# **Gen AI GIT Branching Strategy Document**

# Purpose of the Document

This document enlists recommended Azure DevOps branching cycle that will be employed as part of GenAI DevOps strategy.

# Document Details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Modify Date | Changes | Created By | Reviewed By |
| V1.0 | 11-03-2024 | Created | Nagarjuna Bolla  Sourabh Rai | Rajesh P  Raviraj Naik |

# **Git Branching Strategy- Generative AI**

**Primary Branches:**

**Development Branch:** This branch caters to the ongoing development work. It serves as the integration hub for features and bug fixes before they are merged into the master branch. All feature branches are created from and merged back into this branch.

**Feature Branches:**

These branches are created by developers off the development branch for the purpose of implementing new features or enhancements. These branches should bear clear and descriptive names, for instance, feature/login-page or feature/payment-gateway. Upon completion of the feature, it should be merged back into the development branch based on approval. **(Approvers name TBD)**

**Release Branches:**

When the development branch achieves a stable state and is ready for release, a release branch is created from the development branch. This branch facilitates final testing, bug fixes, code reviews and last-minute changes before being deployed to production based on approval. **(Approvers name TBD)**

**Production (Prod) Branch:**

Once the build and deployment are successfully completed from the release branch, and the application is functioning as expected, the changes are pushed to the Prod Branch based on approvals **(Approvers name TBD)**. This branch reflects the production-ready state where only stable and approved changes are merged**.**

**Environment Branches:**

For each operating environment (e.g., development, QA, production), dedicated branches should be created to track the state of the code deployed to each respective environment. These branches provide a method to hotfix or rollback changes specific to a particular environment.

**Additional Information:**

**Pull Requests:** Pull requests should be used to merge feature branches into the development branch. This process allows for code review before code is merged.

**Hotfix Branches:** In case of critical issues in production, hotfix branches can be created directly from the master branch. After the fix, the hotfix branch should be merged back into both the master and development branches **based on approval (Approvers name TBD)**

**Tagging:** Tagging should be used in the prod branch to mark release points. Tags are references to specific points in Git history, typically used to capture a point in history that is used for a marked version release**.**

**Workflow Overview:**

**A diagram of a software company

Description automatically generated**

Production (Prod)  
(Go-Live)

Release (QA)  
(Code Review)

Develop (Dev)  
(MR/PR)

**Branching Workflow -**

1. Developers commence work on individual feature branches, generating pull requests (PRs) to integrate their changes into the main development branch. This ensures that all new features and updates are reviewed and tested before being merged with the production codebase.
2. Post completion and testing of features on the development branch, a release branch is formed for final review, testing, and preparation for deployment. This branch acts as a buffer zone, allowing for any necessary adjustments before the code is promoted to the production branch.
3. **After comprehensive testing on the release branch, modifications are merged into the production branch and tagged with a version number. This version number signifies a production-ready state for the codebase, facilitating production deployment.**
4. In the event of unexpected issues arising in production that demand immediate resolution, a hotfix branch is initiated from the production branch. The issue is fixed in this branch and then migrated to the production branch with an updated hotfix version tag.
5. It is crucial to ensure that the changes implemented in the hotfix branch are also merged (back merging) into the release and/or development branches. This step is necessary to maintain code consistency across all branches.
6. If the hotfix modifications are not merged into the release/development branches before they undergo regression testing or code review and are subsequently deployed to production, these hotfix changes risk being lost.

**To set up Azure pipelines for different environments (Dev, QA, Prod) -**

1. Determine the mapping of Git branches to each environment.

* Development: ‘**genywebapi-dev’** branch
* QA: ‘**genywebapi-qa**’ branch
* Production: ‘**genywebapi-prod/master**’ branch

2. Create separate pipelines YAML files for each environment.

* **azure-pipelines-dev.yml**
* **azure-pipeline-qa.yml**
* **azure-pipelines-prod.yml**

1. Within each pipeline YAML file, it is necessary to define the branch triggers. These triggers specify which branches will initiate the pipeline, thereby controlling the flow of changes through the pipeline and ensuring that each change is properly built, tested, and deployed.
2. Every YAML file should be customized to meet the specific needs of the project. This includes setting branch triggers, defining build steps, outlining deployment tasks, and more. Carefully configuring these elements will help ensure that your pipeline accurately reflects your development, testing, and deployment processes.
3. For each pipeline YAML file, configure environment-specific variables. These may include connection strings, API keys, or environment configurations. To securely manage sensitive information such as passwords or secret keys, utilize services like Azure Key Vault or pipeline variables. This not only enhances the security of the project but also allows to manage and update sensitive information without modifying the pipeline YAML files.

**Deployment Strategy:**

**Azure DevOps utilizes two primary types of pipelines to facilitate a robust Continuous Integration and Continuous Deployment (CI/CD) process.**

1. **Build Pipeline (Continuous Integration – CI):** The objective of the build pipeline is to compile the project or build the source code, subsequently generating artifacts. These artifacts are essentially the output of build process that needs to be deployed.
2. **Release Pipeline (Continuous Deployment – CD) :** This pipeline is responsible for deploying the application to the Azure App Service. It consumes the artifacts produced by the CI pipeline, deploying them to the appropriate environment.

**Deployment Triggers:**

1. **Development**: Continuous deployment is initiated by code changes to the develop branch. The Azure DevOps pipeline is automatically triggered for the development environment upon each change, ensuring the latest changes are always deployed.
2. **QA**: Automated deployment is activated by merges to the QA branch, ensuring any changes approved for testing are immediately available in the QA environment.
3. **Production**: Continuous deployment is triggered by merges to the production branch, ensuring that approved changes are promptly and reliably deployed to the production environment.

**Note: For achieving continuous deployment, the creation of a release pipeline is essential. This is because, in the event of encountering any issues with the current version in the release pipeline, we can efficiently rollback to the previous version, when necessary, thereby maintaining the stability of the production environment**.

**Appendix –**

**Branch naming conventions for each environment:**  
  
genywebapi-dev

genywebapi-qa

genywebapi-prod