databricks

Machine Learning Model Deployment

Databricks Academy

Course Learning Objectives

- Define batch, pipeline, and real-time deployment methods and identify scenarios in which each type is best suited.
- Discuss the advantages and limitations of each deployment method.
- Describe MLflow’s deployment features.
- Perform batch, pipeline, and real-time inference using related Databricks features such as SDP and Model Serving.
- Describe the features and benefits of Databricks Model serving.

Prerequisites/Technical Considerations

Things to keep in mind before you work through this course

Prerequisites

1. Knowledge of fundamental concepts of regression and classification methods.
2. Familiarity with Databricks workspace and notebooks.
3. Intermediate level knowledge of Python.

Technical Considerations

1. A cluster running on **DBR ML 16.3**
2. **Unity Catalog** enabled workspace
3. **Model Serving** enabled workspace

AGENDA

1. Model Deployment Fundamentals	DEMO	LAB
Model Deployment Strategies		
Model Deployment with MLflow		
02. Batch Deployment		
Introduction to Batch Deployment	✓	✓
03. Pipeline Deployment		
Introduction to Pipeline Deployment	✓	
04. Real-time Deployment and Online Stores		
Introduction to Real-time Deployment		
Databricks Model Serving	✓	✓

databricks

Model Deployment Fundamentals

Machine Learning Model Deployment

Learning objectives

Things you'll be able to do after completing **this module**

- Define model deployment for traditional machine learning models.
- Describe batch, pipeline and real-time deployment.
- Identify scenarios in which each type of deployment is best suited.
- Compare and contrast these three types of deployment on Databricks.
- Describe MLflow's deployment features.
- Describe the concept of Model flavors.

databricks

Model Deployment Fundamentals

LECTURE

Model Deployment Strategies

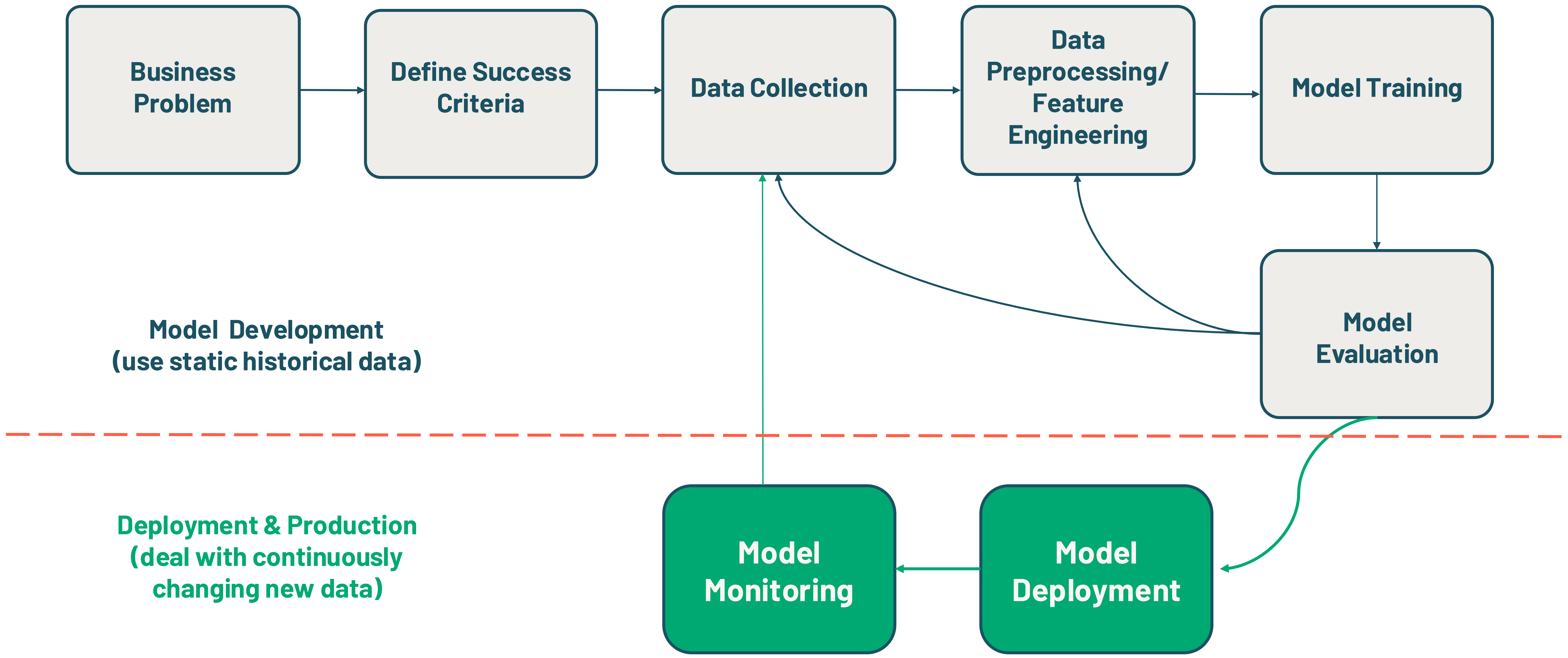
Machine Learning Model Deployment

The process of integrating a machine learning model into a **production environment**, making it accessible for end-users or other systems to **generate predictions** or insights.

(Deployment Strategies: batch, streaming, real-time, or embedded/edge)

The Machine Learning Full Lifecycle

Your Model is ready, then what?


Business Problem
Define Success Criteria
Data Collection
Data Preprocessing/ Feature Engineering
Model Training
Model Evaluation
Model Development (use static historical data)
Deployment & Production (deal with continuously changing new data)
Model Monitoring
Model Deployment

Model Deployment Modes

Comparing methods

Deployment Method	Throughput	Latency	Example Application
Batch	High	High (*hours to days*)	Periodic customer churn prediction
Streaming	Moderate	Moderate (*seconds to minutes*)	Dynamic pricing application
Real-time	Low	Low (*milliseconds*)	Recommendation systems Chatbot
Edge/Embedded	Low	Low (Dependent on device processing power)	IoT Applications. Farm sensor for detecting humidity

databricks

Model Deployment Fundamentals

LECTURE

Model Deployment with MLflow

ML*flow* Components

The four components of MLflow

We recommend using Model Registery in **Unity Catalog**

mlflow™
Trackin

Record and query experiments: code, data, config, results

mlflow™
Project
s

Packaging format for reproducible runs on any platform

mlflow™
Models

General model format that supports diverse deployment tools

mlflow™
Model Registry

Centralized and collaborative model lifecycle management

Model Deployment

ML*flow* and Model Development

What are the benefits of MLflow

Dependency & Environment Management

- Ensures that the deployment environments matches the training environment.
- Ensures that models are run consistently, regardless of where they are deployed.

Packaging Models and Code

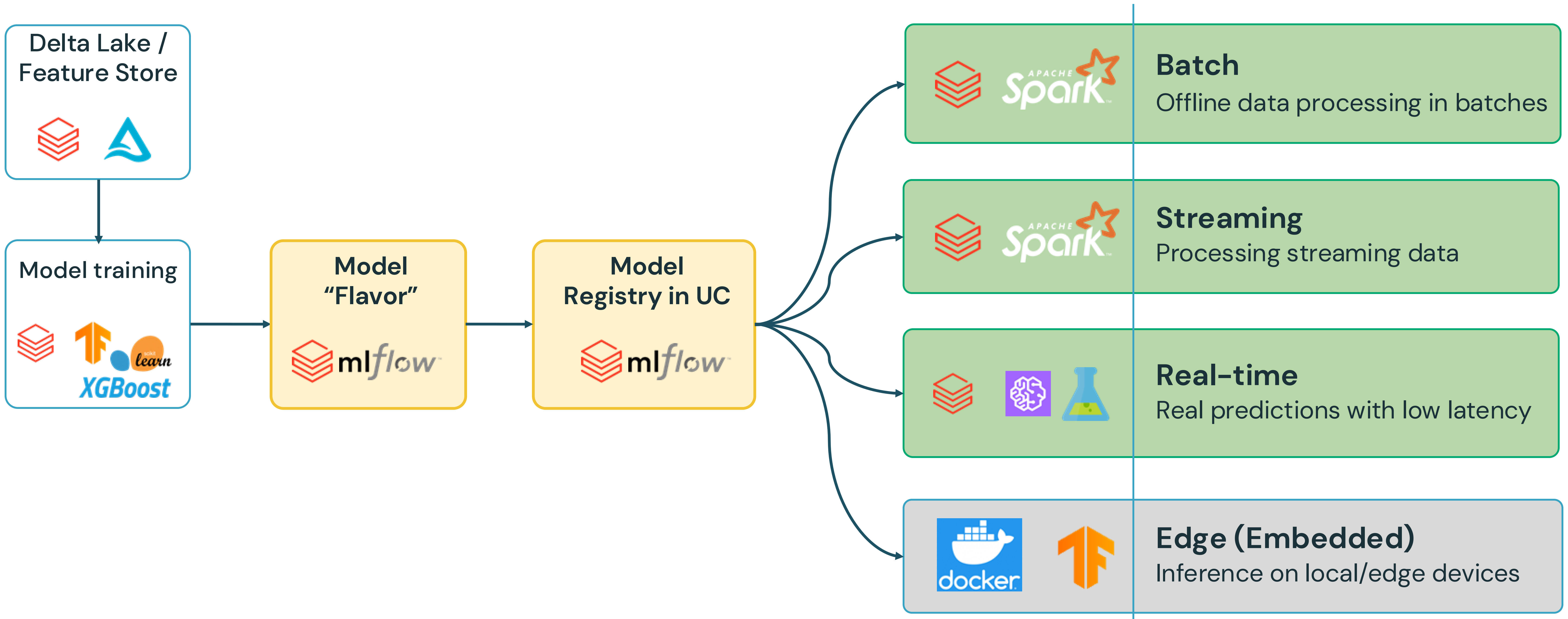
- Any code and configuration are packaged.
- Ensures model can be deployed seamlessly without any missing components

Multiple Deployment Options

- Built-in local Flask Server with MLServer
- Remote container serving:
  - AzureML
  - AWS Sagemaker
- Kubernetes clusters
- Databricks Model Serving

MLflow and Model Deployment Modes

Serving models for batch, streaming, real-time and edge inference


Delta Lake / Feature Store
Model training
XGBoost
learn
Model "Flavor"
mlflow
Model Registry in UC
mlflow
Spark
Batch
Offline data processing in batches
Spark
Streaming
Processing streaming data
Real-time
Real predictions with low latency
docker
Edge (Embedded)
Inference on local/edge devices

MLflow Model ("Flavor")

A standard format for packaging machine learning models

- Each **MLflow Model** is a directory containing arbitrary files, together with an `MLmodel` file.
- `MLModel` file can define **multiple flavors** that the model can be viewed in.
- With MLflow Models deployment tools can understand the model.
- Model file can contain **additional metadata** such as `signature`, `input example` etc.

mlflow Model Flavor

```
# Directory written by
mlflow.sklearn.save_model(model, "my_model")
my_model/
├── MLmodel
├── model.pkl
├── conda.yaml
├── python_env.yaml
 └── requirements.txt
```

```
# MLModelfile
  sklearn:
    sklearn_version: 0.19.1
    pickled_model: model.pkl
  python_function:
    loader_module: mlflow.sklearn
```

ML*flow* Model (“Flavor”)

Built-in model flavors

Python Function (`mlflow.pyfunc`):

- Serves as a **default model interface** for MLflow Python models.
- Any MLflow Python model is expected to be loadable as a python function.
- Allows you to deploy models as Python functions.
- It includes all the information necessary to **load and use a model**.
- Some functions: **`log_model`**, **`save_model`**, **`load_model`**, **`predict`**

mlflow Model Flavor

Example built-in model flavors:

- **Python function**
- Spark MLlib
- Scikit-learn
- PyTorch
- TensorFlow
- LLM models: OpenAI, LangChain, HuggingFace
- Custom falvors

MLflow Model Registry

A centralized model store

- Deploy and organize models.
- Model **versioning.**
- Model lifecycle management with **aliases**. Example: **`champion`** for the **`Production`** stage.
- Collaboration and permission management.
- Full model lineage.
- Tagging and annotations.

mlflow Model Registry

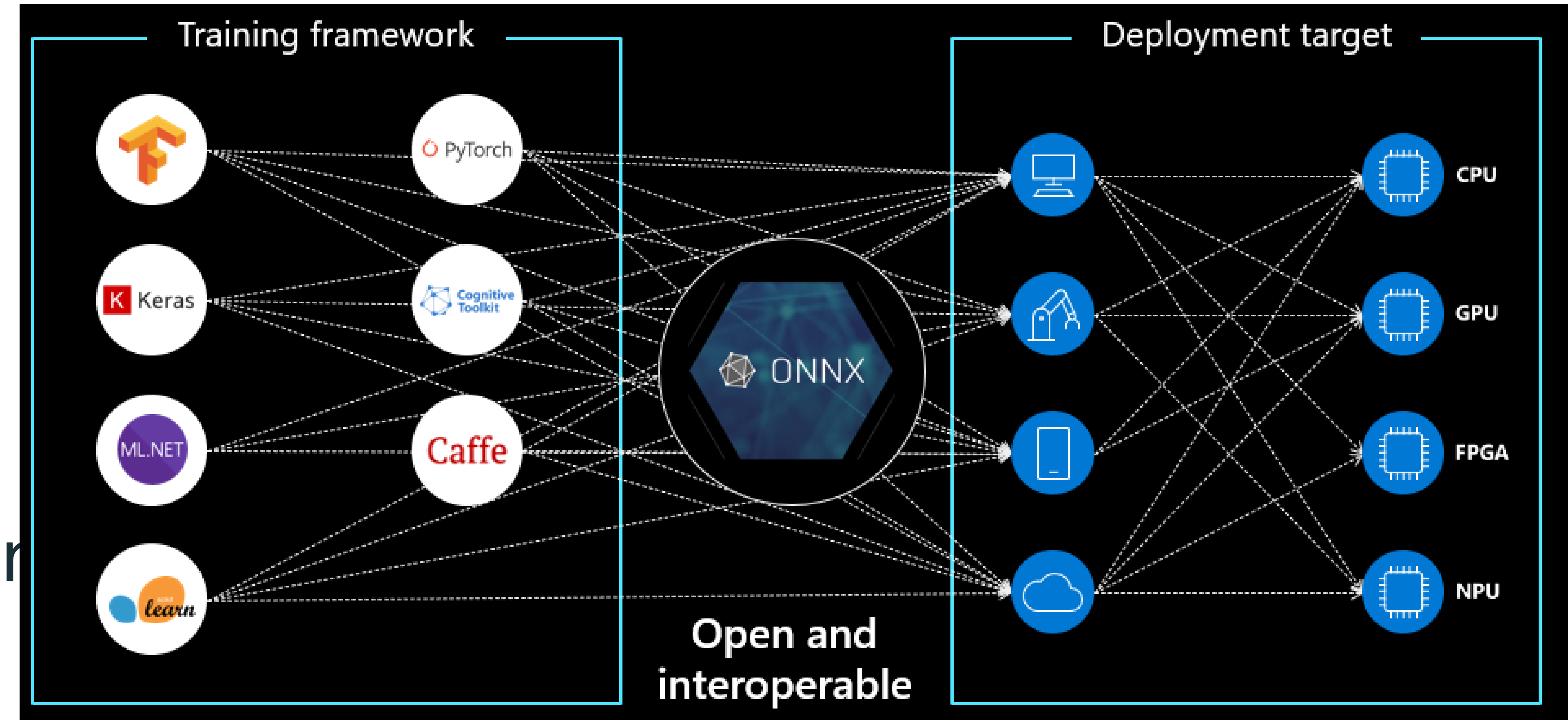
Example built-in model flavors:

- Python function
- Spark MLlib
- Scikit-learn
- PyTorch
- TensorFlow
- LLM models: OpenAI, LangChain, HuggingFace
- Custom falvors

Exporting Models in ML*flow*

Deployment environments

- In some cases, such as on the “Edge”, models need to be deployed in specific hardwares.
- Environment characteristics:
  - Offline / unreliable connectivity
  - Latency requirements
  - Security and compliance
- Export the MLflow model for inferer
  Lite.



MLflow and ONNX

A Keras conversion and logging example

- Convert model to ONNX format.
- Save ONNX model as ONNX flavor.
- Scoring: use ONNX Runtime or convert to native flavor.

