Machine Learning in Production

Conor Murphy
Lead Data Scientist

DATA+AI SUMMIT 2021
FORMERLY SPARK+AI SUMMIT

ORGANIZED BY databricks

#DataAISummit

Introduction

Conor Murphy

- Lead Data Scientist
  - BA, MA Philosophy
  - MSc Neuroscience
- Background

- 4 years grants officer focusing on data-driven humanitarian projects
- 1 year faculty of University New Haven/Galvanize Data Science MS
- 3.5 years @ Databricks
- Started on Spark 1.6
- Focus on productionalizing ML models

Agenda

- Morning (9-11)
  - MLflow Tracking
  - MLflow Models
  - MLflow Model Registry
- Lunch (11-12)
- Afternoon (12-4)
  - 4 Deployment Paradigms
  - CI/CD
  - Monitoring (time permitting)

Housekeeping

- We'll be using...
  - dataaisummit.com for questions
  - Cloud Labs for Databricks access
- Please use the TA's for all your questions
- You have access to the notebooks **until the heat death of the universe** (don't forget to export)
- Cloud Labs will disappear after the course

MLflow

Hardest Part of ML isn't ML, it's Data

"Hidden Technical Debt in Machine Learning Systems," Google NIPS 2015

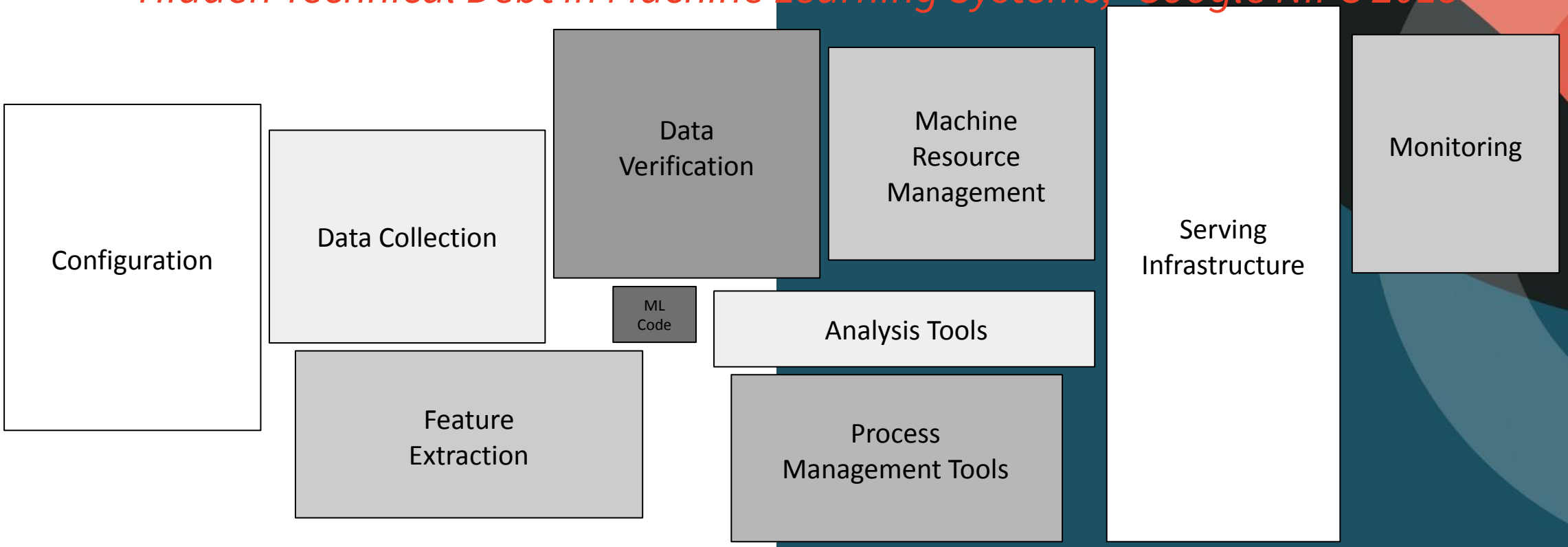
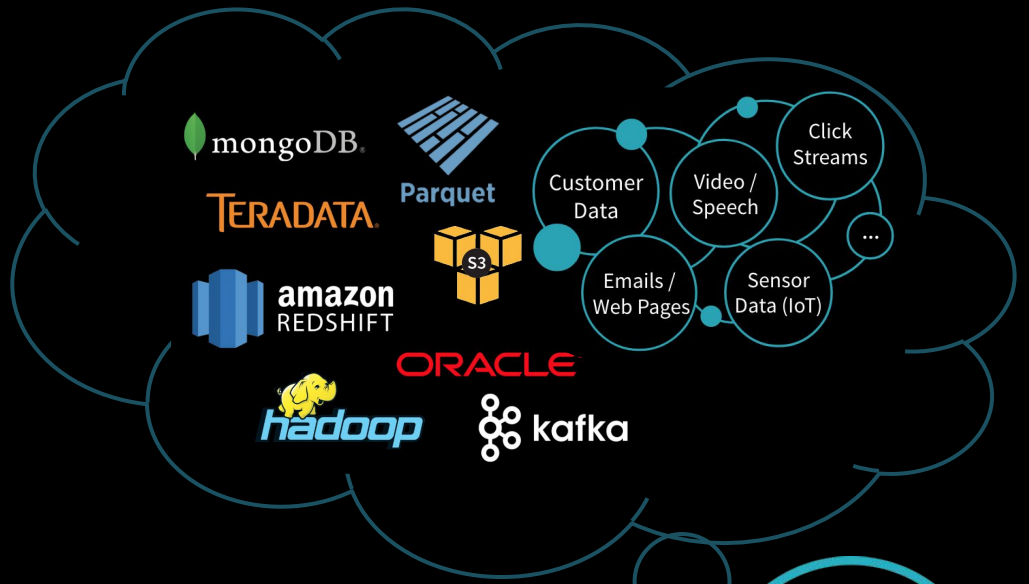


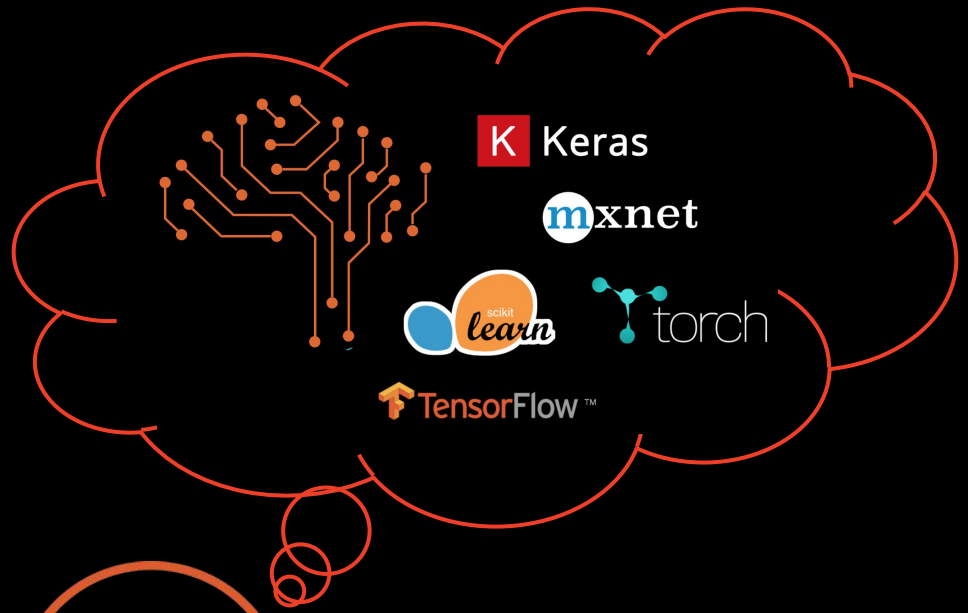
Figure 1: Only a small fraction of real-world ML systems is composed of the ML code, as shown by the small green box in the middle. The required surrounding infrastructure is vast and complex.

Data & ML Tech and People are in Silos



DATA ENGINEERS

X



DATA SCIENTISTS

ML Lifecycle is Manual, Inconsistent and Disconnected

Prep Data

- Low level integrations for Data and ML
- Difficult to track data used for a model

Build Model

- Ad hoc approach to track experiments
- Very hard to reproduce experiments

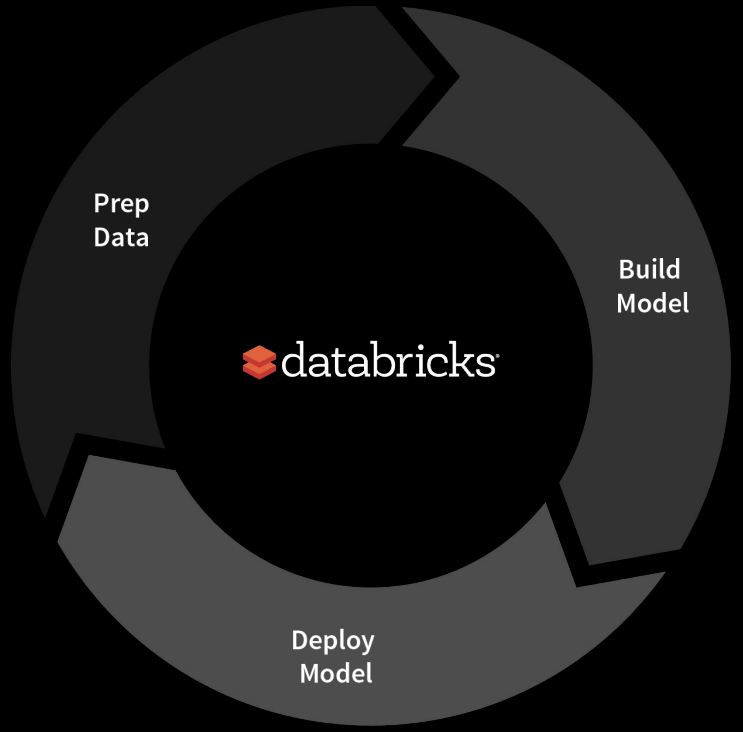
Deploy Model

- Multiple tightly coupled deployment options
- Different monitoring approach for each framework

Amazon SageMaker

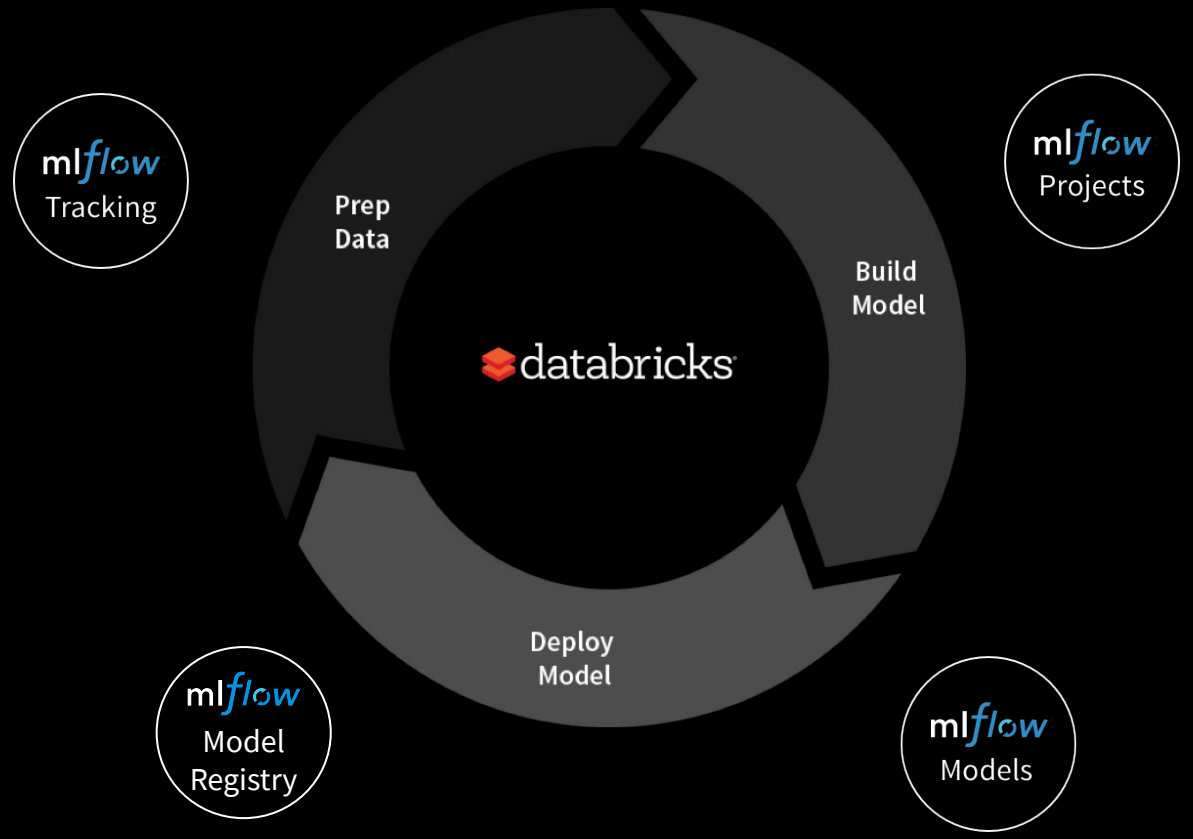
Azure Machine Learning

The need for standardization



Introducing MLflow

Unveiled in June 2018, MLflow is an open source platform designed to manage the complete Machine Learning Lifecycle.



MLflow Components

mlflow Tracking

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

mlflow Projects

Packaging format for reproducible runs on any platform

mlflow Models

General model format that supports diverse deployment tools

mlflow Model Registry

Centralized and collaborative model lifecycle management

databricks.com/mlflow

mlflow.org

github.com/mlflow

twitter.com/MLflow

MLflow Momentum at a glance

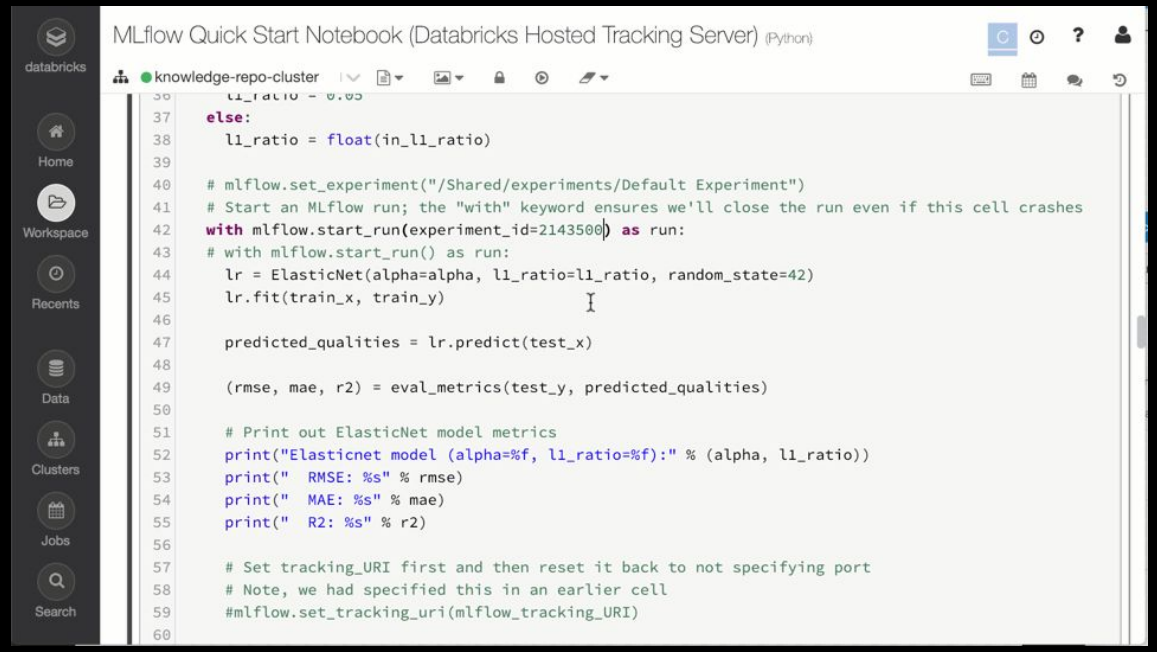
4.7 M
monthly downloads

300
code contributors

40
contributing organizations

Experiment Tracking

Record runs, and keep track of models parameters, results, code, and data from each experiment and in one place.

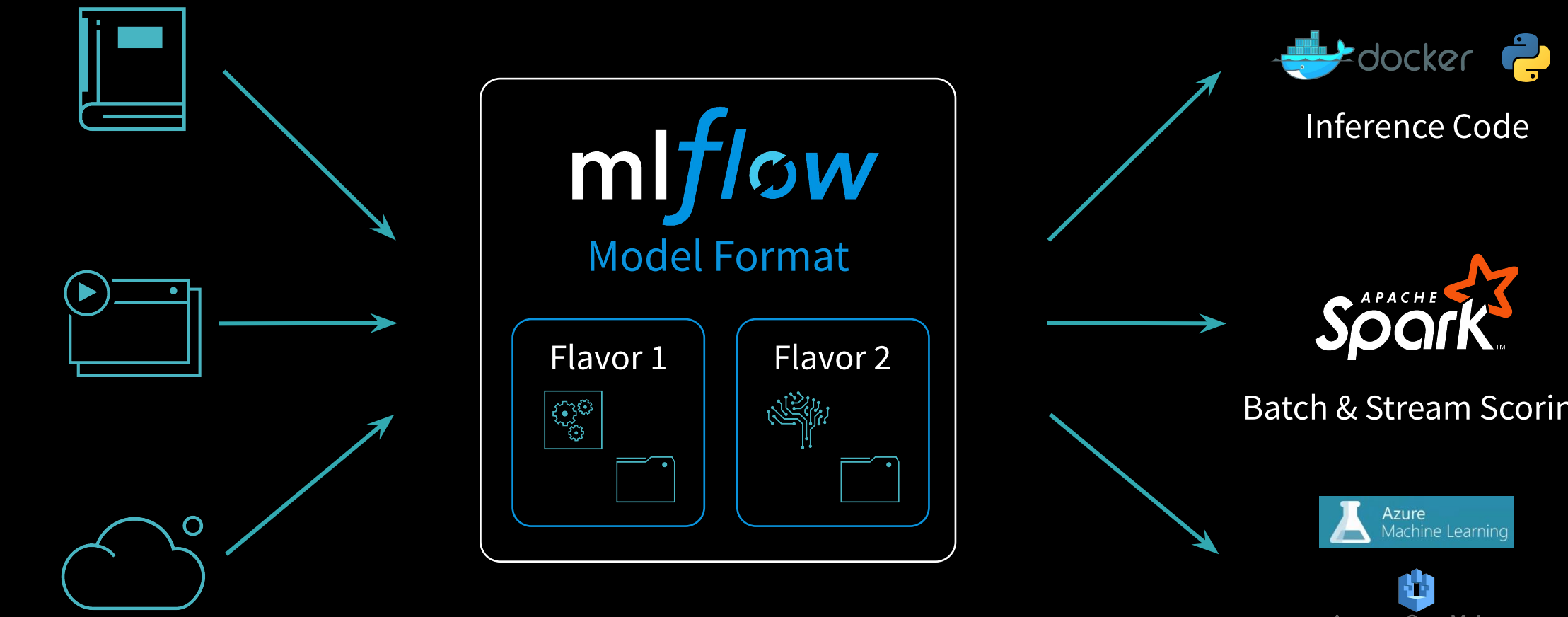


Provides:

- Pre-configured MLflow tracking server
- Databricks Workspace & Notebooks UI integration
- S3, Azure Blob Storage, Google Cloud for artifacts storage
- Experiments management via role based Access Control Lists (ACLs)

