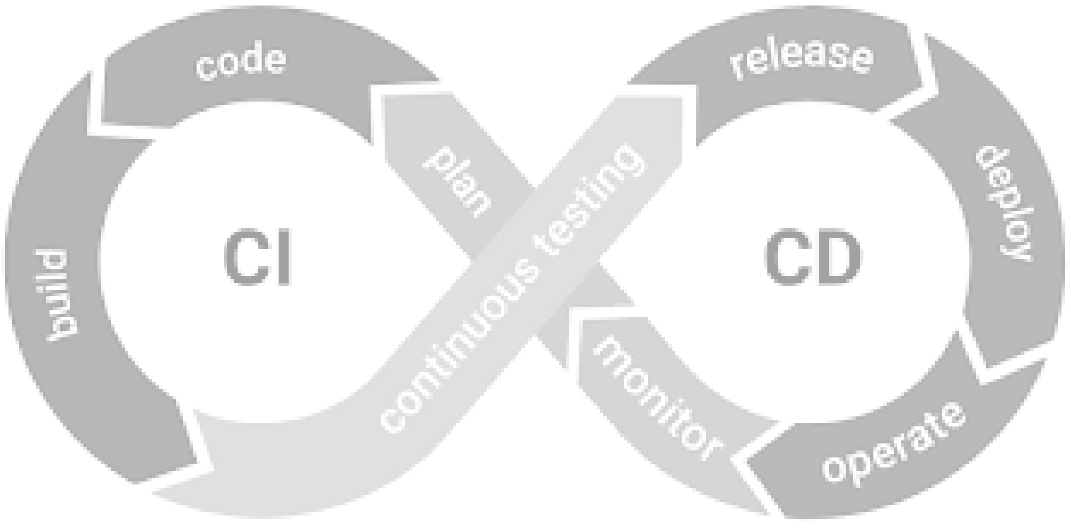
DevOps End-End Process Management for Release Automation

int.prc.001.03 | 03.04.24

UPM 24.x



Contents

Introduction and Goal

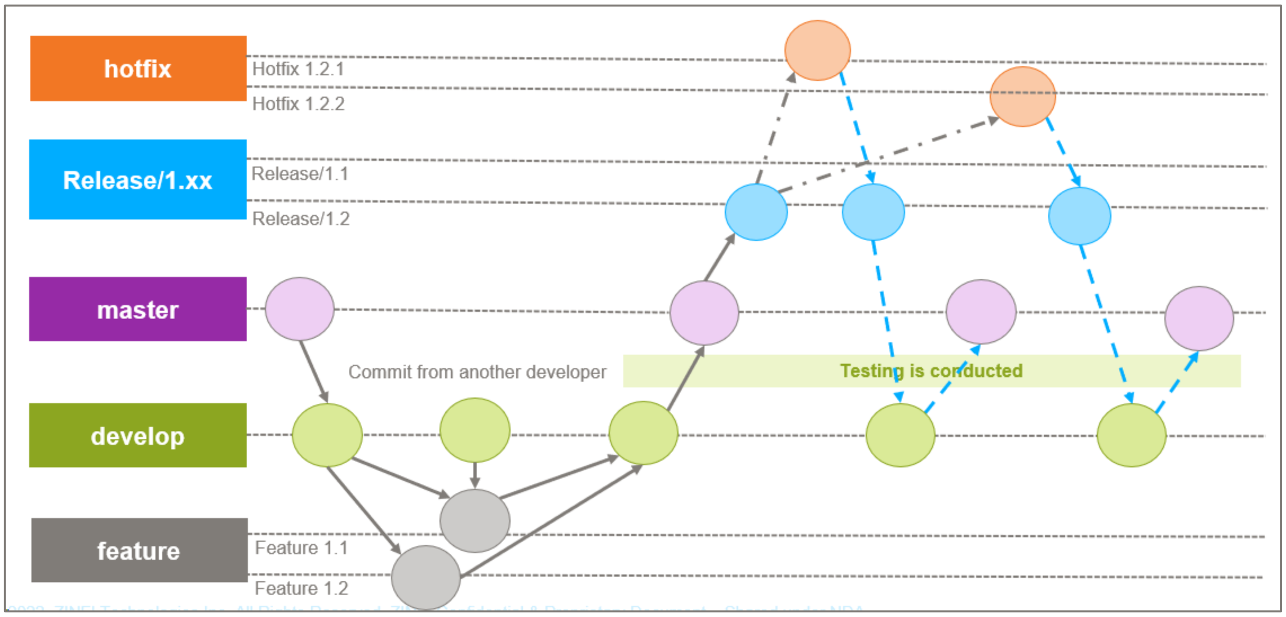
For a successful Automated Release Process to Accelerate application development and development lifecycles, building quality and consistency into an automated build and release process, and increasing application stability and uptime – the following are conducted:

- Azure DevOps - Repos are implemented to Manage UPM Platform Codebases (UI, API, and DB Layer).
- GIT Codebase Branches for Azure Repos are managed.
- Azure DevOps - Pipelines are implemented for automated testing, acceptance, and deployment of portal changes to staging and production environments.
- Pipelines are managed across all Azure Repos for the stable deployment of upgrades/fixes.

Note:

- **Going forward, the following process will be utilized for Patch release and New Version Release.**
- **For Fresh Setup, a New Codebase is to be utilized.**

Code Repo Branching Strategy

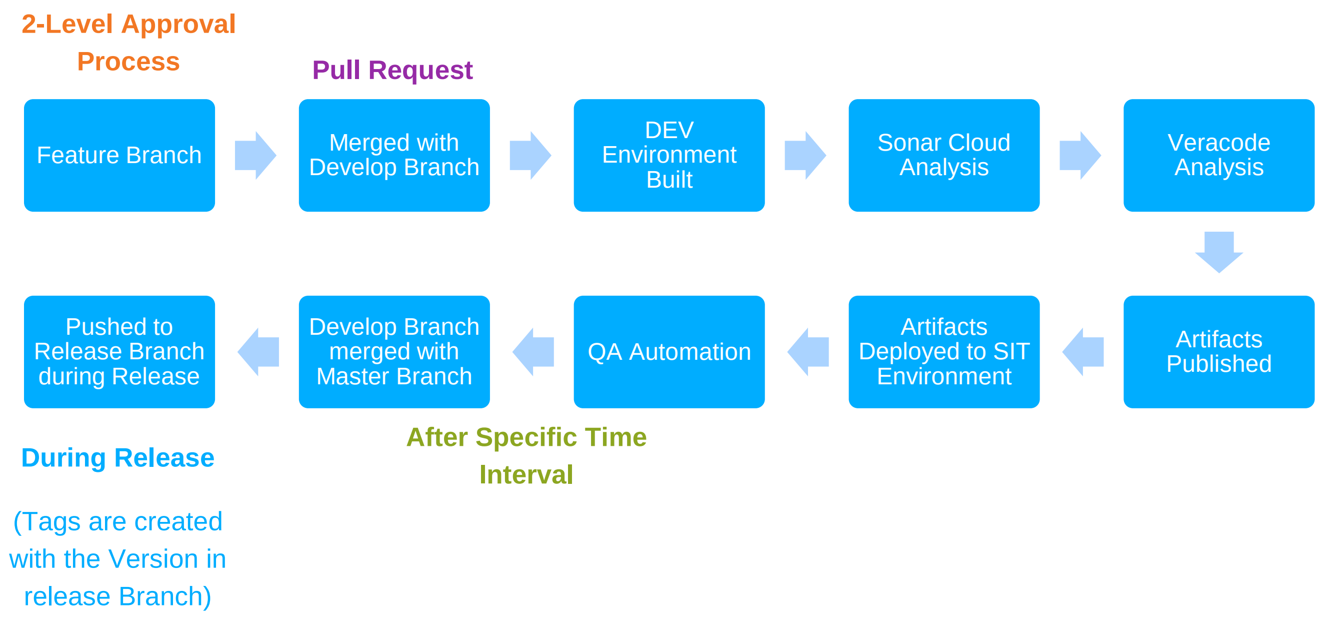


Tools Utilized

- **Azure Repos -** An Azure Repos Git repository serves as a code repository that provides version control and a platform for collaborative projects.
- **Azure Pipelines -** provides a way to build, test, package, and release applications and infrastructure code.

