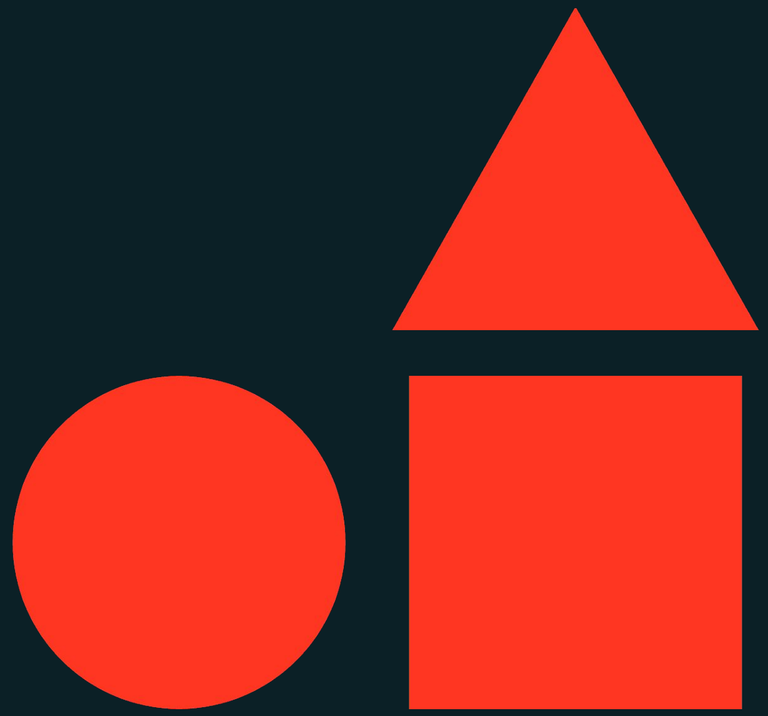
databricks

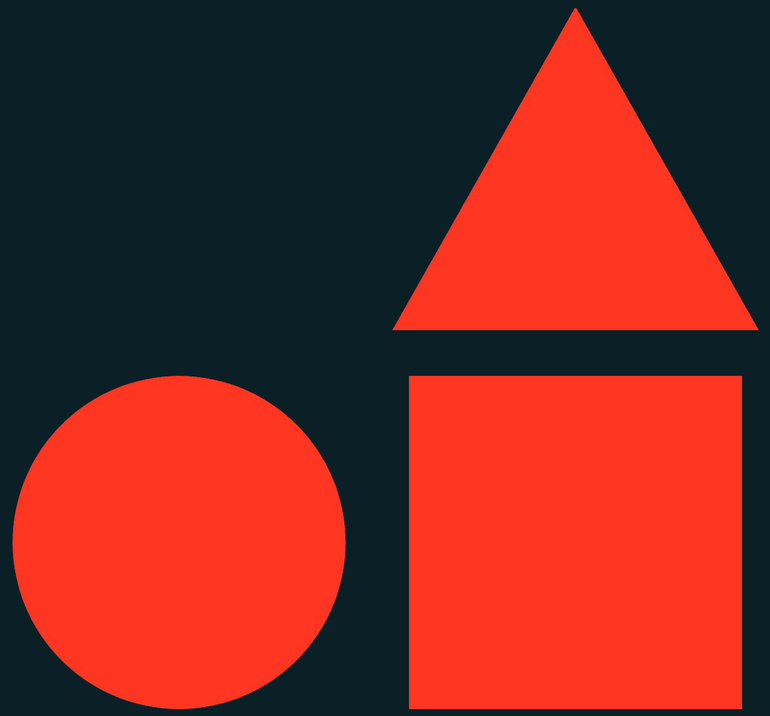
Generative AI Deployment and Monitoring



Databricks Academy

databricks

Model Deployment Fundamentals



Generative AI Deployment and Monitoring

Learning Objectives

- Describe batch, stream and real-time deployment.
- Identify scenarios in which each type of deployment is best suited.
- Compare and contrast batch vs. real-time deployments on Databricks.
- Describe MLflow's deployment capabilities (e.g. Model Flavors, Deploy Client)
- Explain the benefits of using Unity Catalog for registering models.

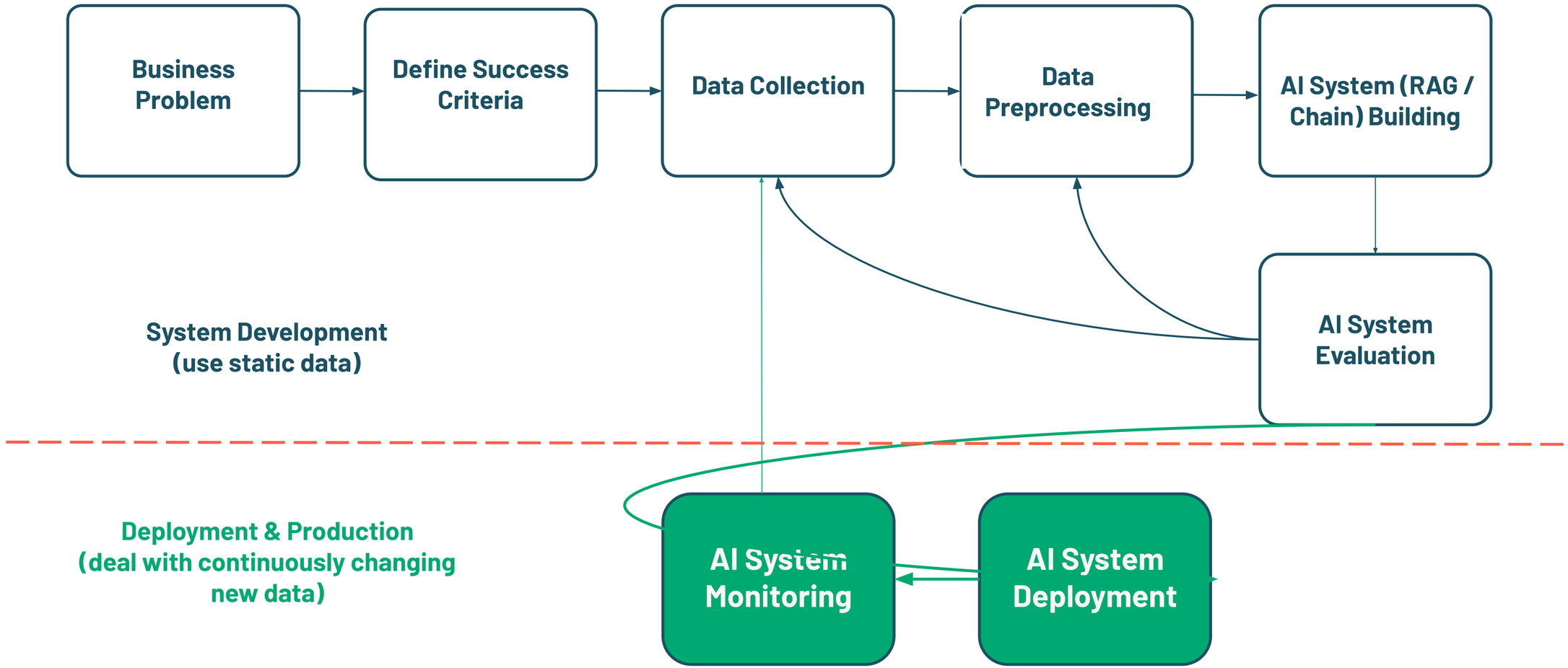
databricks

LECTURE

Model Management

The Gen AI System Lifecycle

Your AI System is ready, then what?



GenAI Deployment

Packaging models/pipelines

With LLMs, **ML logic is packaged in new forms** which could include:

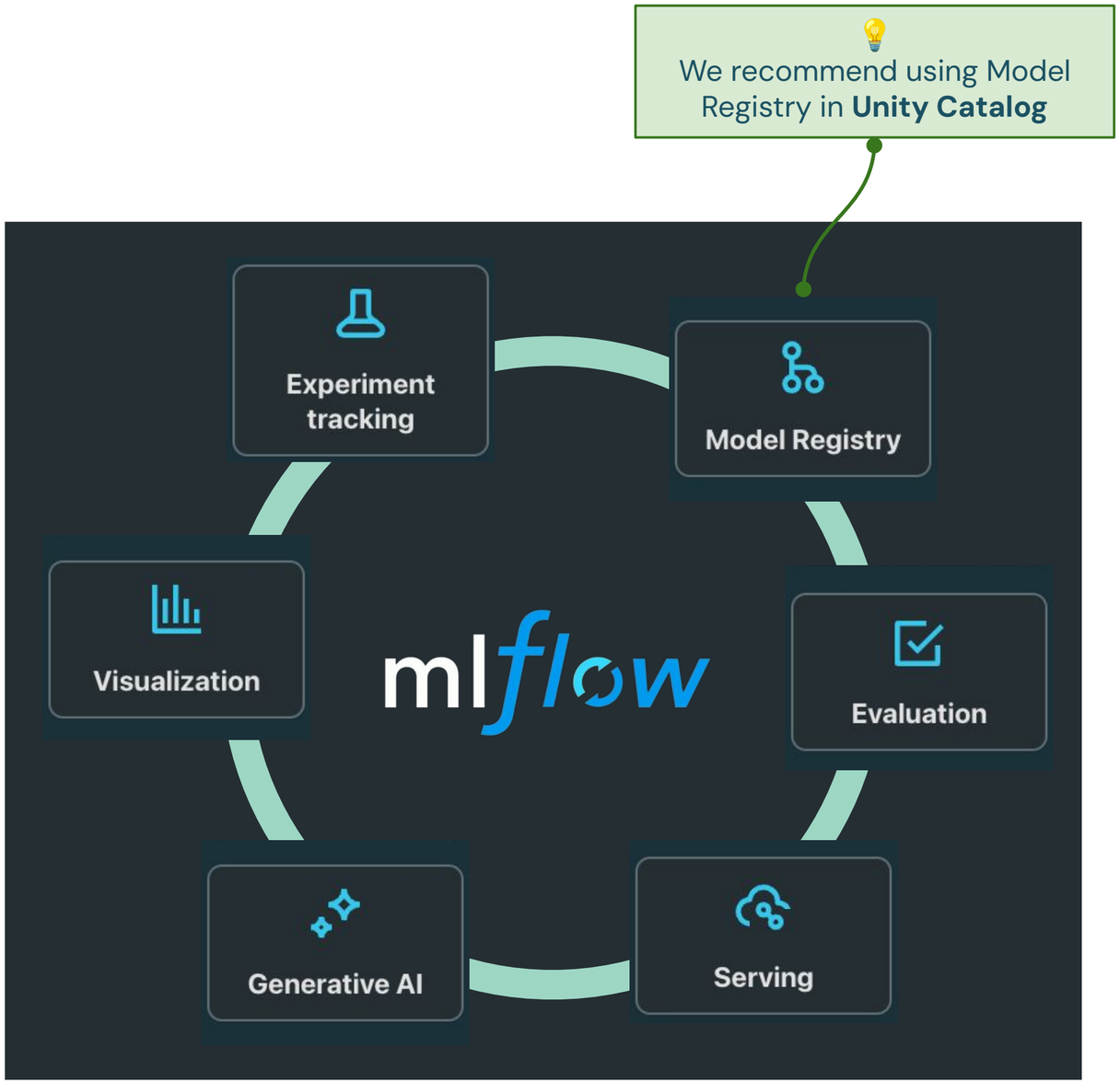
- An engineered prompt (possibly stored as a template)
- A "chain" from LangChain, LlamaIndex etc.
- A lightweight call to an LLM API service such as:
  - (Internally/Self) hosted foundation models (e.g. DBRX)
  - External proprietary models providers (e.g. OpenAI)
- A lightweight call to a bespoke (self) hosted LLM API:
  - Fine-tuned model
  - Pretrained model
- Locally invoking an LLM or an LLM+tokenizer pipeline (e.g. Hugging Face pipeline) running on GPU(s)

All of these still constitute models and pipelines!

MLflow Components

Model management with MLflow

- Manage end-to-end ML and GenAI workflows, from development to production.
- Unified platform for both traditional ML and GenAI applications.
- Generative AI specific model flavors and evaluation metrics.



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 LangChain Flavor

```
# Directory written by
mlflow.langchain.log_model(model, "chain",...)
chain/
├──model
    └── steps
    └── steps.yaml
├── MLmodel
├── lc_model.py
├── conda.yaml
├── python_env.yaml
├── requirements.txt
```

```
# MLModelfile
artifact_path: chain
flavors:
  langchain:
    code: null
    langchain_version: 0.1.5
    model_data: model
  python_function:
    loader_module: mlflow.langchain
...
```

MLflow – 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 Flavors

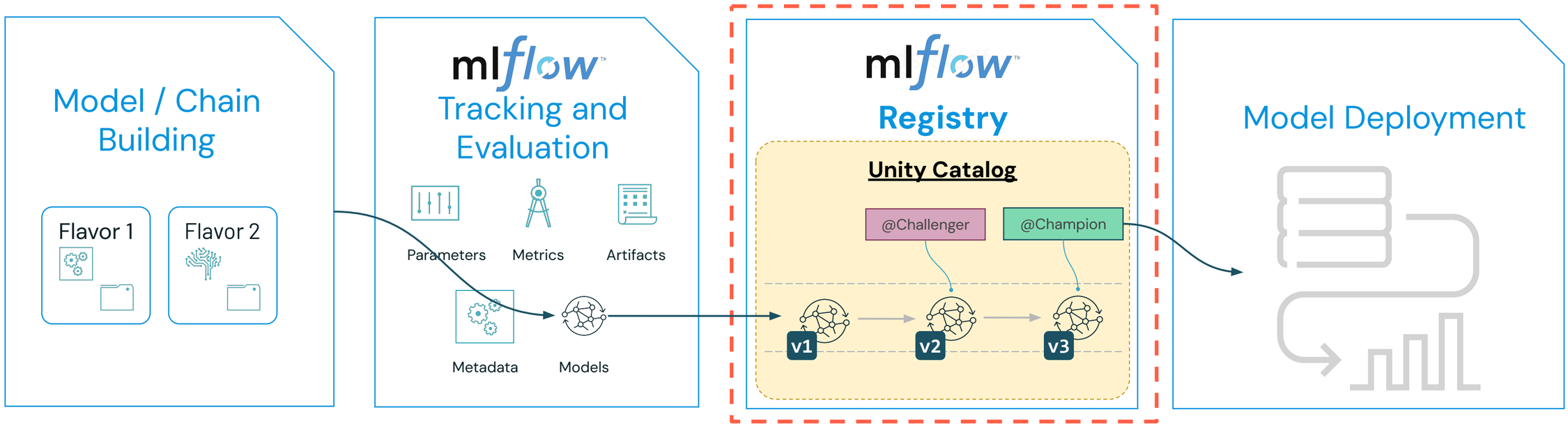
Example built-in model flavors:

- LangChain
- OpenAI
- HuggingFace Transformers
- PyTorch
- TensorFlow
- ONNX
- **Python function**

...

