CSCI E-103

Data Engineering for Analytics to Solve Business Challenges

Getting the most value out of your data initiative investments through a culture of Continuous Improvement

Lecture 13

Eric Gieseke & Ramdas Marali

Harvard Extension, Fall 2025

Agenda

- Final Assignment
- Software Development Lifecycle (SDLC) of a Data Persona
- Creating Robust & Scalable Data Products & Services that are also price-performant
    - Fierce competition in the tech landscape
    - Value added Differentiators
    - Balancing: Capabilities Vs Performance Vs Price
- Automation of Infrastructure & Pipelines(Data/ML) - IaaC & CI/CD
- Observability & Monitoring
- Improving ROI with a Center of Excellence (CoE)
    - Time to Market & Upskilling
    - Capacity Planning & Forecasting
- Data Intelligent Platforms in the age of GenAI

- Lab:
    - Databricks Asset Bundles (DAB)

Final Project - Building the Lakehouse Architecture

Assignment Description

Presentation Agenda

- Problem statement
- Team Introduction (by role)
- Design & Architecture of the Lakehouse Paradigm (Data Architect)
  - Scalable/Quality/Performance/Reliability/Governance
- Data Pipeline - Ingestion/Transformation/Joins/Aggregations (Data Engineer)
- Model - training/Inferencing/life cycle Management (ML Practitioner)
- BI Dashboard - Queries, Dashboard, Notifications (Data Analyst)
  - Connectivity/Scale/Security Concerns
- Summarize Insights
- Next set of planned refinements

Group Presentation Notes

20 slides max
15 min presentation
5 min Q/A

2-3 external judges

Give your team/consulting firm a name

We'll go from Groups 1-N
Folks in other time zones will go first

Operationalizing Pipelines

- Understand Use Case
  - Business Challenges & domain specific nuances
  - Requirements (functional, non-functional) SLAs
- Build for Scale
  - Investment at risk
- Winning the Architecture
  - Build repeatable patterns using frameworks that are config driven
  - Document Best Practices
  - Early Performance Tuning to showcase efficiencies (Project TCO)
  - Infrastructure Efficiencies (storage life cycle, auto termination)
  - Security & Business Continuity
- Observability
  - Monitoring (data & usage)
- Automation & CI/CD
  - Infrastructure as Code (IaC)
  - Testing Automation
- Center Of Excellence (COE)
  - Expertise, best practices, and support

Software Development Lifecycle (SDLC) of Data Personas

Promoting workloads from lower to higher environments

Dev Phase

1

- Understand Requirements
- Implement Business Logic
- Use canned/test data
- Preferably use
  - Single node cluster
  - Spot instances

Stage Phase

2

- Increase dataset
- Cover test scenarios
- Automate
  - Repos
- Move from
  - Interactive to automated
  - User mode to Service Principals

Prod Phase

3

- Fully automated
- Use the right policies
- Use only Service Principals

Service Level Agreement (SLA)

Depends on workload type (as soon as possible is not practical)

Batch

Volume

Processing Time

Streaming

TPS

Tx/Events per second

BI

E2E latency

Concurrency

High Availability

Response Time

Availability = (Uptime / Total Time) x 100%

E.g., 99.999% => 5 minutes/year

Disaster Recovery

Recovery Time Objective (RTO)

Recovery Point Objective (RPO)

Price-Performance

- Choose a platform that provides out of the box price-performance
  - yet gives the control to you to choose from
- Industry-standard benchmarks
  - TPC-DS
    - Analytics / Decision Support (BI)
    - How fast and effectively you query the data warehouse
    - Databricks Sets Official Data Warehousing Performance Record
  - TPC-DI
    - Data Integration / ETL
    - How fast and effectively you build and load the data warehouse
    - How We Performed ETL on One Billion Records For Under $1 With Delta Live Tables

https://github.com/anhhchu/Metimur

https://docs.databricks.com/aws/en/sql/tpcds-eval

Transaction Processing Performance Council (**TPC**)

Cost Optimizations

- Don't make assumptions, experimentation is cheap
  - Spark is mature and scales almost linearly
  - Performance requirements come in the form of **development**, **ecosystem**, or **business** needs.
- Better code
  - Leverage dataframe APIs over home grown UDFs
  - Cache during ML training
  - Use binary formats like Delta
- Right infrastructure
  - Instance type, cluster sizing
- Governance
  - Cluster policies
  - Tags
- Autoscaling & auto termination
- Spot instances, spot fleet
- Higher Spark versions
  - Better performance & bug fixes apart from enhanced capabilities
- Pool -> Serverless
- Monitoring & Alerting
  - Usage overview for chargeback & control runaway costs
  - Tags for attribution

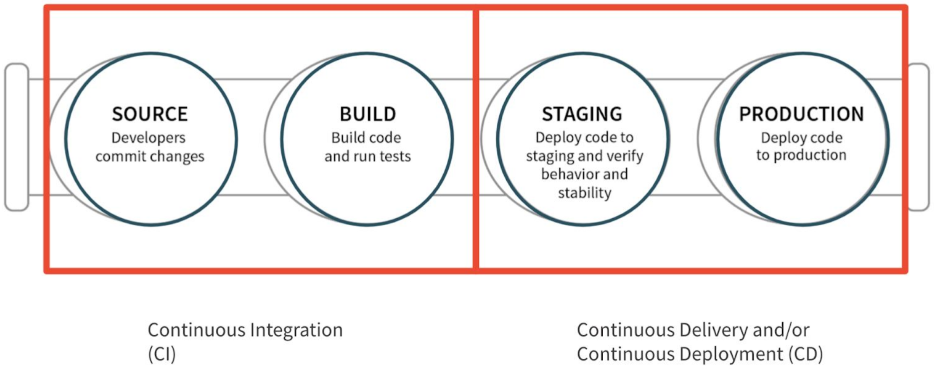
CI/CD

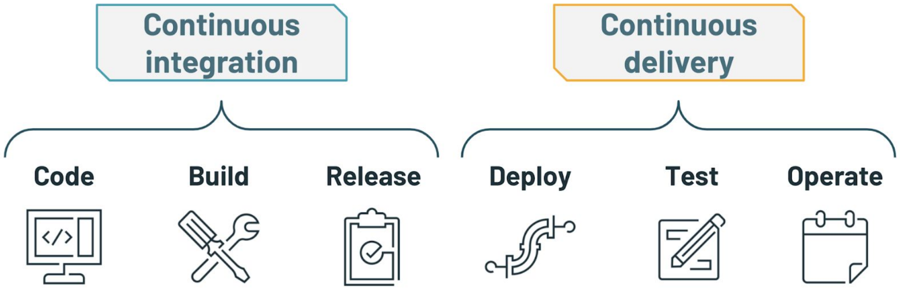
CI = Continuous Integration

... Developers continually make new changes. Code is auto tested and promoted.

