Source Code Management (SCM) and Continuous Integration (CI) Strategy for ACME

January 26, 2018

Geoff Rosenthal ([Geoffrey.Rosenthal@perficient.com](mailto:Geoffrey.Rosenthal@perficient.com))

Sean Wilbur ([Sean.Wilbur@perficient.com](mailto:Sean.Wilbur@perficient.com))



© 2018 Perficient, Inc. All Rights Reserve

Table of Contents

[1. Process Overview 3](#_Toc504745127)

[2. Tool Access 3](#_Toc504745128)

[3. GIT 4](#_Toc504745129)

[1 branch guidance 4](#_Toc504745130)

[2 branch guidance: 4](#_Toc504745131)

[Repository Guidance: 5](#_Toc504745132)

[4. Jenkins 5](#_Toc504745133)

[Node Labels: 6](#_Toc504745134)

[Jenkins Jobs 6](#_Toc504745135)

[5. Sonatype Nexus IQ 7](#_Toc504745136)

[Scanning Stages 7](#_Toc504745137)

[Web Application Overview 7](#_Toc504745138)

[6. Sonatype Nexus Repository Manager 9](#_Toc504745139)

[Groups and Repositories 10](#_Toc504745140)

# Process Overview

ACME is moving towards a repeatable model where artifact packages are specifically versioned based upon a release number and time of build. That version is then stored in asset management for deployment to any target environment. This concept is known as build once, deploy anywhere i.e. a single built package can be deployed to any target environment (DEV, QA, UAT, PROD).

