Continuous Delivery with Jenkins

DevOps Training

Continuous Delivery

Continuous delivery (CD) is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time. It aims at building, testing, and releasing software faster and more frequently. The approach helps reduce the cost, time, and risk of delivering changes by allowing for more incremental updates to applications in production. ***A straightforward and repeatable deployment process is important for continuous delivery.***

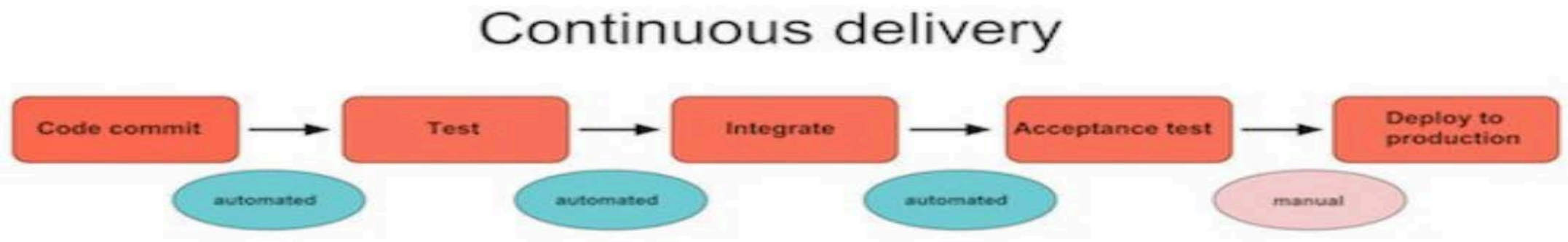
CI vs CD

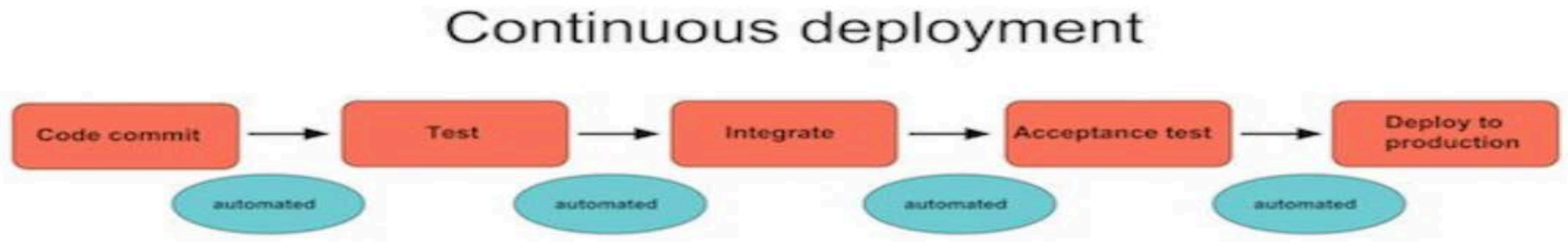
- Continuous Integration usually refers to integrating, building, and testing code within the development environment. Continuous Delivery builds on this, dealing with the final stages required for production deployment

- Continuous Delivery is a software engineering approach in which teams produce software in short cycles, ensuring that the software can be reliably released at any time. It aims at building, testing, and releasing software faster and more frequently.