CD = Continuous Delivery / Deployment

... Continually deploy code into production at click of a button / automatically

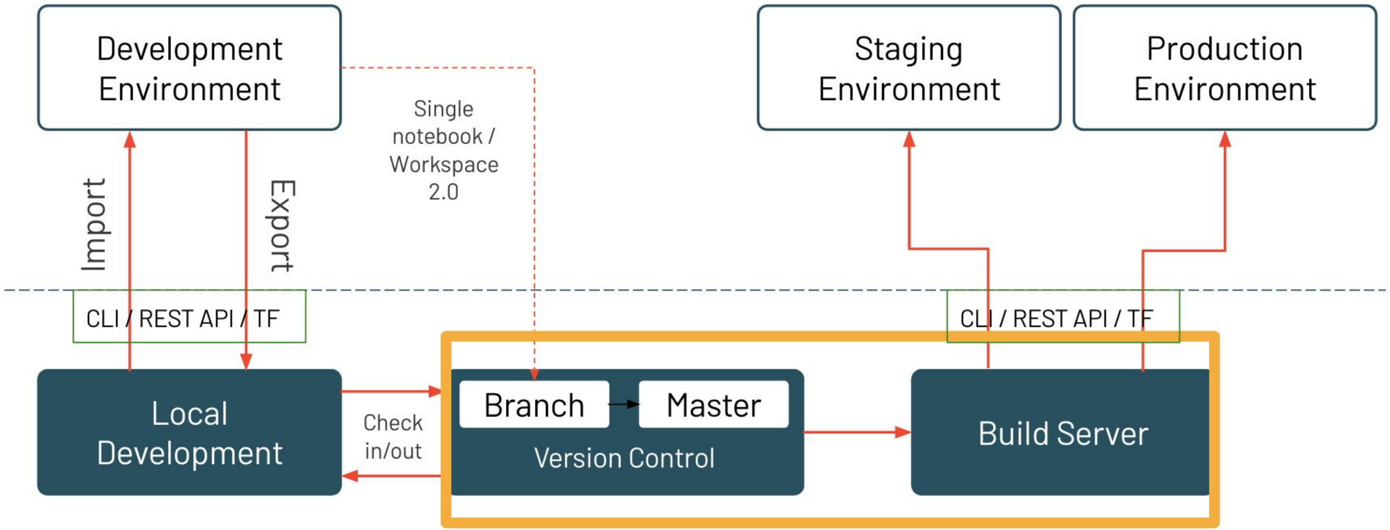




Development Workflows

Code developed in IDE, packaged & deployed as libraries

Notebooks are used for configuration & orchestration of calls to libraries



Infrastructure as Code (IaC)

- Infrastructure as Code is an approach to infrastructure automation based on practices from software development

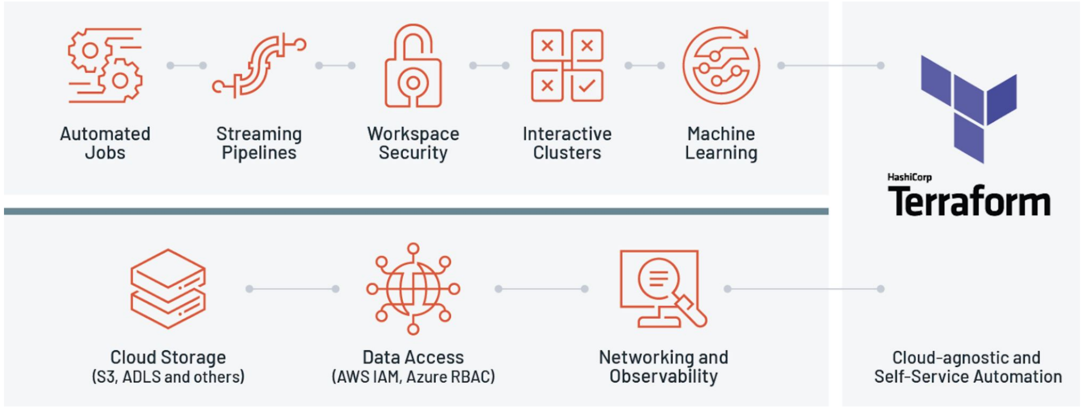
- Core Practices of IaC:
    1. ***Define Everything as Code***
    2. ***Continuously Test and Deliver All Work in Progress***
    3. ***Build Small, Simple Pieces that you can change independently***

- In Databricks, Infrastructure as Code can pertain to both Cloud Infrastructure and Workspace Infrastructure

