Databricks- CICD & DevOps for Data Engineering

Shivam Panicker,
Senior Specialist Solutions Architect, APJ
Databricks Inc.

databricks

Agenda

01 About CI & CD

02 Why do we need it?

03 About DevOps

04 DE with Databricks

05 Notebook based CICD

06 IDE Based CICD

07 Lab

08 Summary

09 Appendix

Objectives

- At the end of the course, you will be familiar with the concepts of CICD & Devops.
- You will have the understanding of how Databricks can be leveraged to build scalable solutions.
- Will be able to apply knowledge hands-on in developing CICD & DevOps solutions with Databricks.

Definitions

- CI- Continuous Integration is the practice of merging all developers' working copies to a shared mainline several times a day that triggers automated build with testing.

- CD- Continuous Delivery is an approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time and, following a pipeline through a "production-like environment", without doing so manually.

- DevOps- is a set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality.

About CI & CD

Continuous Integration is the process of setting up automated practices write from the scratch to develop code with the focus on testing. It intends to automate test from the lowest level of individual code development (unit testing) to integration of various modules (integration testing) to the final product (system testing)

Continuous Delivery is the process of automating & building error free artefacts and deploying them into different environments relevant to the stage of the development life cycle. It uses the modern generation of tools which simplifies code management, running different cycles of testing.

Why do we need it?

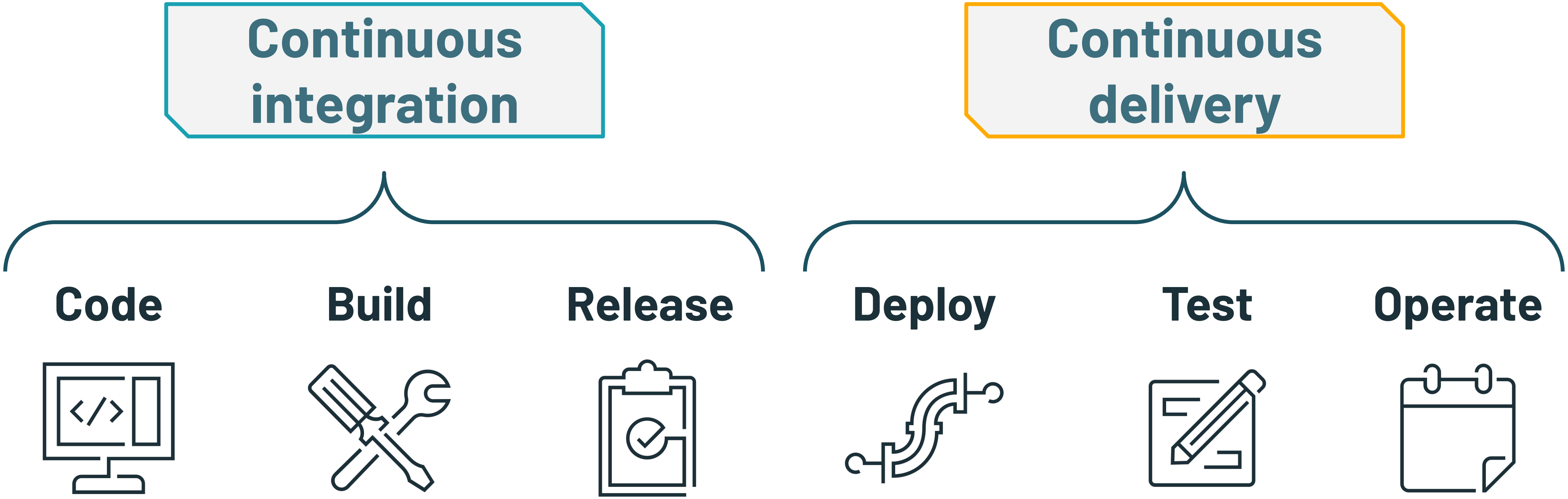
Continuous Integration

- Improved quality of code.
- Efficient bug tracking and fixes.
- Streamlined and expedited review process.
- Effective communication, collaboration and feedback loops.

Continuous Delivery

- Measurable progress.
- Scalable code deployed into production.
- Reduced manual overheads and thus, lesser human error prone deployments.

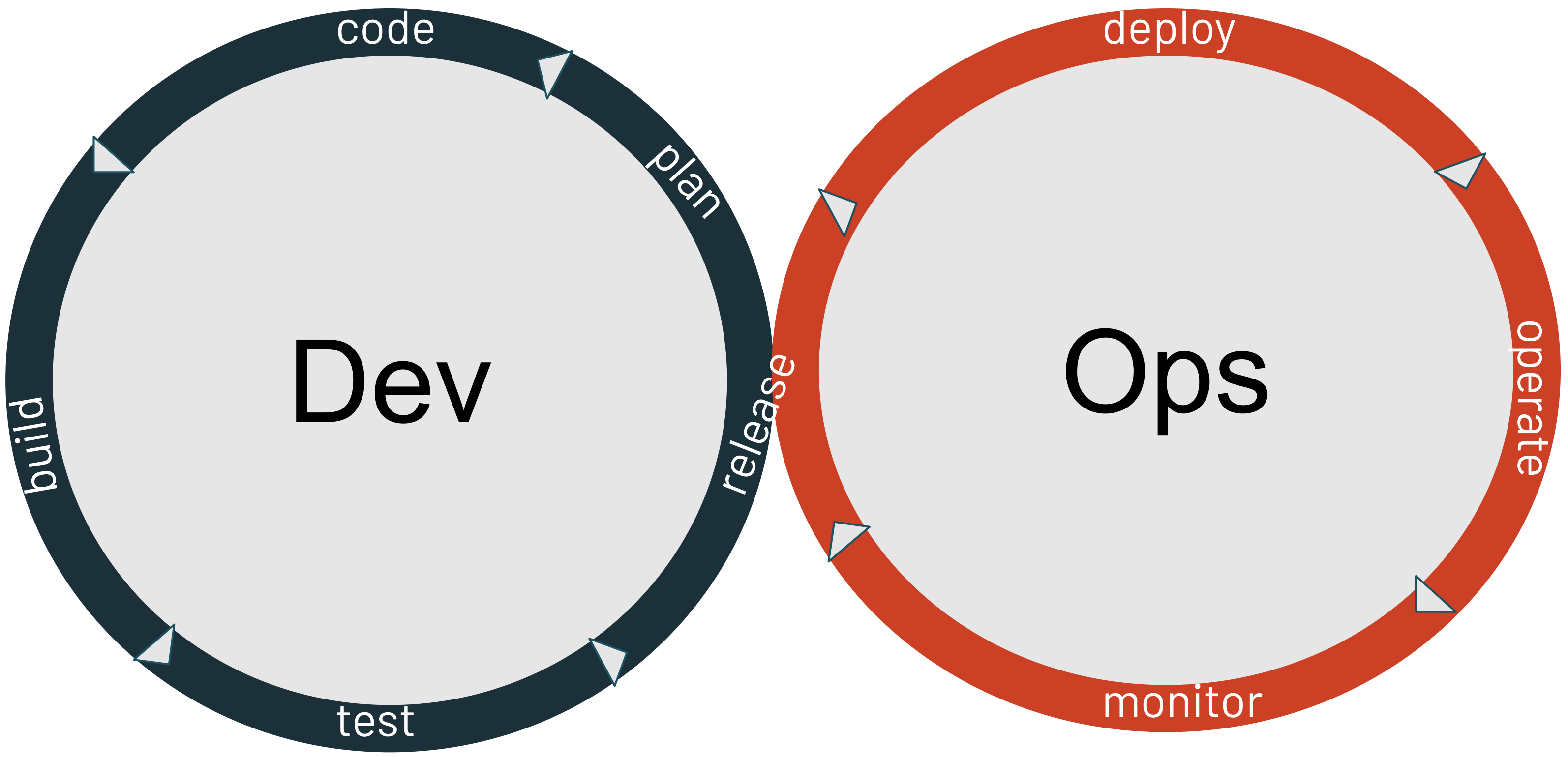
Overview of a typical Databricks CI/CD pipeline



About DevOps

Devops is the process of combining the practices of software engineering and IT operations together to deliver a tangible solution in a faster iteration of time. It enables different personas to work together and automate the software development life cycle with the use of the right tools that can enable Continuous Integration & Continuous development.

