Use the Default Strategy
    my_strategy = tf.distribute.get_strategy()

with my_strategy:
    # wrap all the training code within this scope
    ...
```

Distributing Datasets

TensorFlow

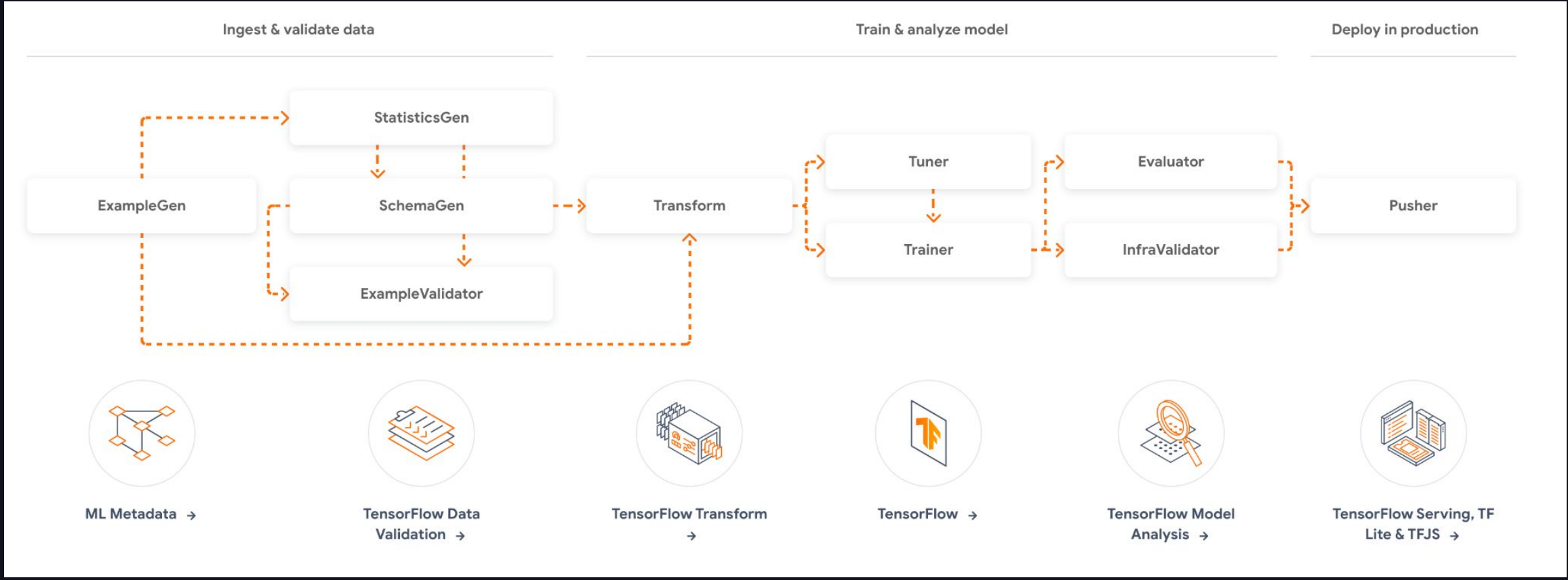
Convert a tf dataset to a distributed dataset in a few lines of code

TODO: Add diff between FILE and DATA modes

```
# There are two shard policies, DATA and FILE
options = tf.data.Options()
options.experimental_distribute.auto_shard_policy =
tf.data.experimental.AutoShardPolicy.DATA
dataset.with_options(options)
```

TF Extended (TFx)

End-to-end machine learning platform for tensorflow



source

Some cool TFx components

TensorFlow

- **Tensorflow Transform**:
  - Performs feature engineering and transforms
  - Alternative to using Spark (as in our case)
- **Tensorflow Model Analysis (TFMA)**
  - Model validations
  - Monitoring issues and drifts in models

TF Serving

Some cool TFx components

- **Tensorflow Serving**
  - Core Tensorflow component responsible for deploying and online serving
  - Loads and serves a model at a high performant REST endpoint
  - Docker containers provided by google with different flavors of tensorflow
  - Can be scaled as needed using dockerized containers for online recommendations

TF Recommenders

TensorFlow

- Extension library for building Deep learning Recommenders
- Provides easy to use wrappers for models, custom metrics and losses
- Exposes google's *scann* approximate nearest neighbor engine as a Layer