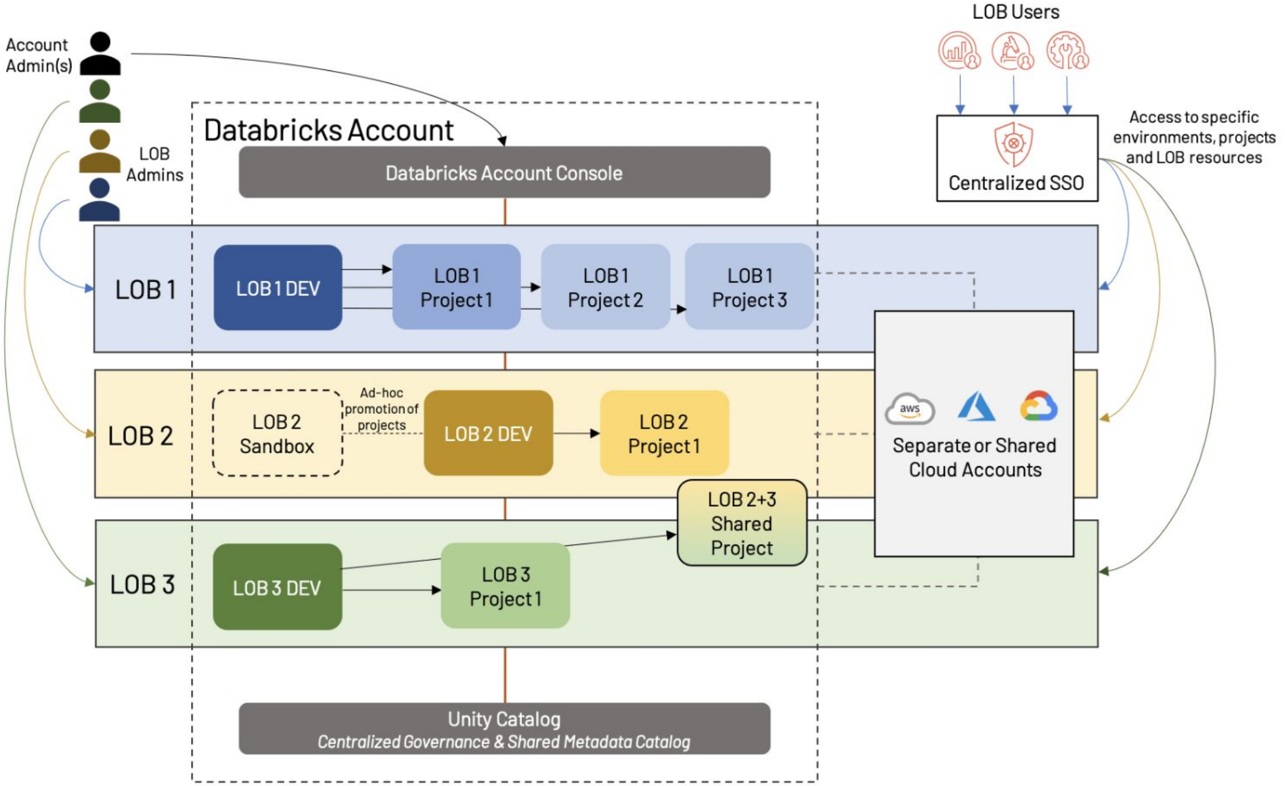
Automation Options

	Source	Cloud	Type	Infrastructure	Language	Agent	Master	Community	Maturity
Chef	Open	All	Config Mgmt	Mutable	Procedural	Yes	Yes	Large	High
Puppet	Open	All	Config Mgmt	Mutable	Declarative	Yes	Yes	Large	High
Ansible	Open	All	Config Mgmt	Mutable	Procedural	No	No	Huge	Medium
SaltStack	Open	All	Config Mgmt	Mutable	Declarative	Yes	Yes	Large	Medium
CloudFormation	Closed	AWS	Provisioning	Immutable	Declarative	No	No	Small	Medium
Heat	Open	All	Provisioning	Immutable	Declarative	No	No	Small	Low
Terraform	Open	All	Provisioning	Immutable	Declarative	No	No	Huge	Low

- Configuration Management vs Provisioning
- Mutable Infrastructure vs Immutable Infrastructure
- Procedural vs Declarative
- Master vs Masterless
- Agent vs Agentless
- Large Community vs Small Community
- Mature vs Cutting Edge
- Using Multiple Tools Together



Workspace Organization



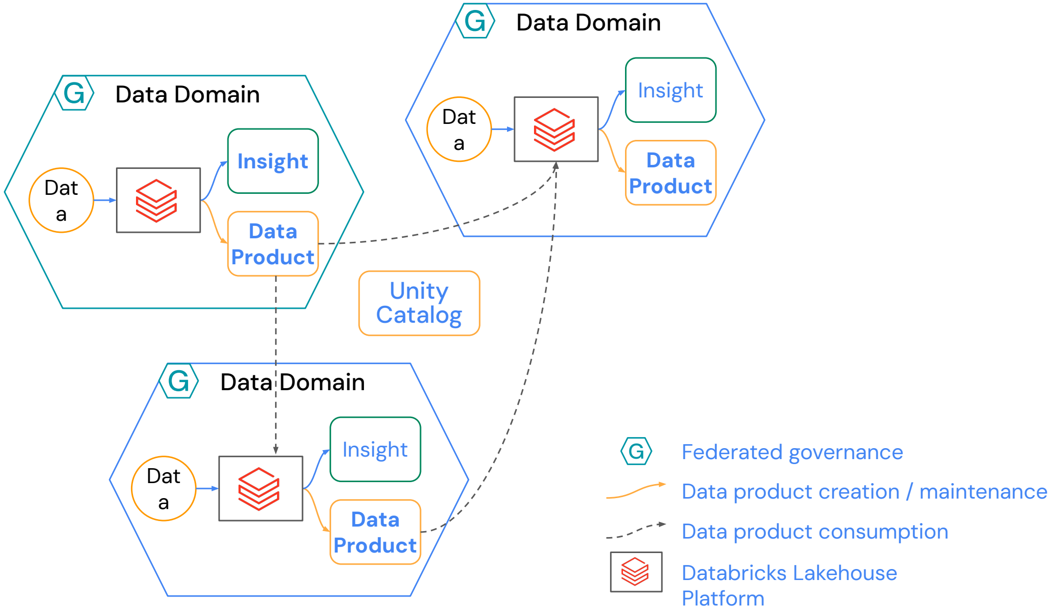
Balance of Isolation & Management overhead