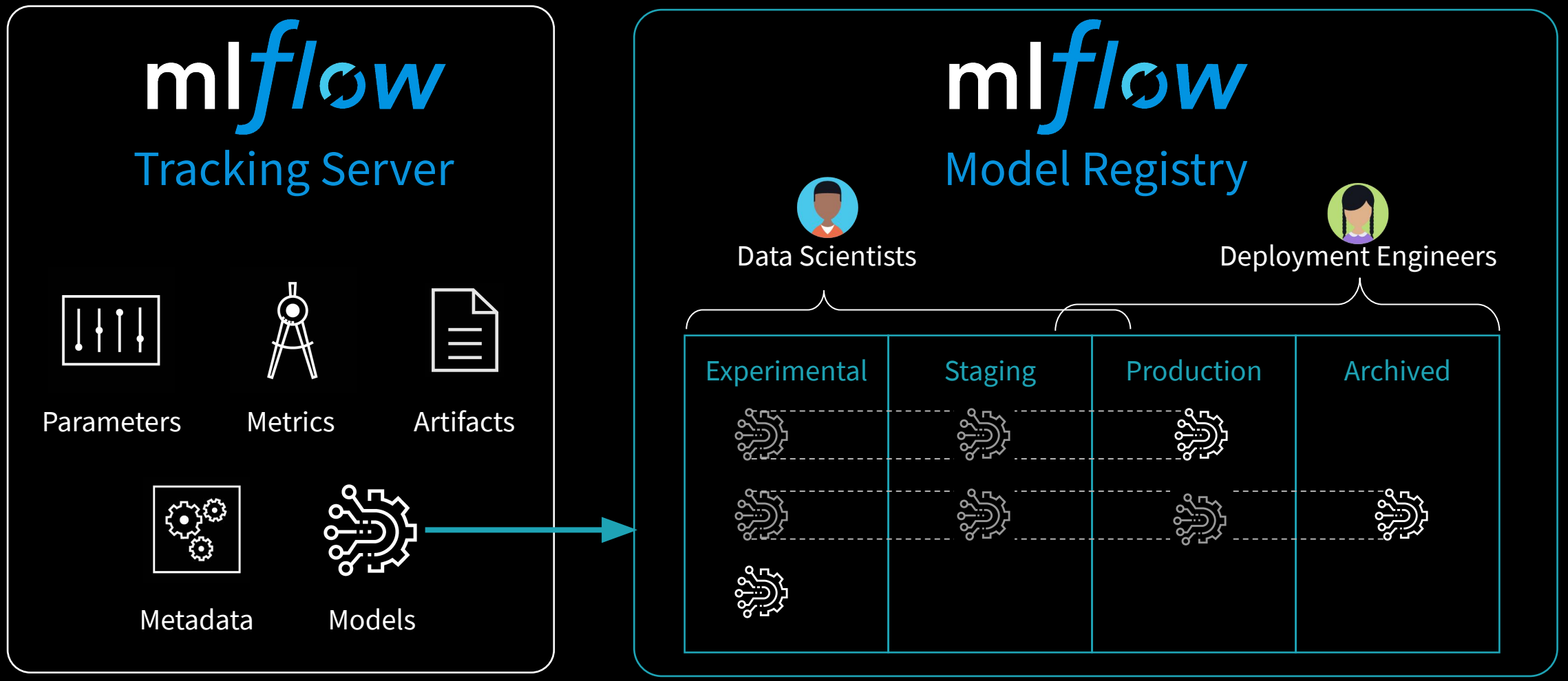
databricks

MLflow Models



mlflow Model Registry

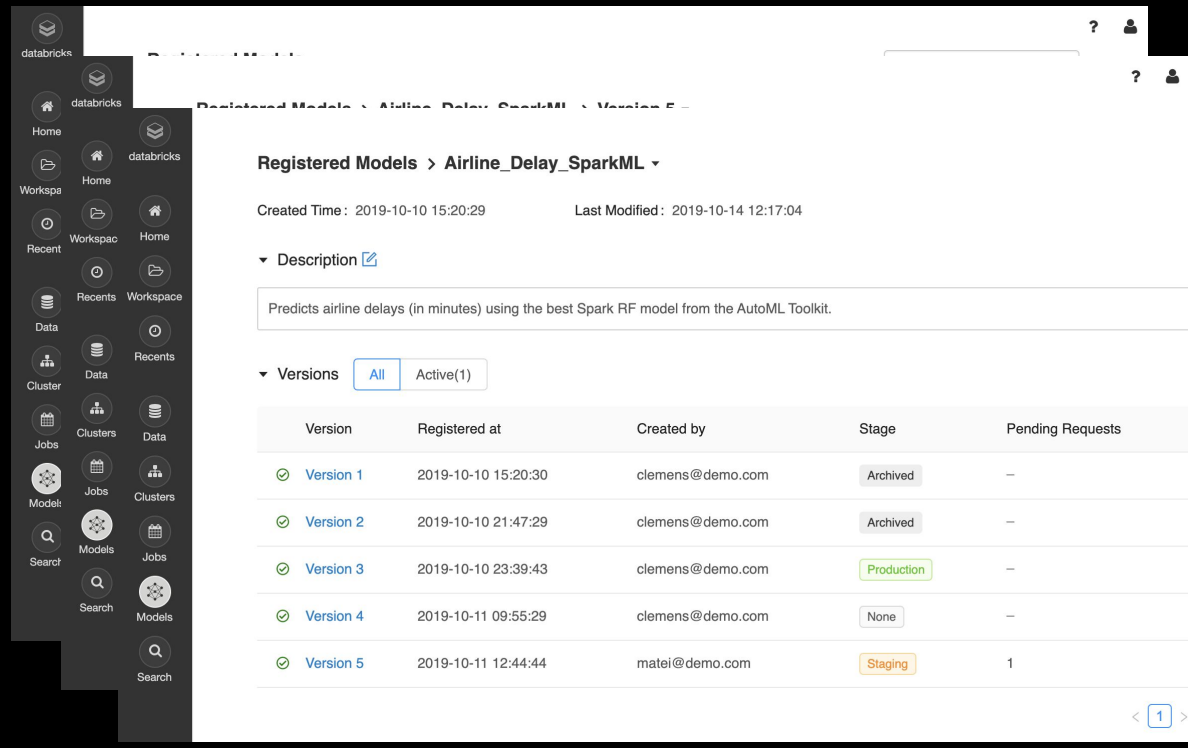
VISION: Centralized and collaborative model lifecycle management



databricks

Model Management

Use one central place to collaboratively manage ML models, from experimentation to online testing and production.



Provides:

- Automatic models versioning
- Integration with governance workflows
- Automatic/manual approval of stage transition
- Full visibility into model lifecycle from experimentation to production

databricks

mlflow Model Registry Benefits

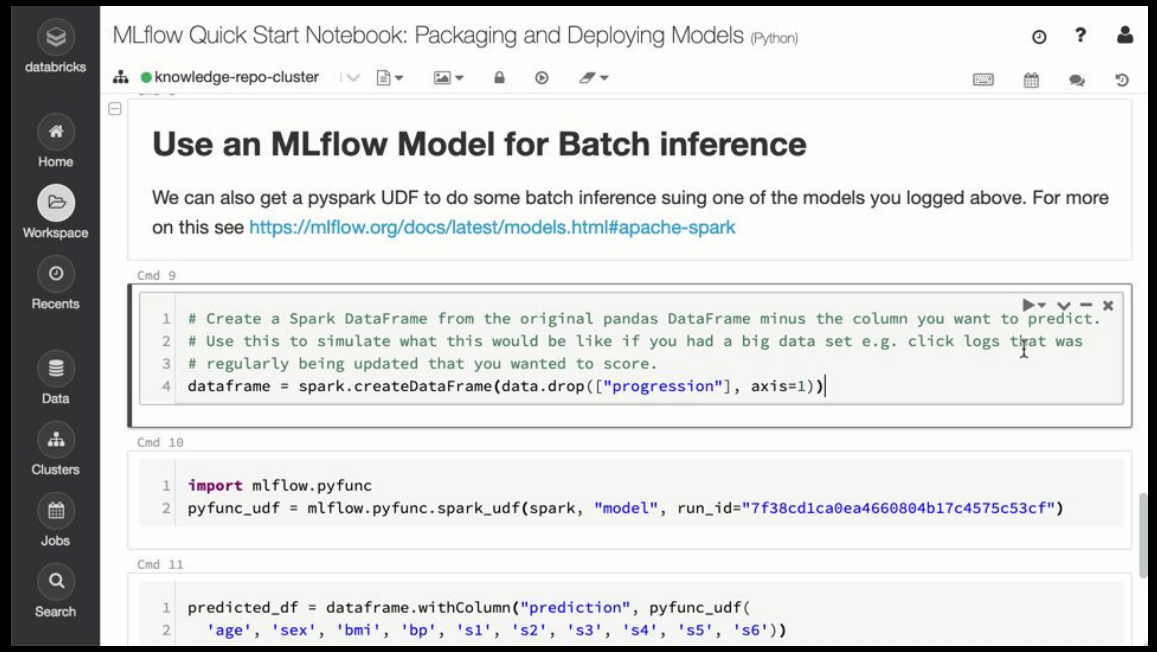
One Collaborative Hub: The Model Registry provides a central hub for making models discoverable, improving **collaboration** and **knowledge sharing** across the organization.

Manage the entire Model Lifecycle (MLOps): The Model Registry provides lifecycle management for models from experimentation to deployment, improving **reliability** and robustness of the model deployment process.

Visibility and Governance: The Model Registry provides full visibility into the deployment stage of all models, who requested and approved changes, allowing for full governance and auditability.

Model Deployment

Quickly deploy models to any platform based on your needs, locally or in the cloud, from experimentation to production.



Supports:

- Databricks Jobs and Clusters for Production Model Operations
- Batch inference on Databricks (Apache Spark)
- REST endpoints via Docker containers, Azure ML, or SageMaker

mlflow Model Deployment Options

In-Line Code

Containers

Batch & Stream Scoring

OSS Inference Solutions

Cloud Inference Services

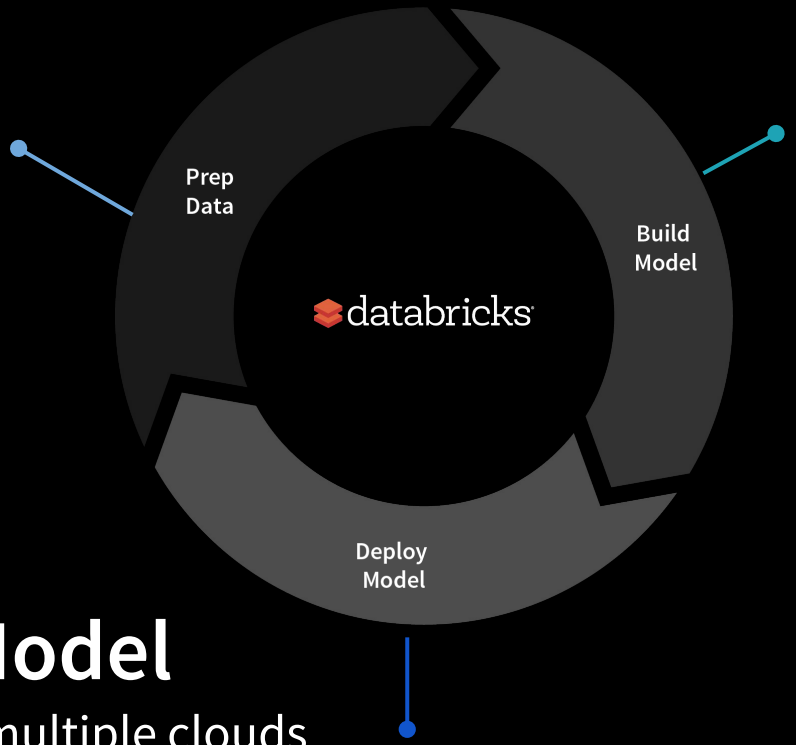
Standardizing the ML Lifecycle on Databricks Unified Analytics Platform

Prep Data

Feed data to Models

Enrich data in experiments

Delta Lake



Build Model

Track Experiments

Reproduce experiments

Machine Learning Runtime

MLflow Project & Tracking

new MLflow Model Registry

Deploy Model

Integrate with multiple clouds

Manage and monitor models

MLflow Models

new MLflow Model Registry

databricks

ML Deployment

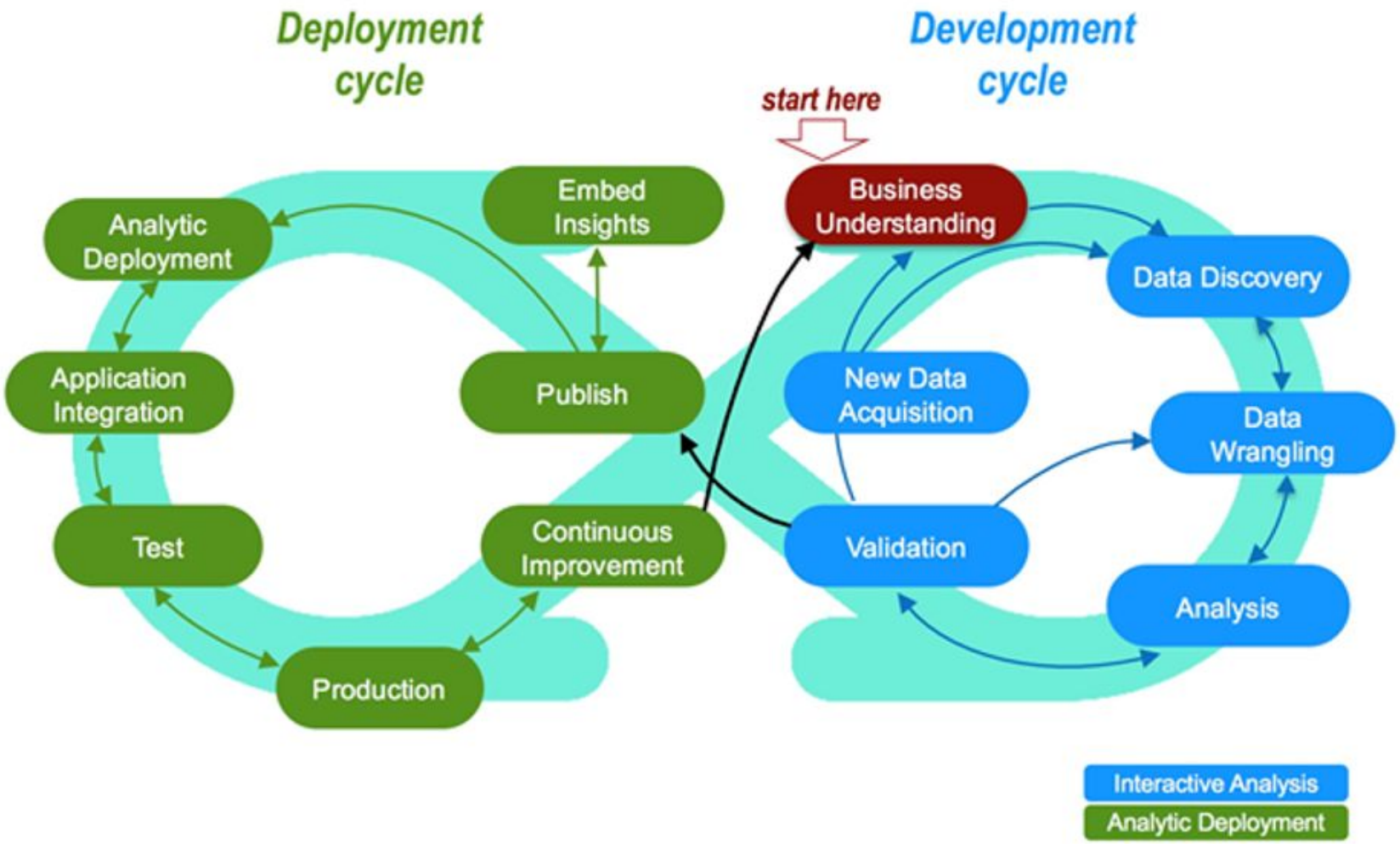
Machine Learning Deployment

- What is ML deployment?
- The Four Deployment Paradigms
- Deployment Requirements
- Deployment Architectures
- Other Issues

What is ML Deployment?

- Data Science != Data Engineering
- Data science is scientific
  - Business problems → data problems
  - Model mathematically
  - Optimize performance
- Data engineers are concerned with
  - Reliability
  - Scalability
  - Maintainability
  - SLAs
  - ...

Closed Loop Systems



databricks

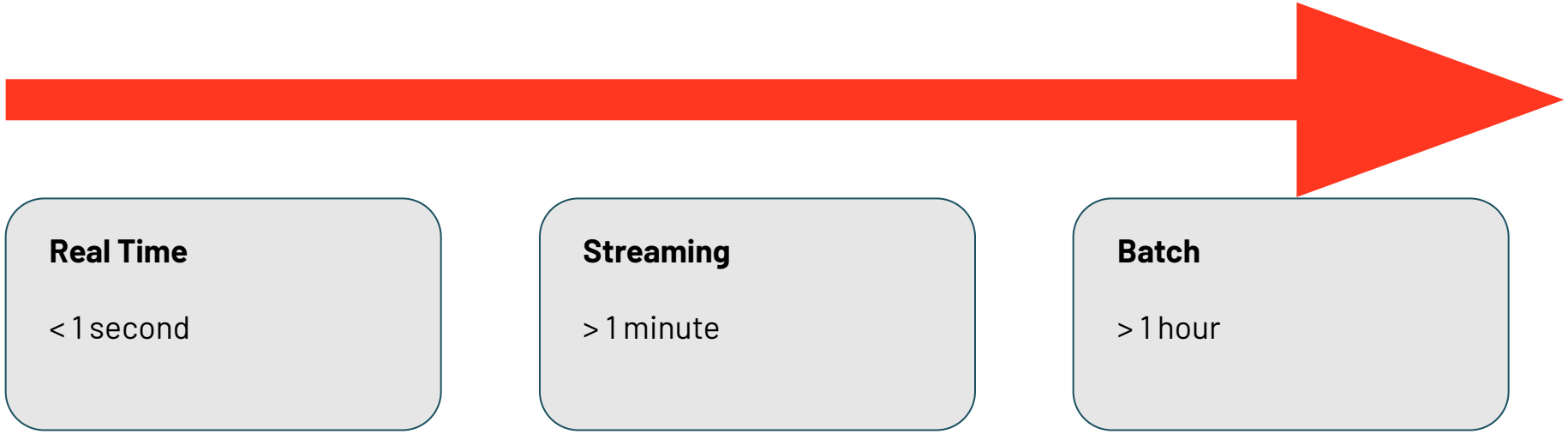
DevOps vs. ModelOps

- DevOps = software development + IT operations
  - Manages deployments
  - CI/CD of features, patches, updates, rollbacks
- ModelOps = data modeling + deployment operations
  - Java, containers, C/C++ and legacy environments
  - Artifact management (Continuous Training)
  - Model performance monitoring (Continuous Monitoring)

The Four Deployment Paradigms

1. **Batch**
   - 80-90% of deployments
   - Leverages databases and object storage
   - Fast retrieval of stored predictions
2. **Streaming (continuous)**
   - 10-15% of deployments
   - Moderately fast scoring on new data
3. **Real Time**
   - 5-10% of deployments
   - Usually using REST (Azure ML, SageMaker, containers)
4. **On-device (edge)**

Latency Requirements (roughly)



Core Requirements

- ML pipeline with featurization logic
- CI/CD pipeline for automation (orchestration, artifacts)
- Monitoring and alerting
- Testing framework (unit + integration + release)
- Version control

