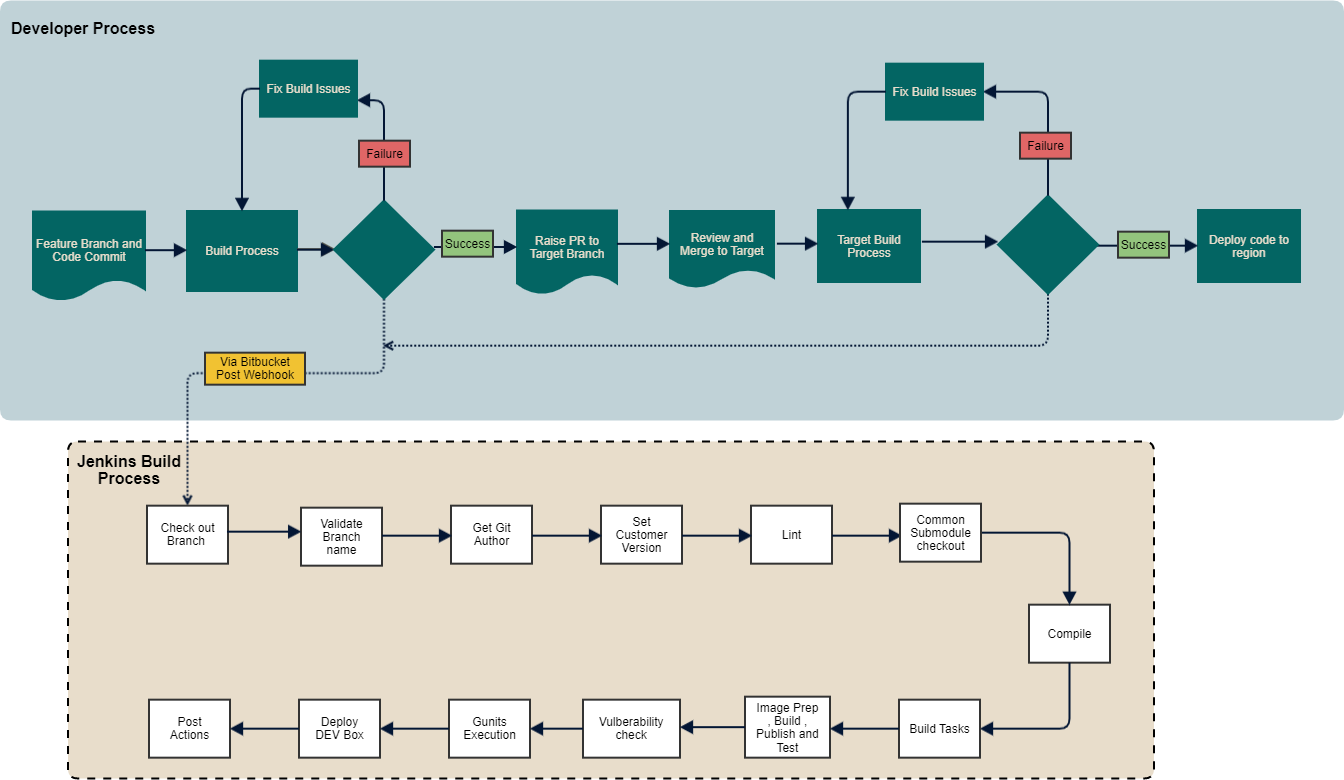
Build Pipeline /Branching Strategy

# Build Pipeline :



## Development Lifecycle :

Developer will create feature branch from Target branch(master or release branch) following branch naming convention set out.

A developer creates a branch in format <branch\_type>/<initials>/<jira-id>\_<freeform description> where

<branch\_type> is "feature" or "bugfix"

<jira-id> is 3-6 alphabetic characters followed by a hyphen, then numbers e.g. ABC-123", JKOL-23542 etc.

Note that "/" is required before and after initials and "\_" is required after the Jira-Id and that some freeform text must also follow the "\_"

Pipeline events trigger notifications to MS Teams channels to communicate that a build has started, succeeded or failed.

Every push of feature branch to Bitbucket server will trigger a branch pipeline build via Bitbucket Post Webhook

Branch pipeline build will run all stages, deploy to *dev-branch* environment and run smoke test (will abort if any quality gates fail) When a feature is complete a Pull Request (PR) to *master* is created for review

To allow merging to *master* a PR must have: minimum successful branch builds minimum number of approvals

no incomplete PR tasks

Merge of PR on Bitbucket server will trigger a *master* pipeline build via Bitbucket Post Webhook

Master pipeline build will run all stages, deploy to first *dev* environment and run smoke test (will abort if any quality gates fail)

## Jenkins Process :

All events in the process have a QG (**Quality gate).** If a check fails it will stop the pipeline

Check out Branch (**QG**)- GIT clone is performed - jenkins pulls the code from Bitbucket

Check Branch name (**QG**)- A check is performed to ensure the branch name matches our branching policy The Push process from any Git Tool prohibits this , this will be a lateral process in addition

Lint (**QG**) - Static code Analysis. Checking good practices and prepare a well formed docker file Common checkout , Compile (**QG**) - Check out Common code linked and Compile the code Build Tasks (**QG**) - Build the war file , Build the data dictionary(Build the Data and type list map)