Lakehouse and Data Mesh

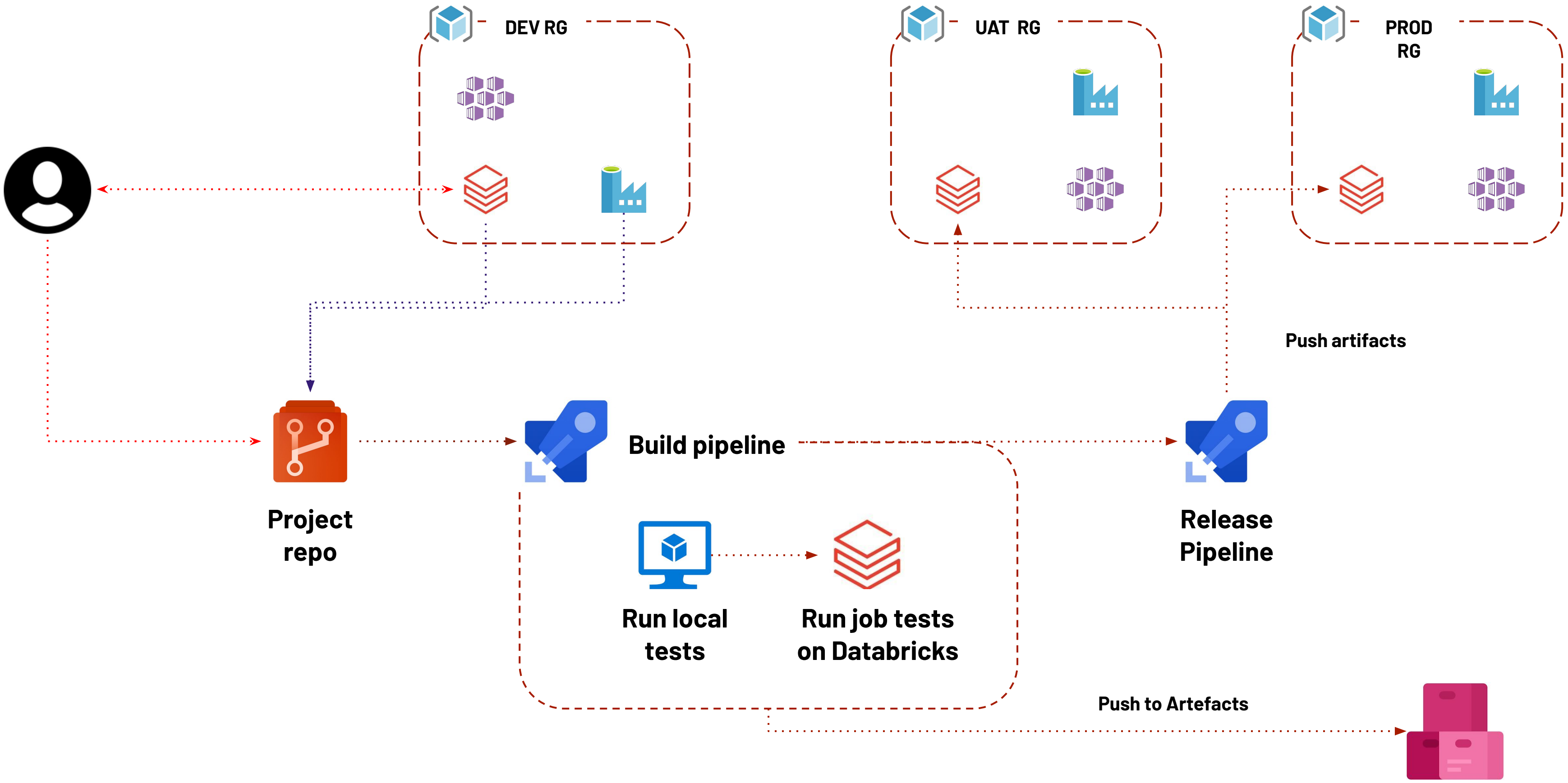
Business units are organized in **Data Domains** and own their respective sources, data and metadata. Using a Lakehouse, they create domain specific insights from data and offer **Data Products** to other domains using **Unity Catalog** a common data catalog. Compliance to organisational rules and industry regulations is ensured via **Federated Governance.**

With a Lakehouse Platform, data domains can be created on different levels:

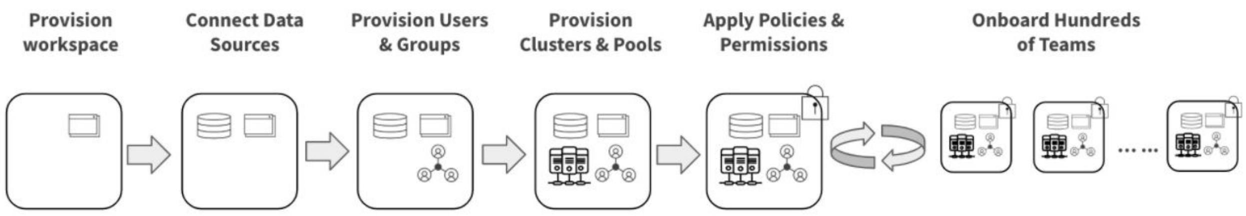
- One Workspace using clusters to isolate domains
- Using a separate Workspace per data domain
- Data domains being full Lakehouses



General CI/CD Workflow on Databricks (Azure DevOps)

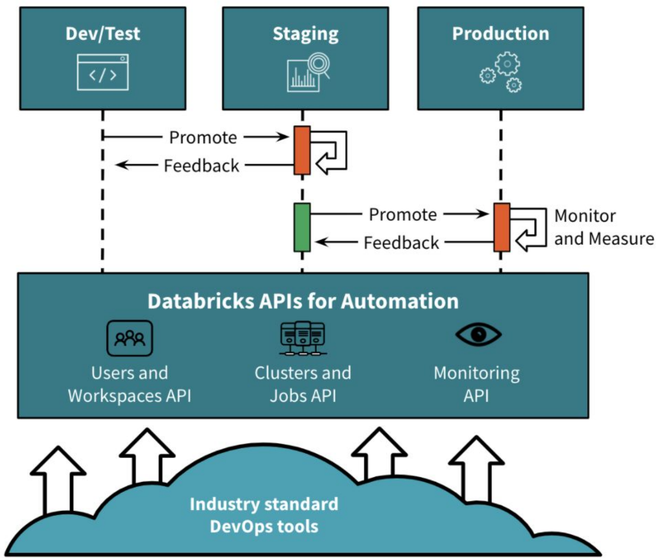


Automation



Fully configured data environments

- REST APIs
- Source Control
- CI/CD server
- Scripting with cloud formation, terraform
  - Deploy workspace
  - Connect data sources
  - Provision users and groups
  - Create clusters and cluster policies
  - Add permissions for users and groups
  - Blog
  - Blog
- Clusters
  - Automated Job Cluster
  - Interactive All-purpose Clusters

