Project Report

Title: End-to-End CI/CD Pipeline Implementation using Azure DevOps

Subtitle: A DevOps Project to Automate Code Integration and Deployment with Azure Repos, Pipelines, and Databricks

Submitted by: Varshitha Challa

Contact details: varshithachalla2554@gmail.com

Abstract

This project focuses on building an end-to-end **CI/CD pipeline using Azure DevOps and Azure Databricks**. The aim is to automate the process of writing code, testing it, and deploying it to production environments without manual effort. Automation ensures that code changes are delivered quickly, reliably, and securely, which is essential in modern software development.

The core of this project is the **integration between Azure DevOps and Databricks**. By setting up development and production Databricks workspaces, the project ensures that code is first tested in a safe development environment before being pushed to the live production environment. Using **Azure Repos** for version control allows developers to collaborate on code, track changes, and use feature branches to isolate updates.

A major part of the project involves configuring **CI pipelines using YAML files**. These pipelines automatically run tests when new code is committed, reducing the risk of bugs entering production. Once the CI process is successful, the CD pipeline automatically deploys the code to the production environment. This continuous workflow improves both the speed and quality of software releases.

The project also emphasizes **security and tracking**. Azure DevOps provides tools for setting up pull requests, requiring code reviews, and managing authentication between DevOps and Azure resources. Email notifications are integrated to keep teams informed of pipeline activities, such as successful builds or failed tests.

Overall, this project demonstrates how companies can use **Azure DevOps to streamline the software delivery lifecycle**, enhance collaboration among teams, and ensure that production environments always receive well-tested, reliable code. It serves as a practical guide for setting up real-world CI/CD pipelines using cloud-based tools.

Objective

The main goal of this project is to show how we can build and manage a **CI/CD pipeline using Azure DevOps**. This means setting up a process where code changes made by developers are automatically tested and deployed without needing manual work every time. This helps teams release updates quickly and safely.

We aim to connect **Azure DevOps with Databricks workspaces** for both development and production. By doing this, we can make sure that the changes made in the development environment can be smoothly moved to the production environment once they are approved and tested. This avoids any disruptions in the live system.

Another important goal is to **manage code using Azure Repos**. Developers create a feature branch for every new update. These branches are reviewed through pull requests, and only approved changes are merged. This method improves team collaboration, prevents mistakes, and keeps the main codebase clean and stable.

Finally, the objective is to **automate the entire development lifecycle**—from writing code, testing it, to deploying it. Using YAML pipelines in Azure DevOps, we can define steps for testing and deployment. This approach saves time, reduces errors, and helps maintain a consistent process across the team

Tools & Technologies Used

1. **Programming Language**
    1. **Scala:** Scala is a powerful, statically-typed programming language that combines object-oriented and functional programming. It is natively supported by Apache Spark, which makes it highly efficient for big data processing. In this project, Scala is used to develop notebooks within Azure Databricks for writing data transformation and processing logic. Its compatibility with the Spark engine makes it a preferred choice for high-performance data engineering workloads.

2. **Cloud Services:**

    1. **Azure DevOps:** Azure DevOps is a cloud-based service by Microsoft that provides a complete suite of DevOps tools. It allows teams to plan, develop, test, and deploy code collaboratively. It includes boards for agile planning, Azure Repos for version control, and Azure Pipelines for continuous integration and delivery (CI/CD). In this project, Azure DevOps is used to manage the entire pipeline from code commit to deployment**.**

2. **Azure Repos:** Azure Repos offers Git-based version control within Azure DevOps. It enables developers to manage code across branches, track changes, and collaborate via pull requests and code reviews. In this project, notebooks from the Databricks development environment are pushed to a feature branch in Azure Repos, ensuring version control and traceability.
3. **Azure Pipelines:** Azure Pipelines automate the process of building, testing, and deploying applications. It supports Continuous Integration (CI) and Continuous Deployment (CD). YAML files define pipeline steps, allowing versioning of pipeline configurations alongside code. In this project, the CI pipeline validates code changes and automatically triggers CD to move tested changes into the production Databricks environment.
4. **Azure Databricks:** Azure Databricks is a unified analytics platform built on Apache Spark. It provides notebook-based development for big data processing, data science, and machine learning. In this project, Databricks is used to write and manage Scala-based notebooks, which are then committed to Azure Repos and deployed to production via Azure Pipelines. It supports automated cluster management and is tightly integrated with Azure services.