Pipeline Readiness and Branch Policy Approvals

Process Definition


2-Level Approval Process
Pull Request
Feature Branch
Merged with Develop Branch
DEV Environment Built
Sonar Cloud Analysis
Veracode Analysis
Artifacts Published
Artifacts Deployed to SIT Environment
QA Automation
Develop Branch merged with Master Branch
Pushed to Release Branch during Release
During Release
(Tags are created with the Version in release Branch)
After Specific Time Interval

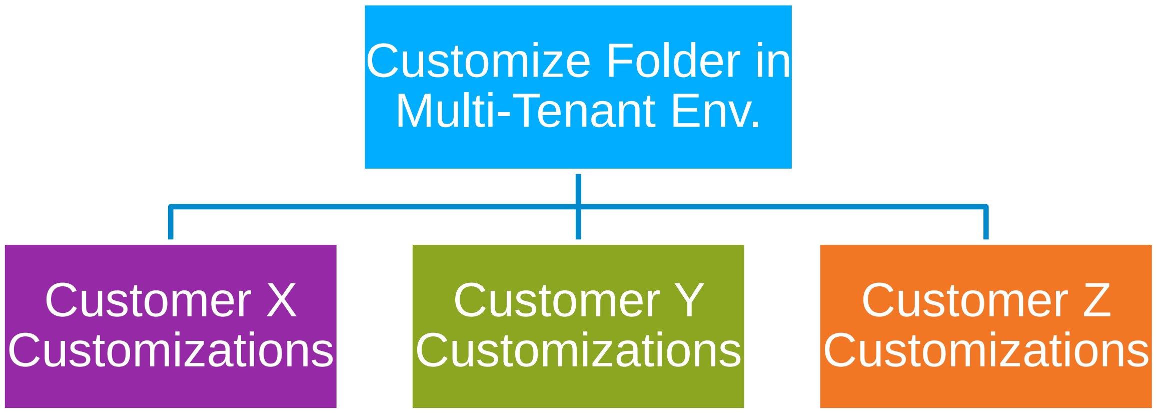
Note: Development is conducted in the Feature Branch. Development undergoes a 2-level Approval Process (Level 1 – Chapters/leaders, Level 2 – Amit/Gouranga). Once approved, the feature is merged with the develop branch via a pull request.

Pipeline Readiness Summary

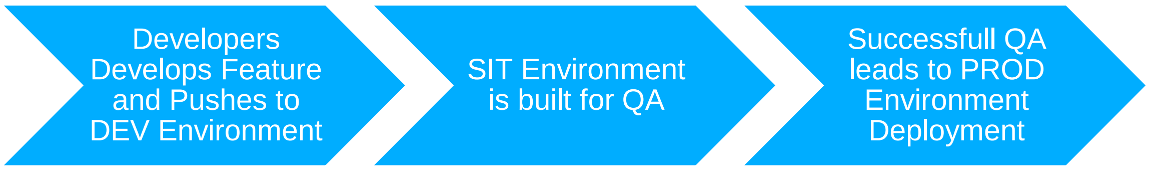
- DEV and SIT Environments are to be created for each customer deployment.
- DEV and SIT Environments are only considered for Feature/Patch Development and Release.
- Develop Branches are utilized to develop the feature/patch.
- Once developed, the feature/patch will undergo an Approval process.
- Once approved, the Pull Request from the Develop Branch to the Master is conducted manually.
- Once the pull is approved in the Master Branch, the code is Merged and Tagged to the Release Branch.
- Hotfixes are processed through Hotfix Branches, pushed to the Release Branch(maybe a new tag is required), pushed to the Develop Branch, and finally pushed to the Master Branch.

Customization Flow

- In the Multi-tenant Environment, a 'Customize' Folder is used to contain all Customer-specific Folders with Customer Names to store customization scripts.


Customize Folder in Multi-Tenant Env.
Customer X Customizations
Customer Y Customizations
Customer Z Customizations

Process Summary


Developers Develops Feature and Pushes to DEV Environment
SIT Environment is built for QA
Successfull QA leads to PROD Environment Deployment