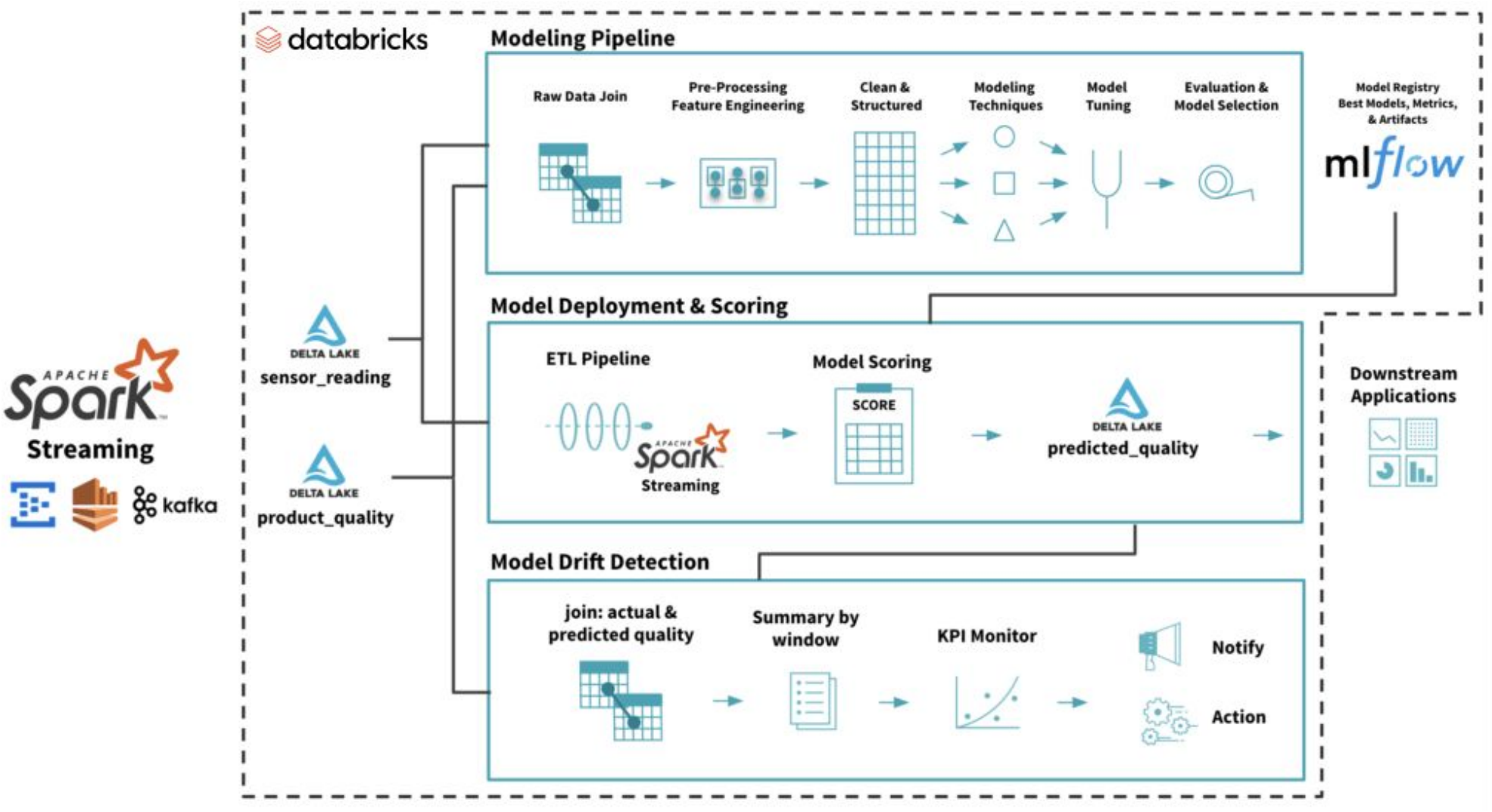
Core+ Requirements

- Model registry
- Data and model drift
- Interpretability
- Reproducibility: data, code, environment, debugging
- Security
- Environment management

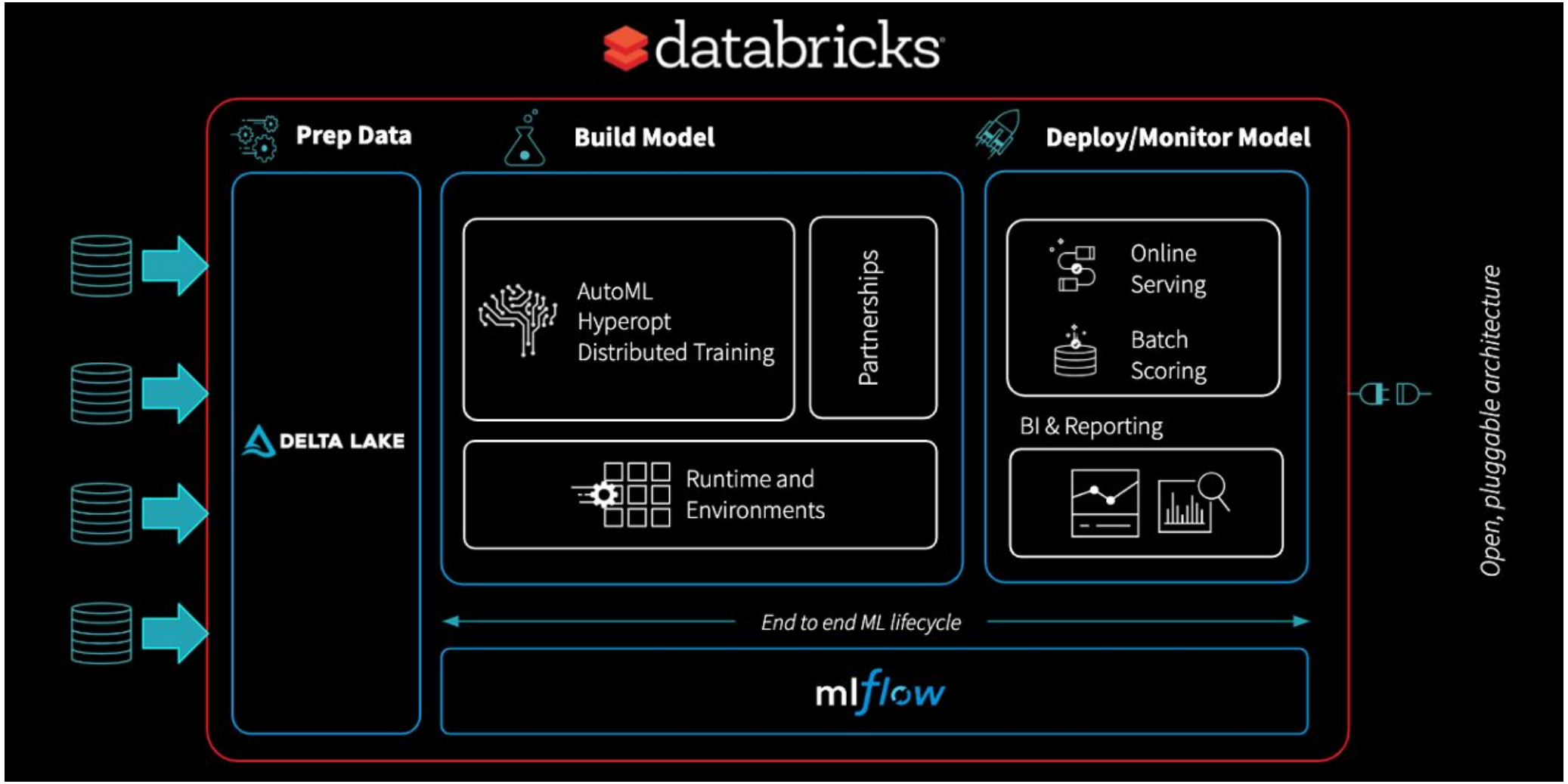
Specialized Requirements

- Feature dictionary
- Cost management
- A/B testing
- Performance optimization

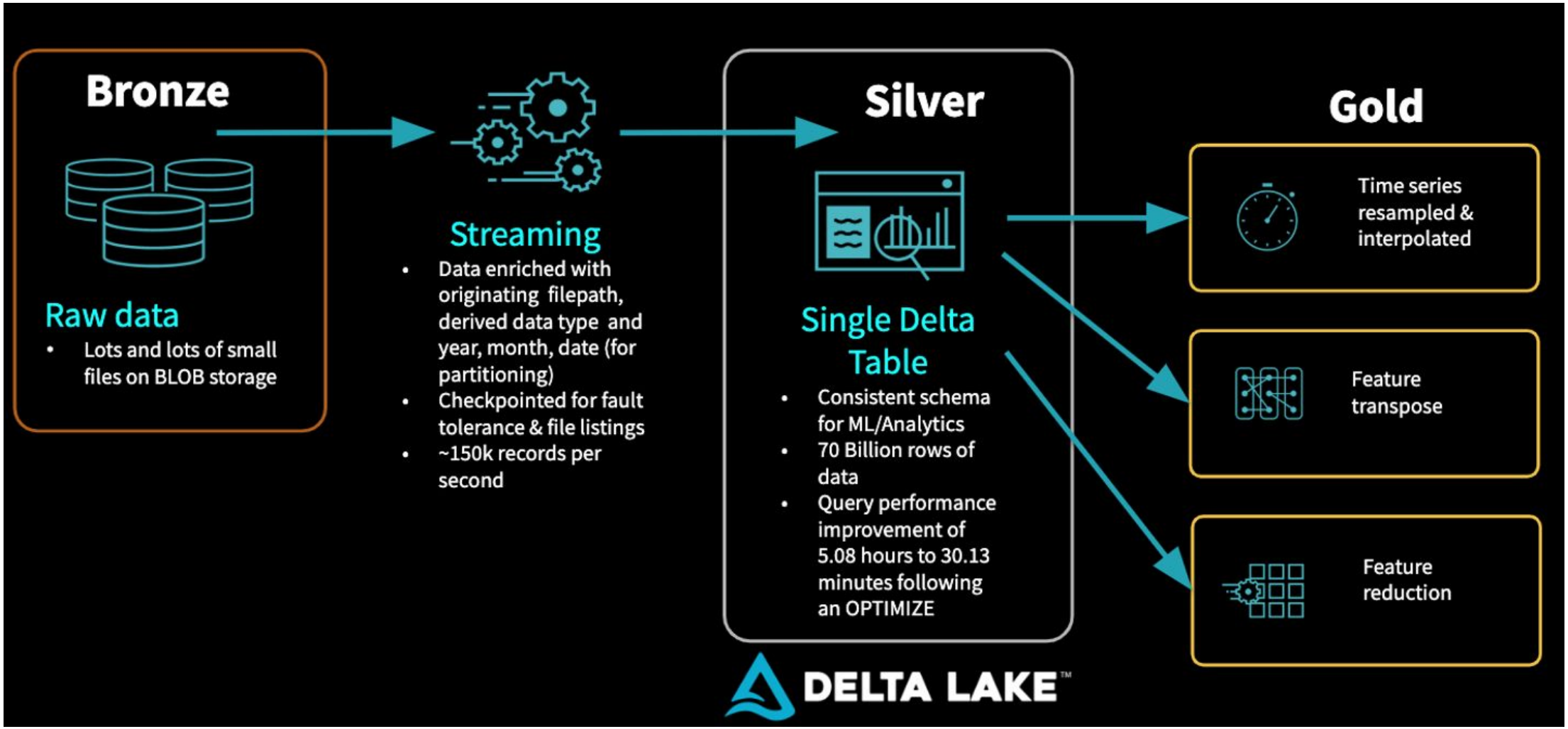
Architecture I



Architecture II



Architecture III



DL Optimizations

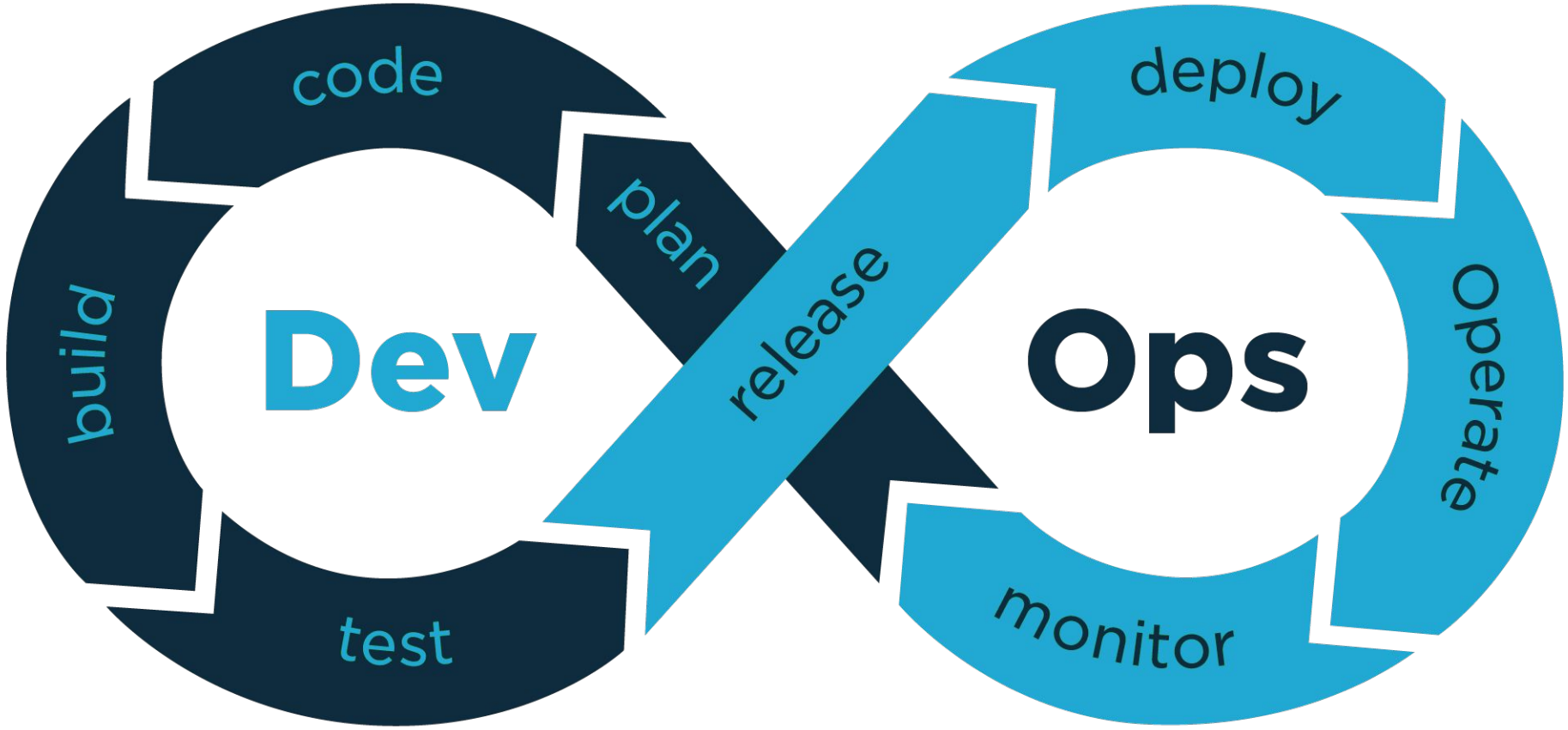
- Quantization: reduce precision of mathematical operations
  - Train normally (e.g. on 64-bit numbers)
  - Reduce to 32- or 16-bit for deployment
  - Generally see 3x improvement
- Weight pruning: reduce size of architecture
- Model topography: retrain using different architectures
  - E.g, compare MobileNet to VGG16

Featurization Logic

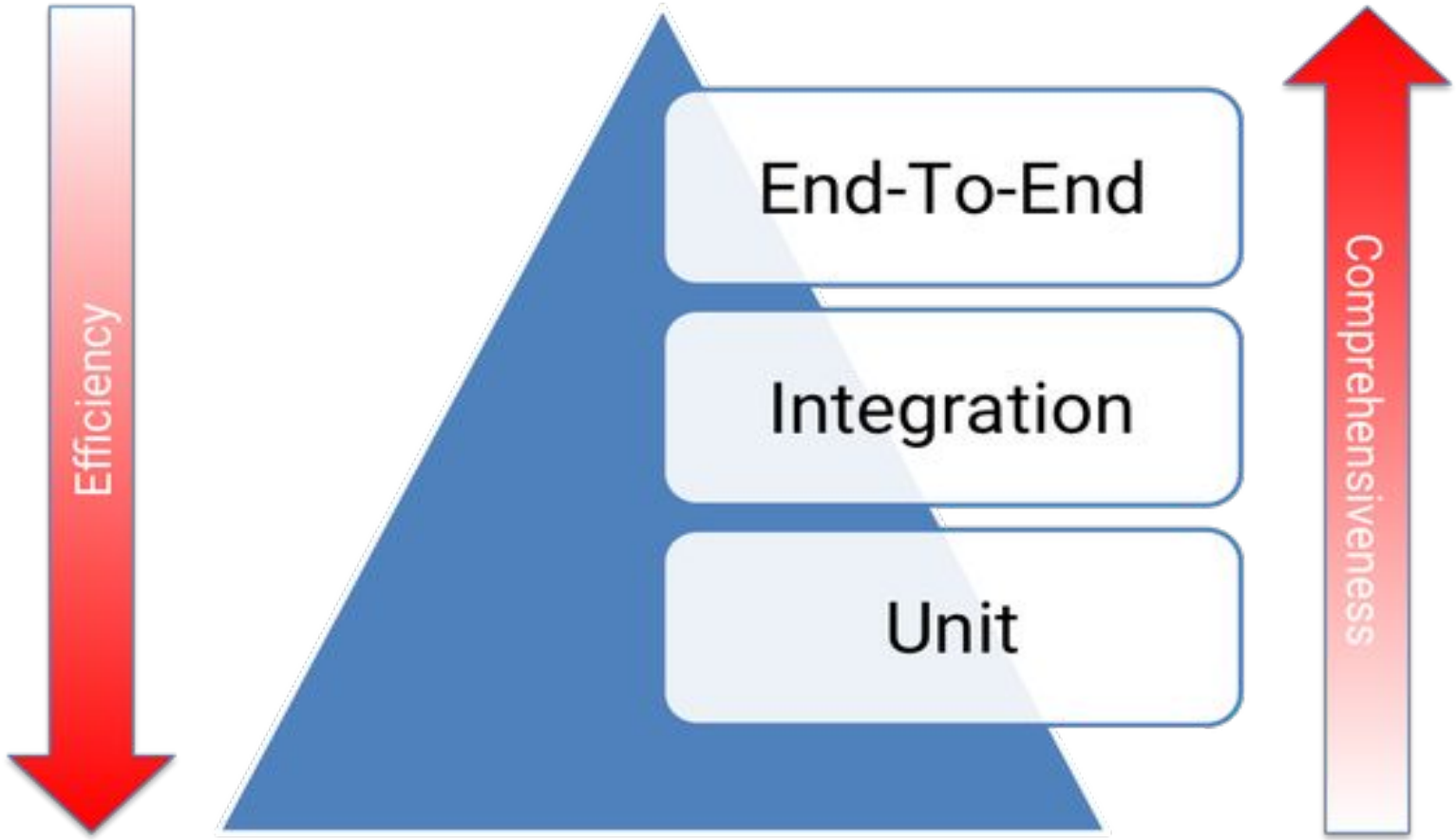
- Apply the same logic to training and scoring data
- Look into MLflow's *pyfunc*
- Confirm production data is available in training
- Consider a Central Model Registry for promoting artifacts to prod