DevOps Lifecycle



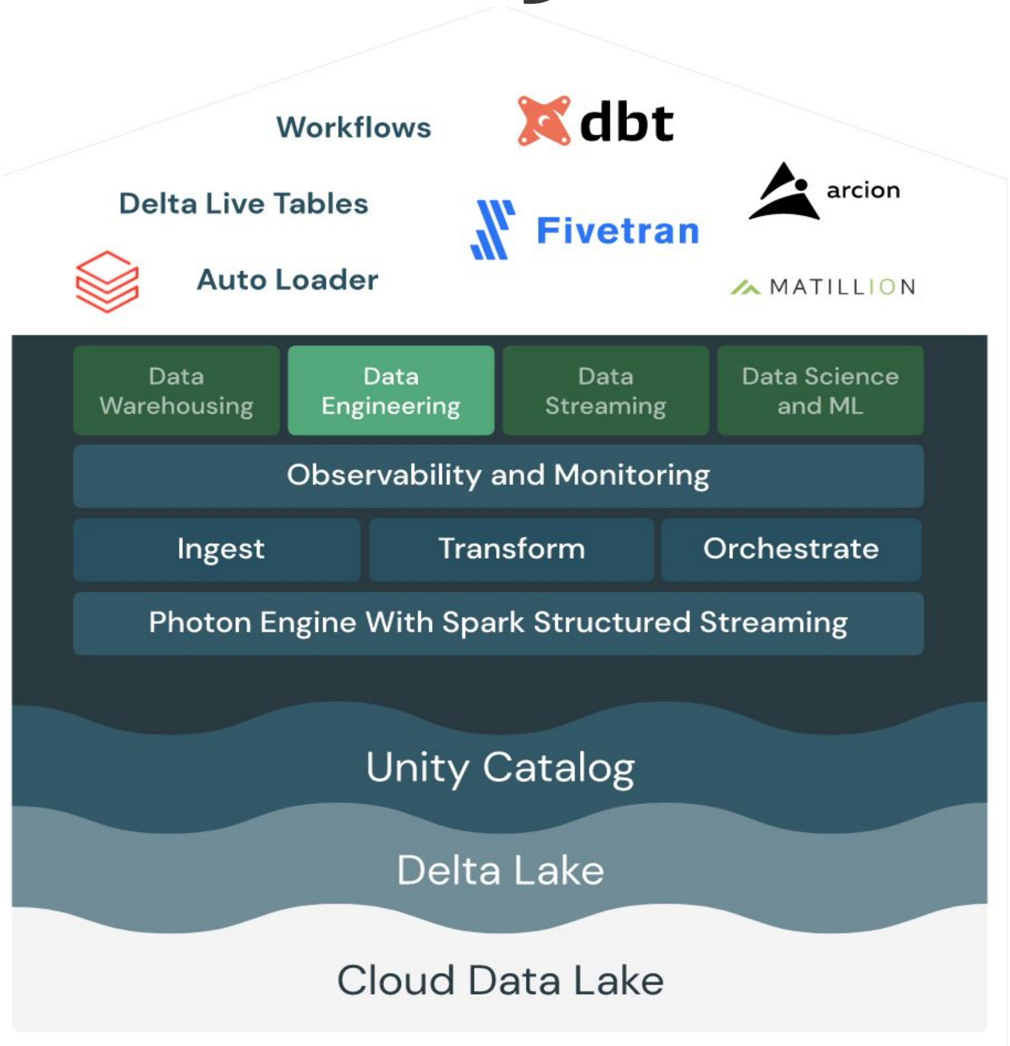
Data Engineering with Databricks

Databricks allows customers to engineer the data pipelines in various ways. Databricks powers the data engineers to develop data pipelines using sql, python, scala and R.

It allows following options to develop the pipelines:

- Notebook driven development,
- IDE based development, and
- Databricks SQL

Data Engineering with Databricks



Simple

Unify your data warehousing and AI use cases on a single platform

Multicloud

One consistent data platform across clouds

Open

Built on open source and open standards

Workloads on Databricks

- Data orchestration through Databricks Workflows

- Delta Live Tables manage your full data pipelines

- Simplifies data engineering with a curated data lake approach through Delta Lake

Thrives within your modern data stack

BI and Dashboards
Power BI, tableau, Looker, MicroStrategy, ThoughtSpot, Qlik

Machine Learning
MathWorks, Labelbox, John Snow LABS, Azure Machine Learning, H2O.ai, Amazon SageMaker

Data Science
PyCharm, jupyter, HEX, RStudio

Data Governance
Collibra, IMMUTA, PRIVACERA, Quest, Alation, AZURE PURVIEW

Data Pipelines
dbt Labs, MATILLION, Azure Data Factory, Informatica, Prophecy

Data Ingestion
Fivetran, arcion, CONFLUENT, Rivery, Airbyte, Qlik

Data Warehousing | Data Engineering | Data Streaming | Data Science and ML

Unity Catalog

Delta Lake

Cloud Data Lake
Microsoft Azure, aws, Google Cloud

Consulting & SI Partners
accenture, avanade, Capgemini, Cognizant, Deloitte., slalom

SQL workloads on Databricks

- Great performance and concurrency for BI and SQL workloads on Delta Lake
- Native SQL interface for analysts
- Support for BI tools to directly query your most recent data in Delta Lake