Example

```
# Convert and log model as ONNX

import onnxmltools
onnx_model =
onnxmltools.convert_keras(model)
mlflow.onnx.log_model(onnx_model, "onnx-
model")
```

```
# Read and score model
import onnxruntime
session =
onnxruntime.InferenceSession(model.Serialize
ToString())
input_name = session.get_inputs()[0].name
predictions = session.run(None, {input_name:
data_np.astype(np.float32)})[0]
```

databricks

Batch Deployment

Machine Learning Model Deployment

Learning objectives

Things you'll be able to do after completing this module

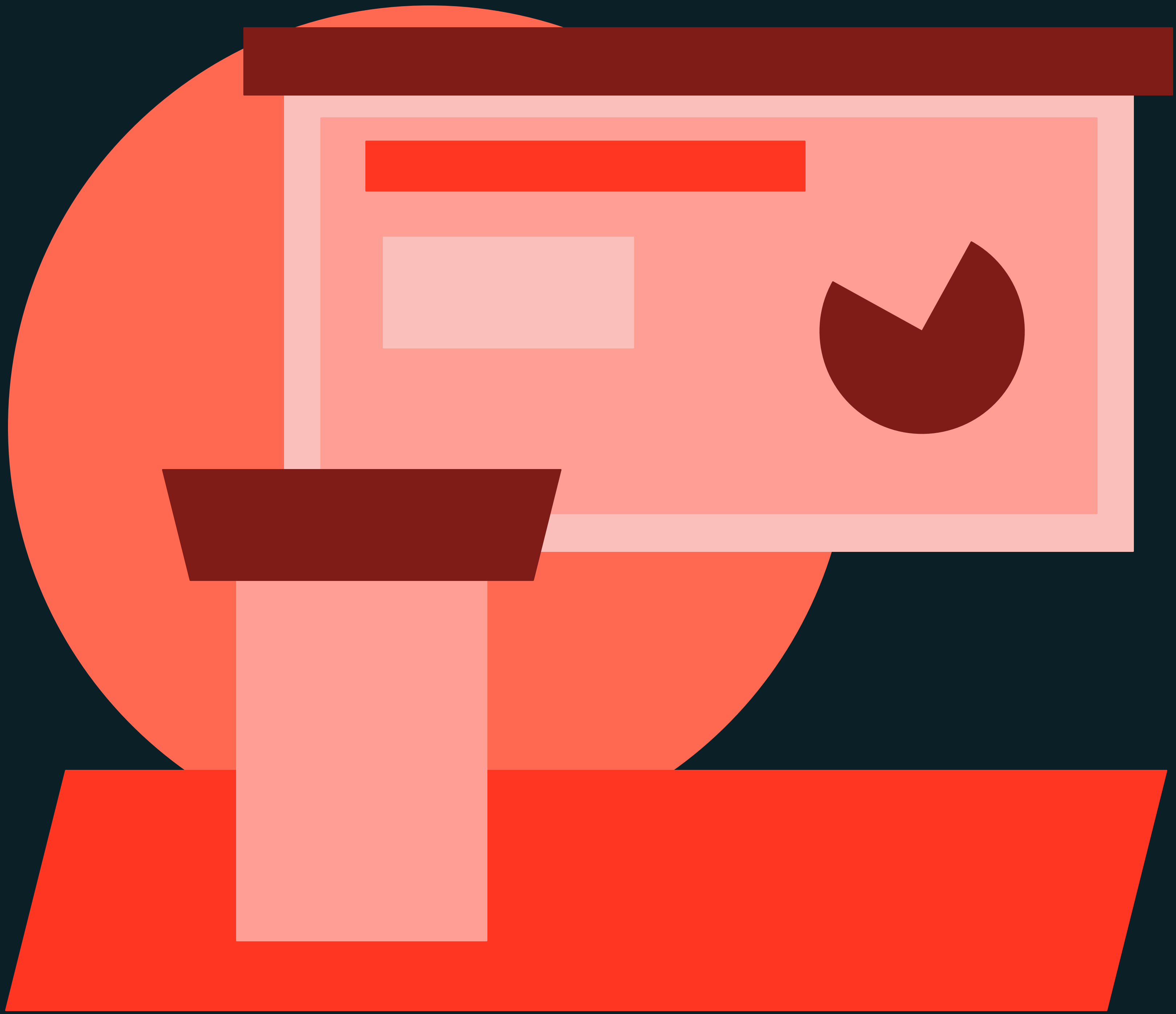
- Describe batch deployment and identify scenarios in which this method is required.
- Identify the advantages and disadvantages of deploying a model via batch processing.
- Load a logged Model Registry model using pyfunc.
- Compute predictions using pyfunc APIs.
- Perform batch inference using Feature Store's score_batch.
- Connect Delta Lake's optimization techniques with batch deployment scenarios.

databricks

Batch Deployment

LECTURE

Introduction to Batch Deployment



Batch Deployment

- Batch processing generates predictions on a **regular schedule** and writes the results out to persistent storage to be consumed downstream (i.e. ad-hoc BI).
- Batch deployment is **the most common deployment strategy**.
- Ideal for cases when:
    - Immediate predictions is not necessary
    - Predictions can be made in batch fashion.
    - Number/volume of (new) records/observations to predict is large.
    - Pace at which input/records change or is received is > 30mins.

Batch Deployment

A typical batch deployment use case

- **Description**: Identifying potential churners in a subscription-based service.
- **Scenario**:
  - A machine learning model trained on historical data, including customer usage patterns, interactions, and subscription details, is collected.
  - The model is saved for later use.
  - At the end of each day/week/month, the collected data for that period is processed in a batch, and churn predictions are generated.

Batch Deployment

Advantages and limitations

Advantages

- **Cheapest** deployment method.
- **Ease** of implementation.
- Efficient per data point.
- Can handle **high volume of data**.

Limitations

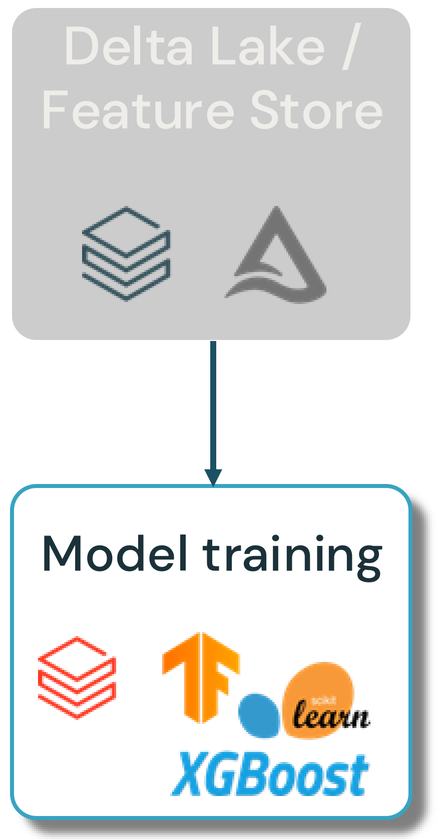
- High **latency**.
- **Stale data**.
- Not suitable for dynamic or rapidly changing data.
- Not suitable for streaming data or real-time applications.

Batch Deployment

A typical batch model deployment workflow

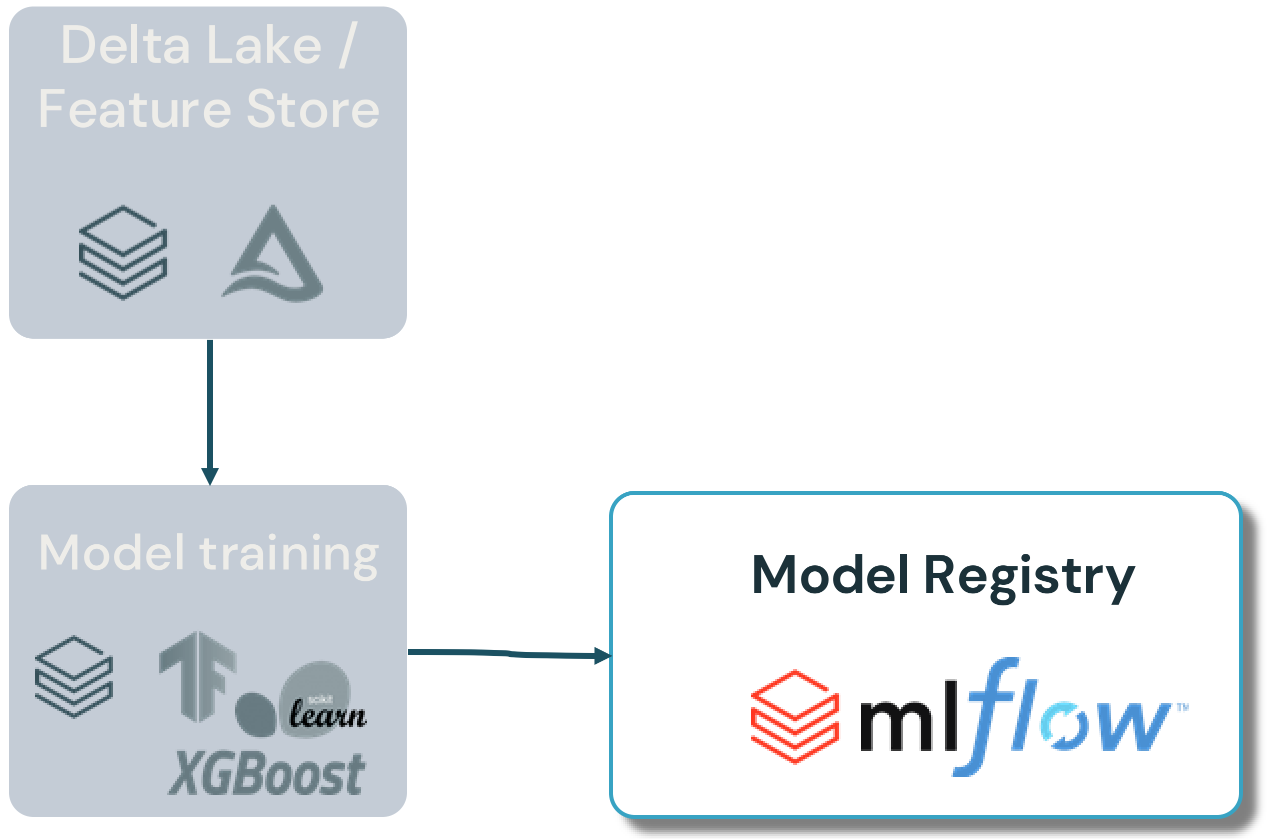
Batch Deployment

A typical batch model deployment workflow



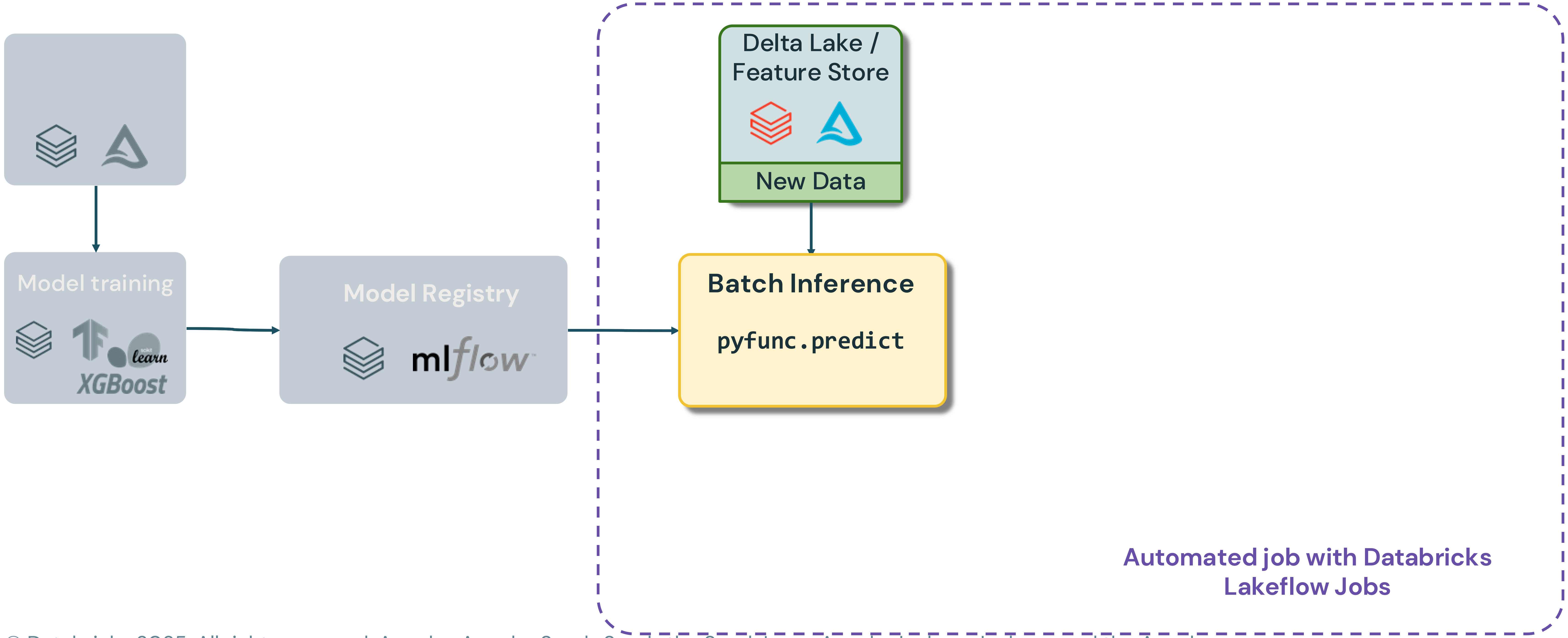
Batch Deployment

A typical batch model deployment workflow



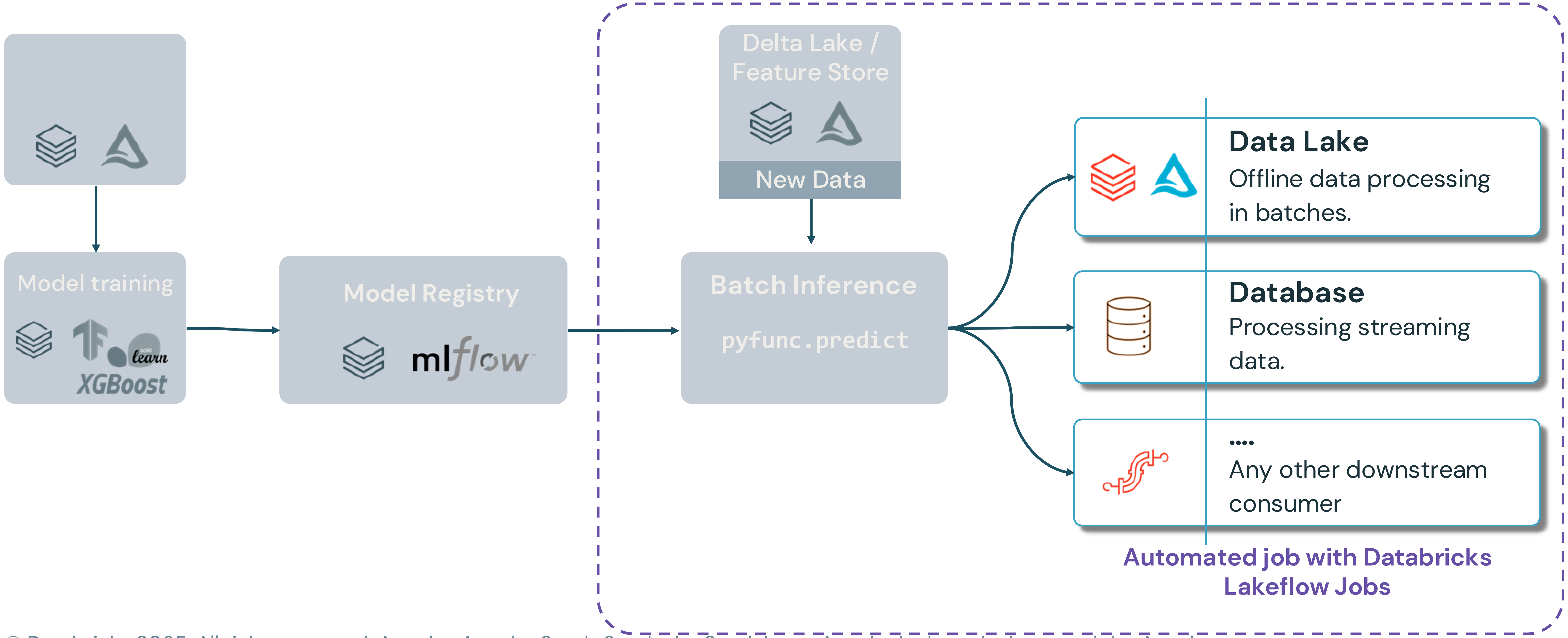
Batch Deployment

A typical batch model deployment workflow



Batch Deployment

A typical batch model deployment workflow

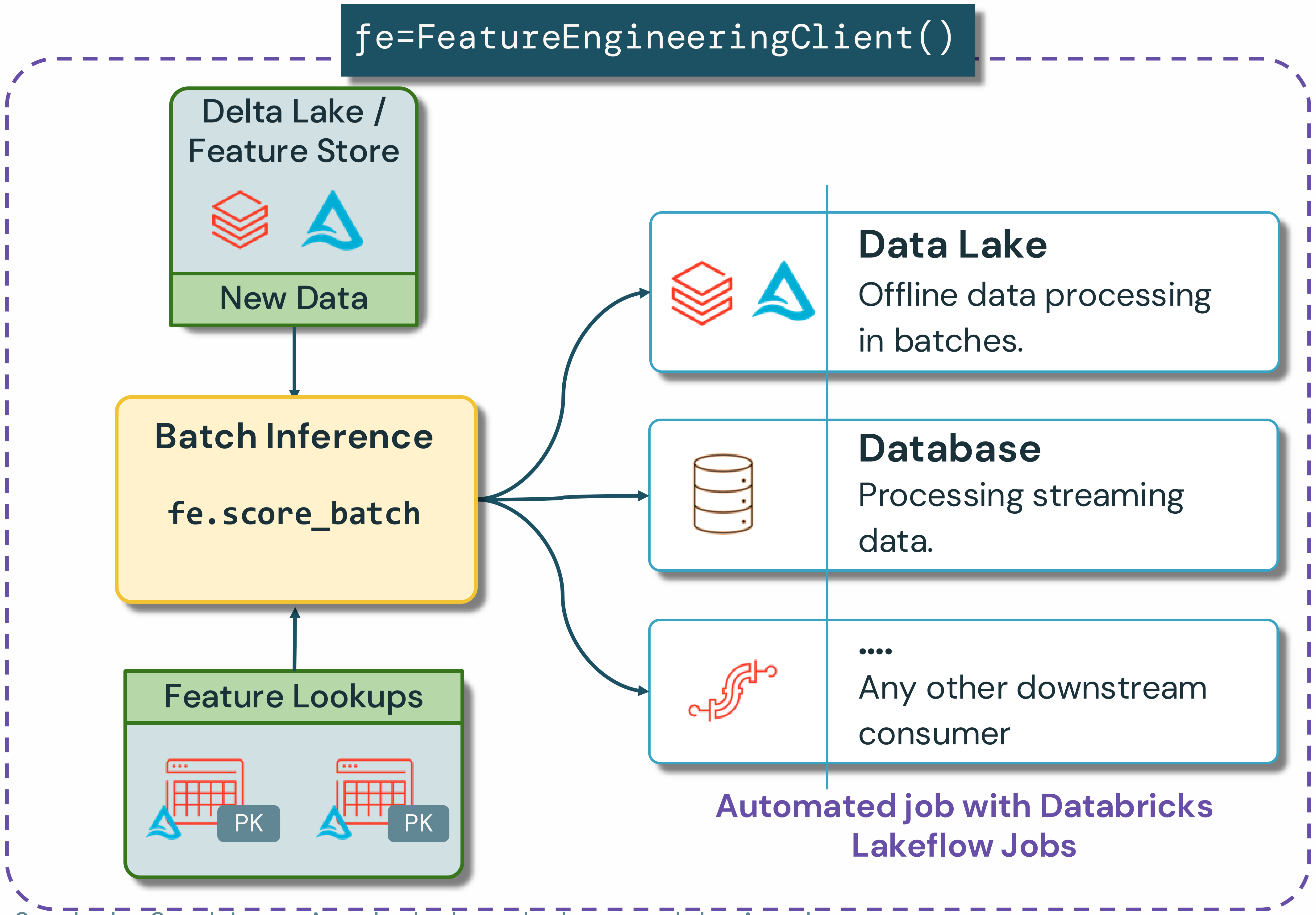

Delta Lake /
Feature Store
New Data
Model training
XGBoost
Model Registry
mlflow
Batch Inference
pyfunc.predict
Data Lake
Offline data processing in batches.
Database
Processing streaming data.
....
Any other downstream consumer
Automated job with Databricks Lakeflow Jobs