CI/CD

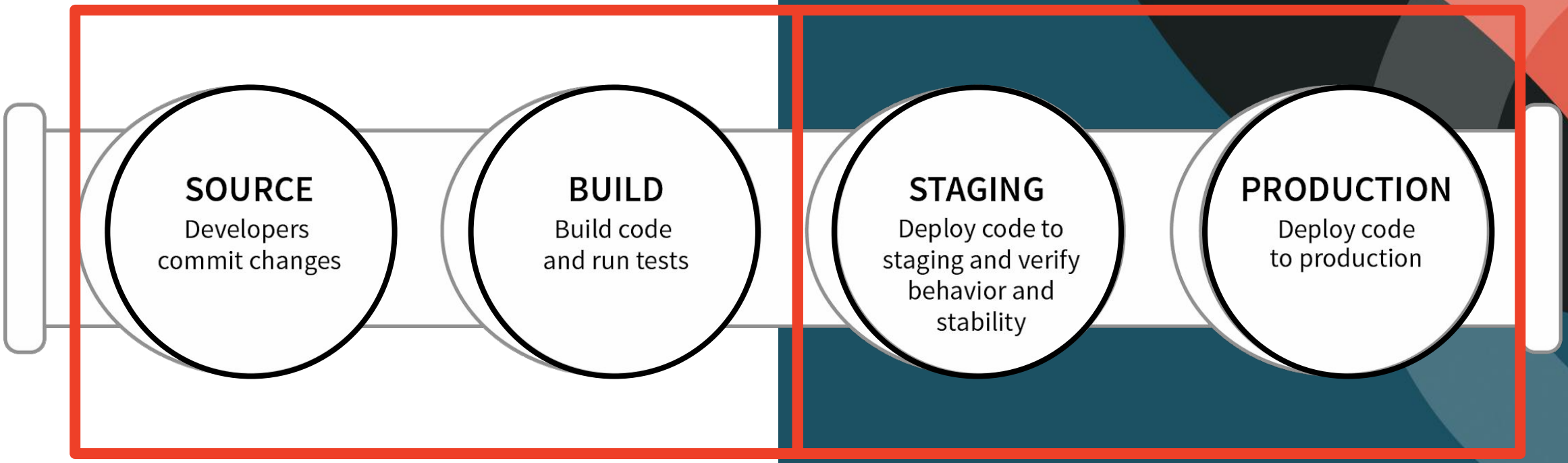
DevOps...


code
plan
build
Dev
test
release
deploy
Ops
operate
monitor

… And the Test Pyramid…



A typical CI/CD pipeline



Continuous Integration (CI)

Continuous Delivery and/or
Continuous Deployment (CD)

Overview of a typical Databricks CI/CD pipeline

Continuous integration

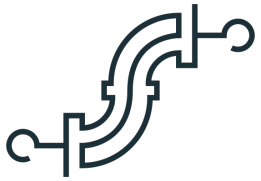
Continuous delivery

Code

Build

Release

Deploy



Test

Operate

Key Benefits of Continuous Integration

- Regressions are captured early by the automated tests
- Building the release is easy as all integration issues have been solved early
- Your QA team spends less time testing and can focus on significant improvements to the quality culture

Key Benefits of Continuous Delivery

- The complexity of deploying software has been taken away
- You can release more often, thus accelerating the feedback loop with your customers
- There is much less pressure on decisions for small changes, hence encouraging iterating faster

CI/CD terminology

- **Build pipeline** - set of steps that build, test (usually only unit tests) & package code or notebooks
- **Release pipeline** - set of steps that performing deployment of the assets
- **Trigger** - event or action that initiates the execution of pipeline. It could be a commit to repository, or explicit execution of the pipeline
- **Asset / Artifact** - deliverable produced by build pipeline, or some other process (model training)
- **Environment** - typically:
  - Development - where development happens
  - Staging - for integration & E2E testing
  - Production - actual work

CI/CD Technologies

	OSS Standard	Databricks	AWS	Azure	Third Party
Orchestration	Airflow, Jenkins	Jobs, notebook workflows	CodePipeline, CodeBuild, CodeDeploy	DevOps, Data Factor	
Git Hooks		MLflow Webhooks			Github Actions, Gitlab, Travis CI
Artifact Management	PyPi, Maven	MLflow Model Registry			Nexus
Environment Management	Docker, Kubernetes, Conda, pyenv		Elastic Container Repository	Container Registry	DockerHub
Testing	pytest				Sonar
Alerting		Jobs	CloudWatch	Monitor	PagerDuty, Slack integrations

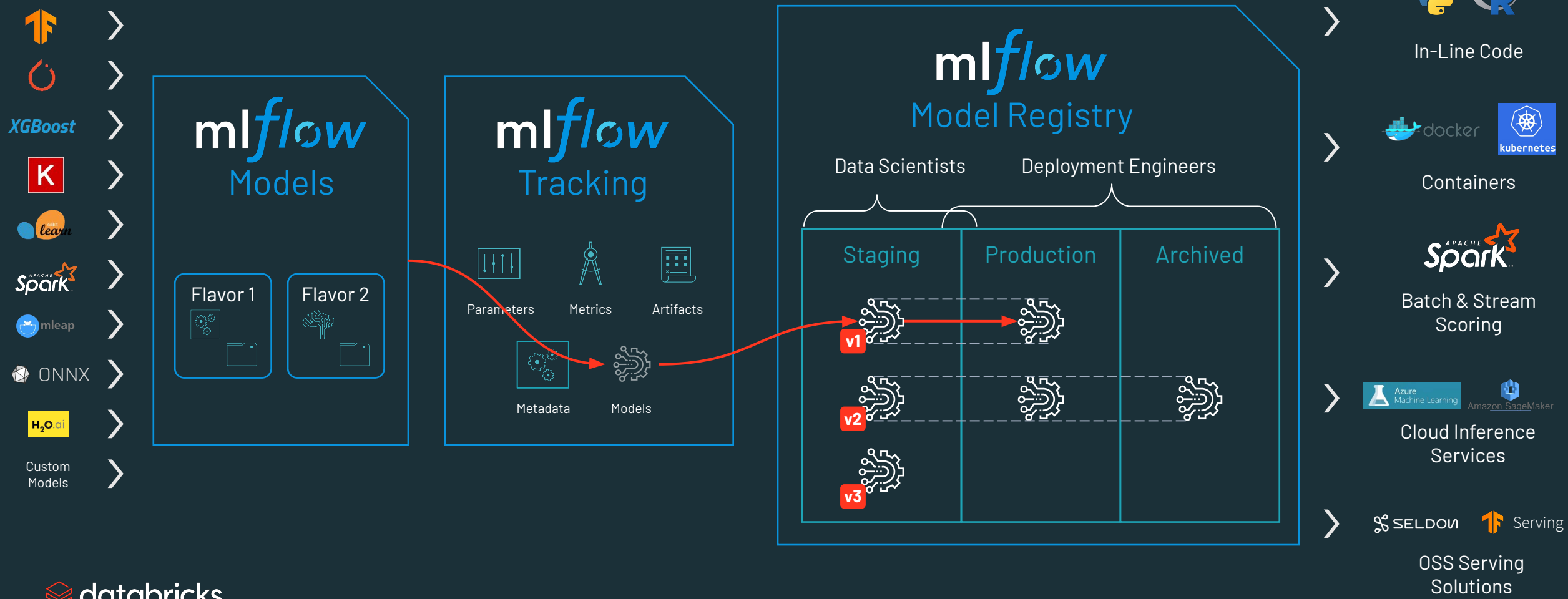
Recommended books

- Designing Data-Intensive Applications by Martin Kleppmann
- Clean Code: A Handbook of Agile Software Craftsmanship by Robert C. Martin
- Refactoring: Improving the Design of Existing Code by Martin Fowler, Kent Beck, Don Roberts
- Test-Driven Development: By Example by Kent Beck
- Code Complete, 2nd ed. by Steve McConnell
- The Pragmatic Programmer: From Journeyman to Master by Andy Hunt, Dave Thomas
- ...

MLOps: Continuous ...

- Continuous Integration (CI) extends the testing and validating code and components by adding testing and validating data and models
- Continuous Delivery (CD) concerns with delivery of an ML training pipeline that automatically deploys another the ML model prediction service.
- Continuous Training (CT) automatically retrains ML models for redeployment
- Continuous Monitoring (CM) concerns with monitoring production data and model performance metrics, which are bound to business metrics

mlflow Model Lifecycle



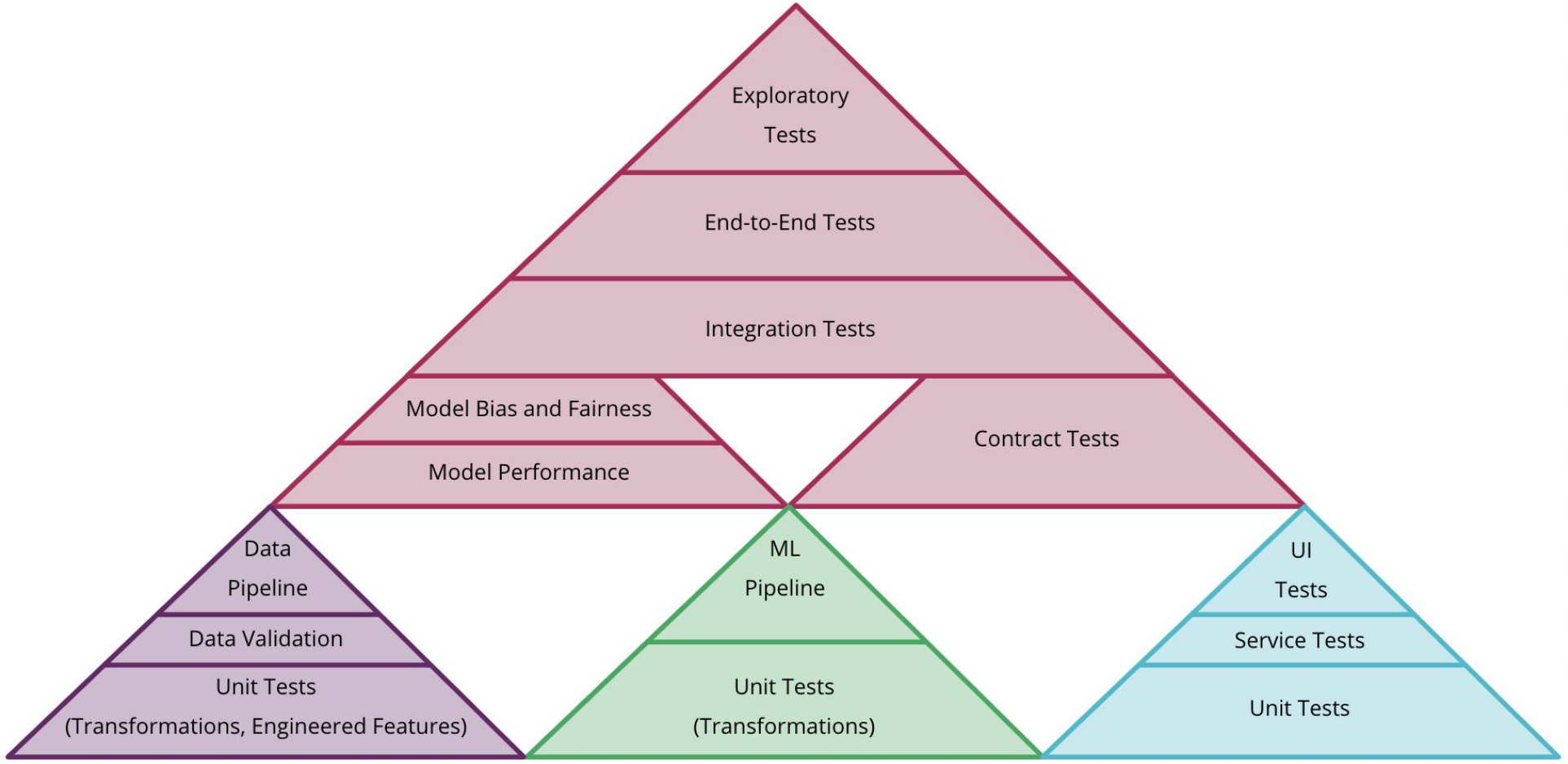
databricks

But Big Data & AI Present new challenges...

3 changeable ingredients go into training an ML model...

Ingredient	Frequency of change	Well defined process/tools for managing change?
Code	~Daily?	Yes!
Configuration	~Daily/Weekly?	Emerging
Data	~Every second?	No? Delta!

And these make for one complicated test pyramid...



Tools to support DevOps & MLOps

- We need tools to work with DBFS, Notebooks, clusters, jobs, ... - import & export objects, etc.:
  - Databricks REST API (too cumbersome to use)
  - Databricks CLI (interface to the REST API)
  - Databricks Terraform Provider (create reproducible environments)
- Databricks Connect - executing code on the Databricks cluster(s) from the local machine/IDE
- MLflow API
- Cloud specific tools & SDKs - azure-cli, etc.

Databricks CLI

Databricks CLI

- Installation:
  - pip install databricks-cli
- Configuration
  - Requires personal access token (PAT)
  - Execute: databricks configure <- it will generate config file on the disk
  - Or use environment variables to dynamically change configuration (handy for CI/CD pipelines, etc.)
- It's possible to have multiple profiles in the configuration file & select them via command-line
  - databricks --profile <profile-name> command ...

Working with objects in workspace: notebooks, ...

- databricks workspace <subcommand>
  - export & export_dir - export a file or directory from workspace to the local disk
  - import & import_dir - import a file or directory
  - mkdirs - create a hierarchy of directories in the workspace
  - ls - lists objects in given directory, or get information about specific file
- Copy directory between workspaces (be careful as it will overwrite files!)
  - databricks --profile ws1 workspace export_dir -o '/Users/...' local-dir && databricks --profile ws2 workspace import_dir -o local-dir "/…."

Working with files on DBFS

- databricks fs <subcommand>
  - ls, mkdirs, cp, mv, rm, cat
- We can use it to publish artifacts – jars, wheels, eggs, ...
  - databricks fs cp --overwrite target/scala-2.11/some.jar 'dbfs:/FileStore/jars/'

Working with clusters & jobs

- databricks clusters <subcommand>
  - create, delete, edit, list, get, resize, start, restart, …
- databricks jobs <subcommand>
  - list, get, create, delete, reset, run-now
- databricks runs <subcommand>
  - list, get, submit, get-output
- databricks libraries <subcommand>
  - install, uninstall, list, …
-
- Although, for that it’s could be easier to use Terraform

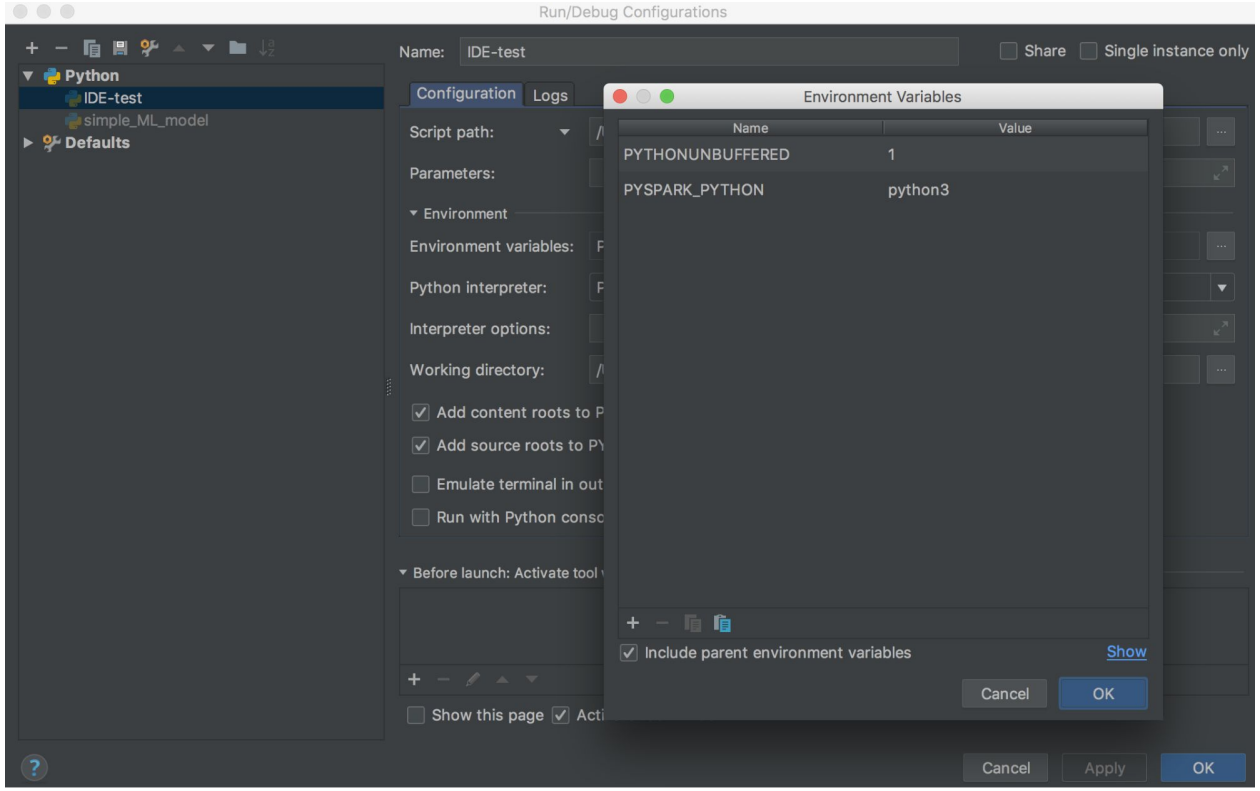
Other objects

- Users & groups
- Secrets
- Cluster policies
- Instance pools
- Tokens
- Stacks:
  - Allow to download or upload multiple resources (jobs, notebooks, libraries, ...) at once using the JSON-encoded description

Databricks Connect

Databricks Connect

- Run Spark jobs on Databricks from your favorite IDE or notebook
- Use with Scala, Python, R, Java
- Step through and debug code
- Shut down idle clusters
- Use VCS integration features
- Faster iteration



databricks

Installation & configuration

- Installation
  - use virtual Python/Conda environments for different versions
  - pip install databricks-connect==<dbr-version>
- On Databricks side, cluster should be started with configuration:
  - spark.databricks.service.server.enabled true
- Configuration
  - databricks-connect configure
  - We need to have separate configurations for different clusters/workspaces
  - It's also possible to specify configuration options via environment variables & Spark configuration
- Instructions for setting up specific IDE / Notebook implementation

Usage

- Just use pyspark, spark-shell, sparkR, and spark-sql from the installed package for interactive work
- Use spark-submit to execute jar, Python script, etc.
- In the code:
  - Python: spark = SparkSession.builder.getOrCreate()
  - Scala: val spark = SparkSession.builder().getOrCreate()
  - …
- Set Hadoop & Spark configuration programmatically, if it's necessary
- You can access DBFS via Hadoop's FileSystem API
- Execute code directly from IDE:
  - Simple for Python - just run script
  - Slightly more complex for Scala/Java (see instructions here)

Limitations & caveats

- No support for Spark Structured Streaming yet
- Allows to work only with one workspace/cluster at the time
- Some of the dbutils are not supported (widgets, notebooks, ...)
- No support for native Scala, Python, and R APIs for Delta table operations
- Versions of the Python & its libraries (and other languages) need to match between the local environment (driver) & executors running on Databricks
  - If doesn't match, could lead to the strange errors that doesn't happen when running code on the Databricks only
  - You can use following script to perform the libraries version check

