Bitbucket - Code Deployment

Table of Contents

[1. Objective 3](#_Toc99039474)

[2. Intended Audience 3](#_Toc99039475)

[3. Prerequisites 3](#_Toc99039476)

[4. Code promotion Workflow 3](#_Toc99039477)

[5. Code Implementation in Development Environment 3](#_Toc99039478)

[6. Code deployment in Quality Assurance (QA) Environment 5](#_Toc99039479)

[7. Code deployment in UAT Environment 5](#_Toc99039480)

[8. Code deployment in Production Environment 5](#_Toc99039481)

[9. Roll Back Plan 6](#_Toc99039482)

[10. Deploying Urgent Fixes 6](#_Toc99039483)

[11. Deployment Permissions 6](#_Toc99039484)

# Objective

This document is aimed to help to deal with deployments in a development workflow and provide some of the best practices for deployments process. This document provides detail steps of Code promotion guidelines through the various environments.

# Intended Audience

This document is intended for Development & QA team (as applicable) to understand Code promotion standard process using Bitbucket.

# Prerequisites

The Developers must be familiar with the basic navigation of Bitbucket

# Code promotion Workflow

Deployments should be treated as part of a development workflow. Deployment workflow will include four environments: Development, Quality Analysis (QA), UAT and Production. In that case the workflow might look like this:

* Developers work on bugs and new features in separate Bit Bucket branches. Really minor updates can be committed directly to the stable development branch after successful Unit testing completion.
* Once features are implemented, they are required merged into the Quality Analysis (QA) branch and deployed to the QA environment for testing.
* Once QA testing completed, Code-base merged into UAT branch and deployed to UAT environment for system integration testing.
* After testing is complete, UAT branches are merged into the release branch.
* On PROD release date, the release branch is merged into production and then deployed to the Production environment.

# Code Implementation in Development Environment

* All developer should PULL the latest version of code from BitBuket Master to local GIT repository.
* For Each Sprint, there should be a Master Branch.
* There should be different branch for every story point (feature development) and bug fixing (general maintenance) of older releases.
* Every story point should be verified locally first. Testing on a local computer removes the need to commit, push and deploy completely.
* Once it’s stable, code should be pushed to a Staging environment for unit testing.
* For each and every changes developer should follow the below workflow.

|  |  |
| --- | --- |
| **Development Workflow** | |
| https://confluence.atlassian.com/bitbucket/files/814201830/824476745/2/1461190963588/branch-diagram.png | Start with the **master** branch This workflow helps to collaborate on developer code with at least one other person. As long as Bit bucket and local repos are up-to-date, developer is ready to get started. |
| Create a **new branch** Use a separate branch for each feature or issue developer works on. After creating a branch, check it out locally so that any changes made will be on that branch. |
| Update, add, commit, and push changes Work on the feature and make commits any time using Git locally. When ready, push commits, updating the feature or maintenance branch on Bitbucket. |
| Get code reviewed to get feedback on code, create a pull request in Bitbucket. From there, developer can add reviewers and make sure everything is good to go before merging. |
| Resolve feedback Now reviewer comment and approve. Resolve their comments locally, commit, and push changes to Bitbucket. All updates will appear in the pull request |
| Merge pull request Before merge; developer may have to resolve merge conflicts if others have made changes to the repo. When pull request is approved and conflict-free, developer can add code to the master |

# Code deployment in Quality Assurance (QA) Environment

* Create a separate branch for QA
* Once features are implemented and considered stable, they get merged into the QA branch and then deployed to the QA environment.
* Testing teams go to QA servers and verify that the code works as intended.
* Testing team will log defects and assign it back to development team

# Code deployment in UAT Environment

* It is very handy to have a separate branch for each sprint called UAT to represent System Integration Assurance environment.
* It will allow different development team working on different story points to deploy multiple branches to the same server simultaneously, simply by merging everything that needs to be deployed to the UAT branch.
* Code from UAT branch should be deployed to the QA environment.
* Testers go to QA servers and verify that the code works as intended.
* Testers will log defects and assign it back to development.

# Code deployment in Production Environment

* Once new functionality/ feature implemented and tested, it can be deployed to production branch.
* Next step is to make a diff between the production and development branches to take a quick review at Codebase that will be deployed to production.
* Final validation of something that’s not ready or not intended for production.
* Once diff review completed, developer can merge the Integration branch into release branch.
* Initialize a deployment of the production branch to Production environment.
* Specify comments for deployment so that team members know exactly what was deployed with version control.
* It’s recommended that deploy major releases to production at a scheduled time, of which the whole team is aware of.
* Deployment needs to be planned during time when the application is least active
* Plan for monitoring application before/after deployment process to ensure that everything deploy as planned.
* Better to perform smoke testing to verify and validate all specified features or fixes that have been deployed to make sure they work properly in production.
* Deployment/Release team should send an email communication to all team members with a summary of changes after every deployment as release notes.

# Roll Back Plan

At sometimes, deployments would be failed. In that case there should have the possibility to rollback.

* After rollback is done, make sure to fix the bug that was discovered.
* Commit it to either the development branch (if it was minor) or a separate bug-fix branch.
* Then proceed with the regular bug-fix branch.
* Bug-fix → development → QA → UAT → production integration workflow

# Deploying Urgent Fixes

At some instance, developers need to deploy a bug-fix to PROD immediately, when a new sprint branch is not yet ready for release process. In such cases, development/release team need to follow the below steps,

* Independent bug fix branch needs to be created.
* All bugs fixing activity should follow the usual development and QA workflow.
* Bug fixing branch should be merge into development sprint branch first.
* Do not merge the full sprint branch into production branch.
* Only merge the bug fixing branch in production.
* Then deploy the production branch as usual. This will ensure that only bug-fix will be deployed to the Production environment without all the other stuff from the development branch that’s not ready yet.

Please be aware that it’s important to merge the bug-fix branch to both the development and production branches. Because production branch never includes anything that doesn’t exist in your stable development branch. The development branch is where developers work all-day, so if fix is only in the production branch, they will never see it and it can cause confusion.

# Deployment Permissions

Every developer should be able to deploy to the QA environment. They just need to make sure they don’t overwrite each other’s changes when they do. All changes from all developers are getting merged into it so it contains all of them.

Ideally, In Production environment should be accessible to a limited number of experienced/Subject Matter expertise team members. These members should always be prepared to fix any server related issue during deployment process.

A picture containing diagram

Description automatically generated

A picture containing diagram

Description automatically generated