Batch Deployment

Offline Feature Tables and Feature Lookups

Offline Feature Tables

- Delta table with a primary key
- Train and serve a model for batch inference with `FeatureEngineeringClient()`
- Supports `FeatureLookup` and `FeatureFunction` for batch model inference
- Not optimized for real-time inference due to latency

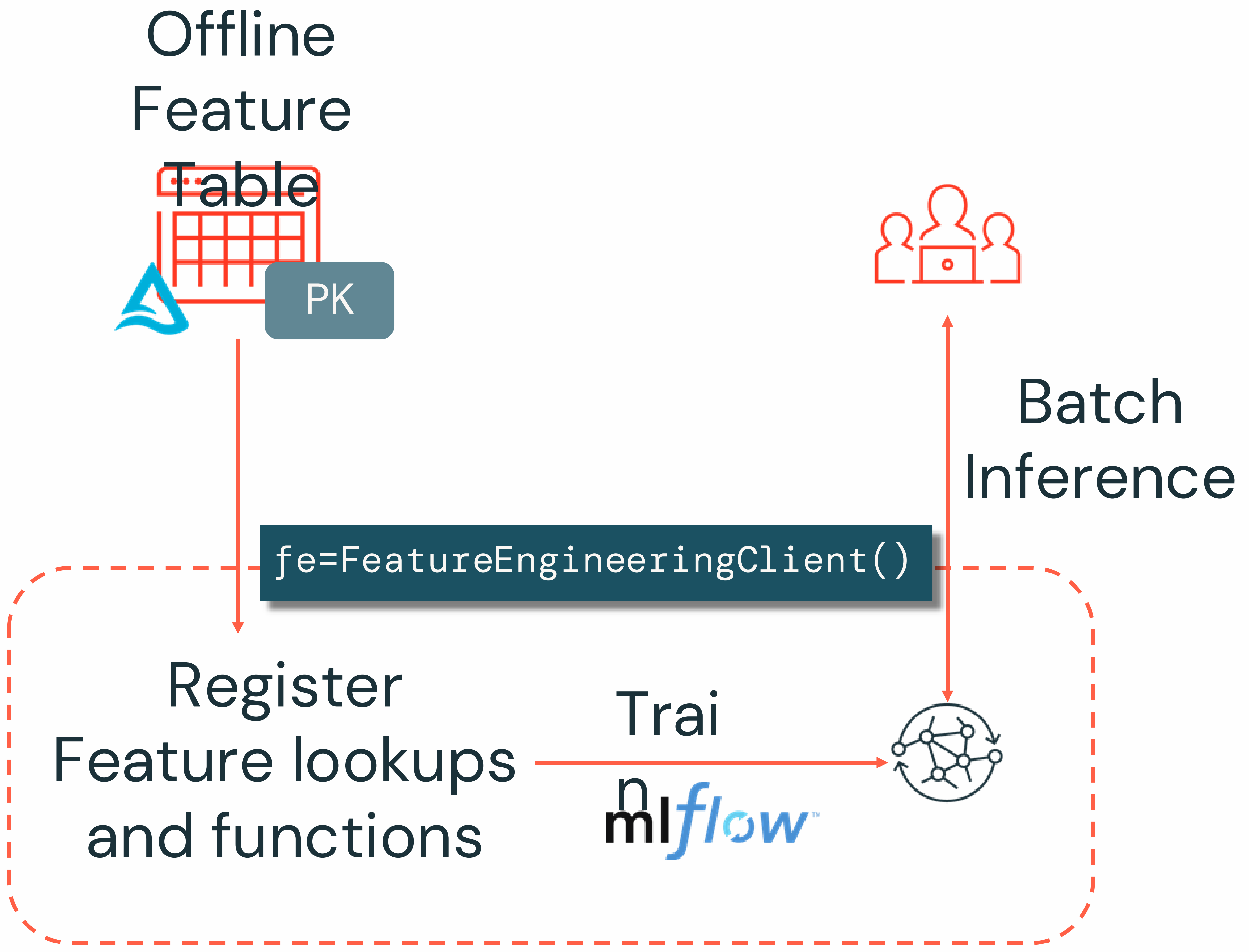

fe=FeatureEngineeringClient()
Delta Lake / Feature Store
New Data
Batch Inference
fe.score_batch
Feature Lookups
PK
PK
Data Lake
Offline data processing in batches.
Database
Processing streaming data.
....
Any other downstream consumer
Automated job with Databricks Lakeflow Jobs

Batch Deployment

Offline Feature Tables

Offline Feature Tables

- Delta table with a primary key
- Train and serve a model for either batch or real-time inference with model serving
- Supports FeatureLookup and FeatureFunction for batch model inference
- Not optimized for real-time inference due to latency


Offline Feature Table
PK
fe=FeatureEngineeringClient()
Register Feature lookups and functions
Train
mlflow
Batch Inference

Performance & Optimization In Batch Deployment

Liquid Clustering

Delta Lake's optimization and performance features

- **Liquid clustering replaces table partitioning and ZORDER** to simplify data layout decisions and optimize query performance.
- Databricks **recommends** liquid clustering for all new Delta tables.
- Databricks client manages all layout and optimization operations for data in your table.

```
CREATE TABLE table1(col0 int,
col1 string)
USING DELTA
CLUSTER BY (col0);
----------------------------------
```

Requires DBR 13.3+

Predictive Optimization

Delta Lake’s optimization and performance features

- Databricks **automatically identifies** tables that would benefit from maintenance operations and **runs them for the user**.
- Automatically run operations;
  - `OPTIMIZE` → Improves query performance by optimizing file sizes
  - `VACUUM` → deleting data files no longer referenced by the table
- You must enable predictive optimization at the account level.

Enable predictive optimization (catalog):

```
ALTER CATALOG catalog1
ENABLE PREDICTIVE OPTIMIZATION;
```

Enable predictive optimization (schema):

```
ALTER SCHEMA schema1
ENABLE PREDICTIVE OPTIMIZATION;
```

Optimize

Delta Lake’s optimization and performance features

- Improve the speed of read queries from a table by coalescing small files into larger ones.
- Databricks **recommends using predictive optimization** to automatically run OPTIMIZE for Delta tables.

Trigger **Clustering** Manually:

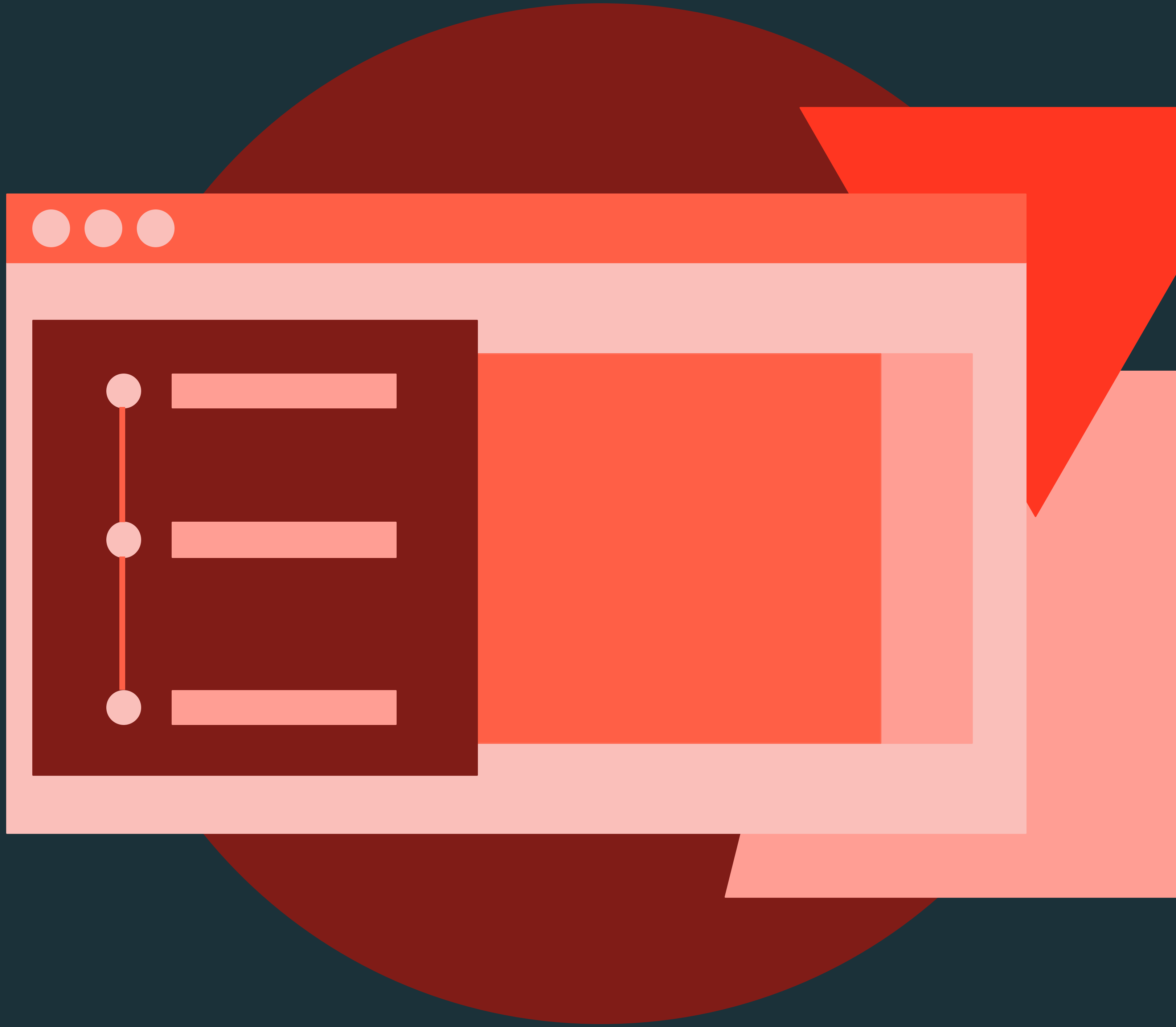
```
OPTIMIZE table1;
```

Used for Z-Ordering;

```
OPTIMIZE table1;
```

DEMONSTRATION

Batch Deployment



Demo

Outline

What we'll cover:

- Data preparation
- Batch deployment **without Feature Store**
  - Fit the model and register the model with UC
  - Use the model for batch inference
- Batch deployment **with Feature Store**
  - Fit the model and register the model with UC
  - Use the model for batch inference
- Performance considerations

Batch Deployment

LAB EXERCISE

Batch Deployment

Lab

Outline

What you'll do:

- Task 1: Load the dataset
- Task 2: Batch inference with feature table
- Task 3: Enable Liquid Clustering on a Delta table

databricks

Pipeline Deployment

Machine Learning Model Deployment

Learning objectives

Things you’ll be able to do after completing this module

- Describe pipeline deployment and identify scenarios in which this method is required.
- Describe Lakeflow Spark Declarative Pipeliness as a tool that can be used to develop and manage inference pipelines.
- Develop a simple Lakeflow Spark Declarative Pipeliness pipeline that performs streaming based inference in its final step.

databricks

Pipeline Deployment

LECTURE

Introduction to Pipeline Deployment

Pipeline / Streaming Deployment

- Streaming deployment involves deploying machine learning models that **can handle real-time or streaming data**.
- Instead of processing data in batches, these models can **continuously perform inference as data arrives**, providing low-latency and continuous insights.
- It often involves setting up a system that continuously receives and processes incoming data, makes predictions in real-time.

Pipeline Deployment

Advantages and limitations

Advantages

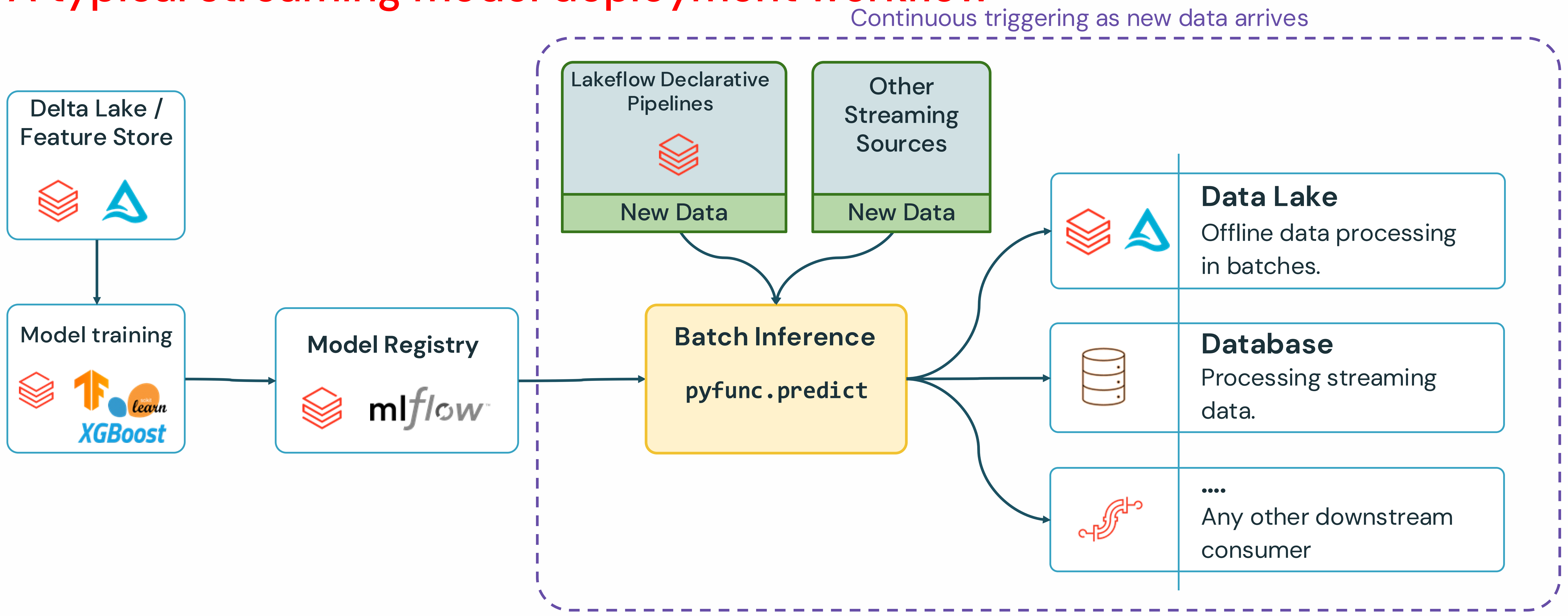
- **Lower latency**.
- Generate predictions and **act sooner**.
- **Not suitable** for applications that require **near real-time** predictions.
- **Event-driven** architecture; enable systems to adopt quickly

Limitations

- More **costly** than batch solution.
- More **complicated** to develop, maintain and monitor compared to batch solution.
- Less throughput compared to batch solution.
- **Resource intensiveness**; computational power to process data

Pipeline Deployment

A typical streaming model deployment workflow


Continuous triggering as new data arrives
Delta Lake / Feature Store
Model training
XGBoost
learn
Model Registry
mlflow
Lakeflow Declarative Pipelines
New Data
Other Streaming Sources
New Data
Batch Inference
pyfunc.predict
Data Lake
Offline data processing in batches.
Database
Processing streaming data.
....
Any other downstream consumer

Lakeflow Declarative Pipelines

Lakeflow Declarative Pipelines

The best way to do ETL on the Databricks Data Intelligence Platform

```
CREATE STREAMING TABLE raw_data
AS SELECT *
FROM cloud_files ("/raw_data",
"json")

CREATE MATERIALIZED VIEW clean_data
AS SELECT …
FROM raw_data
```

Accelerate ETL development
Declare **SQL or Python** and Lakeflow Declarative Pipelines automatically
orchestrates the DAG, handles retries, changing data

Automatically manage your infrastructure
Automates complex tedious activities like **recovery, auto-scaling, and performance optimization**

Ensure high data quality
Deliver reliable data with built-in **quality controls, testing, monitoring, and enforcement**

Unify batch and streaming
Get the simplicity of SQL with freshness
of streaming with one **unified API**

What is a Live Table

Live Tables are materialized views for the lakehouse.

