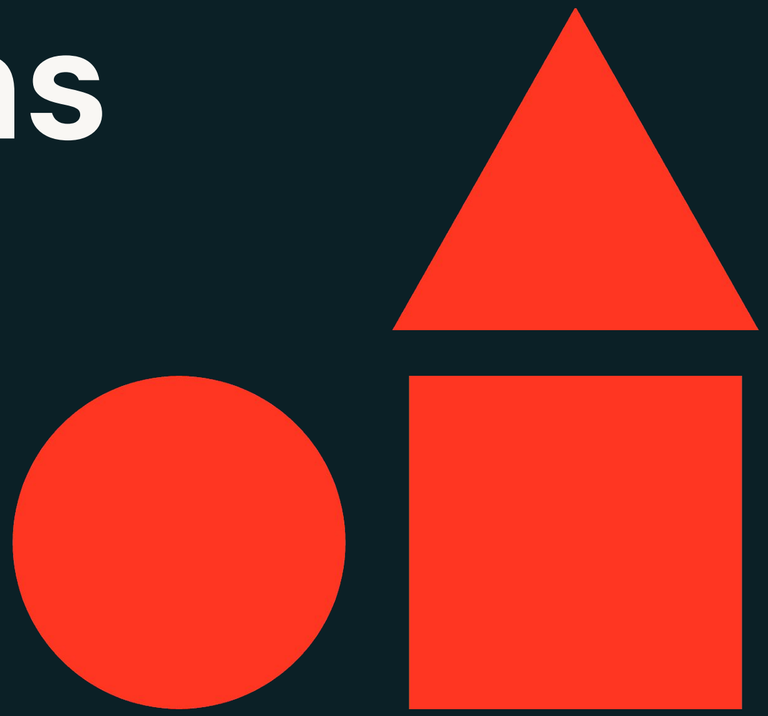
databricks

Advanced Machine Learning Operations



Databricks Academy

Agenda

01. Overview of Machine Learning Operations on Databricks	Lecture	Demo	Lab
Review of MLOps	✓		
Streamlining Development to Deployment	✓		
02. Continuous Workflows for Machine Learning Operations	**Lecture**	**Demo**	**Lab**
Streamlining MLOps	✓		
Streamlining MLOps with Databricks	✓	✓	✓
03. Testing Strategies with Databricks	**Lecture**	**Demo**	**Lab**
Automate Comprehensive Testing	✓	✓	✓
Model Rollout Strategies with Databricks	✓	✓	
04. Model Quality and Lakehouse Monitoring	**Lecture**	**Demo**	**Lab**
Lakehouse Monitoring	✓	✓	✓
05. Streamlining Multiple Environment Deployments - DABs	**Lecture**	**Demo**	**Lab**
Build ML assets as Code	✓	✓	
Course Summary and Next Steps	✓		

Course Learning Objectives

- Understand the full machine learning lifecycle and MLOps
- Identify key components and operational areas of MLOps
- Learn best practices for data, model management, and testing in ML architectures
- Understand Databricks' support for MLOps implementation and automation
- Explore core principles of scalable and reliable MLOps systems
- Understand guiding principles of ML architectures
- Differentiate between direct and indirect environment separation
- Understand CI/CD, testing, and pipeline management in Databricks
- Explain Lakehouse Monitoring and custom metrics setup
- Describe types of drift, model rollout strategies, and A/B testing frameworks
- Understand the concept of Infrastructure as Code and Databricks Asset Bundles (DABs)
- Explore anatomy of ML projects and Databricks MLOps stacks

Prerequisites/Technical Considerations

Things to keep in mind before you work through this course

Prerequisites

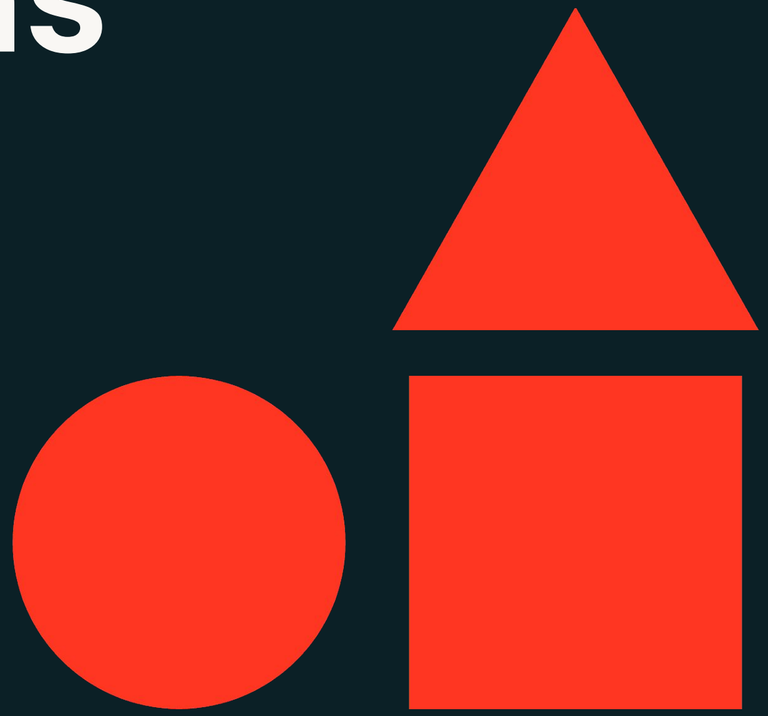
1. Intermediate-level knowledge of traditional machine learning concepts
2. Intermediate-level experience with traditional machine learning development on Databricks
3. Intermediate-level knowledge of Python for machine learning projects
4. **Recommended:** Intermediate-level experience with basic Spark concepts

Technical Considerations

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

databricks

Overview of Machine Learning Operations on Databricks



Advanced Machine Learning Operations

databricks

Machine Learning Operations with Multiple Environments

LECTURE

Review of MLOps

Databricks Academy

Learning objectives

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

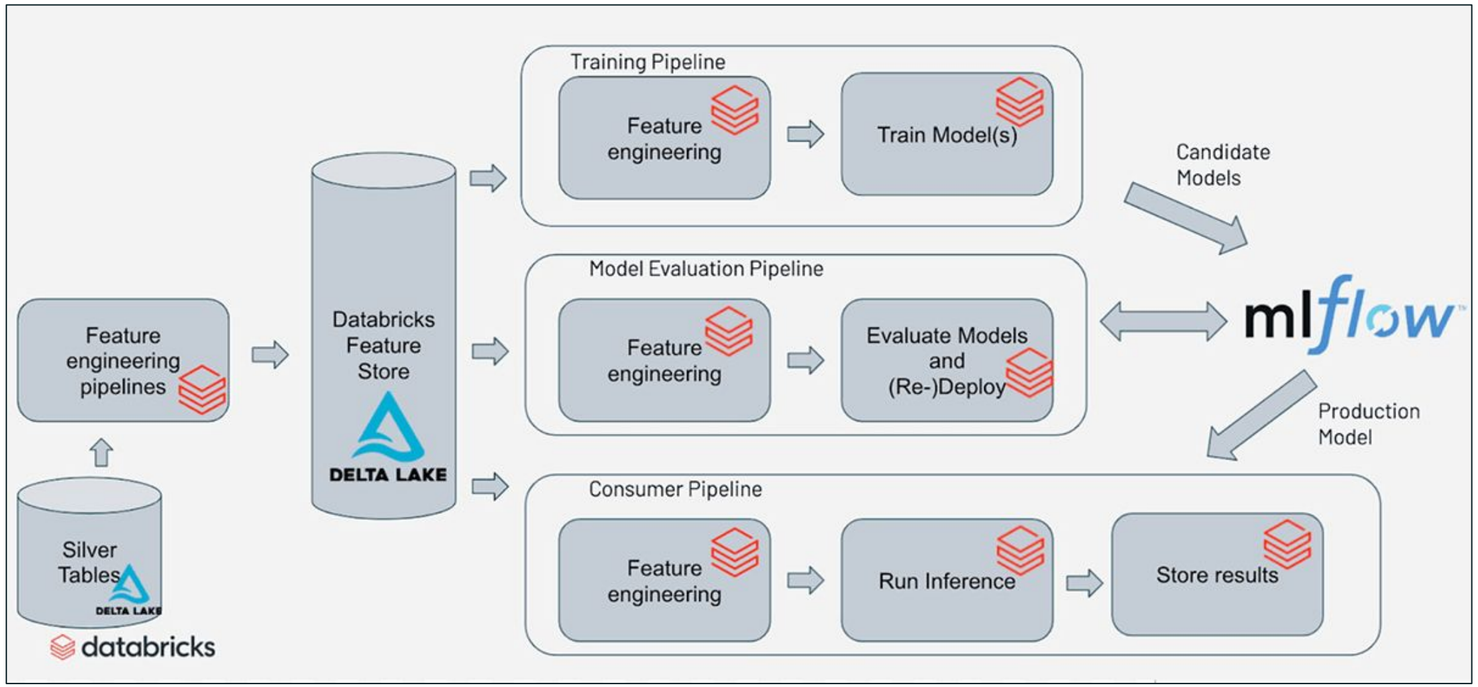
- Understand the Machine Learning Lifecycle
- Identify the Three Main Operational Components of MLOps
- Understand DataOps for Quality Data Management
- Understand DevOps for Model Deployment
- Understand ModelOps for Model Lifecycle Management
- Understand How Databricks Enables Users to Implement Best Practices for MLOps

Operational Excellence in Machine Learning

Enhancing Efficiency and Reliability in Machine Learning Operations

Standardize MLOps processes for consistent and reliable ML.

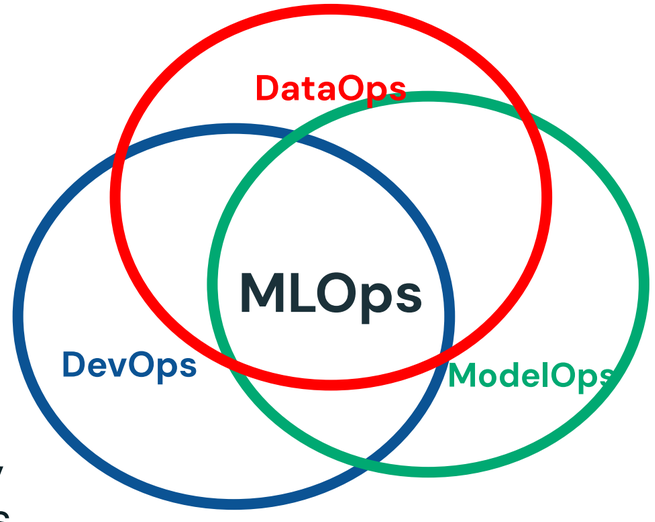
- Implement **DevOps processes** (CI/CD) by automating building, testing, and deploying code.
- **Modularize code** for clearly defined steps, testing and simplify refactoring.
- Encapsulate individual pipelines and workflows into an **orchestrator**.
- Incorporate **versioning** for tracking models and datasets.
- Set up **monitoring**, **alerting**, and **logging**.

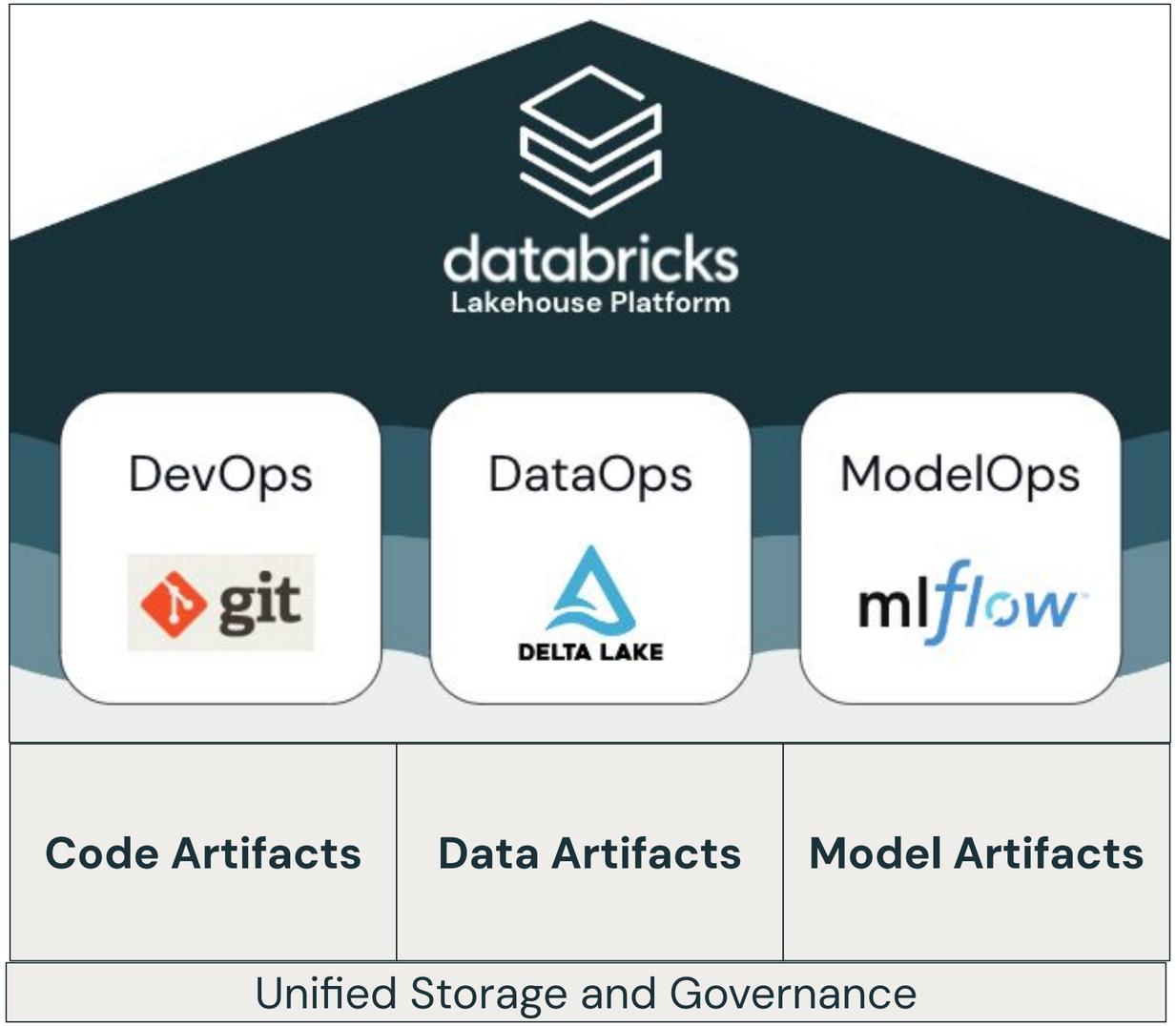


Machine Learning Operations

What is MLOps?

- A combination of **DevOps**, **DataOps**, and **ModelOps.**
- A set of **processes** and **automation** for managing: **models**, **data**, and **code** assets.





Results of MLOps implementation:

- **Efficiency:** Higher quality models in less time.
- **Scalability:** Thousands of models can be overseen, controlled, managed, and monitored.
- **Risk reduction:** Through repeatable workflows, automated monitoring and transparency.

DataOps, DevOps, and ModelOps are…

A set of practices, processes, and technologies to:

DataOps

Ensure data **quality**

- Optimize data **processing**
- Centralize data **discovery**, **management**, and **governance**
- Establish traceable data **lineage** and **monitoring**
- **Enhance collaboration** across teams
- **Monitor** Data

DevOps

Treat Machine Learning as **code**

- Automate **CI/CD**
- Enable continuous **code testing**
- **Version control**
- Establish Production-grade **workflows**
  - Orchestration & Automation
- **Monitor** system performance

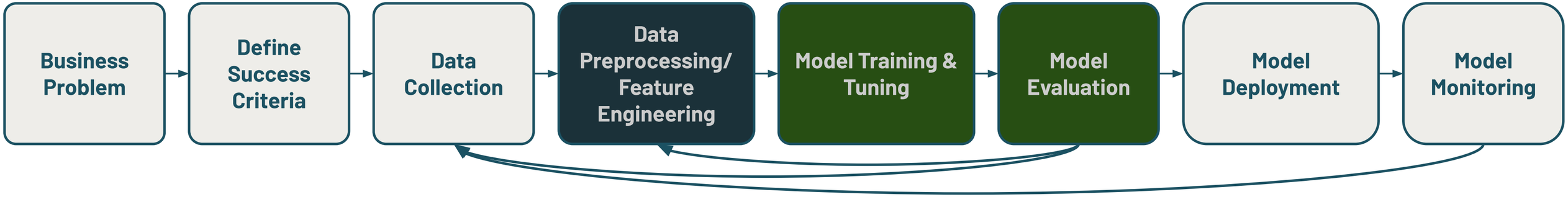
ModelOps

Move beyond models as **objects**

- Treating **model code** as **software**
- Treating models as **data**
- Manage the **model lifecycle**
- **Monitor** Model Performance

MLOps in the Machine Learning Lifecycle

From Data to Deployment with Machine Learning Libraries

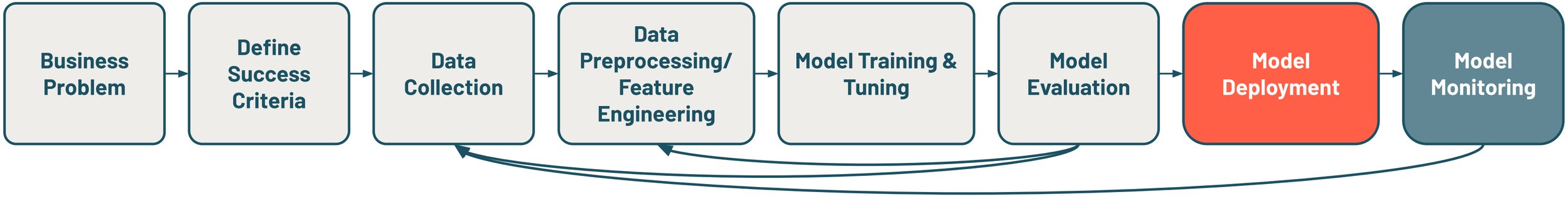


Ingest & transform data	Autoloader and Apache Spark
Track data changes to including versioning & lineage	Delta tables
Build, manage, & monitor data processing pipelines	Delta Live Tables
Ensure data security & governance	Unity Catalog
Exploratory data analysis, dashboards, & general coding	Databricks SQL, Dashboards, and Databricks notebooks
Schedule data pipelines & Automate general workflows	Databricks Workflows
Create, store, manage, & discover features	Databricks Feature Store

No-code model development	Databricks AutoML
Track model development	MLflow model tracking
General coding	Databricks Notebooks & Databricks Runtime
Hyperparameter Tuning and Model Choosing	Scikit-Learn, Hyperopt, Optuna, and Ray
Model code version control and sharing	Databricks Asset Bundles and Databricks Git Folders
Manage model lifecycle	Models in Unity Catalog
Schedule & Automate machine learning workflows	Databricks Workflows
Model monitoring	Lakehouse Monitoring & Mlflow Model Evaluate

MLOps in the Machine Learning Lifecycle

From Data to Deployment with Machine Learning Libraries



Manage model lifecycle	Models in Unity Catalog
Maintain a highly available low latency REST endpoint	Mosaic AI Model Serving
Automate and schedule workloads, from: ETL to ML	Databricks Workflow Jobs and DLT *Databricks also supports integrations with popular third party orchestrators like Airflow.*
Establish CI/CD Pipelines	Databricks Asset Bundles, Azure DevOps, Jenkins, or GitHub Actions
Serve features and vectors	Databricks Feature Store and Vector Database
Flexible Model Serving	MLflow Models & MLflow Model Registry

Monitoring, and Maintaining your Applications	Lakehouse Monitoring
Notifications & Alerts	Databricks SQL
Manage model lifecycle	Models in Unity Catalog
Retrain on Schedule or Trigger	Azure DevOps, Jenkins, or GitHub Actions
Monitor model quality and endpoint health	Service logs, Build logs, Endpoint health metrics, Inference tables
Automatic Data Quality Testing	Delta Live Tables

databricks

Machine Learning Operations with Multiple Environments

LECTURE

Streamlining Development to Deployment

Databricks Academy

Learning objectives

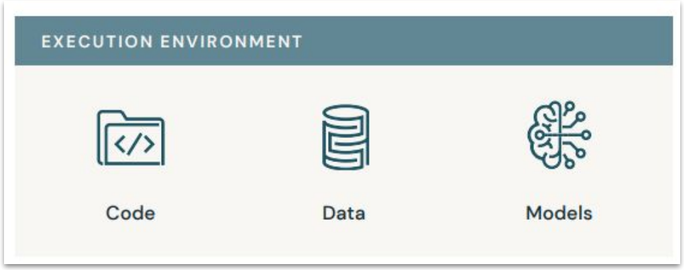
Things you'll be able to do after completing this lesson

- Understand guiding principles of machine learning architectures
- Understand the difference between direct and indirect environment separation
- Understand why direct separation is recommended
- Understand why deploying code instead of models is recommended
- Understand guiding principles of machine learning operations

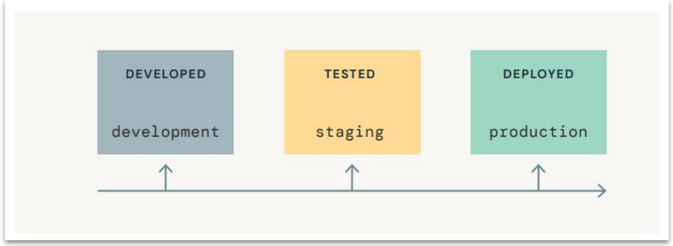
ML Architecture Guiding Principles

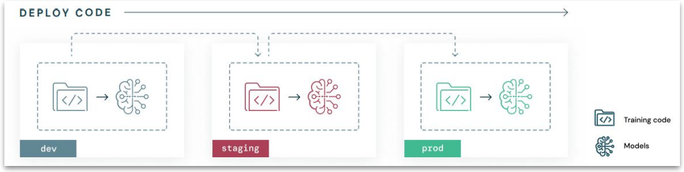
Optimizing Development and Deployment

- Environment Setup
  - Create separate environments for different stages. Example: Development, Staging, Production.
    - Other configurations can also be used to meet the specific needs of your organization.
- Deploy code, not models
  - In most situations, promote code, not models, between environments.
  - Production version of the model is trained using production code.
- Versioning
  - Store pipelines and code in Git, with branch strategies reflecting environment transitions.
  - Move ML logic between branches to parallel environment progression.
- Access Control
  - Centralize management and unify governance of data, features, model development and model lifecycle.



Environments are: where models and data are created or consumed by code





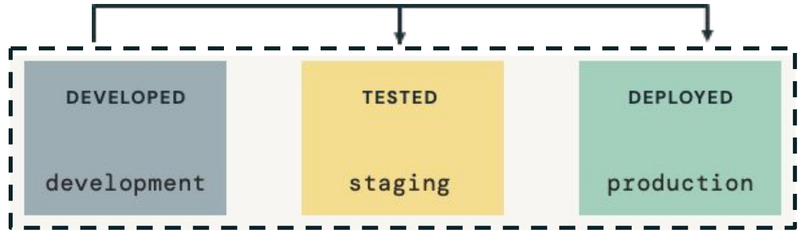
Environment Separation

How many Databricks workspaces do we have?

Direct Separation (Recommended)

- **Completely separate** Databricks workspaces for each environment
- **Simpler environments**
  - Easier to manage **security** and **compliance**
  - More straightforward to **track** changes and issues
- **Reduces the risk** of cross-environment interactions.
- **Scales** well to multiple projects
  - *Might* require more **administrative effort** to manage multiple workspaces.
  - *Might* lead to **higher costs** due to the need for separate resources for each workspace.

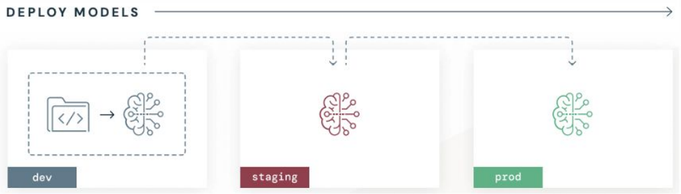
Indirect Separation



- **One Databricks workspace** with enforced separation using **naming conventions** and **access control**.
- **Simpler** overall infrastructure
  - **Less permissions** required
  - Can be more **cost-effective** needing fewer resources.
- Easier cross-environment **collaboration** and sharing.
- **Doesn't scale well** to multiple projects
  - **Complex** individual environment
  - Requires management to **maintain** the separation and prevent accidental cross-environment interactions.
  - Can be more challenging to manage **security** and **compliance** requirements

Deployment Patterns

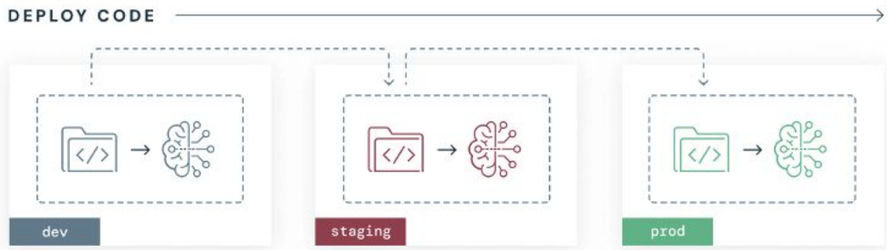
Moving from **Deploy Model** to **Deploy Code**



	Deploy models		
Process	1. Models are trained in dev and promoted to staging 2. Models are tested in staging 3. Models are promoted to prod		
Trade-offs	↑ Simplicity ↑ DS familiarity ↑ Computational cost ↓ Automation ↓ Scalability ↓ Reproducibility		

Deeper dive into benefits of “Deploy Code”

Recommended:



Automation	↑ Supports automated retraining in secure and controlled environments
Data access control	↑ Restricts data access efficiently, where only the production environment requires read permissions to sensitive training data.
Reproducible models	↑ Every step of the model generation process can be replicated consistently across environments, allowing teams to recreate models consistently.
Support for large projects	↑ Scales well for larger projects by promoting modularized, organized, reusable code and structured testing workflows.
Testing	↑ First unit and integration testing in the development environment followed by unit and integration testing in staging, which closely mimics production.
Data science familiarity	↓ Requires the data science team to write production-ready, modular code. ensuring smooth hand-offs to engineering teams
Eng setup & maintenance	↓ Requires robust CI/CD infrastructure, supporting consistent unit and integration testing for all models.

Dimensions of Architecture

Differentiating initial organization/setup and ongoing workflows

Infrastructure

The organization, governance, and setup of environments, data, compute and other resources.