- Simplifies building models for
  - Retrieval (narrow down entire corpus to a few thousand candidates)
  - Ranking (rank top candidates to a handful with more complex architecture)
  - Multitask recommenders (learn from multiple objectives)
  - Deep Cross Networks (capture explicit feature interactions between items)
  - and more

More Extension Libraries and tools

TensorFlow

- **TF Ranking**
  - Build Learning-to-Rank (LTR) models
  - commonly used loss functions like pointwise, pairwise, and listwise
  - ranking metrics like Mean Reciprocal Rank and NDCG
- **TF Agents**
  - Enables building contextual bandit and RL agents

Use pre-trained models easily

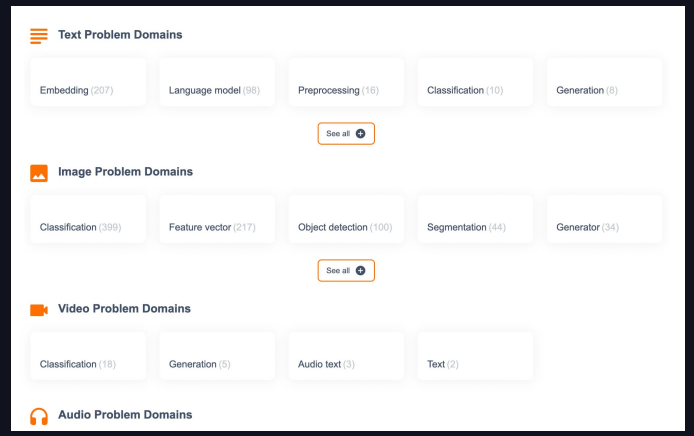
Why reinvent the wheel

TensorFlow

TF Hub

- Collection of the latest pre-trained models for text, image, video, audio, and more
- Easily pull to any existing model as a layer with simple code,

```
# sometimes we also need to load preprocessing layers
bert_path="https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/4"
bert_embeddings = hub.KerasLayer(bert_path, trainable=False)
```



Pure TFx pipeline

End to end Tensorflow and TFx

- Read data from some source
- Data processing using TF pipelines or manually from other file types
- Write tf record files (or other formats supported by tensorflow) to persistent store
- Read and feed tf records to models using tf.data
- Store models to S3 or persistent store