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

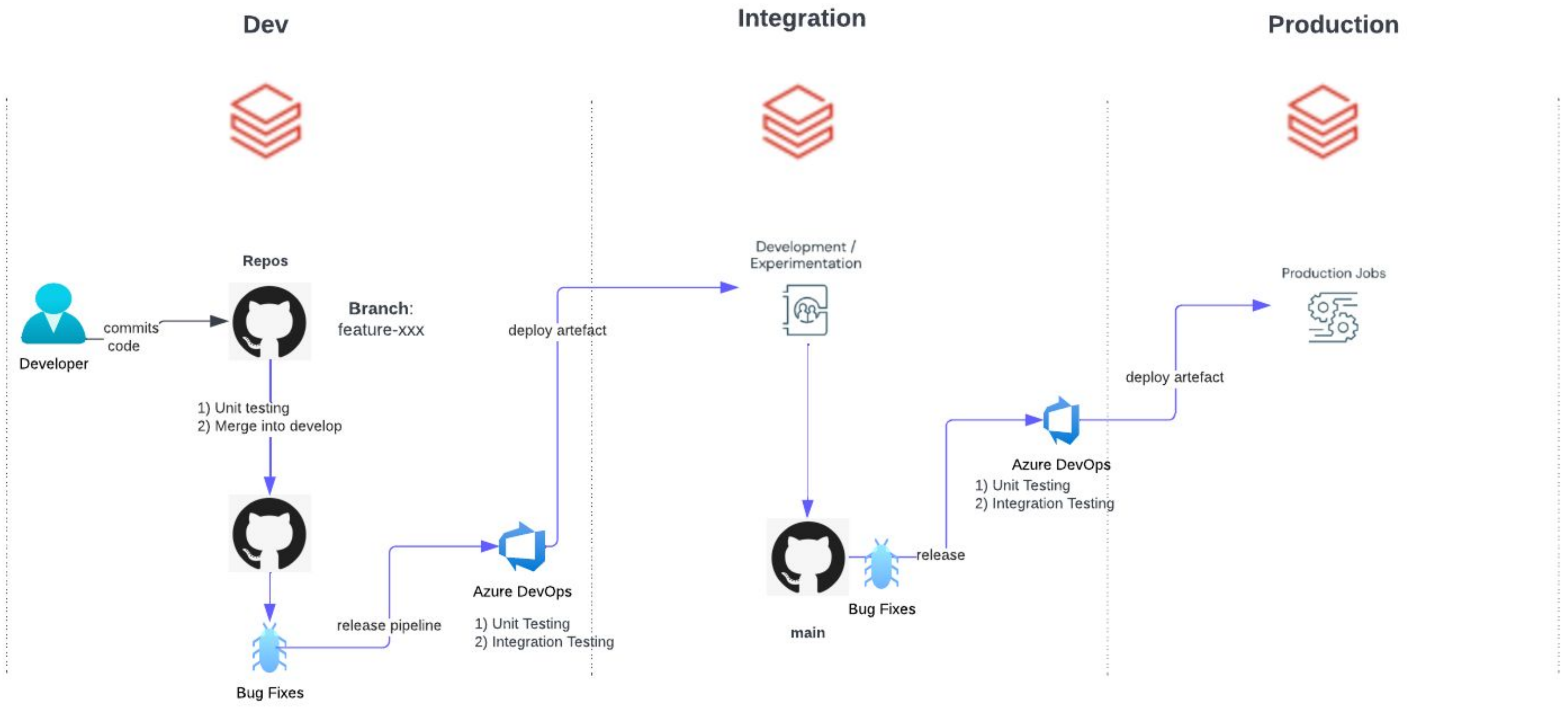
CI/CD for Databricks using Notebooks

databricks

General CI/CD workflow on Databricks

- Develop the code using one of the workflows (notebooks/IDE/mixed)
- Pull notebooks & commit them (dbx/databricks-cli/stack-cli)
- Build assets via build pipeline & perform unit testing (Github Actions, Azure Devops, Jenkins, etc)
- Push notebooks to staging (databricks-cli/stack-cli)
- Push assets to staging (databricks-cli, terraform)
- Run tests (unit/integration/e2e) on staging (dbx, 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)

Notebooks based CICD- Developer Workflow



Writing the testable code

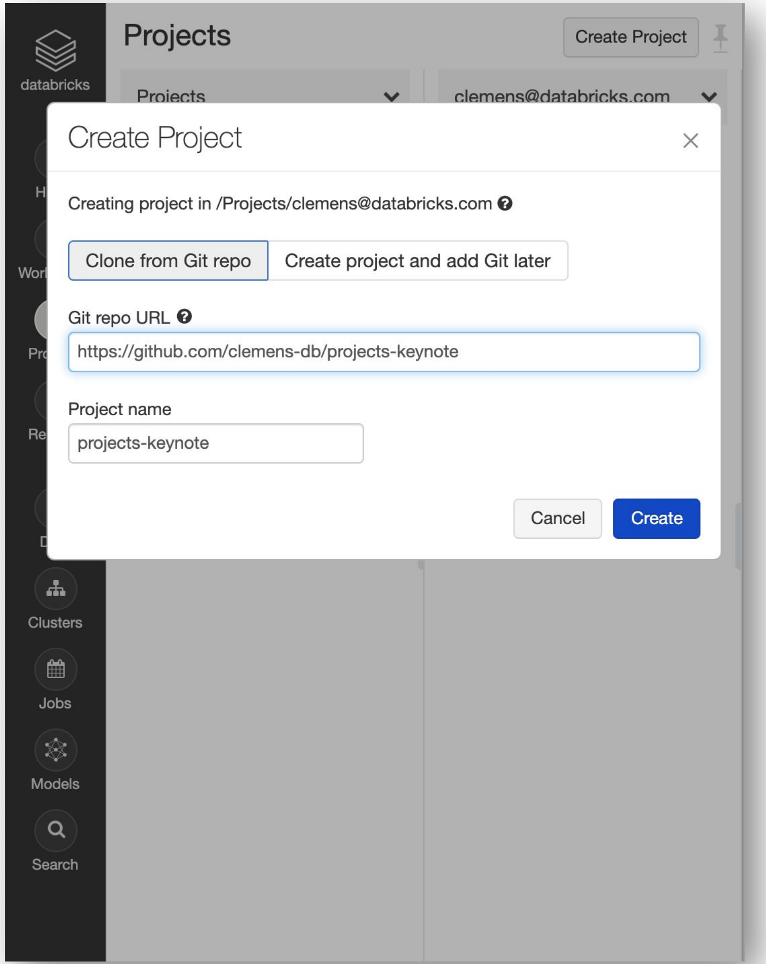
- Split code into testable chunks
  - Individual functions: DataFrame+Configuration in, DataFrame out
  - Business logic – functions, calling other functions
  - No dependency onto the global state/external systems
- Unit tests
  - Test only one specific function, on small amount of data
  - Fast, able 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

Testing libraries for Spark

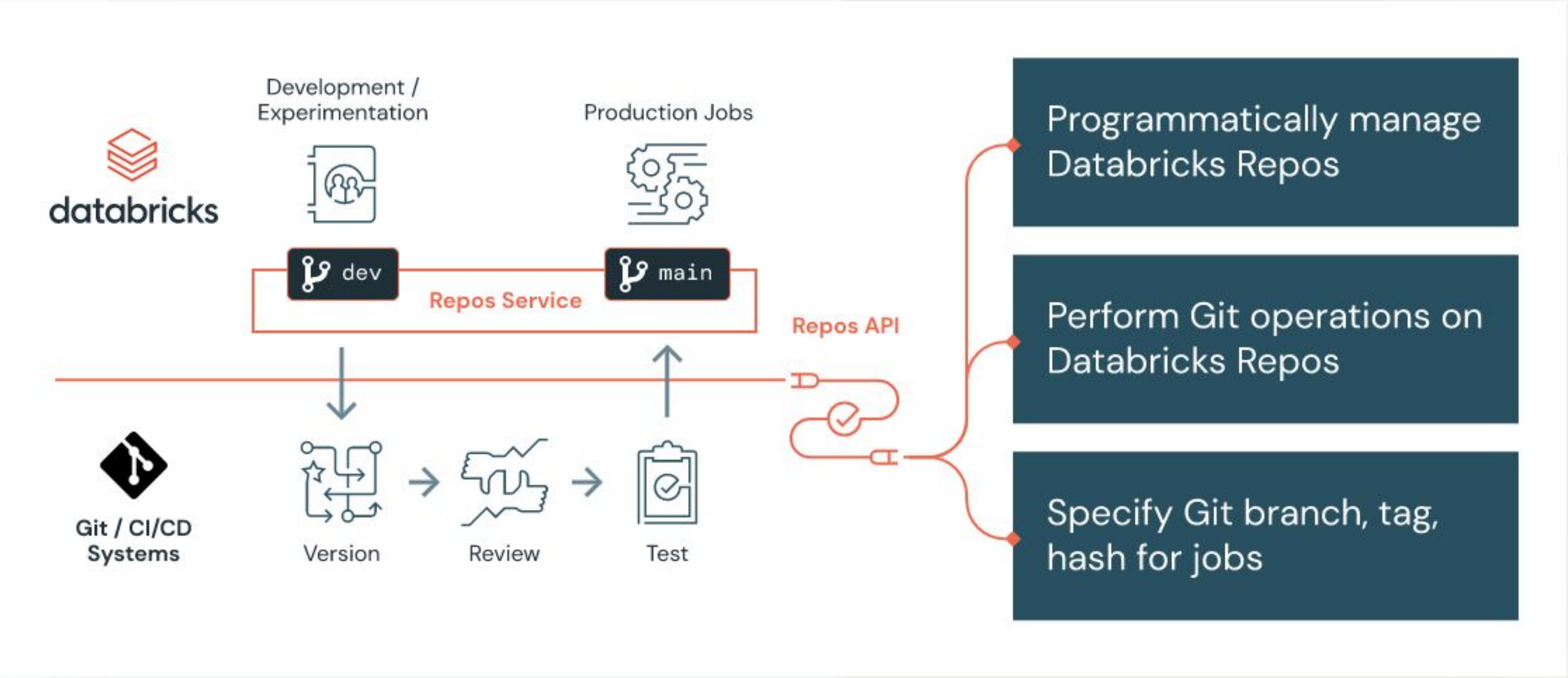
- Built-in Spark test suite
  - Designed to test all parts of Spark
  - Supports RDD, Dataframe/Dataset, Streaming APIs
- spark-testing-base:
  - Scala & Python support
  - Supports RDD, Dataframe/Dataset, Streaming APIs
- spark-fast-tests – Scala, Spark 2 & 3
- chispa – Python version of spark-fast-tests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