- Set up one time (per project or per team/organization)
- Crucial to downstream success of project(s)

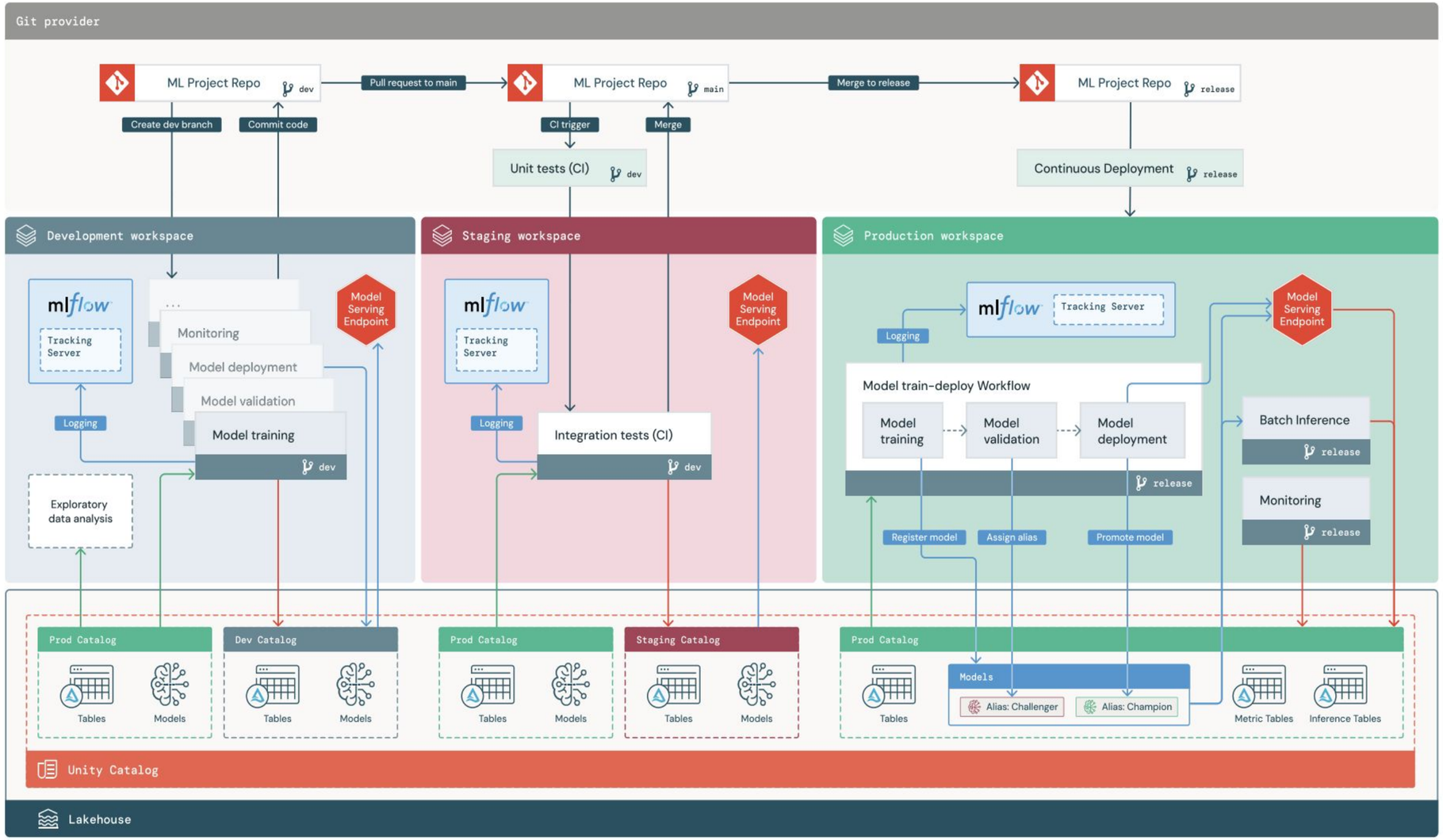
Workflow

The processes that ML practitioners follow within a defined architecture to achieve success on an ML project.

- Repeatable, fluid processes specific to a project
- Aligned to organizational best practices

Machine Learning Architecture

Involves designing and implementing an end-to-end pipeline involving the tasks and tools to solve specific business problems using machine learning.



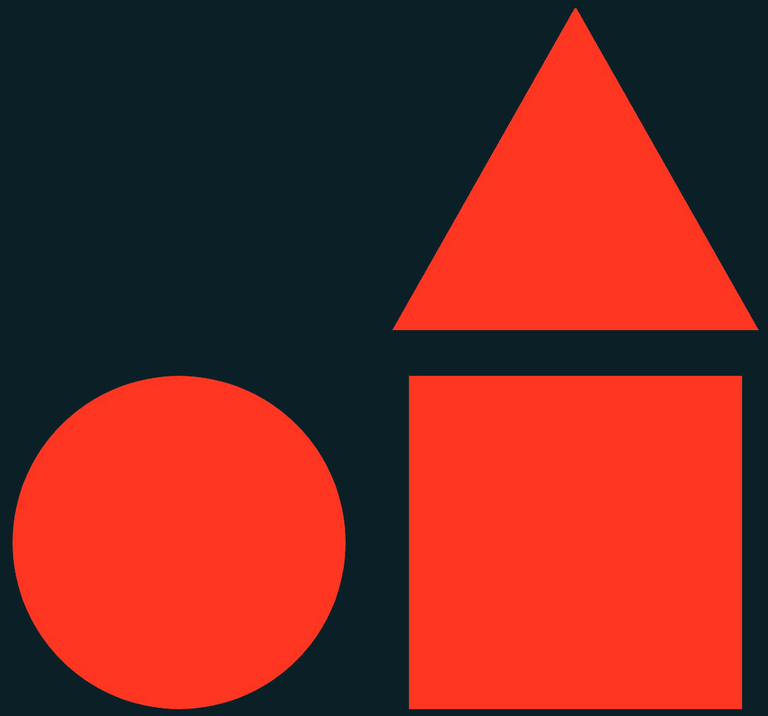
ML Operations Guiding Principles

Optimizing Development and Deployment

- Automated and Orchestrated **Continuous Integration and Continuous Deployment**
  - Set up CI/CD pipelines to automate the machine learning lifecycle.
  - Reducing manual steps and ensuring consistency.
  - Break down ML workflows into manageable modules.
- Automate Comprehensive **Testing**
  - Implement a testing strategy including: unit, integration, and end-to-end tests.
  - Incorporate canary tests and blue/green deployment with CI/CD Pipelines.
- **Monitoring** and **Alerts**
  - Monitor model performance and operations with visuals, custom metrics, and insights.
  - Set up automated alerts to detect issues early.
- Build ML **Assets as Code**
  - Standardize and automate ML asset creation using Infrastructure as Code and predefined templates.

databricks

Continuous Workflows for Machine Learning Operations



Advanced Machine Learning Operations

databricks

Advanced Machine Learning Operations

LECTURE

Streamlining MLOps

Learning objectives

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

- Understand components of CI, CT, CD, and CM
- Understand recommended architecture continuous frameworks
- Understand core tools available with Databricks for continuous frameworks

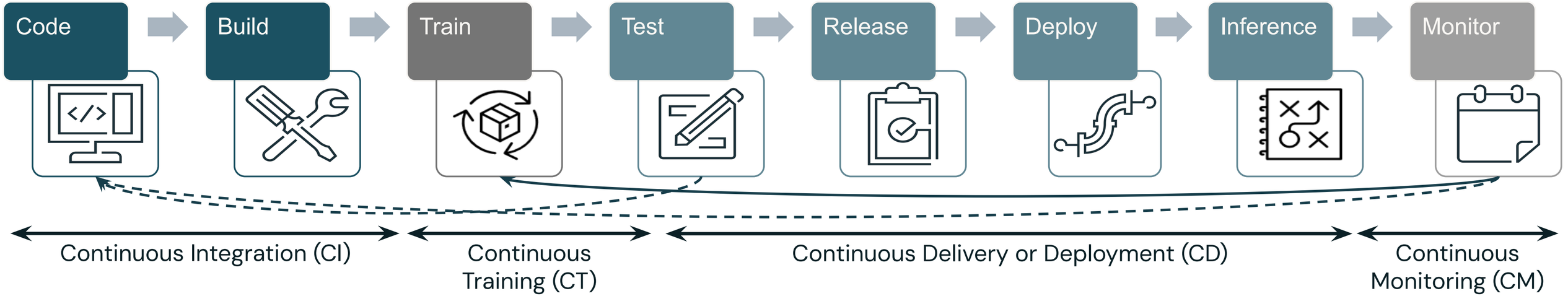
Continuous Terminology

DevOps, Data Ops, Model Ops

- **Continuous Integration (CI)** extends the testing and validating code by adding testing and validating of **code**, **data** and **models**.
- **Continuous Training (CT)** **automatically** (re)trains ML **models** on **data** for (re)deployment.
- **Continuous Deployment (CD)** concerned with **delivery** of a ML training pipeline that automatically deploys another ML **model** and needed **stores**.
- **Continuous Monitoring (CM)** concerned with **continuous monitoring** of **production data** and **model performance metrics**.

Goal: Increase developer productivity, improve model quality, code quality, and accelerate trained model delivery.

Continuous Integration and Training



Continuous Integration (CI)

- Automatically trigger pipelines after code commits.
- Run tests and validation on code, data, and models.
- Track all changes via version control.

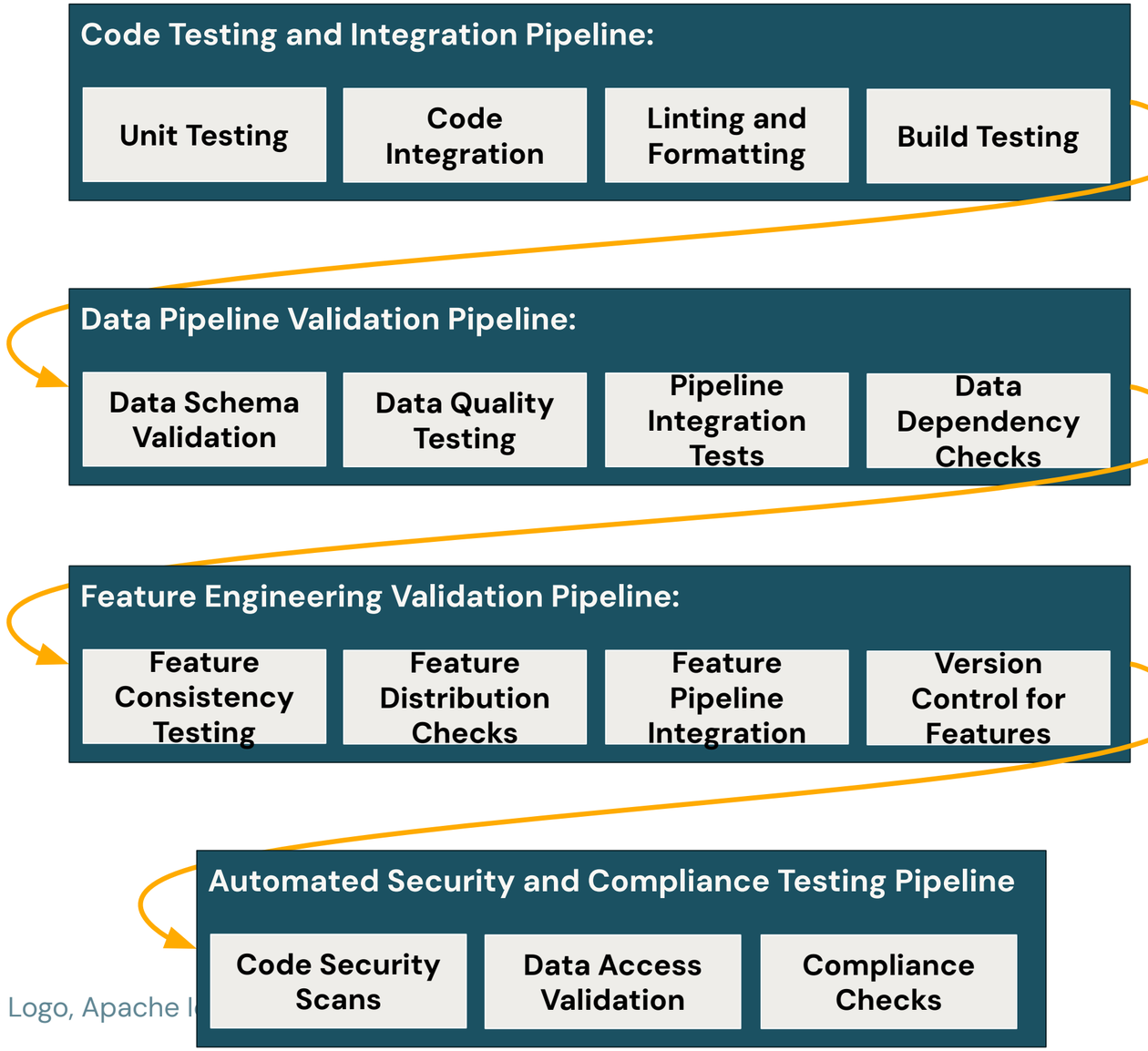
Continuous Training (CT)

- Retrain models automatically when new code or data is available or performance indicates.
- Log all experiments and model metrics for reproducibility.
- Automate hyperparameter tuning.
- Evaluate and validate model.

Continuous Integration Deeper Dive

Possible Pipelines...

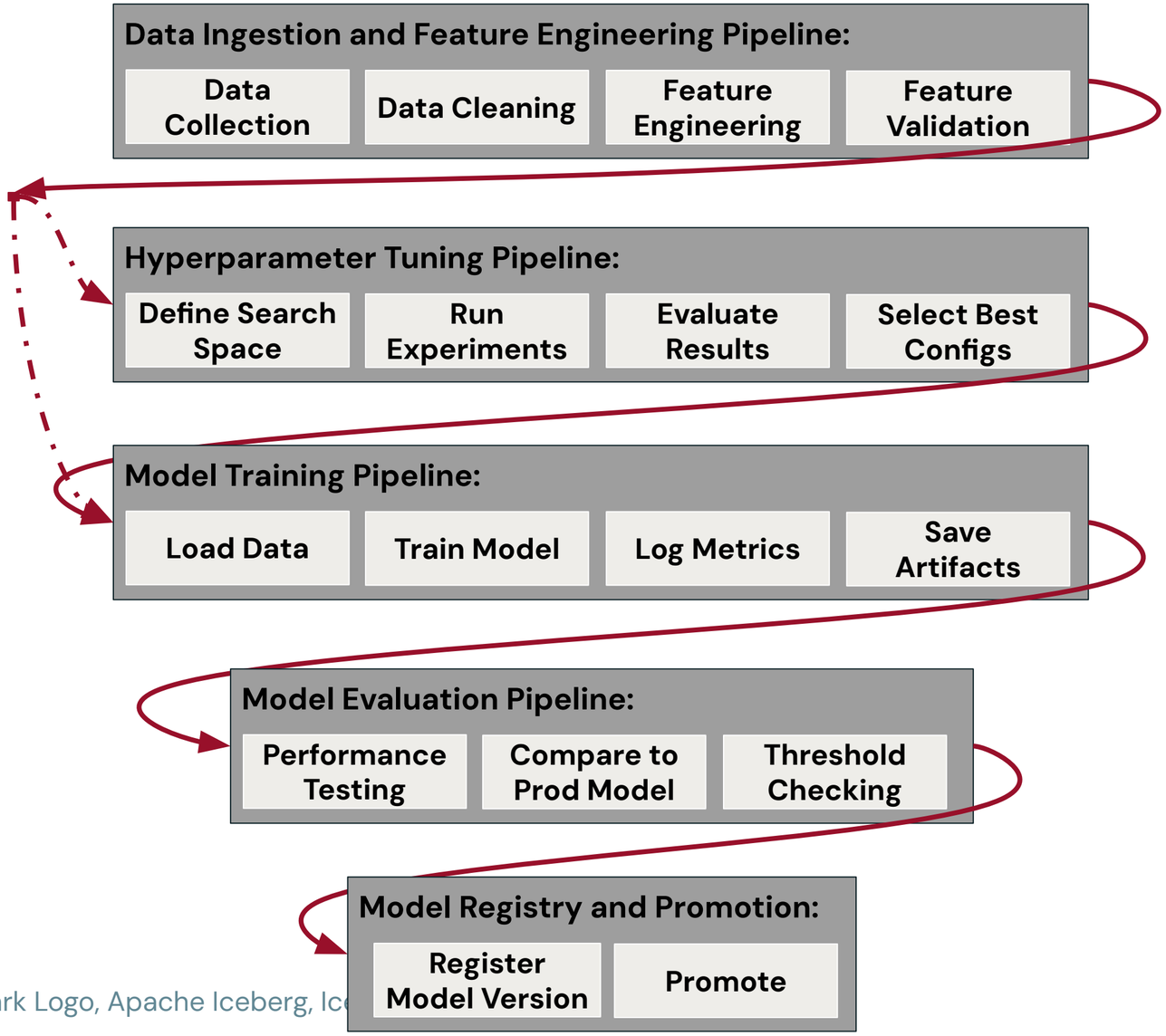
- **Code Testing and Integration Pipeline:**
  - Ensures code changes are merged and tested without breaking the workflow.
- **Data Pipeline Validation Pipeline:**
  - Verifies the integrity of data pipelines and their integration with existing workflows.
- **Feature Engineering Validation Pipeline:**
  - Validates the quality and consistency of features produced by updated feature engineering scripts.
- **Automated Security and Compliance Testing Pipeline:**
  - Ensures that security and compliance checks are in place for code and data handling.



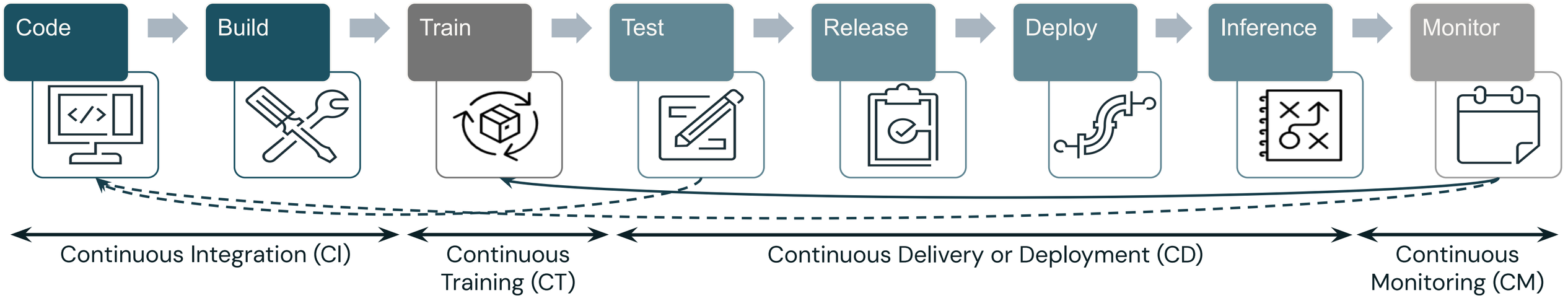
Continuous Training Deeper Dive

Possible Pipelines…

- **Data Ingestion and Feature Engineering Pipeline:**
  - Automate data collection, cleaning, feature creation, and validation.
- **Model Tuning Pipeline *(Optional)*:**
  - Optimize performance by tuning hyperparameters automatically.
  - *May have been previous established.*
- **Model Training Pipeline:**
  - Automate model training with the latest data, logging results and artifacts.
- **Model Evaluation Pipeline:**
  - Test and compare models against production versions to ensure improvements.
- **Model Registry and Promotion Pipeline:**
  - Manage and promote through different lifecycle stages.



Continuous Deployment and Monitoring



Continuous Deployment (CD),

- Validate models in staging before production.
- Deploy code or models automatically across environments.
- Use manual approval ***(Continuous Delivery)*** or automate deployments ***(Continuous Deployment)***.

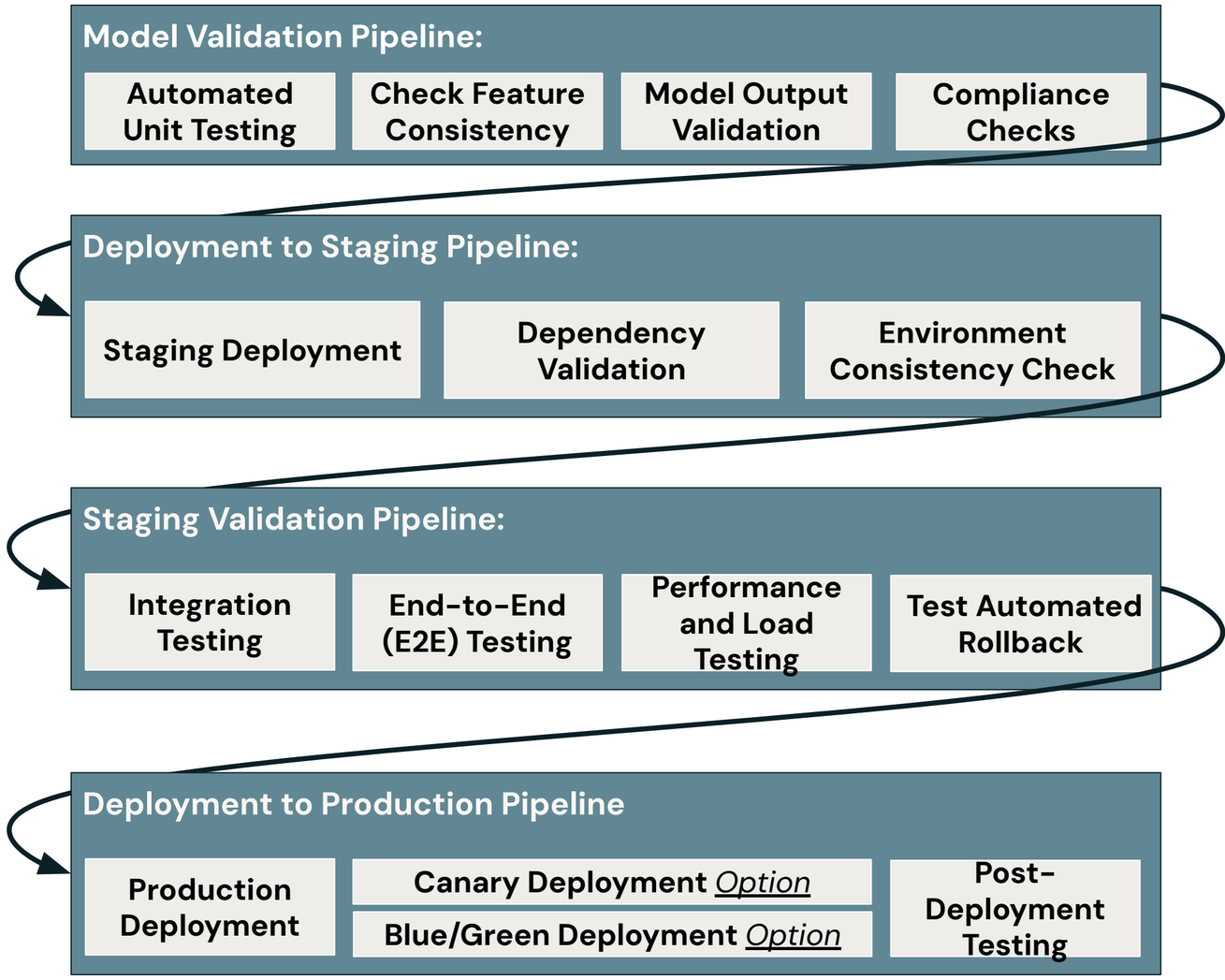
Continuous Monitoring (CM)

- Track model performance and drift in production.
- Automate retraining or notifications based on performance metrics.

Continuous Deployment Deeper Dive

Possible Pipelines...

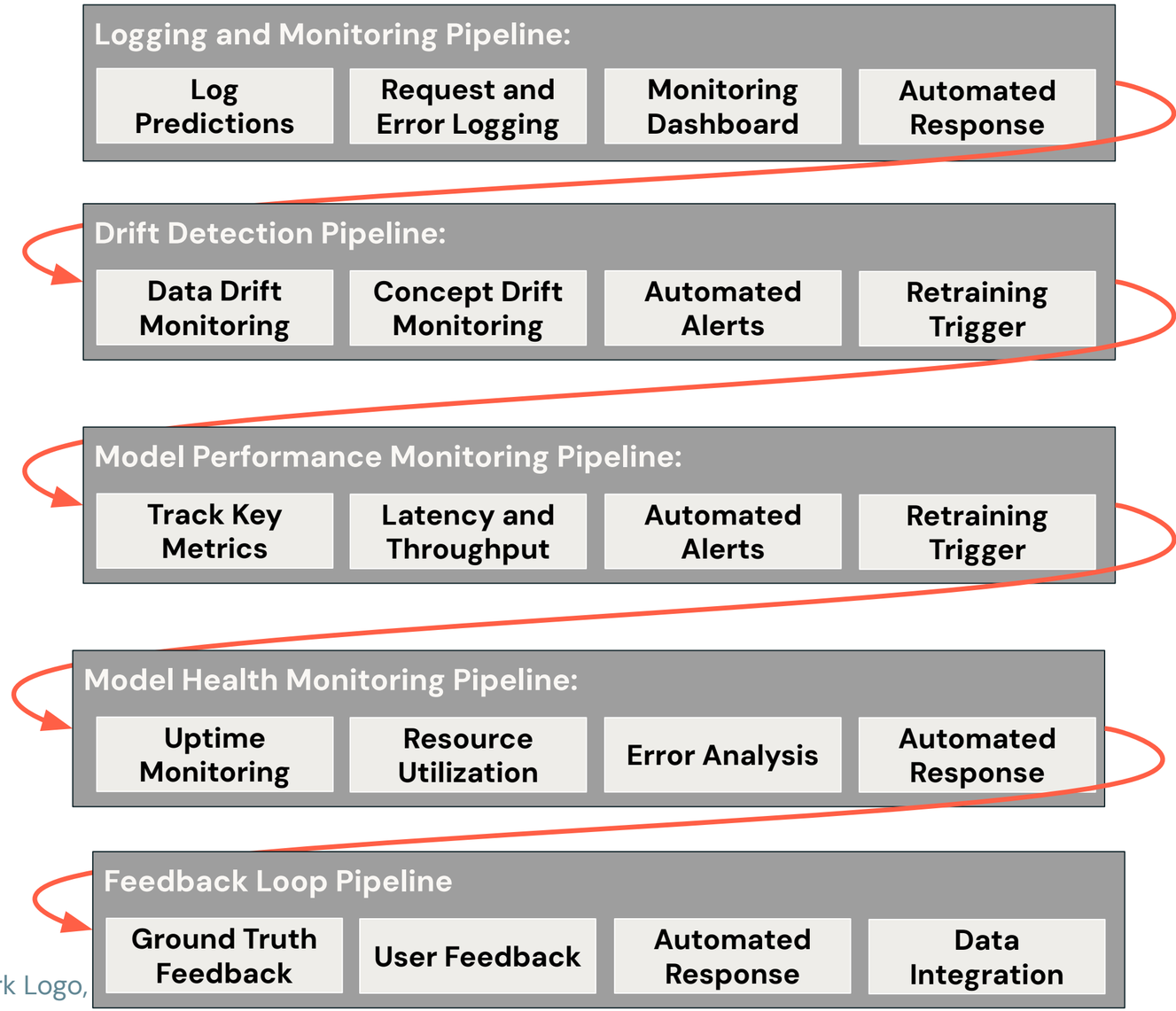
- **Model Validation Pipeline:**
  - Validate models using automated tests and compliance checks.
- **Deployment to Staging Pipeline:**
  - Deploy the model to the staging environment for pre-production testing.
- **Staging Validation Pipeline:**
  - Conduct integration, E2E, and performance testing in staging.
- **Deployment to Production Pipeline:**
  - Move the validated model to production, using options like canary or blue/green deployment.