MLflow and Development Lifecycle

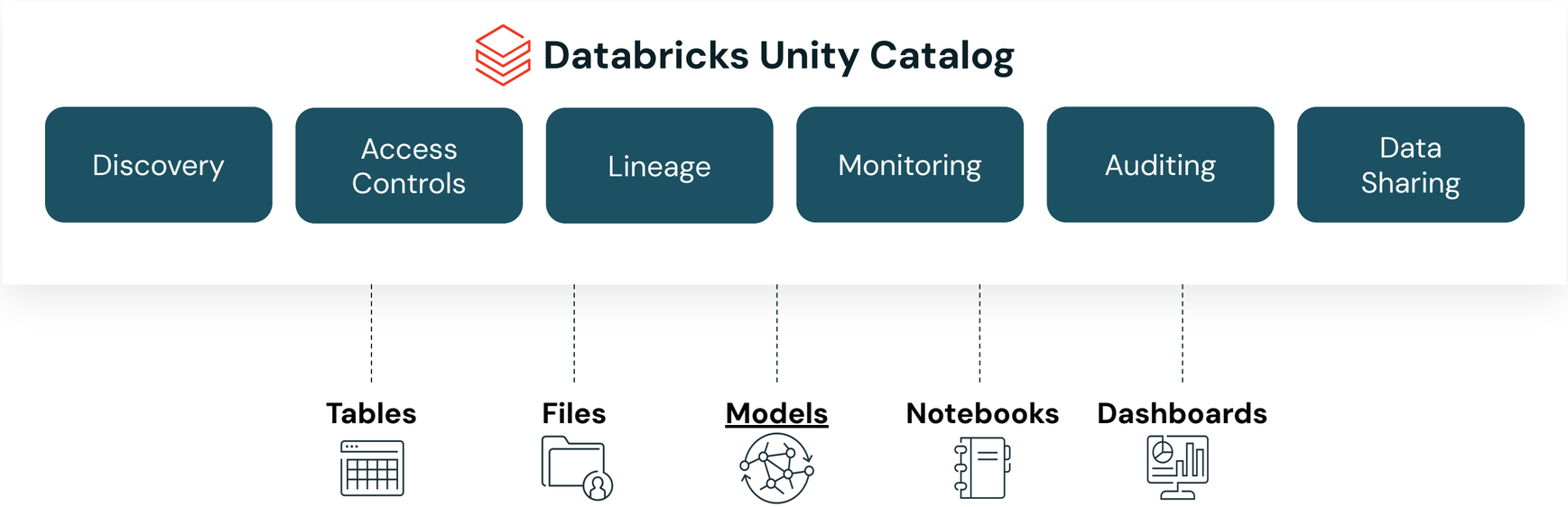
MLflow integration to seamlessly deploy models



Unity Catalog (UC)

Single governance solution for **data and AI assets** on the Lakehouse:

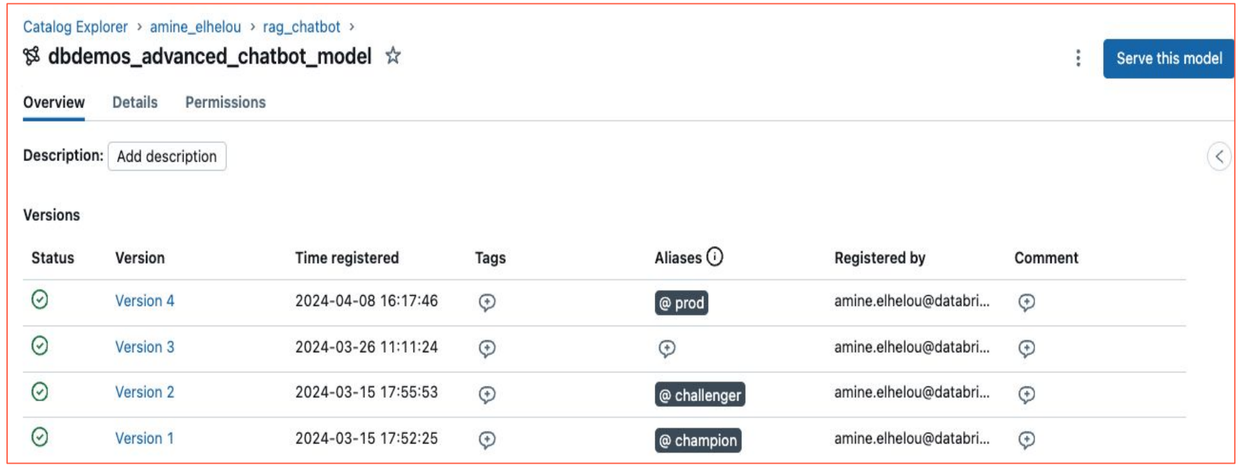
- Unified visibility into data and AI
- Single permission model for data and AI
- Open data sharing

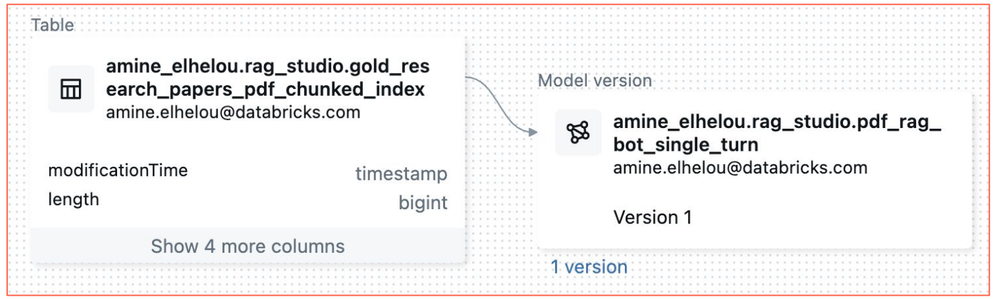


MLflow – Unity Catalog Model Registry

A centralized model store

- **Model** lifecycle management with **versioning** & @**aliases**. (e.g. @champion/challenger)
- Deploy and organize models
- Collaboration and ACLs
- Full model lineage
- Tagging and annotations





ML*flow* and GenAI Deployment

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
- Deploy to major cloud providers.
- **Deploy with Databricks Model Serving**

databricks

LECTURE

Deployment Methods

Gen AI Model Deployment

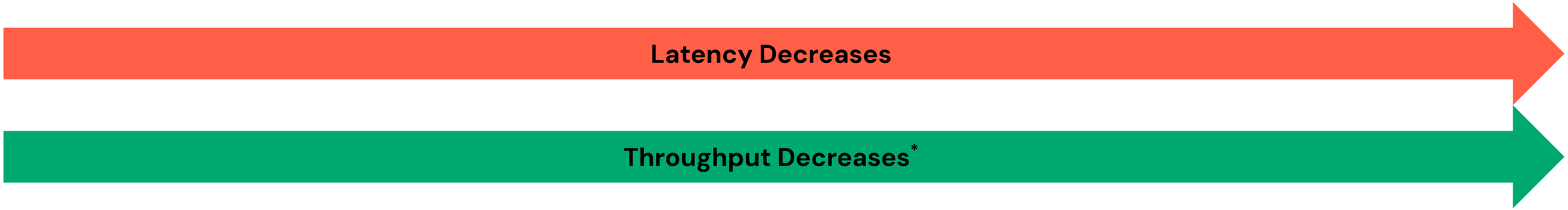
Deploying a GenAI/LLM model in a few words

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

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

Deployment Paradigms

Similar to traditional machine learning extended to GenAI



Batch	Streaming	Real-time
Generate (and store) completions on a table of (text) inputs/prompts	Generate (and store) completions on micro-batches of (text) inputs/prompts as they're processed	Generate completions asynchronously in real-time on individual input/prompt

Note: Edge (on-device) deployment is challenging with large language models due to space requirements.

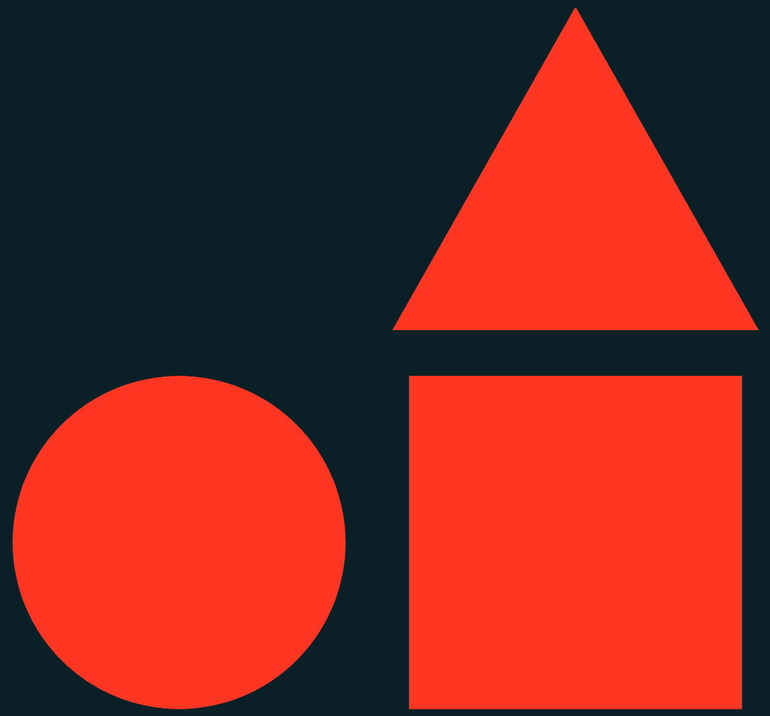
Deployment Paradigms

Comparing methods

Deployment Method	Throughput	Latency	Example Application
Batch	High	High (*hours to days*)	Summarizing financial reports and generating insights.
Streaming	Moderate	Moderate (*seconds to minutes*)	Personalizing marketing messages.
Real-time	Low-High	Low (*milliseconds*)	Chatbots (e.g. customer service, doc assistant)
Edge/Embedded	Low	Low (Dependent on device processing power)	Modify air conditioning temperature in a car using voice command

databricks

Batch Deployment



Generative AI Deployment and Monitoring

Learning Objectives

- Describe batch deployment and identify scenarios in which this method is appropriate.
- Identify the advantages and disadvantages of deploying a model via batch processing.
- Discuss a typical batch model deployment workflow on Databricks.
- Load a logged LM from the model registry and use it for batch inference.

databricks

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 simplest 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 **> 30 mins**

Batch Deployment

A typical batch deployment **use case**

- **Description**: Automated legal research
- **Scenario**:
  - Legal databases are continuously updated with new case laws, statutes, and legal literature.
  - AI system can be trained to automatically collect and preprocess this data.
  - AI system can extract information from analyzed data such as summarization or comparing old legal literature with the new one.

Batch Deployment

Advantages and limitations

Advantages

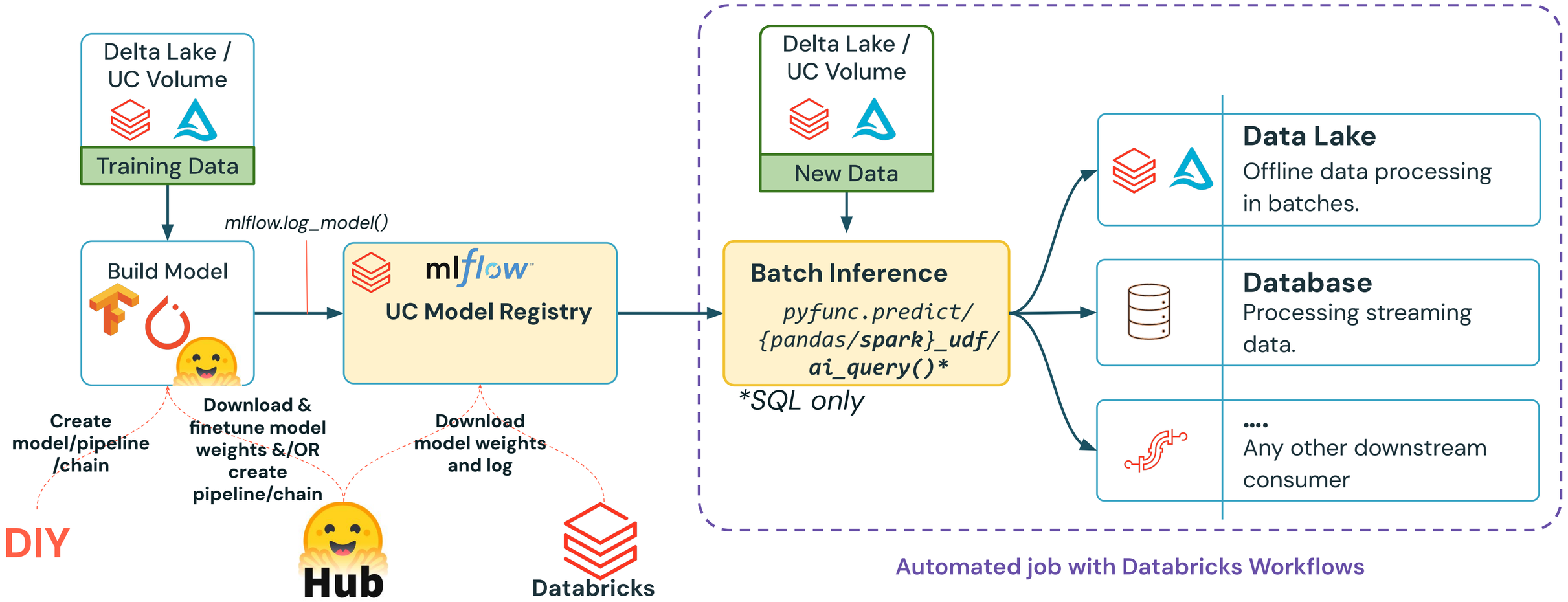
- **Cheapest** deployment method.
- **Ease** of implementation.
- Efficient per data point.
- Can handle **high volume of data**.

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 for (Small/Medium) LMs



Batch Inference from SQL using `ai_query()`

Batch invoke Foundation Models API with automatic parsing of completions

Query Foundations Models API from Databricks SQL

```
SELECT AI_QUERY (
"databricks-dbrx-instruct",
 CONCAT(
 "Based on the following customer review, answer to
ensure satisfaction. Review: ", review)
) as generated_answer FROM reviews;
```

Batch Deployment

Scaling batch inference workloads is NOT friction free

- Access to GPUs with large memory (GPU-RAM) for Larger Models
  - **~10B parameter** model at FP32 (32-bit floating precision) or 4-bytes requires ~10^9 (parameters) x 4 (bytes) **~ 40 Gigabytes of GPU RAM**