- Code samples for all libraries in one place

Git-based Repos in Databricks



CI/CD Integration

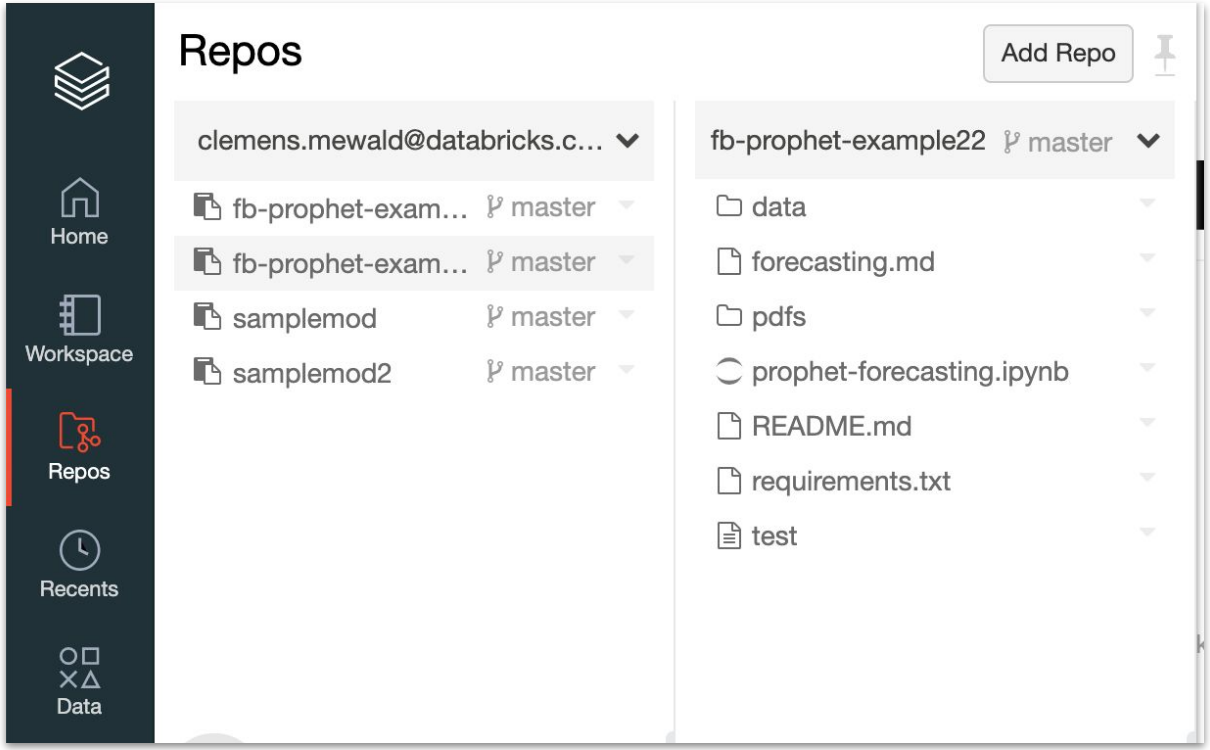


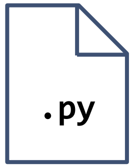
Supported Git Providers

Azure DevOps GitHub GitLab Bitbucket

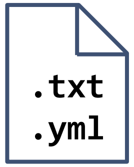
Arbitrary files support in Repos

Seamlessly bring any type of workload to Databricks

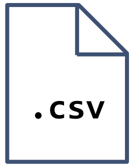




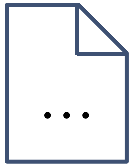
- Portability of code
- Library files – use Python/R files as packages



- Environment specification portability
- Build packages from the same repo



- Small data ease of use
- Relative imports



- Whatever you can do with files “just works”

Using arbitrary files

Package with library functions

dlt-best-practices main

- .gitignore
- conftest.py
- dlt_package
- my_package
- notebooks
- pipelines
- pytest.ini
- README.md
- requirements.txt
- scripts
- setup.py
- tests
- unit-requirements.txt

my_package

- __init__.py
- __version__.py
- code1.py
- functions.py
- code2.py

Automatically import changes

Cmd 2

```
%load_ext autoreload
%autoreload 2
```

Command took 0.20 seconds -- by alexey.ott@databricks.com at 27/01/2022, 17:46:05 on NutterDemo

Cmd 3

```
from my_package.code1 import * # instead of %run ./Code1
from my_package.code2 import * # instead of %run ./Code2
```

Command took 1.14 seconds -- by alexey.ott@databricks.com at 27/01/2022, 17:46:17 on NutterDemo

It's possible to build packages from the same source

Import library functions

Testing notebooks: using %run

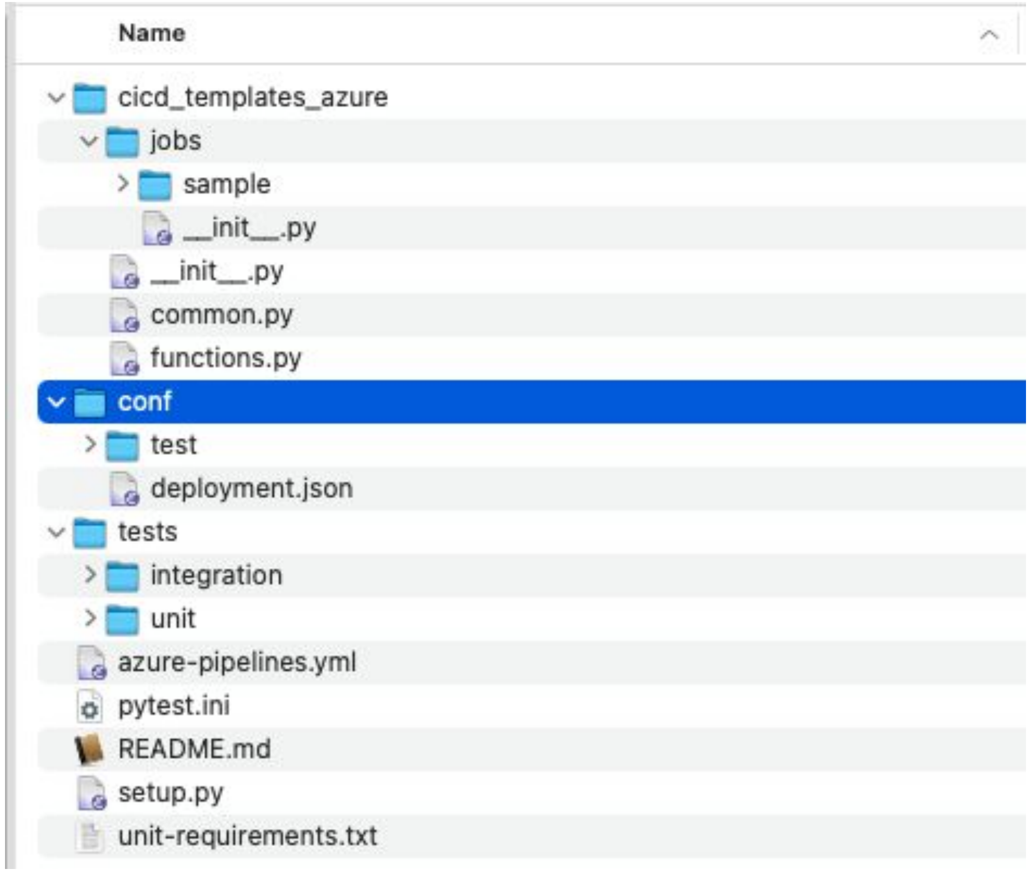
- Workflow:
  - Create separate notebook with functions that you want to test
  - Create separate notebook for tests
  - Import defined functions from notebook **%run**
  - Depending on the language, defined test functions or test classes
  - Execute tests
- This workflow may require special approach 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

CI/CD for Databricks using IDE

databricks

Code organization

- Organize code into testable chunks:
  - Library functions
  - Jobs – doing actual data processing
- Unit tests for library functions
  - Use specialized testing frameworks
- Integration tests for jobs
- Use dbx init to generate a skeleton of Python project