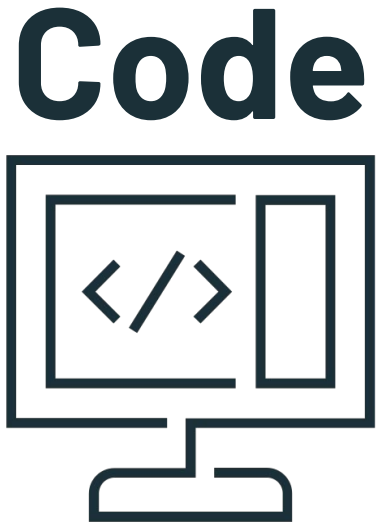
Developing code for Databricks

databricks

Software Engineering with Databricks

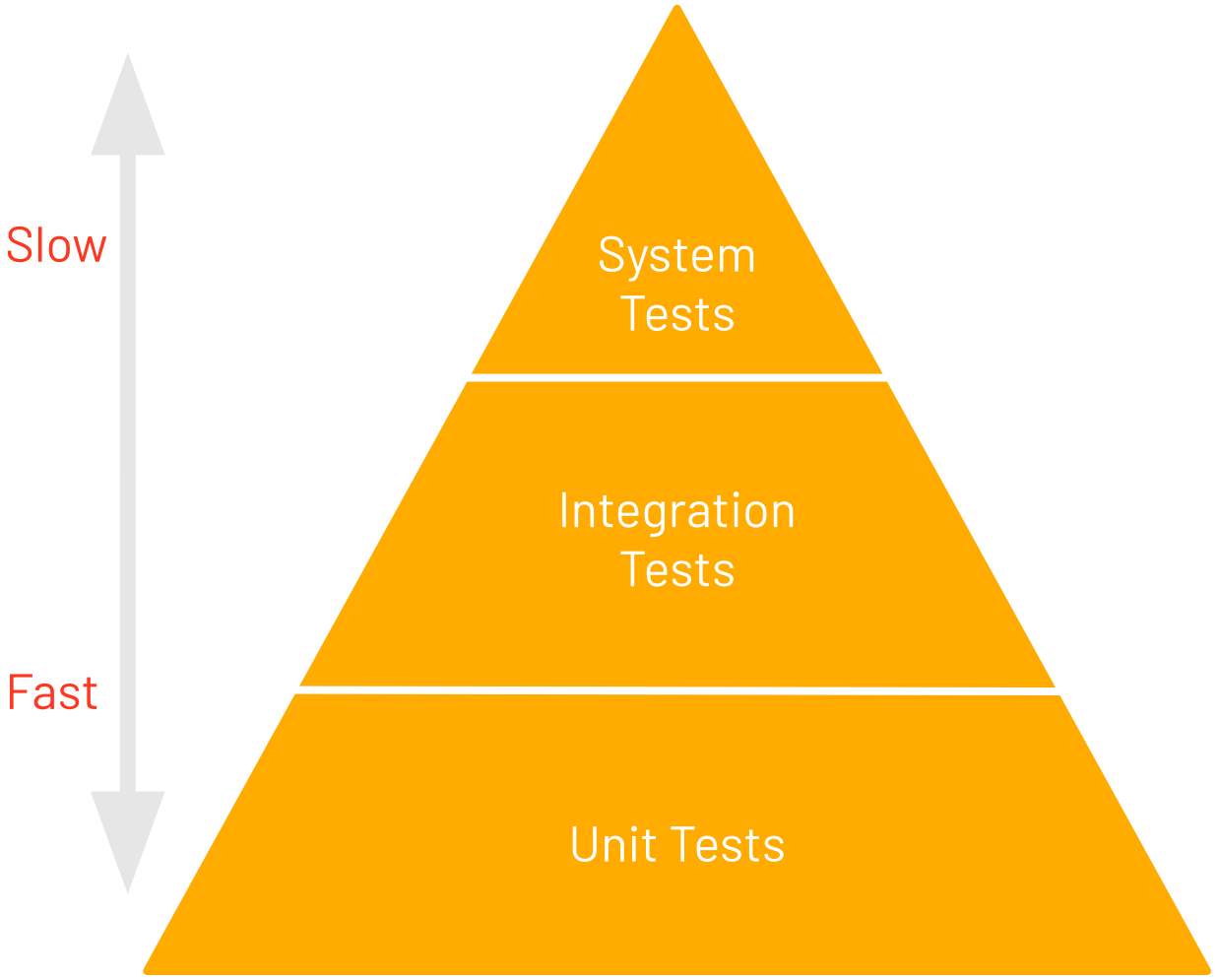
- Develop code and unit tests in a Databricks notebook or using an IDE
- Run tests locally
- Commit code and tests to a git branch
- Get code compiled, tested & deployed via CI/CD pipeline

- Upcoming Workspaces 2.0 will significantly improve the workflow with Repos!



Developing testable Databricks code

CI/CD for Databricks: Testing rationale



- Functional Tests
- Integration with other systems

- Spark Notebook / Job tests

- Core business logic / UDFs (dataframe in, dataframe out)

Spark-specific requirements

- Support for Spark-specific APIs: RDD, Dataframe/Dataset, Streaming, ML, ...
- Specialized assertions:
  - Compare Dataframe/RDD content:
    - As whole, or individual columns
    - Ordered / not ordered
    - Approximate (floating point numbers)
    - Ignoring nulls or not
  - Compare Dataframe schema
- Support testing of UDFs
- Effective data generation
- Emulation of additional components, like, HDFS, Hive, Kafka, ...
- Specifically for Databricks - support for testing of code in notebooks

Writing the testable code

- Split code into testable chunks
  - Individual functions: DataFrame/RDD/Configuration in, DataFrame/RDD out
  - Business logic - functions, calling other functions
  - No dependency onto the global state/external systems
- 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

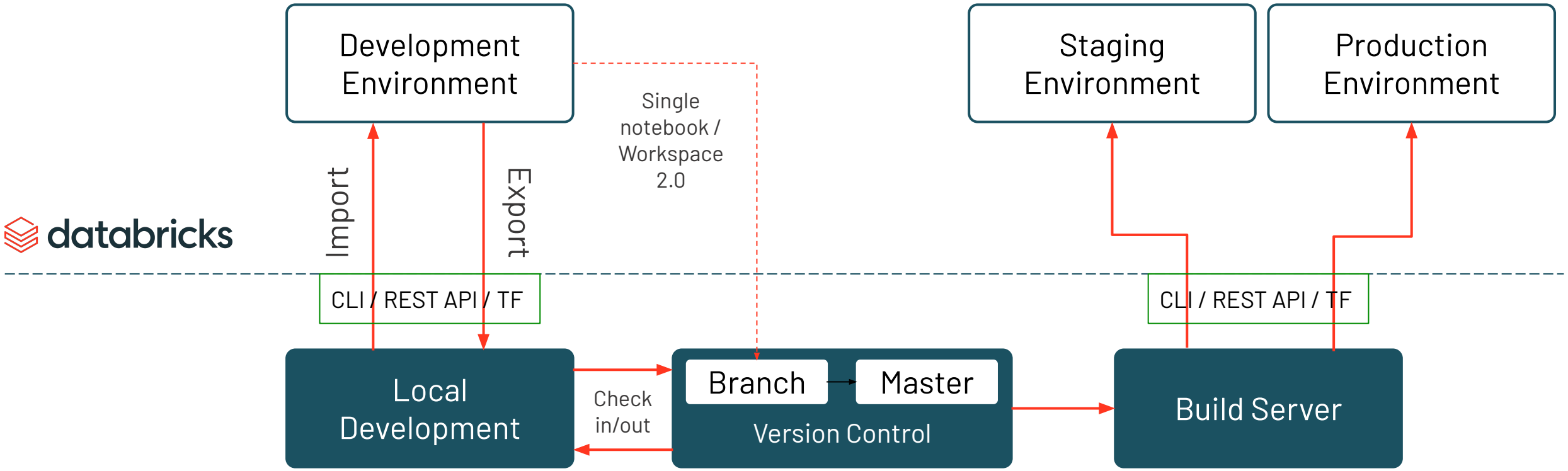
Development workflows

Development workflows

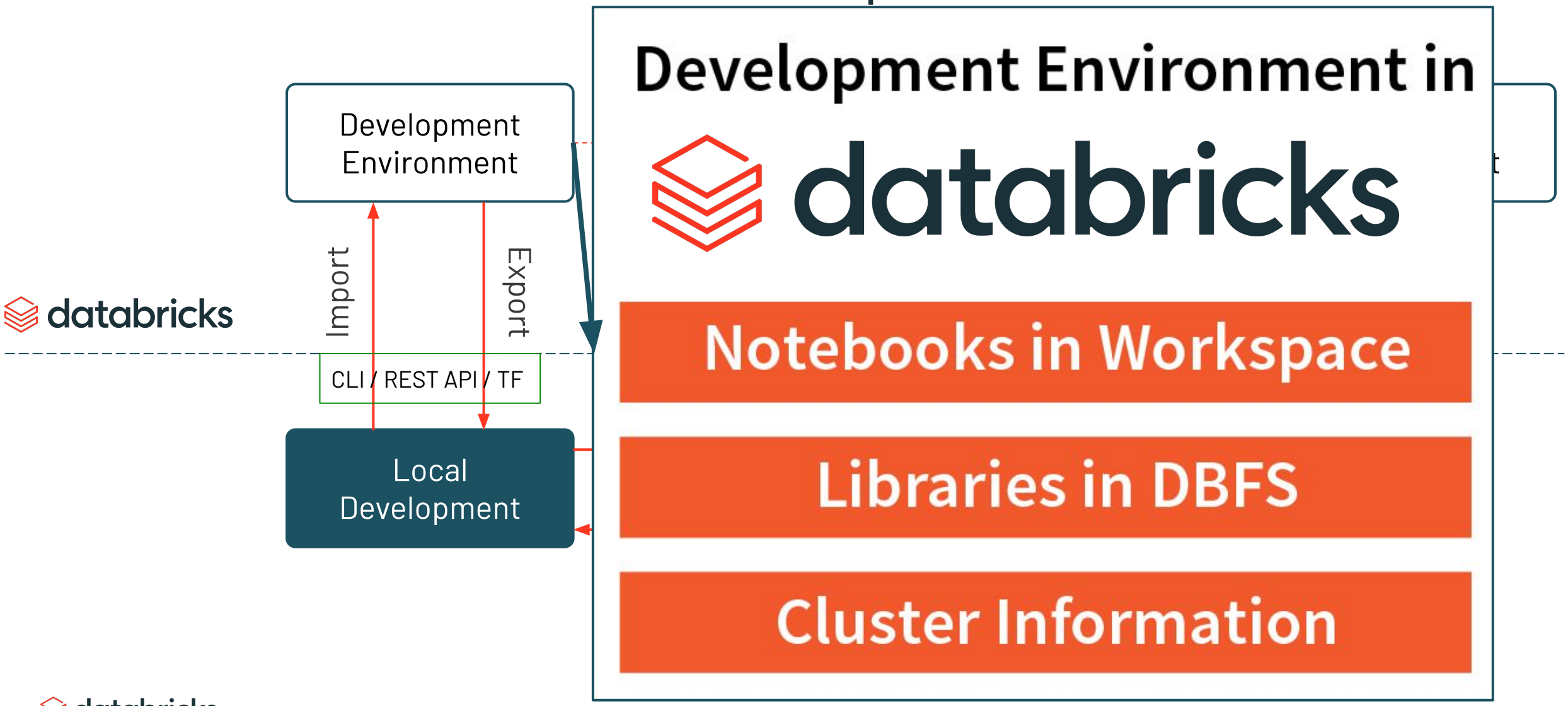
- Notebooks only:
  - Faster experimentation / feedback
  - Harder to automate, challenges with testing, chaining jobs
- Code only:
  - Develop Python / Scala code in IDE, build, execute as jobs
  - Better automation, more tooling, delayed feedback loop
- Mixed (typical usage by our customers):
  - Code developed in IDE, packaged & deployed as libraries
  - Notebooks are used for configuration & orchestration of calls to libraries

CI/CD for Databricks & Notebooks

CI/CD for Notebooks: Overview



CI/CD for Notebooks: Development



Code organization in notebooks

- Put related code into individual notebooks, organize in the testable modules
- Build the "top-level" notebook that calls the individual notebooks via **%run** and/or notebook workflows (dbutils.notebook.* functions)
- Top-level notebook contains configuration parameters and maybe the business logic
- With dbutils.notebook.* functions it's possible to create very complex workflows, passing & returning values, branching based on the return values, retries, exception handling, ...
  - But there are limitations & complexities

Data Functions

Cmd 1

```
1  dbutils.widgets.text("r_number", "42", "Random number")
```

Command took 0.02 seconds -- by alexey.ott@databricks.com at 02/10/2020, 10:59:00 on Shared Autoscaling EMEA

Cmd 2

```
1  r_number = int(dbutils.widgets.get("r_number"))
2
3  if r_number == 42:
4    returnValue = "Answer to the Ultimate Question of Life, the Universe, and Everything"
5  elif r_number > 50:
6    returnValue = "Quite big value"
7  else:
8    returnValue = "Some other value"
9
10 dbutils.notebook.exit(returnValue)
```

MainJob

Cmd 1

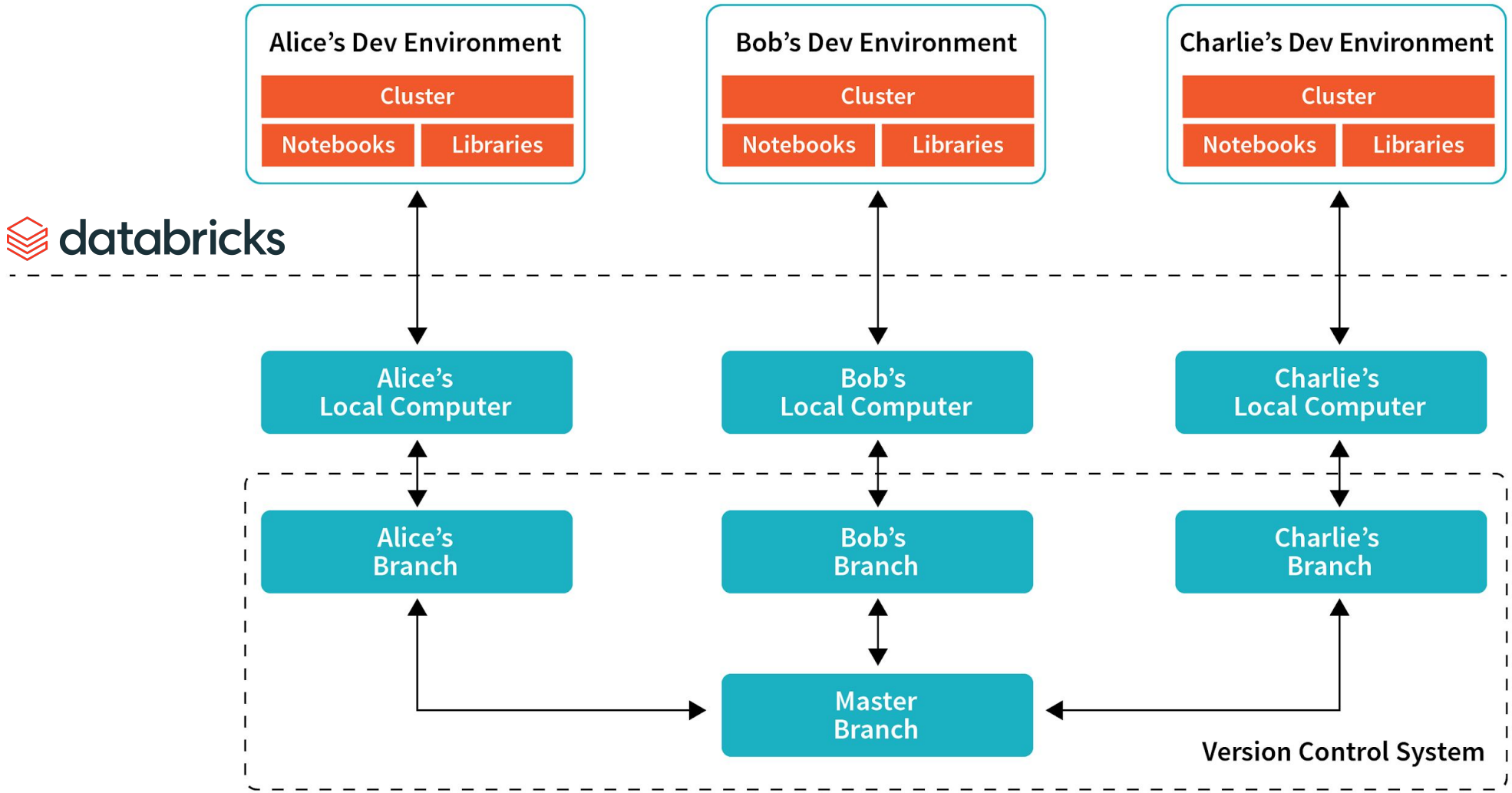
```
1  import random
2
3  for i in range(10):
4    try:
5      if i == 0:
6        r_number = "42"
7      elif random.random() > 0.5:
8        r_number = "not a number!"
9      else:
10       r_number = str(random.randint(0, 100))
11
12     retValue = dbutils.notebook.run("./Data Functions", 60, {"r_number": r_number})
13     print(f"At step {i} for value '{r_number}' we've got: {retValue}")
14   except Exception as e:
15     print(f"We've got the error: {e}")
```

Cancel Running command...

Notebook job #38405
Notebook job #38406
Notebook job #38407
Notebook job #38408
Notebook job #38409

```
At step 0 for value '42' we've got: Answer to the Ultimate Question of Life, the Universe, and Everything
At step 1 for value '87' we've got: Quite big value
We've got the error: An error occurred while calling o619._run.
: com.databricks.WorkflowException: com.databricks.NotebookExecutionException: FAILED
        at com.databricks.workflow.WorkflowDriver.run(WorkflowDriver.scala:75)
```

CI/CD for Notebooks: Development



Version Control with Databricks Notebooks

1. Link Git to your workspace: Azure DevOps, GitHub, BitBucket Cloud, arbitrary Git server (coming in near future)
2. Link Notebook to the Git repository - it's recommended to use separate branch for every notebook during development
3. Commit the updated notebooks to the local branch
4. Check the test results (maybe only unit tests)
5. Merge the notebook into "main" branch
6. Push the changes to the remote branch to trigger execution of test suite

Local Version Control with Databricks Notebooks

1. Check out the desired branch - it's recommended to use separate branch for every notebook during development
2. Pull new changes from the remote branch
3. Export notebooks from the Databricks workspace using the Databricks CLI
4. Commit the updated notebooks to the local branch
5. Push the changes to the remote branch
6. Wait until changes are picked up by CI/CD pipeline

1. (Create and) checkout a branch for development.

```
git checkout -b 'some-dir-0.0.1-dev'
```

2. Download your work from your Databricks Workspace

```
databricks workspace export_dir \
  -o /Users/some.user@email.com/some-dir-0.0.1-dev .
```

3. Do a standard git commit of the work, e.g.

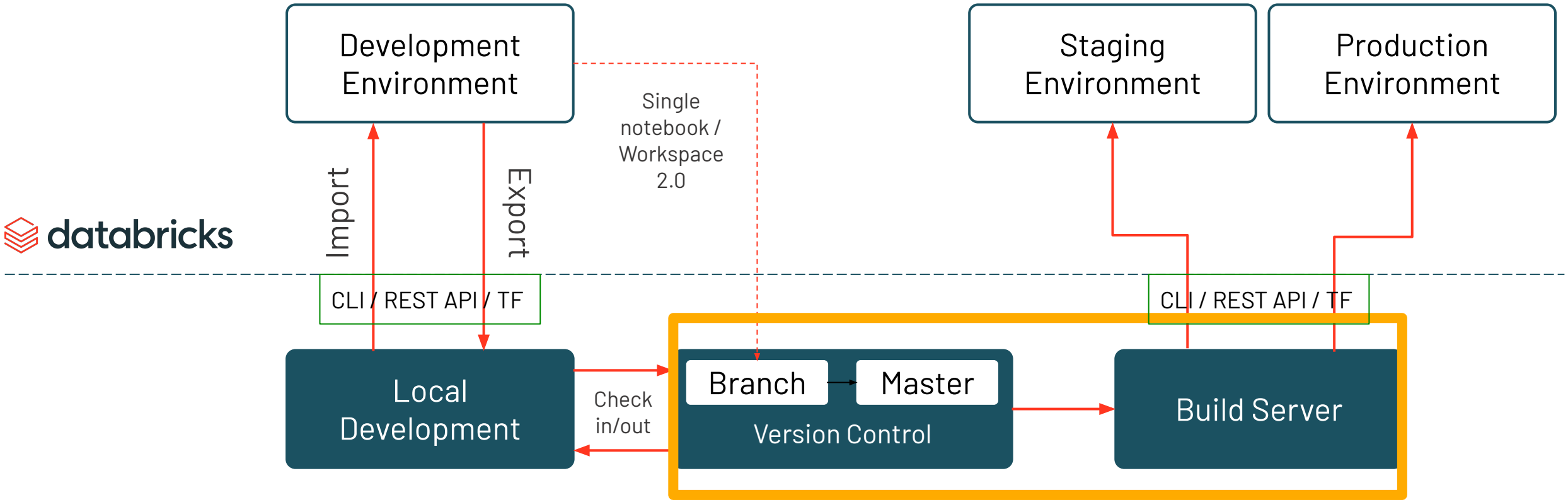
```
git add -A
git commit -m 'merging 0.0.1-dev to qa'
git push --set-upstream origin some-dir-0.0.1-dev
```