Other Tools:

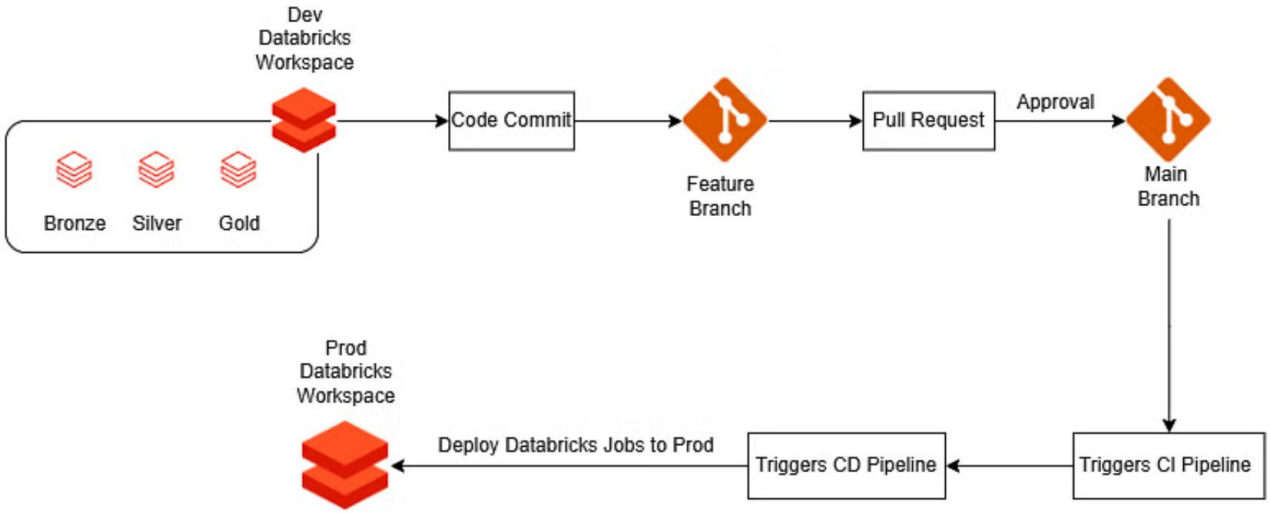
1. **YAML (Yet Another Markup Language):** YAML is used to define the configuration of Azure Pipelines. It provides a structured way to describe the steps and stages of your CI/CD process. In this project, YAML files define when the pipeline runs, which tests it executes, and how code is deployed to production environments, ensuring version-controlled automation.
2. **Git:** Git is a distributed version control system used to track changes in source code during software development. It allows multiple developers to work on the same project at the same time without overwriting each other's work. In this project, Git is the underlying technology used by Azure Repos to manage code versions

Business Overview

Continuous Integration and Continuous Deployment (CI/CD) form the backbone of modern software delivery. CI enables developers to integrate code into a shared repository frequently, with immediate automated testing to ensure stability. CD automates the deployment of validated code to production environments, eliminating manual intervention and minimizing errors. This project leverages these principles using Azure DevOps, a cloud-based suite from Microsoft that integrates planning, development, testing, and deployment into a seamless workflow.

Additionally, version control and pull request workflows enhance traceability, accountability, and collaboration, making it easier for teams to review, approve, and monitor changes before they go live.