Continuous Monitoring Deeper Dive

Possible Pipelines...

- **Logging and Monitoring Pipeline:**
  - Logs all predictions, requests, errors, and performance metrics for future analysis.
- **Drift Detection Pipeline:**
  - Continuously monitors data and concept drift, triggering retraining if drift is detected.
- **Model Performance Monitoring Pipeline:**
  - Tracks key performance metrics like accuracy and latency, triggering alerts if performance degrades.
- **Model Health Monitoring Pipeline:**
  - Monitors infrastructure, ensuring that the model endpoint remains online and resources are utilized efficiently.
- **Feedback Loop Pipeline:**
  - Collects real-world feedback and integrates it into the model for continuous improvement.



CI/CT/CD/CM & Recommended Architecture

My Project Repo

Example of: Shared repository

Databricks Tool: Databricks Git Folder

Approval

Example of: Automated Training and Tuning

Databricks Tool: MLflow & Workflow Jobs

Pull Request to Main

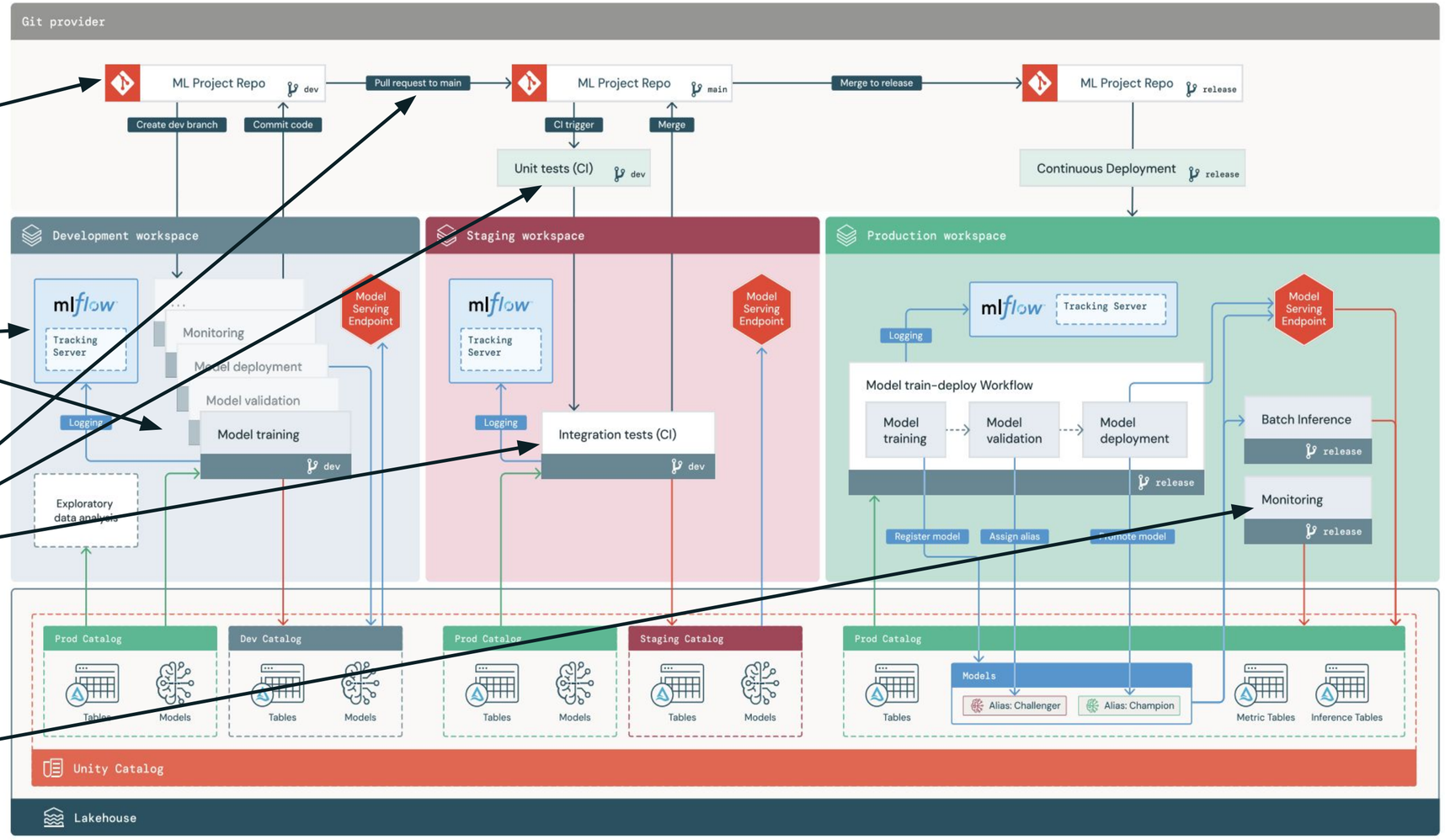
Example of: Triggers automated testing pipelines.

Databricks Tool: Workflow Jobs
Tool: *GitHub Actions & Azure DevOps*

Monitoring

Example of: Monitor Model Performance, Health, and Drift

Databricks Tool: LakeHouse Monitoring & DBSQL



Core Databricks ML CI/CT/CD/CM Tools

Automating and Streamlining Machine Learning Pipelines

- Databricks Workflows
  - Jobs
    - Automates the scheduling and orchestration of ML tasks across different stages.
  - Delta Live Tables
    - Manages ETL pipelines for real-time data, ensuring fresh data for model training.
- Managed MLflow
  - Tracks experiments, manages models, and automates deployment across environments.
- Databricks Git Folder
  - Enables version control of code and notebooks for collaborative development.

databricks

Continuous Workflows for Machine Learning Operations

LECTURE

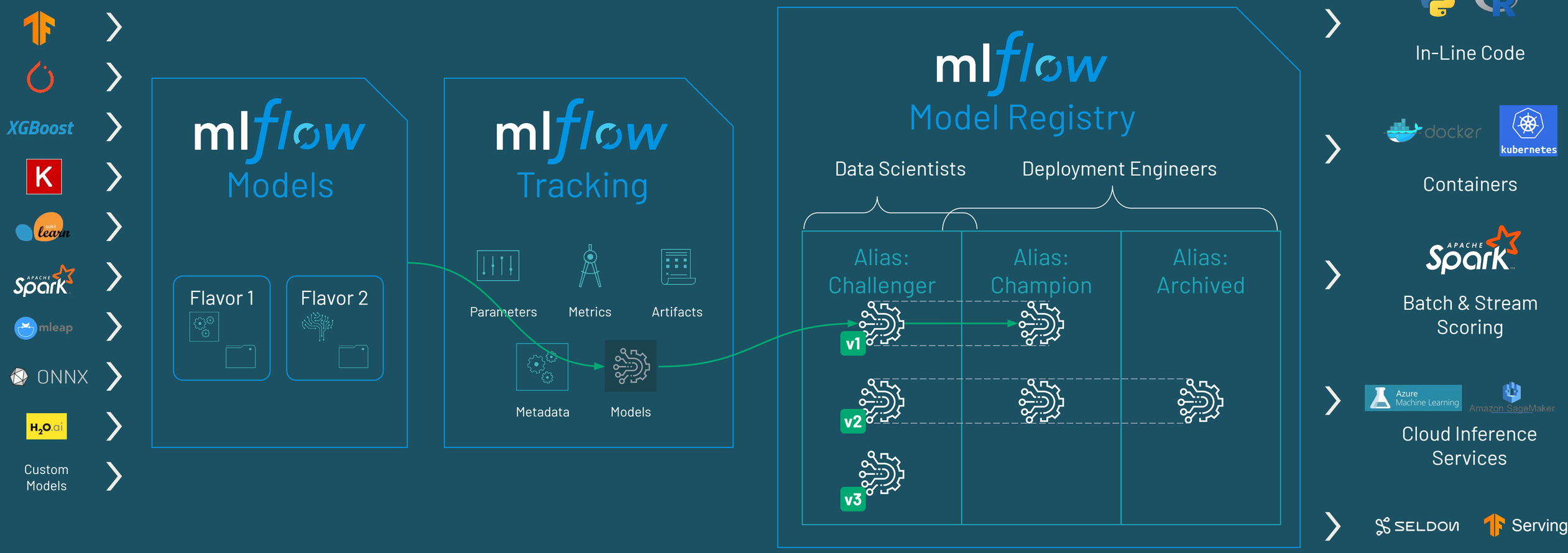
Streamlining MLOps with Databricks

Learning objectives

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

- Understand the importance of modern task orchestration for managing complex multi-data flows in machine learning.
- Explore how Databricks workflows provide fully managed, cloud-based task orchestration.
- Identify the building blocks of Databricks workflows, including jobs, control flows, and triggers.
- Understand how to utilize control flows within Databricks workflows.
- Develop the ability to use simple authoring tools designed for all data practitioners in Databricks.
- Learn how to leverage real-time monitoring to gain actionable insights from workflows.
- Understand how seamless AI integration enhances Databricks workflows.

mlflow Model Lifecycle

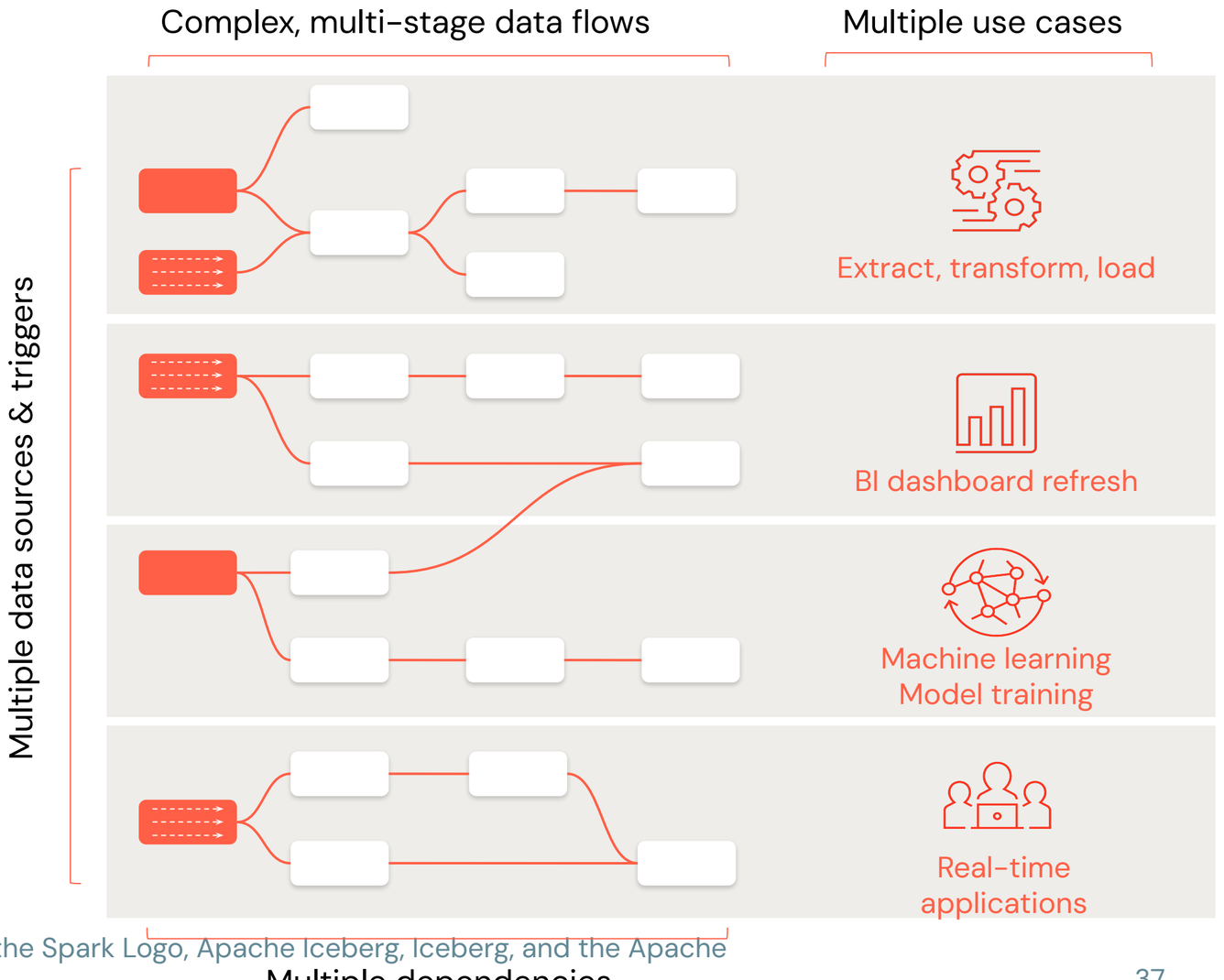


databricks

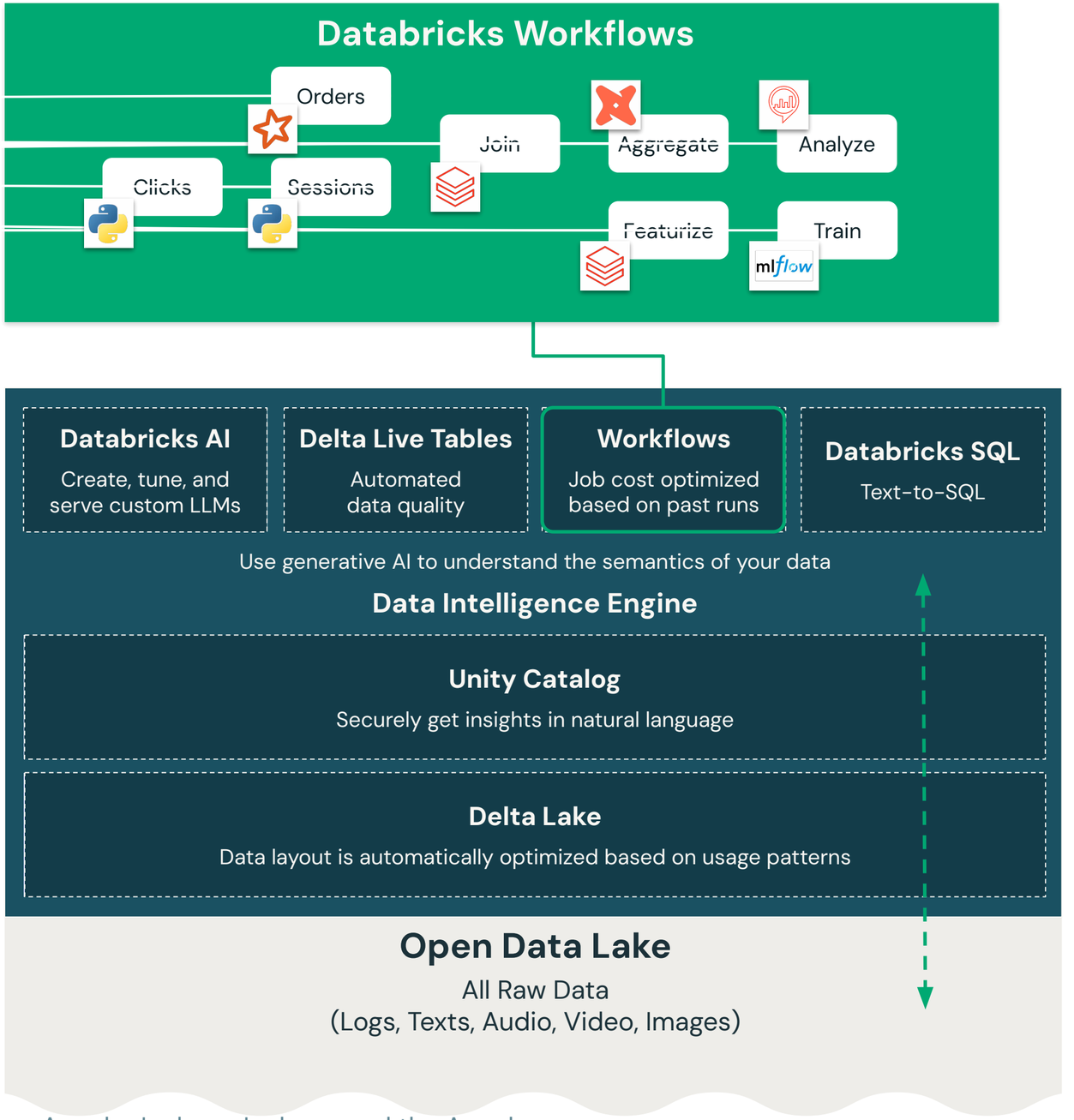
Modern Machine Learning Requires Modern Task Orchestration

Requirements:

- Handle **multiple, varied data sources** and assets including streaming
- Orchestrate complex **multi-step** data flows
- Maintain **dependencies** between different steps and workflows
- Support multiple **use cases** such as ETL, BI and AI including real-time use cases



Databricks Workflows:



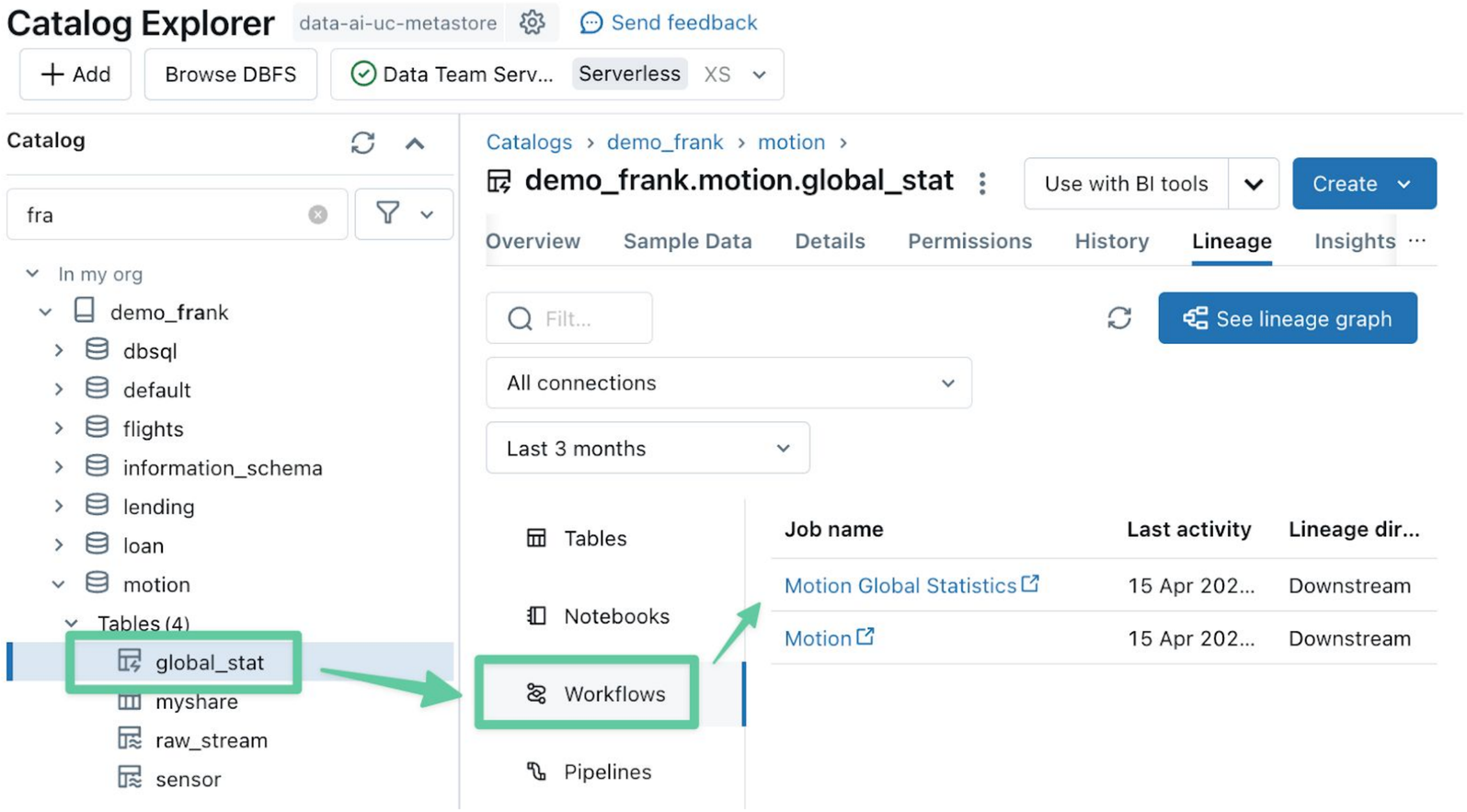
Workflows is a **fully-managed cloud-based general-purpose task orchestration service** for the entire unified Lakehouse.

- Allows you to **build ETL/ML task orchestration** using a simple **UI, CLI, bundles, etc.**
- **No Additional Cost**
- **99.95%** Uptime
- Reduces infrastructure **overhead**
- Cloud-provider **independent**
- **Serverless** Compute is Generally Available (7/2024)
  - *<60 second startup*

Fully Integrated with Unity Catalog

Automated Governance

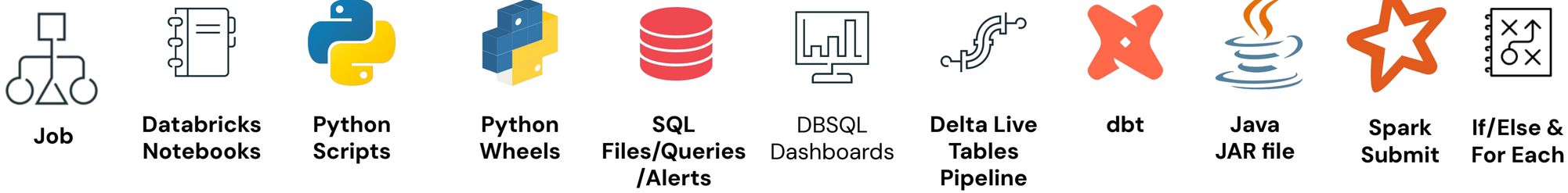
- Automatic real-time data lineage for Workflows
- Connect tables to Workflows and jobs



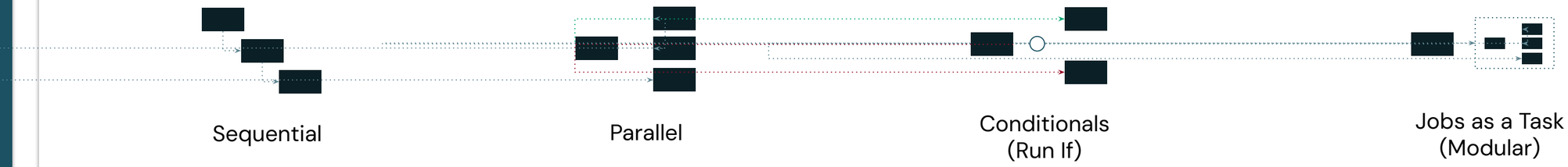
Building Blocks of Databricks Workflows Job

A unit of orchestration in Databricks Workflows is called a Job.

Jobs consist of one or more **Tasks**



Control flows can be established between **Tasks.**



Jobs supports different **Triggers**

Modular Task Management and Orchestration

Source of your Task code:

Databricks Workspace
Quickstart from the File Browser

Remote Git Repository
Simplify CI/CD

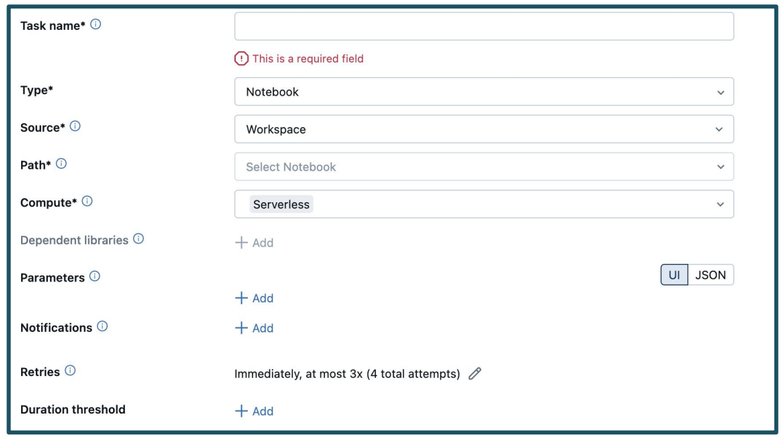
Install external dependent libraries
e.g. JARs, Python Eggs and Wheels
from the following sources:

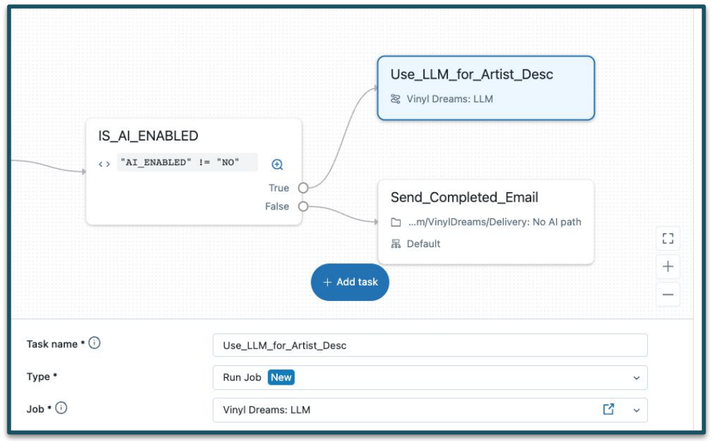
Maven
Workspace
PyPI
DBFS
CRAN
Upload from your machine!

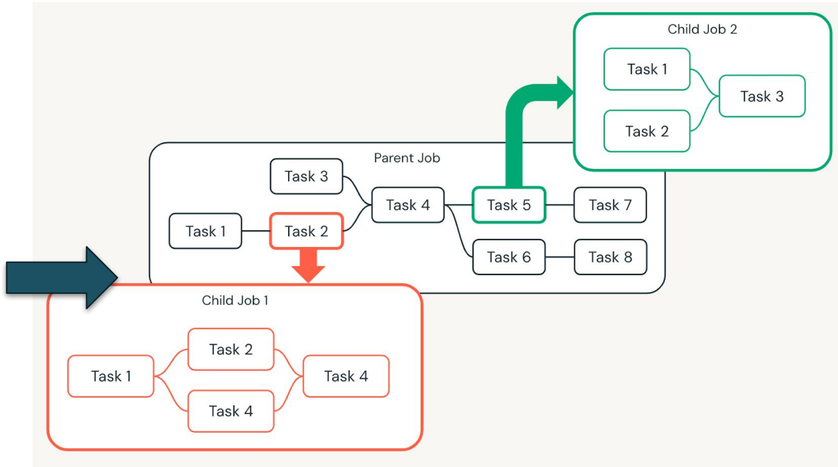
Pass Parameters:

Parameters can be passed to:
Notebook, JAR, Spark Submit, Jobs, SQL, Python Wheel and Script Tasks.