- Create inference pipelines (batch & streaming)

Databricks+MLFlow

Simplifying model management

MLFlow

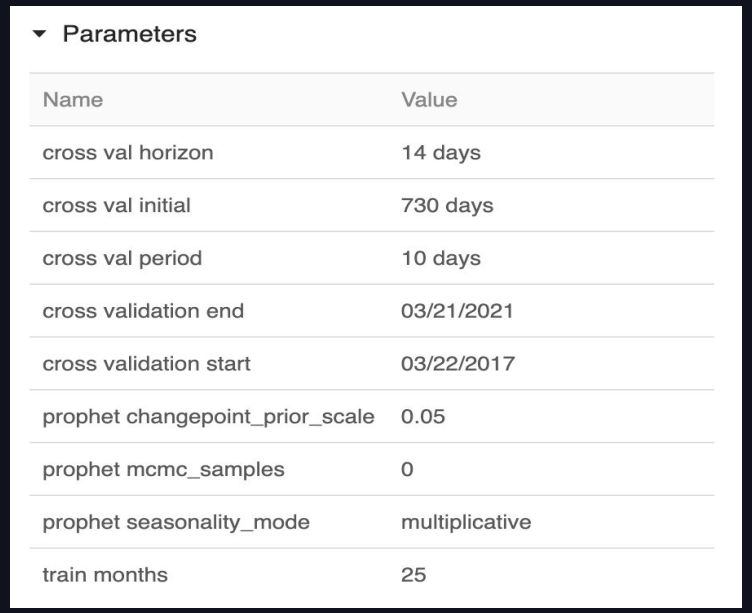
- Open source platform for ML lifecycle
- Enables Experiment tracking, reproducibility, deployability and model management
- Works with most common ML libraries
- Seamlessly integrates with Spark and tensorflow

source

Experiment Tracking

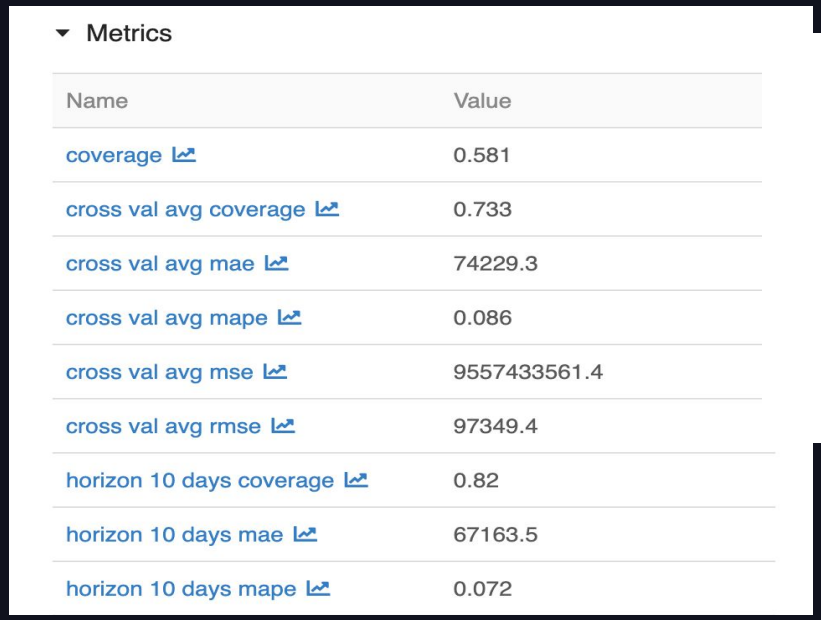
MLFlow on databricks

- Track Hyperparameters, Metrics and Artifacts associated with each and every run
- **Compare metrics** from different runs and use that to **select** the best model
- **Autologging** capabilities for common frameworks



Parameters

Name	Value
cross val horizon	14 days
cross val initial	730 days
cross val period	10 days
cross validation end	03/21/2021
cross validation start	03/22/2017
prophet changepoint_prior_scale	0.05
prophet mcmc_samples	0
prophet seasonality_mode	multiplicative
train months	25



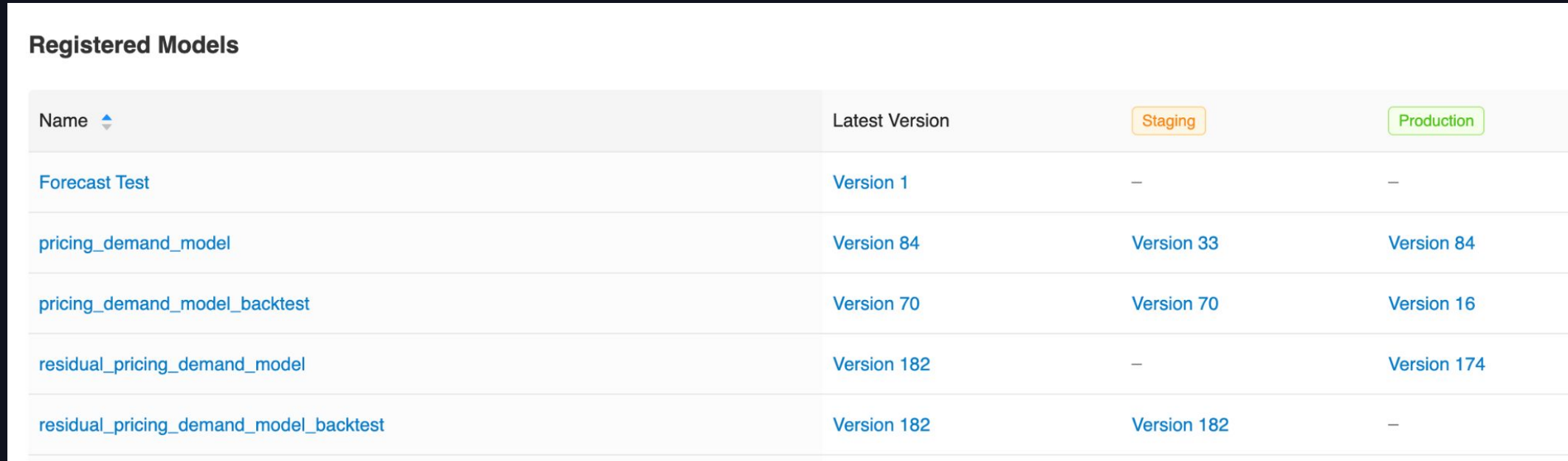
Metrics

Name	Value
coverage	0.581
cross val avg coverage	0.733
cross val avg mae	74229.3
cross val avg mape	0.086
cross val avg mse	9557433561.4
cross val avg rmse	97349.4
horizon 10 days coverage	0.82
horizon 10 days mae	67163.5
horizon 10 days mape	0.072

Model Management

MLFlow on databricks

- **Manage models** using MLFlow models and Model Registry
- Transition model versions between Staging, Production and Archived stage
- Single source of truth for all models and its versions ensuring **consistency**



