# Azure DevOps Documentation

## Introduction

This document outlines a structured branching strategy, including best practices for feature development, code reviews, and deploying updates. Following these guidelines will help teams work effectively, reduce errors, and deliver high-quality software.

## 1. Branching Strategy

In Azure DevOps, it's important to maintain a structured branching strategy to ensure code quality and facilitate collaboration. The primary branches you'll work with are:

* **Main (Production Branch)**: This branch contains the stable version of your code that is ready for production deployment.
* **Test (Test Branch):** This branch is used for integration testing and quality assurance before code is merged into the Main branch
* **Development (Development Branch)**: This is where the latest development happens. Features are integrated here before they are considered stable.
* **Feature Branches**: These branches are created from the Develop branch for specific features or bug fixes. They allow developers to work independently without affecting the main codebase.

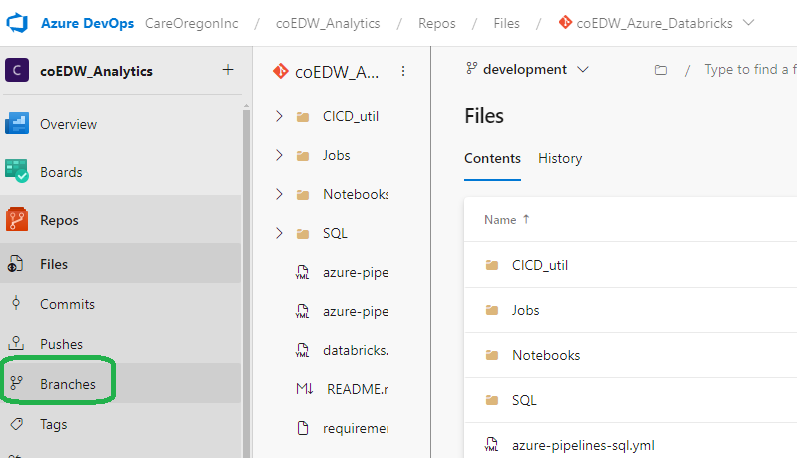
### Creating a Feature Branch

**Why Create a Feature Branch?**

* **Isolation**: Work on new features or bug fixes can happen independently without affecting the main codebase.
* **Collaboration**: Multiple developers can work on different features simultaneously.
* **Code Quality**: Allows for focused testing and reviews before merging changes.

**Steps to Create a Feature Branch**:

1. Open your Azure DevOps project.
2. Navigate to **Repos**.
3. Click on **Branches**.



1. Click on the **New branch** button. You will be navigated to Create Branch popup.

A screenshot of a computer

Description automatically generated

A screenshot of a login box

Description automatically generated

1. Name your branch using a descriptive name (e.g., STSK00001\_CICD).
2. Click **Create**.

A screenshot of a computer

Description automatically generated

### 1.2 Branch Policy

To maintain code quality while allowing efficient collaboration, the following branch policy is enforced:

* **Commit:** Changes cannot be directly committed to the Development, Test, or Main branches because these branches are linked to specific environments. Committing directly could disrupt these environments, leading to potential issues or instability.
* **Reviewer:** When a Pull Request (PR) is raised, at least one reviewer must approve the changes before they can be merged into the Development, Test, or Main branches. This ensures that the code is reviewed for quality, correctness, and potential issues before being added to these important branches.

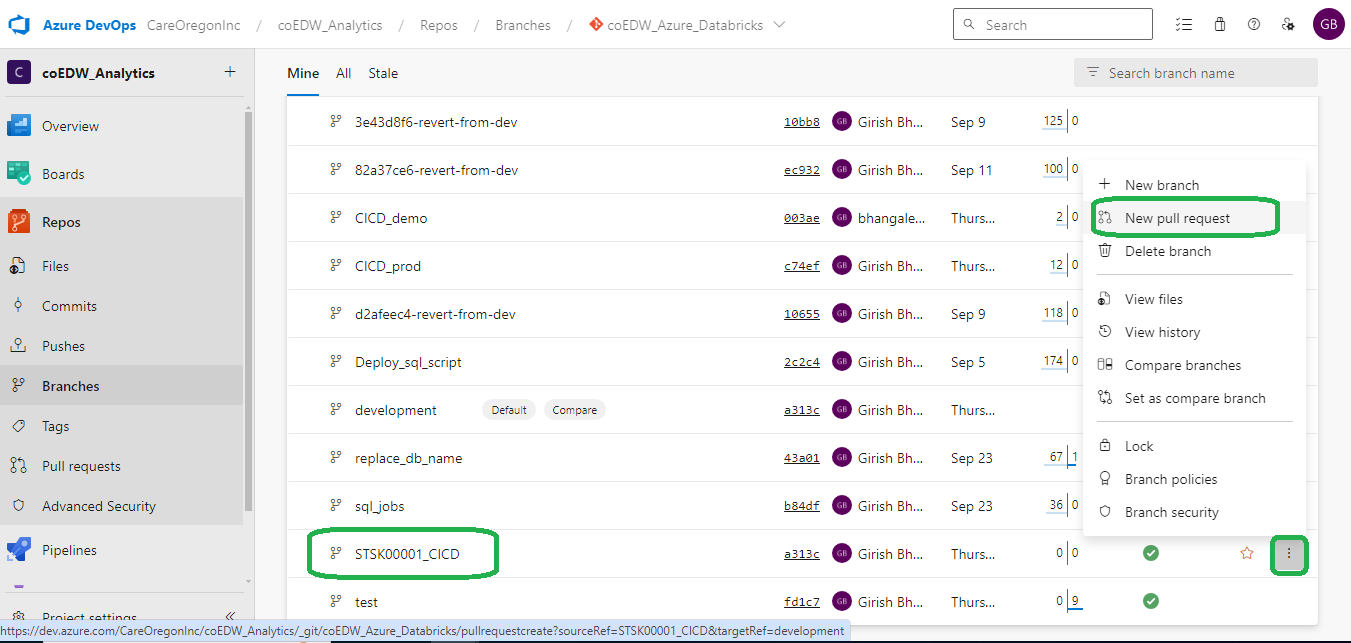
By adhering to these policies, we can ensure a stable and high-quality codebase while fostering effective collaboration among team members.