Jobs and Tasks can pass parameters.







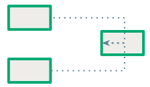
More on Control Flows

Build more **dynamic, mature, and sophisticated** data pipelines with Databricks Workflows!

Task Dependencies

When a task is ***Done***, it can be in a **Success**, **Failure**, or **Excluded** state.

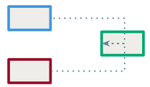
All Succeeded
Default behaviour



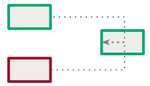
At Least 1 Succeeded
e.g. Fan in with at least some success

None Failed
e.g. Run task(s) at the end of DAG if nothing fails

All Done
e.g. Perform clean up even if tasks have failed or excluded

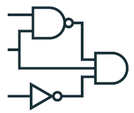


All Least 1 Failed
e.g. Perform clean-up with observability or specific actions



All Failed
e.g. Perform clean-up with observability or specific actions

Parameterisation



Job Parameters
Passed into each Task with behaviour based on the type
e.g. additional options for JARs, spark-submit, Python Args

Dynamic Value references
Special set of templated variables that provide introspective metadata about job and task
e.g. run_id, job_id, start_time

Task Values
Custom parameters that can be shared between Tasks in a Job
e.g. anything that can be programmatically set or retrieved!

Webhooks

Allows customers to build **event-driven integrations** with Databricks.

Supported destinations are **Slack**, **Email**, **Microsoft Teams**, **PagerDuty**, and **Webhooks,** with the below **notification events**:

For example, you can send a message to a Slack #channel when:

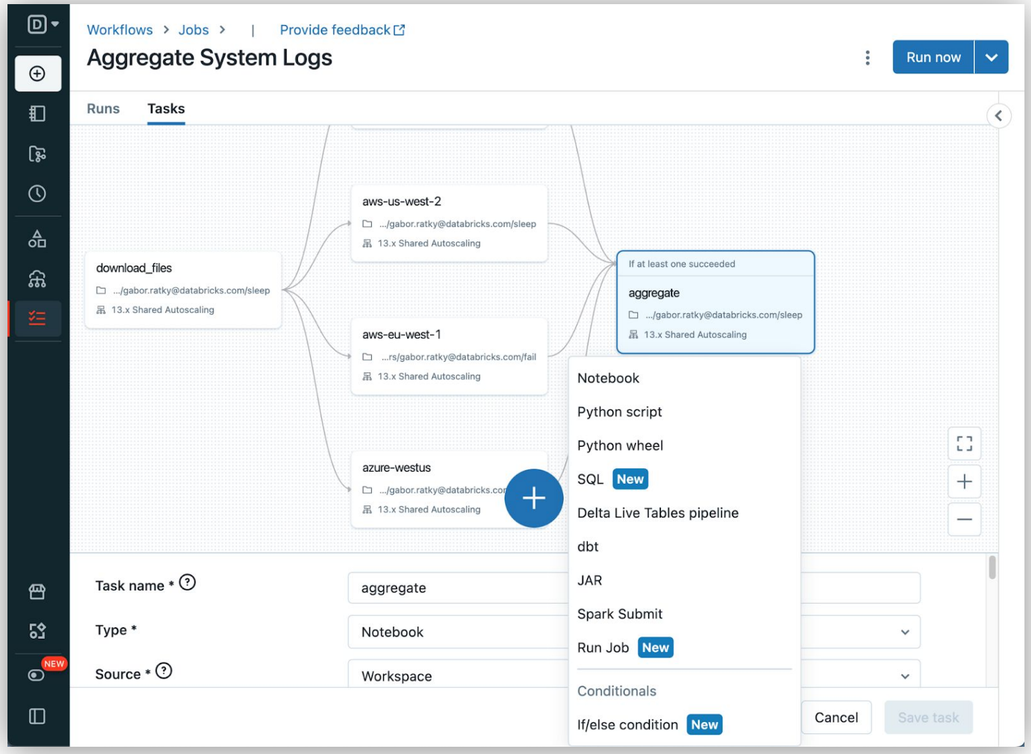
On start: Send a message to a when a job or a parent run is started

On success: when a job or a parent run finished without any errors

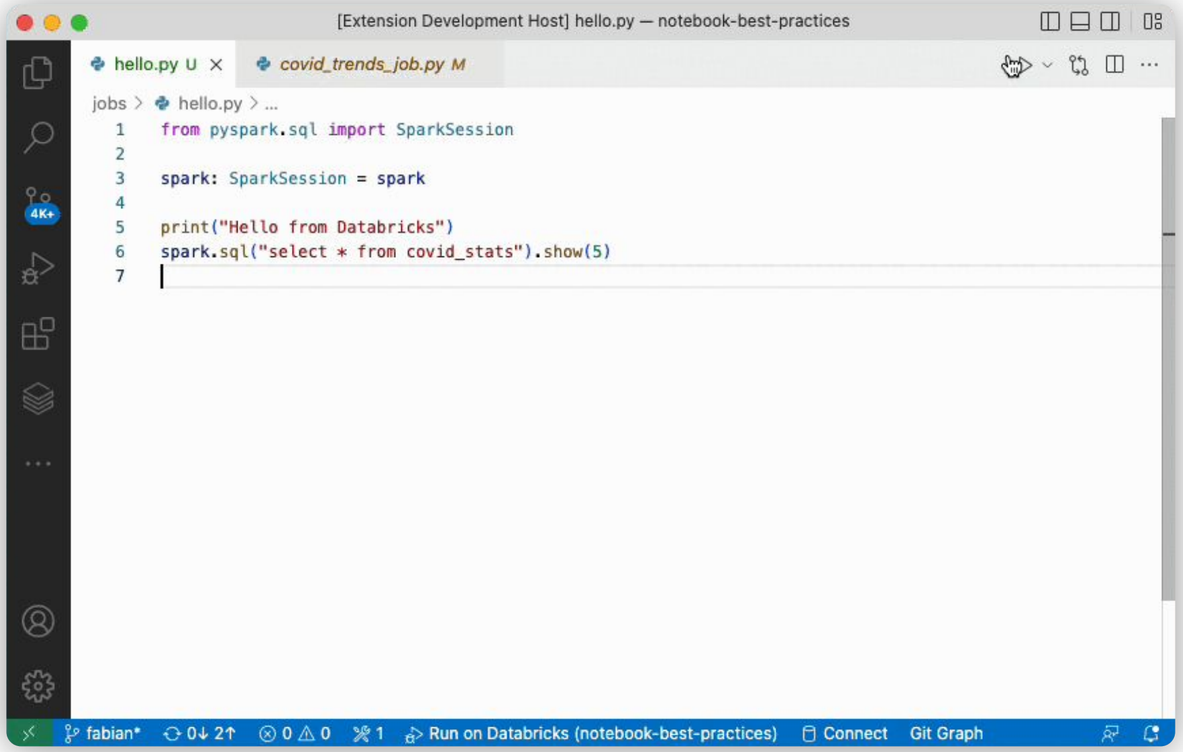
On failure: when a job fails or a parent run is terminated with one of the children in a failed state.

Duration warning: when a job exceeds the configured duration.

Simple authoring for all data practitioners

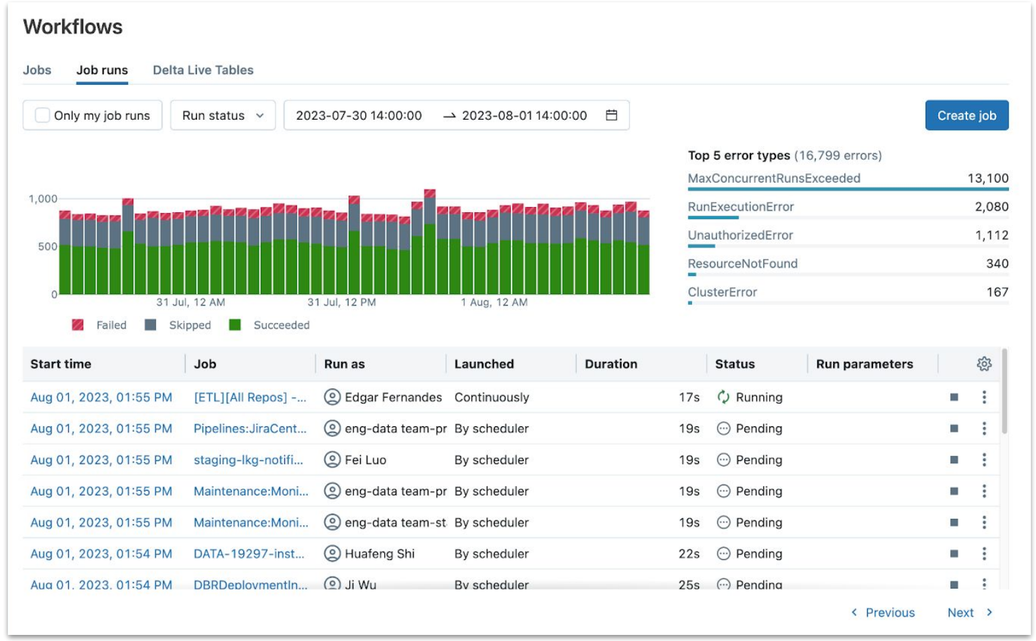


Build sophisticated workflows inside your Databricks workspace with a few clicks

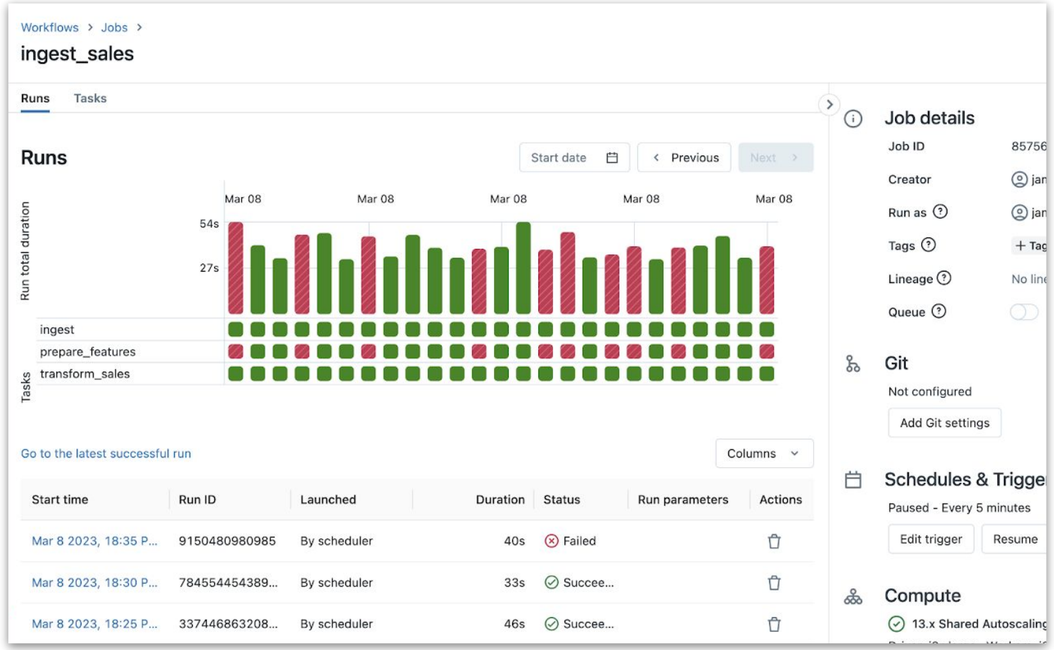


Or connect your favorite IDE to develop workflows locally and run them on Databricks

Actionable insights from real-time monitoring



A simple and intuitive monitoring UI provides real-time metrics and detailed analytics for every workflow run



Drill down to understand which tasks are failing and why. Troubleshoot issues before your customers are impacted

Timeline View of task execution

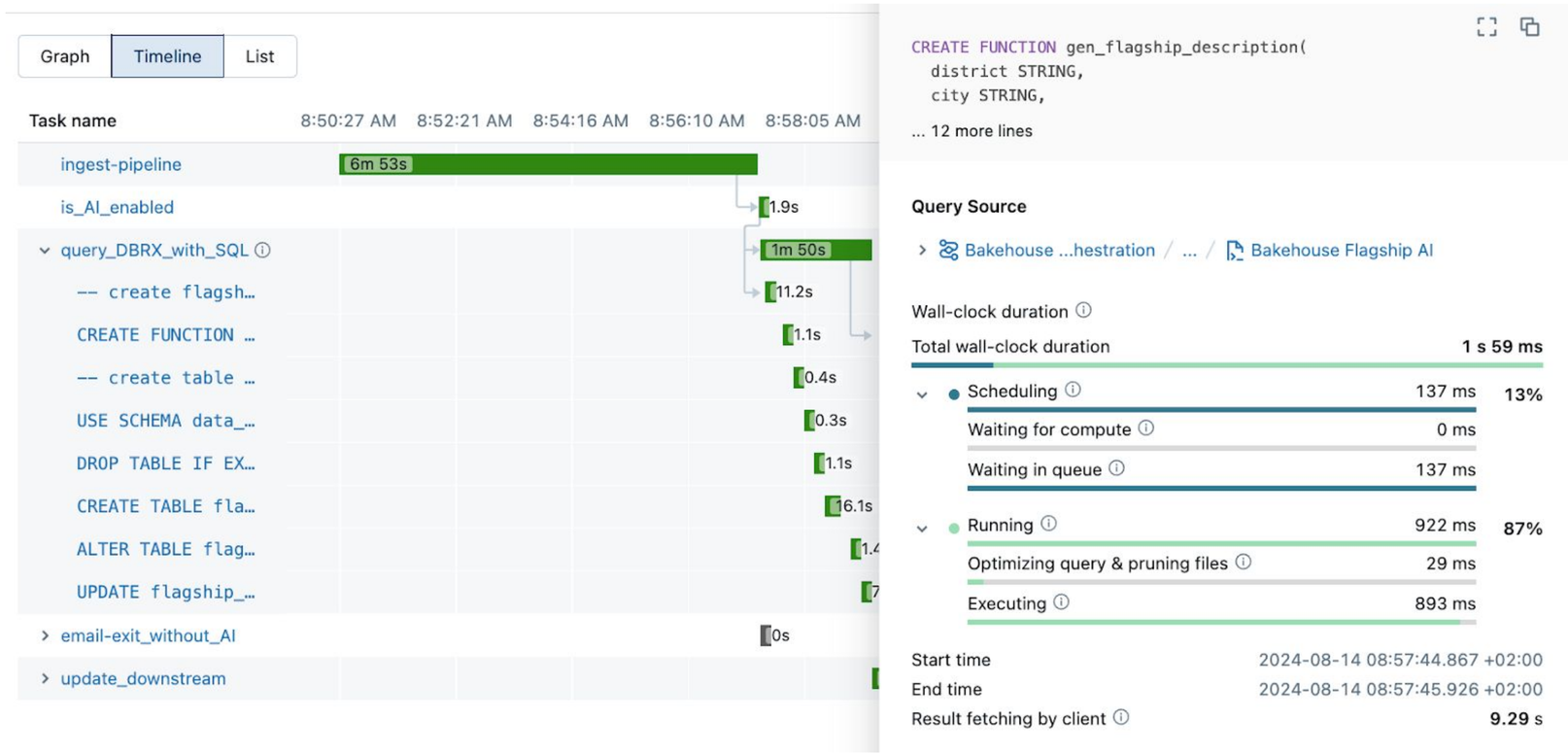
Improved observability for job runs

Visualize critical path for troubleshooting and optimizations

Easily understand non-trivial workflow runs

Find workloads optimization opportunities

Diagnose and retry failed or slow tasks

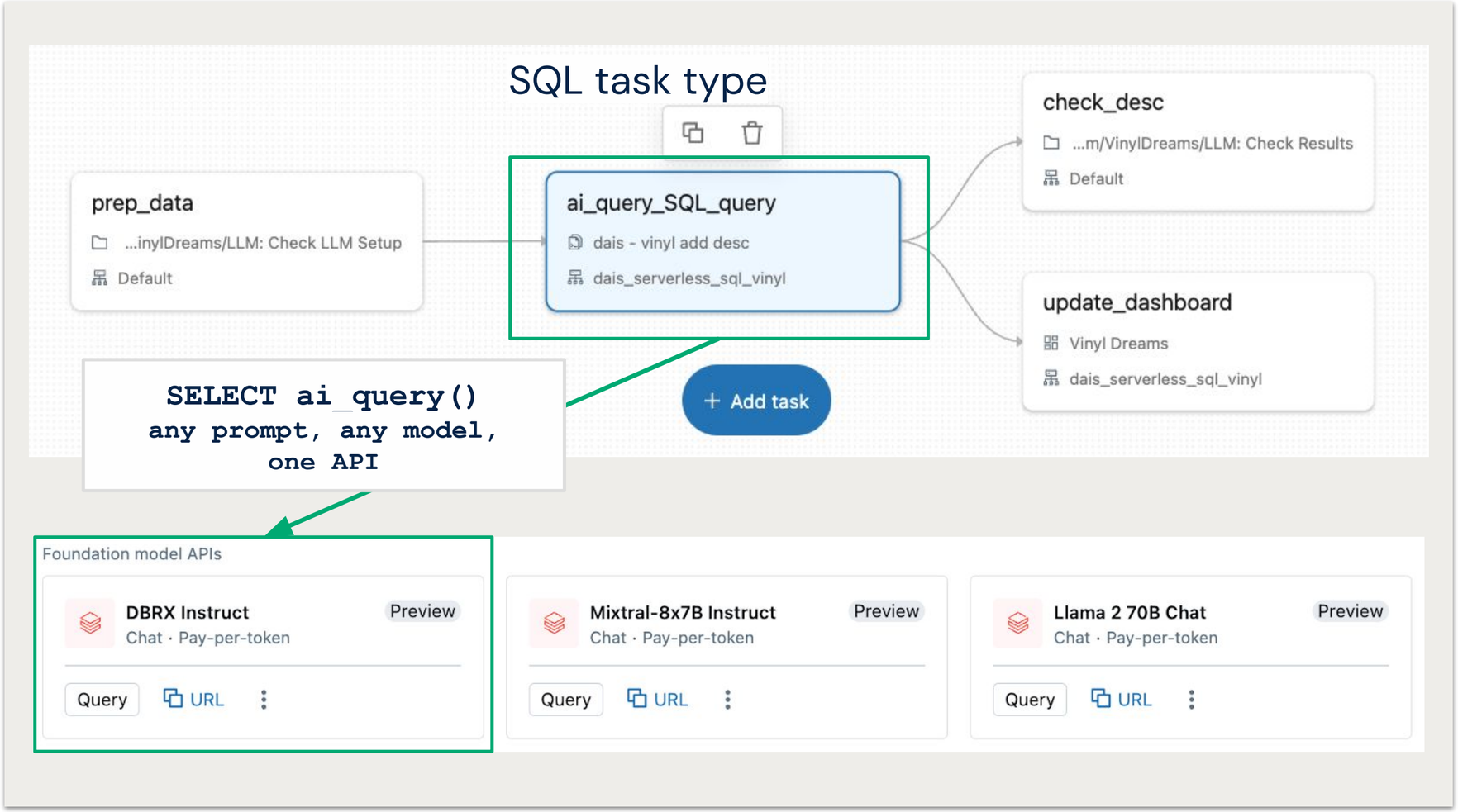


Seamlessly Integrated with AI

Use any foundation model such as DBRX or 3rd party external models

Databricks Workflows

Any Hosted Model

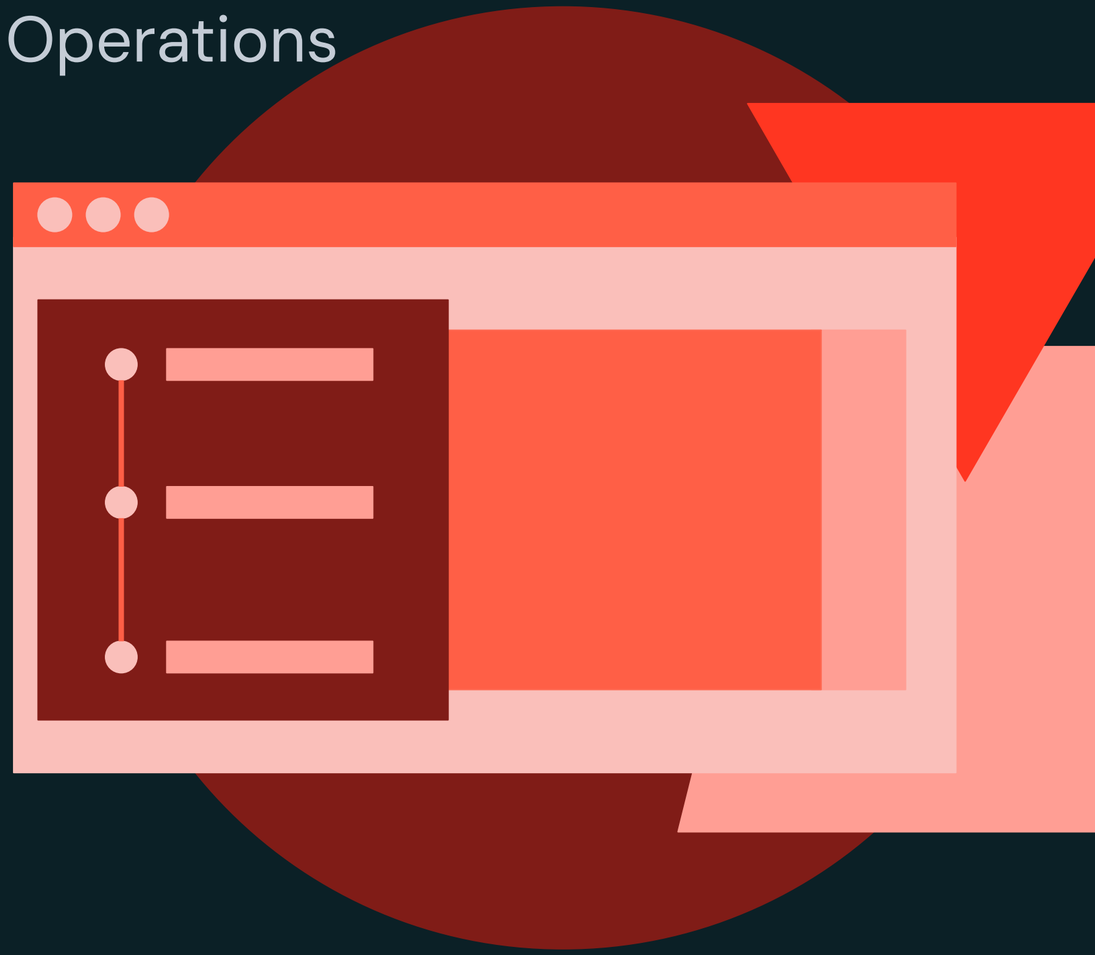


databricks

Continuous Workflows for Machine Learning Operations

DEMONSTRATION

Building a CI/CD Pipeline with Databricks CLI



databricks

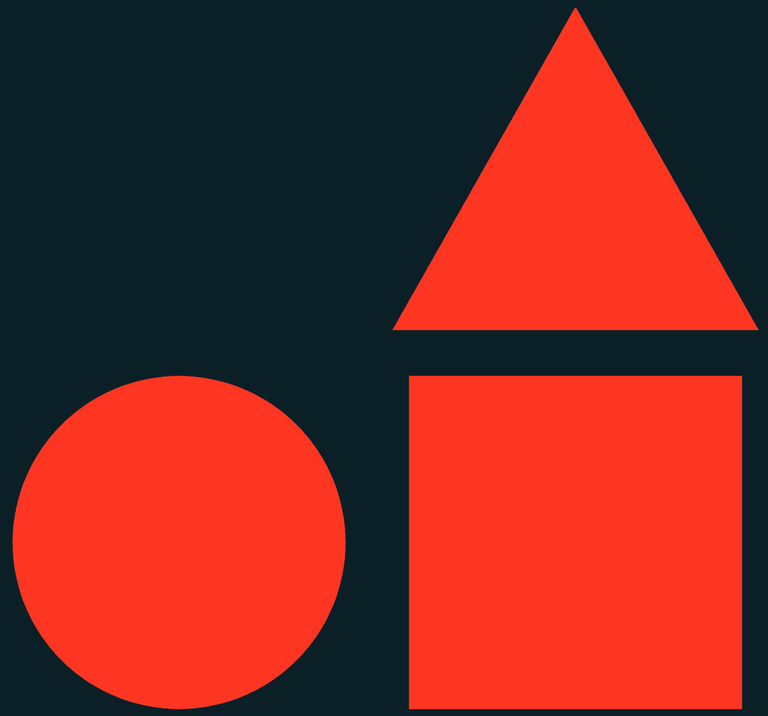
Continuous Workflows for Machine Learning Operations

LAB EXERCISE

Building a CI/CD Pipeline with Databricks CLI

databricks

Testing Strategies with Databricks



Advanced Machine Learning Operations

databricks

Testing Strategies with Databricks

LECTURE

Automate Comprehensive Testing

Learning objectives

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

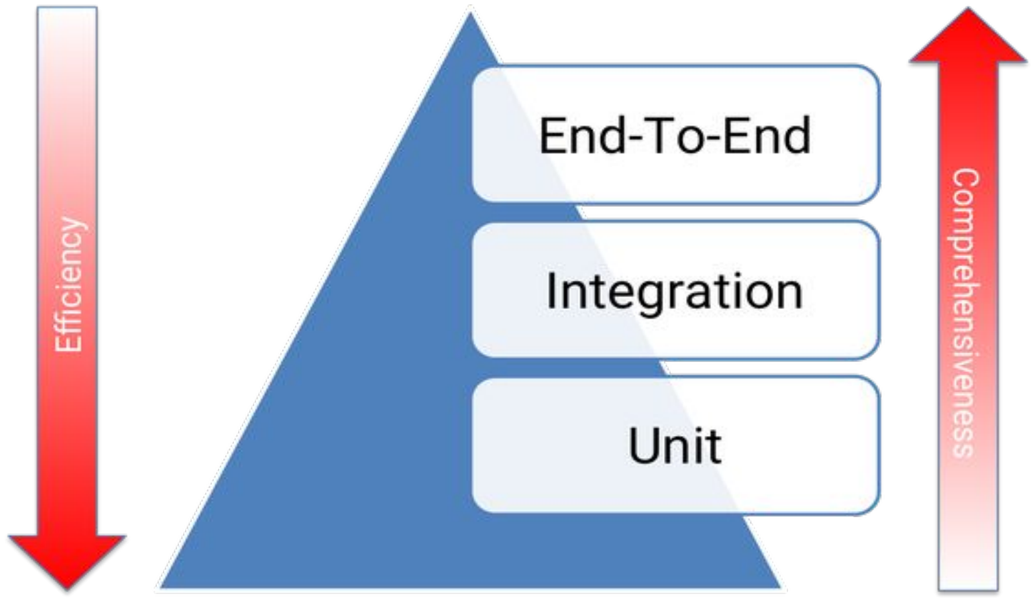
- Explain the structured approach to testing and best practices.
- Describe how Databricks allows for keeping test code separate from your notebook
- Understand common testing frameworks for various languages, e.g. unittest
- Interpret Databricks for end-to-end testing and online evaluation

Testing Pyramid

Illustrating how the test effort should be optimized towards simpler tests, reserving more complex/expensive tests for edge cases

- Unit tests
  - Test only one specific function, on small amount of data
  - Fast, able to run locally
- Integration tests
  - Test business logic on limited amount of data
  - Usually run in a separate environment, as part of CI implementation
- Acceptance/End-to-End tests
  - Run on data, close to production (volume/velocity/...)
  - Run in environment, close to production

Testing Pyramid...