Development flow

- Make changes using the IDE of choice
- Run unit tests locally (see examples), debug errors
- Commit the code into a repository
- CI/CD process pickups changes & performs:
  - Run unit tests
  - Run integration tests on Databricks
  - Publish test results
  - Promote changes to other environments if necessary
- Use dbx & other tooling to implement CI/CD pipelines

CI/CD integrations for Databricks

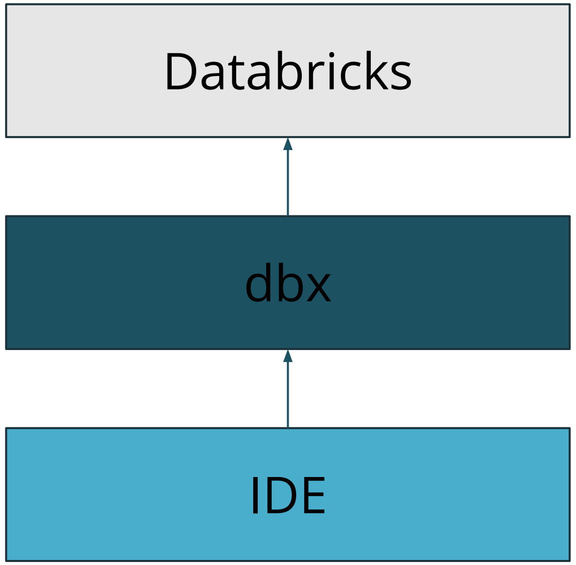
Existing integrations

- Azure DevOps
  - Use dbx init 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 dbx init to generate pre-configured project template

DBX

databricks

dbx: extended support for IDE development



dbx is an extension of the Databricks CLI that makes it **simple** to:

- Build, run, and test your code on Databricks from your local IDE
- Quickly iterate on your project while retaining the efficiency of using an IDE for modularity and testing
- Nimbly manage multiple execution environments and deployment configurations
- Get access to the full feature set in Databricks Runtime via holistic, batch execution (e.g., Photon, Unity Catalog, Feature Store, AutoML)

dbx Powerful command line tooling

- Build, run and test:

  > **dbx configure** Configure your project to run against Databricks
  > **dbx deploy** Deploy a Databricks Workflow as scripts or a versioned build artifact
  > **dbx launch** Trigger the Workflow locally and check the results on Databricks

- Quickly iterate

  > **dbx execute** Submit code directly on All-Purpose Compute
  > **dbx sync** Replicate local changes to Databricks Repos

- Bootstrap your project

  > **dbx init** Initialize a template for a Python package with tests, CI/CD, and more

dbx Configuration driven deployment

- Supports deploying single or multi-task workflows
  - Define tasks and execution environments in flexible YAML or JSON
    - Databricks workspace(s) to run in
    - Compute resources and library dependencies
- Ship code to different environments from the terminal
  - Develop against dev:
    > dbx execute --environment=dev model-train
  - Test in staging:
    > dbx deploy --environment=staging model-tests
    > dbx launch --environment=staging model-tests
  - Deploy to production:
    > dbx deploy --environment=prod model-train
    > dbx launch --environment=prod model-train

dbx

Minimal Project

```
.
├── .dbx
│   ├── lock.json
│   └── project.json
├── conf
│   ├── deployment.yml
├── my_script.py
```

⬅️ Databricks Workspace metadata

⬅️ Job configurations

⬅️ Your code

```
# run on All-Purpose Compute
> dbx execute --cluster-name='Shared Autoscaling' --no-rebuild --no-package my-job
```

Databricks CLI

databricks

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 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, …
- When in doubt, try databricks clusters --help

Library management

databricks

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 ADLS, S3 or GCS (depends on the DBR version, also need to have cloud storage authentication configured correctly)
- 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, …

Installing from private repositories

- Python, R:
  - Change global settings using [cluster or global init scripts]
- Private Maven repositories:
  - Before DBR 11.0 – no, because libraries were resolved in the control plane. Need to put to DBFS/ADLS, or use init script to install libraries
  - DBR 11+ – yes, libraries are now resolved locally on the cluster (still may need to use init scripts to update config files)

```
#!/bin/bash

cat << 'EOF' > /etc/pip.conf
[global]
timeout  = 60
index-url =
https://<host>/<path>/simple
trusted-host = <host>
EOF
```

Recommendations for building Scala/Java artifacts

- Generate uberjar (or fat jar) only with your code and internal dependencies (like, that are only available in the private Maven repositories)
- Don't include into uberjar following dependencies as will conflict with Databricks Runtime versions (mark them as provided):
  - Core Spark dependencies (spark-core, spark-sql, …) and spark-sql-kafka
  - Delta Lake
- Open source dependencies it's better not include into uberjar, but attach to clusters/declare in the jobs definitions (mark them as provided):
  - This decrease the size of the fat jar
  - It's easier to change them in case of new versions released, especially to fix vulnerabilities or other critical issues

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

Jobs Scheduling

Jobs Scheduling & orchestration

- Databricks Workflows: Built-in job scheduling (doc):
  - Periodic scheduling of the jobs (cron-like right now)
  - Execute notebook / jar / Python script / Spark-submit / DLT / DBSQL / DBT
- Apache Airflow
  - Airflow provider for Databricks
  - Execute notebook / jar / Python script / DLT pipeline as a job
  - Execute SQL queries against Databricks cluster / SQL Endpoint
- Azure Data Factory
  - Execute notebook / jar / Python script
  - Triggers: time-based, ADF Event-based