A live table is:

- Defined by a SQL query
- Created and kept up-to-date by a pipeline

```
CREATE OR REFRESH LIVE TABLE report
AS SELECT sum(profit)
FROM prod.sales
```

Live tables provides tools to:

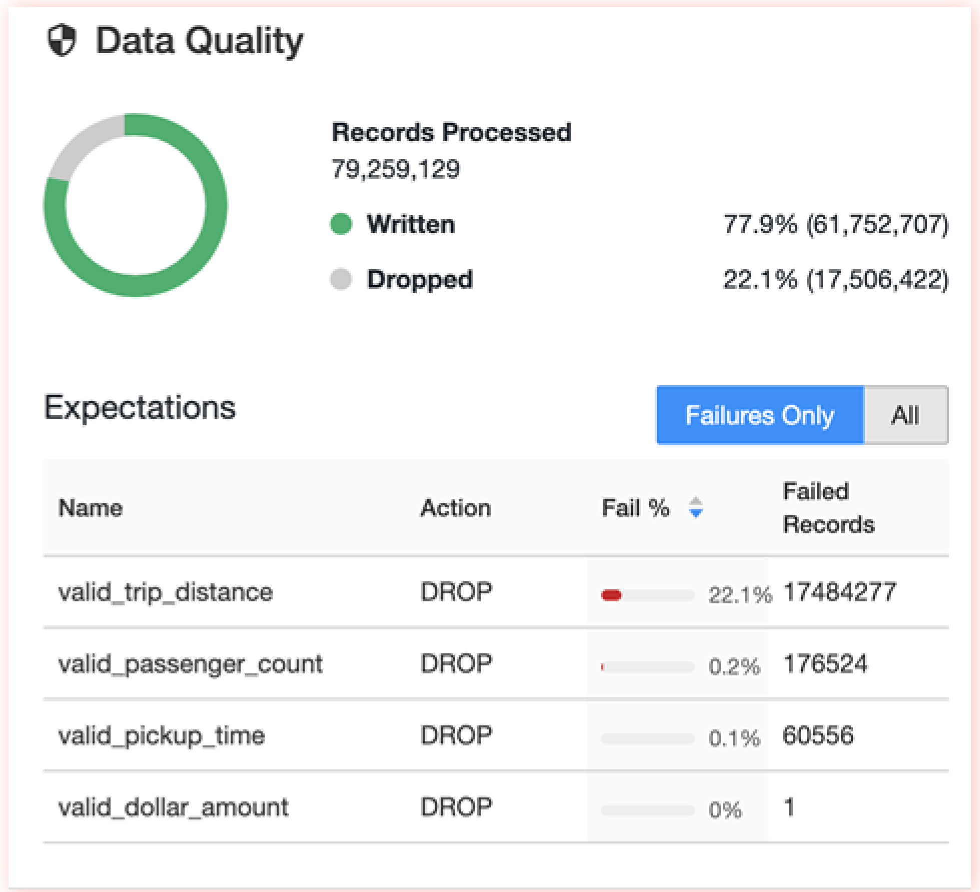
- Manage dependencies
- Control quality
- Automate operations
- Simplify collaboration
- Save costs
- Reduce latency

Data Quality Validation and Monitoring

Lakeflow Declarative Pipeline Features

- Define **data quality and integrity controls** within the pipeline with data expectations
- Address data quality errors with flexible policies: **fail, drop, alert, quarantine**(future)
- All data pipeline runs and **quality metrics** are captured, tracked and reported

```
/* Stage 1: Bronze Table drop invalid rows */
CREATE STREAMING LIVE TABLE fire_account_bronze AS
( CONSTRAINT valid_account_open_dt EXPECT (acconut_dt is not null
and (account_close_dt > account_open_dt)) ON VIOLATION DROP ROW
COMMENT "Bronze table with valid account ids"
SELECT * FROM fire_account_raw ...
```


Data Quality

Records Processed
79,259,129

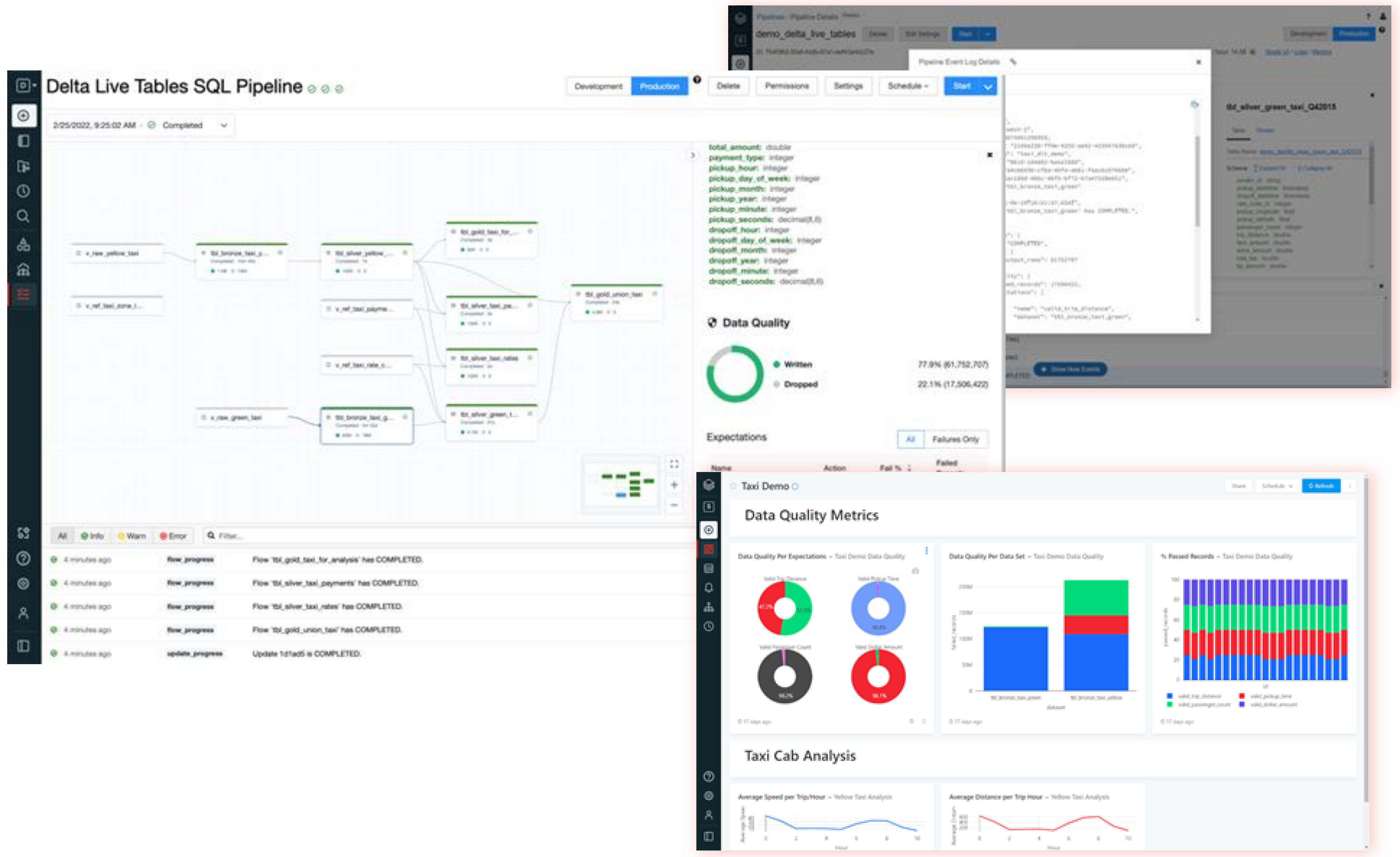
● Written	77.9% (61,752,707)
● Dropped	22.1% (17,506,422)

Expectations

Name	Action	Fail %	Failed Records
valid_trip_distance	DROP	22.1%	17484277
valid_passenger_count	DROP	0.2%	176524
valid_pickup_time	DROP	0.1%	60556
valid_dollar_amount	DROP	0%	1

Data Pipeline Observability

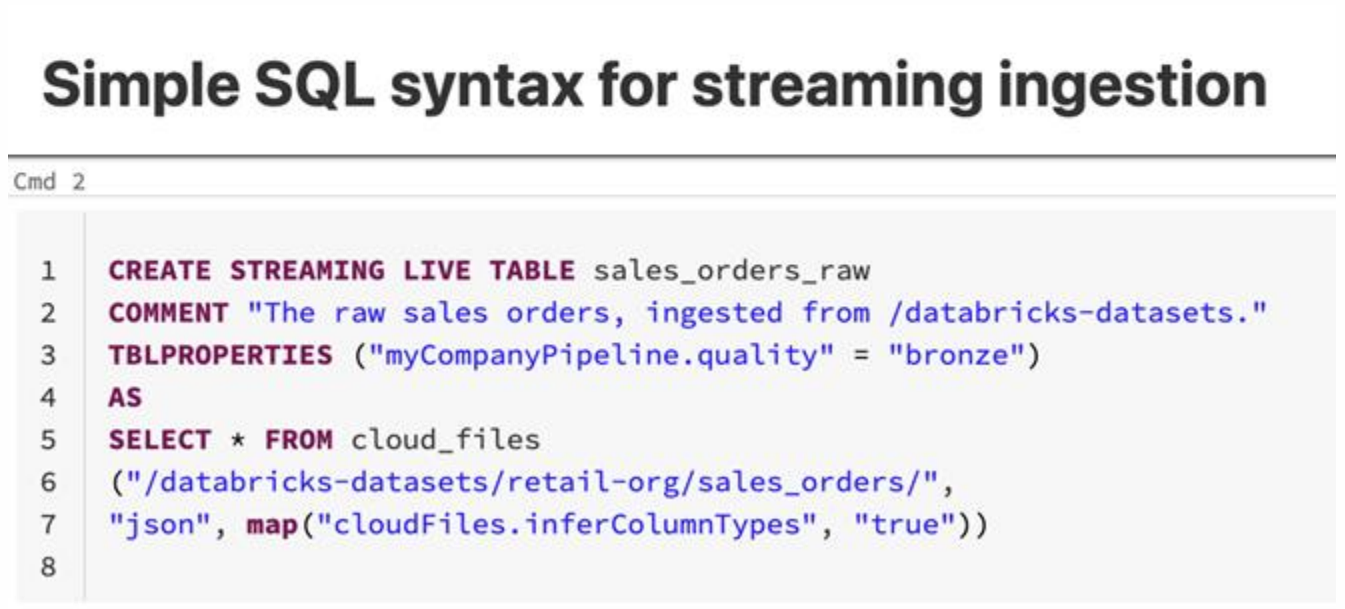
Lakeflow Declarative Pipeline Features



- High-quality, high-fidelity **lineage diagram** that provides visibility into how data flows for impact analysis
- Granular **logging** for operational, governance, quality and status of the data pipeline at a row level
- Continuously **monitor** data pipeline Lakeflow Jobs to ensure continued operation
- Notifications using Databricks SQL

Continuous or Scheduled Data Ingestion

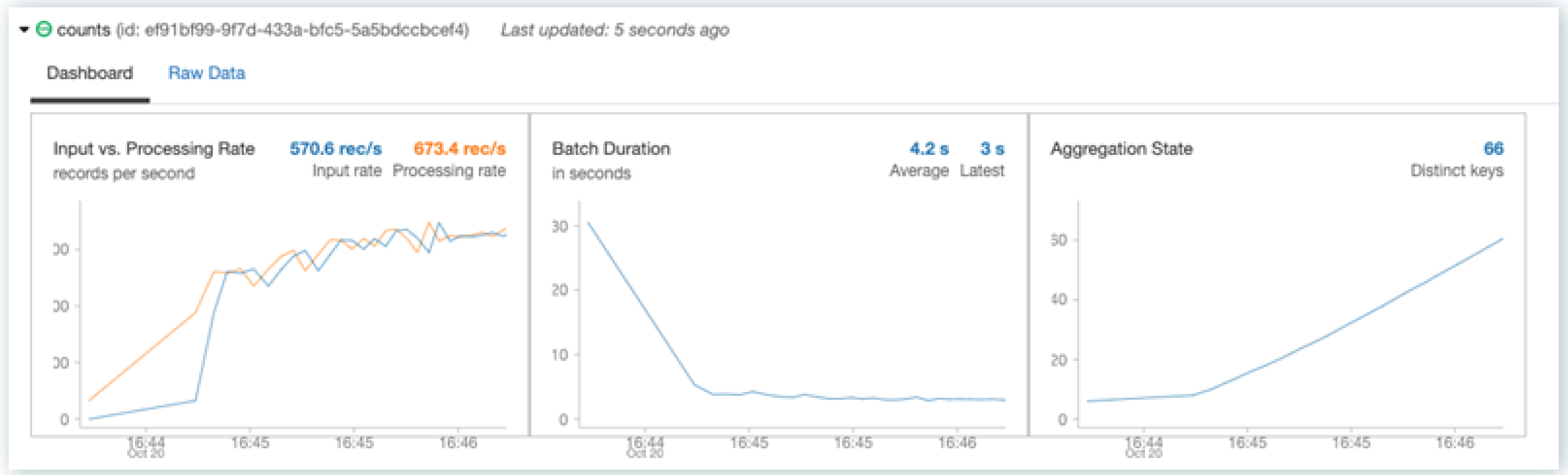
Lakeflow Declarative Pipeline Features



Simple SQL syntax for streaming ingestion

Cmd 2

```
CREATE STREAMING LIVE TABLE sales_orders_raw
COMMENT "The raw sales orders, ingested from /databricks-datasets."
TBLPROPERTIES ("myCompanyPipeline.quality" = "bronze")
AS
SELECT * FROM cloud_files
("/databricks-datasets/retail-org/sales_orders/",
"json", map("cloudFiles.inferColumnTypes", "true"))
```



counts (id: ef91bf99-9f7d-433a-bfc5-5a5bdccbcef4) *Last updated: 5 seconds ago*

- **Incrementally** and efficiently **process new data files** as they arrive in cloud storage using Auto Loader
- Automatically **infer schema** of incoming files or superimpose what you know with Schema Hints
- Automatic **schema evolution**

Schema Evolution ✓ JSON ✓ CSV ✓ AVRO ✓ PARQUET

Creating Your First Live Table Pipeline

SQL to Lakeflow Declarative Pipelines in three easy steps…

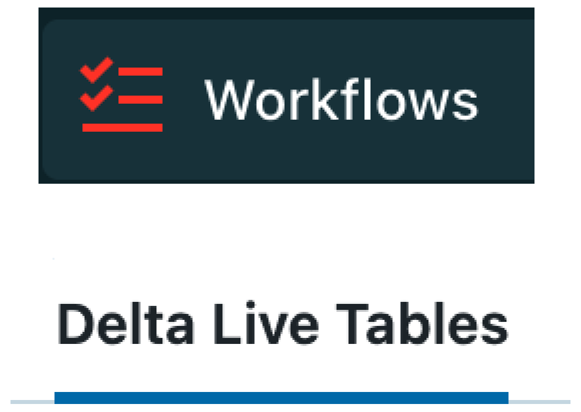
Write create live table

- Table definitions are written (**but not run**) in notebooks
- Databricks Repos allow you to **version control** your table definitions.

```sql
CREATE LIVE TABLE daily_stats
AS SELECT sum(rev) - sum(costs) AS profits
FROM prod_data.transactions
GROUP BY day
```

Create a pipeline

- A Pipeline picks **one or more notebooks** of table definitions, as well as any **configuration** required.



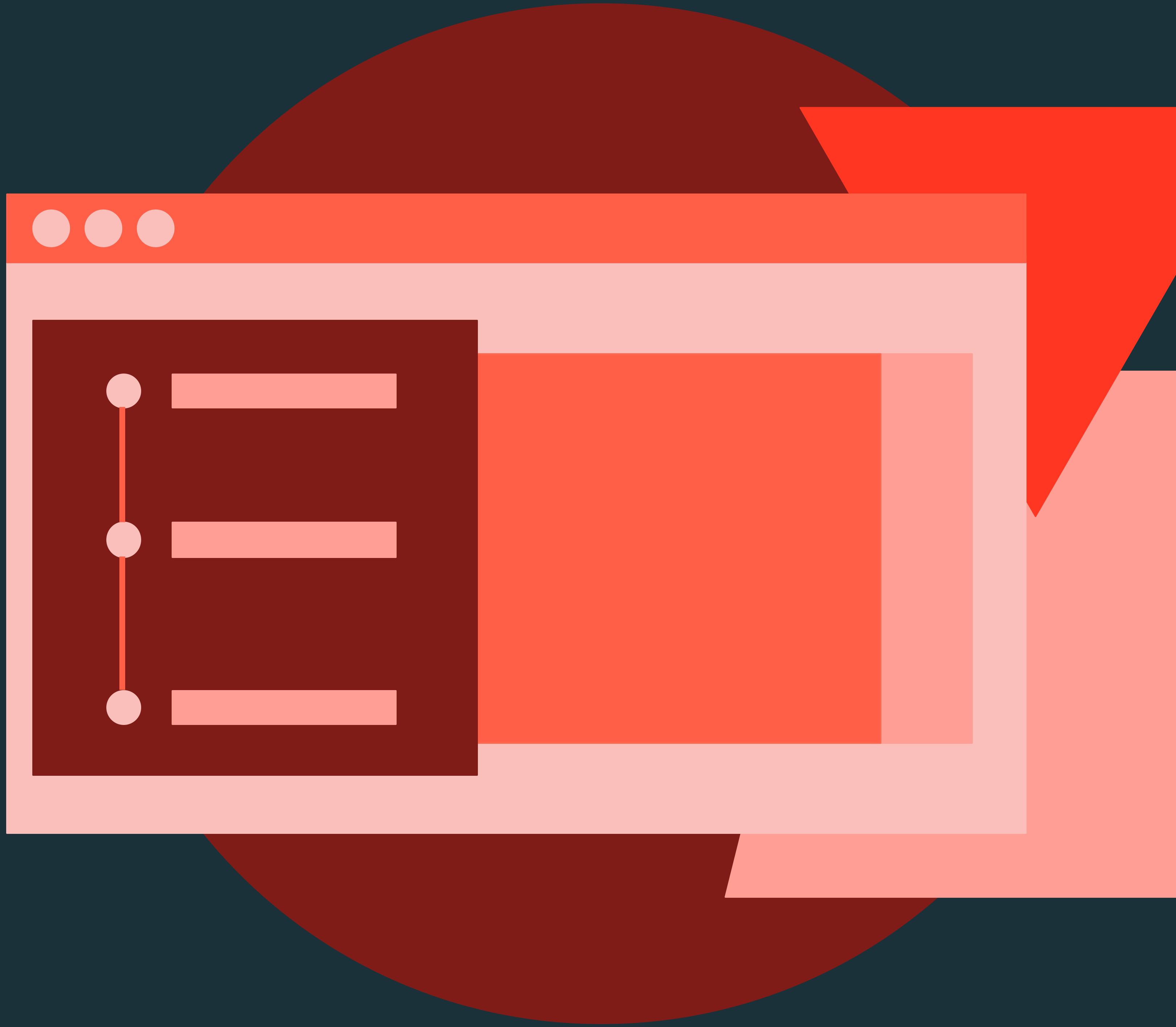
Click start

- Lakeflow Declarative Pipelines will **create or update** all the tables in the pipelines.