- Budget: cost of acquiring/provisioning HW while ensuring maximum utilization

- Parallelization is not trivial

Databricks Blog: LLM Inference Best Practices

Batch Deployment

Other batch inference methods using OSS integrations

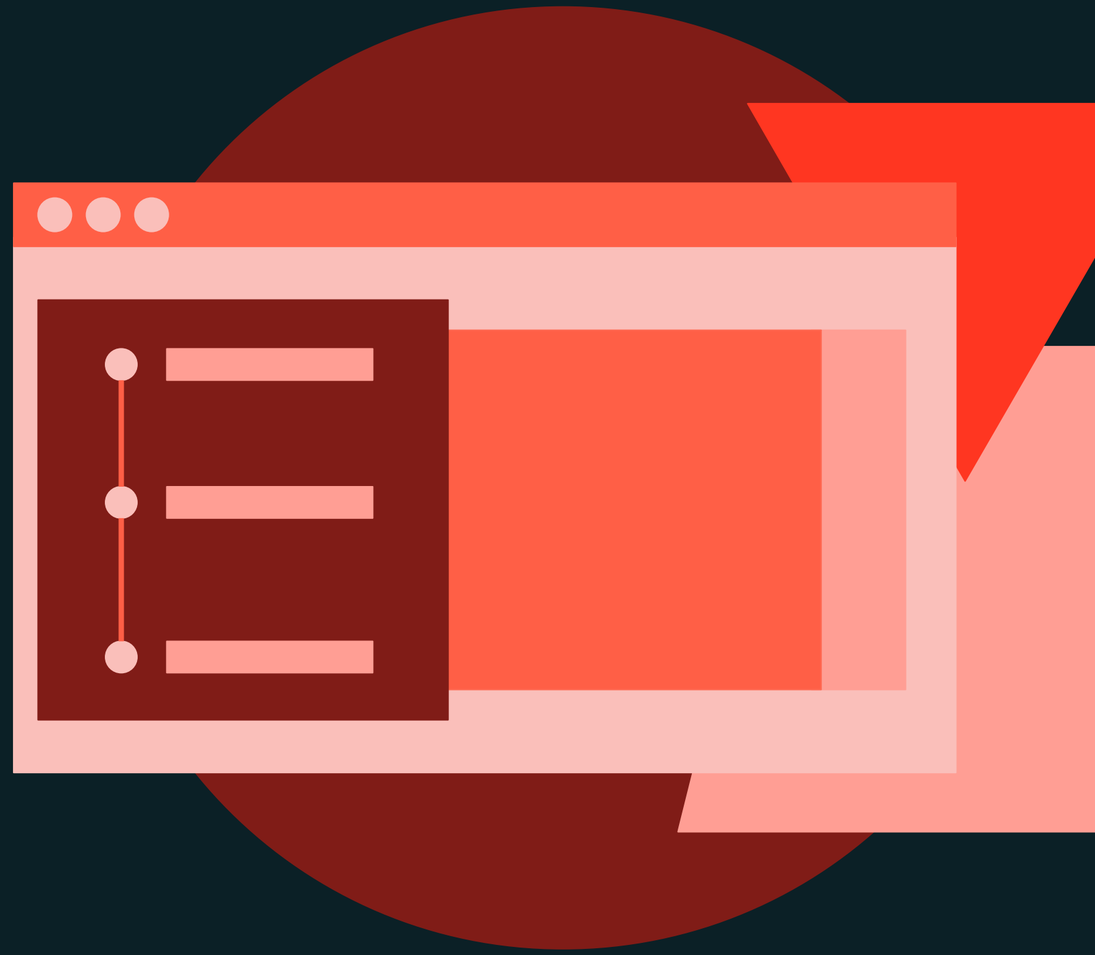
- **TensorRT™**
  - **Tensorflow**-friendly SDK (from **NVIDIA®**) for high-performance batch inference on GPUs
  - Databricks notebook example
- **vLLM**
  - **Transformer**-friendly library for memory-efficient inference on GPUs (NVIDIA® & AMD)
  - Example notebooks for DBRX, Mixtral 8x-7B and Mistral-7B

- **Ray on spark** (AWS | Azure | GCP)
  - Pythonic distributed computing primitives for parallelizing/scaling Python applications

databricks

DEMONSTRATION

Batch Inference Using SLM



Demo Outline

What we'll cover:

- Data and model preparation
- Model development and registration to Model Registry
- Manage model stages
- Create a production workflow for batch inference
  - Single-node batch inference
  - Multi-node batch inference
- Batch inference using ai_query()

databricks

LAB EXERCISE

Batch Inference Using SLM

Lab Outline

What you'll do:

- **Task 1**: Develop a LLM pipeline
- **Task 2**: Register the model to Model Registry
- **Task 3**: Manage model stages
- **Task 4**: Load and the model and using for batch inference
- **Task 5**: Use ai_query() for batch inference

databricks

Real-time Deployment

Generative AI Deployment and Monitoring

Learning Objectives

- Describe real-time deployment and identify scenarios in which this method is required.
- Discuss challenges of real-time deployment systems.
- Describe features of Databricks Model Serving.
- Discuss a typical real-time model deployment workflow on Databricks.
- Serve a model with model serving using the UI and the API.

databricks

LECTURE

Introduction to Real-time Deployment

Real-time Deployment

- The process of 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 chatbots, message intent 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.

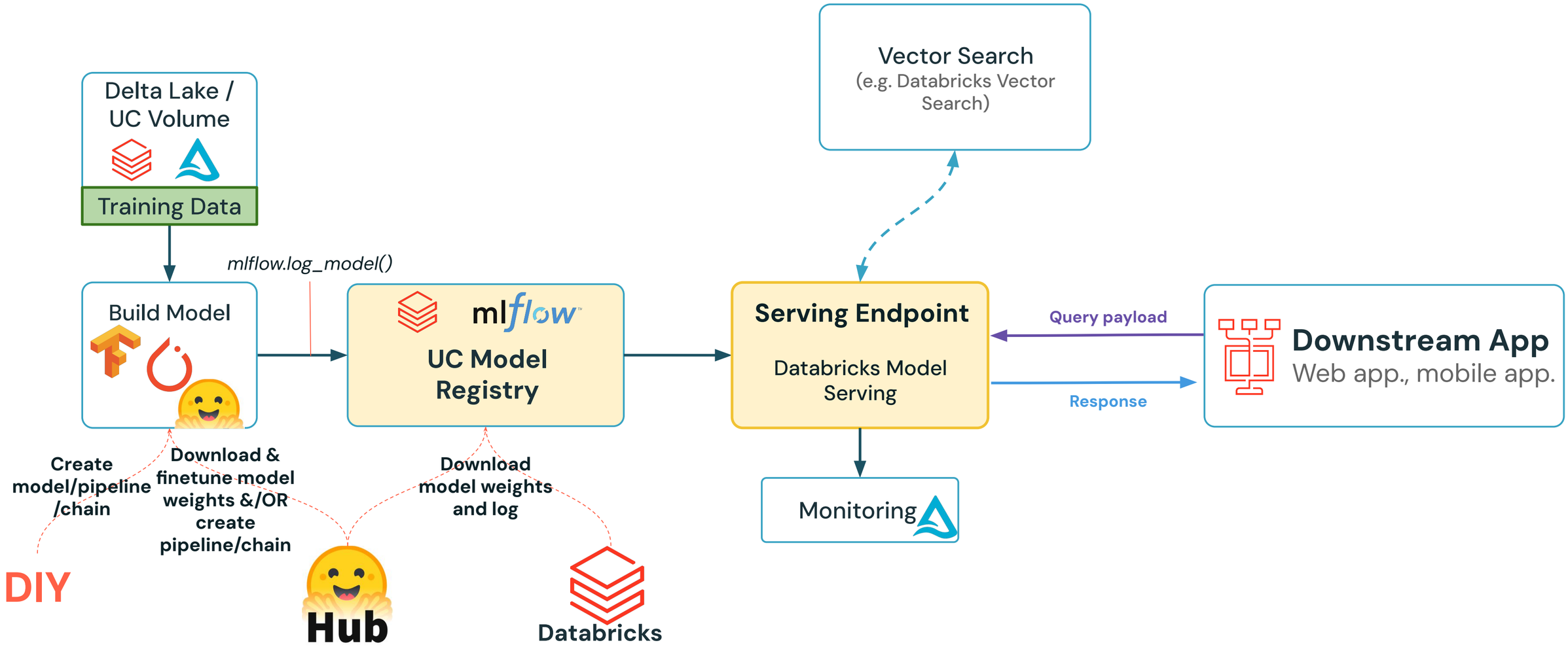
Real-time Deployment

A typical real-time deployment **use case**

- **Description**: Real-time Intent Detection for social media platform
- **Scenario**:
  - A fine-tuned large language model for classifying social media post, is deployed in the real-time environment.
  - The model provides immediate classification of posts over a REST API.
  - If post content is classified as toxic/violent/harmful it is taken down/removed.
- **Requirements**: Low latency, immediate action, 24/7 uptime, continuous monitoring

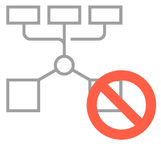
Real-time Deployment

A typical real-time model deployment workflow for LMs



Challenges with building Real-time AI Systems

Most AI systems don't get into production

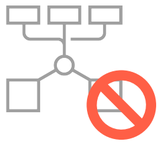


Infrastructure is hard

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

Challenges with building Real-time AI Systems

Most AI systems don’t get into production



Infrastructure is hard

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

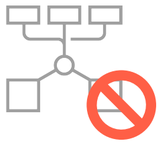
Deploying real time models needs disparate tools

Data teams use diverse tools to develop models

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

Challenges with building Real-time AI Systems

Most AI systems don’t get into production



Infrastructure is hard

Real-time AI 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, LLMs and Serving, adding complexity and cost

Operating production AI requires expert resources

Steep learning curve of deployment tools.

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

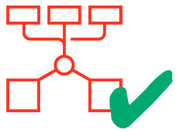
databricks

LECTURE

Databricks Model Serving

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 workloads

Accelerate deployments with Lakehouse-Unified Serving

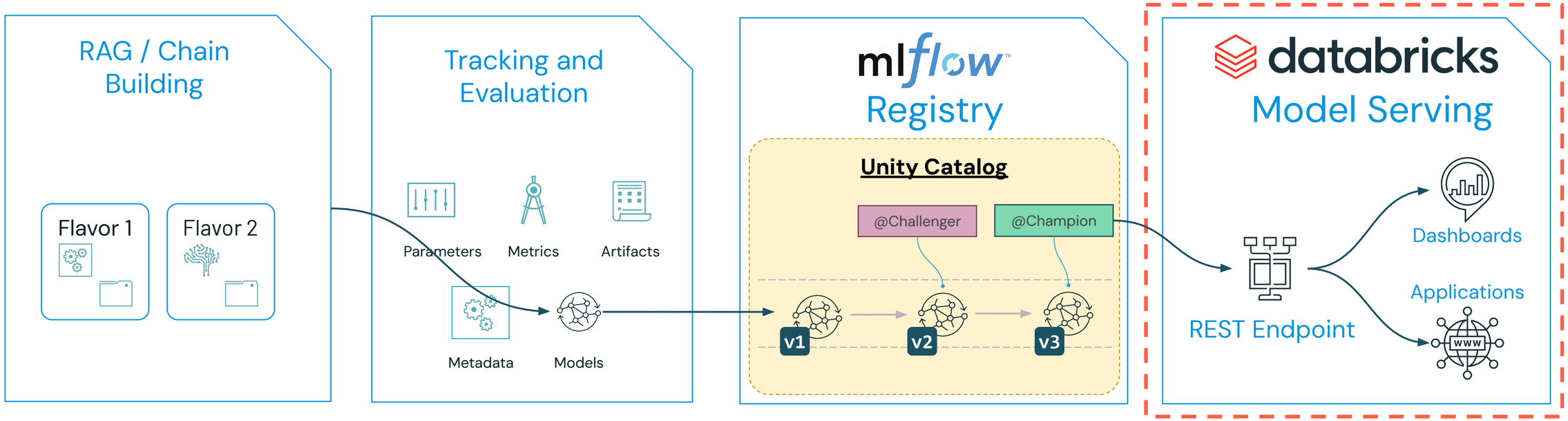
Automatic feature lookups, monitoring and unified governance that **automates deployment** and reduce errors

Simplified Deployment

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

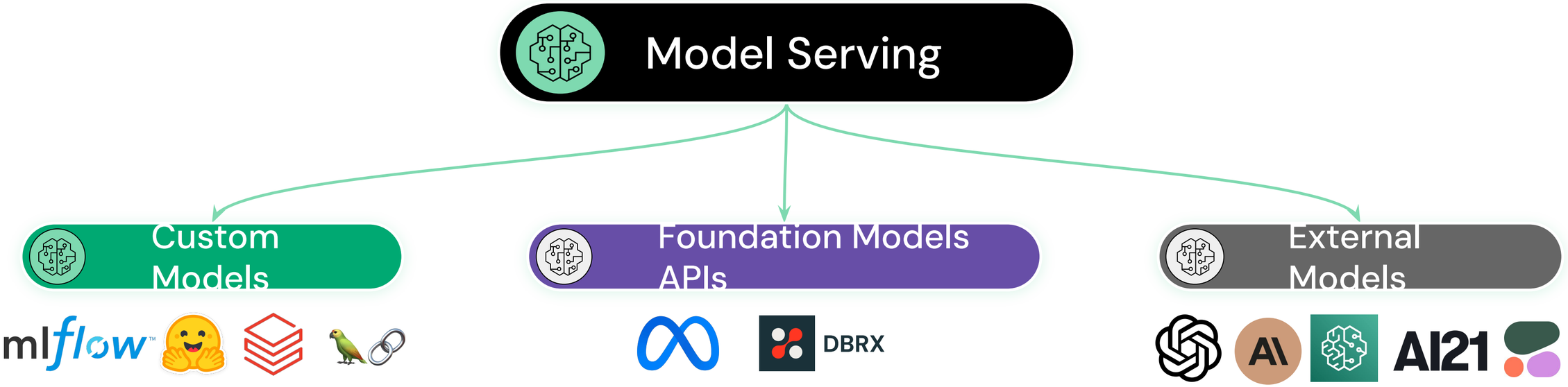
MLflow and Development Lifecycle

MLflow integration to seamlessly deploy models



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.

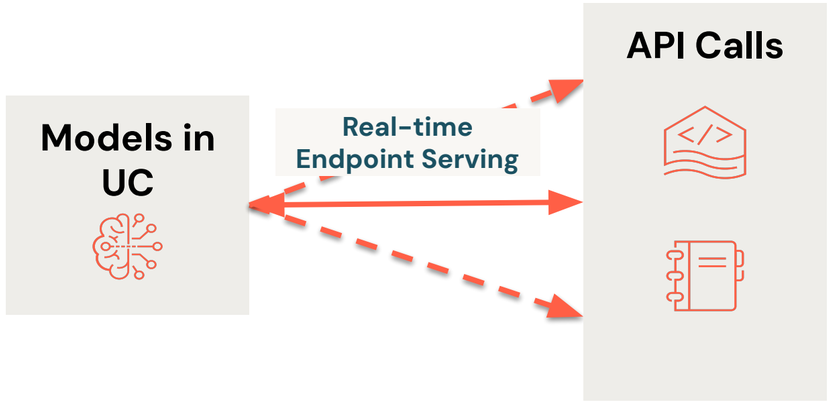
Govern external models and APIs.

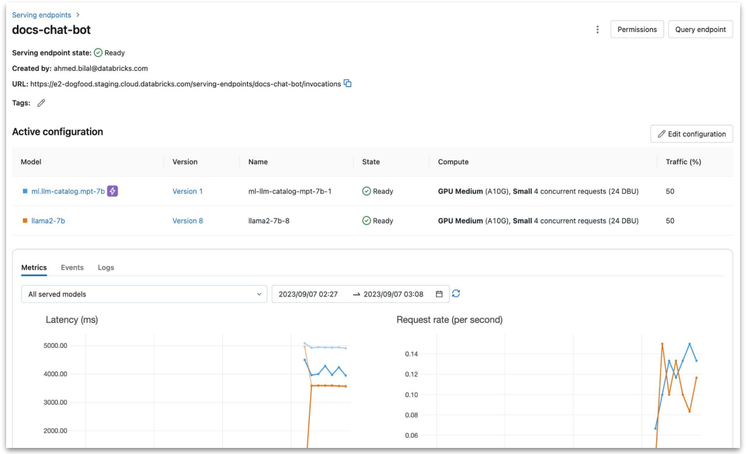
Previously announced as "AI Gateway" at Data+AI Summit 2023.

Databricks Model Serving

Deploy registered models behind real-time, production-grade APIs

- Production-ready, serverless solution to simplify real-time ML model deployment.
- Deploy models as an API to integrate model predictions with applications or websites.
- Built-in payload logging and infrastructure/system observability.