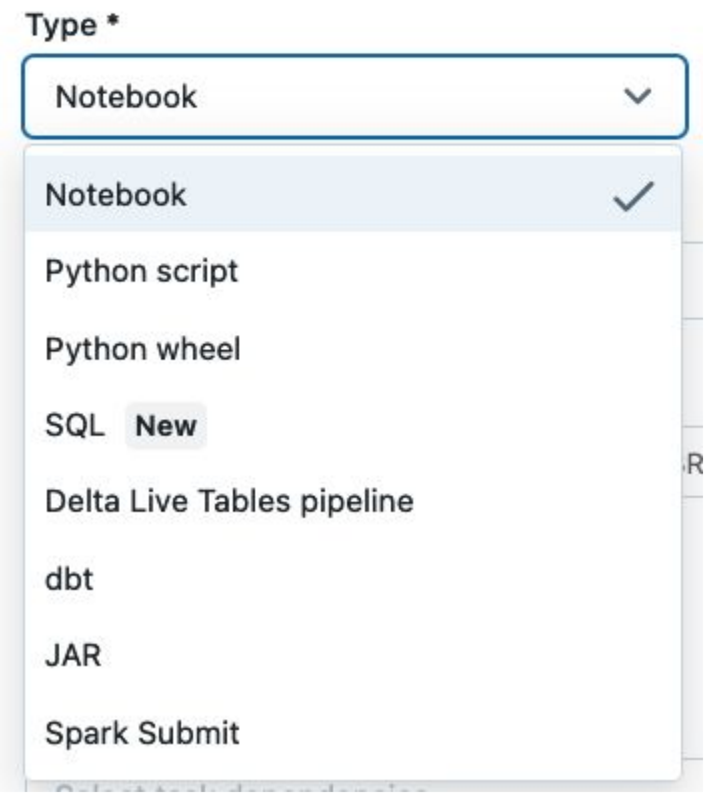
Databricks Workflows

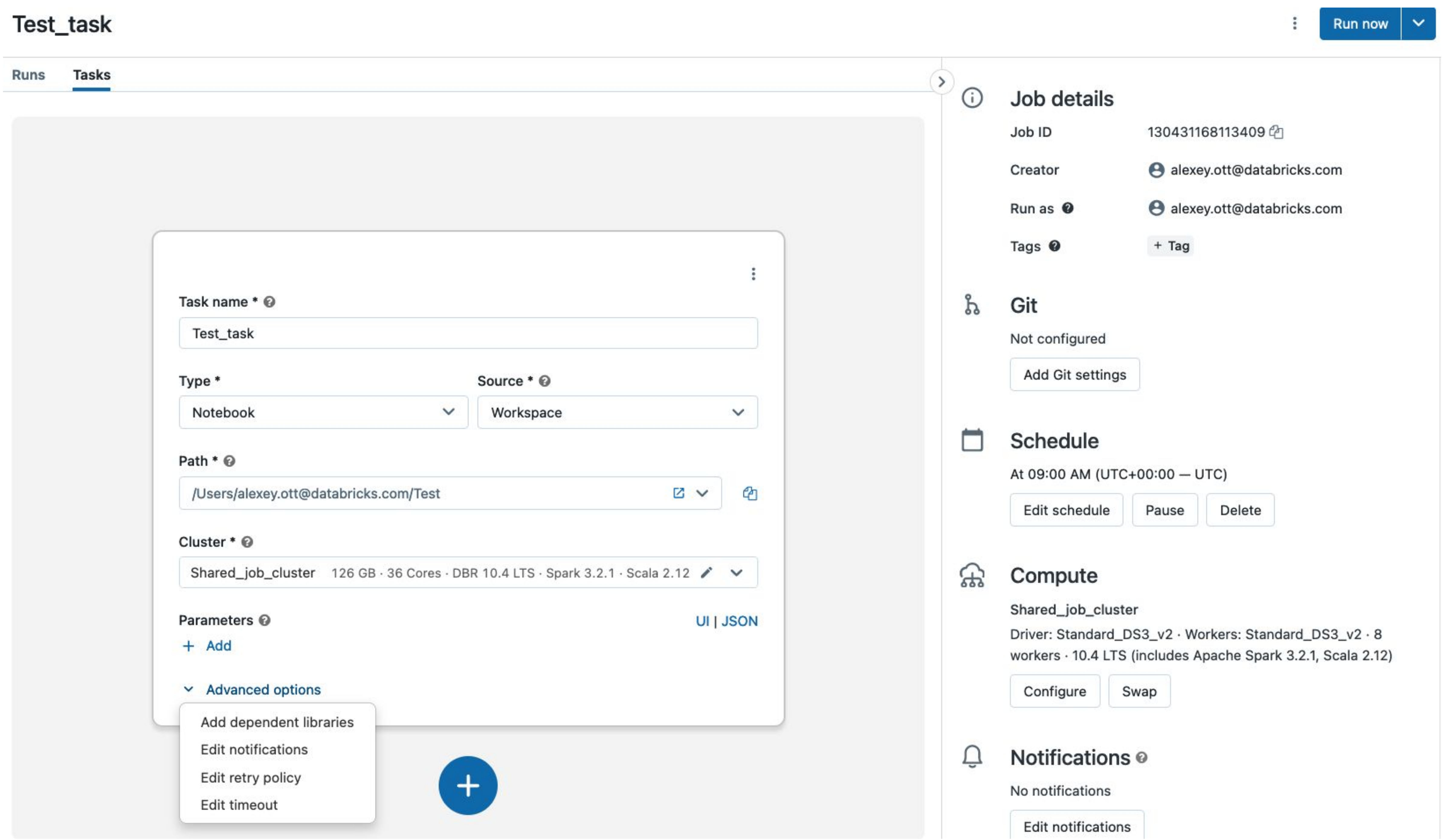
- Scheduled or executed one time
- Permanent (created via UI/REST API/Terraform) or ephemeral (submitted via REST API / Azure Data Factory / Airflow)
- Can run on ephemeral (cheaper) or existing clusters (more expensive)
- Multiple tasks in a job
- Jobs cluster reuse for permanent jobs!
- We can pass parameters to a job (use dbutils.widgets.get() to get them in notebook. In "normal" programs - they are just command-line parameters)

Task types in Databricks Workflows

We can execute:

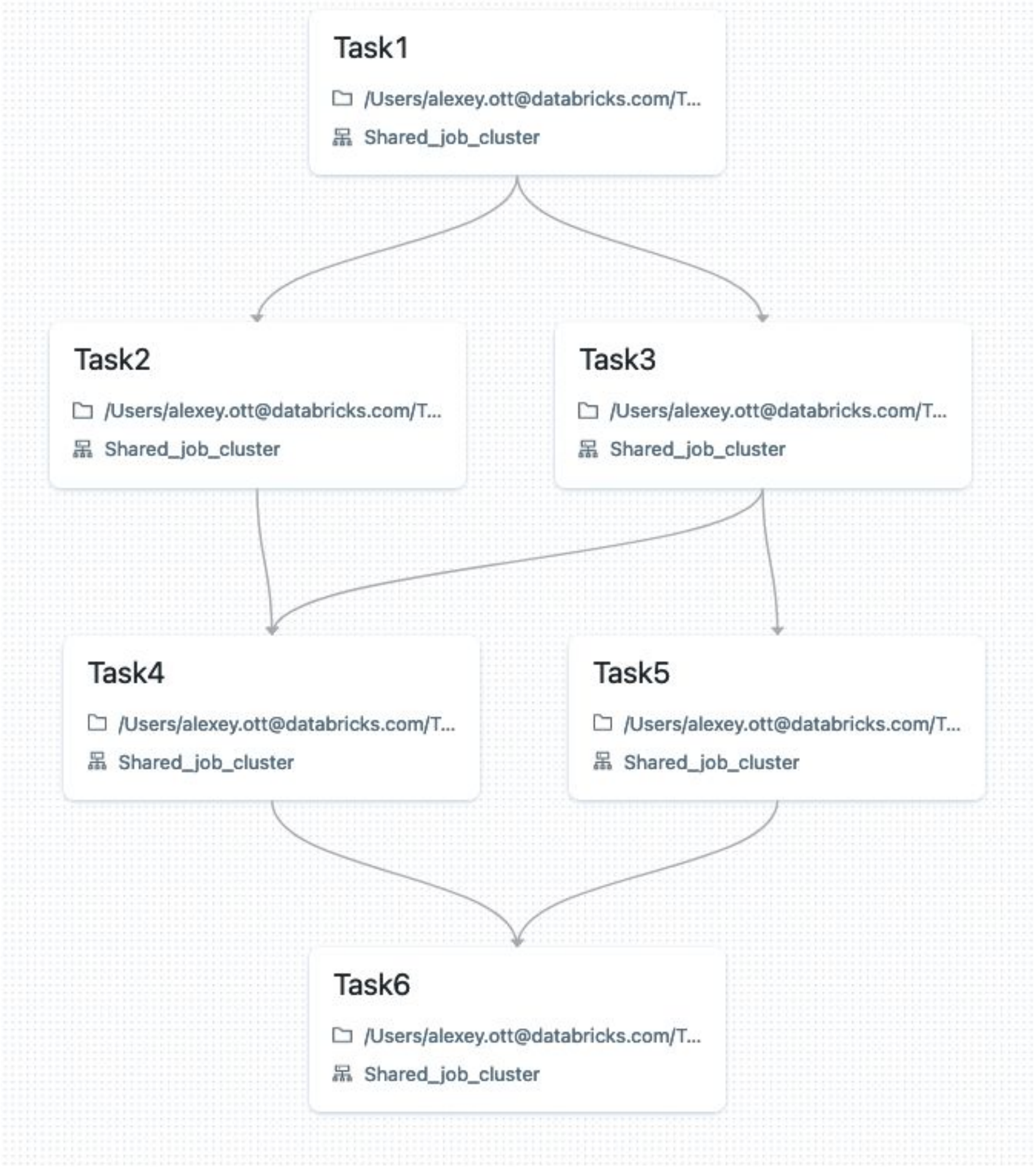
- Databricks notebooks
- Jar files
- Python code – source file & wheels
- Arbitrary code supported by spark-submit (for example, R code)
- DBSQL Dashboards/Queries/Alerts
- Delta Live Tables pipelines
- DBT