Build, Preparing and Publishing Image, Vulnerability checks (**QG**) - Build the image, take the OS image Add Java, Tomcat, Application layer (which comes from the previous stage in the pipeline) Container image test using the google container image testing framework

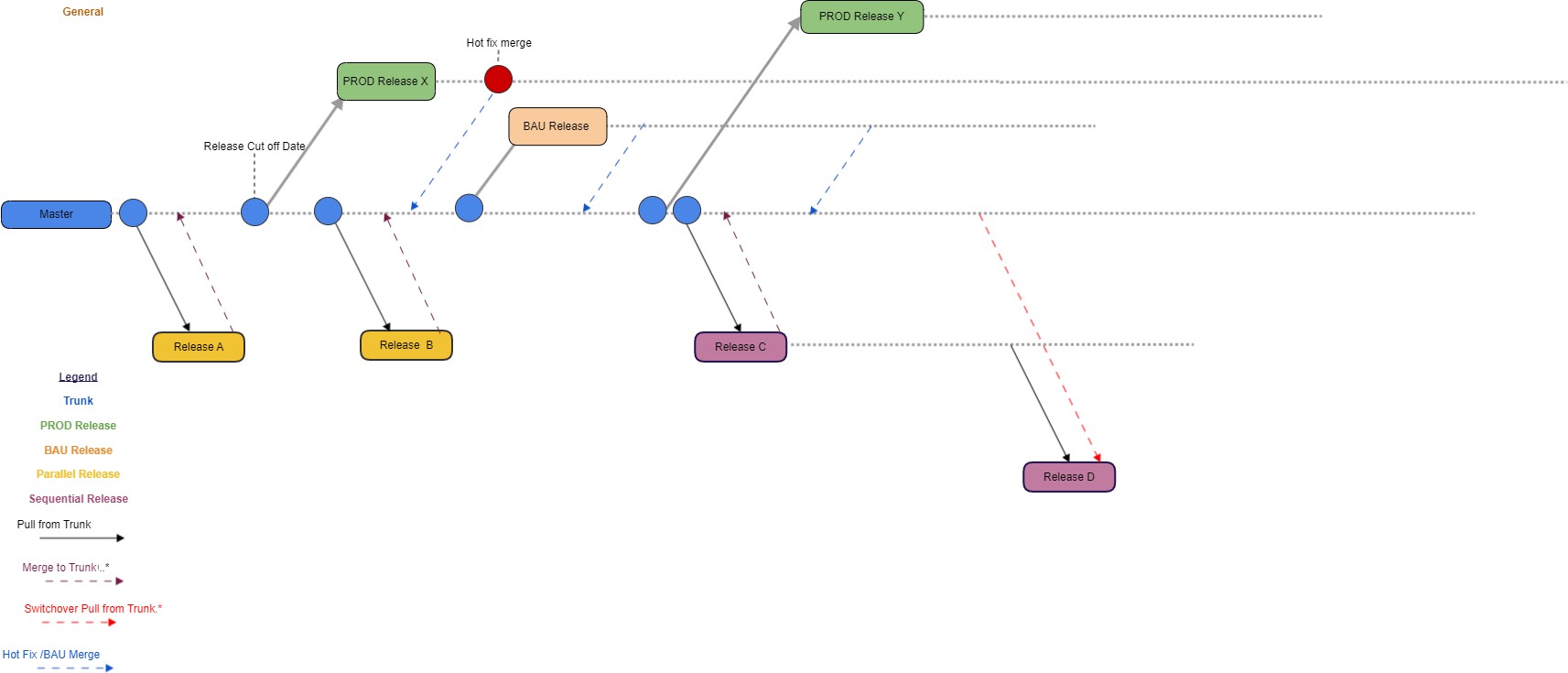
Publishing - Push the image to nexus 3

Uses Third Party tool to perform vulnerability scanning on our docker container Gunit (**QG**)- Run the Gunits test

Deploy (**QG**)- A notification is sent to an MS teams channel to communicate that a deployment has started. "envx deployment notifications". The container is deployed to the target environment. Run a sanity check to ensure the app is up.

Declarative Post Actions (**QG**) - Perform a cleanup of any files no longer required. Remove docker containers that have been deployed

# Branching Strategy



## Development Approach :

**1.** We follow Trunk based development with short-lived dev/ feature branches to merge back to master. This enables continuous delivery and creates less merge conflicts.

## Release Development Strategy :

1. Feature branches will be created for each release with Trunk as base and will periodically update from Trunk and get merged to Trunk at the end.
2. Parallel branches can be created for independent release which enables parallel development for independent release.
3. Dependent feature branches are needed for sequential releases with stretched and dependent functionalities. In this case Release C shows a stretched and have extended functionalities from Release B so dependent branch will be created from the parent

**a.** Dependent feature branch will be constantly get updates from master which has the updates from the dependent branch and other also from other features merged to Trunk

## Future State Release Strategy :

**1.** Aiming to implement a toggle feature leading to minimal short-lived feature branches. Solution design is in-progress and are reviewing the candidates which can fit as a result .

## Production Release :

1. Each Production Release is taken off Master and spinned into a new version, Ex: RX.Y.Z (X - Release Number, Y - Version, Z- Sub-Version if any)
2. Hotfix fix will be off respective release available in PROD and merged back to master.