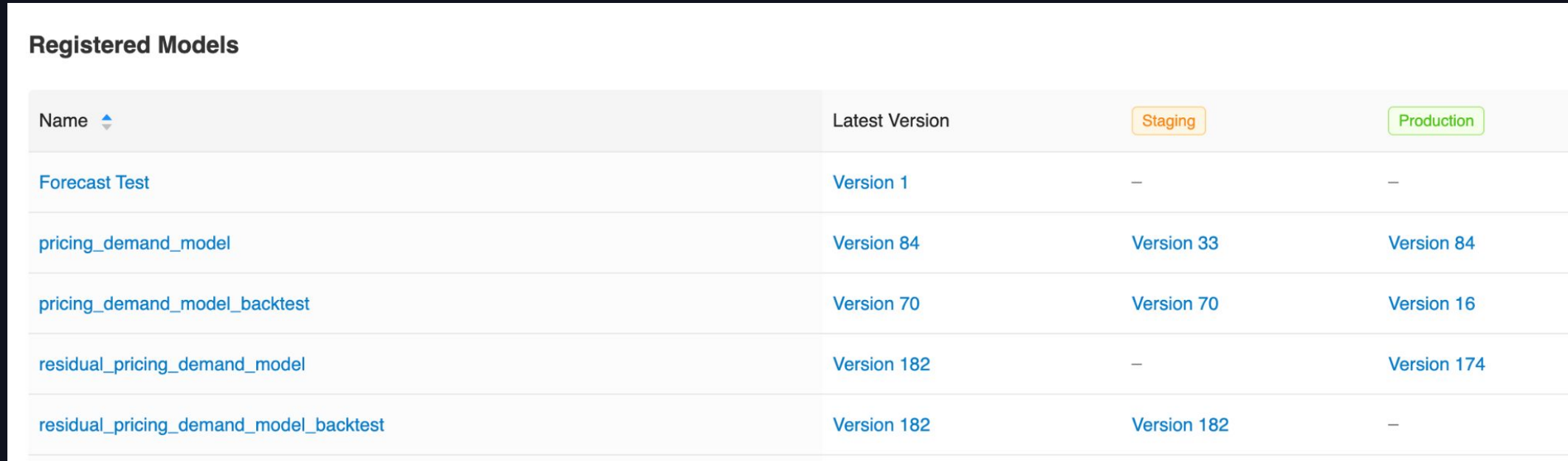
Registered Models

Name	Latest Version	Staging	Production
Forecast Test	Version 1	–	–
pricing_demand_model	Version 84	Version 33	Version 84
pricing_demand_model_backtest	Version 70	Version 70	Version 16
residual_pricing_demand_model	Version 182	–	Version 174
residual_pricing_demand_model_backtest	Version 182	Version 182	–

MLFlow Model Serving

MLFlow on Databricks

- **Manage models** using MLFlow models and Model Registry
- Transition model versions between Staging, Production and Archived stage
- Single source of truth for all models and its versions ensuring **consistency**



Registered Models

Name	Latest Version	Staging	Production
Forecast Test	Version 1	–	–
pricing_demand_model	Version 84	Version 33	Version 84
pricing_demand_model_backtest	Version 70	Version 70	Version 16
residual_pricing_demand_model	Version 182	–	Version 174
residual_pricing_demand_model_backtest	Version 182	Version 182	–

Databricks+Spark +Tensorflow+MLFlow

One simple recipe to conquer them all

TFRecords

Native tensorflow format

- Tensorflow native file format
- Stores sequences of binary records as **Examples**
- Binary format improves performance as they take less space and can be read efficiently
- Optimized for use with TFx
- Very efficient in storing sequential data

TF records with Spark

Making spark speak the same language

- Reading/writing TFrecords from Spark ensures both Spark and TFx uses same file format