Test_task
Run now
Runs
Tasks
Task name *
Test_task
Type *
Notebook
Source *
Workspace
Path *
/Users/alexey.ott@databricks.com/Test
Cluster *
Shared_job_cluster 126 GB · 36 Cores · DBR 10.4 LTS · Spark 3.2.1 · Scala 2.12
Parameters
UI | JSON
+ Add
Advanced options
Add dependent libraries
Edit notifications
Edit retry policy
Edit timeout
Job details
Job ID
130431168113409
Creator
alexey.ott@databricks.com
Run as
alexey.ott@databricks.com
Tags
+ Tag
Git
Not configured
Add Git settings
Schedule
At 09:00 AM (UTC+00:00 — UTC)
Edit schedule
Pause
Delete
Compute
Shared_job_cluster
Driver: Standard_DS3_v2 · Workers: Standard_DS3_v2 · 8 workers · 10.4 LTS (includes Apache Spark 3.2.1, Scala 2.12)
Configure
Swap
Notifications
No notifications
Edit notifications

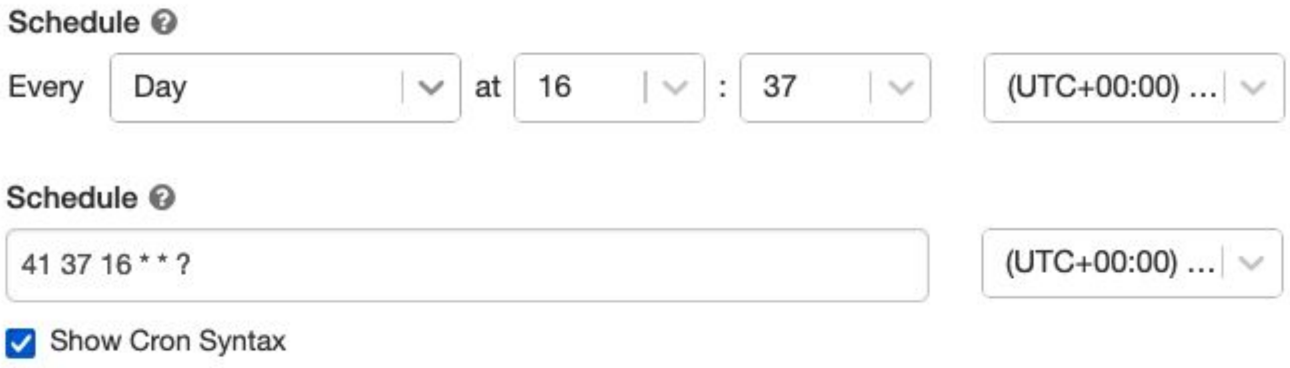
Multiple tasks in Databricks Workflows

- It's possible to specify multiple tasks in a single job
- Linear, or non-linear dependencies
- Job cluster(s) reuse – faster start times for next task(s)
- It's possible to pass values between tasks
- It's possible to rerun failed tasks (repair)
- Fully supported for automation via REST API / Terraform / Databricks CLI



Databricks Job Scheduling

- Create a job through the UI



- Create a job using the API (using cron
- Advanced job options
  - Alerts
  - Max concurrent runs
  - Timeout
  - Retries
  - Permissions

Summary

databricks

- We have covered the basic concepts of Continuous Integration (CI) & Continuous Development (CD) which helps in building reusable patterns of delivering code into production.

- Covered the basic elements of Data Engineering using Databricks and how notebooks and/or IDEs can be used for efficient coding.

- Use services such as Azure DevOps, Github Actions, etc to manage CICD and promote code into different environments.

- Databricks tooling such as DBX, Databricks-CLI to manage communication between Databricks APIs and other services such as Azure DevOps, Github Actions.

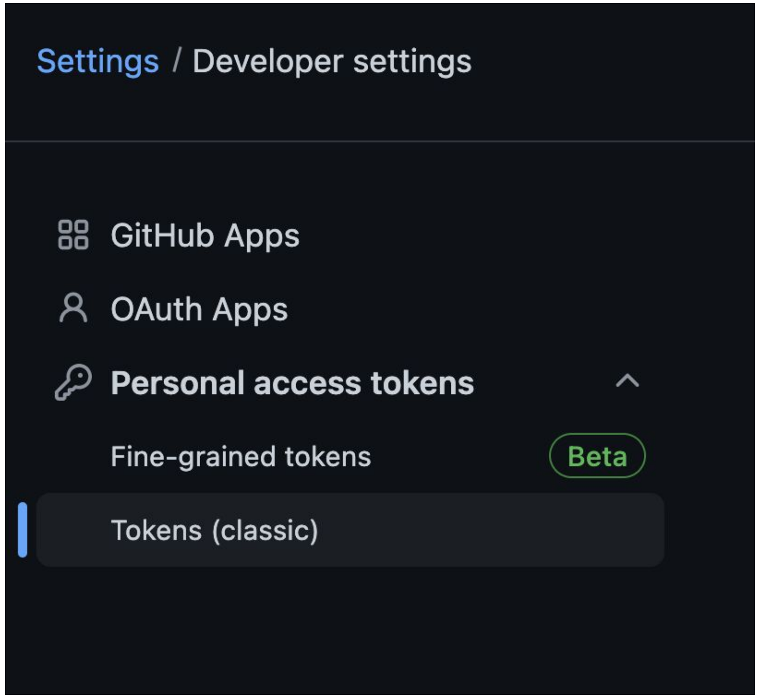
- Databricks Orchestration using native Workflows other external services such as Airflow, Azure Data Factory, etc.

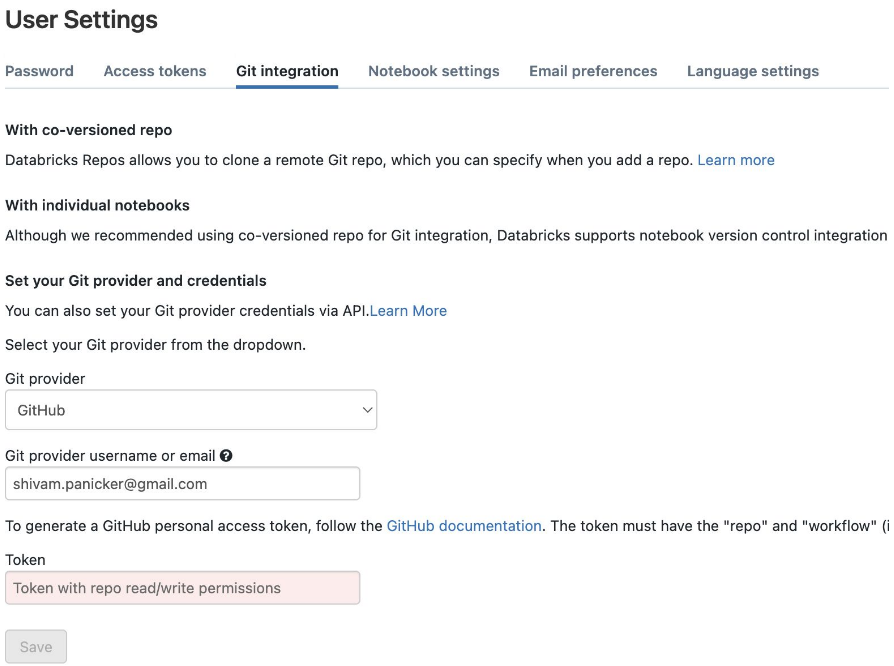
Labs

- As part of the lab today, we will be working with retail dataset
- Dataset consists of sales, stores, customer and product info
- The lab will focus on using Notebooks and perform unit testing on individual batch transformations. Further, we will run some integration tests to ensure nothing breaks upon running black box tests from bronze to silver to gold layers.
- Post the unit and integration tests, we shall create an artefact and deploy into production.