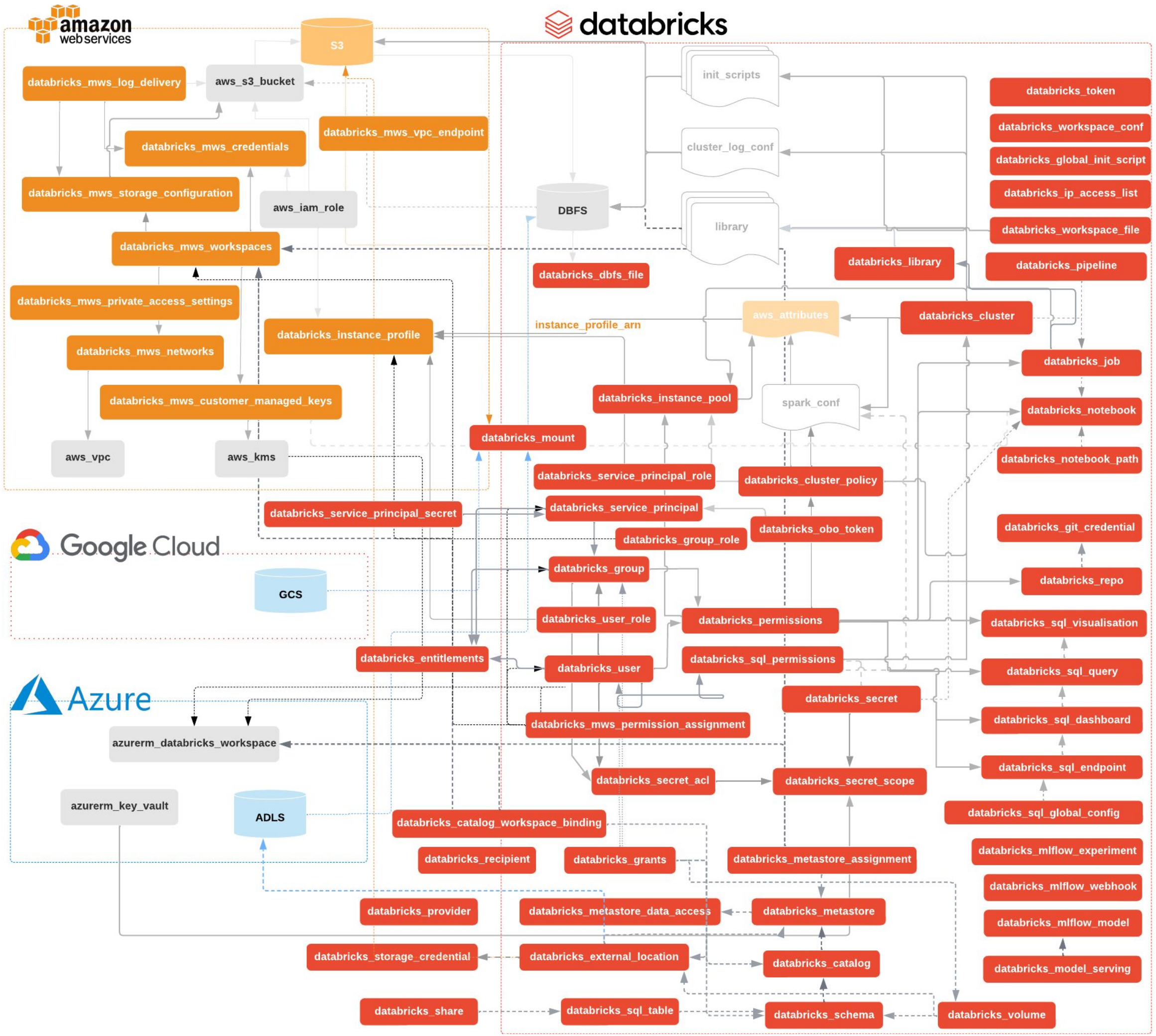


Databricks automation at scale

Terraform Provider

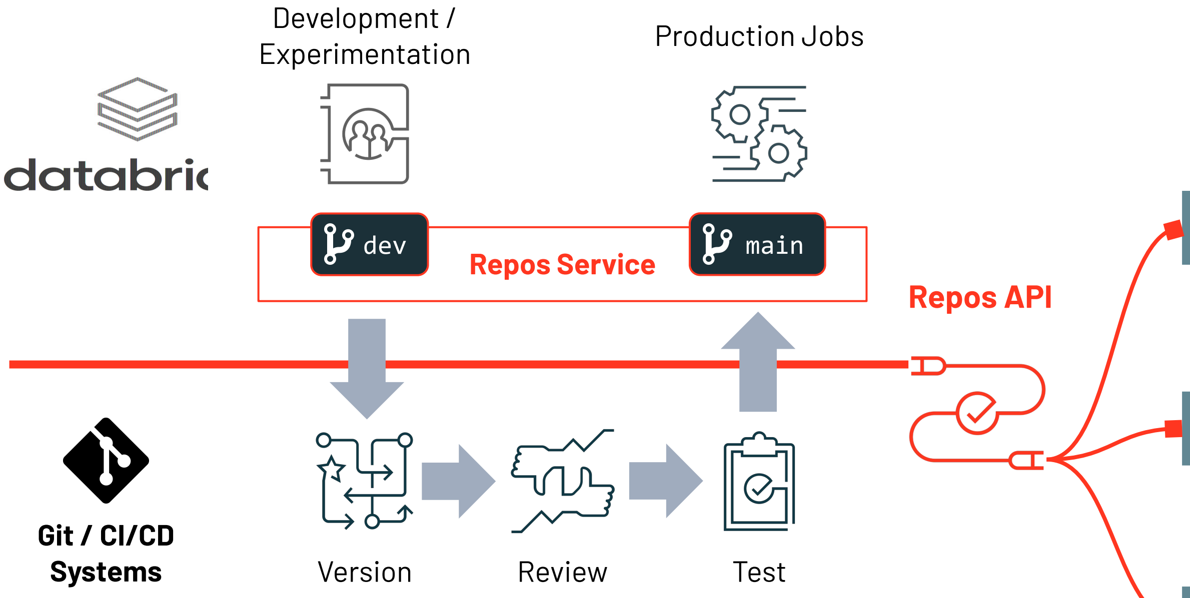
Cloud Agnostic
Code Reuse
Repeatability
Predictable
Supports all REST APIs