Pipeline Deployment

DEMONSTRATION

Pipeline Deployment



Demo

Outline

What we'll cover:

- Data preparation
- Model preparation
- Configure a pipeline to run batch inference
    - Config variables
    - Create a SDP pipeline

Real-time Deployment and Online Stores

Machine Learning Model Deployment

Learning objectives

Things you'll be able to do after completing this module

- Describe real-time deployment and identify scenarios in which this method is required.
- Discuss challenges of real-time deployment systems and how Databricks model serving address these challenges.
- Describe features of Databricks Model Serving.
- Serve a model with model serving using the UI and the API.
- Serve multiple model versions of a model to a serving endpoint by splitting the incoming traffic.
- Serve custom model with Databricks Model Serving.

databricks

Real-time Deployment and Online Stores

LECTURE

Introduction to Real-time Deployment

Real-time Deployment

- The process of deploying and serving machine learning models in a production environment where **predictions are generated instantly** in response to incoming data or requests.
- Crucial for applications that require **low-latency responses**, such as fraud detection, autonomous systems, and other time-sensitive tasks.

With the emergence of new Gen AI applications, this deployment method is becoming increasingly common, especially as large language models need to be served in real-time.

Real-time Deployment and Online Stores

LECTURE

Introduction to Real-time Deployment

Real-time Deployment

- The process of deploying and serving machine learning models in a production environment where **predictions are generated instantly** in response to incoming data or requests.
- Crucial for applications that require **low-latency responses**, such as fraud detection, autonomous systems, and other time-sensitive tasks.

With the emergence of new Gen AI applications, this deployment method is becoming increasingly common, especially as large language models need to be served in real-time.

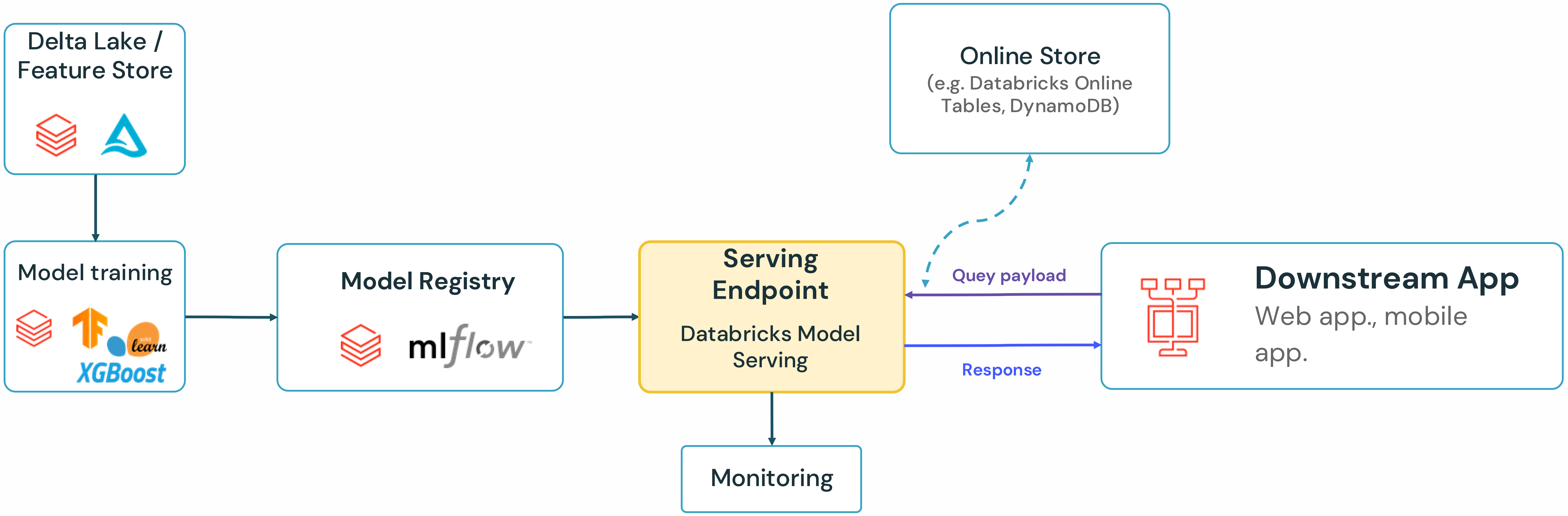
Real-time Deployment

A typical real-time deployment use case

- **Description**: Real-time Fraud Detection in Online Banking
- **Scenario**:
  - A pre-trained machine learning model, specifically designed for fraud detection, is deployed in the real-time environment.
  - The model provides immediate predictions over a REST API.
  - An immediate decision is made to flag the transaction.
- **Requirements**: Low latency, immediate action, 24/7 uptime, continuous monitoring

Real-time Deployment

A typical batch model deployment workflow



Challenges with building Real-time ML Systems

Most ML models don't get into production

ML infrastructure is hard

Real-time ML systems require fast and scalable serving infrastructure, which is costly to build and maintain

Challenges with building Real-time ML Systems

Most ML models don't get into production

ML infrastructure is hard

Real-time ML systems require fast and scalable serving infrastructure, which is costly to build and maintain

Deploying real time models needs disparate tools

Data teams use diverse tools to develop models

Customers use separate platforms for data, ML and Serving, adding complexity and cost

Challenges with building Real-time ML Systems

Most ML models don't get into production

ML infrastructure is hard

Real-time ML systems require fast and scalable serving infrastructure, which is costly to build and maintain

Deploying real time models needs disparate tools

Data teams use diverse tools to develop models

Customers use separate platforms for data, ML and Serving, adding complexity and cost

Operating production ML requires expert resources

Steep learning curve of deployment tools.

Model deployment is bottlenecked by limited engineering resources, limiting the ability to scale AI

Databricks Model Serving

Integrate your model into your websites and applications as an API

Production-Grade Serving

Highly available, low latency, scalable serving that works for small and large Lakeflow Jobs

Accelerate deployments with Lakehouse-Unified Serving

Automatic feature lookups, monitoring and unified governance that **automates deployment** and reduce errors

Simplified Deployment

Simple and flexible deployment through **UI** or **API**

databricks

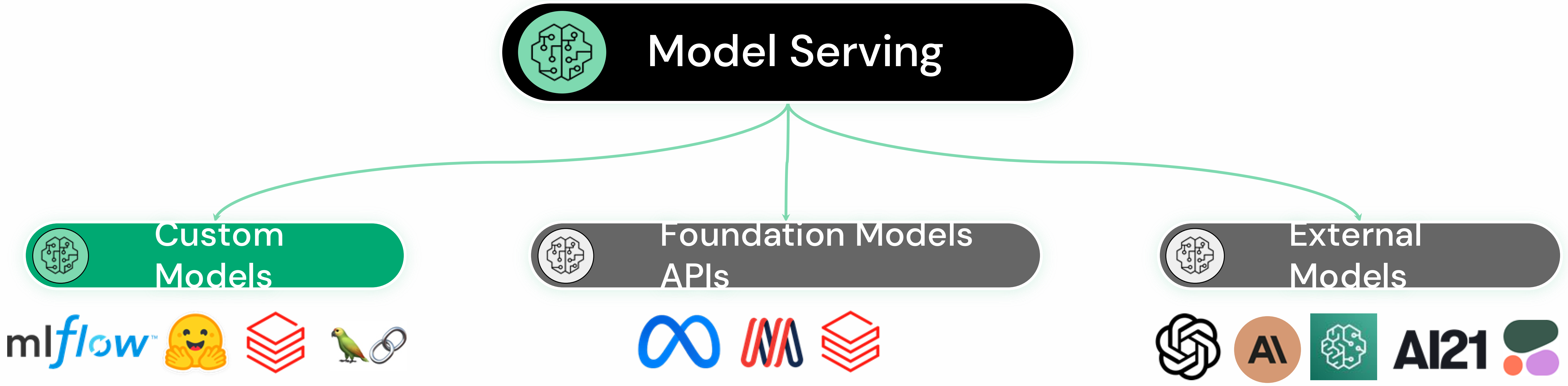
Real-time Deployment and Online Stores

LECTURE

Databricks Model Serving

Databricks Model Serving

Unified UI, API & SDK for managing all types of AI Models



Deploy any model as a REST API with Serverless compute, managed via MLflow.

CPU and GPU. Integration with Feature Store and Vector Search.

Databricks curates top Foundation Models and provides them behind simple APIs.

You can start experimentation immediately, without setting up serving yourself.

Govern external models and APIs.

This provides the governance of MLflow AI Gateway, plus the monitoring and payload logging of traditional Databricks Model Serving.

Core Features of Model Serving

Support real-time production ML Lakeflow Jobs

Real Time

- **Low overhead latency**: <100ms
- **Throughput:** 3K+ QPS
- **Availability:** 99.5%
- **Scalable**: Automatically scales up/down to handle bursty traffic
- **Secure**: PrivateLink and IP-allowlist

Lakehouse Unified

- **Feature Store Integrated:** Automated feature/vector lookups
- **MLflow Integrated:** Fast & easy model deployment
- **Quality & Diagnostics:** Built-in-metrics, integrate with Lakehouse Monitoring
- **Unified governance:** Manage data & AI with UC

Simplified Deployment

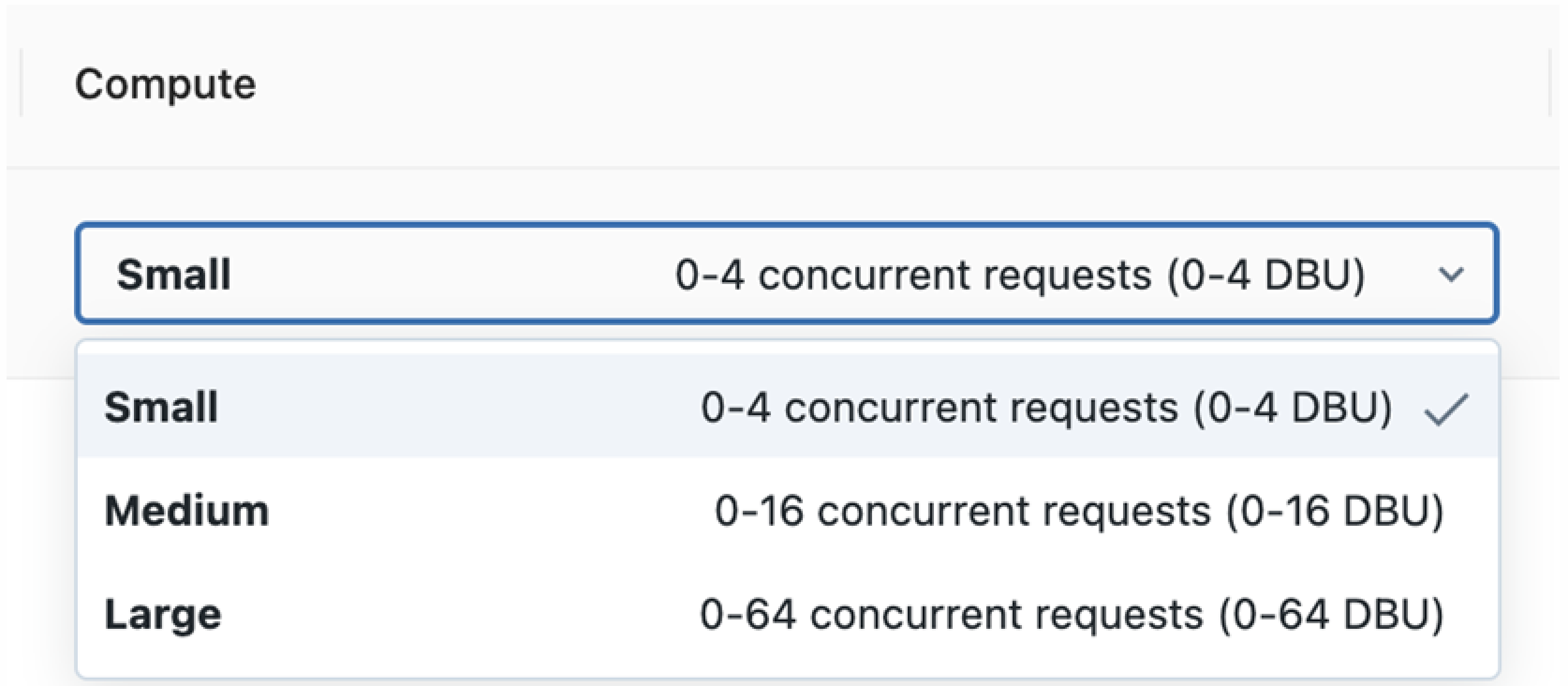
- **Simple:** Endpoints UI and API for simple deployment
- **CPU and GPU Support**
- **Flexible:** Traffic splitting for staged roll-out and A/B testing

Feature: Serverless Compute

Quickly scale up/down to handle your bursty traffic

For operational ML use cases that need to be on 24/7, use our out-of-the-box autoscaling.

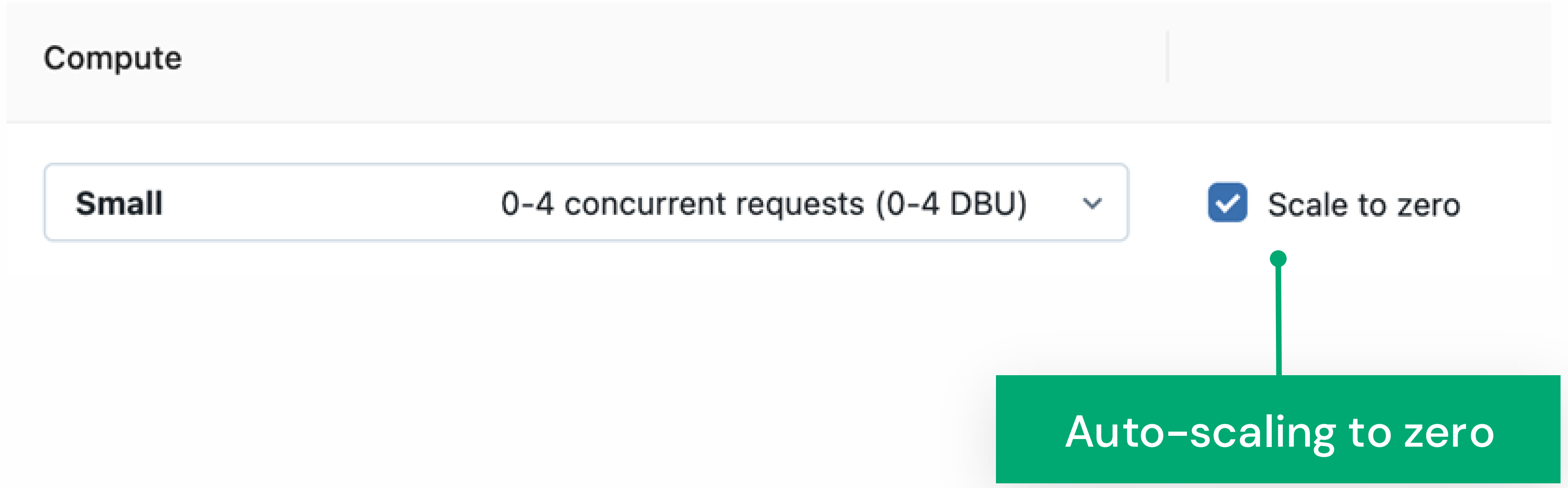
- **GPU Support**: Deploy LLMs with ease
- **Fast Autoscaling**: Compute is managed and kept warm by Databricks, allowing rapid scaling.
- **Scale within Workload size**: Autoscale within selected range.
- **Scale to zero:** Save costs for use cases with predictable, non-24/7 traffic



Compute

Small 0-4 concurrent requests (0-4 DBU)

Small	0-4 concurrent requests (0-4 DBU) ✓
Medium	0-16 concurrent requests (0-16 DBU)
Large	0-64 concurrent requests (0-64 DBU)