Setup

- Integrating github account with Databricks Repos
  - Github --> Settings --> Developer Settings --> Personal access tokens
    - Generate new token (classic) --> Enter password --> Enter note
    - Select repo and workflow scope --> Submit --> Copy token
  - Login to Databricks --> User Settings --> Git Integration
    - Git provider (Github) --> Git provider username or email (your git credentials) --> Token

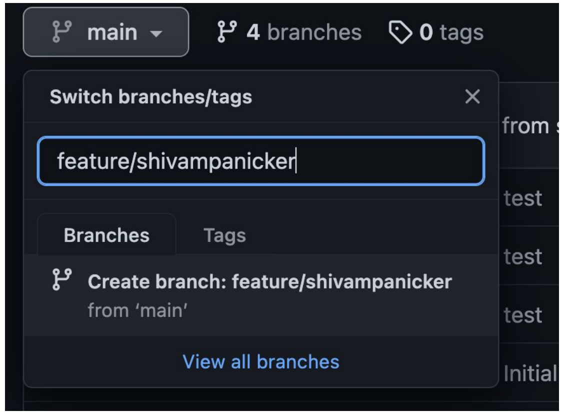




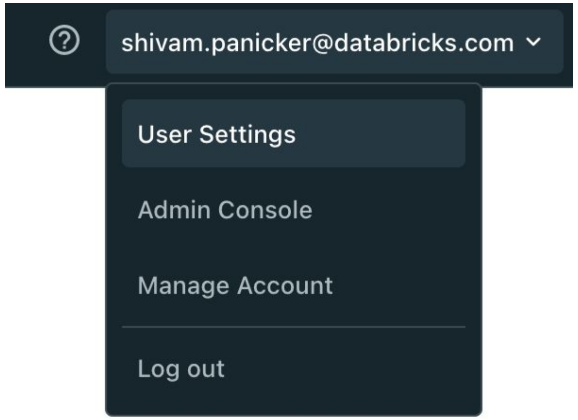
Setup

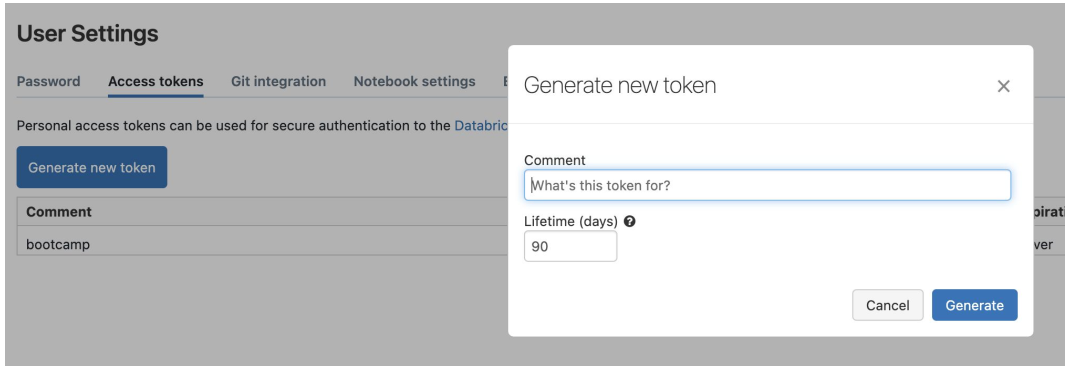
- Fork the repo- https://github.com/shivampanicker/cicd_with_databricks.git

- Create 2 branches
  - develop – on: main
  - feature/<username>: on develop



- Generate a personal access token and from Databricks Workspace and copy it in your notepad.

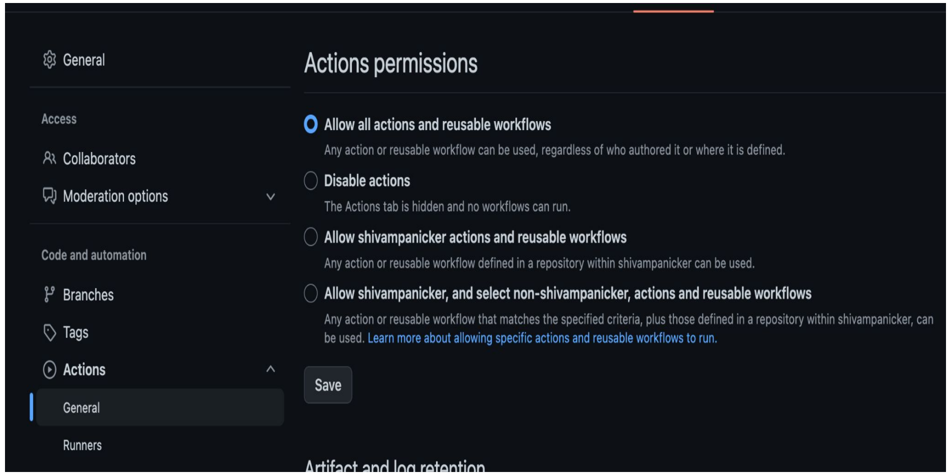




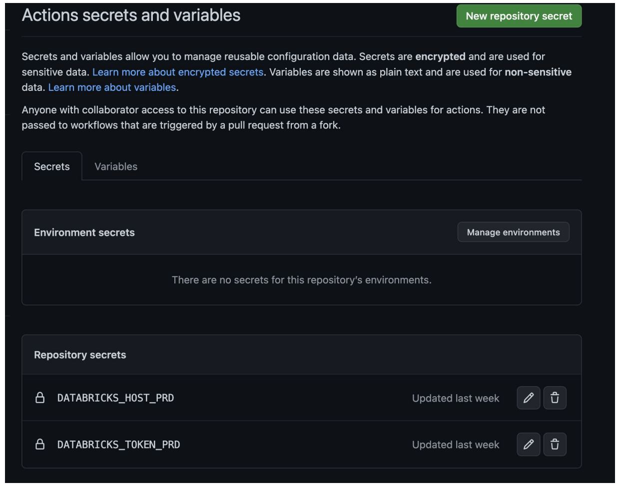
databricks

Setup

- Configure github actions (CICD)
  - Go to the forked repository.
  - Settings--> Actions
  - General --> Allow all Actions and reusable workflows



- Configure secrets
  - Go to Settings--> Secrets and Variables
  - Actions- Add new repository secret.
    - DATABRICKS_HOST_PRD https://<databricks_workspace_name>.com/
    - DATABRICKS_TOKEN_PRD <personal access token generated in DB>



databricks

Action time!

- Go to Databricks Repos and clone your repository.
- Checkout branch- *feature/<username>*
- There are few code level changes required before one raises pull request.
  - Navigate to this file: *src/main/tests/bronze/test_load_data_into_bronze*
  - Set expected_num_files = 2
  - Check the Silver layer unit tests in the *src/main/tests/silver/* folder and add an assertion. Feel free to write any test case.
- Review the integration_suite test which uses Files in Repos feature and fill in the missing elements.
  - src/main/python/gold/gold_layer_etl.py
  - src/main/tests/integration_suite/test_integration_gold_layer_etl
- Review the dbx deployment file
  - Open cicd_with_databricks/deployment/deploy-job.yaml and update notebook_path variable to your Databricks Repos location.

- Review files under .github/workflow/ to understand the CICD plan.

Action time!

- Go to Databricks Repos and clone your repository.
- Checkout branch- `feature/<username>`.
- There are few code level changes required before one raises pull request:
  - Navigate to this file: `src/main/tests/bronze/test_load_data_into_bronze`
  - Set `expected_num_files = 2`
  - Check the Silver layer unit tests in the `src/main/tests/silver/` folder and add an assertion. Feel free to write any test case.

databricks

Action time!

- Review the integration_suite test which uses Files in Repos feature and fill in the missing elements:
  - `src/main/python/gold/gold_layer_etl.py`
  - `src/main/tests/integration_suite/test_integration_gold_layer_etl`
- Review the dbx deployment file
  - Open `deployment/deploy-job.yaml`
  - Update `notebook_path` variable to your Databricks Repos location.
- Review files under `.github/workflow/` to understand the CICD plan.

Action time!

- Create a pull request* and view the CICD unit testing job that spins up in github → actions.
- Once it succeeds, merge the pull request into develop branch* and view the CICD integration testing job that spins up
- Once integration tests are completed on develop branch, raise a PR from develop branch into main*. View the CICD job that spins up, runs unit & integration tests.
- Once it succeeds, merge the pull request into main* and view the CICD job that creates Databricks workflow jobs and launches them.

***Make sure that you are requesting to merge to the forked repository**

Thank you

Appendix

databricks

- https://en.wikipedia.org/wiki/Continuous_integration
- https://en.wikipedia.org/wiki/Continuous_delivery
- DevOps- <Definition>