Phase wise Implementation of CI/CD Infrastructure

Code Management and Branching via Azure Repos

Build and Deploy via Azure Pipelines

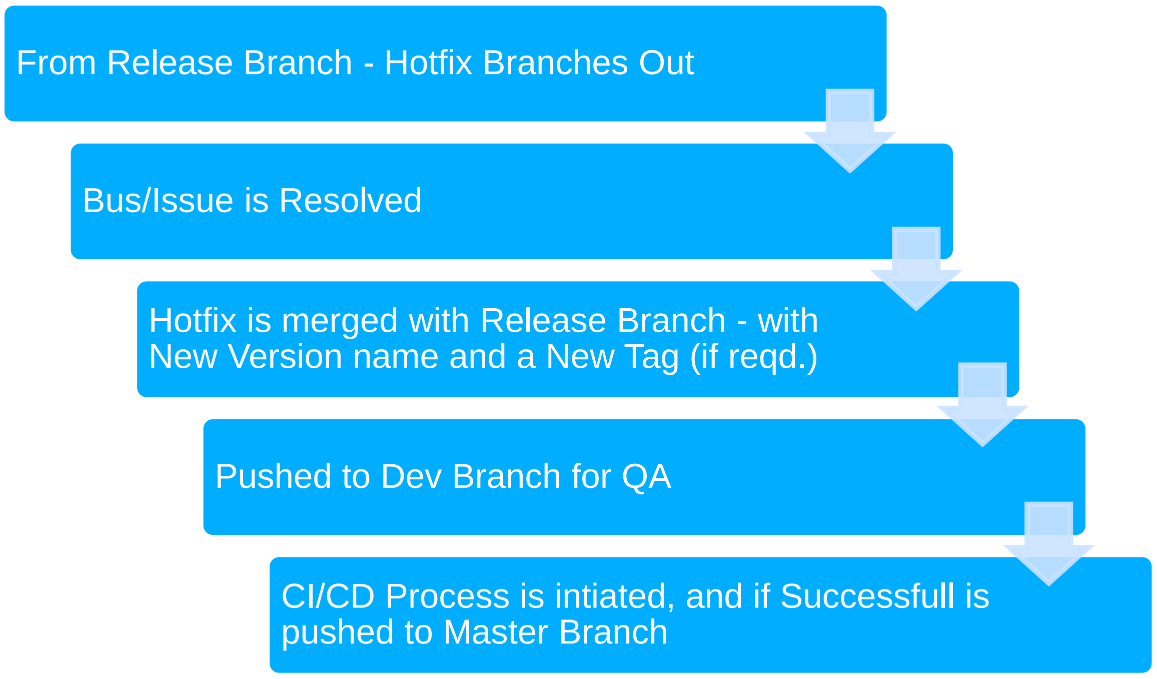
- **CI Pipeline -** A merge to Azure Repos Git triggers a CI pipeline. This pipeline runs the same checks as the PR pipeline with some important additions. The CI pipeline runs integration tests. These integration tests shouldn't require the deployment of the solution, as the build artifacts haven't been created yet. If any of the checks fail, the pipeline ends and the developer will have to make the required changes. A successful run of this pipeline results in creating and publishing-built artifacts.
- **CD pipeline trigger -** The publishing of artifacts triggers the CD pipeline.
- **CD release to SIT -** The CD pipeline downloads the build artifacts that are created in the CI pipeline and deploys the solution to a staging environment. The pipeline then runs acceptance tests against the staging environment to validate the deployment. If any acceptance test fails, the pipeline ends and the developer will have to make the required changes. If the tests succeed, a manual validation task can be implemented to require a person or group to validate the deployment and resume the pipeline.
- **CD release to PROD -** If the manual intervention is resumed, or there's no manual intervention implemented, the pipeline releases the solution to production.

Using proven CI and CD practices to deploy application or infrastructure changes provides various benefits, including:

- **Shorter release cycles -** Automated CI/CD processes allow you to deploy faster than manual practices. Many organizations deploy multiple times per day.
- **Better code quality -** Quality gates in CI pipelines, such as linting and unit testing, result in higher-quality code.
- **Decreased risk of releasing -** Proper CI/CD practices dramatically decrease the risk of releasing new features. The deployment can be tested before release.
- **Increased productivity -** Automated CI/CD frees developers from working on manual integrations and deployments so they can focus on new features.

- **Enable rollbacks -** While proper CI/CD practices lower the number of bugs or regressions that are released, they still occur. CI/CD can enable automated rollbacks to earlier releases.

Hotfixes Management – Branching Strategy


From Release Branch - Hotfix Branches Out
Bus/Issue is Resolved
Hotfix is merged with Release Branch - with New Version name and a New Tag (if reqd.)
Pushed to Dev Branch for QA
CI/CD Process is intiated, and if Successfull is pushed to Master Branch