- Compute Resources
- Storage
- Security



Repos API and Git-based Jobs

Run production workloads on Databricks following good CI/CD practices



Programmatically manage Repos Roadmap

E.g., set up automation to automatically create Databricks Repos for all of your repositories

Perform Git operations on Repos In PREV

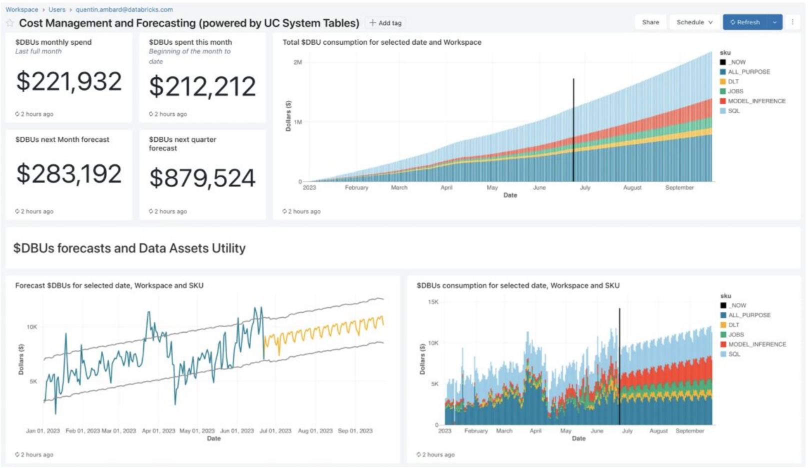
E.g., set up automation to always keep Repos in a production folder at a specified version

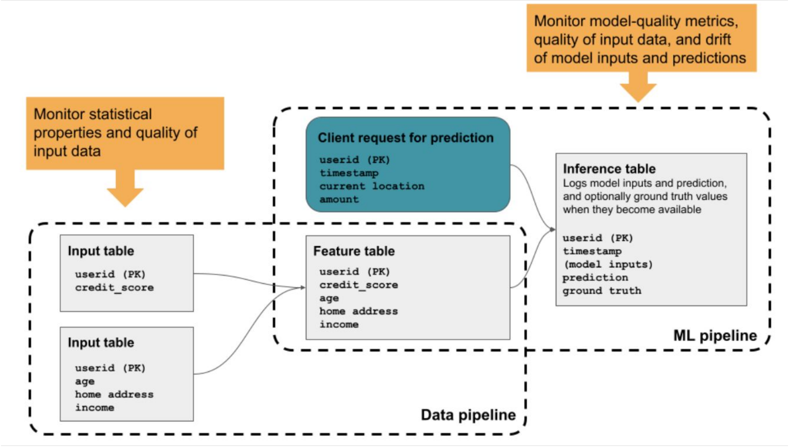
Specify Git branch, tag for Jobs Roadmap

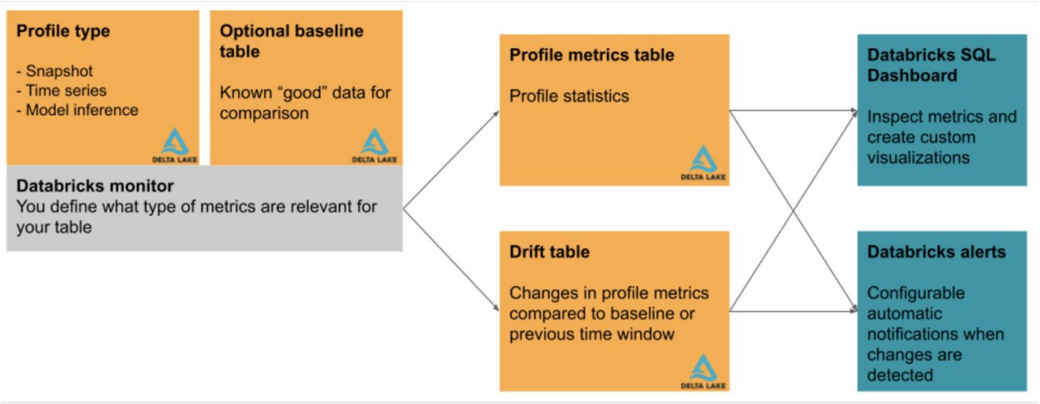
E.g., update the Repo to a specific branch or tag and run Jobs against the notebooks in the repo

Observability

- System Tables for operational intelligence
  - Monitor your **consumption** and leverage the lakehouse AI capabilities to forecast your future usage, triggering alerts when billing goes above your criterias
  - Monitor **accesses** to your data assets
  - Monitor and understand your **platform usage**
  - [Db demos]







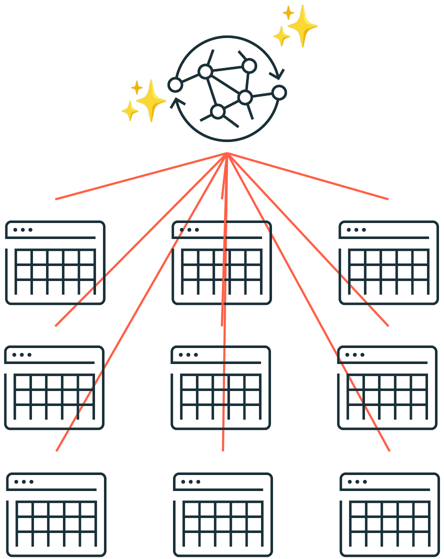
Data teams have SLAs on **delivery** not **quality**.