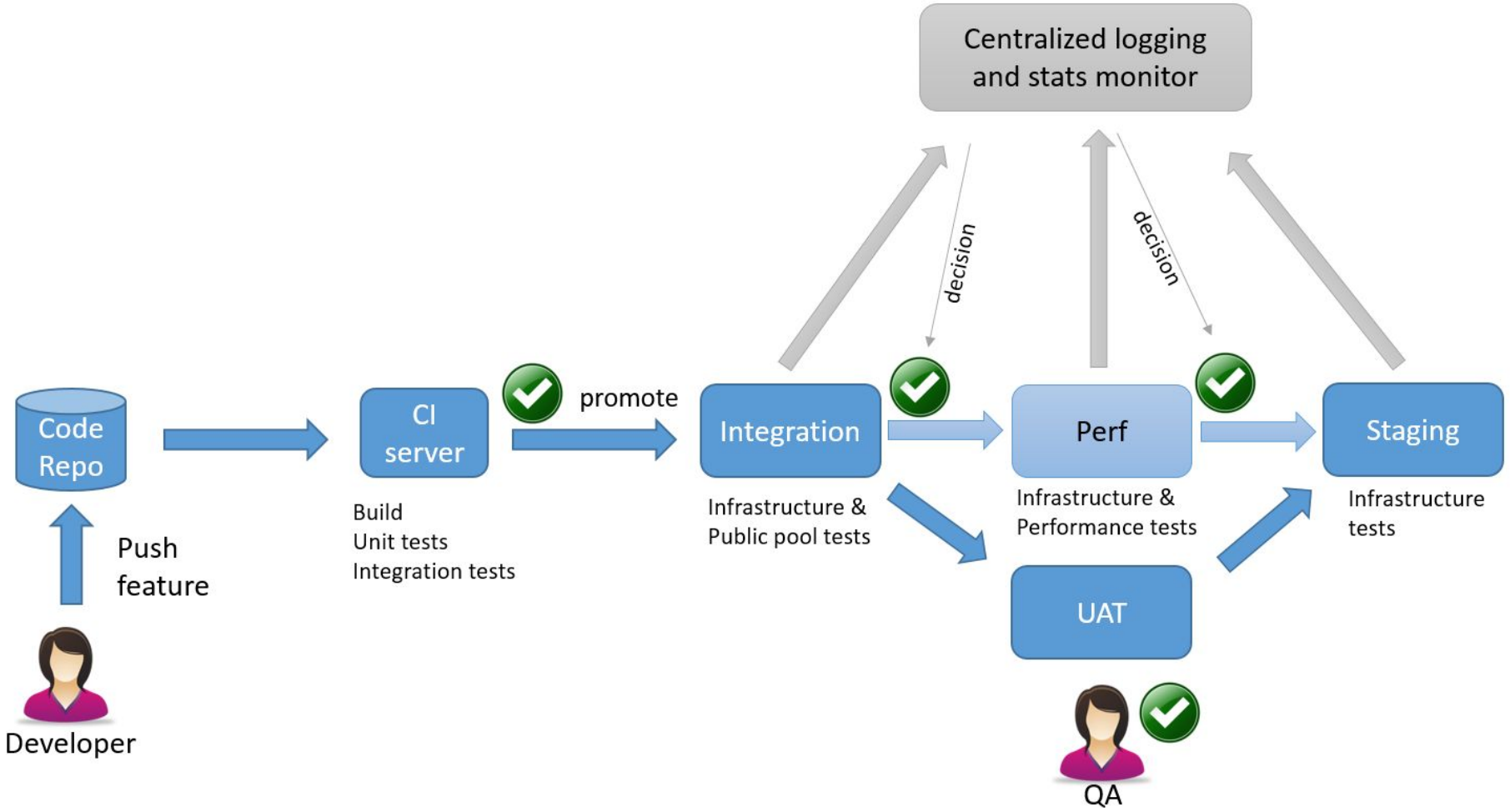
- Continuous Deployment means that every change goes through the pipeline and automatically gets put into production, resulting in many production deployments every day. In order to do Continuous Deployment you must be doing Continuous Delivery

CI Vs CD

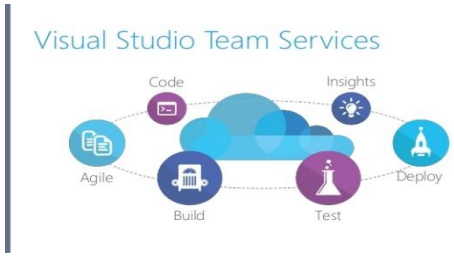

Continuous delivery
Code commit
Test
Integrate
Acceptance test
Deploy to production
automated
automated
automated
manual


Continuous deployment
Code commit
Test
Integrate
Acceptance test
Deploy to production
automated
automated
automated
automated

How it Works


Centralized logging and stats monitor
decision
decision
Code Repo
Push feature
Developer
CI server
Build
Unit tests
Integration tests
promote
Integration
Infrastructure & Public pool tests
Perf
Infrastructure & Performance tests
Staging
Infrastructure tests
UAT
QA

CI/CD Tool Options



Principles

Continuous Integration

- Automate the build
- Make the build self-testing
- Everyone commits to the baseline every day
- Every commit should be built
- Keep the build fast
- Test in a clone of the PROD environment
- Make it easy to get the latest deliverables
- Everyone can see the results of the latest build
- Automate deployment

Continuous Delivery

- The process for releasing/deploying software MUST be repeatable and reliable
- Automate everything
- If somethings difficult or painful, do it more often
- Keep everything in source control
- Done means “released”
- Build quality in!
- Everybody has responsibility for the release process.
- Improve continuously.