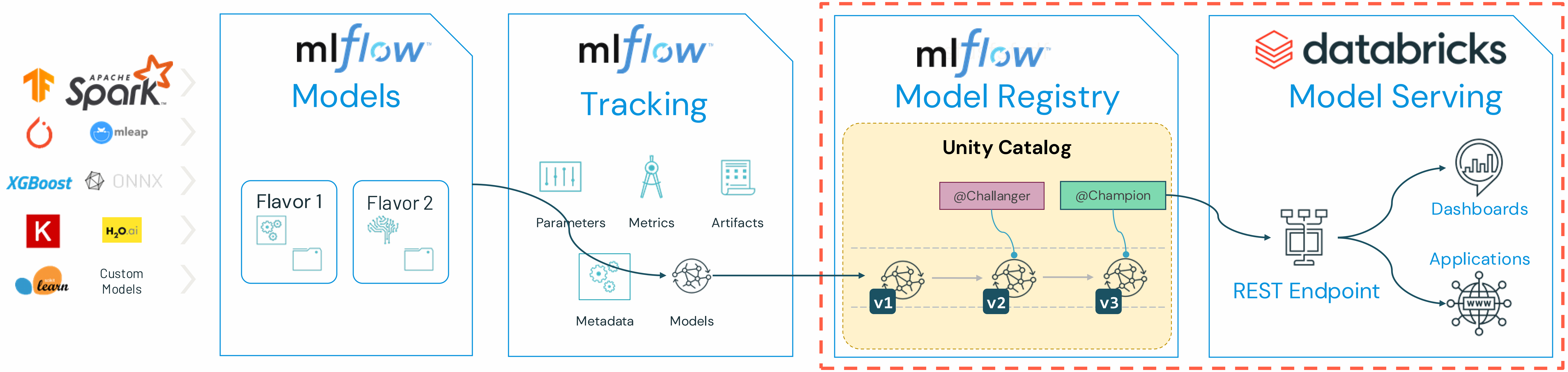
Compute

Small 0-4 concurrent requests (0-4 DBU)

[✓] Scale to zero

Feature: mlflow Integration

MLflow integration to seamlessly deploy models



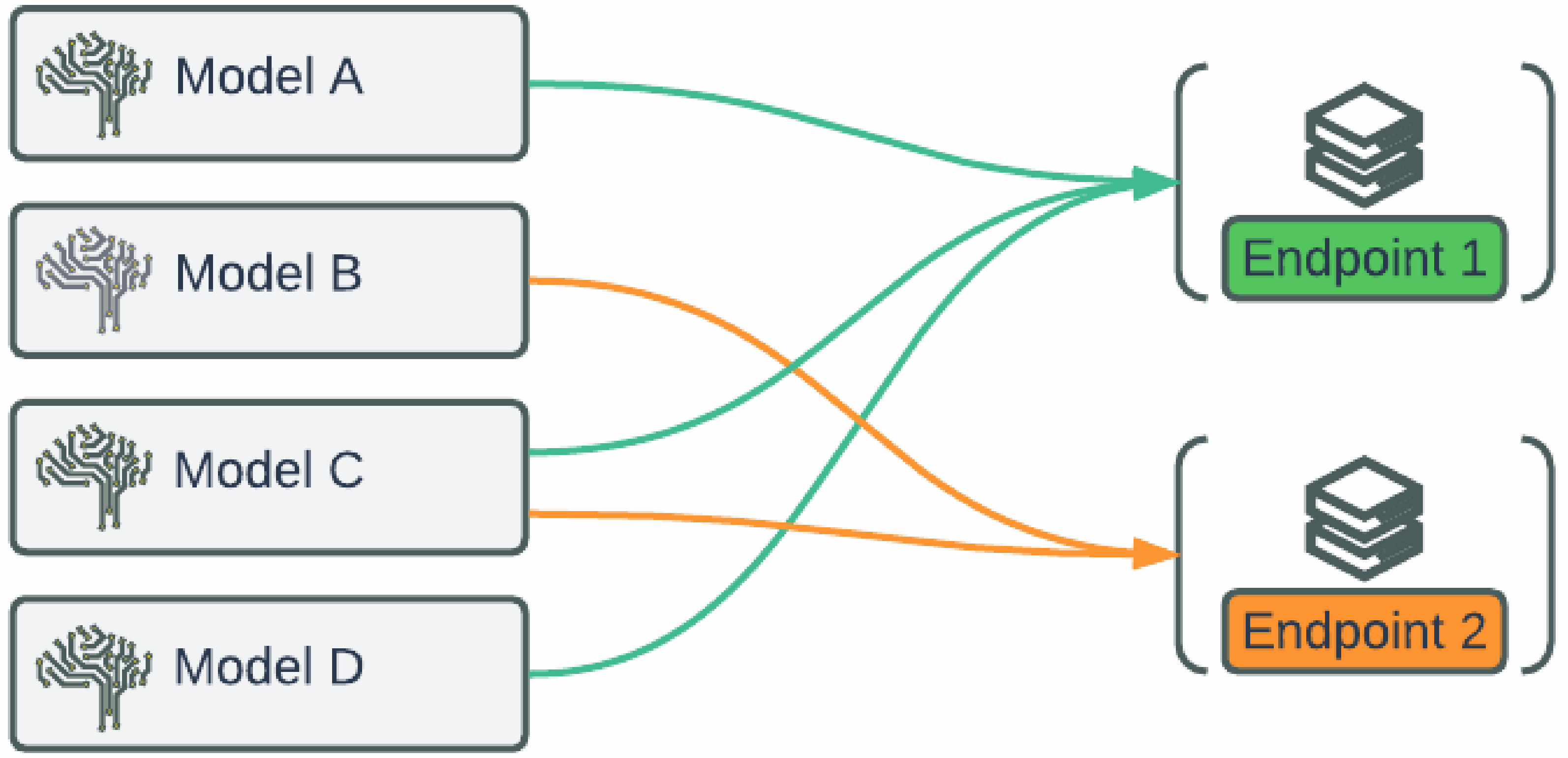
- **Faster path to production**: Natively connects to MLflow Model Registry, enabling fast and easy deployment of models.

Feature: Endpoint-Centric Workflow

Easily create and manage serving endpoints using UI or API

New endpoint-centric workflow streamlines deployment, offering simplicity, stability, and flexibility

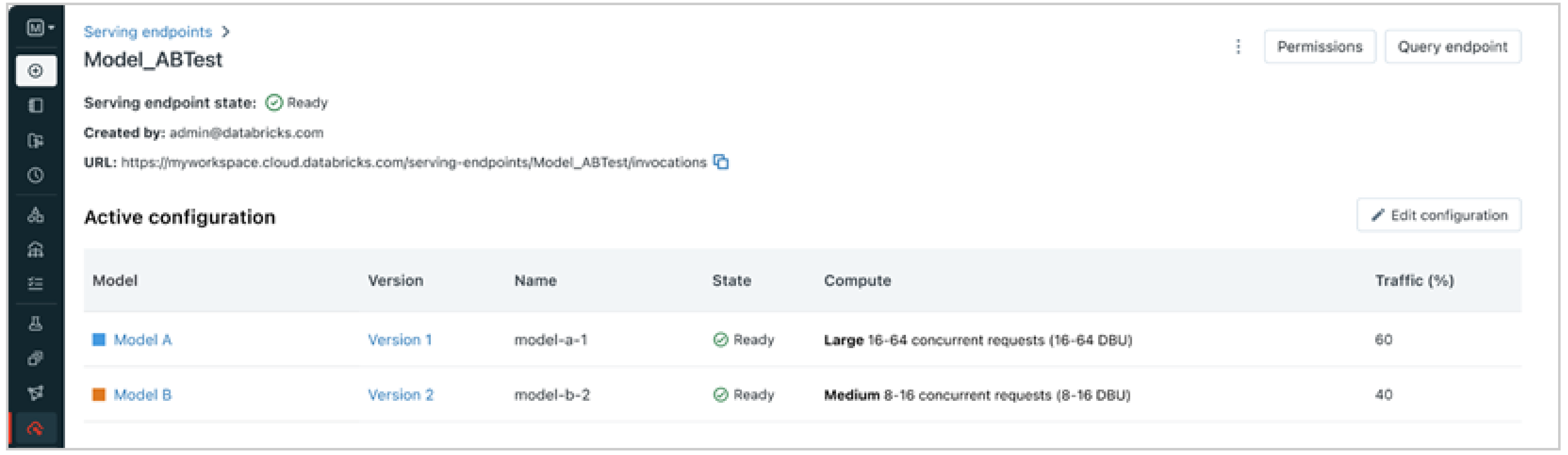
- **Stable Scoring URI**: provides 1:1 mapping with scoring URI
- **Flexible Deployments**: Deploy multiple models behind same endpoint, or same model behind multiple endpoints
- **Staged Rollout:** Gradually roll out models to minimize risk and ensure stability



Feature: A/B testing models with Endpoint

Safely and securely deploy new models to the endpoint.

- **Flexible Deployments**:
  Host multiple models behind endpoints and distribute traffic among models to progressive rollout models or perform online model validation



Serving endpoints >

Model_ABTest

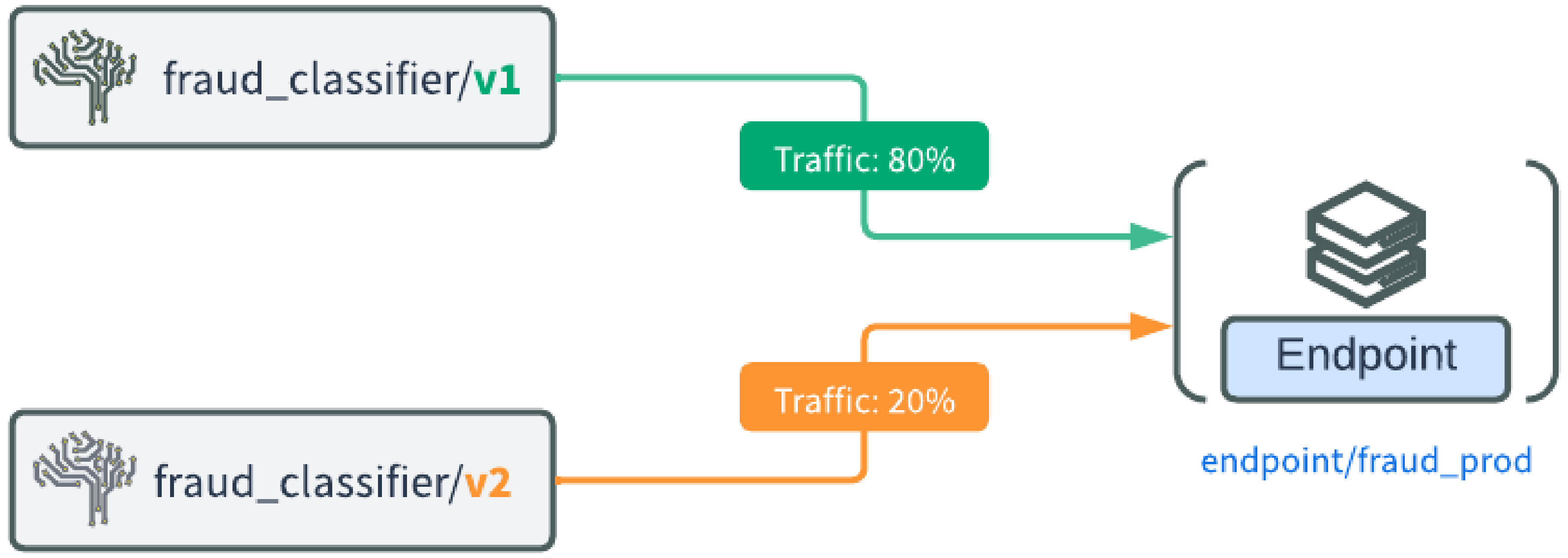
Serving endpoint state: Ready

Created by: admin@databricks.com

URL: https://myworkspace.cloud.databricks.com/serving-endpoints/Model_ABTest/invocations

Active configuration

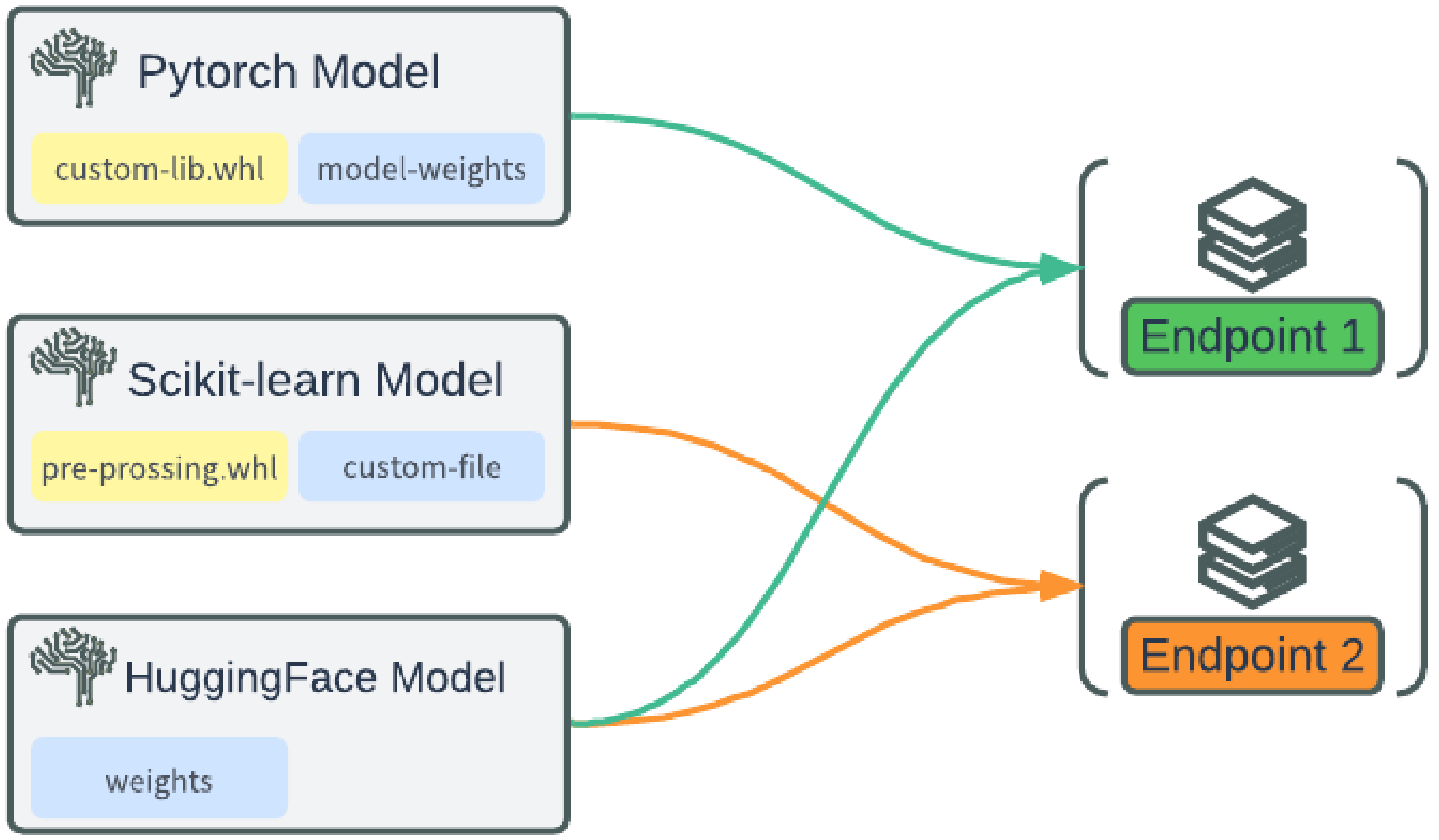
Model	Version	Name	State	Compute	Traffic (%)
🟦 Model A	Version 1	model-a-1	✓ Ready	**Large** 16-64 concurrent requests (16-64 DBU)	60
🟧 Model B	Version 2	model-b-2	✓ Ready	**Medium** 8-16 concurrent requests (8-16 DBU)	40



Feature: Serve custom models, libraries, artifacts

Supports packing models with custom artifacts

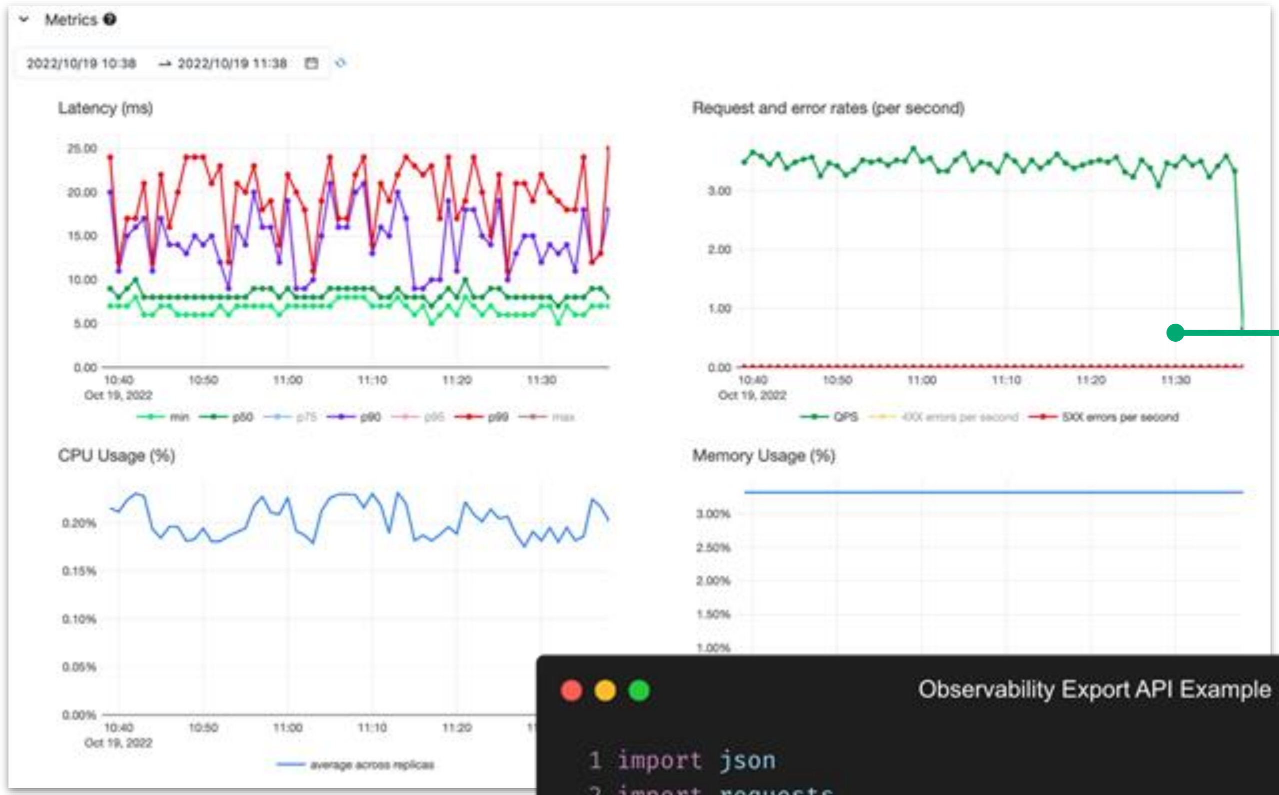
- **Custom libraries**: Include custom libraries or libraries from private mirror server while logging your model and use them with model deployments.
- **Custom Artifacts**: Package custom files and artifacts with your models and serve them with model serving.
- **Custom Models**: Serve models trained with various ML libraries(scikit-learn, HuggingFace, Pytorch)