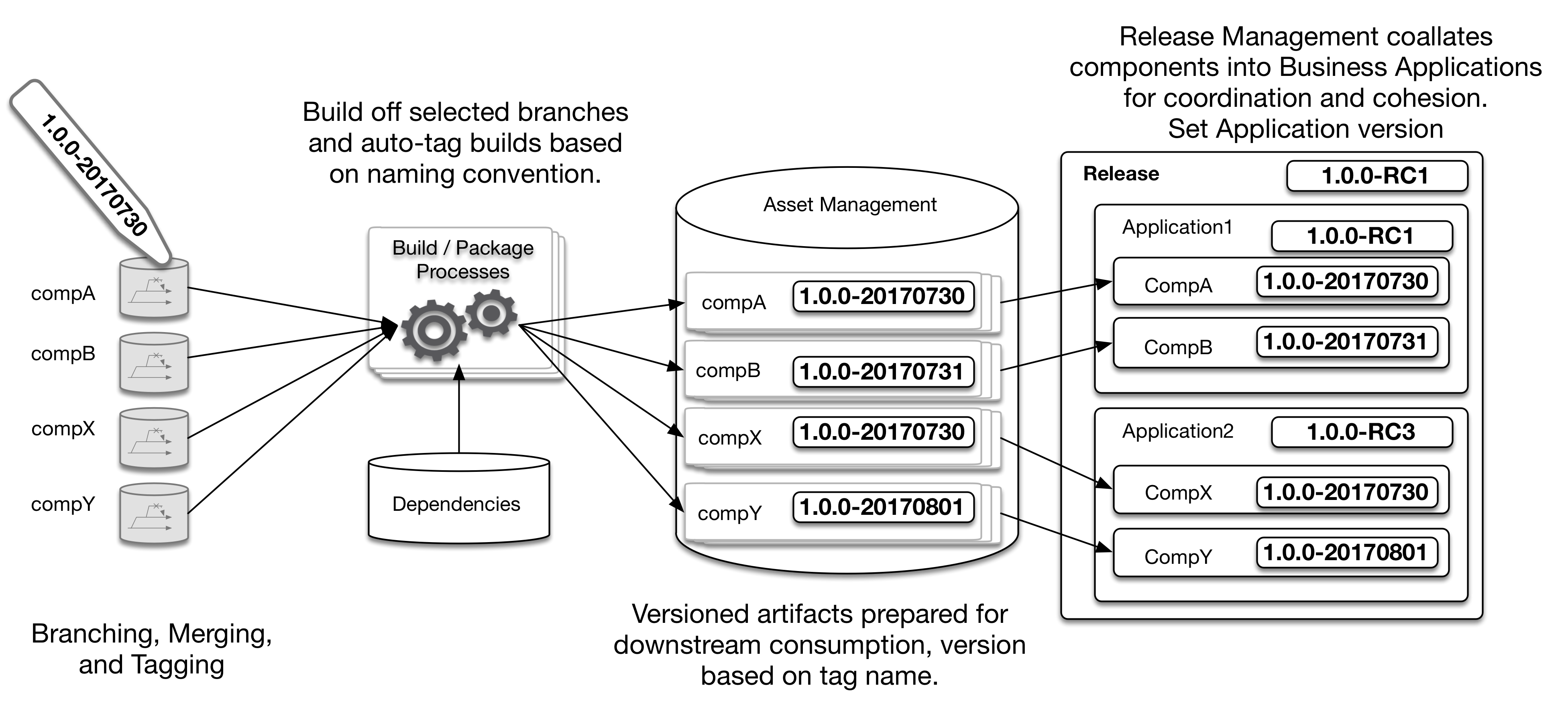


Figure : Versioning Strategy

Within ACME, for Node.js projects GIT is used for SCM, Jenkins is used for CI, Sonatype Nexus IQ (IQ) is used for 3rd party security scanning, and Sonatype Nexus Repository Manager (Repo Manager) is used for a package repository.

# Tool Access

For all DevOps tools you should be able to login with your ACME domain credentials. For any access issues contact Tool Admin (admin@acme.com) on the DevOps.

The following tools can be accessed within the ACME lab environment:

GIT project: https://github.com/project

-Login based upon Github id/password and project organizations

Jenkins: <http://jenkins.acme.com/jenkins>

Nexus IQ: <https://nexusiq.acme.com>

Nexus Repo Manager: <https://nexusrepo.acme.com>

# GIT

As of <a date> the SCM strategy had been applied to <a project>. This section will specifically discuss the <a project> repository. The intention was to repeat this strategy and process with all ACME applications as project priorities allow. Having 1 long running branch is the preferred management strategy. However, some teams voiced that they were not comfortable with a single branch strategy, thus a 2 branch strategy is also acceptable.

With either strategy:

* Development teams should be using a strategy that limits branches to only a long running Master branch and Develop branch
* Feature branches should be limited to a short lifespan (no longer than a day or so) and they should be deleted immediately when they are no longer needed