Unit Tests / Small Tests

You can use unit testing to help improve the quality and consistency of your notebooks' code.

- Checks a single component of an application
- Focuses on testing functionality of individual units only
- Does not uncover issues arising when different modules interact

Language	Example Testing Frameworks
Python	unittest, pytest
Scala	ScalaTest
R	testthat

unittest

```
# This function takes a string and returns it reversed
def reverse(s):
    return s[::-1]

import unittest

# Checks if the reverse function correctly reverses
class TestHelpers(unittest.TestCase):
    def test_reverse(self):
        self.assertEqual(reverse('abc'), 'cba')

# Run the unit test and Assert the tests success.
r = unittest.main(argv=[''], verbosity=2, exit=False)
assert r.result.wasSuccessful(), 'Test failed; see logs above'
```

Organize functions and unit tests

A Few Common Approaches for Organizing Functions and Unit Tests

Store functions and unit tests outside of notebooks.

Benefits:

- Functions can be reused across multiple environments.
- Test frameworks are better designed to run tests outside of notebooks.

Challenges:

- Not supported for Scala notebooks.
- Increases the number of files to track and maintain.

Store functions in one notebook and unit tests in a separate notebook.

Benefits:

- Functions are reusable across multiple notebooks.
- Clear separation of logic and testing, making it easier to manage and debug.

Challenges:

- Increases the number of notebooks to manage.
- Functions are limited to use within notebooks.
- Can be more difficult to test outside of notebooks.

Store functions and unit tests within the same notebook.

Benefits:

- Single notebook for easier tracking and maintenance.

Challenges:

- Functions are harder to reuse across notebooks.
- Functions are not easily usable outside of the notebook context.
- Testing is more confined to the notebook environment.

Testing notebooks: using %run

Keep Your Test Code Separate From Your Notebook

- Workflow:
  - Create separate notebook with functions that you want to test
  - Create separate notebook for tests
  - Use %run to import the functions into test notebook
  - Depending on the language, defined test functions or test classes
  - Execute tests

Example: Model Regression Testing

```
# Import the evaluate_model function from the main notebook
%run ./main_notebook

# Load baseline and new model predictions from Unity Catalog
baseline_pred =
   spark.read.table("catalog.schema.baseline_predictions")
new_pred =
   spark.read.table("catalog.schema.new_model_predictions")

# Evaluate RMSE
baseline_rmse = evaluate_model(baseline_pred, "label", "prediction")
new_rmse = evaluate_model(new_pred, "label", "prediction")

# Check if the new model is within 5% of the baseline RMSE
assert new_rmse <= baseline_rmse * 1.05, f"Test failed: New RMSE
{new_rmse} exceeds baseline {baseline_rmse}"

print(f"Test passed: New RMSE = {new_rmse}, Baseline RMSE =
{baseline_rmse}")
```

What Should You Test in ML?

Essential Aspects to Validate in Machine Learning Pipelines

Data Validation

- Data is in the correct form and free from errors.

```
def test_data_no_missing_values():
    data = load_data()
    assert data.isna().sum().sum() == 0,
    "Data contains missing values"
```

Data Transformations

- Ensure transformations are carried out properly.

```
def test_data_normalization():
    data = load_data()
    normalized_data = normalize(data)
    assert normalized_data.max() <= 1 and
    normalized_data.min() >= 0, "Normalization
    failed"
```

Modeling Functions

Test functions written to build and operate your model

```
def test_loss_function():
    y_true, y_pred = [0, 1, 0], [0.1, 0.9, 0.2]
    loss = calculate_loss(y_true, y_pred)
    assert abs(loss - expected_loss) <
    tolerance, "Loss calculation is incorrect"
```

Model Integration

Model integrates and functions well with the system

```
def test_model_api_integration():
    resp = client.post('/predict',json={'input': sample})
    assert resp.status_code == 200, "Model API
    integration failed"
    assert resp.json()['prediction'] == expected_output,
    "Incorrect prediction from API"
```

Testing notebooks Using Testing Frameworks

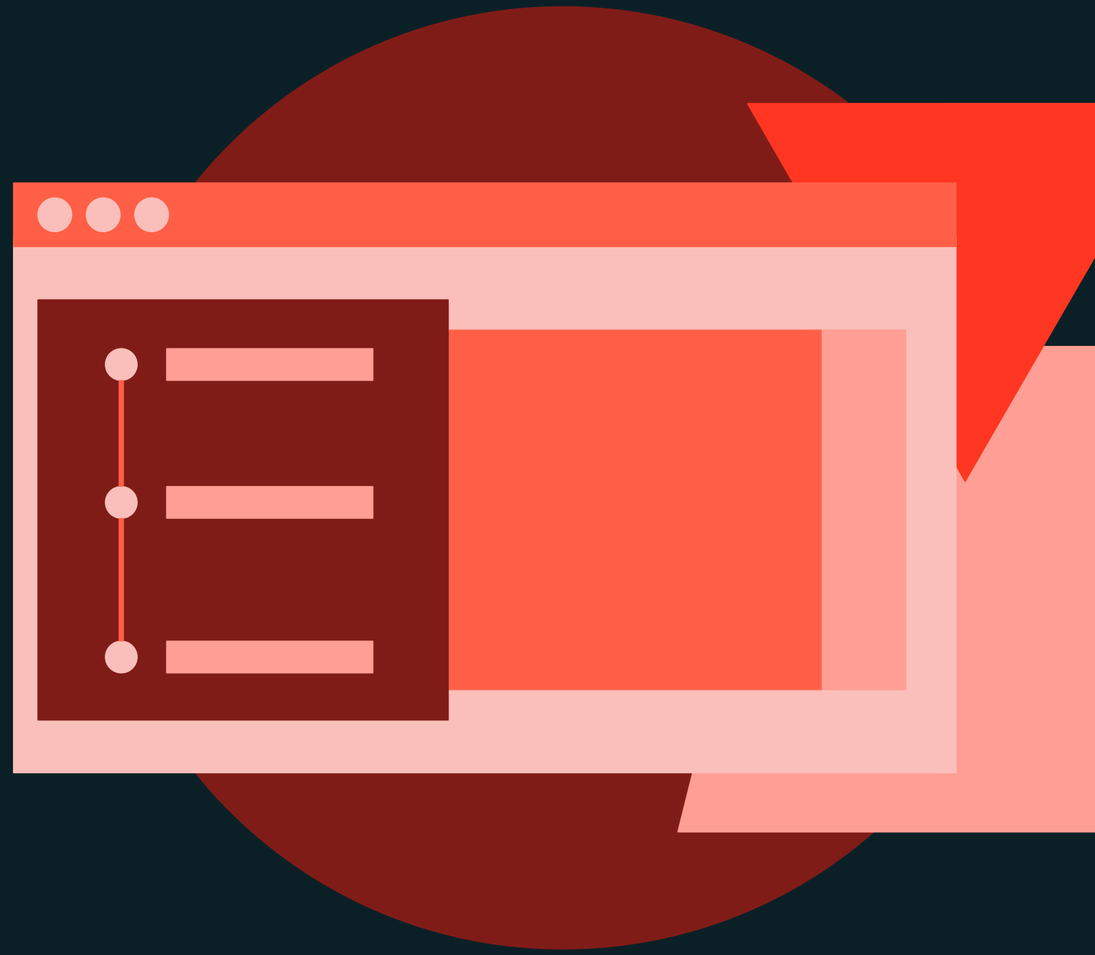
- Pytest
  - Popular testing framework for Python.
  - Supports fixtures, parameterized tests, and built-in assertions.
  - Works well with Databricks Python notebooks by importing functions for isolated tests.
- Testthat
  - Primarily used for R notebooks.
  - Allows robust unit testing with easy-to-write test cases.
  - Supports reproducibility and detailed error messages for R code.
- ScalaTest
  - Widely used for testing Scala code in Databricks notebooks.
  - Supports behavior-driven and test-driven development.
  - Provides support for both unit testing and integration testing of Spark jobs.
- SQL
  - SQL queries can be validated using assertions in notebooks.
  - Useful for testing data transformations, aggregations, and schema validation.
  - Ensures query outputs match expected results in unit tests.

databricks

Testing Strategies with Databricks

DEMONSTRATION

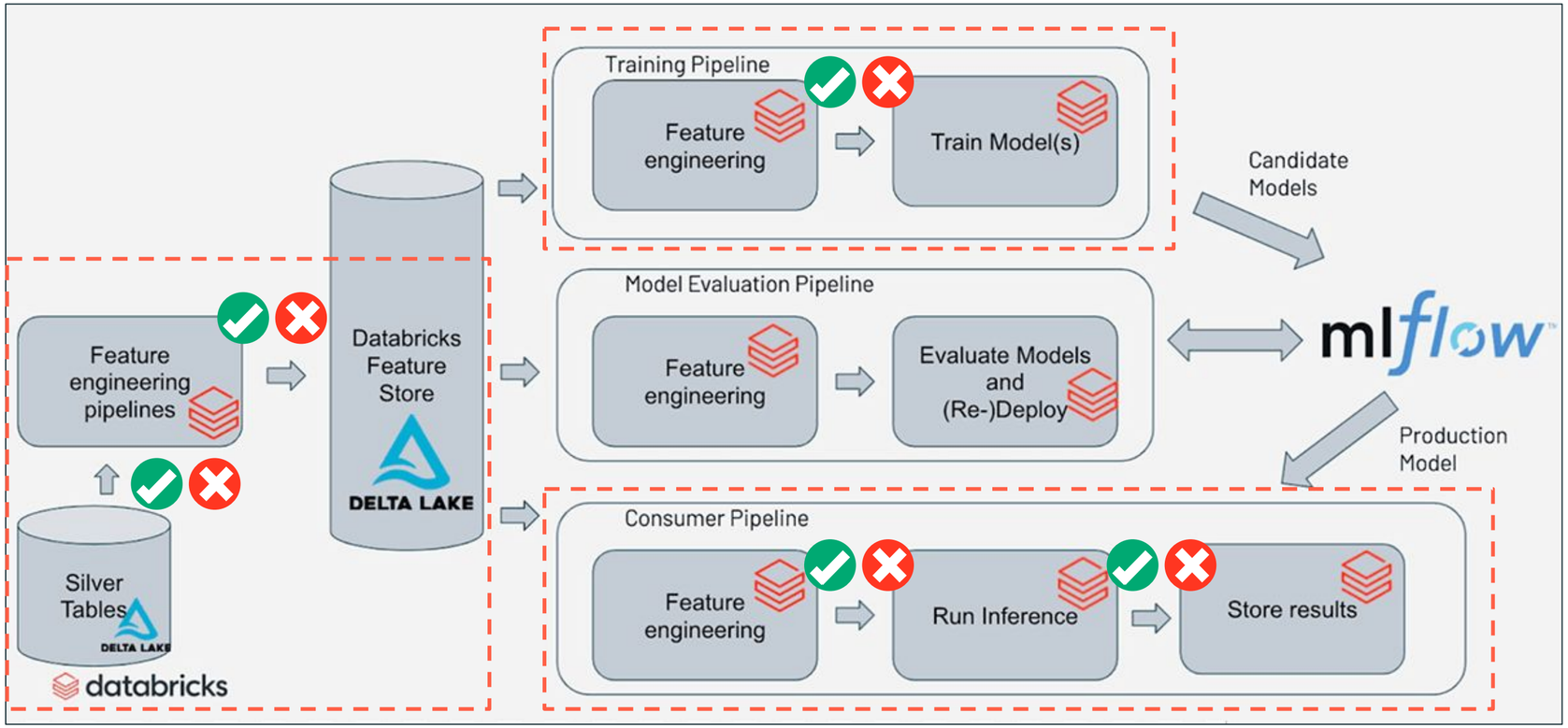
Common Testing Strategies



Integration Tests

Ensure that different components of your system work together correctly.

- Identifies issues, ensures end-to-end functionality, and verifies requirements.
- Tests combined modules together and ensures they are interacting and working as expected
  - e.g. feature engineering and model training, inference and monitoring
- Often performed in Staging Environment which should match the production environment as closely as possible.



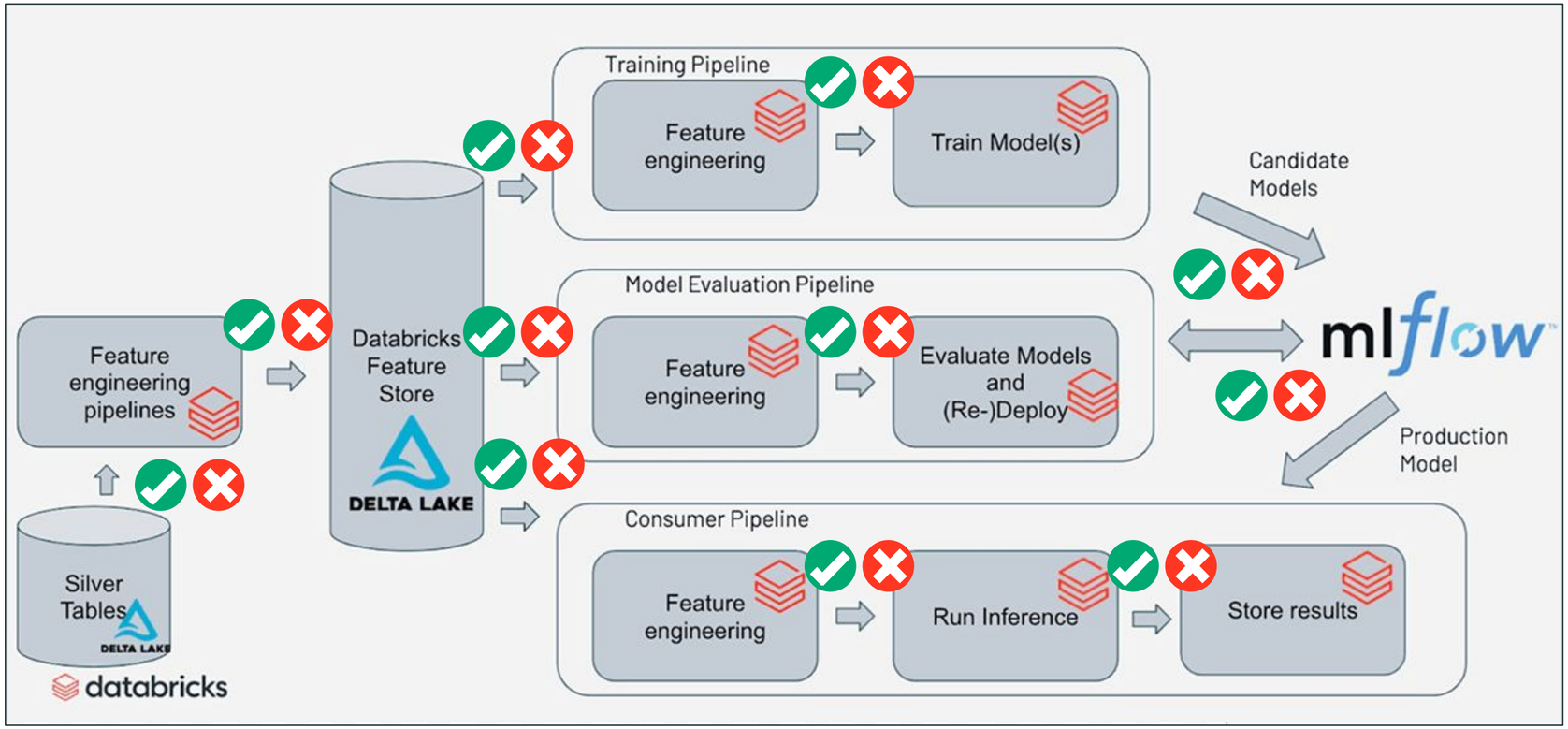
Suggestions for Time and Cost Reductions:

- Use small subsets of data or run fewer training iterations.
- Instead of a full-scale load testing in integration tests, just test small batch jobs or requests to a temporary endpoint.

End-to-End Tests (E2E)

Validates the entire system workflow from start to finish, mimicking real-world scenarios.

- Covers the complete process, from data ingestion to final output, including all intermediate steps.
- Crucial to ensure operational separation between development (dev), staging, and production (prod) environments



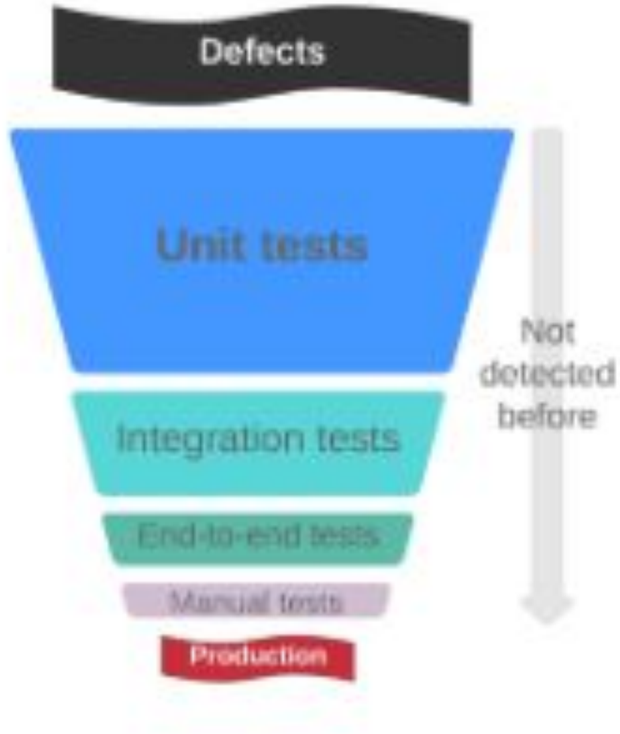
Suggestions:

- Run on data, close to production (volume/velocity/...)
- Run in environment, close to production

Optimize tests in the CI/CD process

We cannot afford to run all the tests for every change at any stage of the pipelines, we need to be more selective.

- **“Zero-defect in production” is not a goal these expectations would not lead to a good ROI**
- Funnel for catching defects:
  - **Catch** them as **soon as possible**
    - The **later** defects are detected, the **more expensive** their handling.
  - Implement tests for catching **defects that cannot be caught earlier**.
- Ensure the ability to **detect and roll back** issues **in production**
  - e.g. using additional approaches such as canary deployments, experiments, probing, etc.

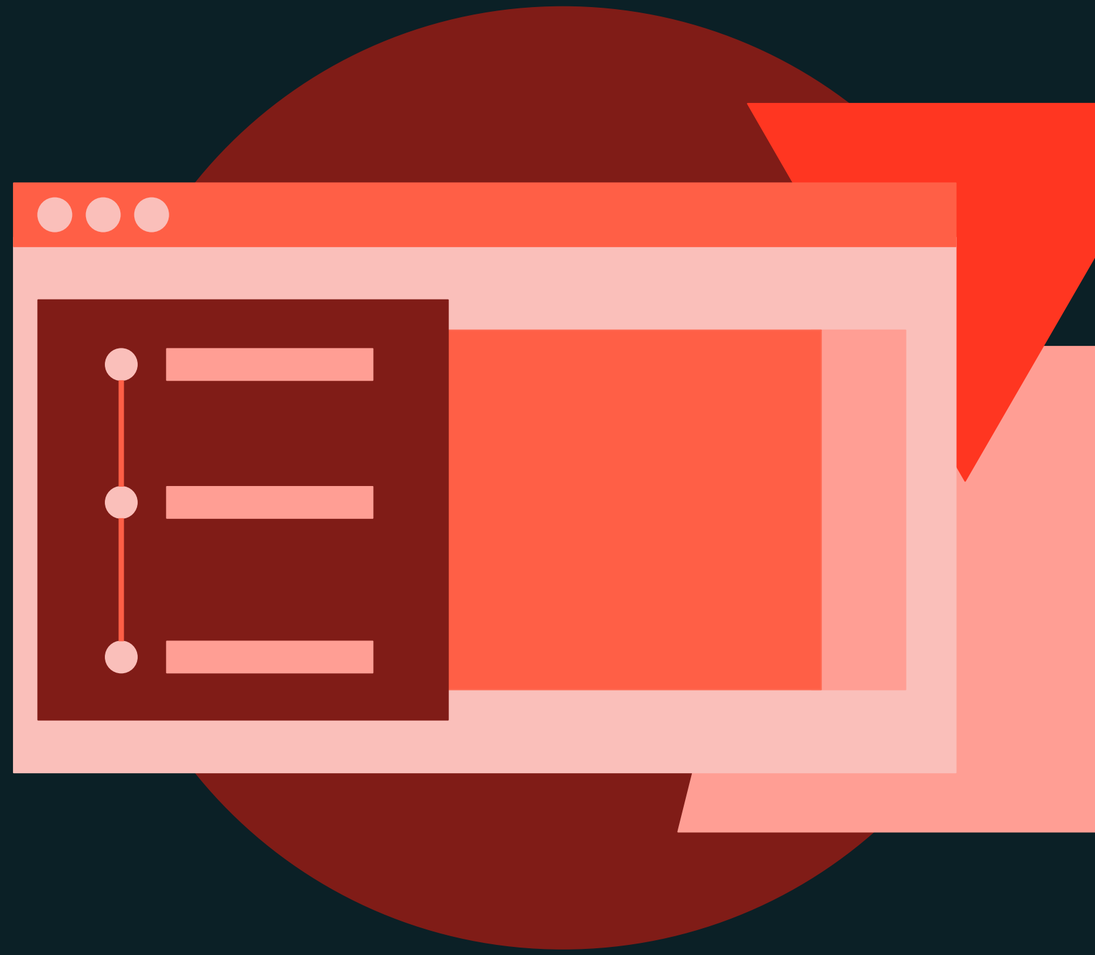


databricks

Testing Strategies with Databricks

DEMONSTRATION

Integration Tests with Databricks Workflows



databricks

Testing Strategies with Databricks

LECTURE

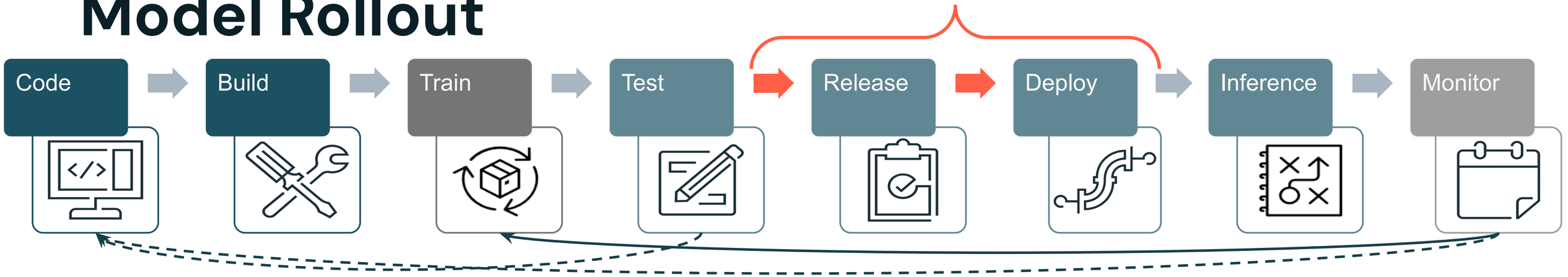
Model Rollout Strategies with Databricks

Learning objectives

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

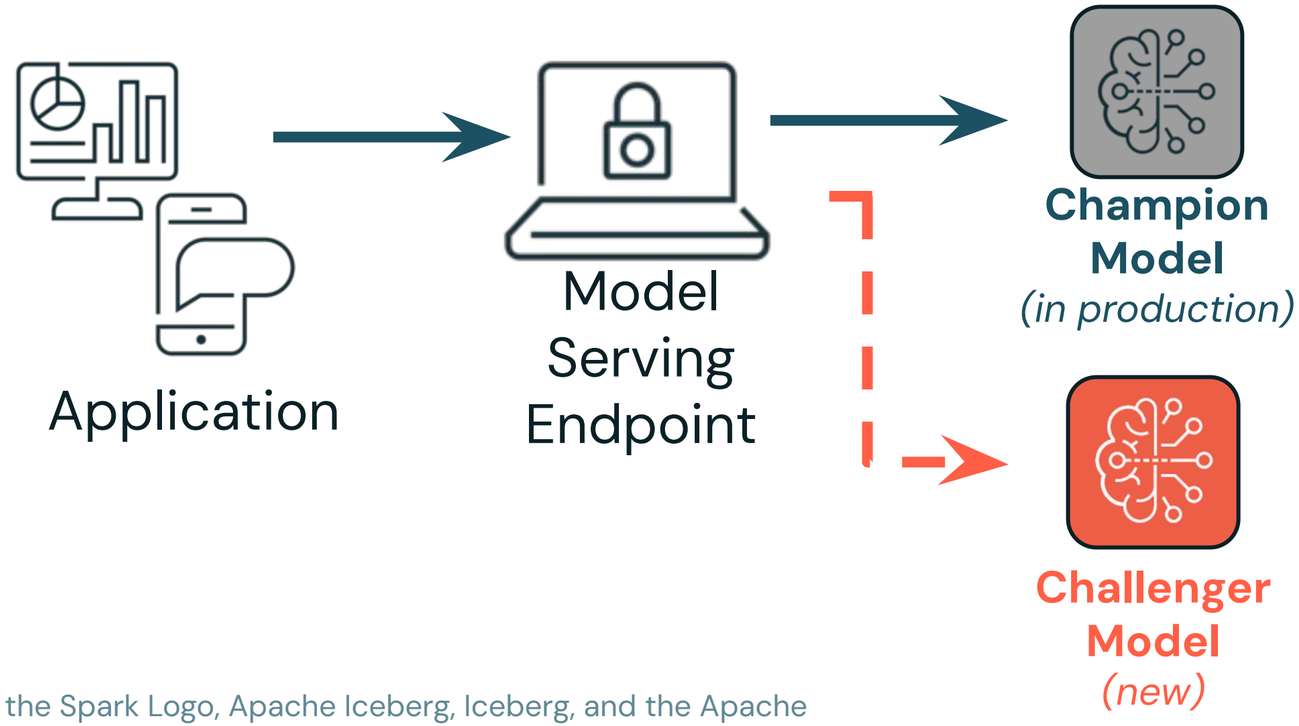
- Describe various rollout strategies: shadow, rolling, blue-green, canary, A/B testing
- Describe A/B testing framework and metrics
- Understand how Mosaic AI Model Serving solves key challenges for model rollout strategies

Model Rollout



Objective

Determine whether to continue with the current production model (Champion Model) or replace it with the new version (Challenger Model).