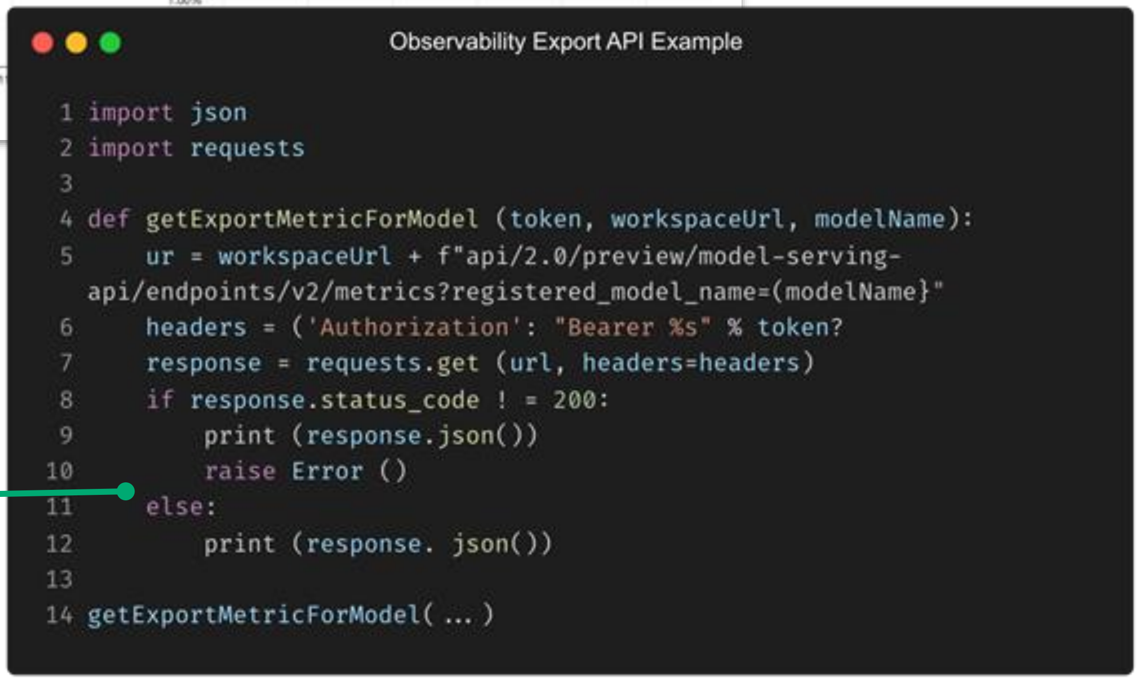
Feature: Monitoring and Observability

Export your endpoint observability for alerting

- **Metrics**: Ready-to-use metrics dashboard for endpoints
- **External Export API**: If you have a centralized observability platform, export your endpoint metrics with our export API
- **Alerting**: Set up alerts after export with Databricks SQL or your external platform of choice



Metrics Dashboard


Observability Export API Example

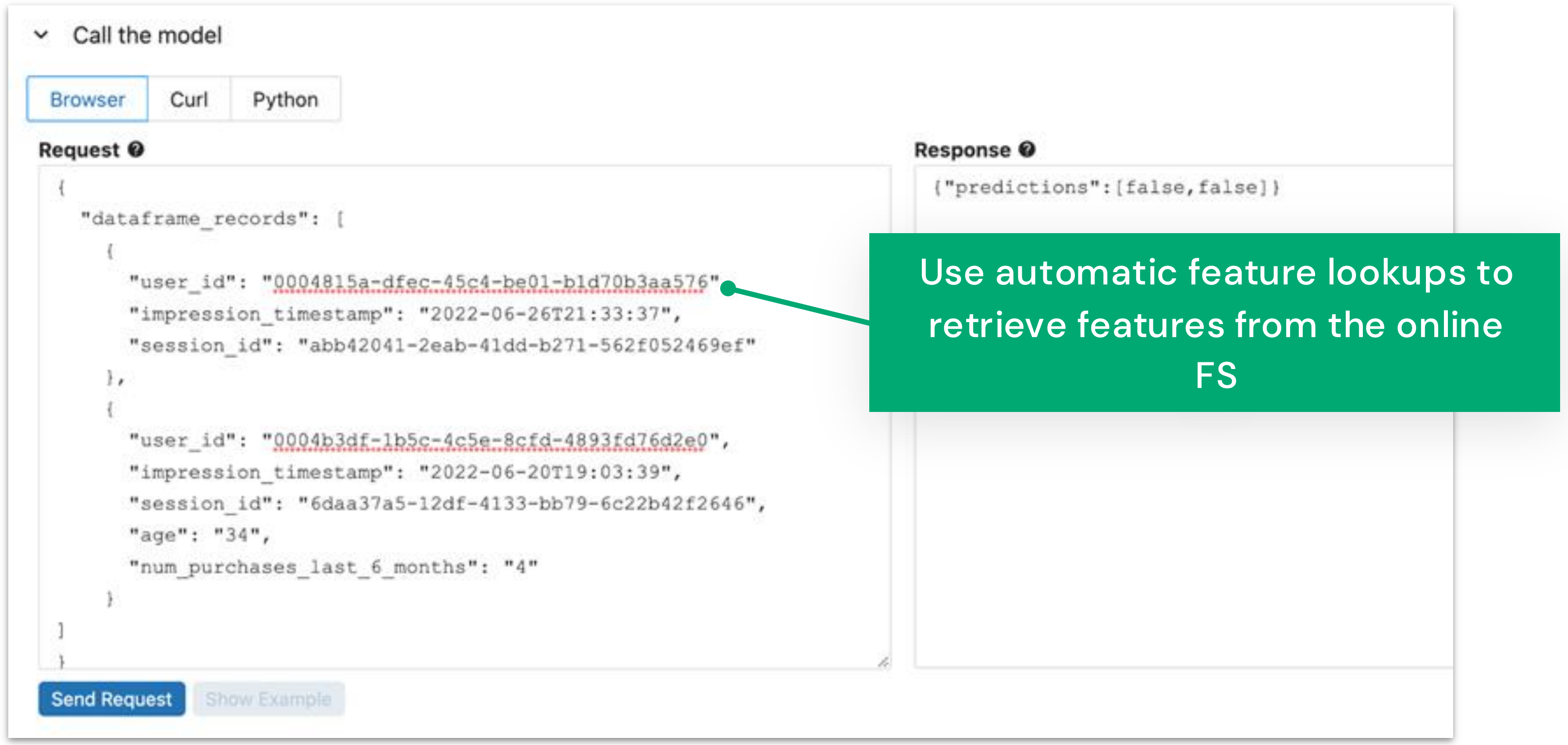
```
import json
import requests

def getExportMetricForModel (token, workspaceUrl, modelName):
    ur = workspaceUrl + f"api/2.0/preview/model-serving-
api/endpoints/v2/metrics?registered_model_name=(modelName}"
    headers = ('Authorization': "Bearer %s" % token?
    response = requests.get (url, headers=headers)
    if response.status_code ! = 200:
        print (response.json())
        raise Error ()
    else:
        print (response. json())

getExportMetricForModel( ... )
```

API for External Intg.

Feature: Automated Lookups

Automatic feature and vector lookups

- **Automatic Feature Lookups**: define your features once, publish them to the online store, and Databricks will automatically grab and join the correct features to complete the payload for inference
- **Automatic Vector Lookups**: Seamlessly integrate with Vector Stores for vector search while using LLMs.



```
{
  "dataframe_records": [
    {
      "user_id": "0004815a-dfec-45c4-be01-b1d70b3aa576",
      "impression_timestamp": "2022-06-26T21:33:37",
      "session_id": "abb42041-2eab-41dd-b271-562f052469ef"
    },
    {
      "user_id": "0004b3df-1b5c-4c5e-8cfd-4893fd76d2e0",
      "impression_timestamp": "2022-06-20T19:03:39",
      "session_id": "6daa37a5-12df-4133-bb79-6c22b42f2646",
      "age": "34",
      "num_purchases_last_6_months": "4"
    }
  ]
}
```

```
{"predictions":[false,false]}
```

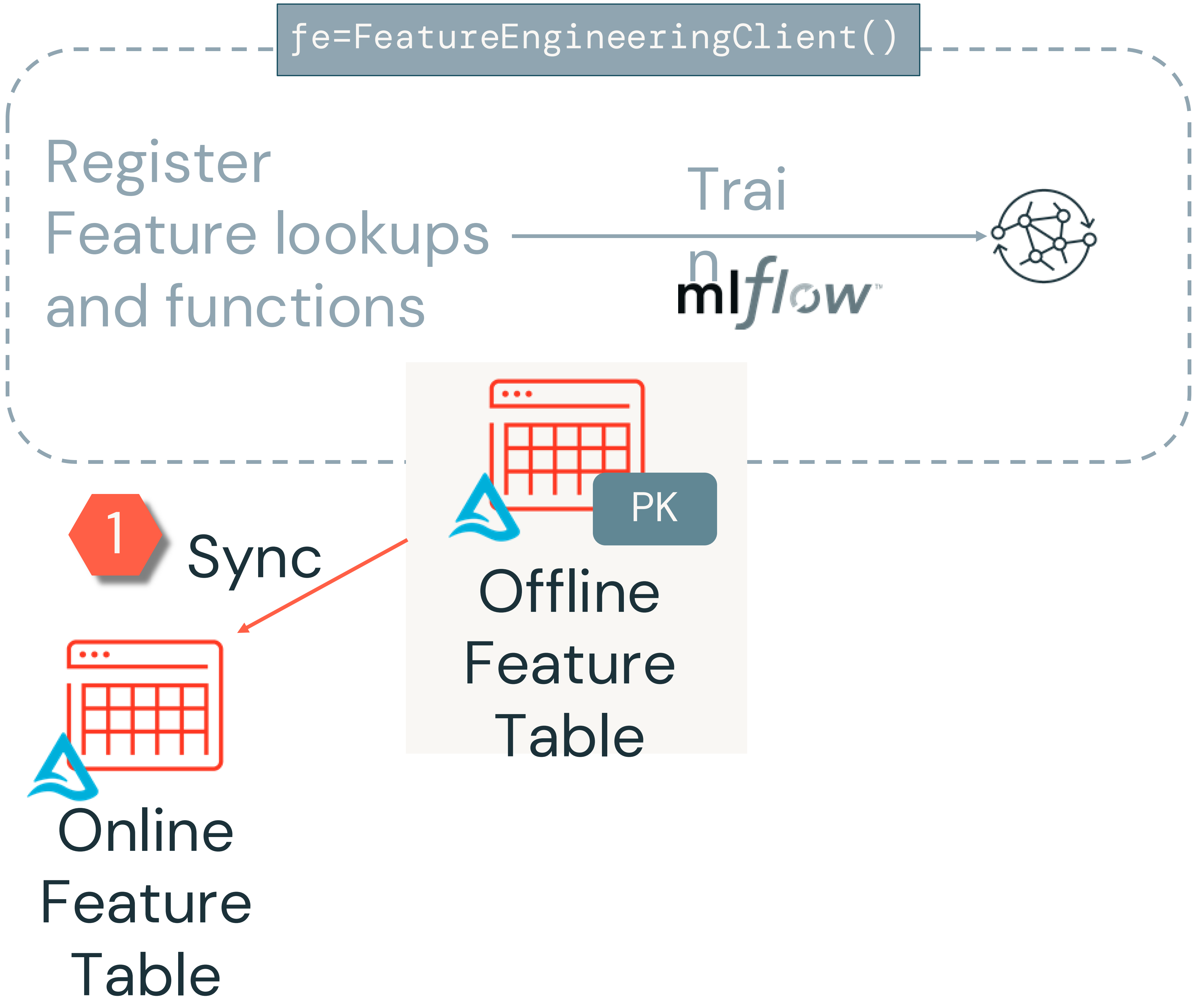
Use automatic feature lookups to retrieve features from the online FS

Model Deployment

Online Feature stores

Online Feature stores

- Supports Mosaic AI model serving, feature serving, and RAG patterns
- Low-latency and high-throughput
- Serverless auto-scaling
- Point-in-time correctness
- Supports scheduled updates



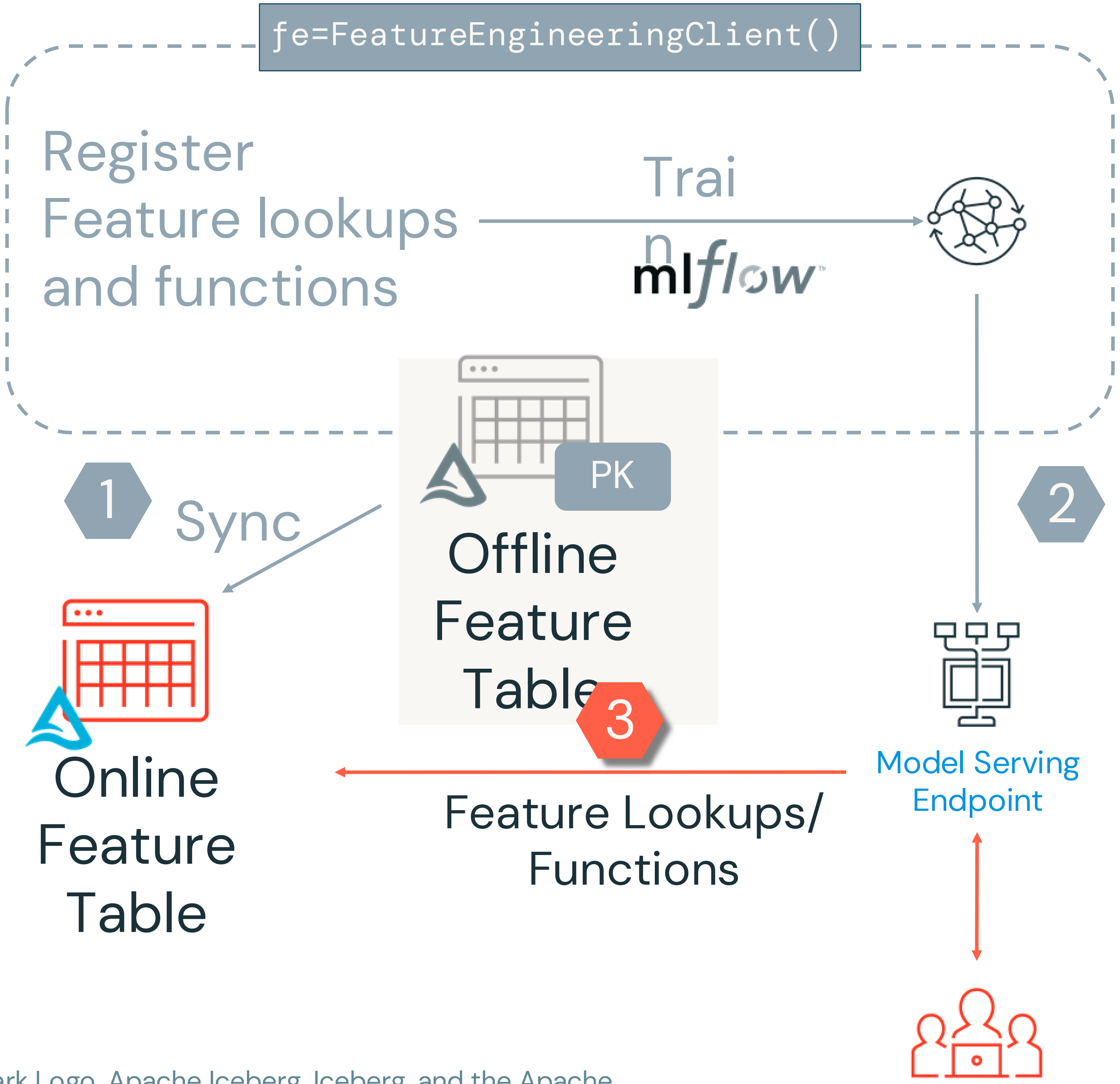
fe=FeatureEngineeringClient()
Register Feature lookups and functions
Trai n
mlflow
PK
Offline Feature Table
1
Sync
Online Feature Table

Model Deployment

Online Feature stores

Online Feature stores

- Supports Mosaic AI model serving, feature serving, and RAG patterns
- Low-latency and high-throughput
- Serverless auto-scaling
- Point-in-time correctness
- Supports scheduled updates