## 1 branch guidance

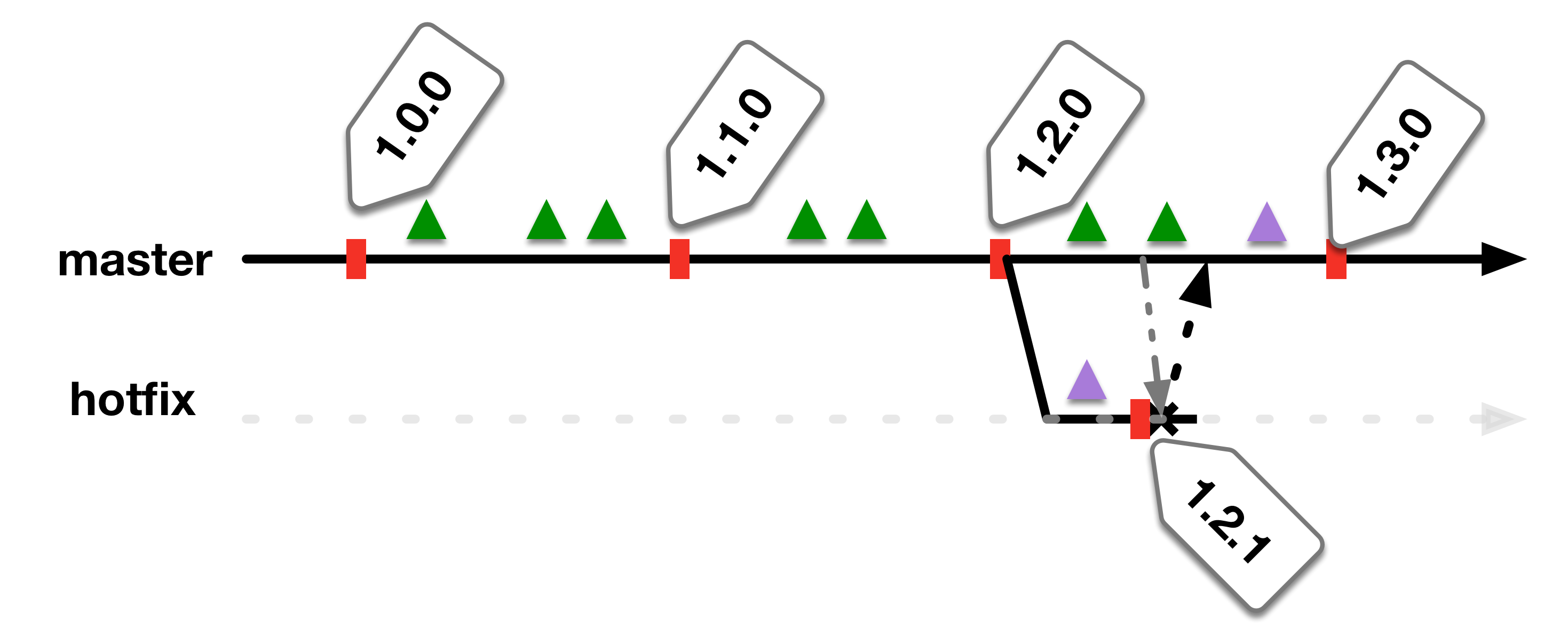


Figure : Single Branch Strategy

## 2 branch guidance:

* This strategy provides for development work (develop branch) and release readiness work (master branch) in parallel