Model Rollout Strategies

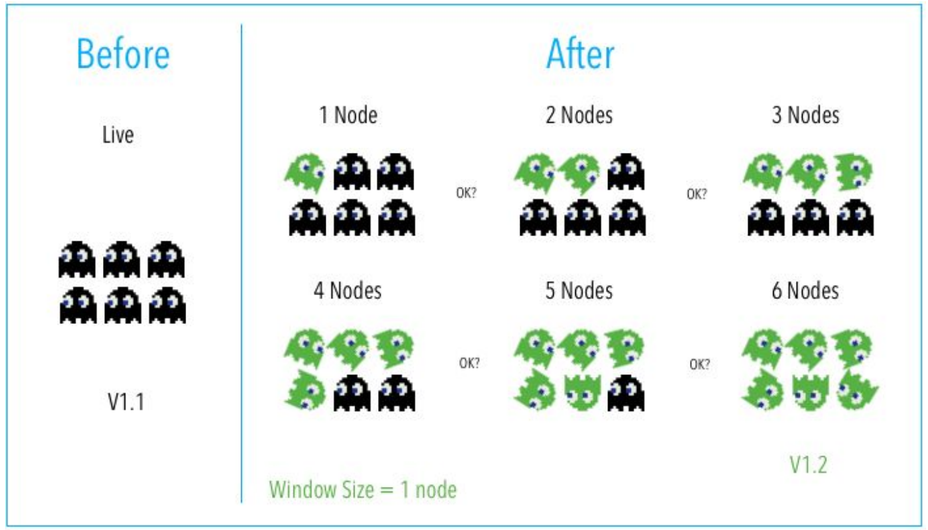
Shadow and Rolling

Shadow

New deployment shows existing system but not used for decision making.

Rolling

Updates nodes in a target environment incrementally in batches with the new version.



Model Rollout Strategies

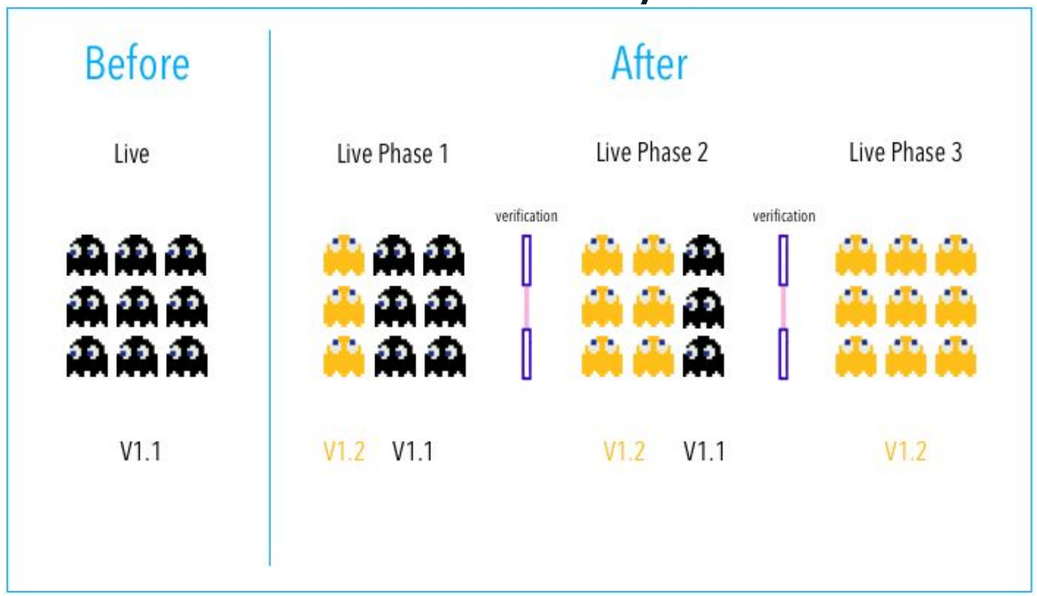
Blue-Green and Canary

Blue-Green

Utilizes two identical environments, a “blue” and a “green” environment with different versions of an application or ser

Canary

Releases an application or service incrementally to a



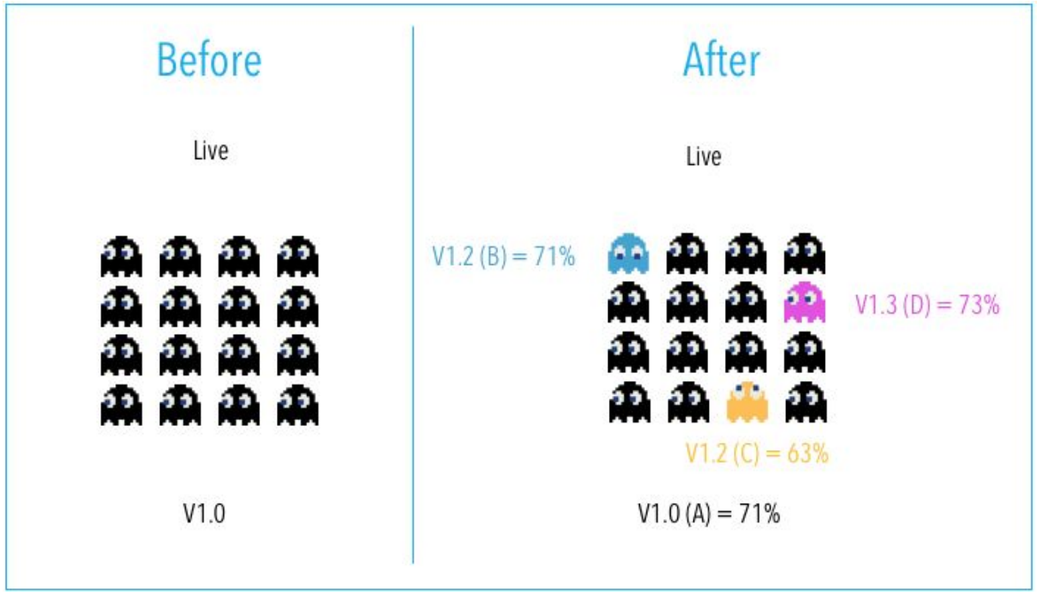
Model Rollout Strategies

A/B Testing

Different versions of the same service run simultaneously in the same environment for a period of time.

Framework: A/B testing is essentially an experiment where two or more variants of a deployment are shown to users at random, and statistical analysis is used to determine which variation performs better for a given conversion goal.

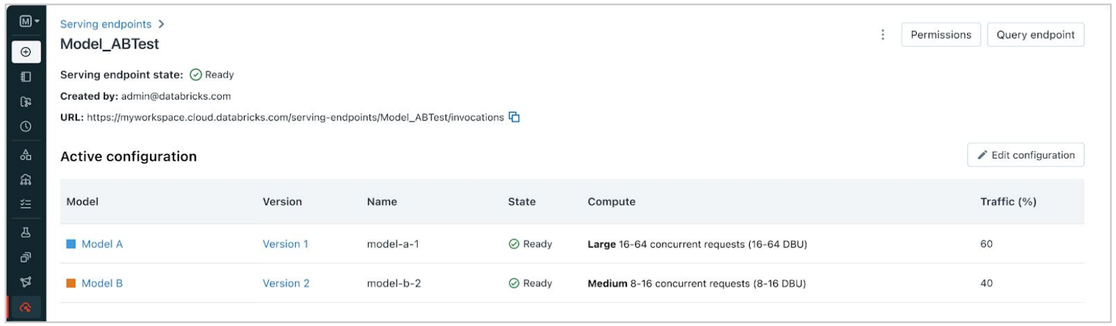
- Collect data
- Identify goals
- Generate hypothesis
- Create variations
- Run experiment
- Analyze results

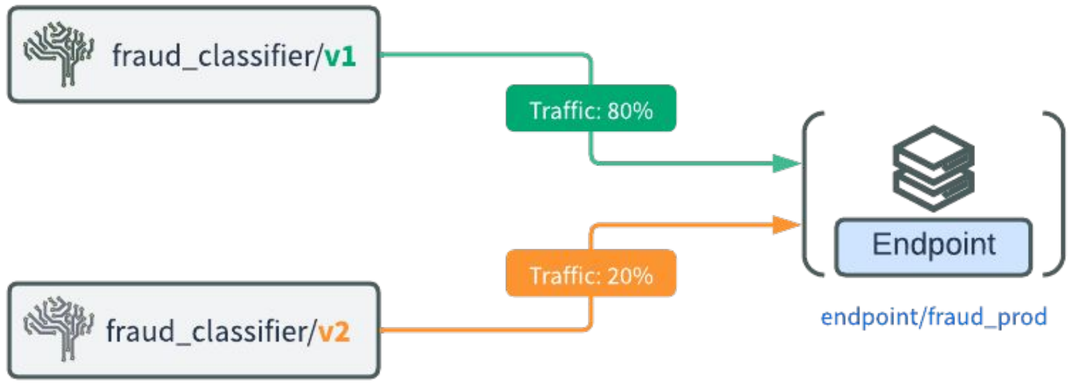


Databricks Model Serving

Online evaluation

- Supports online evaluation strategies such as A/B testing or canary deployments through the ability to serve multiple models to a serving endpoint



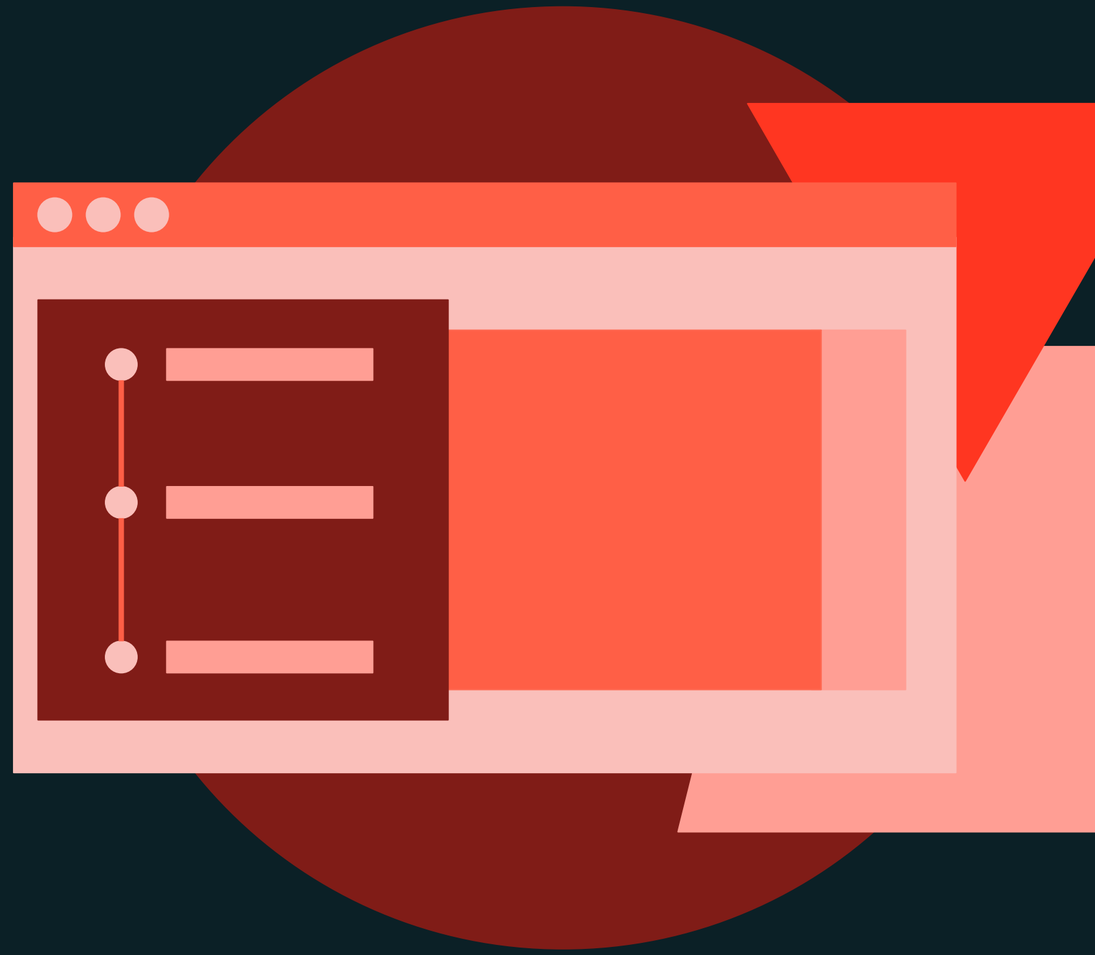


databricks

Testing Strategies with Databricks

DEMONSTRATION

Model Rollout Strategies with Mosaic AI Model Serving



databricks

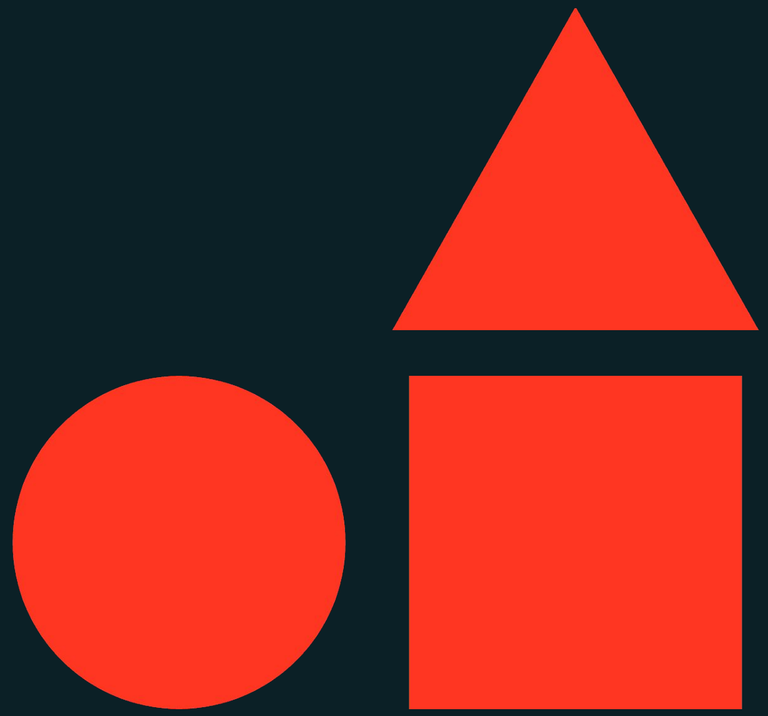
Testing Strategies with Databricks

LAB EXERCISE

Rollout Strategies with Workflows

databricks

Model Quality and Lakehouse Monitoring



Advanced Machine Learning Operations

databricks

Model Quality and Lakehouse Monitoring

LECTURE

Lakehouse Monitoring

Learning objectives

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

- Identify differences between classic monitoring and modern MLOps monitoring
- Explain what problems Lakehouse Monitoring solves
- Explain various capabilities Lakehouse Monitoring
- Understand how to use Databricks SQL queries to build custom metrics and how to setup SQL alerts
- Explain how DBSQL and AI/BI Dashboards help with monitoring within Databricks
- Explain how to integrate monitoring and alerts with Databricks Workflows

Monitoring in Machine Learning

Types of Monitoring and Types of Drift

Types of Model Monitoring

Business Requirements

- Ensuring the ML solution **aligns with and fulfills specific business objectives**.

Resource Utilization

- Ensuring **efficient resource utilization** within the ML infrastructure.

Traceability

- **Tracking data lineage** to understand the origin, movement, and transformation of data.

Model Performance

- Detecting and addressing types of **drift or degradation**.

Types of Drift

Data Drift:

- Occurs when the **statistical properties** of the input data **change over time**.

Concept Drift:

- Happens when the **relationship** between input features and the **target variable** changes.

Model Quality Drift:

- Reflects a **decrease in the model**'s predictive performance over time.

Bias Drift:

- Involves shifts in model outcomes that could lead to **unfair treatment** of certain groups.

Classical Monitoring vs. Modern Monitoring

Bridging Siloed Practices: Unified Approach to Data and ML Monitoring

Classical Monitoring

Same needs, different paradigms, different tools, different approaches

Data Engineers/Analysts

- Measure **data quality** flowing through pipelines and dashboards.

Data Scientists/ML Engineers

- Measure **ML model quality** and **Data quality affects ML model quality**

Modern Monitoring

Shared monitoring goals, tools, and approach

- Align monitoring for **data quality** and **model performance** across teams.
- Use **shared dashboards** and **pipelines** for real-time collaboration.
- Automate **model retraining** based on detected drifts or performance drops.

What is Lakehouse Monitoring

Data
Automated profiling of any Delta table

Lakehouse Monitoring
Quality across your entire data + AI estate

AI
Model drift and performance

Unity Catalog

Metadata | Governance | Serverless

Why Lakehouse Monitoring?

Monitor all tables in your Lakehouse using different analysis metrics based on table type(s)

Snapshot Table

Bronze/Silver/Gold

monitor

- Columns

Time Series Table

Feature table

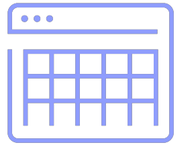
monitor

- **TimeStamp**
- Columns/Features

Model(s)

- Databricks batch scoring pipeline
- Databricks Model Serving Endpoint
- ETL to ingest from external serving (request logs) or batch pipelines

Inference Table



monitor

- **TimeStamp**
- Features
- **Prediction** column
- Label column
- **Model ID**

Table Monitoring : Capabilities

Monitor Key Metrics

For table fields such as Nulls, null %, zeros, zero %, avg, distincts, distinct %, max, min, stdev, median, max/min/avg length, value frequencies, quantiles, row counts

Group Metrics Over Time

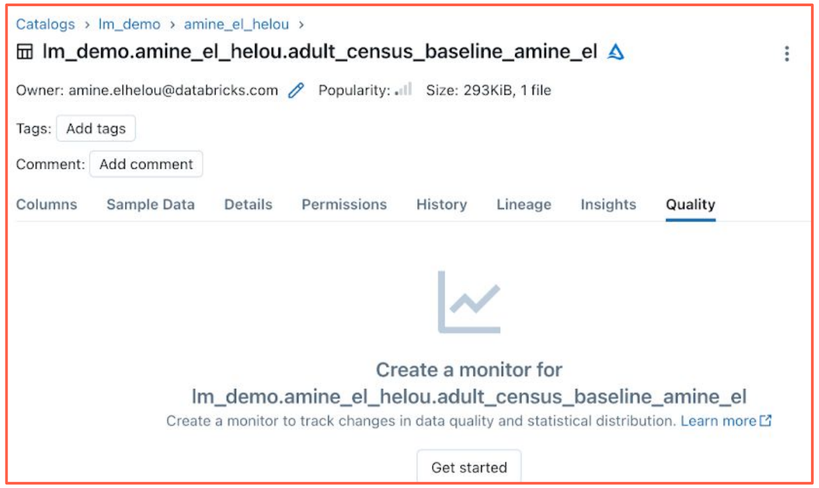
Monitor metrics over time windows, e.g. every day, every 5 minutes, over *n* weeks

Monitor Data Slices

Slice metrics based on columns or predicates, e.g. `state`, `product_class`, `"cart_total > 1000"`

Monitor Tables and ML Models

Consistent quality & drift monitoring of all your production assets



UI (UC Data Explorer)

```python
import databricks.lakehouse_monitoring as lm

# Set up monitoring parameters
lm.create_monitor(
  table_name="my_UC_table",
  …)
# Refresh monitoring metrics
lm.run_refresh("my_table")
```

Python API

Table Monitoring : Capabilities

Custom or Auto-thresholds

Set thresholds or use auto-generated alert thresholds based on baseline data

Dashboards and Alerts

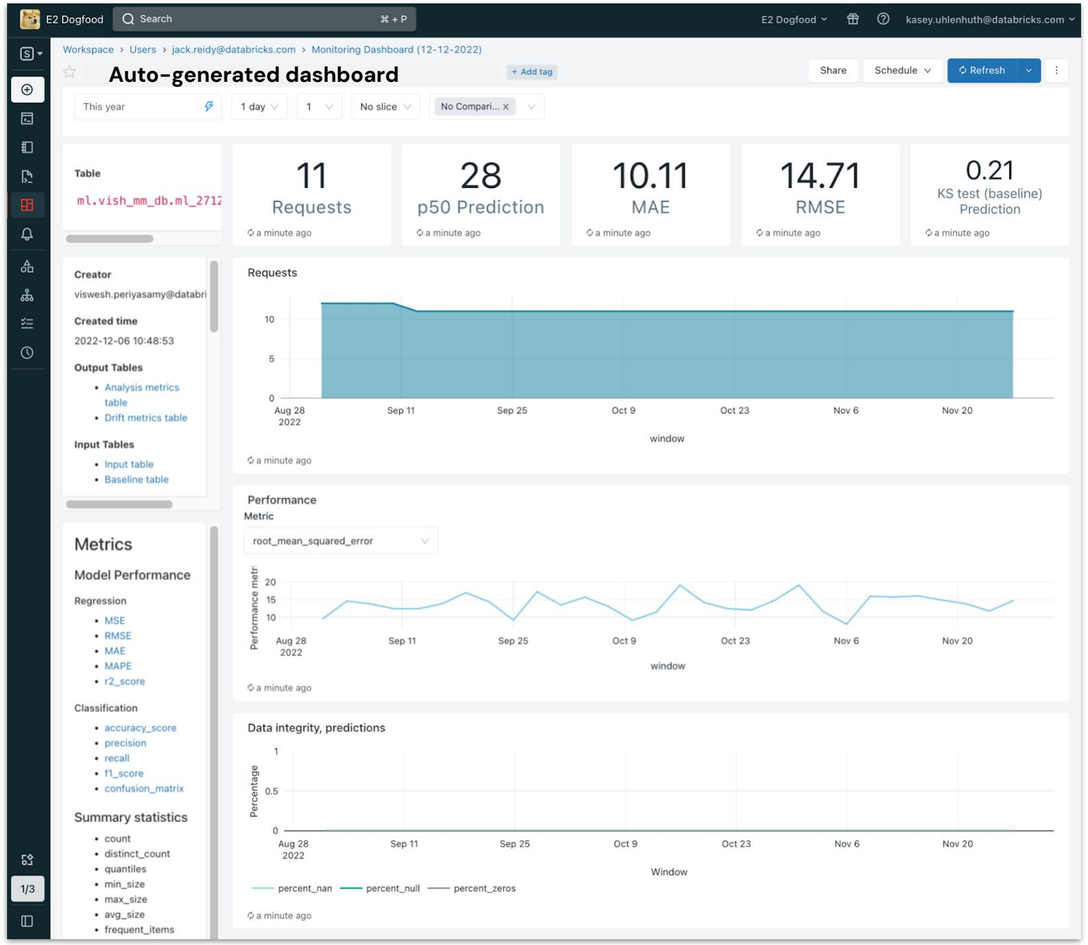
Auto-generated DB SQL dashboards to visualize metrics & trends, SQL Alerts for notifications

Open Monitoring Results

Monitoring results stored in open Delta tables to build custom analytics using your favorite BI tool

Simple Operations

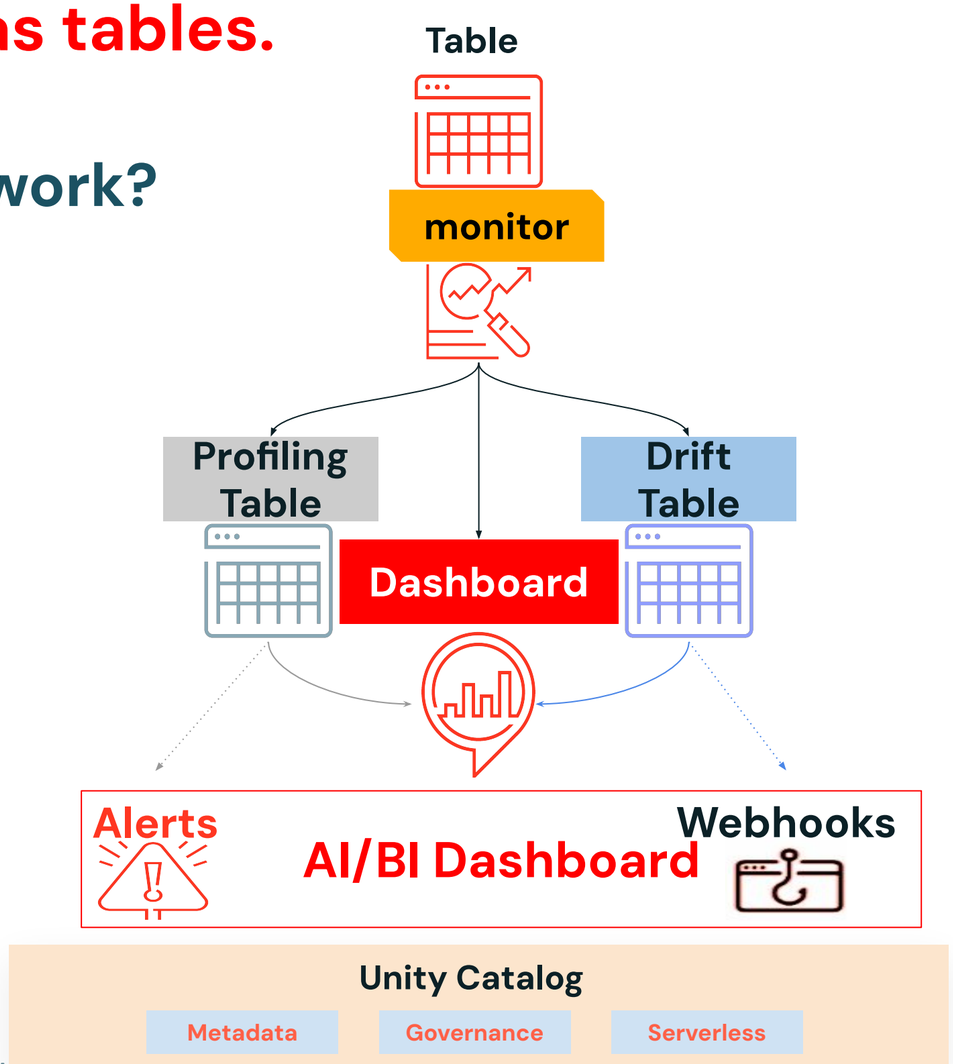
Databricks Serverless eliminates infrastructure management and scaling complexity



Monitoring a table in the Lakehouse

Whether data comes from data ingestion or model outputs, data manifests as tables.