- Enables the flexibility of using Spark or TFx at any point of our ML lifecycle

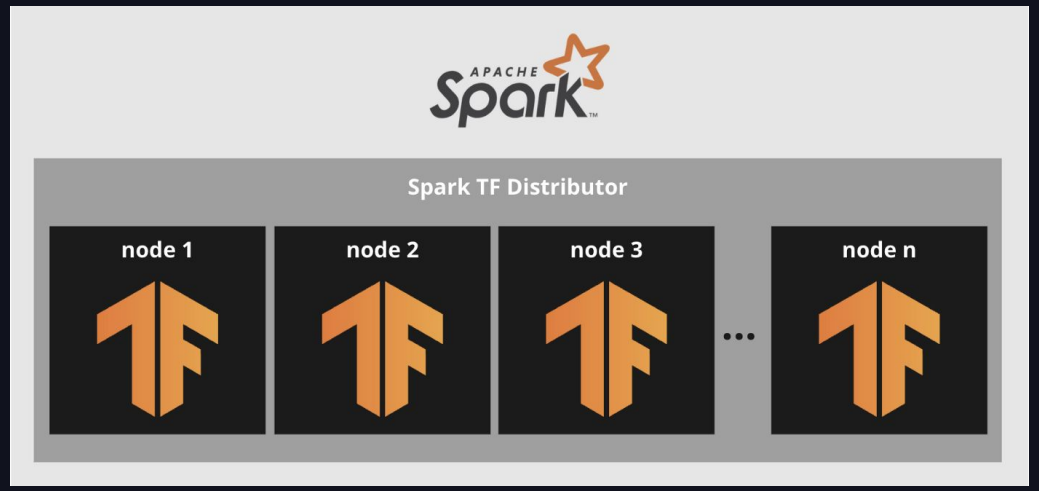
```
# Write tfRecords as Example from a Spark Dataframe
some_df.write.format("tfrecords")
       .option("recordType", "Example")
       .mode("overwrite")
       .save(save_path))
```

Other approaches (like Petastorm) achieves similar goals but loses interoperability with TFx

Spark Tensorflow Distributor

Run tensorflow code on spark clusters

- Part of the TensorFlow ecosystem
- Enables running **tf.distribute strategies** inside Spark jobs
- **Delegate** cluster management and communication to **Spark**
- Also runs on CPU clusters
- Supports custom_strategy to use other tf.distribute strategies (TPUs, GPUs, etc)
- Update train method with new strategy, add custom_strategy=True to the runner



Spark Tensorflow Distributor

Straightforward usage,

- Define both dataset loading and TF model building code in a **train** method
- Pass the number of workers in the cluster and train method to the distributed runner

```
num_workers = 10

# add local_mode=True for local testing

# you can also specify another strategy in the train method and
turn on custom_strategy=True

MirroredStrategyRunner(num_slots=num_workers).run(train)
```

Chief node

Pro Tips

- Treat callbacks and other operations differently on chief
  - Don’t need all the workers (over)writing the model checkpoints to the same path!