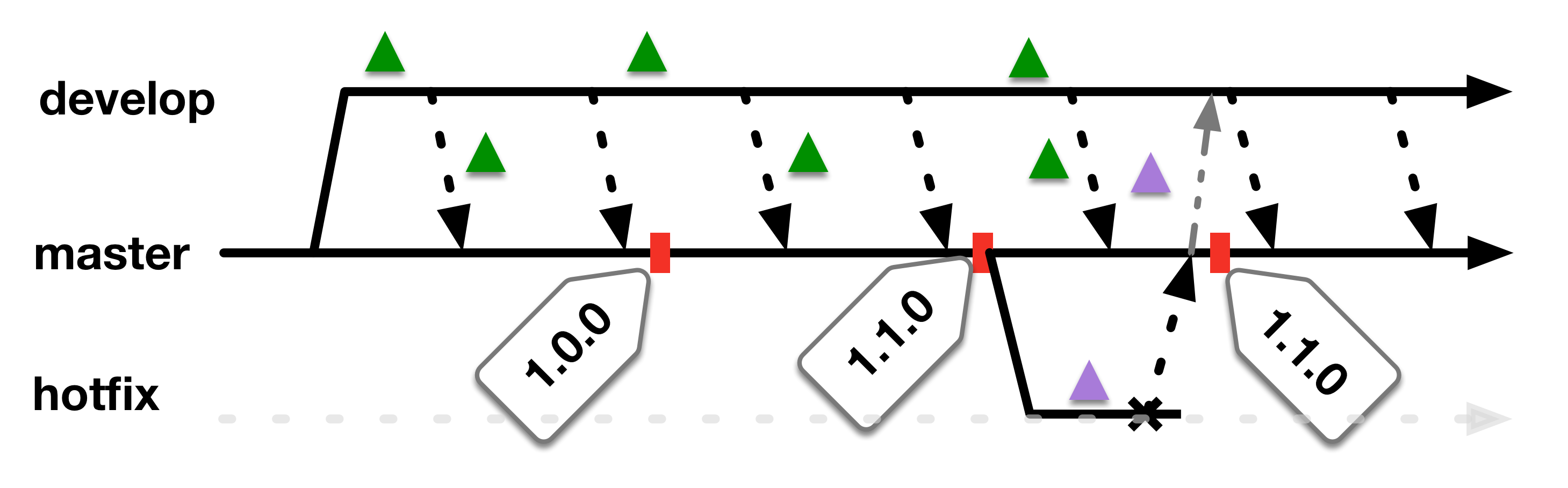


Figure : 2 Branch Strategy

## Repository Guidance:

* For Node apps, Package.json **MUST** be kept up to date with release semantic versioning. Work with your Product Manger to ensure you are using the correct semantic versioning for the current product lifecycle
* Builds will be created **ONLY** from the master or develop branches
* **Jenkinsfile-ci** is intended to be a CI build definition as a unit test for the developers. This build definition will continuously checkout source, run npm commands for build, IQ scan and upon successful build and scan it will deploy **<APP>** to the DEV environment. **\*\*\*Note these builds are intended to be immediate throw away.\*\*\***
  + Builds take place regularly on a set time
  + This CI Jenkins process will run an IQ scan for the IQ “Build” stage. The IQ report should be accessible from the link in the completed Jenkins job
  + The master branch will deploy to the DEV environment
  + The develop branch will deploy to the DEV2 environment
* **Jenkinsfile** is intended to be a release candidate (RC) build definition. When run, this build will checkout source, run npm commands for build, IQ scan, and upon success of those steps it will package the build into a release candidate, upload the package to Repo Manager, and tag the GIT repository with the build version.
  + RC build frequency must be defined by the development team
  + The build version label will be reflected in the package label and Repo Manager artifact label.
    - Label: <artifact>-<semantic version>-<build timestamp> e.x. App-2.0.1-20171211-105718
  + The RC Jenkins process will run an IQ scan for the IQ “Build” stage. The IQ report should be accessible from the link in the completed Jenkins job
  + When the job completes, you should see the newly added tag in the GIT repository
  + When the job completes, you should see the new artifact version in the Repo Manager “ACME-RC” repository
* The **deploy** folder within the **<APP>** repository contains the deployment job information. Currently we are using Jenkins as a stop gap deployment solution until a deployment tool is selected and implemented.
  + **Jenkinsfile-deployment** is a generic deployment job. The job requires input parameters for the source Repo Manager repository, version to deploy and target deployment environment. The job will then, on the requested deploy target environment, download the requested artifact version from Repo Manager, extract it, and deploy it using the deploy-nodeapp.sh script.
  + **deploy-nodeapp.sh** is the actual deployment script. **\*\*\*NOTE\*\*\*** that this script is intended to be used to deploy the **<APP>** into production. Any changes/updates to this script should be discussed with Ops. This script should be used so that the same deployment steps are used in the lower environments and in production.

# Jenkins

Jenkins has been updated to follow a distributed builds architecture (<https://jenkins.io/doc/book/architecting-for-scale/)>. The Jenkins master has been upgraded and enhanced with various plugins to try to make a more robust, easy to use system.

Jenkins nodes (a.k.a slaves) have been added to the system, and node labels are being leveraged. For a given job, the appropriate node labels should be specified to limit the job from only building on the intended node(s).