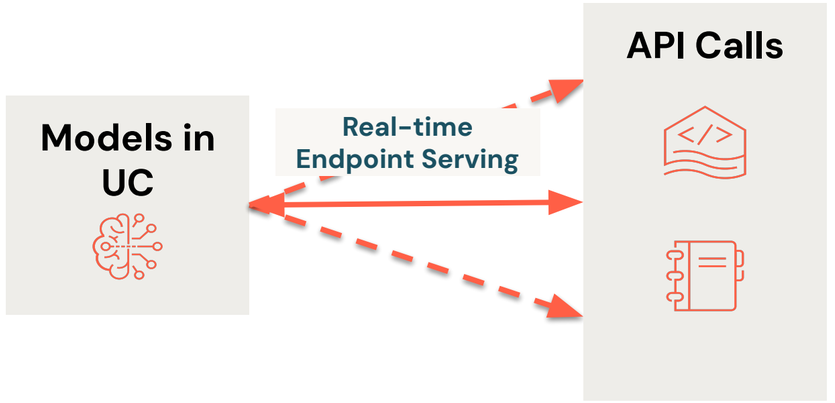


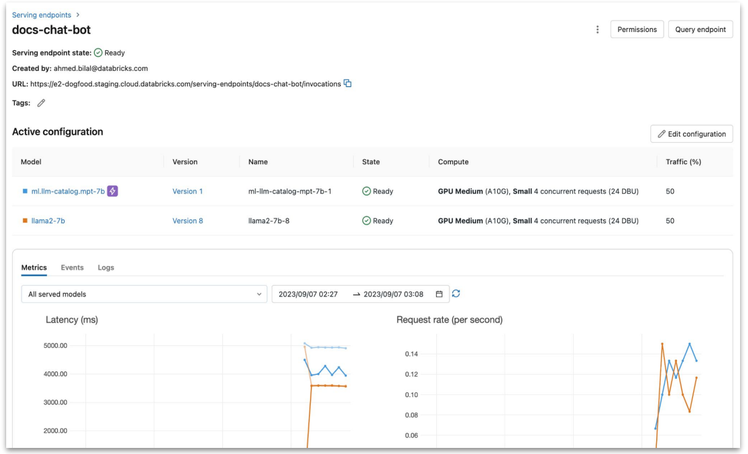
Databricks Model Serving

Deploy registered models behind real-time, production-grade APIs

Benefits:

- Reduces operational costs
- Streamlines the ML lifecycle
- Enables Data Science teams to focus on the core task of integrating production-grade real-time ML into their solutions
- Autoscaling and serverless compute
- Works with custom, foundational, and external models
- Built-in A/B testing

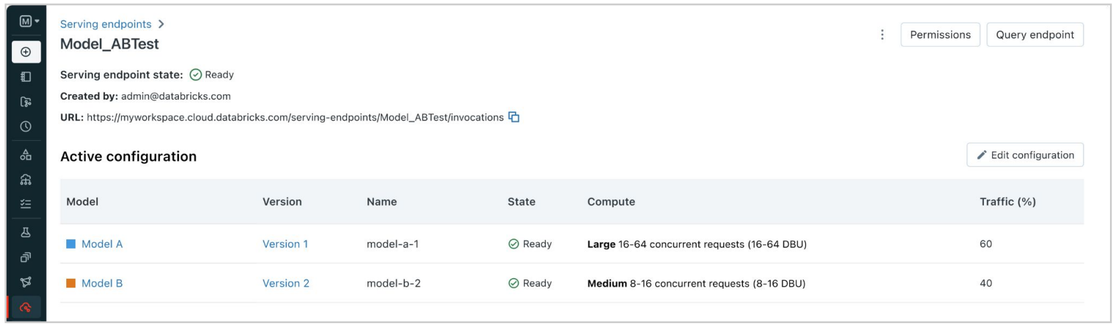


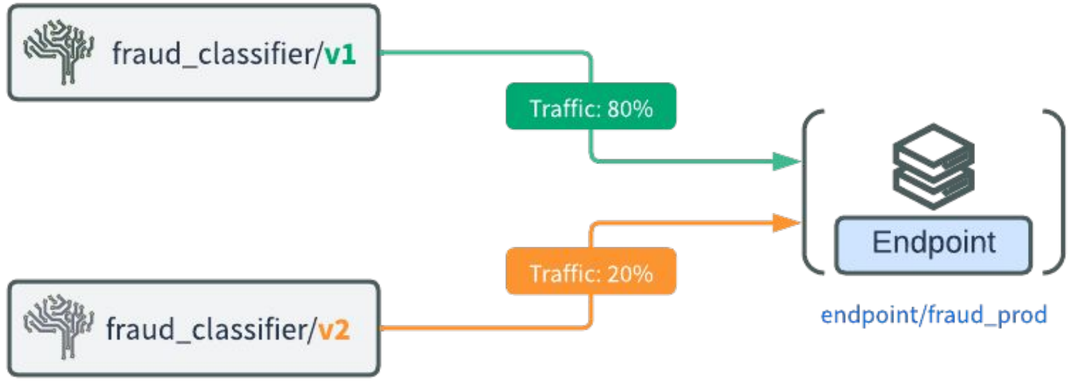


Databricks Model Serving

Online evaluation

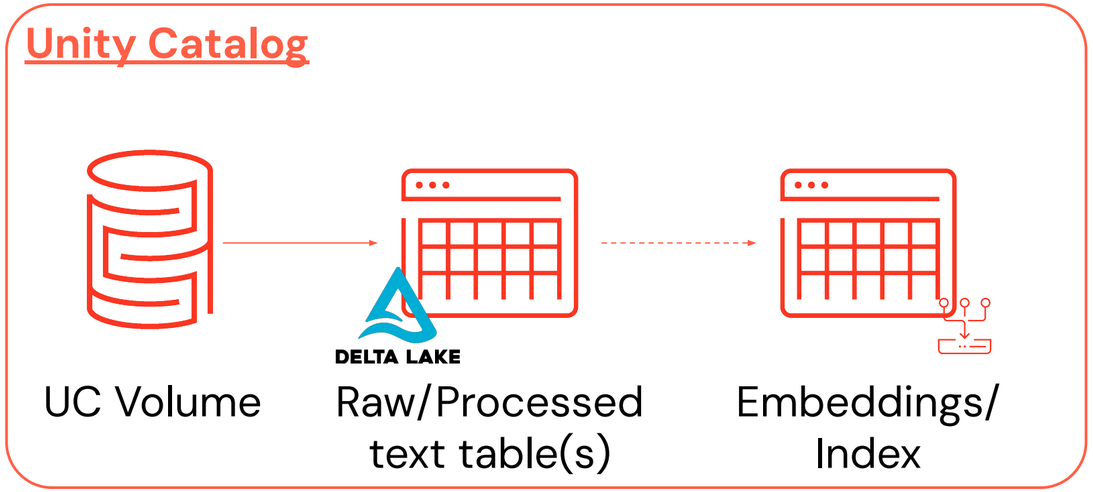
- Supports online evaluation strategies such as A/B testing or canary deployments through the ability to serve multiple models to a serving endpoint





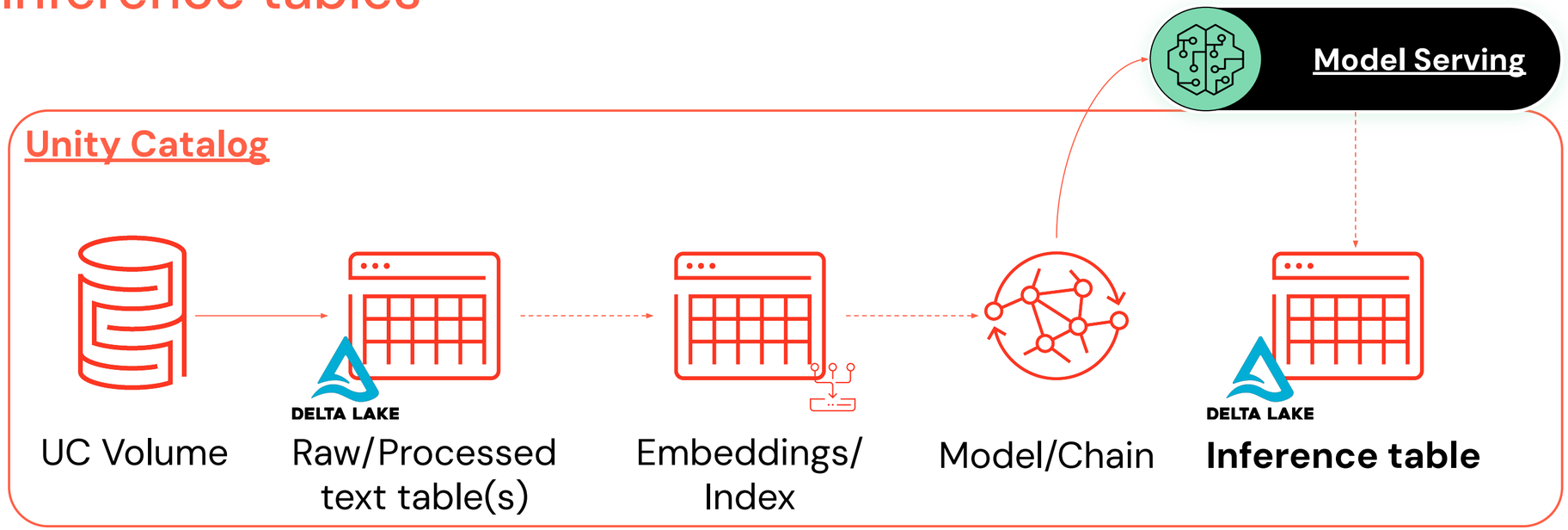
Databricks Model Serving

Inference tables



Databricks Model Serving

Inference tables



Inference tables for monitoring and debugging models:

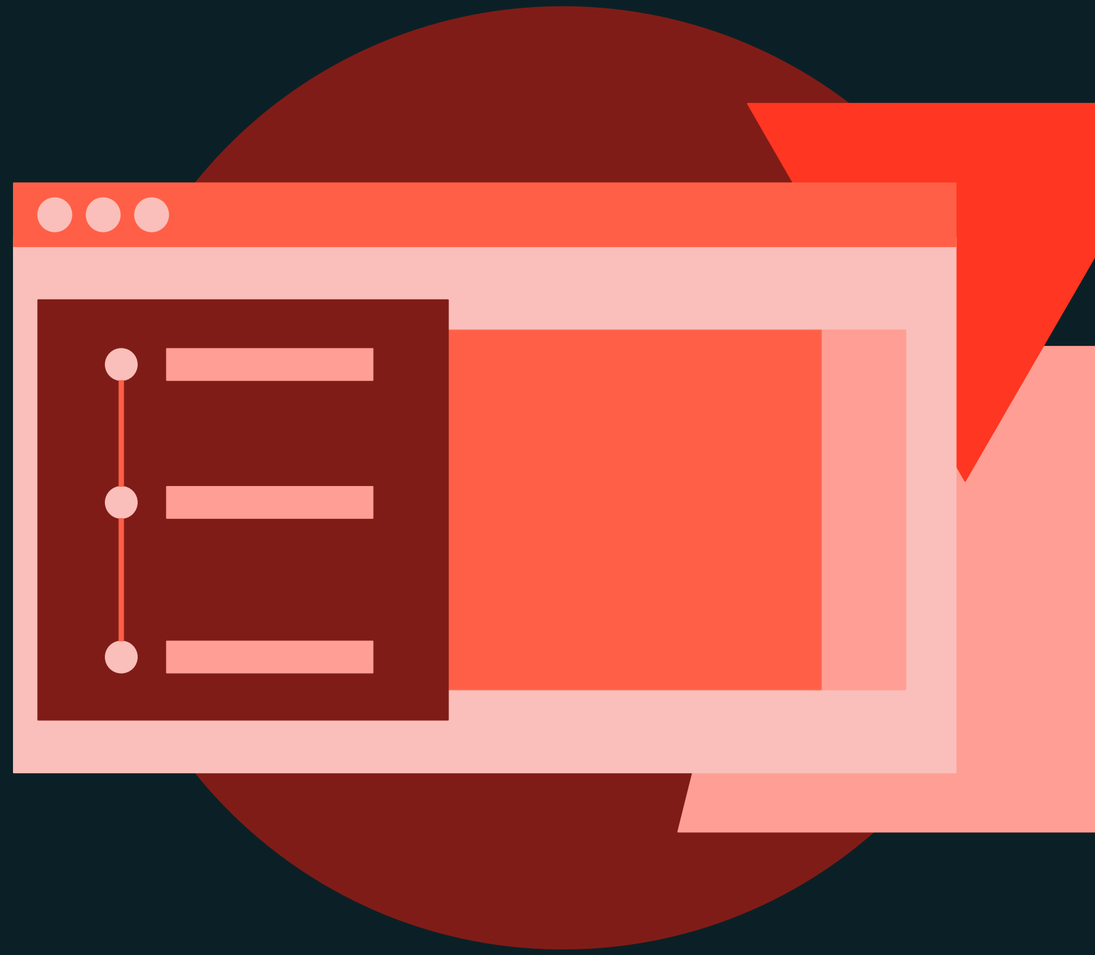
Each [request-response] is appended to a delta table in Unity Catalog

- Use as retraining/fine-tuning dataset for next iteration of your model(s)
- Perform diagnostics and debugging of suspicious inferences
- Create a dataset of mislabels data to be re-labele

databricks

DEMONSTRATION

Serving External Models with Model Serving



Demo Outline

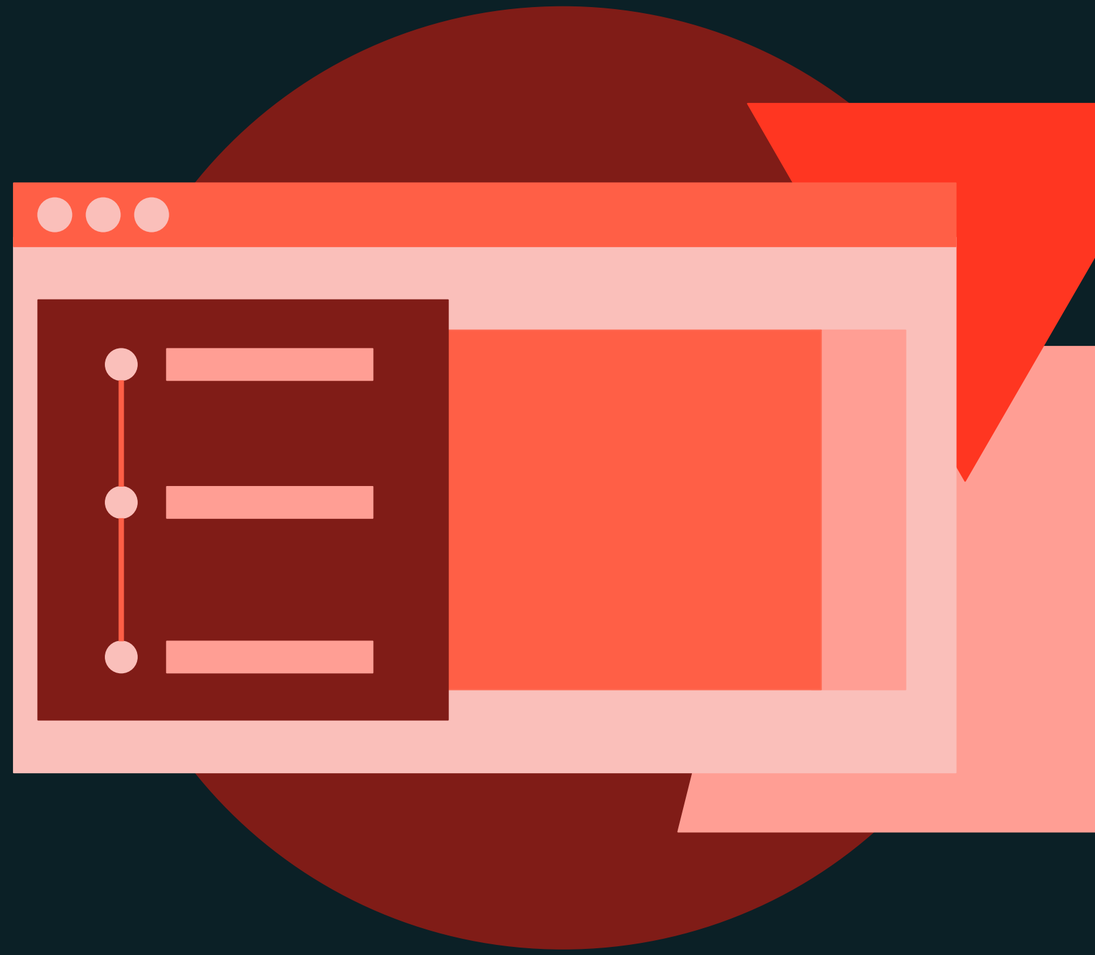
What we'll cover:

- Deploy an external model with Model Serving
- Query served model
  - Query via the UI
  - Query via AI Playground
  - Query via the SDK

databricks

DEMONSTRATION

Deploying an LLM Chain to Databricks Model Serving



Demo Outline

What we'll cover:

- Pre-load a chain/model ready to be deployed
- Deploy chain to a model serving endpoint
  - Explore UI and metrics
- Query served model
  - Query via the UI
  - Query via AI Playground
  - Query via the SDK

databricks

LAB EXERCISE

Custom Model Deployment and A/B Testing

Lab Outline

What you'll do:

- **Task 1**: Task 1: Get Model Version
- **Task 2:** Deploy Model with SDK
  - Configure and Deploy Endpoint
  - Create Inference Table via Model Serving UI
- **Task 3:** Configure A/B Testing Using the UI
- **Task 4:** Query the Endpoint
- **Task 5:** Inspect Inference Table

databricks

AI System Monitoring

Generative AI Deployment and Monitoring

Learning Objectives

- Explain the importance of ongoing AI system monitoring.
- List relevant metrics to be analyzed for various AI system components and the AI system as a whole.
- Describe Lakehouse Monitoring as a tool for monitoring the online performance and stability of your GenAI application.

databricks

LECTURE

AI Application Monitoring

Monitoring AI Systems

Continuous logging and review of key component/system metrics

Why?