- **Reactive issue management** : data consumers encounter issues with data before the data team
- **Bottlenecked operations** : data consumers don't have self-serve experiences and must go through data team to add or use data
- **Long Root-Cause-Analysis** : it can take weeks to debug quality issues and rollback changes

Data Quality Monitoring Features

Anomaly Detection

- Enabled at the **schema level**
- ✨ Intelligently detects data quality anomalies for all tables
- Meant for: Scalable quality monitoring and actionable insights



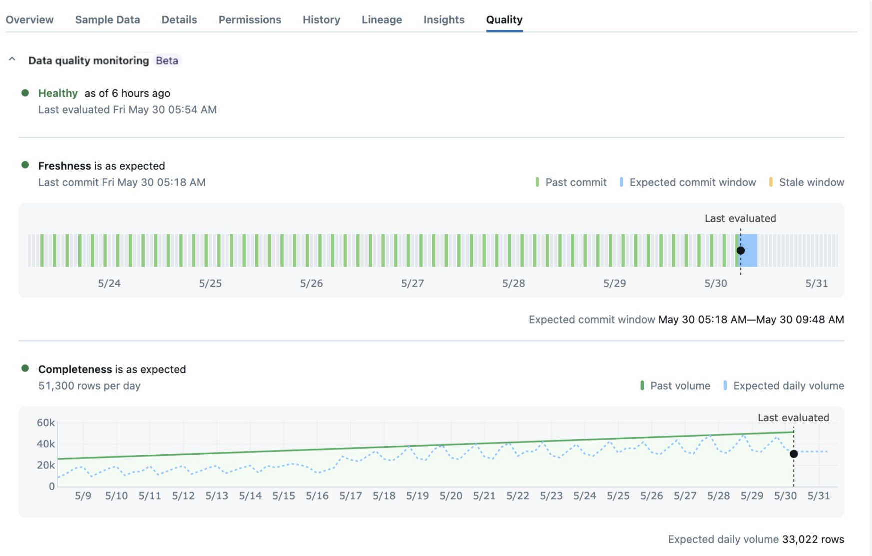
Data profiling

- Enabled at the **table level, f.k.a Lakehouse Monitoring**
- 📊 Provides a table profile for your most important tables
- Meant for: DIY table monitoring and quick summary statistics

AI-Powered

Intelligent and built on your data

- **Freshness:** tracks how recently data was updated, flagging tables as stale if updates are delayed.
- **Completeness:** checks if row volume in the last 24 hours falls below expected levels.
- **Segmentation:** View quality metrics by slice (e.g., freshness of vendor_id=”walmart”)