The deployment target labels should only be used for deployment jobs i.e. builds should not run on deployment nodes

## Node Labels:

LEGACY – used for all legacy build/deploy jobs before we moved to the new SCM and CI strategy

NodeJs – used to build Node.js applications

DEV – used as a deployment target to the dev environment

DEV2 – used as a deployment target to the dev2 environment

QA - used as a deployment target to the qa environment

QA2 - used as a deployment target to the qa2 environment

UAT - used as a deployment target to the uat environment

UAT2 - used as a deployment target to the uat2 environment

## Jenkins Jobs

**Deployment Folder**

All relevant Jenkins jobs for developers have been placed into the folder named “Development”. Here you will find the jobs for the CI and RC builds. You will also find the job to deploy to lower environments. These jobs are based upon Jenkinsfiles as specified:

Branch: Develop

APP – Build develop CI: Jenkinsfile-ci: creates throw away build on regular schedule that is deployed to dev

APP – Build develop RC: Jenkinsfile: creates build on regular schedule that is tagged in GIT and stored in RC repo

Branch: Master

APP – Build master CI: Jenkinsfile-ci: creates throw away build that is deployed to dev2

APP – Build master RC: Jenkinsfile: creates build that is tagged in GIT and stored in RC repo

APP – Deploy QA: deploy/Jenkinsfile-caller: deploys latest from RC repo to qa environment on regular schedule. Can be set to specific semantic version or specific version.

APP – Deploy QA2: deploy/Jenkinsfile-caller: deploys latest from RC repo to qa2 environment on regular schedule. Can be set to specific semantic version or specific version.

APP – Pipeline Deployment: deploy/Jenkinsfile-deployment: Allows user to manually select any release candidate version from Nexus Repo and deploy that version to any lower environment

**Administration Folder**

This folder contains jobs that are intended to have restricted access (for example the UAT deployment job). All Jenkins users should be able to view this folder so that developers can see history (like UAT deployment history). Only the administrator user has access to run these jobs.

# Sonatype Nexus IQ

IQ is a tool which scans open source packages in built applications for known security vulnerabilities and policy violations (e.x. out of date EULAs).

## Scanning Stages

IQ provides stages for a given application to run scans and provide a report against. The stages are: build, stage, release, and operate. The following IQ stage mapping was created:

For one branch SCM model( like iOS/Android )

APP - Build master CI --- Build

APP - Build master RC --- Release

For two branch (4 build) model ( like OLB below )

APP - Build develop CI --- Develop

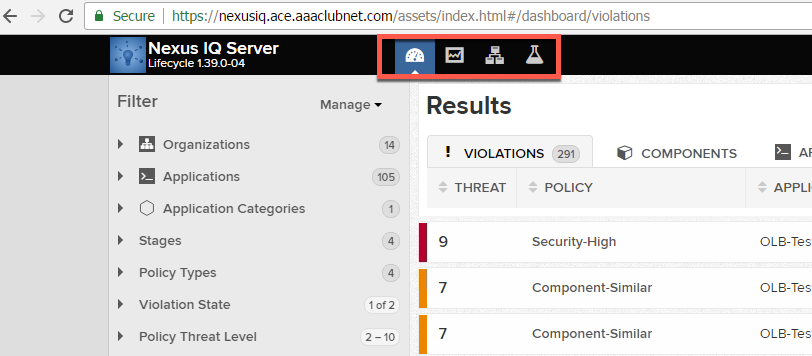
APP - Build develop RC --- Stage Release

APP - Build master CI --- Build

APP - Build master RC --- Release

## Web Application Overview

Once logged in, you can navigate via the buttons across the top.