Used to help diagnose issues before they become **severe** or **costly**

Data to Monitor

Input data (tricky with existing models)
Data in vector databases/knowledge bases
Human feedback data
Prompt/queries and responses (legality)

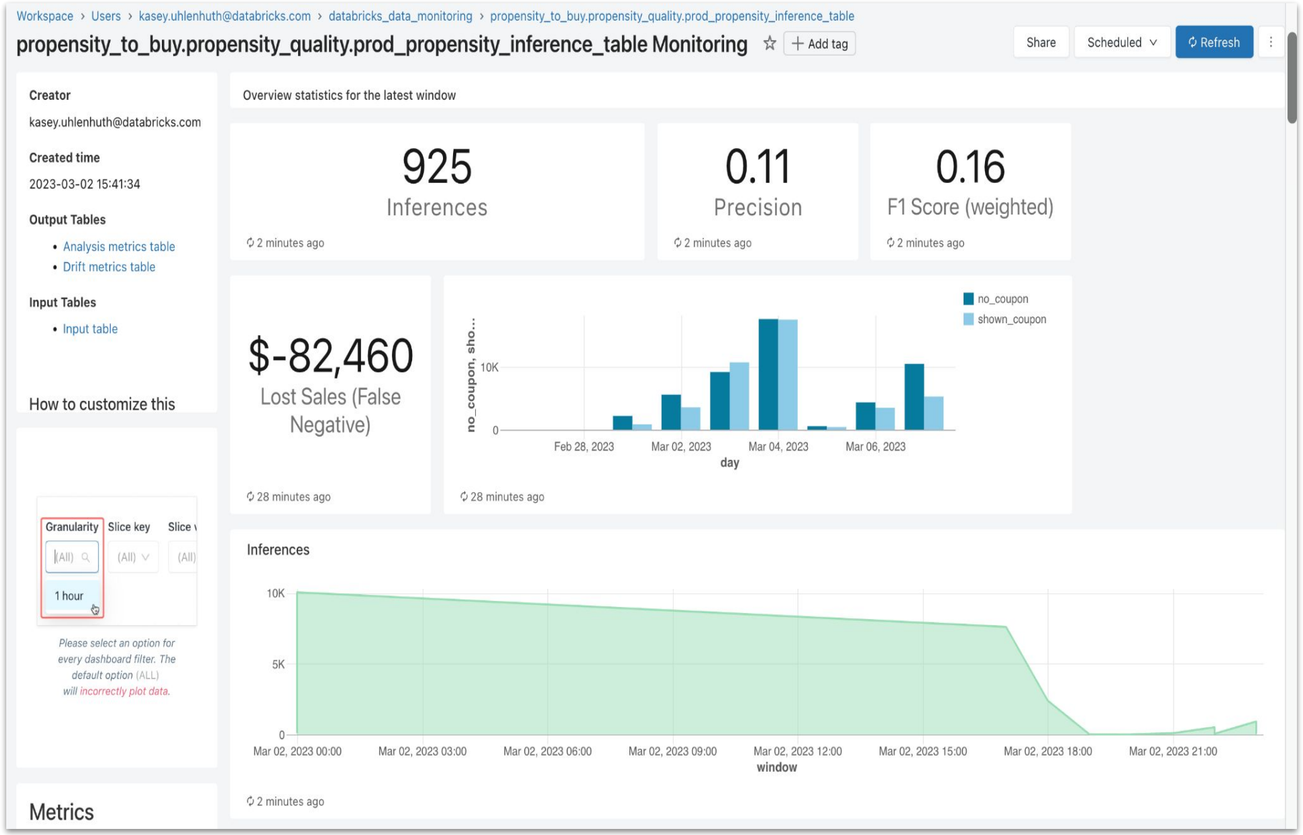
AI Assets to Monitor

Mid-training checkpoints for analysis
Component evaluation metrics
AI system evaluation metrics
Performance/cost details

Databricks Lakehouse Monitoring

Automated insights and out-of-the box metrics on data and ML pipelines

- **Fully managed** so no time wasted managing infrastructure, calculating metrics, or building dashboards from scratch
- **Frictionless** with easy setup and out-of-the-box metrics and generated dashboards
- **Unified** solution for data and models for holistic understanding

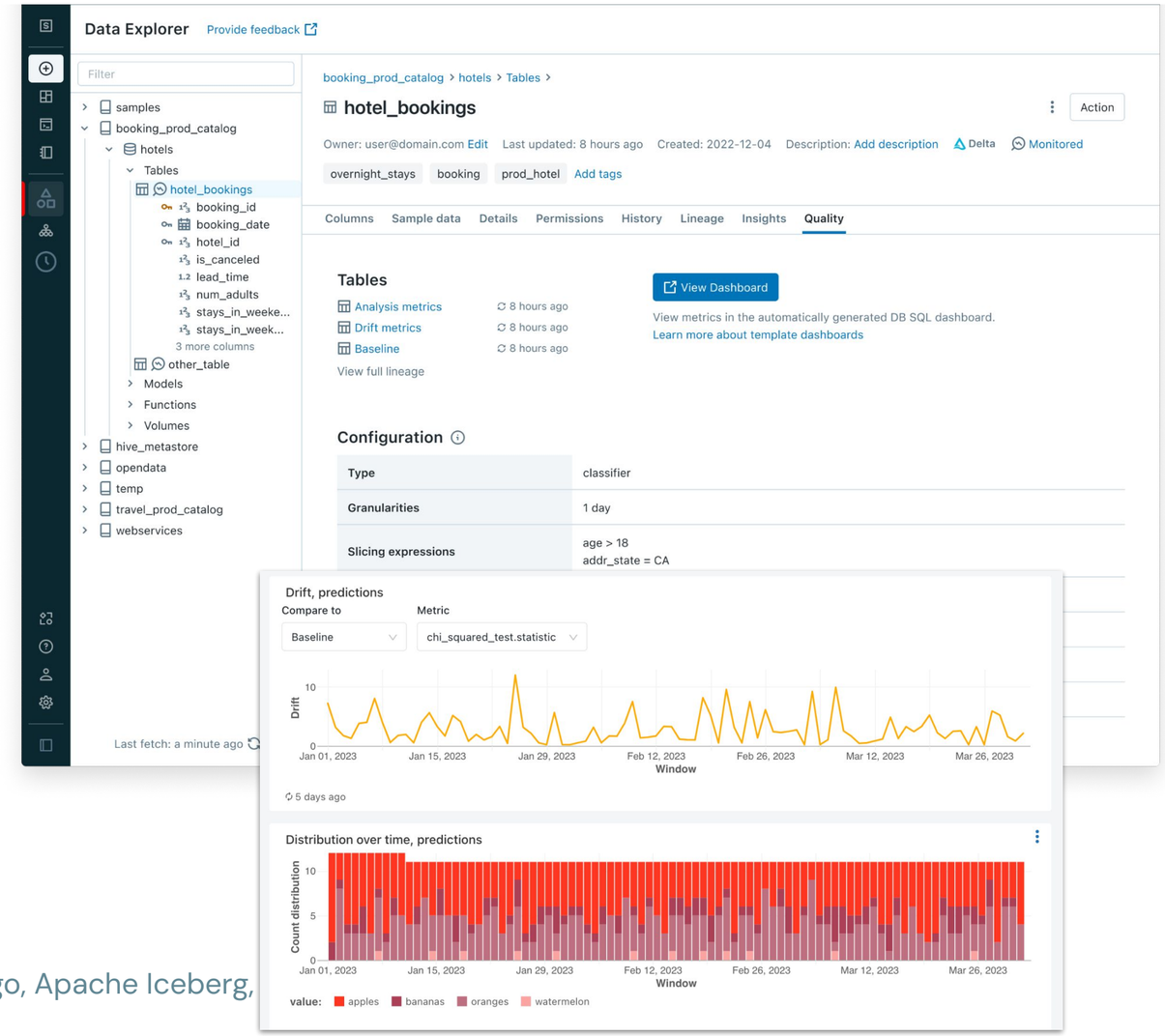


Built on Unity Catalog

Background service that incrementally processes data in Unity Catalog tables

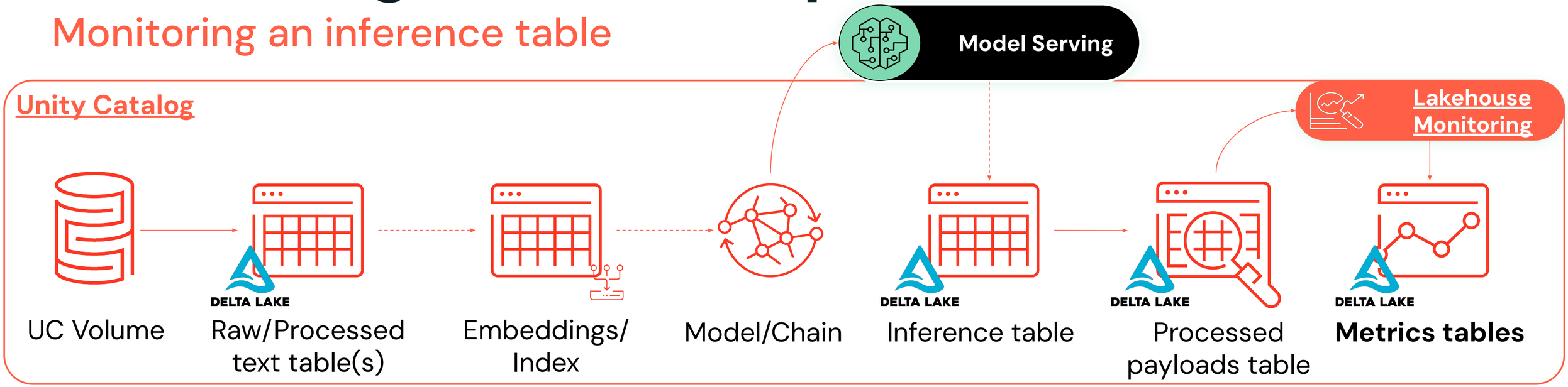
For each monitored table:

- Calculates **profile metrics** table
- Calculates **drift metrics** table
- Supports **custom metrics** as SQL expressions
- **Auto-generates DBSQL dashboard** to visualize metrics over time
- Automatic PII detection
- Input expectations & rules



Monitoring model's responses

Monitoring an inference table



Create quality monitoring of production data and models

1. **Unpack inference table**, calculate LLM-related evaluation metrics and materialize into processed table
   a. Schedule this as a triggered streaming job
2. **Enable Lakehouse Monitoring** on processed payloads table
3. Analyze dashboard and create SQL alerts on relevant calculated metrics

Lakehouse Monitoring Workflows

A comprehensive monitoring tool for your data and AI system

Architecture Stages

- Development
  - Build into project within dev environment
- Testing
  - Develop integration tests that run a few iterations to ensure all monitoring is working
- Production
  - Deploy as part of the project to Production, including generating a baseline table
  - Write monitor tables to the prod catalog

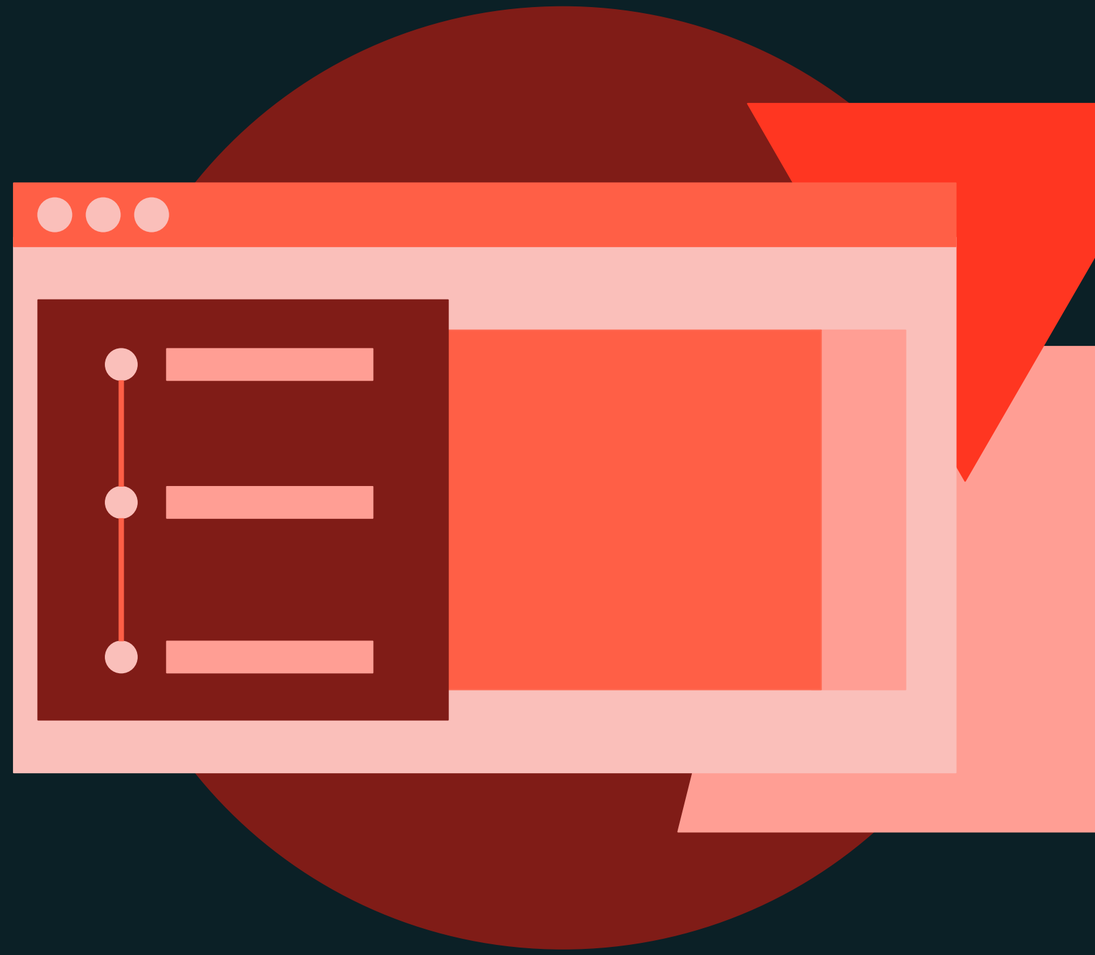
Workflow Tips

1. Set up monitoring tables for all components
2. Model cost against performance
3. Refresh the tables and dashboard regularly
4. Set up key monitoring alerts
5. Connect rerun triggers to performance

databricks

DEMONSTRATION

Online Monitoring an LLM RAG Chain



Demo Outline

What we'll cover:

- Unpack inference table for previously deployed RAG chain
- Calculate, append additional LLM-related metrics and materialize
- Create lakehouse monitor via UI (and API)
- Navigate generated dashboard
- Create SQL alert on specific metric

databricks

LAB EXERCISE

Online Monitoring

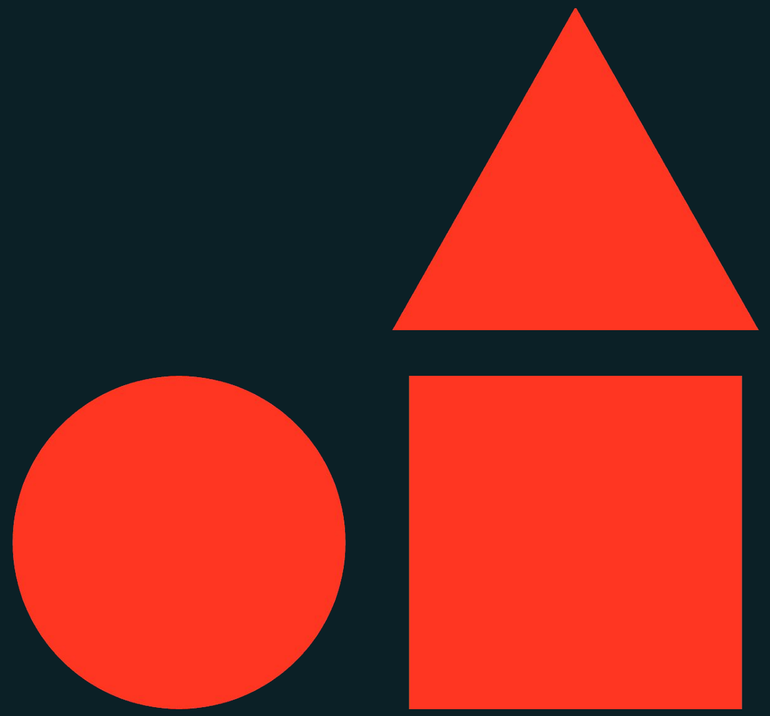
Lab Outline

What you'll do:

- **Task 1:** Define Evaluation Metrics
- **Task 2:** Unpack the Request Payload
- **Task 3:** Compute Metrics
- **Task 4:** Save the Processed Inference Table
- **Task 5:** Create a Monitor on the Inference Table
- **Task 6:** Review the Monitor Details
- **Task 7:** View the Monitor Dashboard

databricks

LLMOps Concepts



Generative AI Deployment and Monitoring

Learning Objectives

- Describe the basics of LLMOps.
- Compare and contrast LLMOps from MLOps.
- Explain the recommended LLMOps architecture and its requirements.
- Describe workflows associated with each stage of the LLMOps cycle.

databricks

LECTURE

MLOps Primer

Let us start from conventional ML

What is MLOps

MLOps is the set of **processes and automation**

for **managing data, code and models**

to **improve performance, stability and long-term efficiency** of ML systems

MLOps = DataOps + DevOps + ModelOps

Why does MLOps matter?

Success depends on quality data and operations practices

- Defining an effective strategy
  - ML systems built on **quality data**
  - **Streamlining** process of taking solutions to **production**
  - Operationalizing **cost & performance monitoring**
- So what?
  - Accelerated time to realizing business value
  - Reduction in manual oversight

Real-world Example

Databricks customer CareSource accelerated their model's development and deployment, resulting in a **self-service MLOps solution for data scientists** that reduced ML project time from 8 weeks to 3-4 weeks.

The CareSource team can extend this approach to other machine learning projects, realizing this benefit broadly.

Learn more about the work here.

Multi-environment Semantics

Defining Development, Staging, and Production environments

Development

An environment where data scientists can **explore, experiment, and develop**

Staging

An environment where machine learning practitioners can **test** their solutions

Production

An environment where machine learning engineers can **deploy and monitor** their solutions

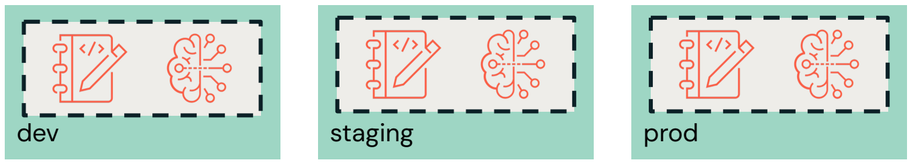
Environment Separation

How many Databricks workspaces do we need?

Model Code

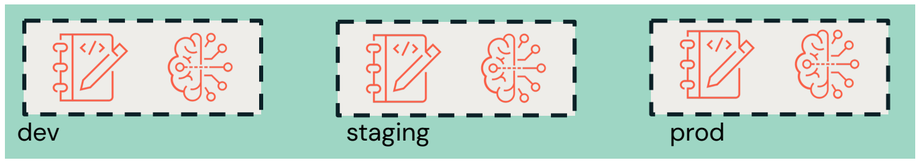
Model Artifact

Direct Separation



- Completed separate Databricks workspaces for each environment
- Simpler environments
- Scales well to multiple projects

Indirect Separation



- One Databricks workspace with enforced separation
- Simpler overall infrastructure requiring less permission required
- Complex individual environment
- Doesn’t scale well to multiple projects

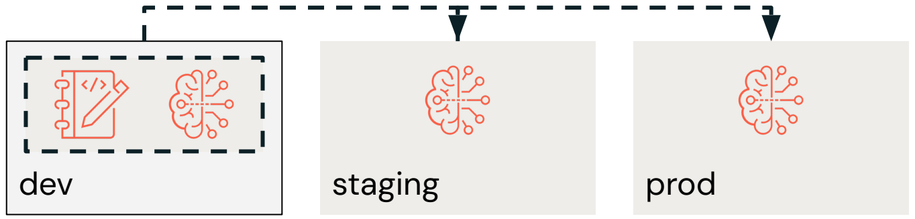
Deployment Patterns

Moving from Deploy Model to Deploy Code

Model Code

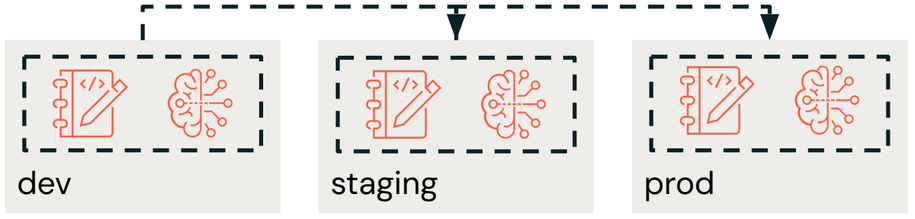
Model Artifact

Deploy Model



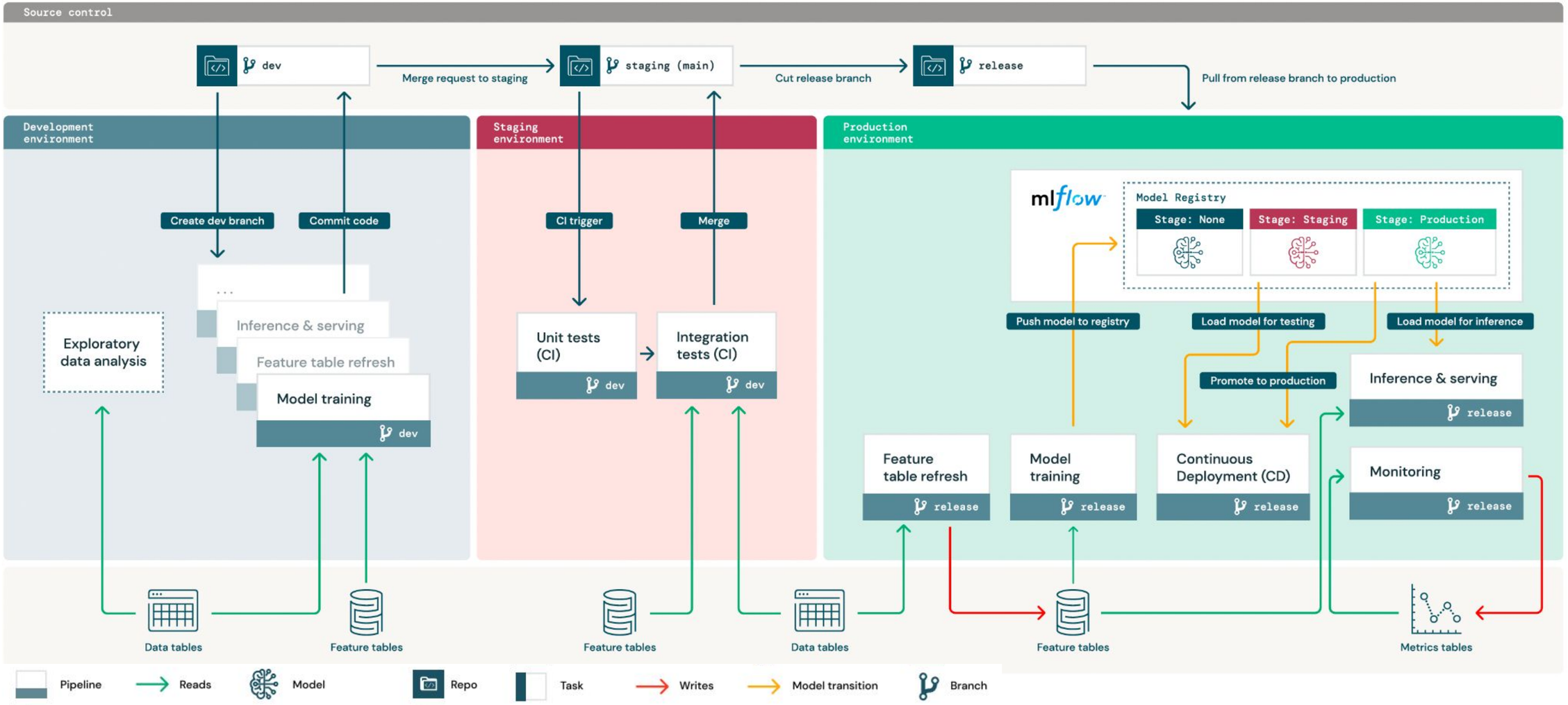
- Model is trained in development environment
- Model artifact is moved from staging through production
- Separate process needed for other code (inference, monitoring, operational pipelines)

Deploy Code (recommended)



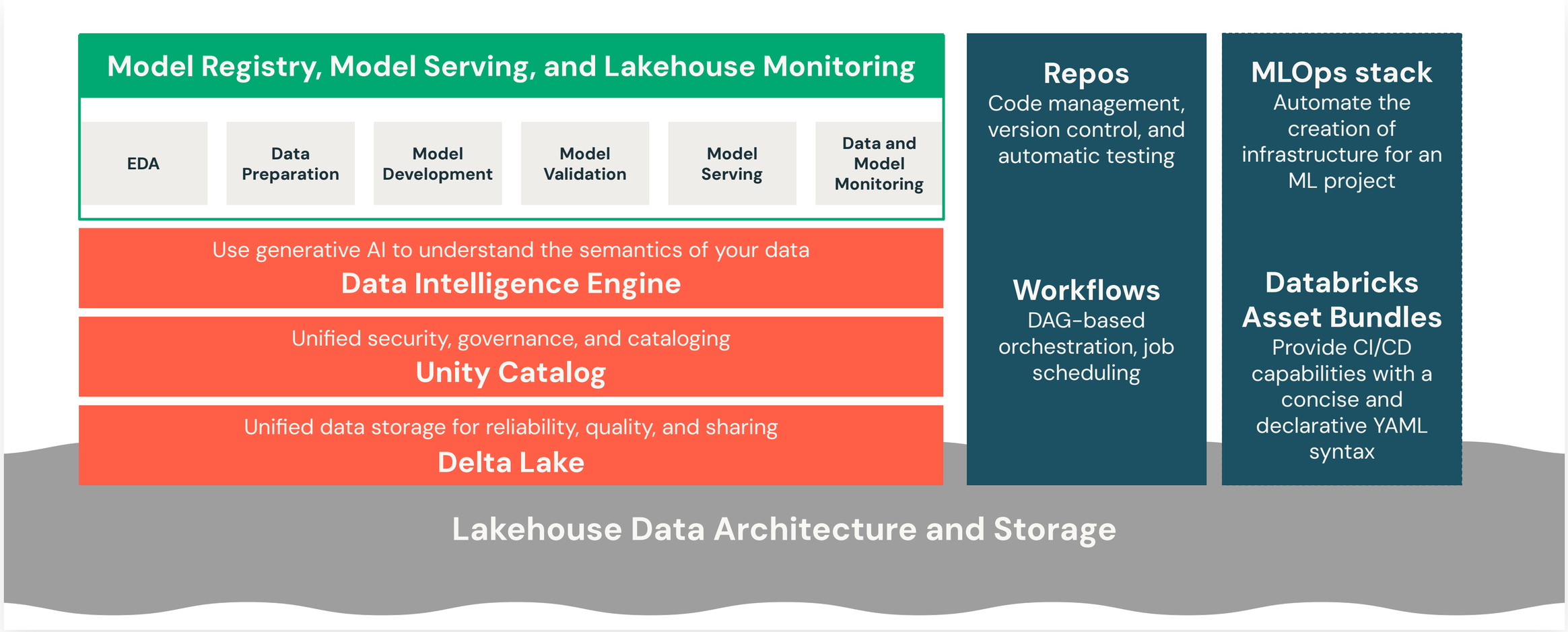
- Code is developed in the development environment
- Code is tested in the staging environment
- Code is deployed in the production environment
- Training pipeline is run in each environment, model is deployed in production

Recommended “deploy-code” architecture (MLOps)



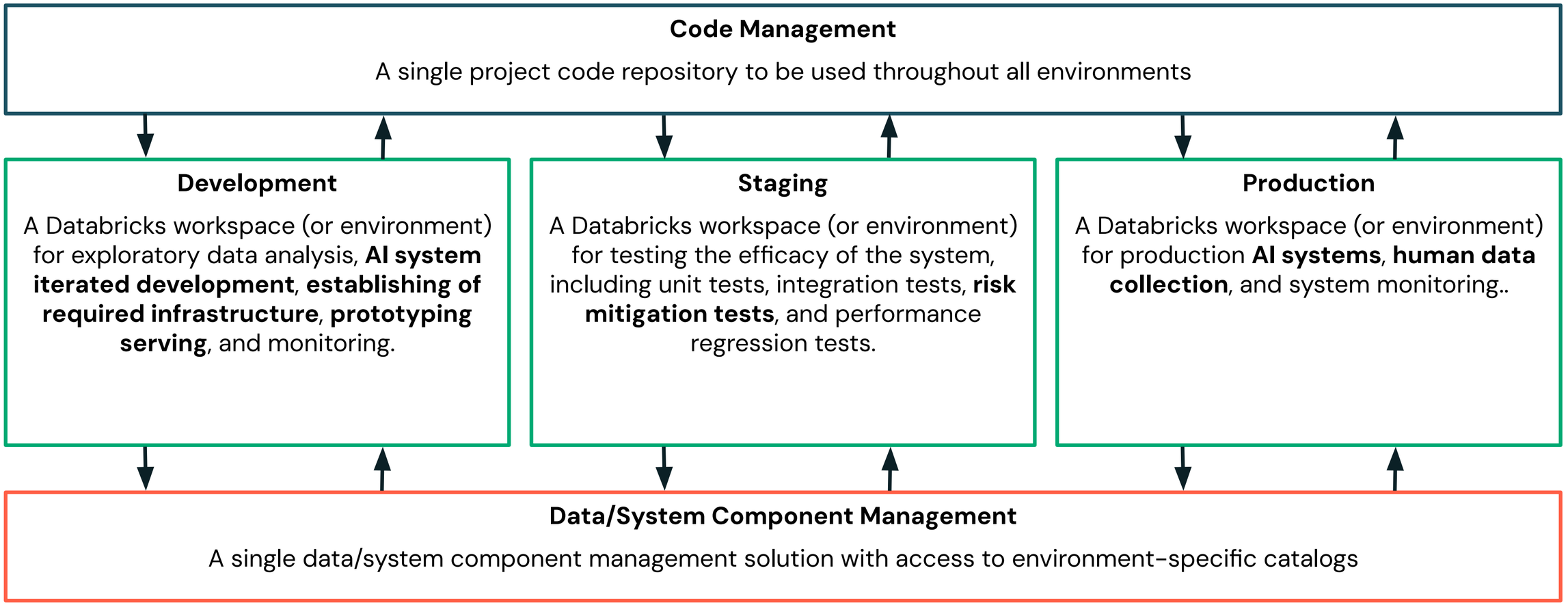
A Single Platform for Modern MLOps

Combining DataOps, DevOps, and ModelOps solutions



Recommended LLMOps Architecture

A high-level view of code, data, and GenAI environments



databricks

LECTURE

LLMOps vs MLOps

What About LLMOps?