4. Merge and close the branch per your standard practice.

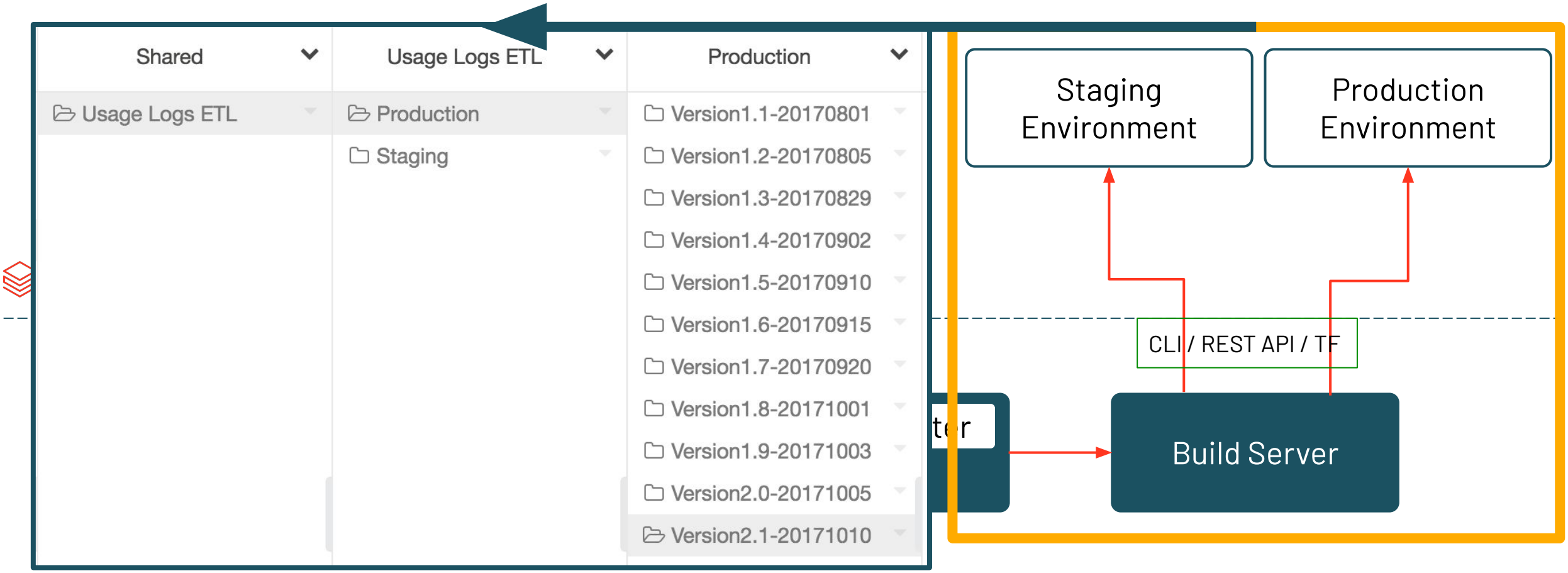
5. Checkout a new branch and upload to your workspace

```
git checkout some-dir-0.0.1-qa
databricks workspace import_dir -o . \
  /Users/some.user@email.com/some-dir-0.0.1-qa
```

CI/CD for Notebooks: Integration



CI/CD for Notebooks: Staging/Release Deployment



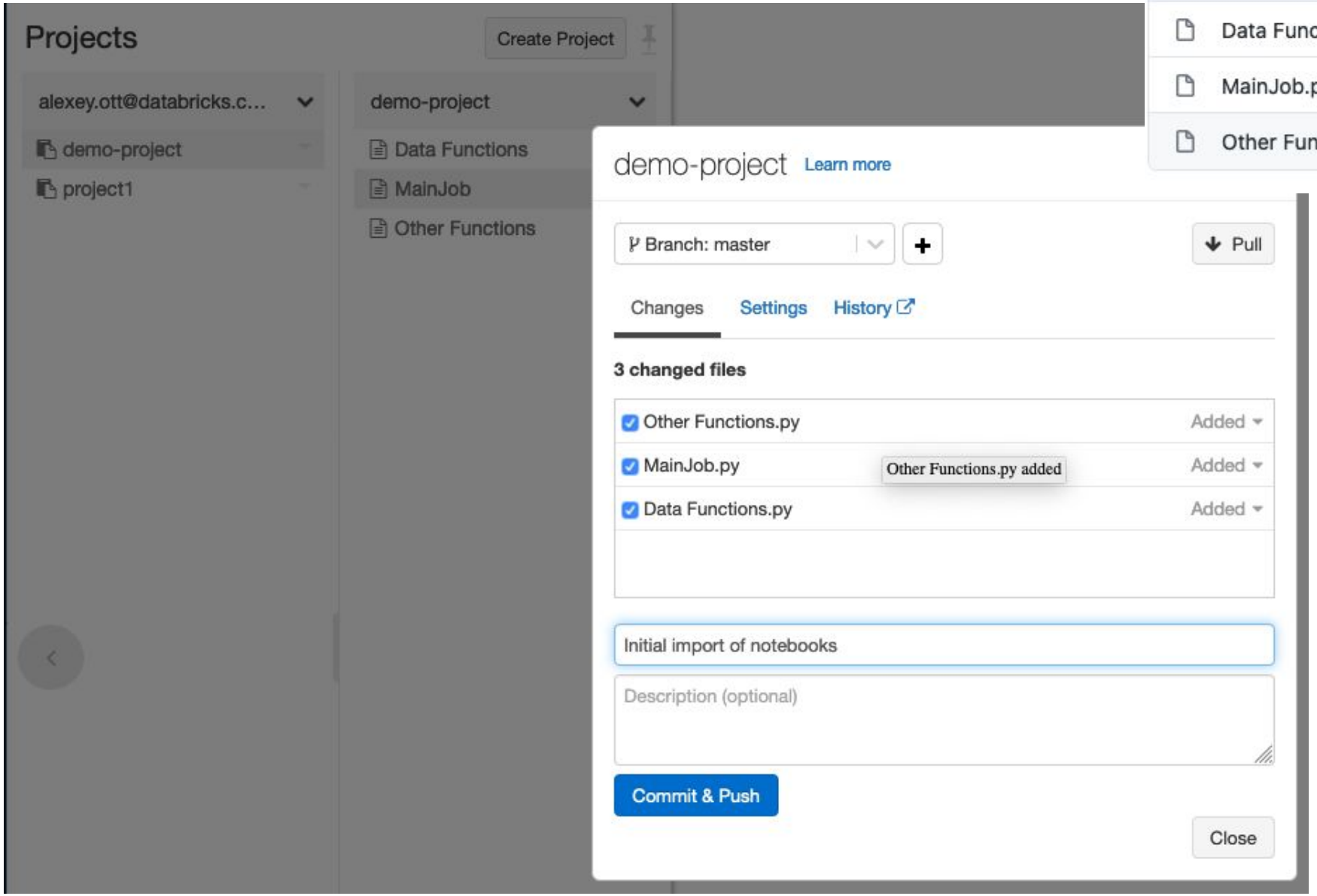
Testing notebooks: using %run

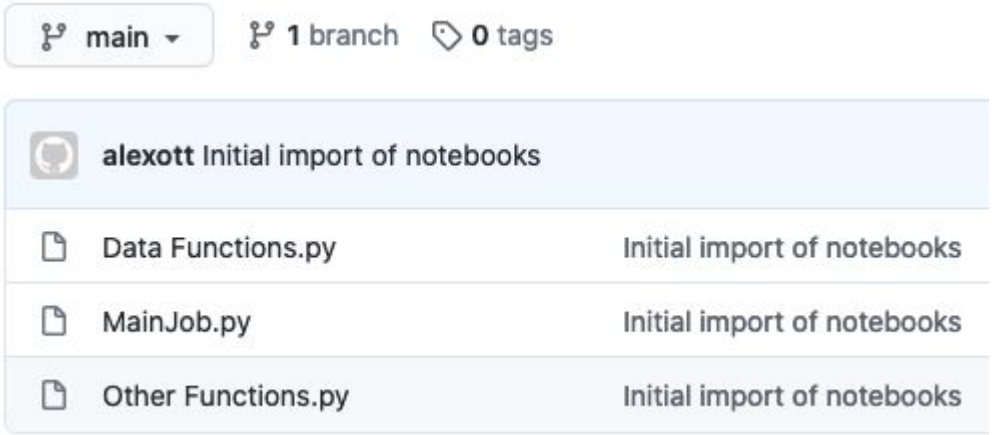
- Workflow:
  - Create separate notebook with functions that you want to test
  - Create separate notebook for tests
  - Import defined functions from notebook **%run**
  - Depending on the language, defined test functions or test classes
  - Execute tests
- This workflow may require special approach when using test frameworks - they often rely on the annotations and auto-discovery, so may not work with notebooks:
  - Solution - create test suites explicitly & execute them

Testing notebooks: using Nutter

- Nutter:
  - Developed by Microsoft
  - Allows to write tests in Python
  - Execute tests interactively / scheduled as job / triggered via command-line
  - Code is split into run / assert stages, with optional before / after calls - *should follow naming conventions*!
  - We can also use some of the frameworks listed later, like, Chispa
  - Possible parallel execution of the tests (but be careful)
  - Publish results in JUnit format
  - Easy to integrate with Azure DevOps & other systems

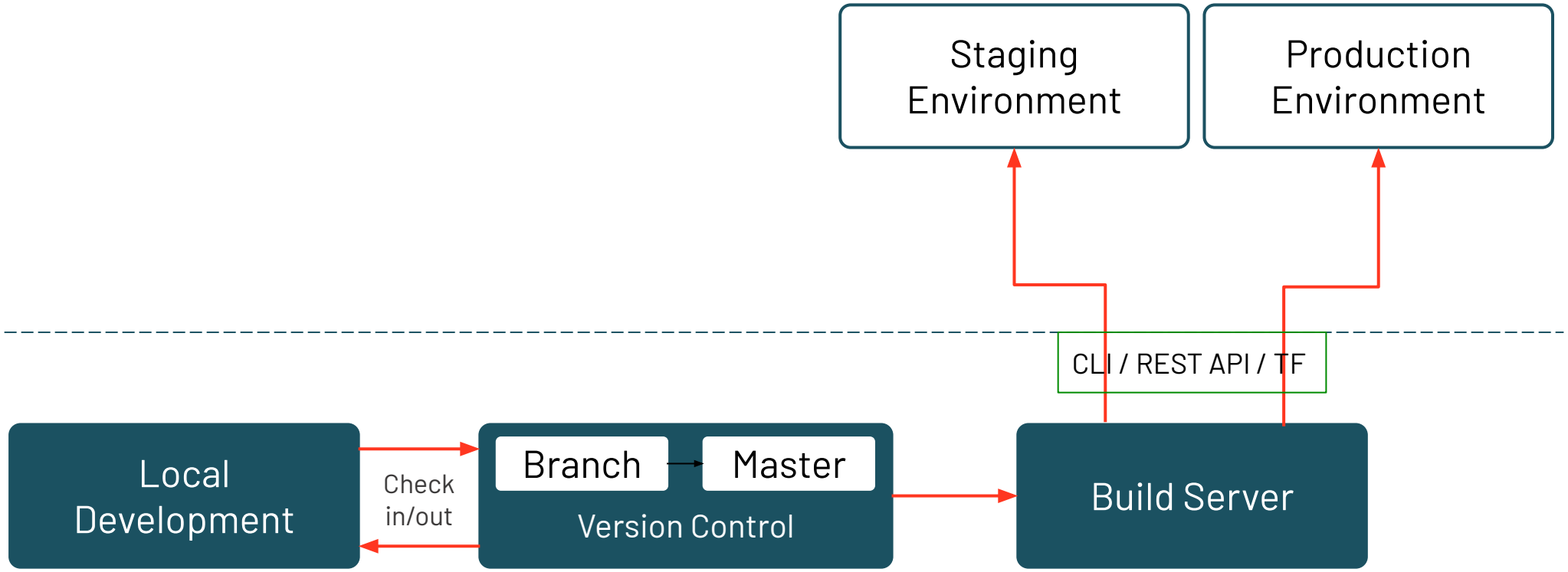
Workspaces 2.0: Repos



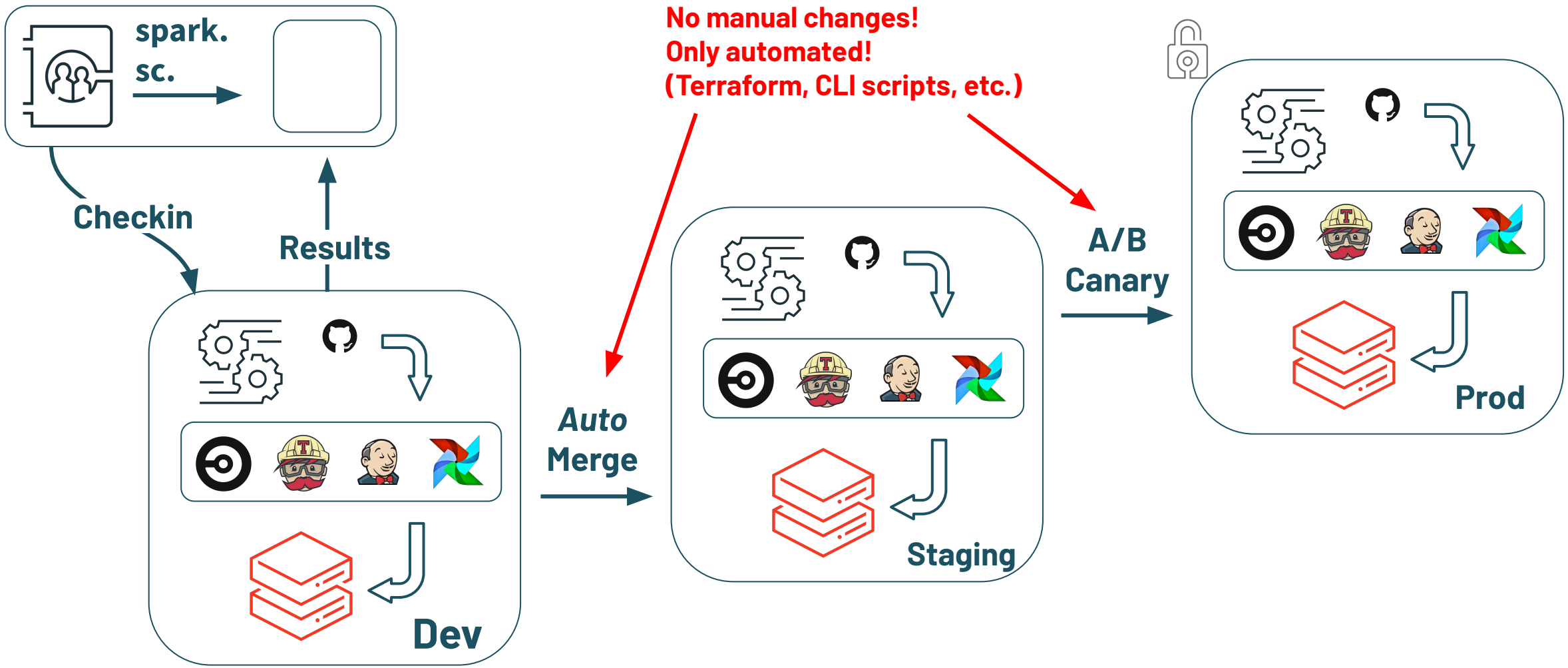


CI/CD for Databricks & IDE (code only)

CI/CD for IDE: Overview



CI/CD With Databricks and DB Connect / CLI



Testing libraries for Spark

- Built-in Spark test suite
  - Designed to test all parts of Spark
  - Supports RDD, Dataframe/Dataset, Streaming APIs
- spark-testing-base:
  - Scala & Python support
  - Supports RDD, Dataframe/Dataset, Streaming APIs
- spark-fast-tests - Scala, Spark 2 & 3
- chispa - Python version of spark-fast-tests
- pytest-spark - Python, native integration with pytest

- Code samples for all libraries in one place

Library management

Artifacts

Libraries & other artifacts built by build server are pushed:

- to DBFS - dedicated location, versioned.
  - pros: simple to implement
  - cons: harder to maintain, when multiple workspaces are used
- to Artifact store - Azure DevOps, Artifactory, ...
  - pros: easy access from multiple workspaces, integrated with CI/CD systems
  - cons: need external system, may not always work well with authenticated services, ...