How does it work?



Background(scheduled) service that incrementally processes data in Unity Catalog tables:

- Calculates **profile metrics** stored in Unity Catalog table
- Calculates **drift metrics** stored in Unity Catalog table
- Supports **custom metrics** as SQL expressions
- **Auto-generates dashboard** to visualize metrics over time

Out-of-the-box Standard Metrics & Custom Metric

These can be exported to any BI tool (we connect to DBSQL automatically)

- **Out-of-the-Box Metrics:** Get a standard set of metrics by default

Distributional statistics for inputs, output, ground-truth

Minimum, maximum, standard deviation, quantiles, top occurring value, ...

Anomaly detection and drift for training-vs-scoring and scoring-vs-scoring

Delta/changes in nulls and counts, PSI, KS divergence, Mean shift, Total Variation distance, L-inf distance, χ^2 test, Wasserstein distance, ...

Model quality metrics (if labels are provided)

Classification: Accuracy, F1, precision, recall Regression: MSE, RMSE, MAE, R^2, ...

- **Custom Metric Support**: Express custom business metrics as SQL expressions when you configure your monitor

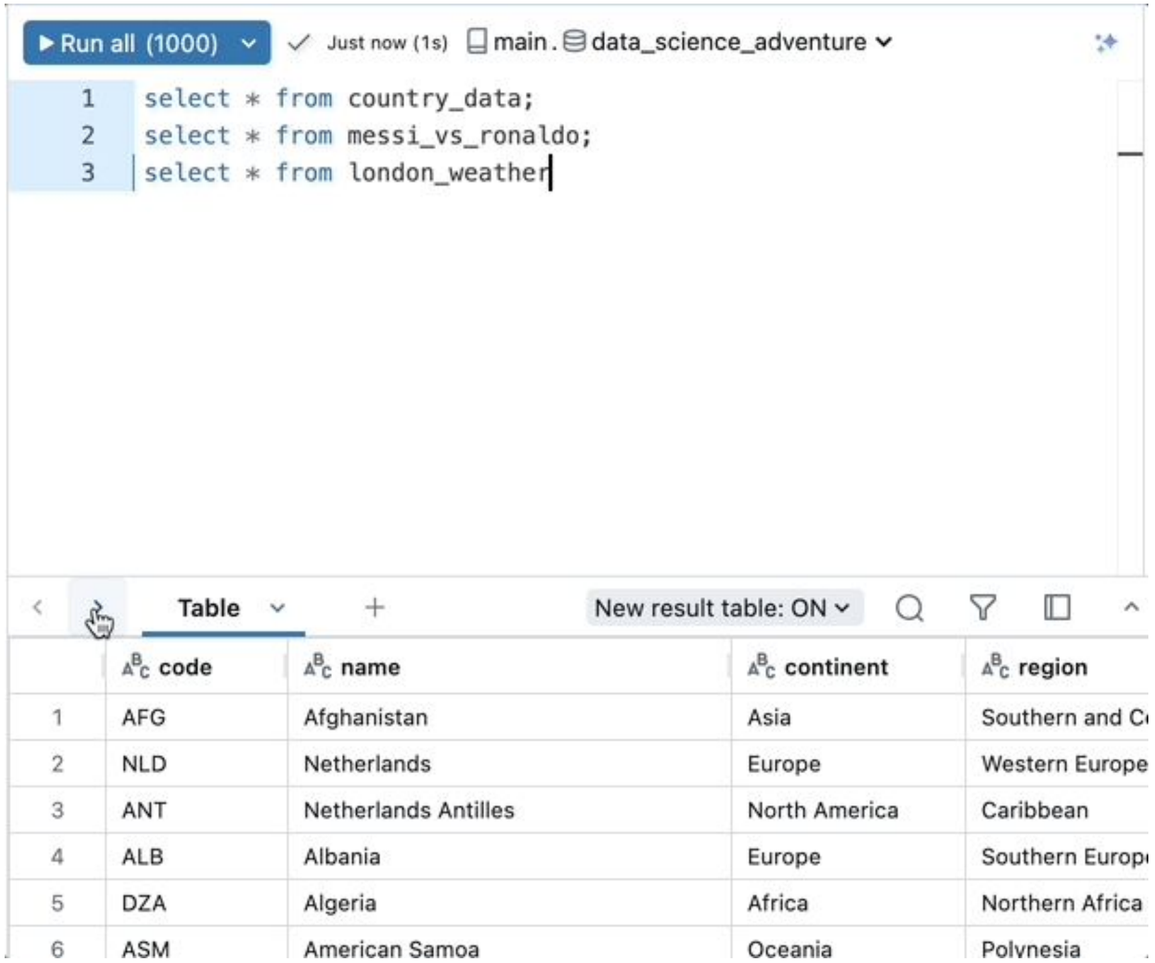
Custom metrics

Expressed as SQL expressions

Custom Metrics using Databricks SQL Queries

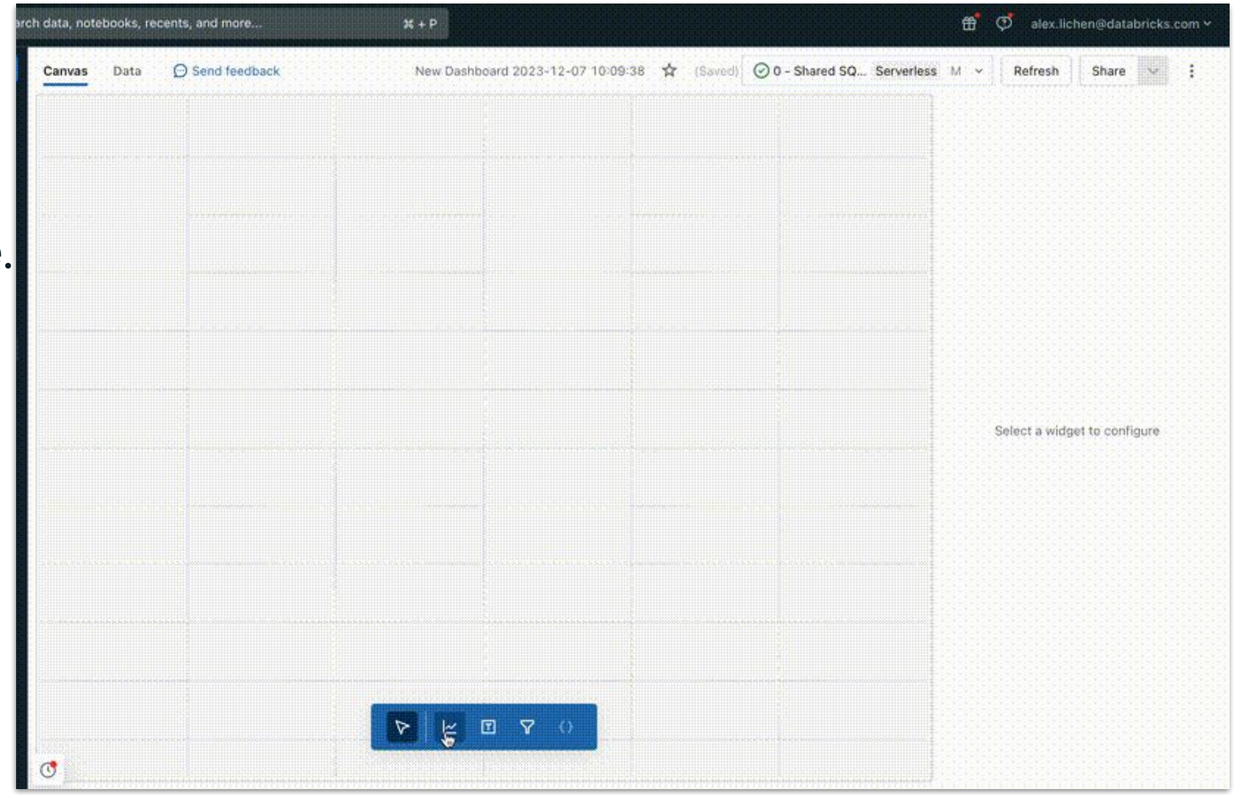
Enhancing collaboration and efficiency with AI powered SQL Editor

- **Use Databricks SQL Query to Create Metrics and Tables** and add the tables or visualizations to a dashboard.
- **Databricks Assistant** is an AI-based assistant that enhances **user productivity** by generating SQL queries, explaining complex code, and **fixing errors**.
- Describe your task in **English**, and Databricks Assistant will **automatically handle SQL generation**, code explanation, and error correction



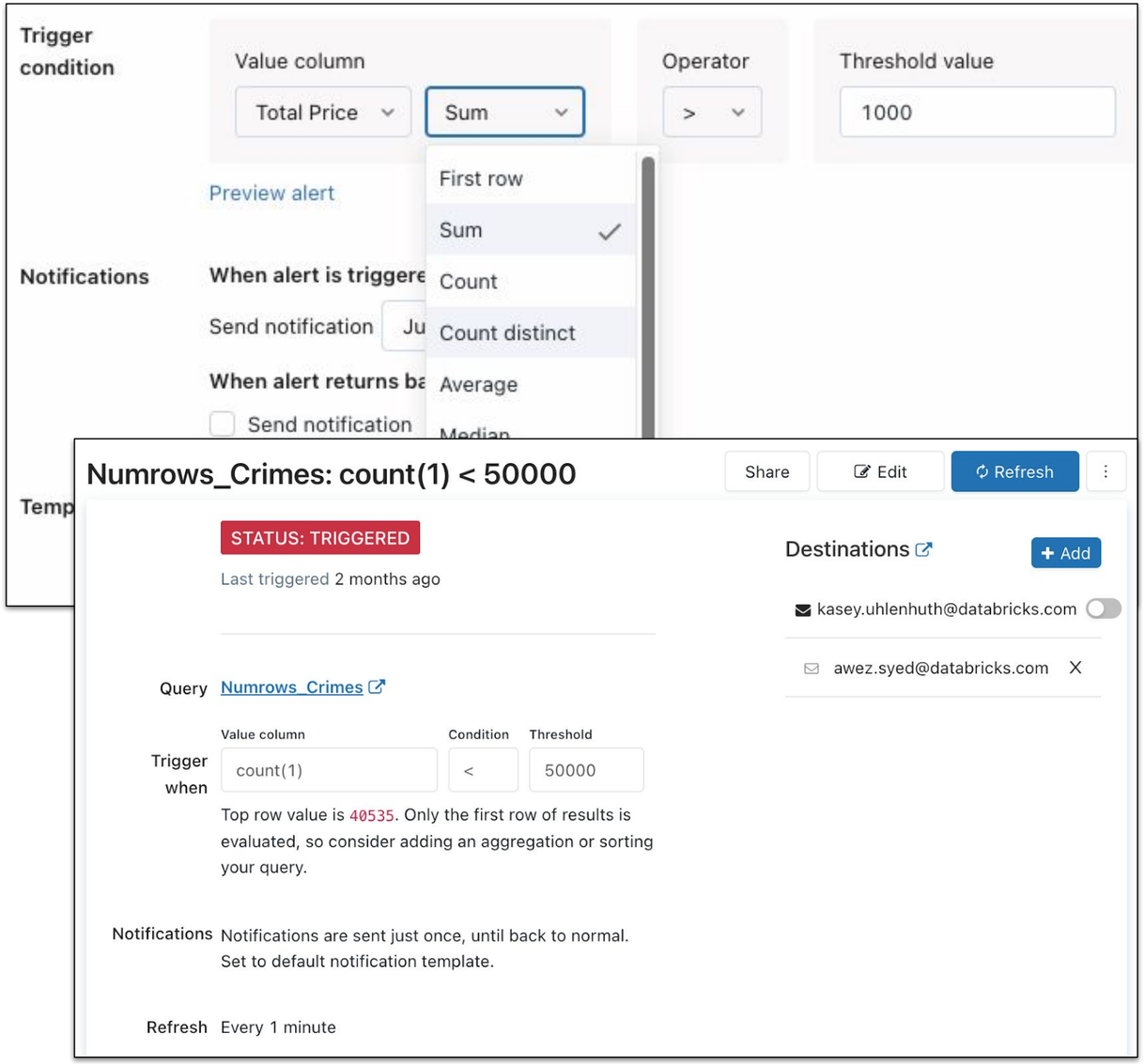
Visualize performance in auto-generated AI/BI Dashboards

- **Frictionless: dashboard automatically created** for you.
- **Visualize Performance**: Gain insights into data integrity, performance, drift, and more.
- **Share with stakeholders:** Can be shared with other users or groups.
- **Customizable**: dashboard can be edited and **visuals of custom metrics added** to best suit your needs
- **AI Assistant**: quickly generate custom widgets with Databricks AI Assistant



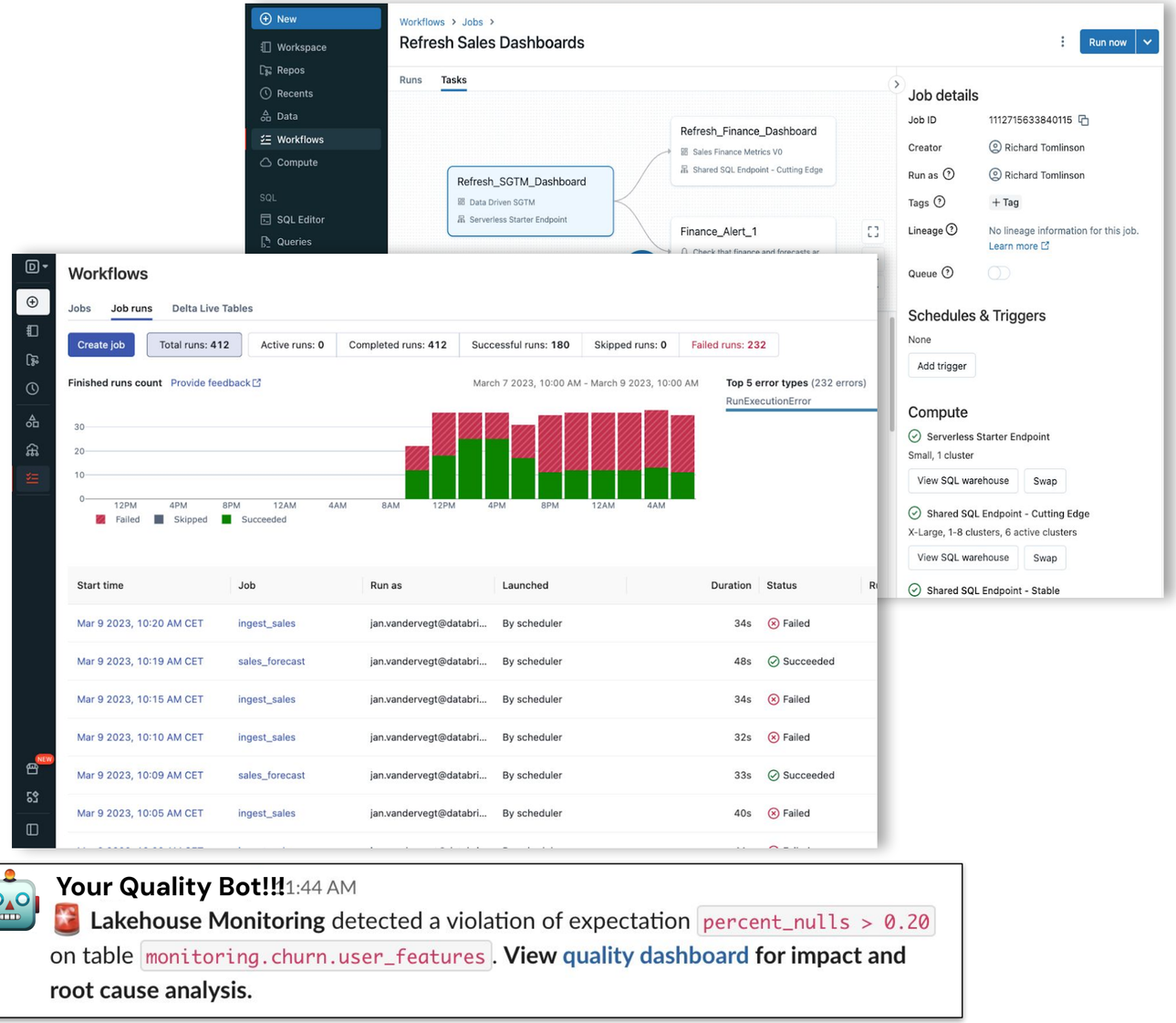
What are Databricks SQL alerts?

- Databricks SQL alerts **periodically run** queries, **evaluate defined conditions**, and **send notifications if** a condition is met.
  - **Scheduled Execution:** Automatically runs queries at defined intervals to check specific conditions.
  - **Multi-Channel Notifications:** Receive alerts via Email, Slack, Webhook, MS Teams, PagerDuty, and more.
- **Explore** the documentation for in-depth setup and customization options.



Orchestrate your SQL queries, dashboards, and alerts with Workflows

- Create, schedule, operate and monitor workflows that include Databricks SQL objects such as queries, dashboards and alerts
- Automate and schedule your Databricks SQL production workloads with advanced workflow orchestration, reliable monitoring and observability
- Set up a Databricks Job and include your SQL tasks such as refreshing dashboards or triggering alerts

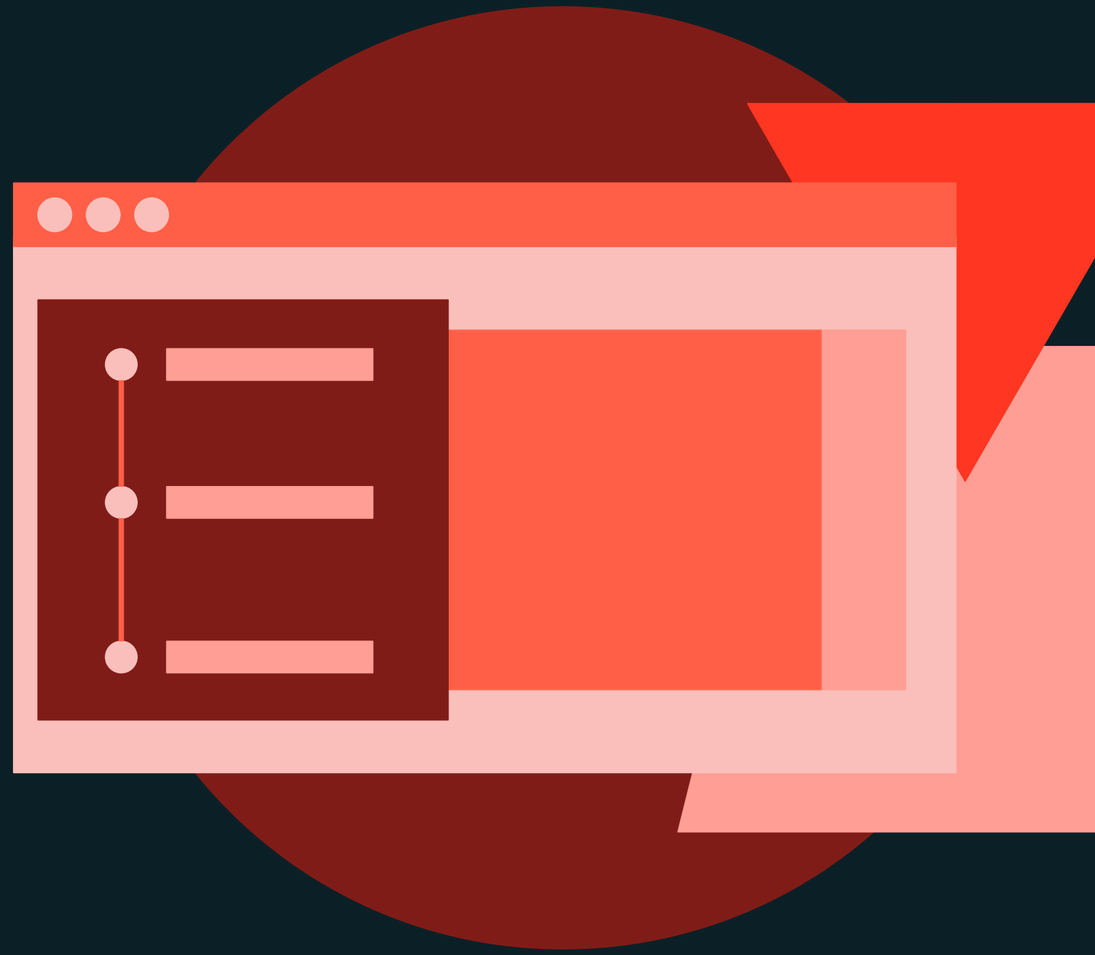


databricks

Model Quality and Lakehouse Monitoring

DEMONSTRATION

Monitoring Model Quality



databricks

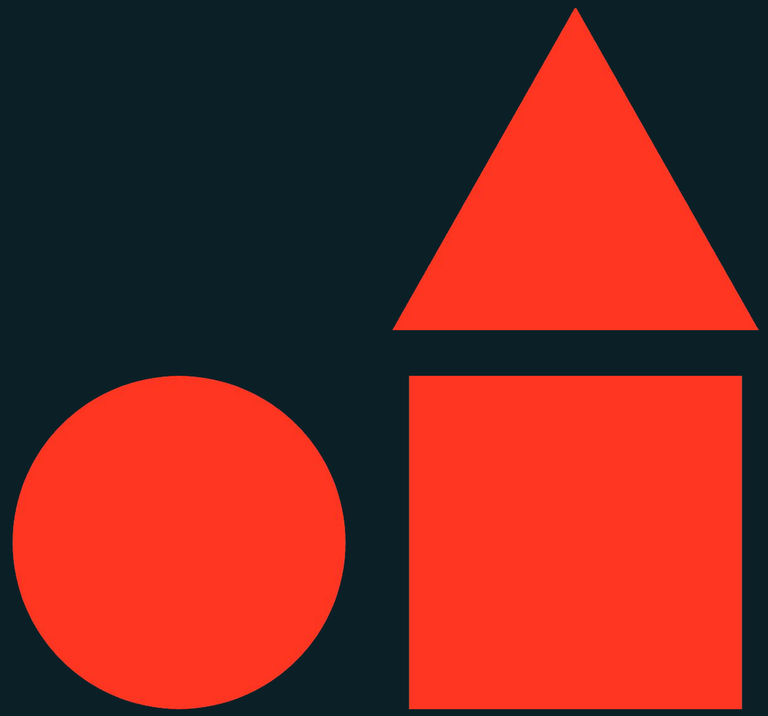
Model Quality and Lakehouse Monitoring

LAB EXERCISE

Monitoring Drift with Lakehouse Monitoring

databricks

Streamlining Multiple Environment Deployments - DABs



Advanced Machine Learning Operations

databricks

Streamlining Multiple Environment Deployments - DABs

LECTURE

Build ML Assets as Code

Learning objectives

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

- Understand why infrastructure as code is important
- Explain complexities of anatomy of ML projects
- Describe what problems DABs attempts to solve
- Understand the difference between traditional MLOps stacks and Databricks MLOps stacks

Infrastructure as Code

- The way to describe infrastructure objects using general purpose or specialized language
- **Pros:**
  - **Easy to create new** environments & maintain them
  - **Versioned**, easy to rollback in case of problems
  - **Changes** are made via pull requests, that are **tested** by build pipelines, **published** by release pipelines to multiple environments (staging/production)
  - Could help to build **completely automated releases**, without “manual” access to production environments
  - Usually it’s easy to i**ntegrate with CI/CD** pipelines
- Popular implementations: **Databricks Asset Bundle**, HashiCorp Terraform, Ansible, AWS CloudFormation, Azure Resource Management templates, …

Anatomy of your ML projects in Databricks

Let’s describe them

Consist of a variety of components

Code: Notebooks, Python .whl, JAR, dbt, etc.

Execution Environment: Databricks Workspace, compute configuration

Resources: Databricks Workflows, MLflow Tracking, Model Serving and Registry, Dashboards and Alerts…

Produce a variety of data products

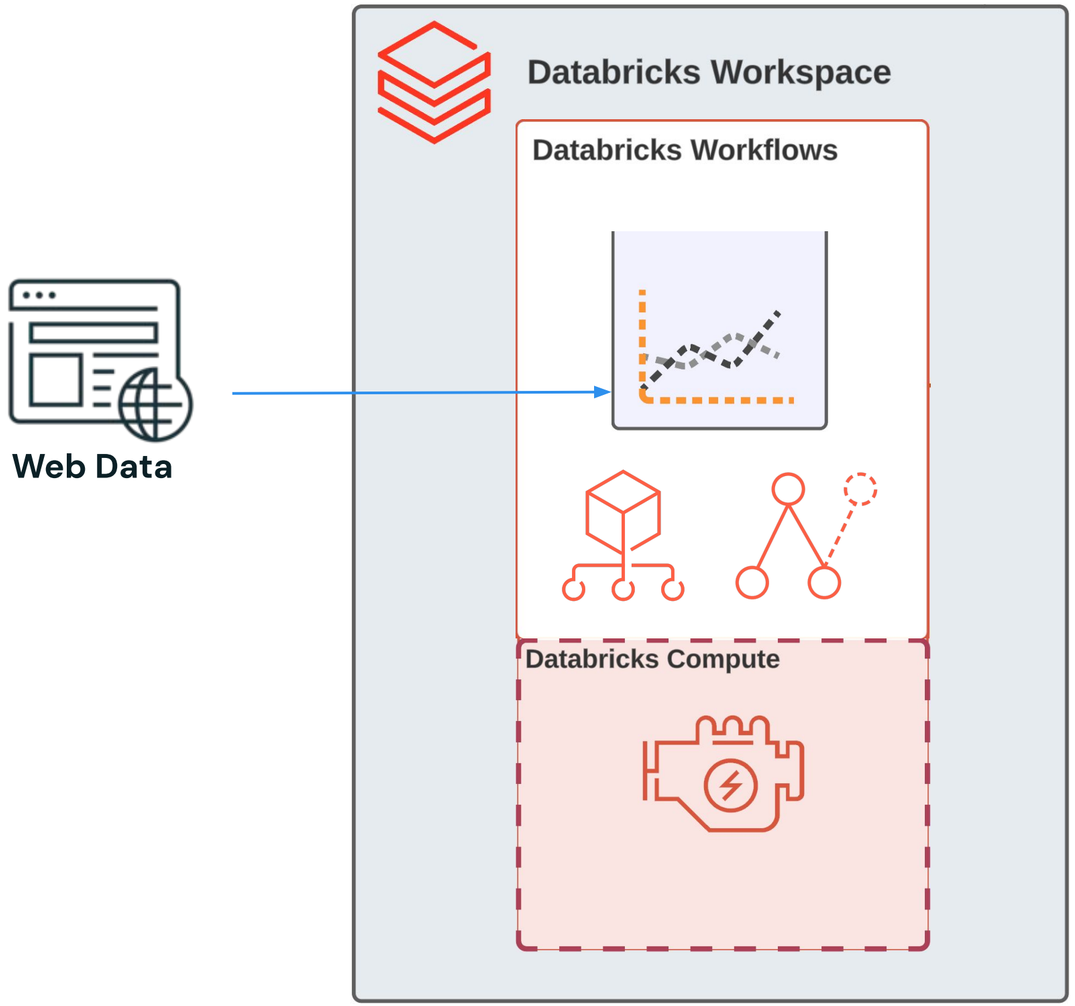
Create tables and pipelines, machine learning models, dashboards, call external services, etc.