Architecture



1. Dev Databricks Workspace (Bronze → Silver → Gold)

- The development workspace is organized into three layers:
  - **Bronze**: Raw data ingestion
  - **Silver**: Cleaned and filtered data
  - **Gold**: Final transformed data for reporting or analytics
- Developers create or modify notebooks here and perform unit tests.

2. Code Commit

- Once the development work is complete, code is committed from the Dev Databricks workspace to the linked Azure Git Repo.
- This commit is pushed to a feature branch, isolating new changes from the main production code.

3. Feature Branch → Pull Request → Main Branch

- A **pull request (PR)** is raised from the feature branch.
- The code is then reviewed and approved by team members or maintainers.
- Upon approval, the code is merged into the main branch, marking it ready for production deployment.

4. Triggers CI Pipeline

- Merging to the main branch automatically triggers the CI **(Continuous Integration)** pipeline.
- This pipeline:
  - Validates the code (via tests)
  - Builds any dependent components
  - Prepares the deployment artifacts (e.g., notebooks, scripts)

5. Triggers CD Pipeline → Prod Deployment

- Once CI is successful, it triggers the CD (Continuous Deployment) pipeline.
- The CD pipeline is responsible for:
    - Authenticating with the Prod Databricks workspace
    - Deploying the tested Databricks notebooks and jobs to production
    - Finalizing production readiness

6. Prod Databricks Workspace

- The production workspace now contains the newly deployed and tested code.
- It is isolated from development to ensure stability.
- The data and notebooks here are considered reliable and ready for end-users or downstream systems.

Implementation

1. **Workspace Setup:** - Created Azure Resource Group. - Provisioned two Azure Databricks workspaces: one for development and one for production.

2. **Version Control:** - Initialized Azure DevOps account and created Azure Repos. - Cloned Repos into the Dev Databricks workspace for direct commit tracking.

3. **Branching Strategy:** - Feature branches were created for every new commit. - Branch policies enforced to prevent direct commits to the main branch. - Pull request approvals were mandatory before merging.

4. **CI/CD Pipeline Setup:** - YAML templates were used to define CI workflows. - Each commit triggered unit and integration tests automatically. - Successful builds triggered CD workflows to deploy updates to the production workspace.

5. **Authentication & Connection:** - Service connections established between Azure DevOps and Azure Resource Groups. - Secure tokens and secrets managed in the Azure DevOps Library.

6. **Folder Organization:** - Dev workspace included folders like "CICD" for YAML files, tokens, and scripts.

7. **Notifications:** - Integrated email alerts for pull request events and build status.

Key Features

- **Faster Releases:** Automatic pipeline triggers reduce manual overhead.
- **Quality Assurance:** CI pipelines validate each code change with predefined tests.
- **Tracking:** Visibility into each deployment through logs and notifications.
- **Version Control:** Git-based workflows in Azure Repos ensure secure collaboration.

Challenges Faced

- Encountered hosted parallelism limitations in Azure DevOps; resolved via Microsoft form submission.
- Initial difficulties in setting up secure tokens for service connections.
- Ensuring accurate YAML configurations across environments.

Results & Observations

- Achieved seamless automated deployment from Dev to Prod environments.
- Improved team collaboration using version-controlled notebooks.
- Decreased deployment failures due to structured review and testing processes.

Future Enhancements

- Introduce rollback mechanisms for failed builds.
- Add deployment approvals and release gates for production pushes.
- Integrate real-time monitoring tools such as Azure Monitor.

Conclusion

This project successfully showcases how Azure DevOps services can be harnessed to build a scalable, secure, and automated CI/CD pipeline for Databricks. Through effective use of version control, branching strategies, and YAML-based configurations, the development cycle is optimized, promoting efficient and error-free code delivery.

References

- Microsoft Azure DevOps Documentation
- Azure Databricks Docs
- Git and YAML Reference Guides
- Project Cookbook (if applicable)