## 2. Creating a Pull Request (PR)

After completing your feature development, you need to merge your changes back into the Develop branch.

**Steps to Create a PR:**

1. Navigate to **Repos** > **Branches**.
2. Locate your feature branch.
3. Click on the ellipsis (...) next to your branch name.
4. Select **New Pull Request**.



1. Fill out the PR form:
   * **Title**: A concise title for the changes.
   * **Description**: Describe the changes made and any relevant context.
   * **Reviewers**: Add team members for code review.

A screenshot of a computer

Description automatically generated

1. Click **Create** to submit the PR.

## 3. Running CI/CD Pipelines

Continuous Integration and Continuous Deployment (CI/CD) pipelines automate the process of building, testing, and deploying your application.

**How to Run CI/CD Pipeline:**

1. Navigate to **Pipelines** in Azure DevOps.

A screenshot of a computer

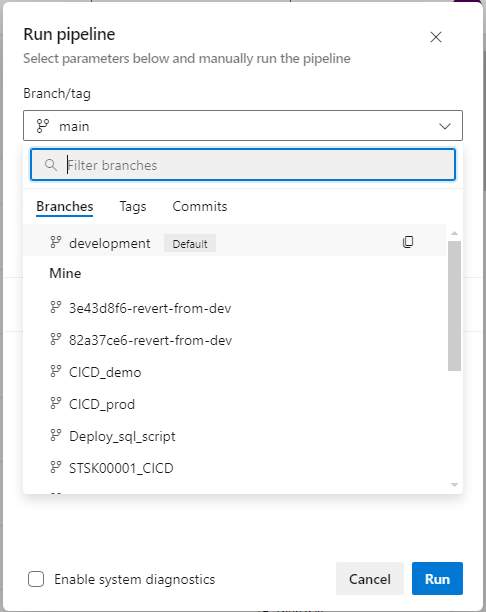
Description automatically generated

1. Click on your desired pipeline. (e.g., **coEDW\_Azure\_Databricks\_jobs**)
2. Click **Run Pipeline**.

A screenshot of a phone

Description automatically generated

1. Select the branch you want to build (e.g., development).



1. Click **Run** to start the pipeline.

A screenshot of a computer

Description automatically generated

## 4. Deployment Strategies

### 4.1 Dev Workspace Deployment

To deploy the code to dev databricks workspace development branch should be used.

* **Environment**: dev
* **Branch**: Development

**Steps:**

1. Create a PR from feature branch to development.
2. Upon approval, merge changes into the development branch.
3. Trigger the CI/CD pipeline for the dev environment.
4. Ensure the pipeline targeting the Development branch.

### 4.2 Test Deployment (Test)

To deploy the code to test databricks workspace Test branch should be used.

* **Environment**: test
* **Branch**: test

**Steps:**

1. Create a PR from development to test.
2. Upon approval, merge changes into the test branch.
3. Trigger the CI/CD pipeline for the Test environment.
4. Ensure the pipeline targeting the Development branch.

### 4.3 Production Deployment (Main)

To deploy the code to prod databricks workspace main branch should be used.

* **Environment**: Production
* **Branch**: Main

**Steps:**

1. Create a PR from test to Main.
2. Upon approval, merge changes into the Main branch.
3. Trigger the CI/CD pipeline for the Production environment.
4. Ensure the pipeline targeting the main branch.

**Conclusion**

By following this structured approach to branching, PR creation, and deployment, you can maintain a high level of code quality and ensure smooth collaboration within your team. Implementing CI/CD pipelines further automates and enhances your development workflow, making deployments more efficient and reliable.