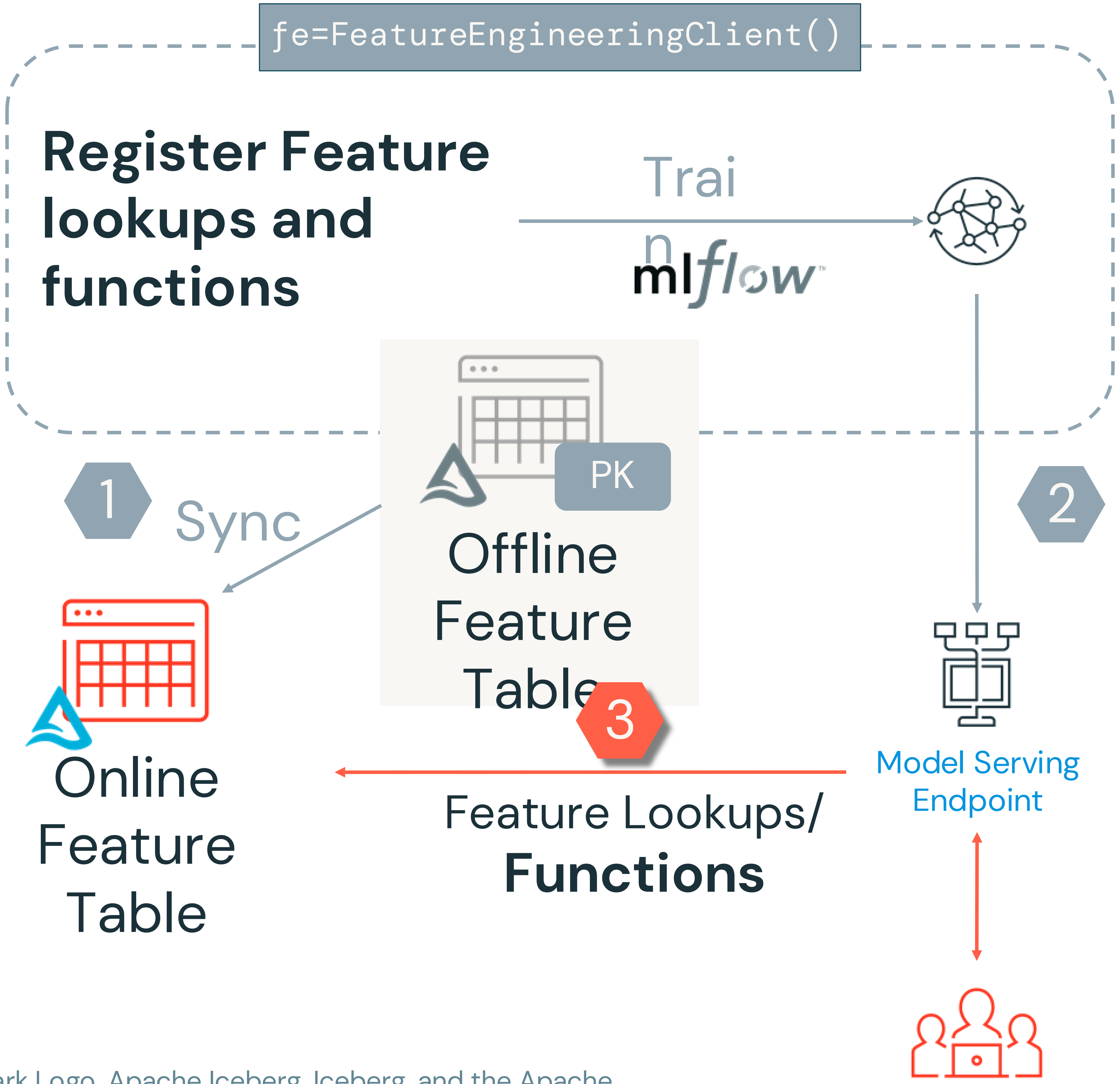


Feature Functions

Compute On-Demand Features

Feature Functions

- Automatically calculate on-demand features during inference requests
- Feature functions are Python UDFs governed by UC
- Provide function and input bindings during training
- Supports all data types supported by Feature Store except MapType and `ArrayType`

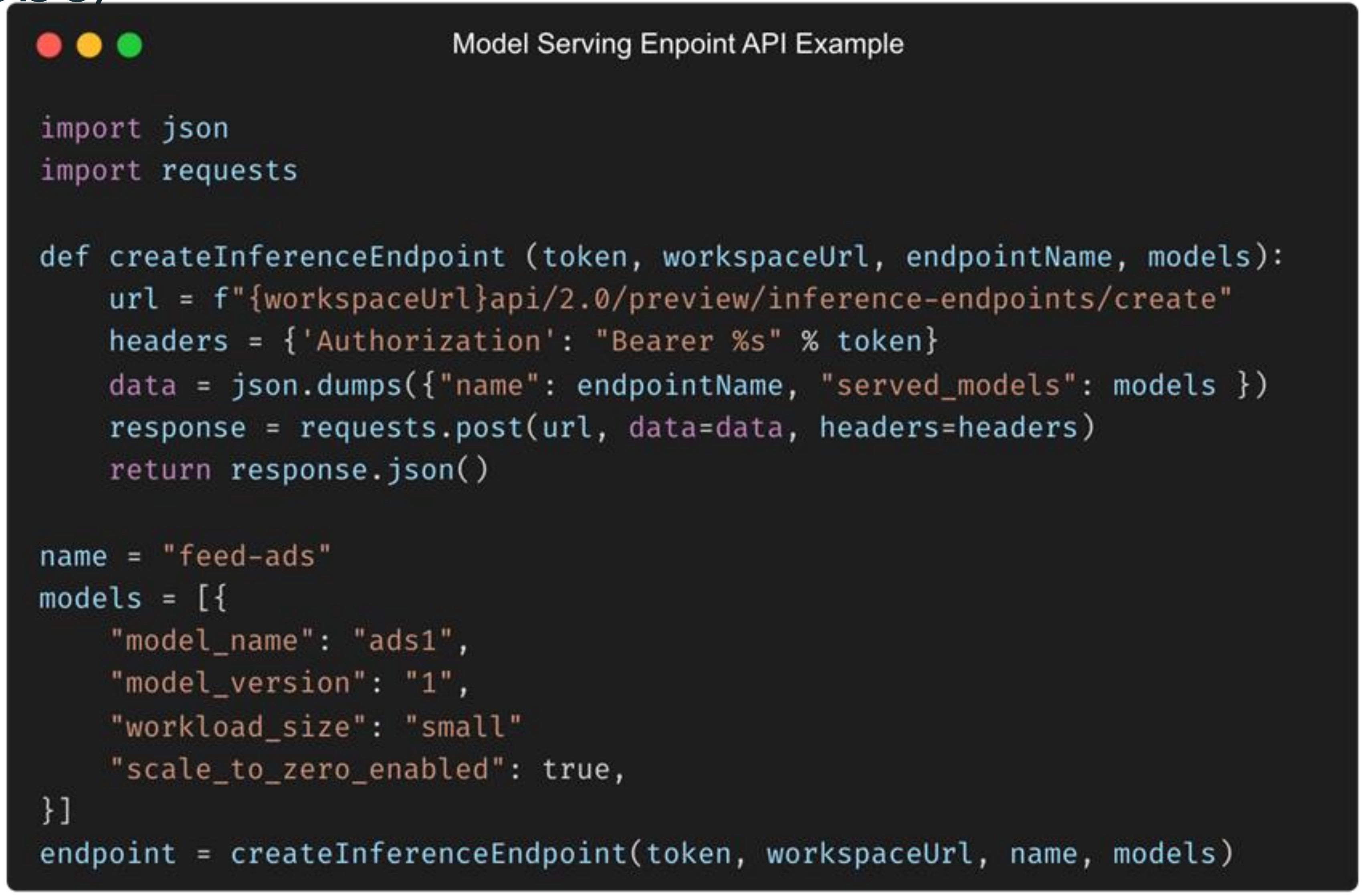

fe=FeatureEngineeringClient()
Register Feature lookups and functions
Train
mlflow
PK
Offline Feature Table
1
Sync
2
Online Feature Table
3
Feature Lookups/ Functions
Model Serving Endpoint

Feature: Endpoint API

Deploy models as APIs within your CI/CD system

Automated deployment in your Lakeflow Jobs;

- **Create/manage endpoints** with;
    - REST-API
    - Python SDK
    - Go SDK
- **Integrate with:**
    - Your CI/CD process
    - Terraform



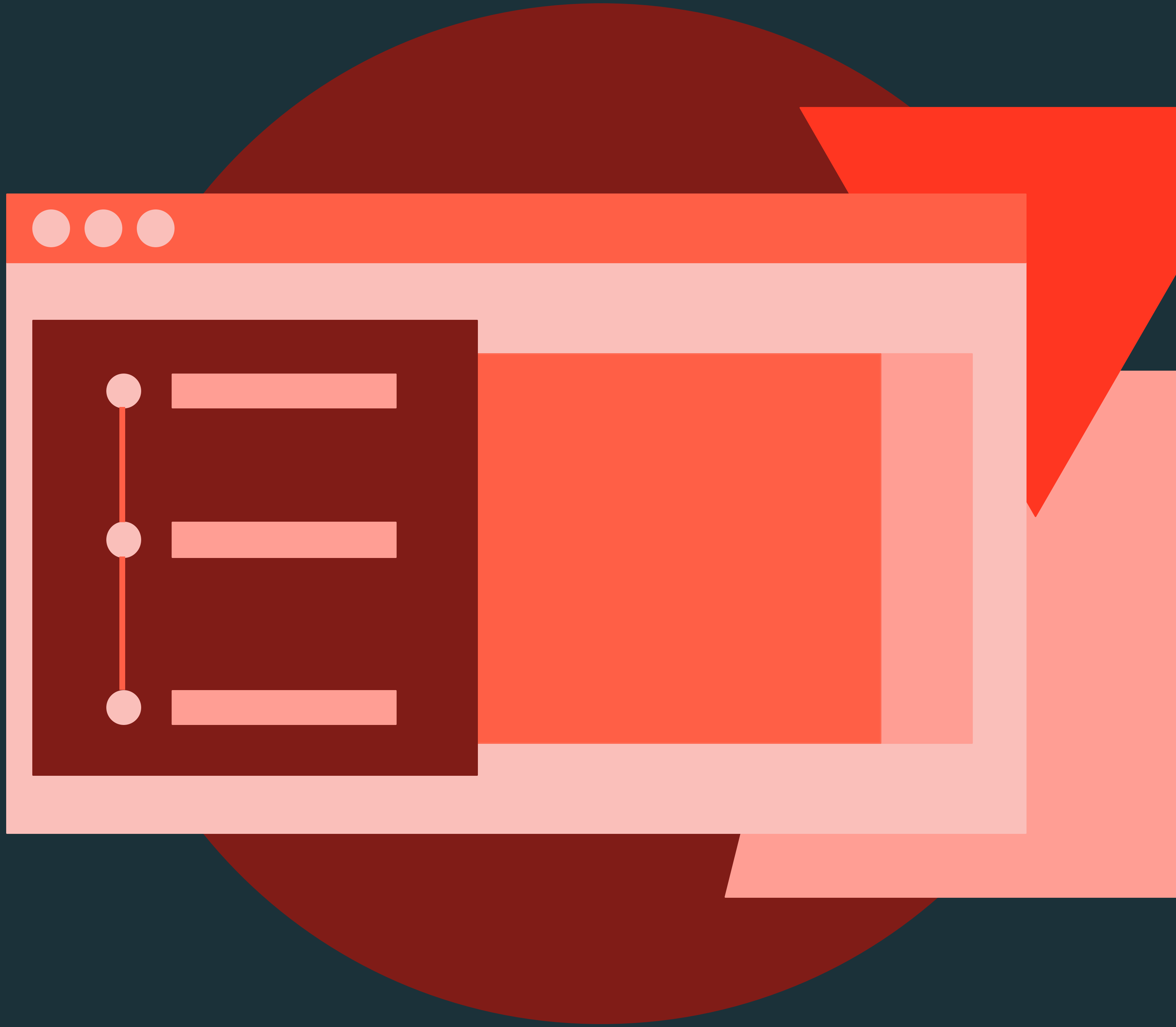
```
import json
import requests

def createInferenceEndpoint (token, workspaceUrl, endpointName, models):
    url = f"{workspaceUrl}api/2.0/preview/inference-endpoints/create"
    headers = {'Authorization': "Bearer %s" % token}
    data = json.dumps({"name": endpointName, "served_models": models })
    response = requests.post(url, data=data, headers=headers)
    return response.json()

name = "feed-ads"
models = [{
    "model_name": "ads1",
    "model_version": "1",
    "workload_size": "small"
    "scale_to_zero_enabled": true,
}]
endpoint = createInferenceEndpoint(token, workspaceUrl, name, models)
```

databricks

Real-time Deployment and Online Stores

DEMONSTRATION

Real-time Deployment with Model Serving



Demo

Outline

What we'll cover:

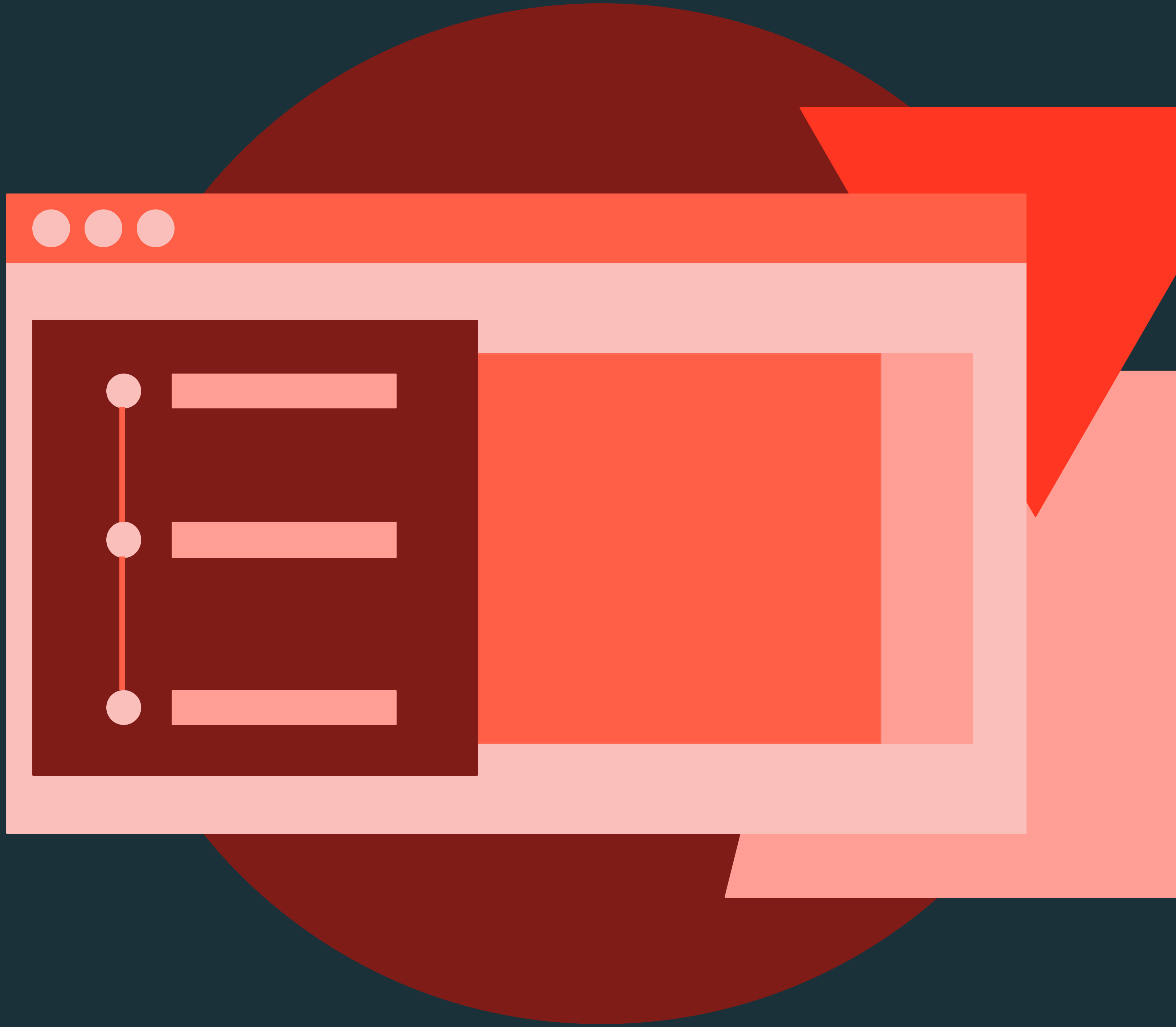
- Fit and register models
- Real-time A/B testing deployment with Model Serving
  - Serve the models using the UI
  - Serve the models using the API
- Real-time deployment with Online Features
  - Fit and log the a model with feature table
  - Setup Databricks Online Tables
  - Deploy the model with online features

databricks

Real-time Deployment and Online Stores

DEMONSTRATION

Custom Model Deployment with Model Serving



Demo

Outline

What we'll cover:

- Fit and register a custom model
  - Define wrapper class
  - Train base model
  - Wrap and log the custom model
- Serve the custom model
- Query the endpoint

databricks

Real-time Deployment and Online Stores

LAB EXERCISE

Real-time Deployment with Model Serving

Lab

Outline

What you'll do:

- Data and model preparation
- Simple real-time model deployment
  - Task 1: Serve the model using the UI
  - Task 2: Query the endpoint
- Real-time model deployment with online store
  - Task 3: Create a Databricks Online Table
  - Task 4: Deploy the model with online store
  - Task 5: Query the endpoint

databricks

Summary and Next Steps

Machine Learning Model Deployment

databricks