The task determines the components

A simple report might consist of a notebook running on single node compute

A full MLOps pipeline would require MLflow, Feature Store, and Model Serving components

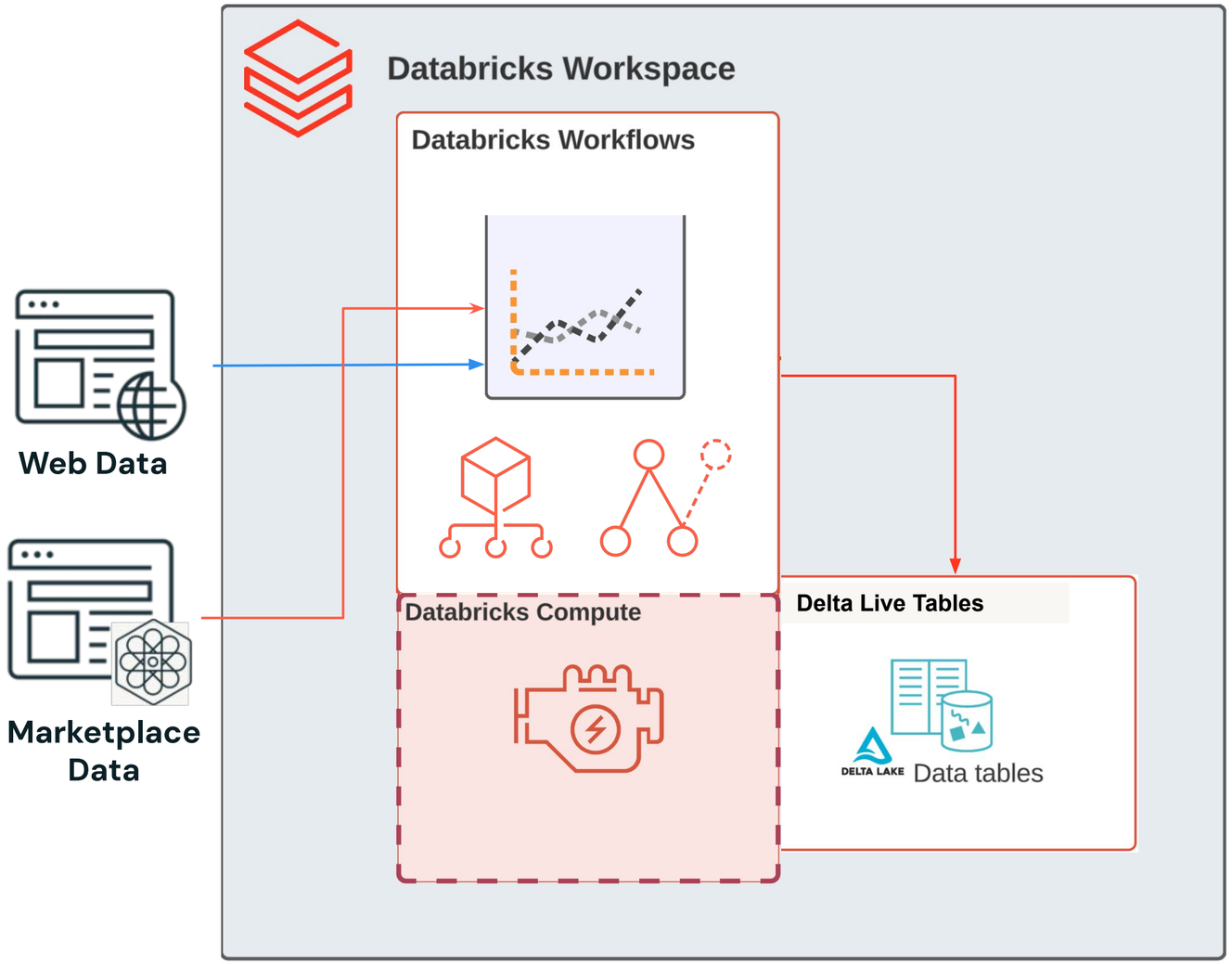
Let’s break this down further…



- **ML Product:** Predictive model for market trend forecasting
- **Code:** Python Notebooks leveraging ML libraries (e.g., sparkML, scikit-learn, PyTorch)
- **Resources:** Databricks Workflow, MLflow for experiment tracking
- **Execution Environment:** Databricks Workspace with ML runtime, compute configuration optimized for training
- **Data Source:** Historical web data and market reports for model training

What if our task changes?

Just add the necessary components to complete the task



- **ML Product:** Predictive model for market trend forecasting *+persisted table*
- **Code:** Python Notebook
- **Resources:** Databricks Workflow, MLflow for experiment tracking *+Delta Live Tables*
- **Execution Environment:** Databricks Workspace with ML runtime, compute
- **Data Source:** Historical web data and market reports for model training *+Marketplace data*

Databricks Asset Bundles

Write code once, deploy everywhere

What are Databricks Asset Bundles?

YAML files that specify the artifacts, resources, and configurations of a Databricks project.

How do bundles work?

The **new databricks CLI** has functions to **validate, deploy and run** Databricks Asset Bundles using bundle.yml files

Where are bundles used?

Bundles are useful during **development and CI/CD** processes

See Sample Bundle Walkthroughs & Tip

A closer look

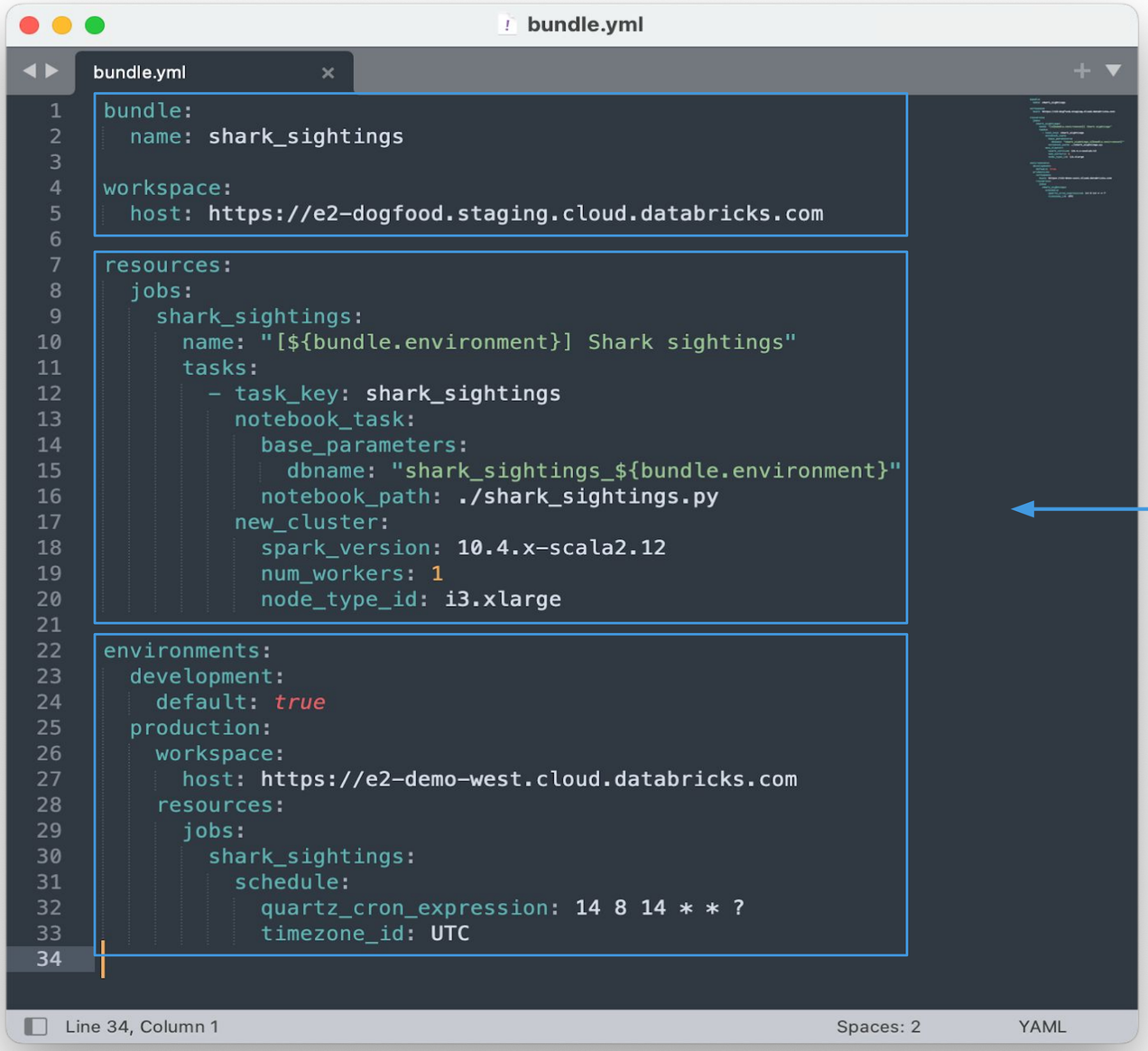
Name and default Workspace

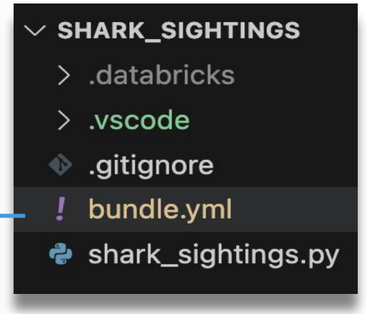
Resource configurations

- Jobs, DLT pipelines, MLflow, etc.
- Follows REST API schema

Environment-based specs

- Control project behavior in different environments





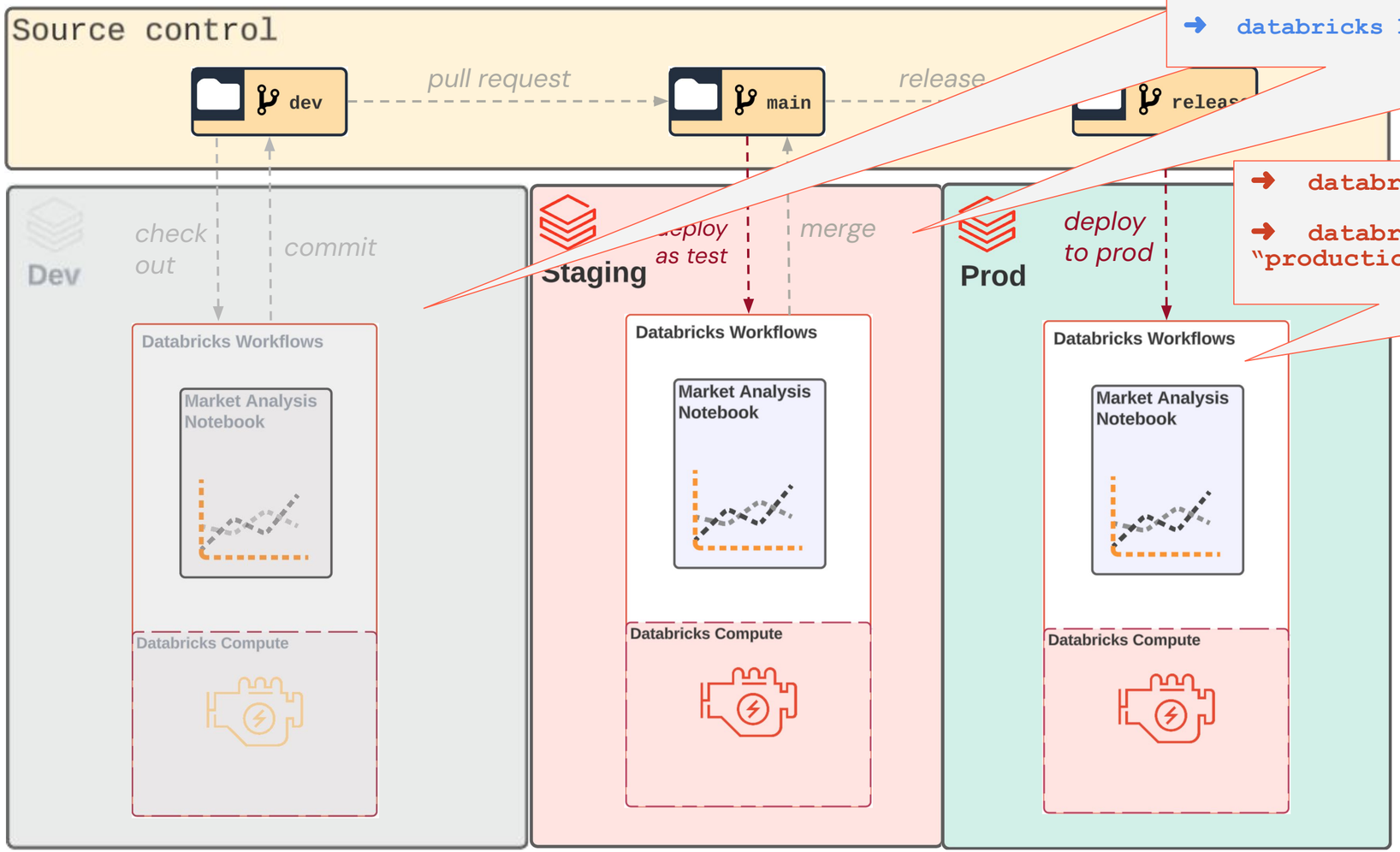
Where can we use bundles?

As part of Development and CI/CD processes

```
➜ databricks bundle deploy -e "dev"
➜ databricks bundle run pipeline -refresh-all -e "dev"
```

```
➜ databricks bundle deploy -e "staging"
➜ databricks bundle run pipeline -refresh-all -e "staging"
```

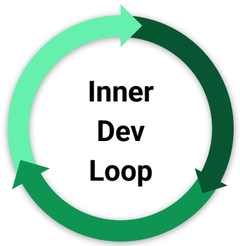
```
➜ databricks bundle deploy -e "production"
➜ databricks bundle run pipeline -refresh-all -e "production"
```

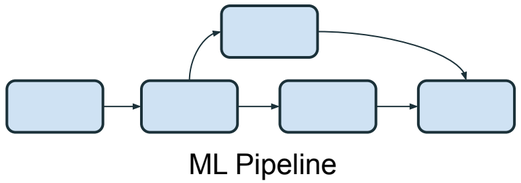


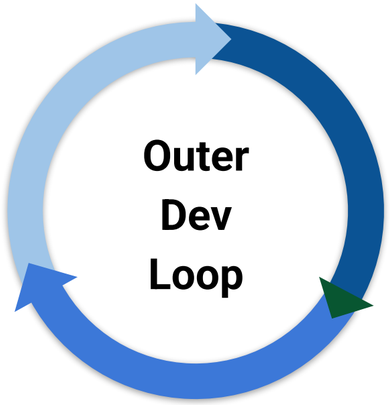
- Deploy and run your project, tweak configs, deploy and test changes
- Deploy to multiple workspaces for testing differences
- Specify environment-based overrides and variables
- Deploy and run from IDEs, terminals, or Databricks

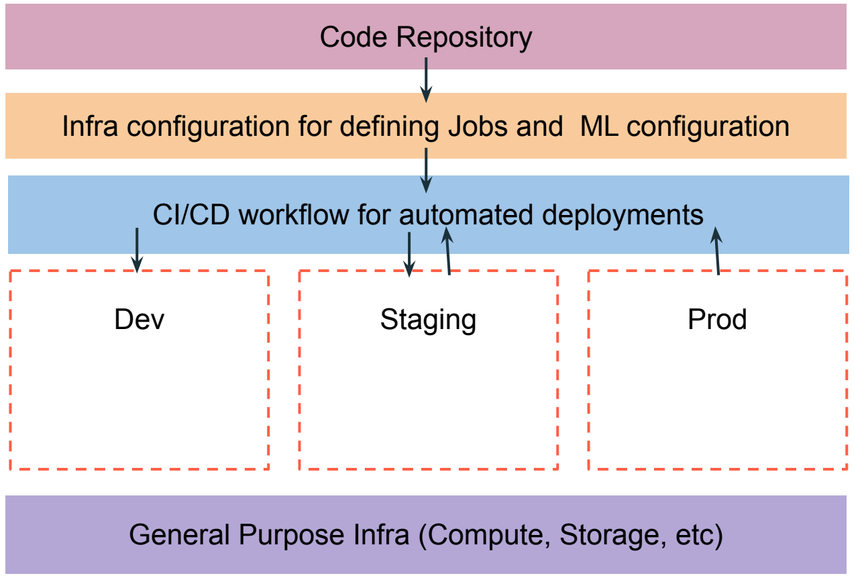
Implementing and managing infrastructure for MLOps is difficult!

Companies spend 80% of resources and time on DevOps activities in a ML project, leaving little time to add value through new models









Databricks
MLOps
Stacks

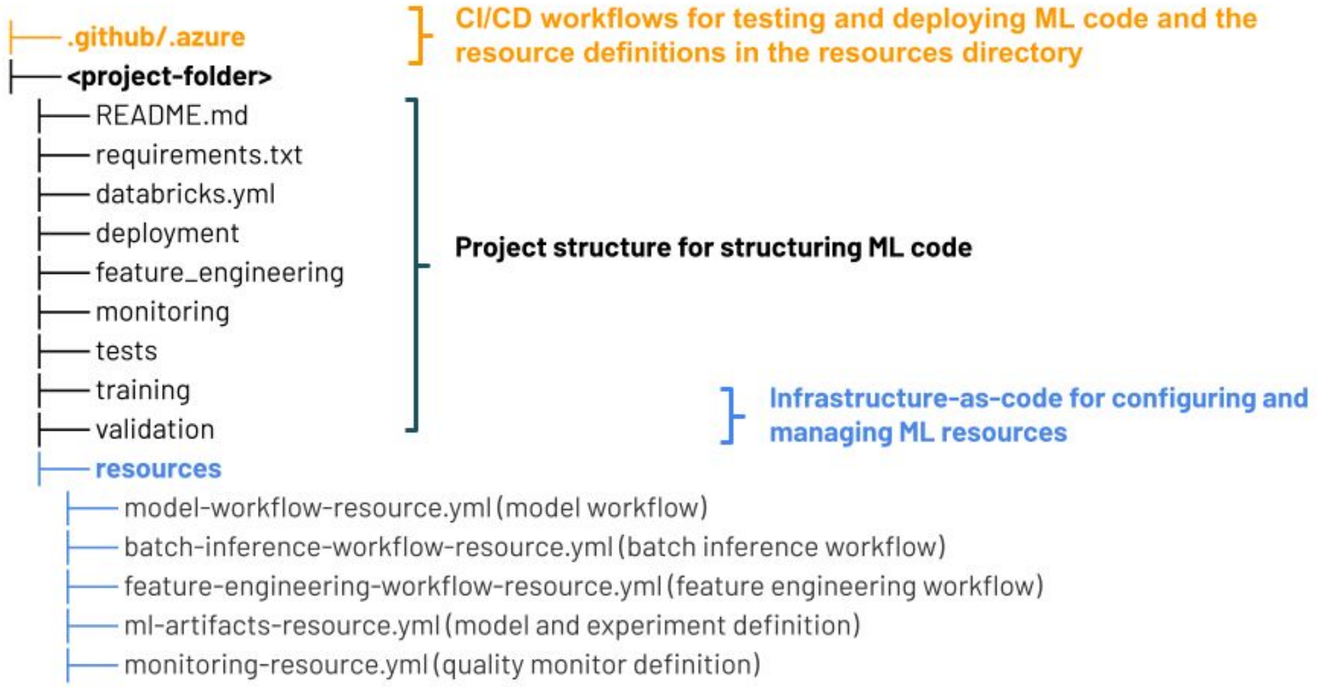
Default MLOps Stack

Provides out-of-the-box of the following tooling

Component	Tool in Databricks
ML model development code	Databricks notebooks, MLflow
Feature development and management	Feature engineering
ML model repository	Models in Unity Catalog
ML model serving	Mosaic AI Model Serving
Infrastructure-as-code	Databricks Asset Bundles
Orchestrator	Databricks Jobs
CI/CD	GitHub Actions, Azure DevOps
Data and model performance monitoring	Lakehouse monitoring

MLOps Stacks create a repo containing CI/CD, infra-as-code and sample project structure for productionizing ML

- Use `'databricks bundle init mlops-stack'` and **answer some question** to generate the core folder structure on workstation using CLI
- All environment and resource configurations are stored in `databricks.yml` as well as other YAML files inside `resources/` folder



Databricks MLOps Stacks

A customizable framework for managing the ML lifecycle on Databricks

Follows all **MLOps best practices**

- ➔ Create and manage **production MLOps infrastructure** on Databricks
- ➔ Integrates with common CI/CD providers like **GitHub** and **Azure DevOps**
- ➔ Manage AI assets (experiments, features, models, monitoring, etc.) and workflows as **IaC with Databricks Asset Bundles**

Focus on **ML**, not infrastructure

- ➔ DS get started with **project development option** in Stacks
- ➔ MLEs easily set up CI/CD and customize the architecture as needed via the **CI/CD option** in Stacks
- ➔ DS then safely deploy project to production through secure CI/CD pipelines and workflows setup by the MLEs

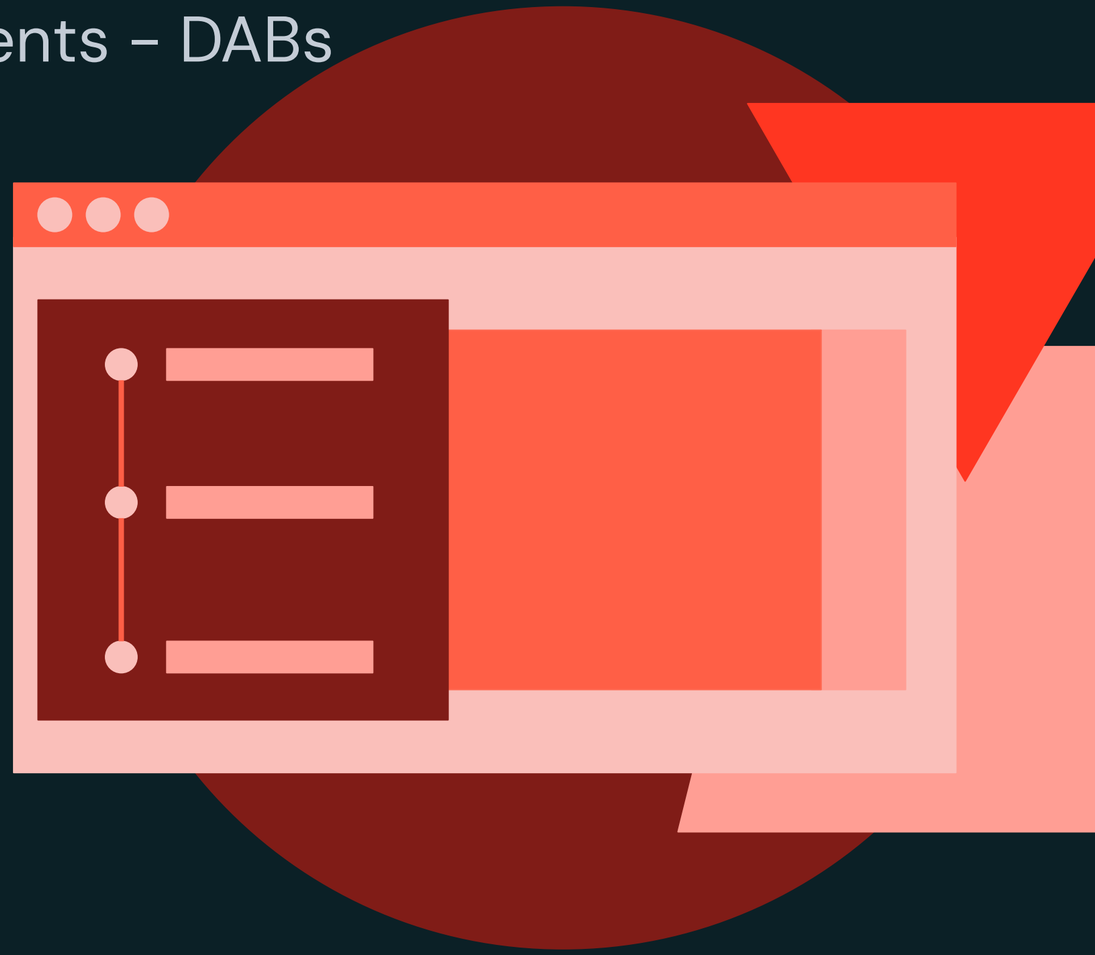
Refs: Big Book of MLOps | mlops-stacks repo

databricks

Streamlining Multiple Environment Deployments - DABs

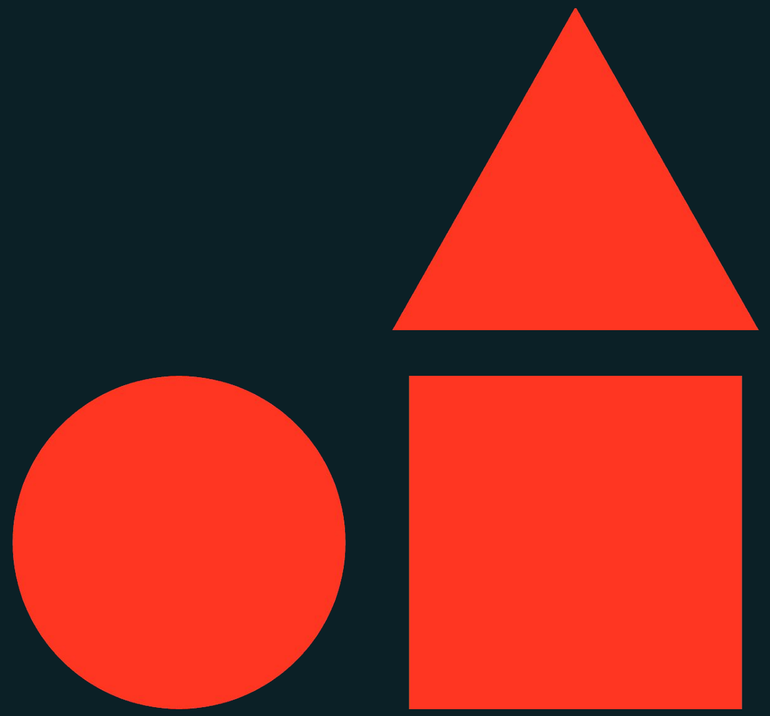
DEMONSTRATION

Working with Asset Bundles



databricks

Summary and Next Steps



Advanced Machine Learning Operations

databricks