- tf.distribute works by specifying a tf_config environment variable on each node
  - config also includes a node index where *node 0 is the chief node*
- We can find the role of a node using this code,

```
if 'TF_CONFIG' in os.environ:
    tf_config = json.loads(os.environ['TF_CONFIG'])
    node_index = tf_config['task']['index']
    is_chief = node_index == 0
```

Dont forget to save your models!

Pro Tips

Since the training happens on a cluster and not just a single instance,

we don't have access to the final trained model unless we handle it explicitly!

That means you might be running training for days *but there would be no way to retrieve your final model (yikes!).*

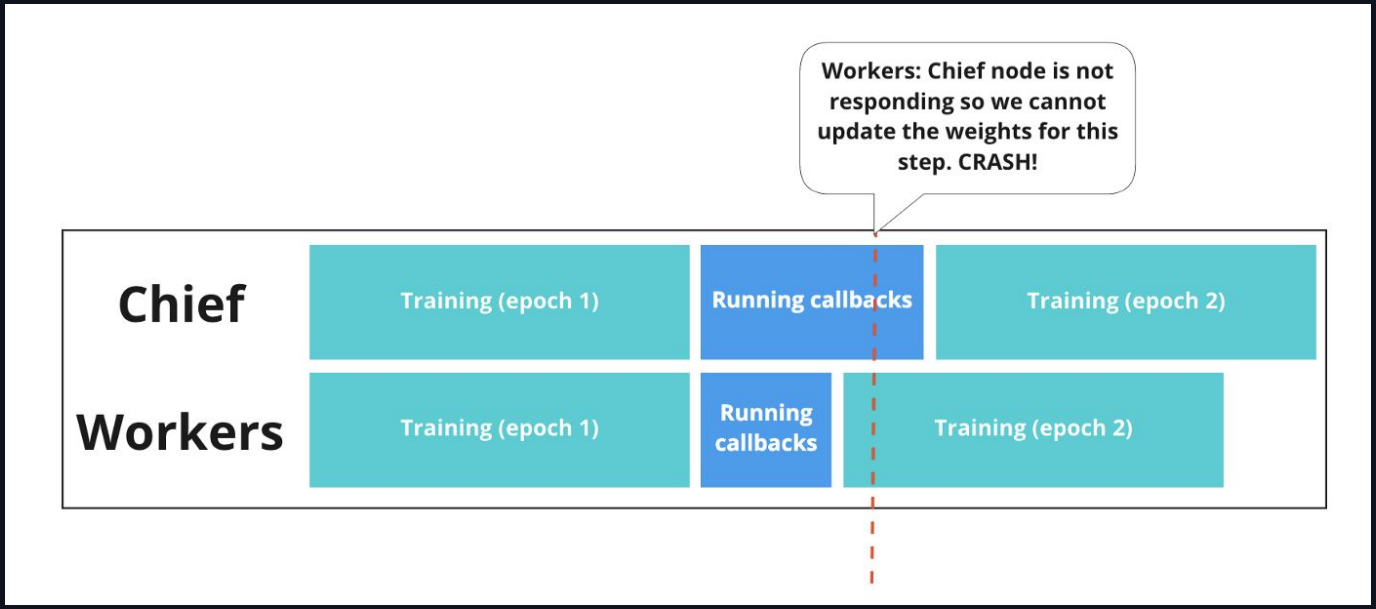
We can use ModelCheckpoint or BackupAndRestore callbacks to handle this

```
save_best_callback = ModelCheckpoint(
                        filepath=s3://persist_path, # Somewhere on
                        persistent storage
                        monitor=monitor_metric,
                        save_best_only=True)
```

Callbacks crash distributed training

Pro Tips

We noticed that if the model save callback was just added to the chief node, distributed training crashes. This was due to the fact that the workers moved on from the executing callbacks and started training for the next epoch while the chief node was still running the callbacks.