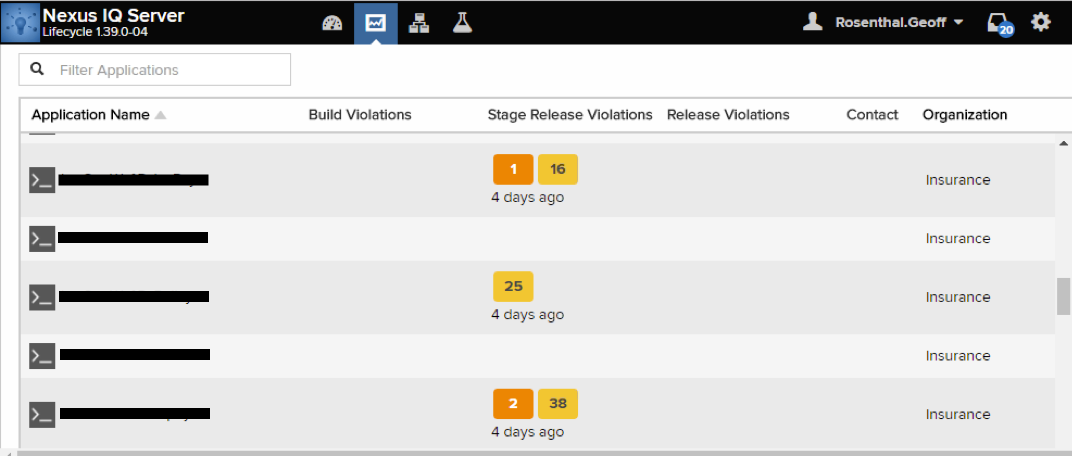
**Dashboard**

The dashboard view gives you access to view and search based upon filter criteria. Use this view if you know the name of what you’re looking for i.e. organization, application name, stages, policy types, violation state, or and/or policy threat level.



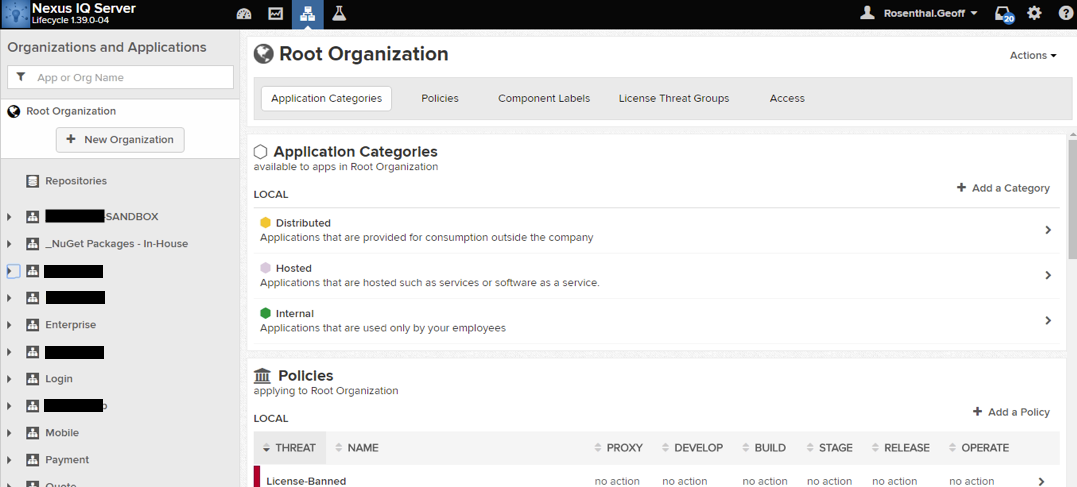
**Reporting**

The reporting view provides quick access to all of the application reports and is organized by application name and report stage (i.e. build, stage, release, operate). To get quick access to a report, type the application name into the application filter at the top left to view the reports for that application.



**Organization and Policies**

The organization and policies view displays how IQ has been organized with organizations, applications, policies, and user access. Scanning policies are displayed in order of inheritance first from the organization and then for a specific application.

****

# Sonatype Nexus Repository Manager

Repo Manager houses all source code packages including 3rd party packages and those developed in house. 3rd party packages, such as those form Nugget or NPM, are proxied from their open source repositories and identified as “proxy” repositories. ACME package repositories are identified as “hosted” repositories within Repo Manager.

## Groups and Repositories

ACME-RC – Stores all release candidate build packages that are created by eBiz development. This repository should have a fairly aggressive cleanup policy as once a RC package makes it through to production there is no need to keep all of the RC build packages from that particular release version.

* ACME-Releases – Includes all artifact versions that have been released to production
* NPM-Proxy – Proxy repository for packages from <https://registry.npmjs.org>
* Nugget – Proxy repository for packages from http://www.nuget.org/api/v2/