Continuous Delivery Approach

Continuous Delivery Approach

As part of this Continuous Delivery initiative, the goal is to enable the organization to reliably deliver quality software at a faster pace. In order to achieve this, we will need to standardize on our development and release processes across all applications and implement a pipeline with a Shift-Left approach. The guidelines in this document will help to achieve this goal.

Standardize Artifact Versioning

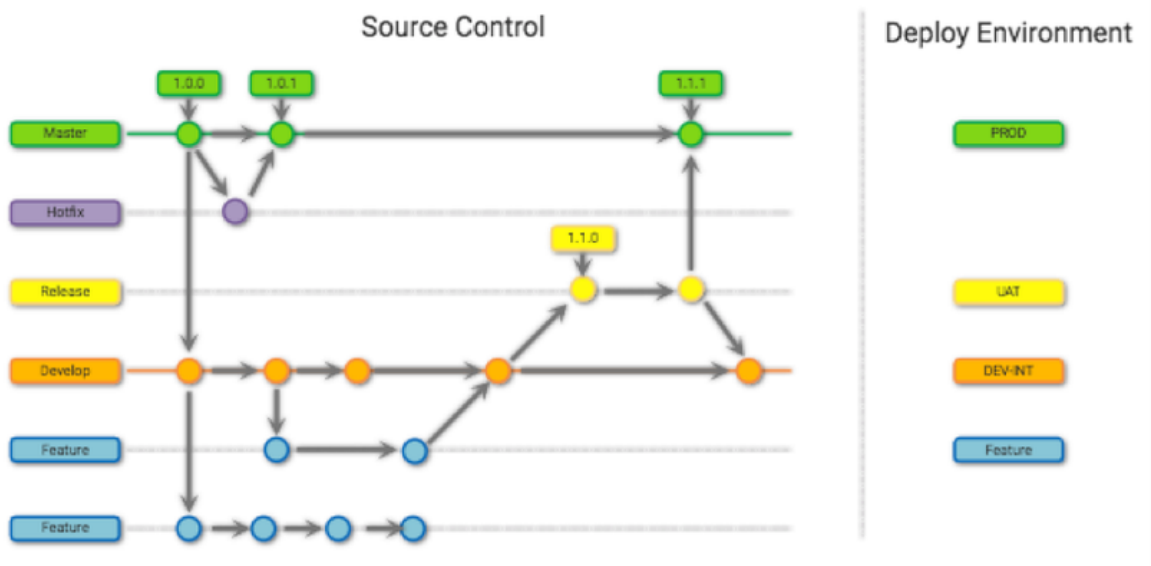
Using a standardized artifact versioning scheme enables a build once and reuse approach. Standardize on the following semantic versioning scheme.

[MAJOR][MINOR][PATCH]

Digit	When to Increment	Example
MAJOR	Introducing changes that are not backwards compatible	1.0.0 [Symbol] 2.0.0
MINOR	Scheduled feature releases	1.0.0 [Symbol] 1.1.0
PATCH	New Build and Patches	1.0.0 [Symbol] 1.0.1

Standardize Branching Strategy

Utilize GitFlow branching strategy.



BRANCH	PURPOSE	DETAILS

FEATURE	Unit of change for a Story scheduled for an upcoming release	• Developer uses JIRA to create branch from the develop br
		• Developer pushes often to feature branch
		• CI builds triggered on push to branch
		• Deployed to DEV environment
		  • VM based apps: One-Click deployment manually trigg
		  • Containerized apps: Automatically deployed using rolli

DEVELOP	Integration Branch: Reflects code for next scheduled release	• Long living branch.
		• Dev Leads merge to the protected branch.
		• Builds triggered on merge.
		• Deployed to protected DEV-INT environment.
		  • VM based apps: Automatically deployed nightly.
		  • Rolling update deployments for containerized applicati

RELEASE	Release Candidate: Used for preparing and vetting a release before production	• Short living branch that is merged to master and deleted w production. • Cut from develop branch when Dev Lead determines there the next release. • Automatically increment the minor version number on the d creating the release branch. • Release Git tag created for every build. • All -SNAPSHOT references in pom.xml is stripped as part (are immutable release versions) • One-Click deployment triggered manually to the protected
MASTER	Reflects code currently in production	• Used for record keeping
HOTFIX	Production fixes	• Branches from a release GIT tag. • Similar to a release branch: Fix is vetted on this branch pri production.

Standardize SCM

If there is a mix of SCM systems (SVN, ClearCase, BitBucket). In order to have a common set of workflows, we need to standardize on using BitBucket as our source control system.

Issue Tracking

All code changes must be associated with a JIRA ticket. Jira workflow will be standardized across all applications.

Independent Application Repositories

Each application that produces a jar/war should be in its own repository. This will allow independent branching and versioning. The application will also have its own dedicated JIRA project.

Standardize Packaging

All deployable applications should produce RPMs or a Docker image as part of the build process.

Repeatable Deployments

All deployments should be full deployments (no patch deployments). Deployments should be fully automated and exercised as part of the non-prod and prod deployments. Application deployments should have the capability to deploy the new binaries to servers ahead of time with the existing version of the application running. Activation of the new application version can occur as an independent step (example: stop instance, change symlink, and start instance).

Enforced Quality Gates

BRANCH	GATE
ALL BRANCHES	• Enforce that builds fail on unit test failure • Automated Smoke Testing triggered after deployment

DEVELOP, RELEASE, HOTFIX	Enforce the following Merge Gates • Must have successful build on source branch • Must have at least one code review Approval • Must pass Sonar Quality Gates Enforce the following as part of build • Fail build if Checkmarx security scan does not pass
	CheckMarx scans on every build. Fail build if medium or high finding exists or Low finding exist for longer than 60 days.
RELEASE, HOT FIX	Integrate the following tests as part of the pipeline • Regression Tests • Performance Tests
LOCAL BRANCH	Developer should configure SonarLint in IDE to continuously monitor for Code Quality Issues

Tool Integration

TOOL	DETAIL
IDE	Standardize on Intellij/Eclipse. Configure the following plugins: • Sonar Lint • Team City • Bitbucket Linky
TEAMCITY	Should have the following configured: • Multi-Branch configured for dynamic branch job creation/deletion • Docker based ephemeral build agents • Commit Status Publisher to notify BitBucket of build status • CheckMarx

BITBUCKET	hould have the following configured:
	• SonarQube configured
	• JIRA Application Link
	• TeamCity Trigger Plugin
	• GitFlow branching strategy
	• Pull Request Merge Checks
	  • Sonar Quality Gates
	  • At least one code review Approval
	  • Successful Build

SONAR	Leverage the commercial Enterprise edition to take advantage of the Branching and Dashboard reporting capabilities (6.x community edition version removes a good amount of the dashboard capabilities that were available in 5.x). Should have the following configured: • Create a Citi quality profile based off of the base Sonar Way quality profile. • Each application should have quality gates defined and will be enforced during BitBucket
OPENSHIFT	Integrate OpenShift for containerized deployments. Feature level builds should automatically deploy to the DEV cluster and will require enough resources to handle the projected large number of feature builds.