Workaround for crashing callbacks

Pro Tips

- Add a dummy callback to all workers to save the model to some temp path
- while the chief saves to a persistent store (like S3),

```
save_best_callback_dummy = ModelCheckpoint(
                                    filepath="some_random_path",
                                    monitor=monitor_metric,
                                    save_best_only=True)


callbacks = [save_best_callback if is_chief \
             else save_best_callback_dummy]
```

- This can be used for saving checkpoints to enable recovery in case of cluster failure

Batch Inference in Spark

- Use pandas UDFs to run model inference with the latest model from registry
- Pandas UDF needs an output schema
- Call the UDF in your existing inference pipeline to generate recommendations,

```
inference_data.withColumn("recs", batch_lookup_udf(f.struct(*model_cols)))
```

- If your models are pure tensorflow, we can load MLFlow models directly in spark with

```
mlflow.pyfunc.spark_udf
```

Recap

How we use this

ML Stage	*How*
Data Ingestion	Use **Spark** to read from data sources
Feature Engineering	Create data pipelines in **Spark** Write data to persistent store as **TFRecords**
Model Building	Read **TFRecords** in distributed **tensorflow** Train distributed **tensorflow** model on **Spark** Save trained model to **MLFlow**
Model Selection and Management	Pick the best model from **MLFlow** and add to **Model Registry** Use production **MLFlow** model in all downstream applications
Batch Inference	Load model from **MLFlow** model registry into a pandas UDF Add the UDF into the **Spark** inference pipeline

Flexible Data Pipelines

Mix-and-match a framework at any ML stage

Tensorflow

Read data from some source

Data processing using tf pipelines or manually from other file types

Write tf record files (or other formats supported by tensorflow) to persistent store

Read and feed tf records to models using tf.data

Store models to S3 or persistent store

Create inference pipelines (batch & streaming)

Spark

Read data from some source

Data preprocessing and feature engineering on spark

Write spark dataframes as parquet (or json, csv) to persistent store

Read and feed data to models as Spark DFs

Store models on MLflow or persistent store

Build inference pipelines

Unlocking Realtime Inference

Enables exciting use cases

Get direct feedback from members

Any use case where early intervention is valuable, like

- In-session Recommendations
- Intent and shopping pattern detection
- Location based trending products
- Advanced recommendations using Reinforcement Learning

Spark Streaming