Attaching artifacts to clusters/jobs

- Prefer to use IaC (Infrastructure-as-Code) - Databricks Terraform provider:
  - pros: versioned, easy to rollback not working changes
  - pros: changes are made via pull requests, tested by build pipelines, published by release pipelines to multiple environments
  - pros: could be completely automated on release of artifacts
  - cons: need to have/learn one more tool
- Alternatives - scripts using Databricks CLI or REST API
  - cons: harder to maintain

IaC example for libraries

```
data "databricks_spark_version" "latest" {}
data "databricks_node_type" "smallest" {
  local_disk = true
}
resource "databricks_dbfs_file" "library_jar" {
  source = pathexpand(var.jar_path)
  path = "/FileStore/jars/my-cool-lib-v1.2.jar"
}
```

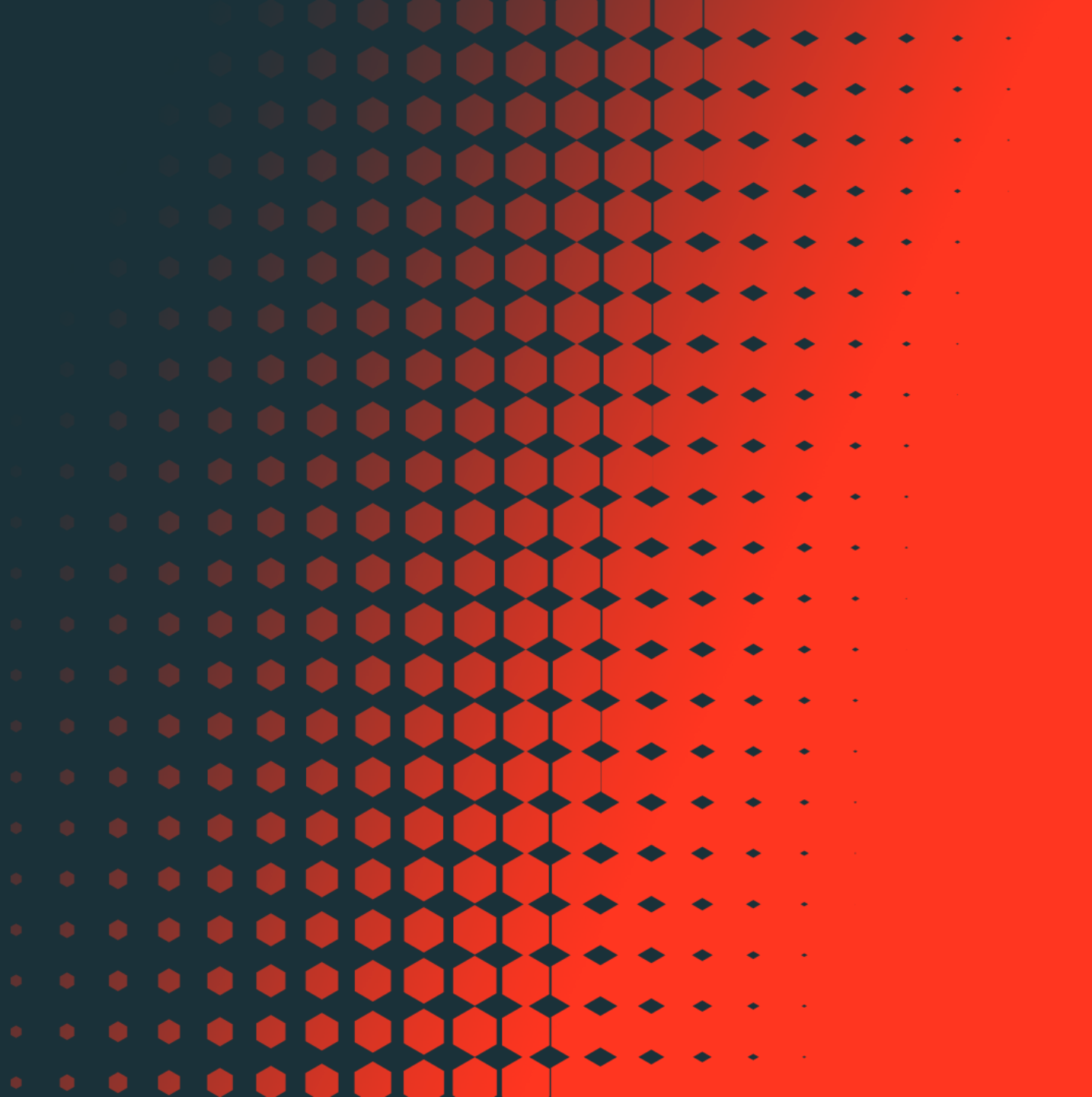
```
Unstaged changes (1)
modified   job1.tf
@@ -5,7 +5,7 @@ data "databricks_node_type" "smallest" {

 resource "databricks_dbfs_file" "library_jar" {
   source = pathexpand(var.jar_path)
-  path = "/FileStore/jars/my-cool-lib-v1.2.jar"
+  path = "/FileStore/jars/my-cool-lib-v1.3.jar"
 }

 resource "databricks_job" "job1" {

Recent commits
```

```
resource "databricks_job" "job1" {
  name = "Job 1"
  max_concurrent_runs = 1
  new_cluster  {
    num_workers = var.num_workers
    spark_version = data.databricks_spark_version.latest.id
    node_type_id  = data.databricks_node_type.smallest.id
  }
  notebook_task {
    notebook_path = var.notebook_path
  }
  library {
    jar = "dbfs:${databricks_dbfs_file.library_jar.path}"
  }
}
```

PR is reviewed

Build & Release Pipelines

Scheduling

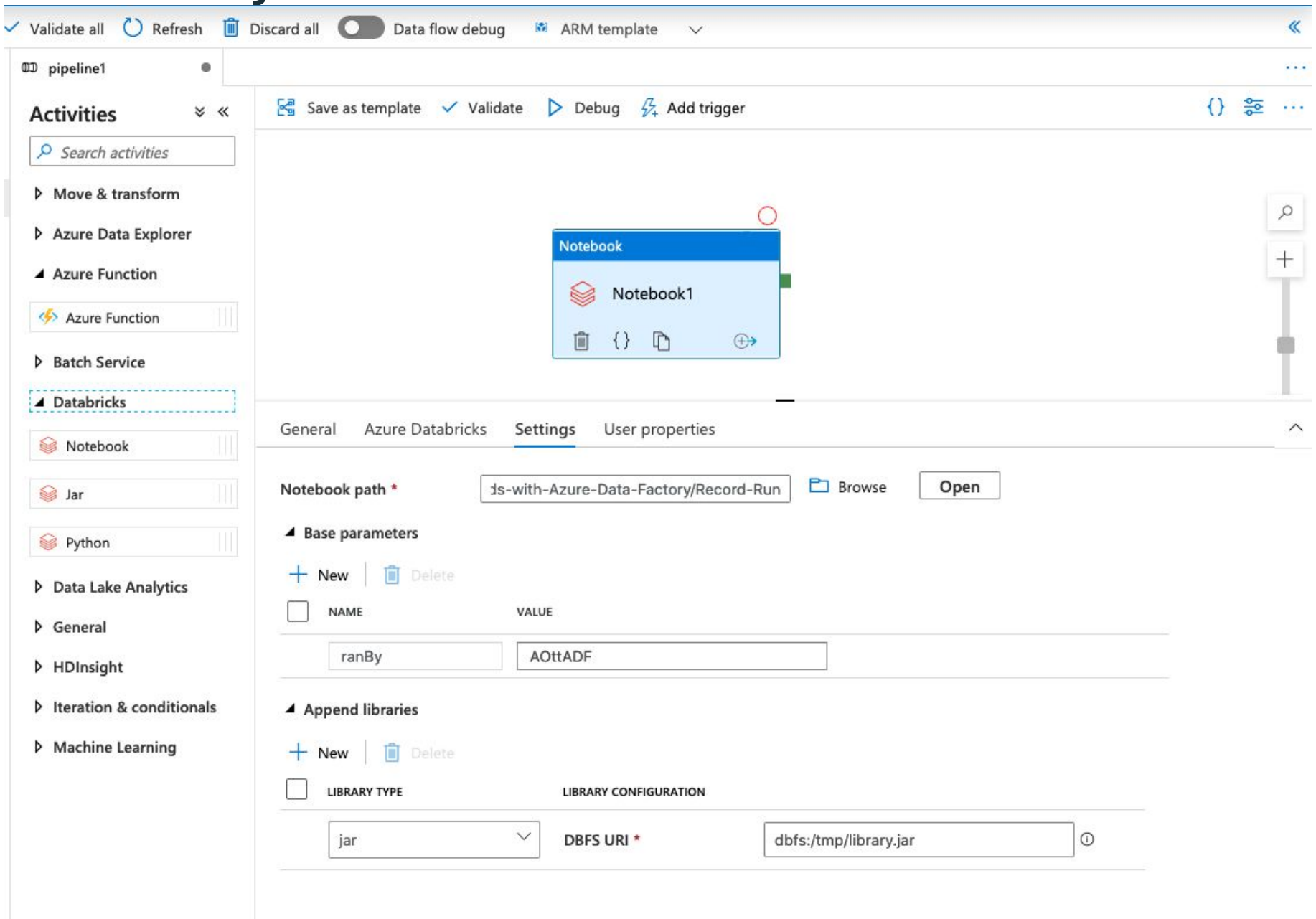


databricks

Scheduling

- Built-in job scheduling (doc):
  - Periodic scheduling of the jobs (cron-like right now)
  - Execute notebook / jar / Python script
- Apache Airflow (doc)
  - Contrib module
  - Execute notebook / jar / Python script
- Azure Data Factory
  - Execute notebook / jar / Python script
  - Triggers: time-based, ADF Event-based

Azure Data Factory

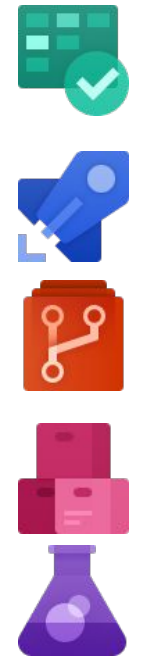


databricks

Azure DevOps

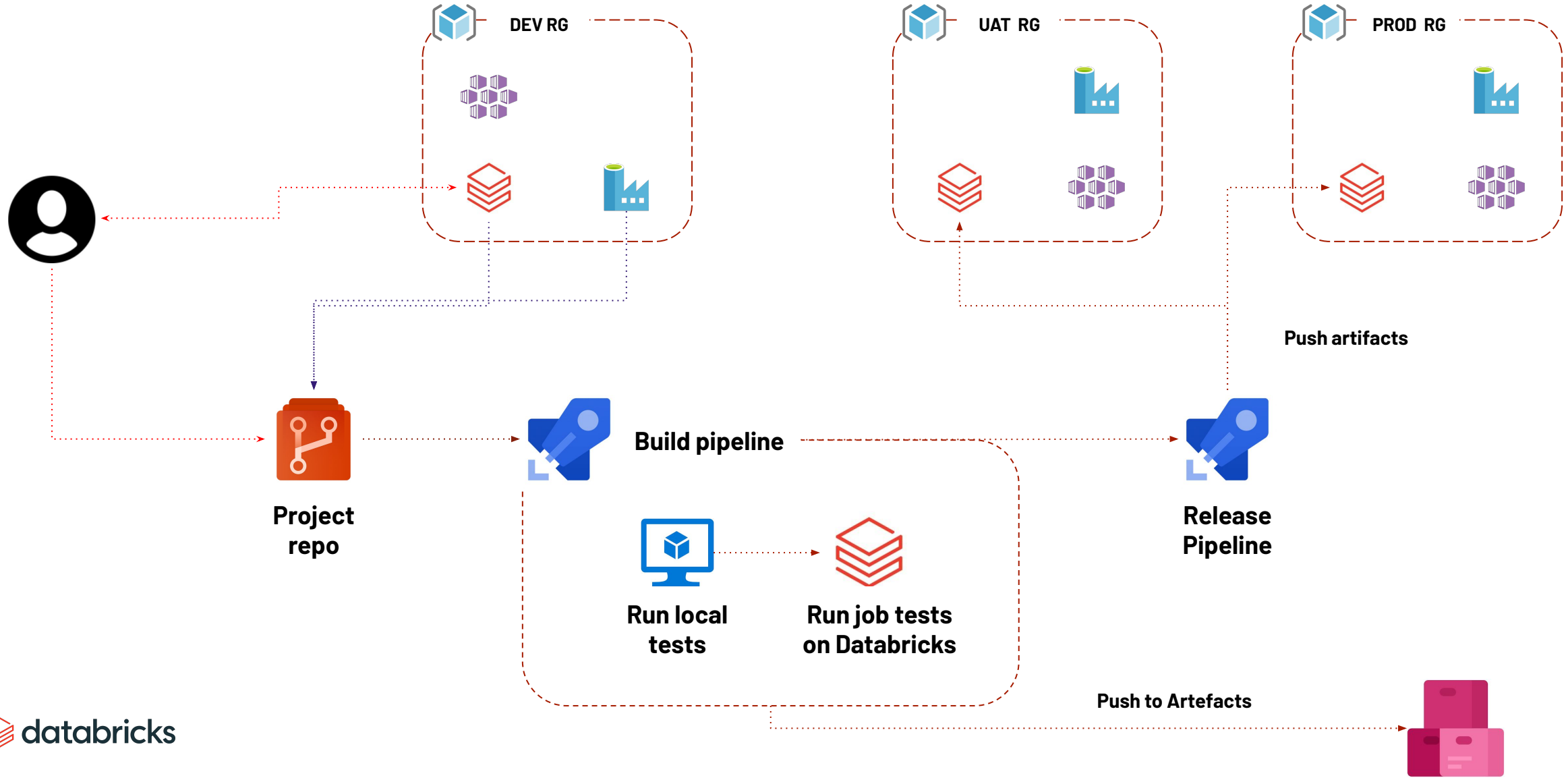
Azure DevOps

- Implementation of full DevOps lifecycle
- Components:
  - Boards - Agile tools to plan & track team work, integrated with other components
  - Pipelines - CI/CD implementation - build, test & deploy for different languages, operating systems, ...
  - Repositories - Git repositories with pull requests, etc.
  - Artifacts - host external packages, or artifacts from you CI/CD
  - Test plans - support for different types of testing
- Transparent integration with Azure Databricks



CI/CD for Databricks

General CI/CD workflow on Databricks

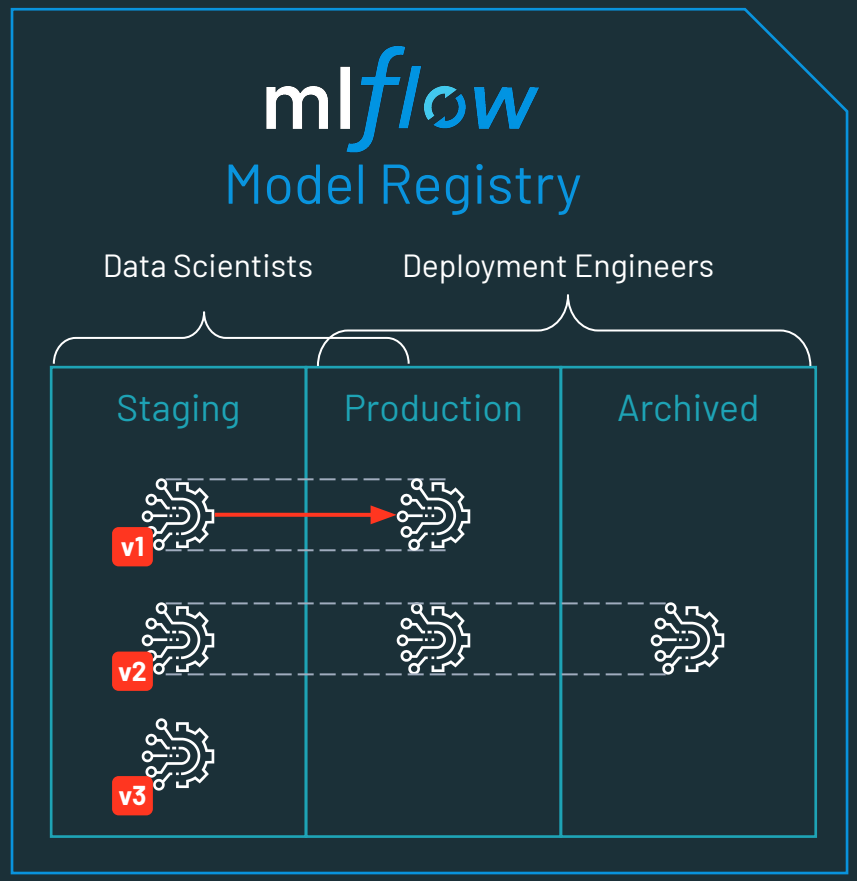


databricks

General CI/CD workflow on Databricks

- Develop the code using one of the workflows (notebooks/code/mixed)
- Pull notebooks & commit them (databricks-cli/stack-cli)
- Build assets via build pipeline & do unit testing
- Push notebooks to staging (databricks-cli/stack-cli)
- Push assets to staging (databricks-cli, terraform)
- Run tests (unit/integration/e2e) on staging (databricks-cli, schedulers)
- If successful, push notebooks & assets to production (databricks-cli/stack-cli)
- Reconfigure jobs / cluster to use new notebooks/assets, if necessary (databricks-cli, terraform)

mlflow Model Deployment



In-Line Code

Containers

Batch & Stream Scoring

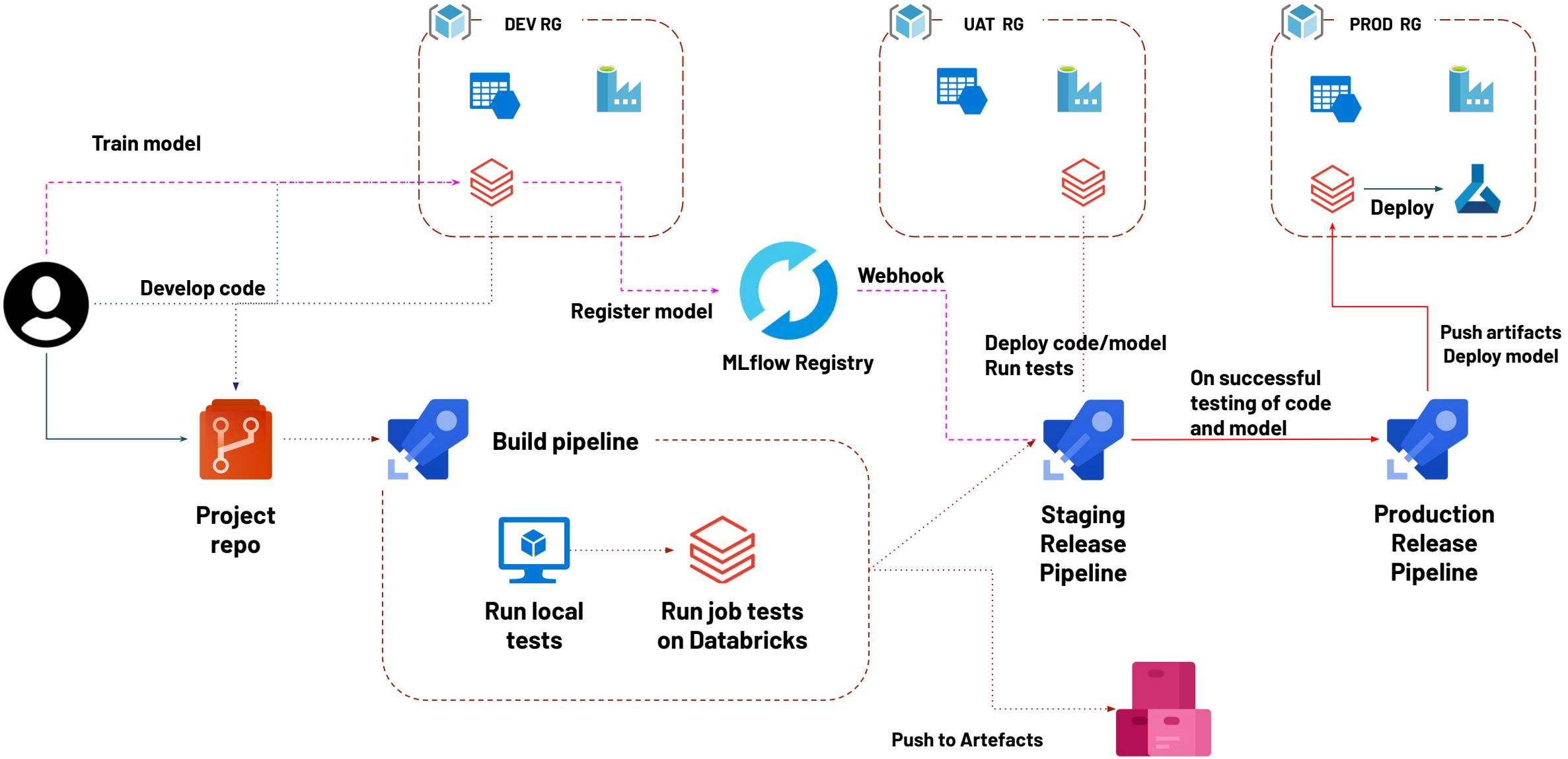
Cloud Inference Services

OSS Serving Solutions

```
model_udf =
     mlflow.pyfunc.spark_udf(
          spark,
          model_uri='models:/forecast/production')
```

databricks

CI/CD workflow on Databricks for ML



databricks

Machine Learning workflow on Databricks

- ML model training job triggering:
  - Model code changes (traditional triggers on commit) - sometimes because of the concept drift, etc.
  - Model configuration changes (traditional triggers on commit)
  - Data changes (explicit triggering, or automated if we detected the data drift)
- Trained model is registered in MLflow & moved to staging
- Acceptance tests are executed
- Model is promoted from staging to production & deployed (Azure ML, Databricks, other systems)

- Webhooks in the MLflow registry may help with better integration with CI/CD pipelines

CI/CD integrations for Databricks

Existing integrations

- Azure DevOps
  - Use cicd-templates to generate pre-configured project template
  - Continuous integration and delivery on Azure Databricks using Azure DevOps
  - Implement CI/CD with Azure DevOps (MS Learning)
  - Additional DevOps integrations via DevOps for Azure Databricks by Microsoft DevLabs or 3rd party package by Data Thirst
- Jenkins
  - Continuous integration and delivery on Azure Databricks using Jenkins
- Github Actions
  - Use cicd-templates to generate pre-configured project template

cicd-templates project

- Databricks Labs project
- Allow to generate a project skeleton
- Generates skeleton for AWS or Azure
- Supports Azure DevOps & GitHub Actions
- Placeholders for unit & integration tests
- Support for several pipelines in the same project
- Auxiliary scripts for deployment