Comparing and contrasting LLMOps from traditional MLOps by area

Dev Patterns	Packaging	Serving	API Governance	Cost and Performance	Human Feedback
The workflows and patterns that practitioners follow when developing machine learning and generative AI solutions	The composition of entire machine learning and generative AI applications in order to effectively deliver and scale solutions	The deployment of machine learning and generative AI applications for output storage or delivery to the end user.	The access control and management of APIs used as a part of the machine learning or generative AI application.	The constant measurement and evaluation of the cost of developing and maintaining the application and its value and performance in production.	The inclusion of data generated by humans into the evaluation or iteration of machine learning and generative AI applications.

Note: We'll walk through each of these areas and map to the LLMOps reference architecture.

Development Patterns in LLMOps

Comparing and contrasting LLMOps from traditional MLOps

Dev Patterns	Packaging	Serving	API Governance	Cost and Performance	Human Feedback

- **Incremental** development patterns
  - Developers typically **start with off-the-shelf external models and move toward more custom models** through RAG, fine-tuning, or pre-training
- Inclusion of text templates
  - Many interfaces to GenAI solutions **include text templates** to facilitate LLM behavior within the application – these must be developed, iterated on, and managed as part of an LLM pipeline

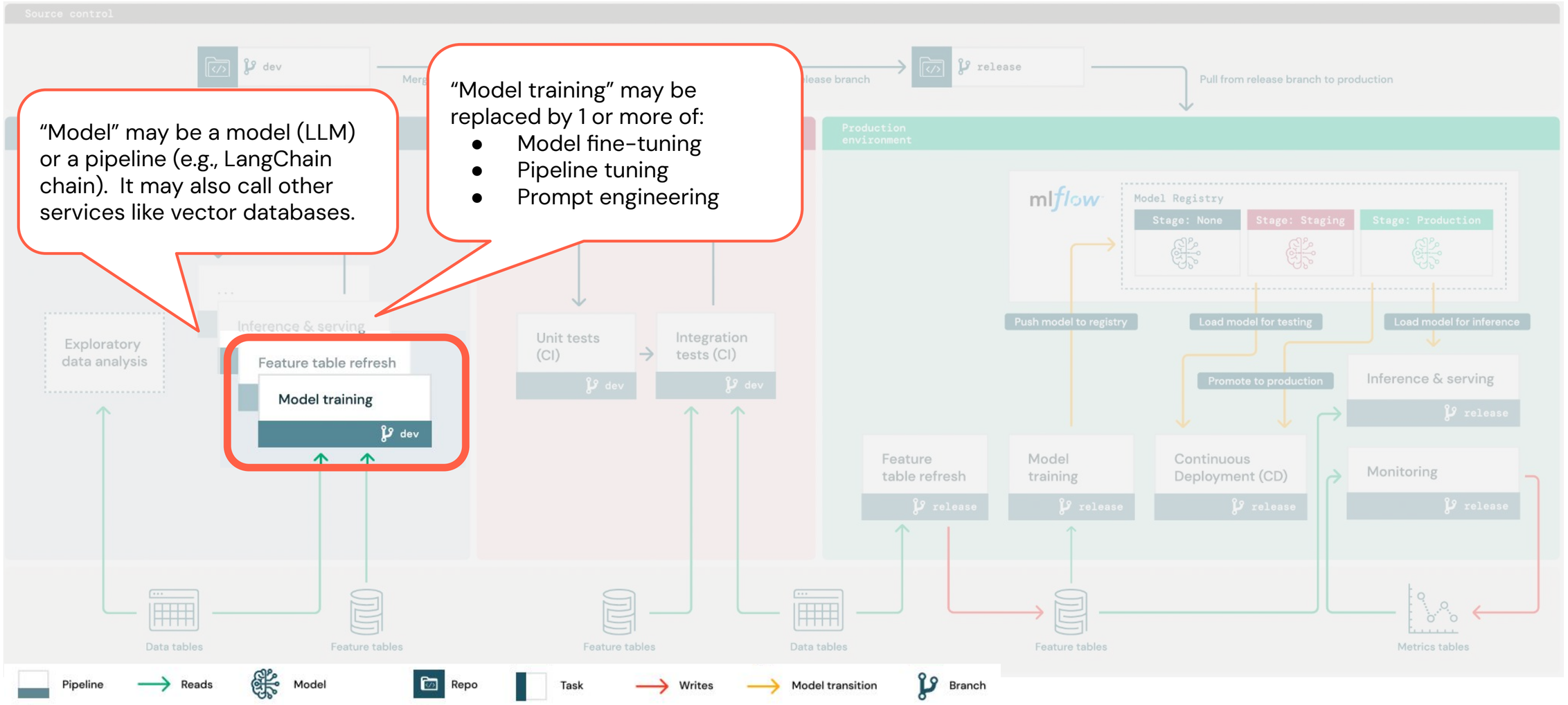
Packaging Artifacts in LLMOps

Comparing and contrasting LLMOps from traditional MLOps

Dev Patterns	**Packaging**	Serving	API Governance	Cost and Performance	Human Feedback

- Package entire applications
  - Rather than deploying a single model, large applications need to be packaged and deployed together
  - This can include calls to other LLMs or GenAI applications as a part of a deployed pipeline
- Environment configurations
  - Additional configurations need to be made to the package environment to manage things like other endpoints and embeddings, including testing external systems are working

Recommended “deploy-code” architecture (LLMOps)



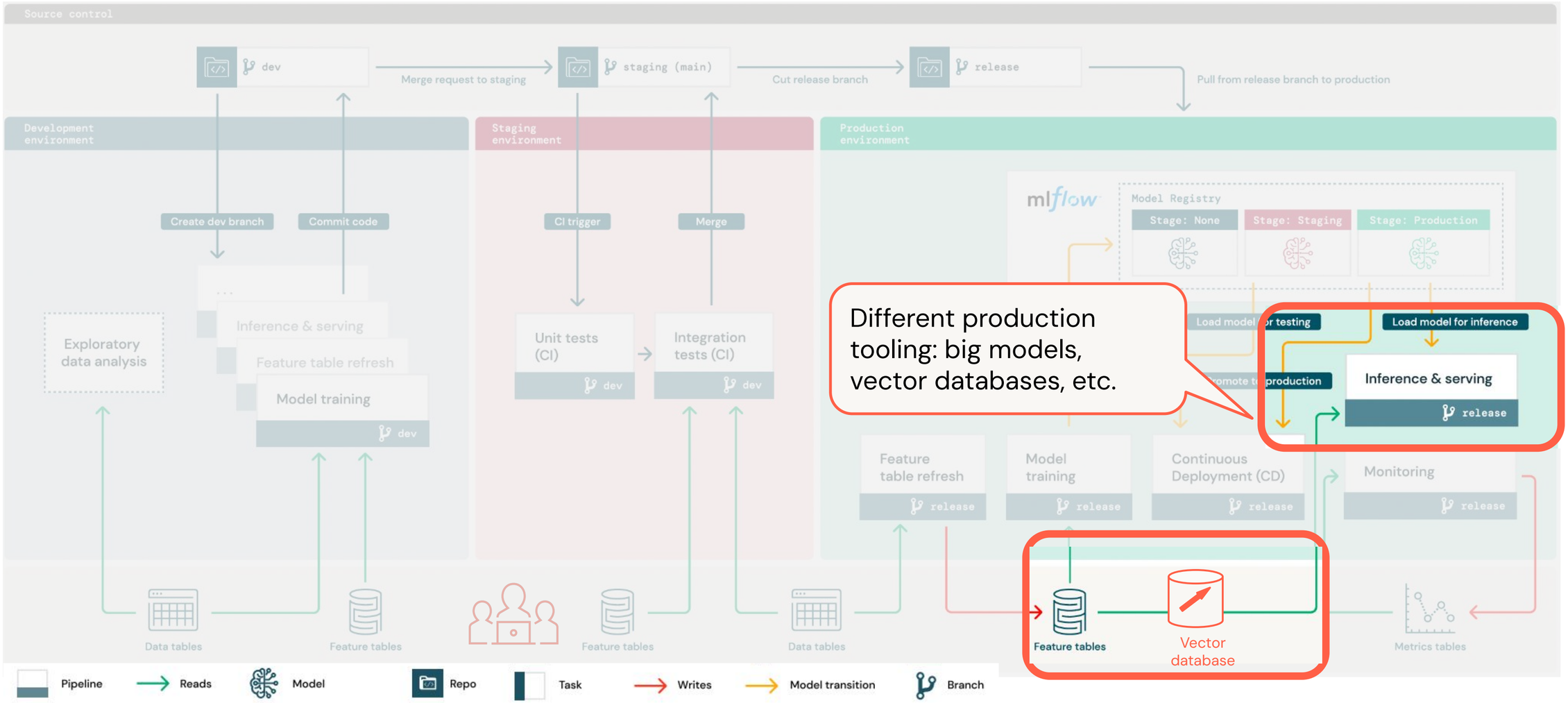
Serving Applications in LLMOps

Comparing and contrasting LLMOps from traditional MLOps

Dev Patterns	Packaging	**Serving**	API Governance	Cost and Performance	Human Feedback

- Additional components like vector databases
  - When augmenting LLM applications with contextual data, new systems like vector databases need to be set up and used for scale
- GPU infrastructure
  - Due to the size of language models used in many applications, GPUs are required for the serving and deployment of AI systems
- User interface components
  - While some might choose to manage this separately, developers might need to include end user interface components as a part of their serving deployment

Recommended “deploy-code” architecture (LLMOps)



API Governance in LLMOps

Comparing and contrasting LLMOps from traditional MLOps

Dev Patterns	Packaging	Serving	API Governance	Cost and Performance	Human Feedback

- Managing access to endpoints
  - With the addition of new endpoints being created for various components of the project, access to those endpoints needs to be governed

- Governing use of application
  - Calls to endpoints should be logged and reviewed regularly to evaluate abuse
  - Guardrails can be put in place to prohibit future abuse

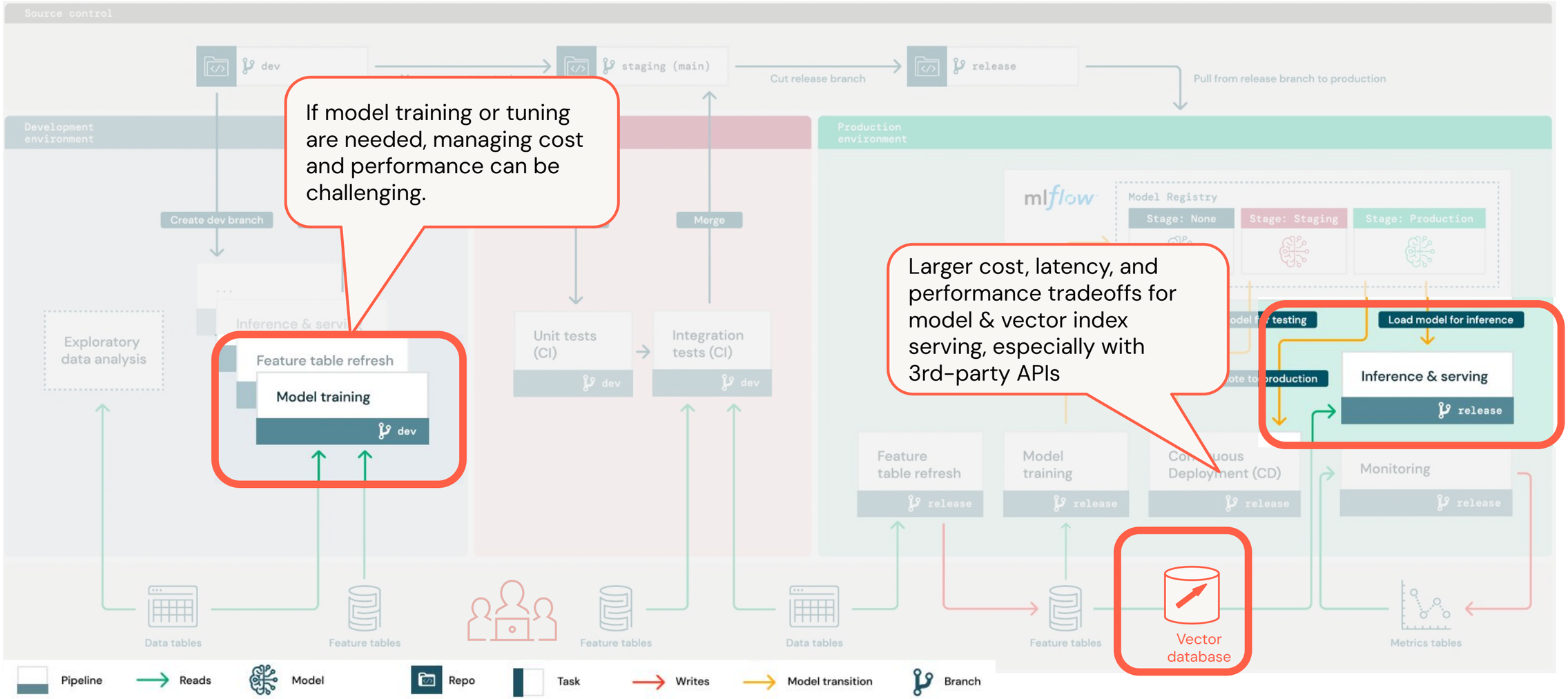
Cost and Performance in LLMOps

Comparing and contrasting LLMOps from traditional MLOps

Dev Patterns	Packaging	Serving	API Governance	Cost and Performance	Human Feedback

- Model size increases cost
  - LLMs are larger in size, and frequently result in more cost
- (API-based) Models have use-based cost
  - (External/FMAPI) Models offers per-token based pricing that needs to be considered
- Techniques for reducing model size/cost
  - Reduce model size
  - Reduce number of queries to external models, if possible (use “caching”)
  - Reduce query input and/or constraint output size

Recommended “deploy-code” architecture (LLMOps)