- A stream like kinesis or kafka gives us access to real-time data as its being received from the clients
- Raw data is usually wrapped up in complicated nested json objects which needs to be properly cleaned up into structured data.
- Spark streaming lets us connect directly to the streaming endpoint and build pipelines that will clean and process the data using queries very similar to what you would do with a non-streaming DataFrame.
- In addition to this with databricks we can also use MLFlow model registry to load models and use them for inference. You can directly create a UDF from an MLFlow model which can then be used to run inference on a streaming dataframe (mlflow.pyfunc.spark_udf approach).

MLFlow Model Serving on Databricks

Quickly create an inference endpoint with a model

- Build an online inference service using MLFlow model serving
- Databricks makes it as easy as click on a button
- The new endpoint can then be used in a pipeline for inference

TODO: add example

Custom Model Serving Service

Custom TF models are not supported on MLFlow yet

- scann and tf recommenders are not supported on MLFlow currently
- Need to create a custom docker image that includes latest model
    - *Dockerfile.serving (to create the serving image is as simple as this)*

```
FROM google/tf-serving-scann:2.5.1
COPY models/ ${LOCAL_MODEL_PATH}
```

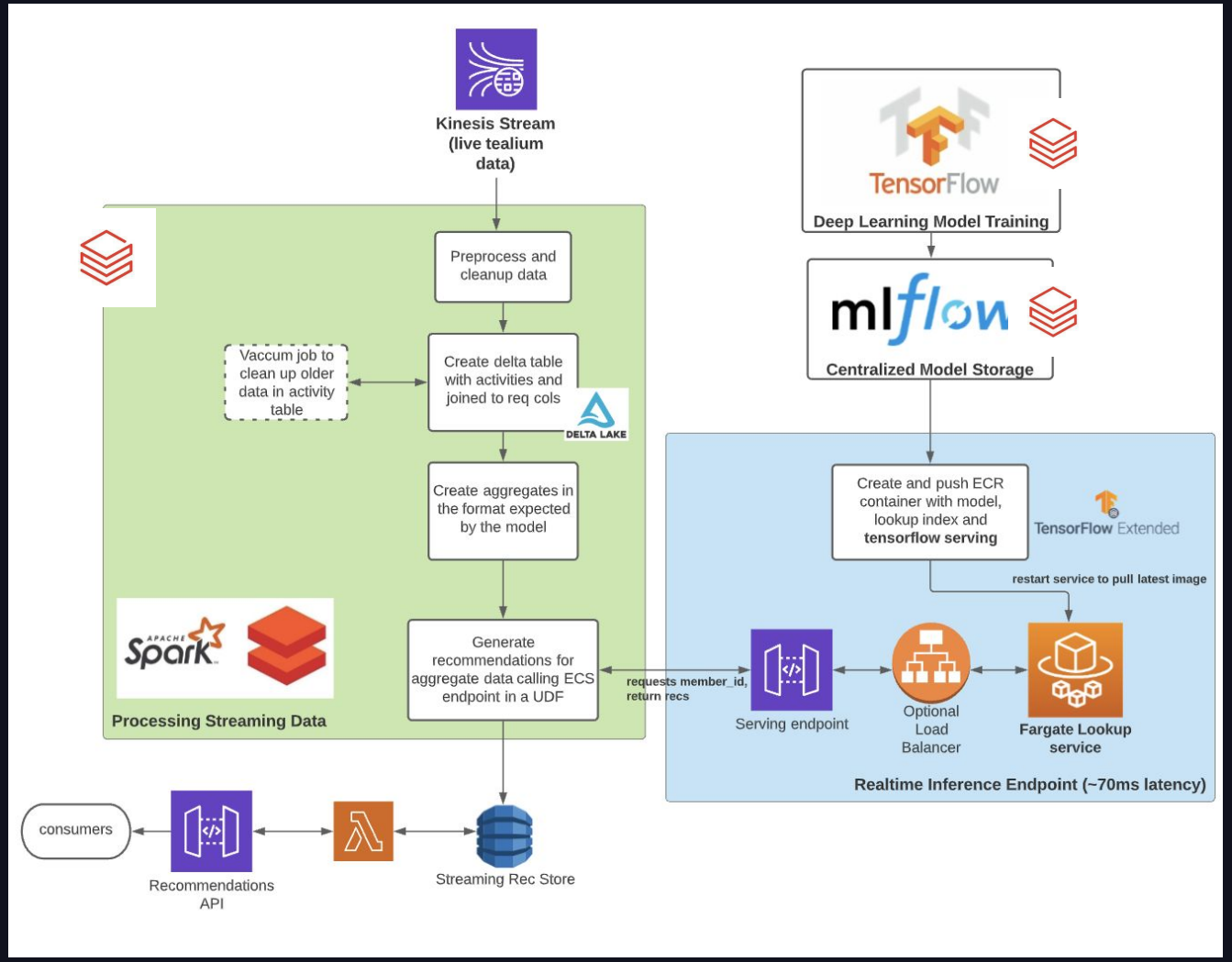
- Then we can hit serving endpoints like:

    http://<service-ip>:<serving-port>/v1/models/<model_name>:predict

Realtime Inference with Spark Streaming

3 main components:

- ➔ Realtime inference service
- ➔ Spark streaming pipelines
- ➔ Realtime Database and API



Pros of this Approach

- Great for smaller teams who want to focus on ML and less infrastructure
- Quickly transition from prototype to production models
- Flexibility to choose right framework for the use case at any stage of the ML lifecycle
- Scalable deep learning pipelines with minimal MLOps code
- Run distributed deep learning loads on Spark GPU/CPU clusters with fault tolerance
- Use all of tensorflow ecosystem and extension libraries
- Central storage, versioning, and management of models using MLflow

Open Challenges

- Distributed training does not gracefully handle fault recovery (in the case of a dead worker)
  - Parameter server strategy could be a better fit for this case
- Tuning and finding the right amount of nodes for the cluster still requires some manual effort
- Cryptic and messy Tensorflow error messages
- APIs between Keras and Tensorflow are in a state of flux

DATA+AI
SUMMIT 2022

Sample notebook at

ORGANIZED BY databricks

DATA+AI
SUMMIT 2022

Thank you

Ronny Mathew
Manager, Data Science