- More information in blog post and project's wiki

DEMO

Databricks Runtime Upgrades

New Databricks runtime versions

- Bring new functionality (new SQL capabilities, more functions, ...)
- Bring performance improvements (adaptive query execution, more optimizations for Delta, etc.)
- New libraries & library versions included

Problems:

- Could break existing code when APIs change
- May require recompilation of all libraries (Scala 2.11 -> Scala 2.12, etc.)
- May require upgrade of used libraries (that also may have API changes)
- Primarily on major upgrades: DBR 5.x/6.x -> 7.x/8.x

Performing safe DBR versions upgrade

Requirements:

- Have decent amount of unit tests
- Have decent integration/acceptance tests
- Read through the release notes of selected Databricks Runtime version
  - For major versions, look for migration guides, like, this for Spark 3
- IaC for jobs/clusters/libraries/…

How:

- make PR for IaC & allow CI/CD to execute unit & integration tests
- Analyze tests failures
- Make changes in the code & IaC
- …

```
Unstaged changes (1)
modified   job1.tf
@@ -13,7 +13,7 @@ resource "databricks_job" "job1" {

  new_cluster {
    num_workers = var.num_workers
-   spark_version = "6.4.x-scala2.11"
+   spark_version = "7.6.x-scala2.12"
    node_type_id  = data.databricks_node_type.smallest.id
  }
```

Infrastructure as Code

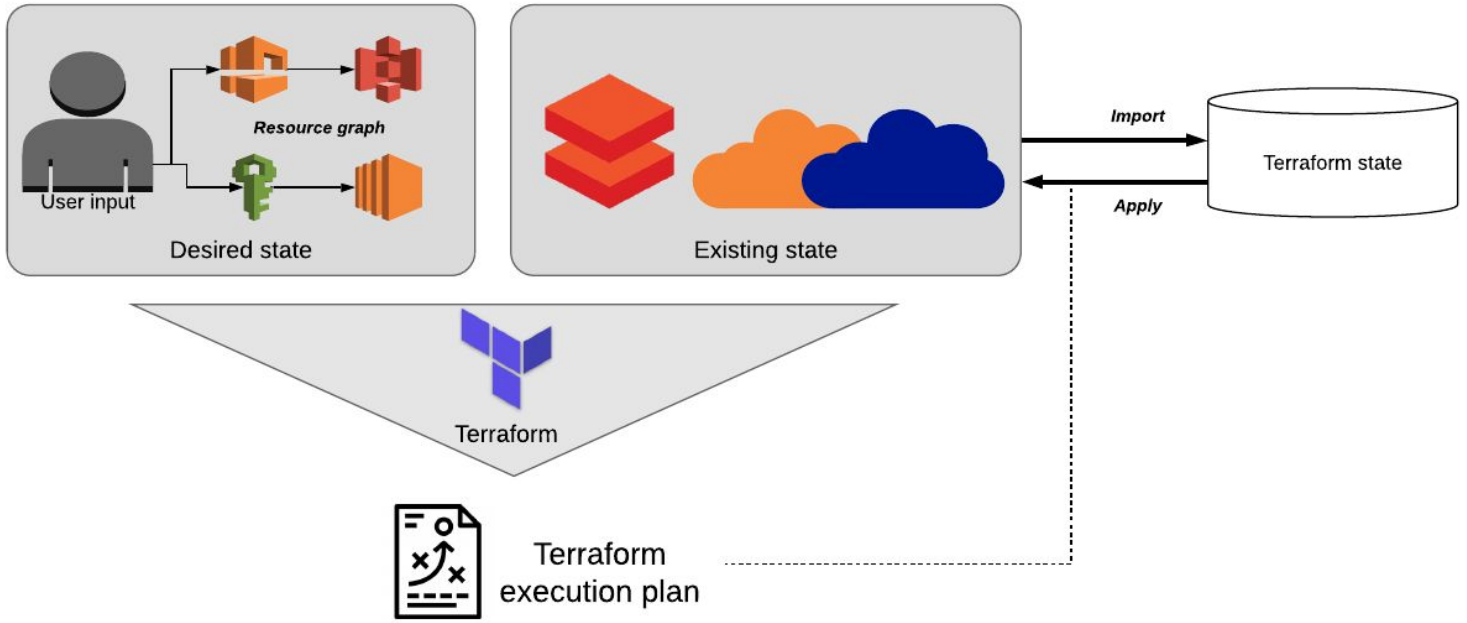
Infrastructure as a Code

- 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 integrate with CI/CD pipelines
- Popular implementations: HashiCorp Terraform, Ansible, AWS CloudFormation, Azure Resource Management templates, ...

Terraform

Why Terraform?

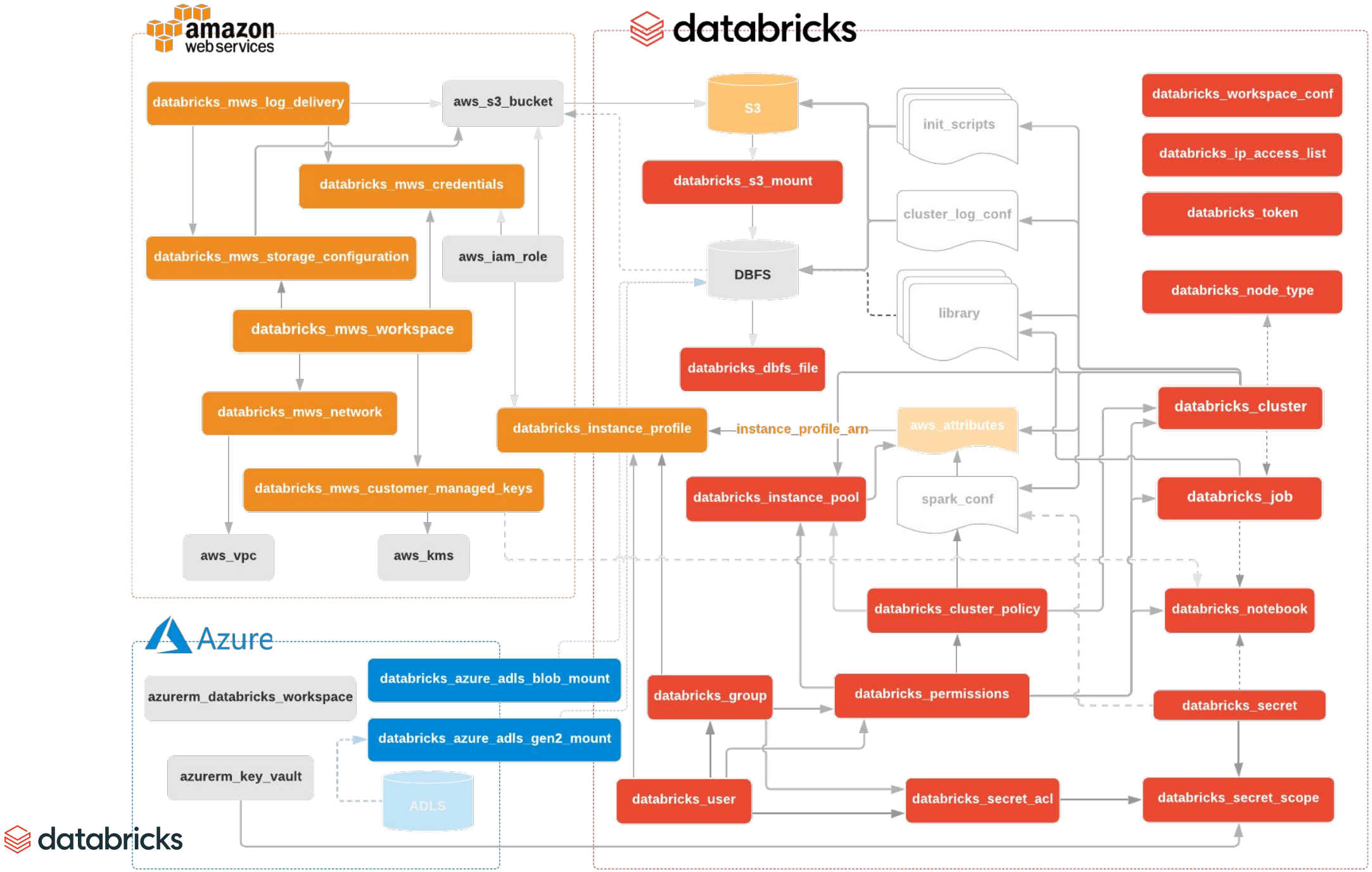
- Declarative syntax
- A lot of integrations
- Keeps track of infrastructure state
- Automatic handling of dependencies between objects
- One command create/update/delete of all objects
  - Easy creation of new environments
  - Disaster recovery, etc.



Databricks Terraform Provider

- Supports all objects that are exposed via Databricks REST API (clusters, jobs, users/groups, notebooks, ...)
- Plays well with other Terraform providers, including cloud: Azure, AWS, GCP
- Different authentication methods
- Well documented
- Used by hundreds of customers worldwide
- Part of Databricks Labs

More information at
https://registry.terraform.io/providers/databrickslabs/databricks/latest


amazon web services
databricks
databricks_mws_log_delivery
aws_s3_bucket
databricks_mws_credentials
databricks_mws_storage_configuration
aws_iam_role
databricks_mws_workspace
databricks_mws_network
databricks_mws_customer_managed_keys
aws_vpc
aws_kms
databricks_instance_profile
instance_profile_arn
Azure
azurerm_databricks_workspace
databricks_azure_adls_blob_mount
databricks_azure_adls_gen2_mount
azurerm_key_vault
ADLS
S3
databricks_s3_mount
DBFS
databricks_dbfs_file
init_scripts
cluster_log_conf
library
aws_attributes
spark_conf
databricks_instance_pool
databricks_cluster_policy
databricks_group
databricks_permissions
databricks_user
databricks_secret_acl
databricks_workspace_conf
databricks_ip_access_list
databricks_token
databricks_node_type
databricks_cluster
databricks_job
databricks_notebook
databricks_secret
databricks_secret_scope
databricks

Example: Job with Notebook

```
provider "databricks" {
}

data "databricks_current_user" "me" {}
data "databricks_spark_version" "latest" {}
data "databricks_node_type" "smallest" {
  local_disk = true
}

resource "databricks_notebook" "this" {
  path     = "${data.databricks_current_user.me.home}/Terraform"
  language = "PYTHON"
  content_base64 = base64encode(<<-EOT
    # created from ${abspath(path.module)}
    display(spark.range(10))
    EOT
  )
}
```

```
resource "databricks_job" "this" {
  name = "Terraform Demo
(${data.databricks_current_user.me.alphanumeric})"

  new_cluster {
    num_workers   = 1
    spark_version = data.databricks_spark_version.latest.id
    node_type_id  = data.databricks_node_type.smallest.id
  }

  notebook_task {
    notebook_path = databricks_notebook.this.path
  }

  email_notifications {}
}

output "notebook_url" {
  value = databricks_notebook.this.url
}

output "job_url" {
  value = databricks_job.this.url
}
```

Additional information

Resources

- Databricks Labs CI/CD Templates
- Data engineering with Azure Databricks (MS Learning course)
- Example of template for Azure DevOps
- Example of using Nutter with Repos & Azure DevOps
- Blog posts:
  - Continuous Integration & Continuous Delivery with Databricks
  - Using MLOps with MLflow and Azure
  - Automate continuous integration and continuous delivery on Databricks using Databricks Labs CI/CD Templates
  - Productionizing Machine Learning: From Deployment to Drift Detection
  - MLflow Model Registry on Databricks Simplifies MLOps With CI/CD Features

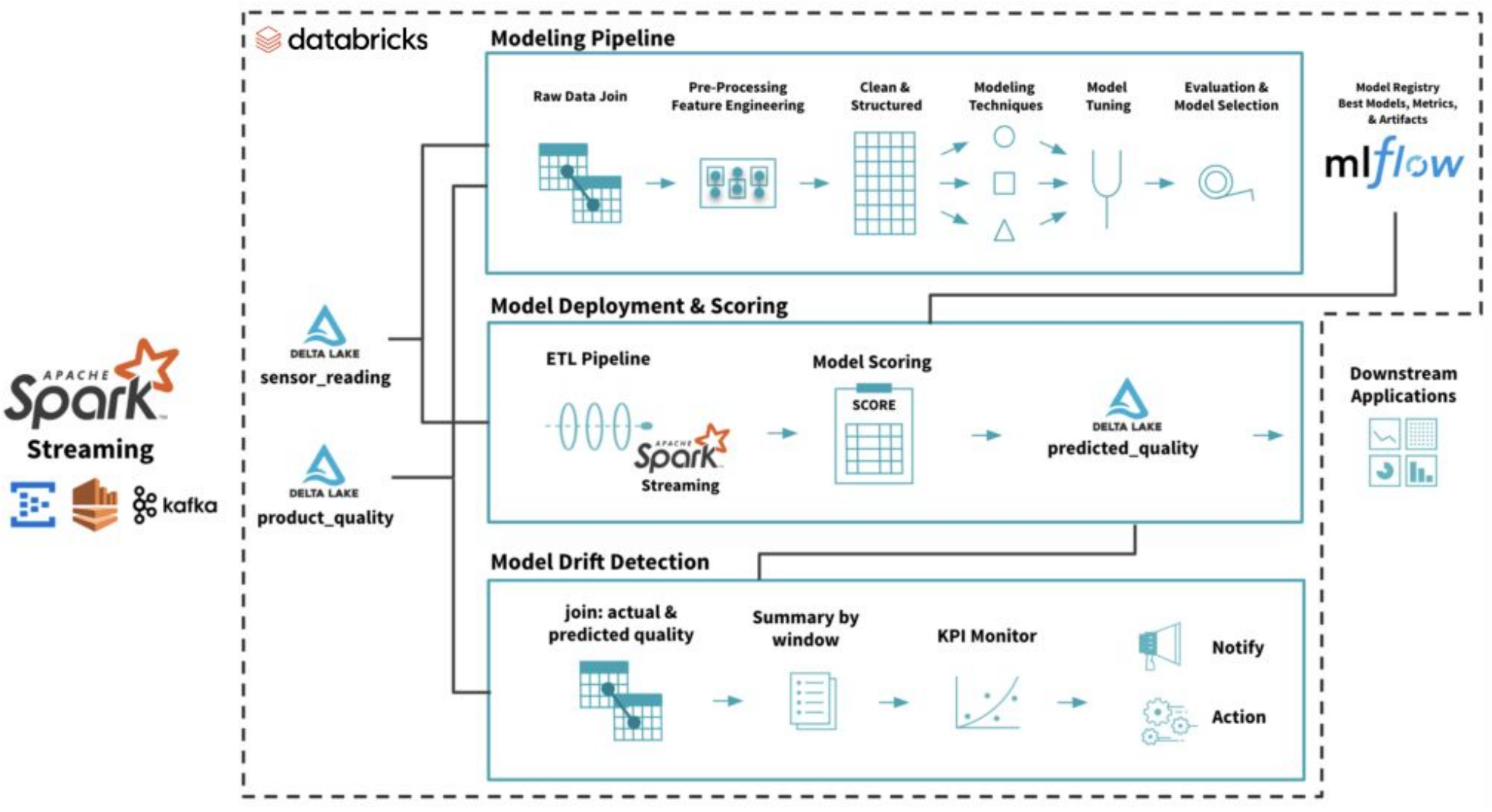
Resources

- MLflow project
- MLflow on Databricks
- Data + AI Summit 2020 sessions:
  - Continuous Delivery of ML-Enabled Pipelines on Databricks using MLflow
  - Introducing MLflow for End-to-End Machine Learning on Databricks
  - Productionalizing Models through CI/CD Design with MLflow
  - Scaling Production Machine Learning Pipelines with Databricks
  - Machine Learning Data Lineage with MLflow and Delta Lake
  - Thursday Morning Keynotes - about Workspaces 2.0
- https://ml-ops.org/

Monitoring

Other Issues

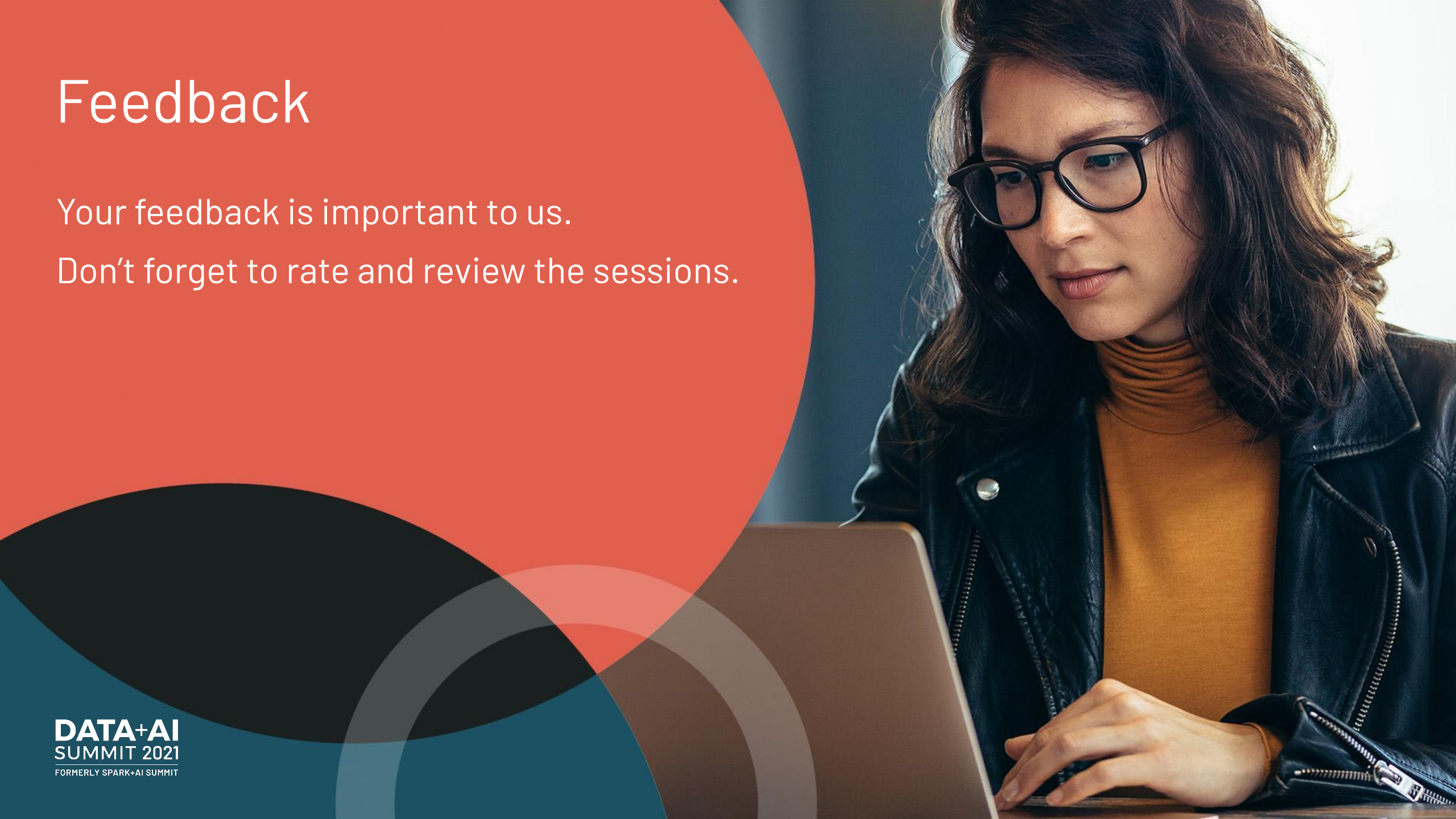
Architecture I



Feedback

Your feedback is important to us.

Don’t forget to rate and review the sessions.



DATA+AI SUMMIT 2021
FORMERLY SPARK+AI SUMMIT