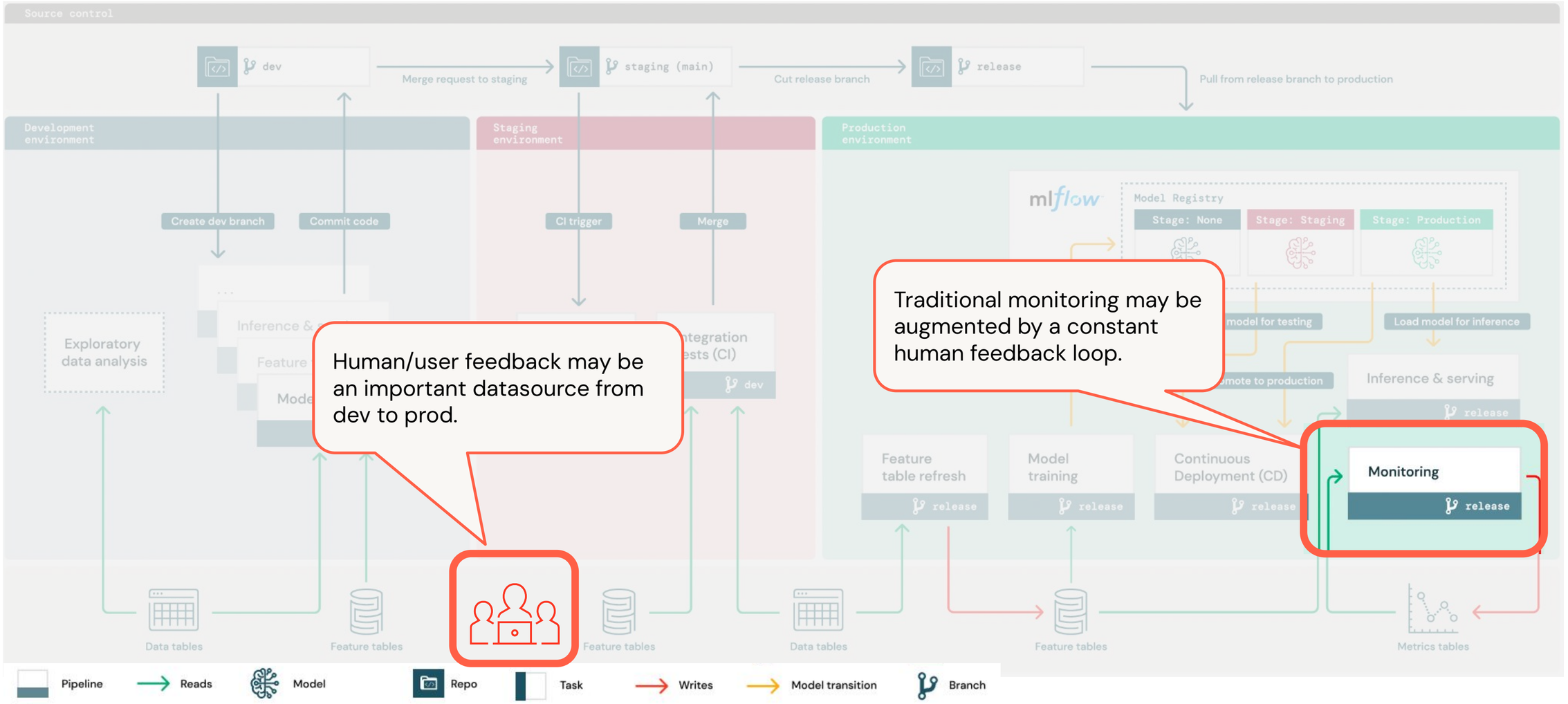
Human Feedback in LLMOps

Comparing and contrasting LLMOps from traditional MLOps

Dev Patterns	Packaging	Serving	API Governance	Cost and Performance	Human Feedback

- Collection of data
  - Explicit satisfaction *(i.e. with* 👍 *and* 👎*)*
  - User queries
  - Human/Users corrections/suggestions
- Use of data (managed!)
  - Response examples
  - Tests
  - Architectural/strategic decisions

Adapting MLOps for LLMs



Databricks Asset Bundles (DABs)

Databricks Asset Bundles

Where are bundles used?

Write code once, deploy everywhere

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

What are Databricks Asset Bundles?

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

How do bundles work?

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

A Closer Look

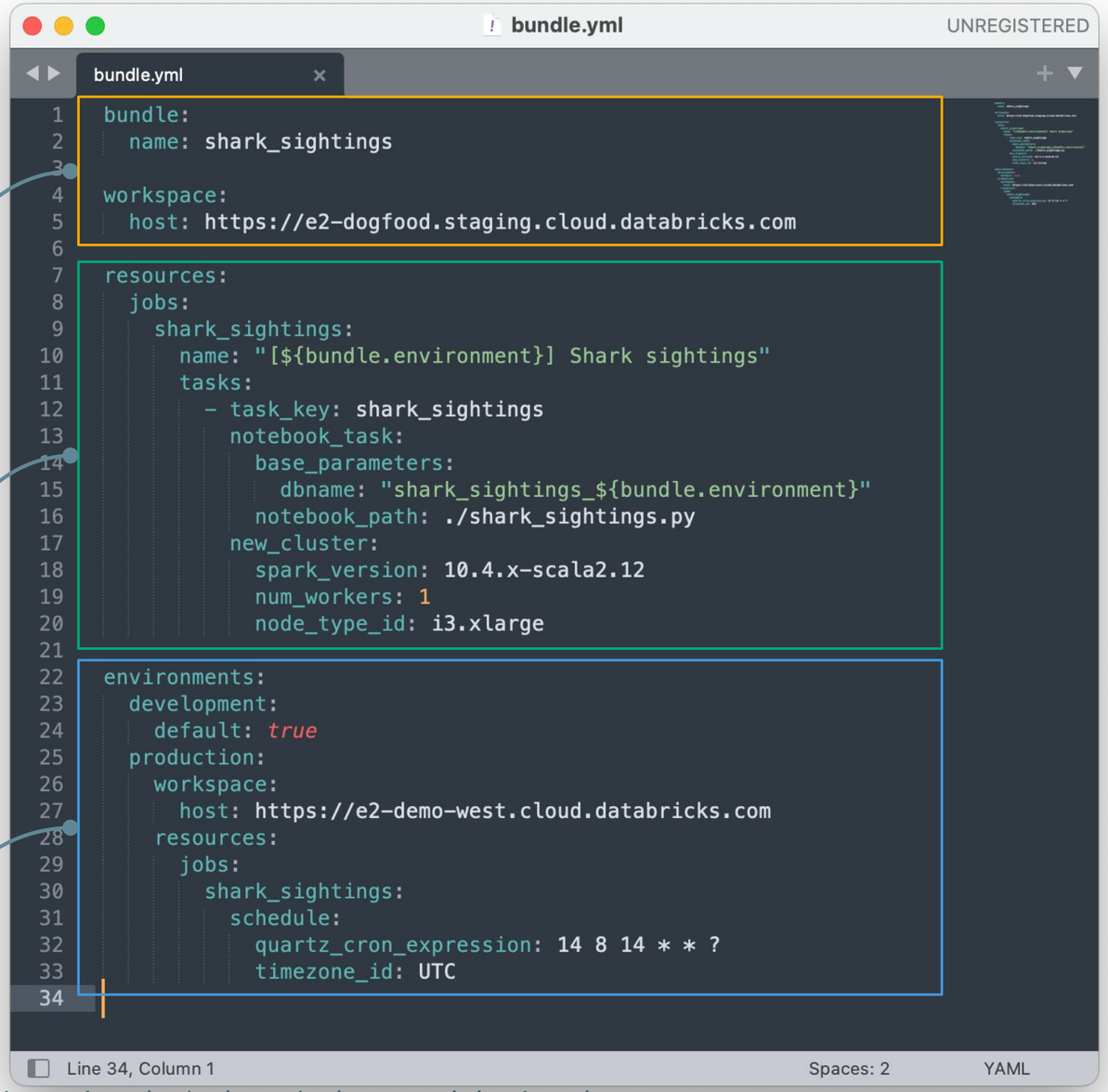
Name and default Workspace

Resource configurations

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

Environment-based specs

- Control project behavior in different environments



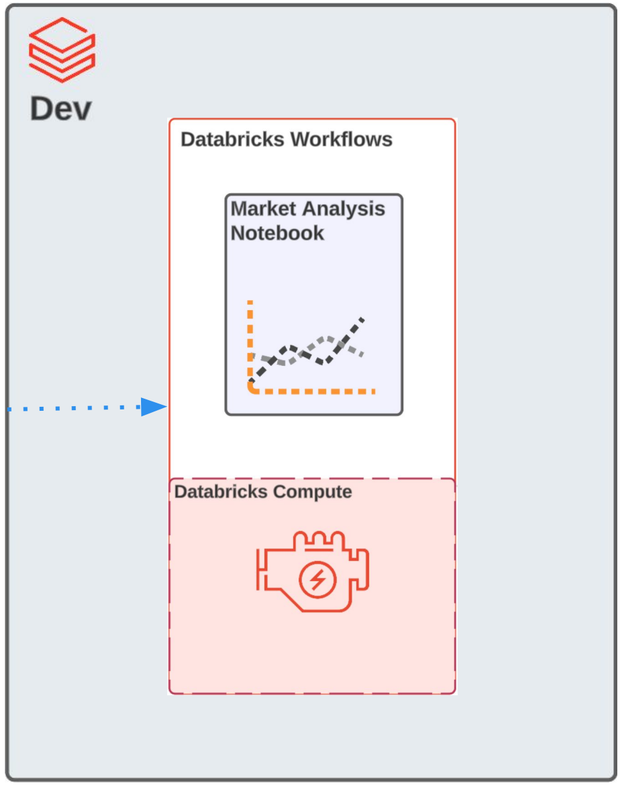
Where Can We Use Bundles?

As part of active development

Alice

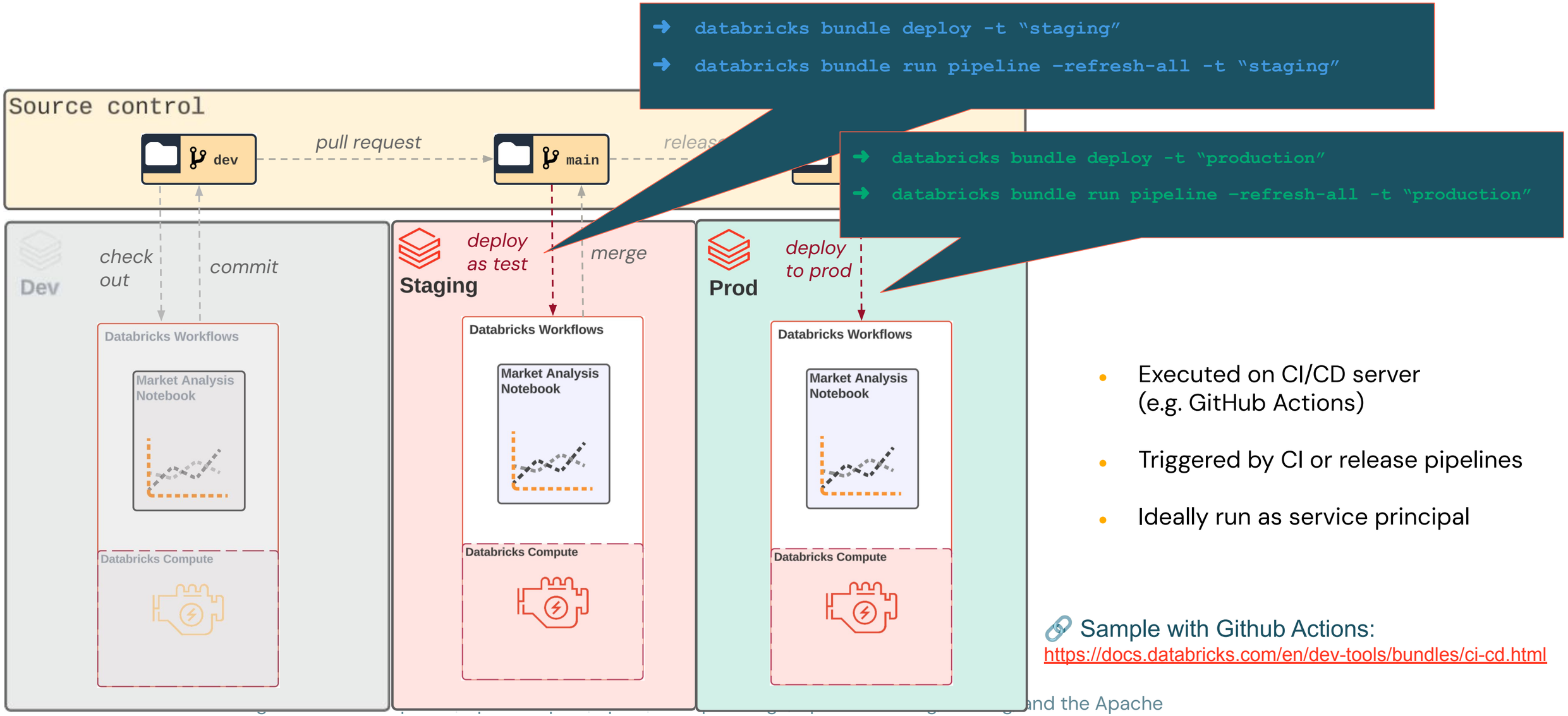
```
$ databricks bundle deploy -t "development"

$ databricks bundle run pipeline -refresh-all -t "development"
```



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

Where can we use bundles?



- Executed on CI/CD server (e.g. GitHub Actions)
- Triggered by CI or release pipelines
- Ideally run as service principal

🔗 Sample with Github Actions:
https://docs.databricks.com/en/dev-tools/bundles/ci-cd.html

Benefits of Using DABs

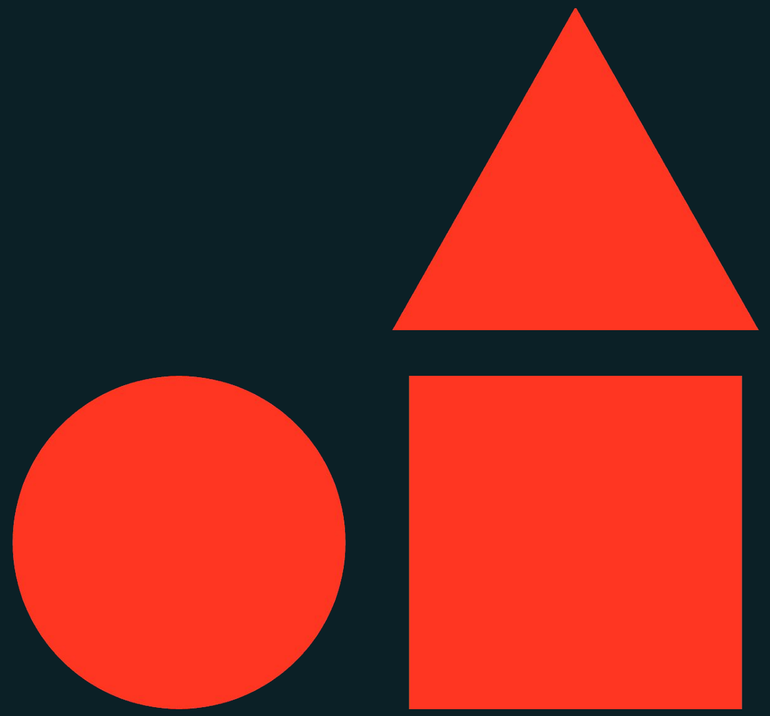
Write code once, deploy anywhere

- **Democratizing best practices for CI/CD and project management**
  - Co-version configs with code and resources
  - Automation friendly (CLI vs. 1000 REST calls)
  - Collaborate and deploy with confidence (isolation, automated testing, etc)

- **Unified, single way to define and deploy your projects**

databricks

Summary and Next Steps



Generative AI Deployment and Monitoring

databricks