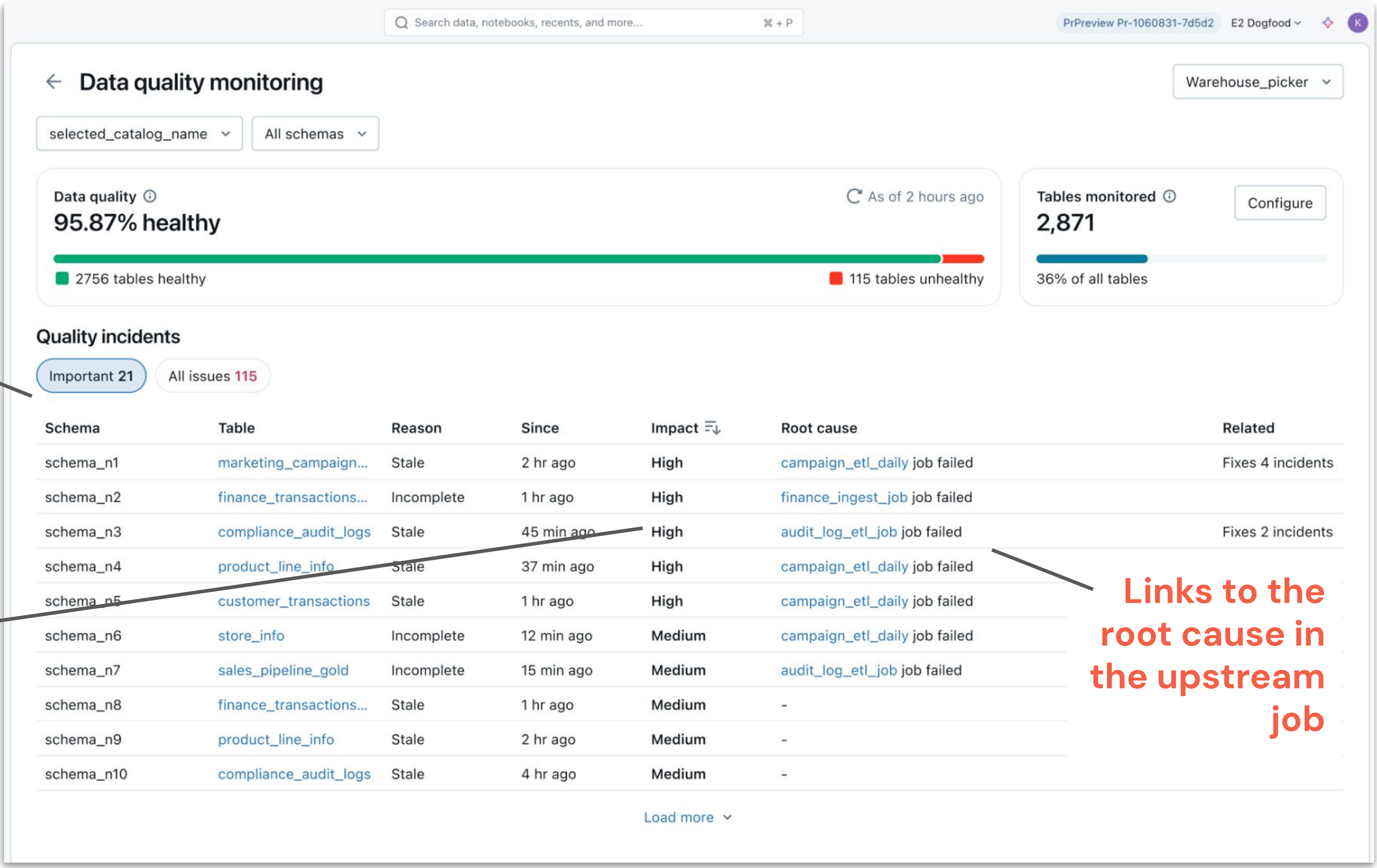


~~Bespoke rules per table~~

Simple and scalable to all tables

Holistic view across all tables

Incidents sorted by downstream impact



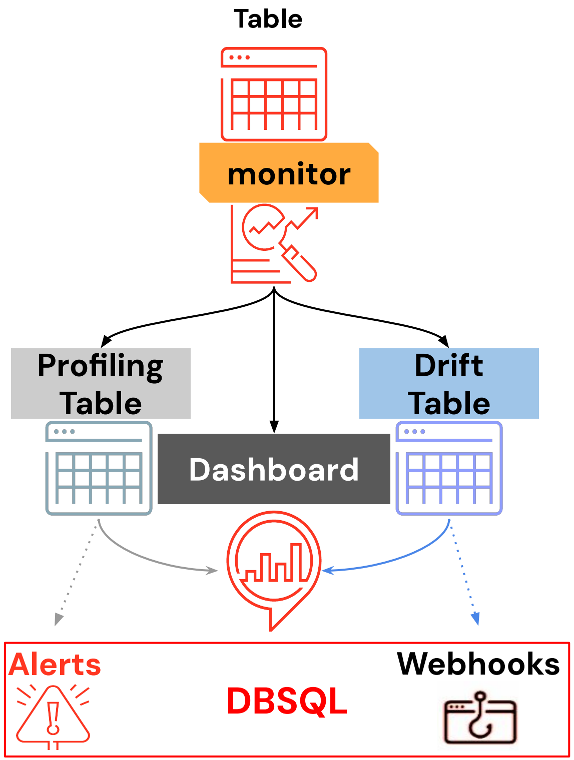
Links to the root cause in the upstream job

Whether data comes from data ingestion or model outputs,

data manifests as **tables.**

Monitoring a table in the Lakehouse

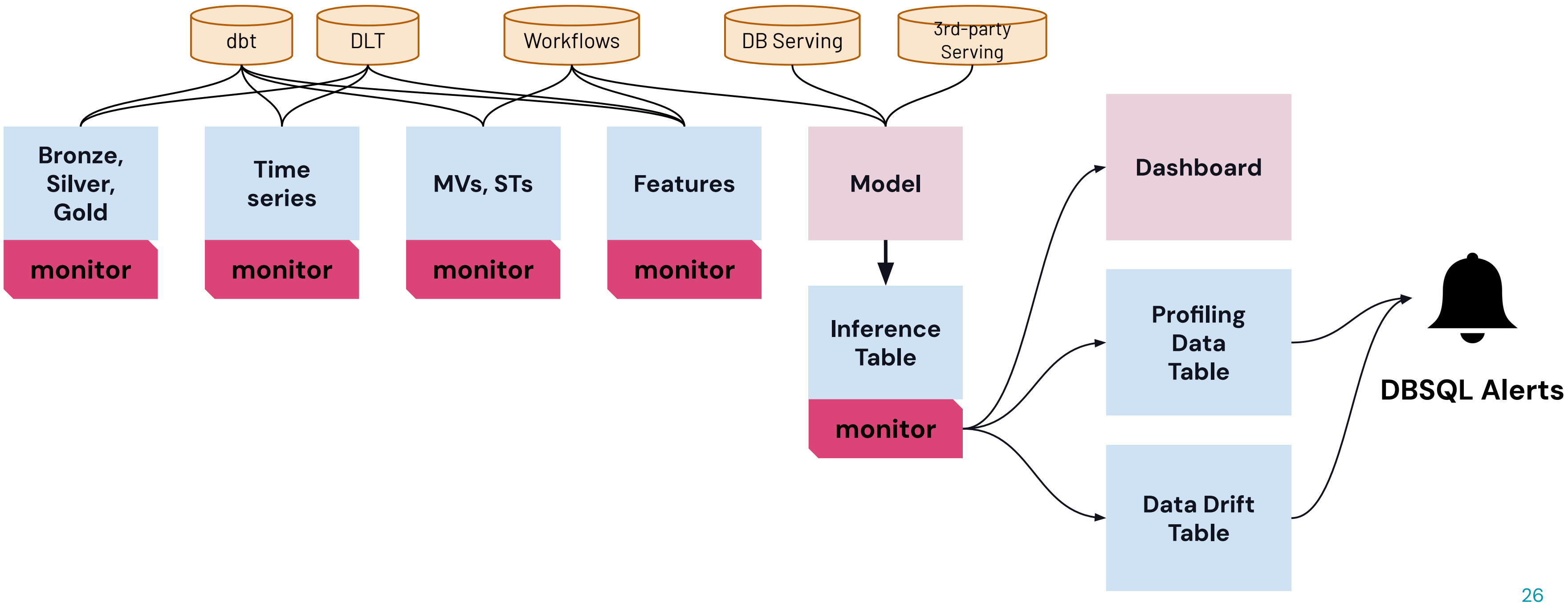
How does it work?



*f.k.a Lakehouse Monitoring

Data Profiling*

Automated profiling of your most important tables



DQX - Data Quality Framework

Provided by Databricks Labs

DQX is a data quality framework for Apache Spark that enables you to define, monitor, and address data quality issues in your Python-based data pipelines.

Motivation | Installation | User guide | Demos | Reference