Best Practices

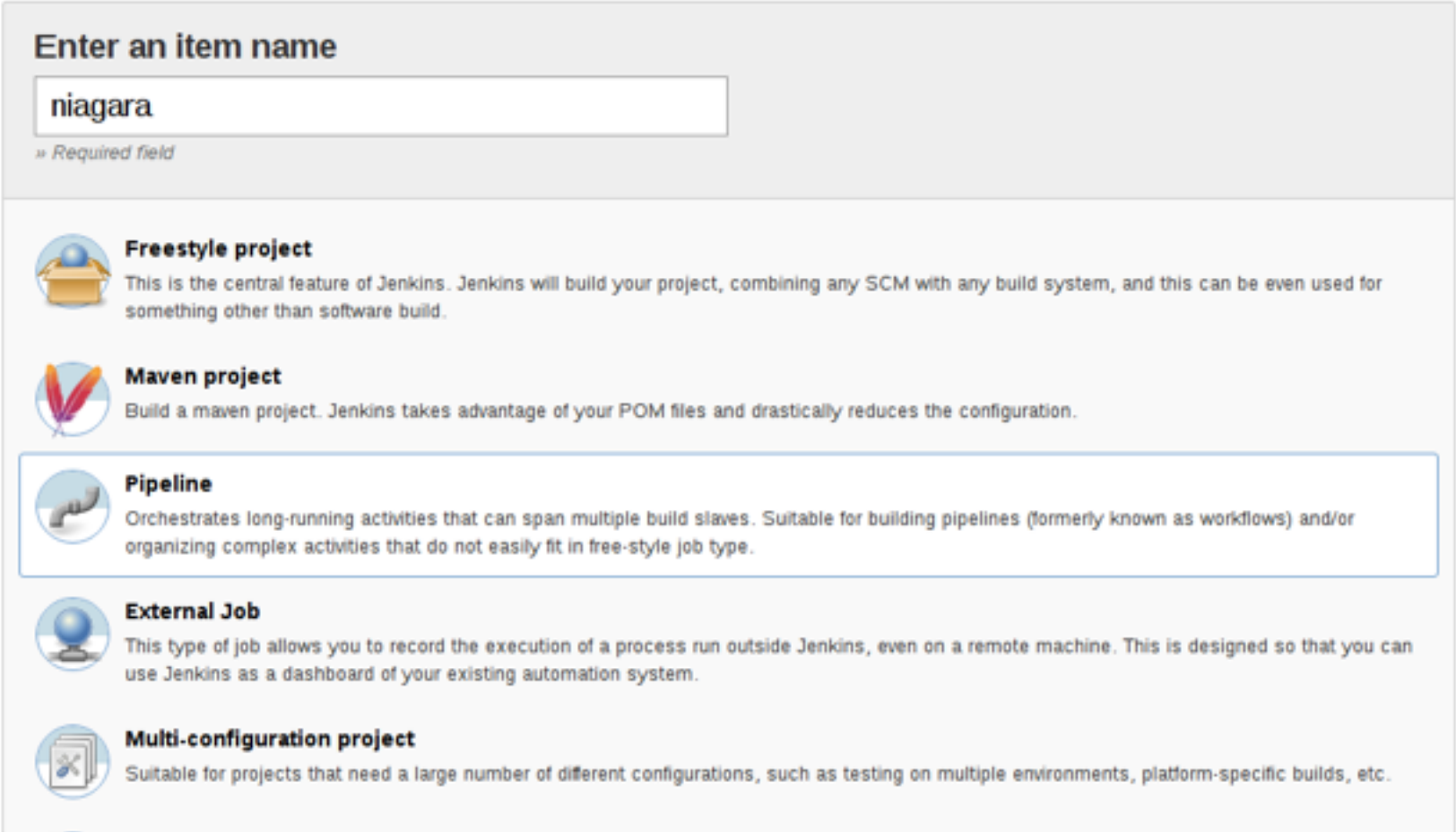
- **Continuous Integration**
  - Don't check-in on a Broken build
  - Always run all commit tests locally before committing, or get your CI server do it for you
  - Wait for commit test to **Pass** before moving on. Never go home on a Broken build
  - Time-box fixing before reverting (e.g. try to fix in 10 mins, if not then revert)
  - User TDD / BDD
- **Continuous Delivery**
  - Build the binaries only one
  - Deploy the same way to every environment
  - **Smock test** your deployment
  - Use a container technology (**Docker**) if possible as makes deployment simple

Jenkins 2.0

Pipeline as Code

- Introduce “Pipeline” as a new type in Jenkins
- Codify an implicit series of stages into an explicit pipeline definition (`Jenkinsfile`) in source control
- Resumability/durability of pipeline state
- Extend the DSL to meet organizational needs

Creating a Pipeline



Creating a Pipeline

Jenkins File

Jenkinsfile

```
node('java8') {
    stage('Checkout') {
        checkout scm
    }

    stage('Build') {
        sh 'mvn clean install'
    }

    stage('Test') {
        sh 'mvn test'
    }

    archiveArtifacts artifacts: 'target/**/*.jar', fingerprint: true
}
```

Snippet Generator

Steps

Sample Step: stash: Stash some files to be used later in the build

Name

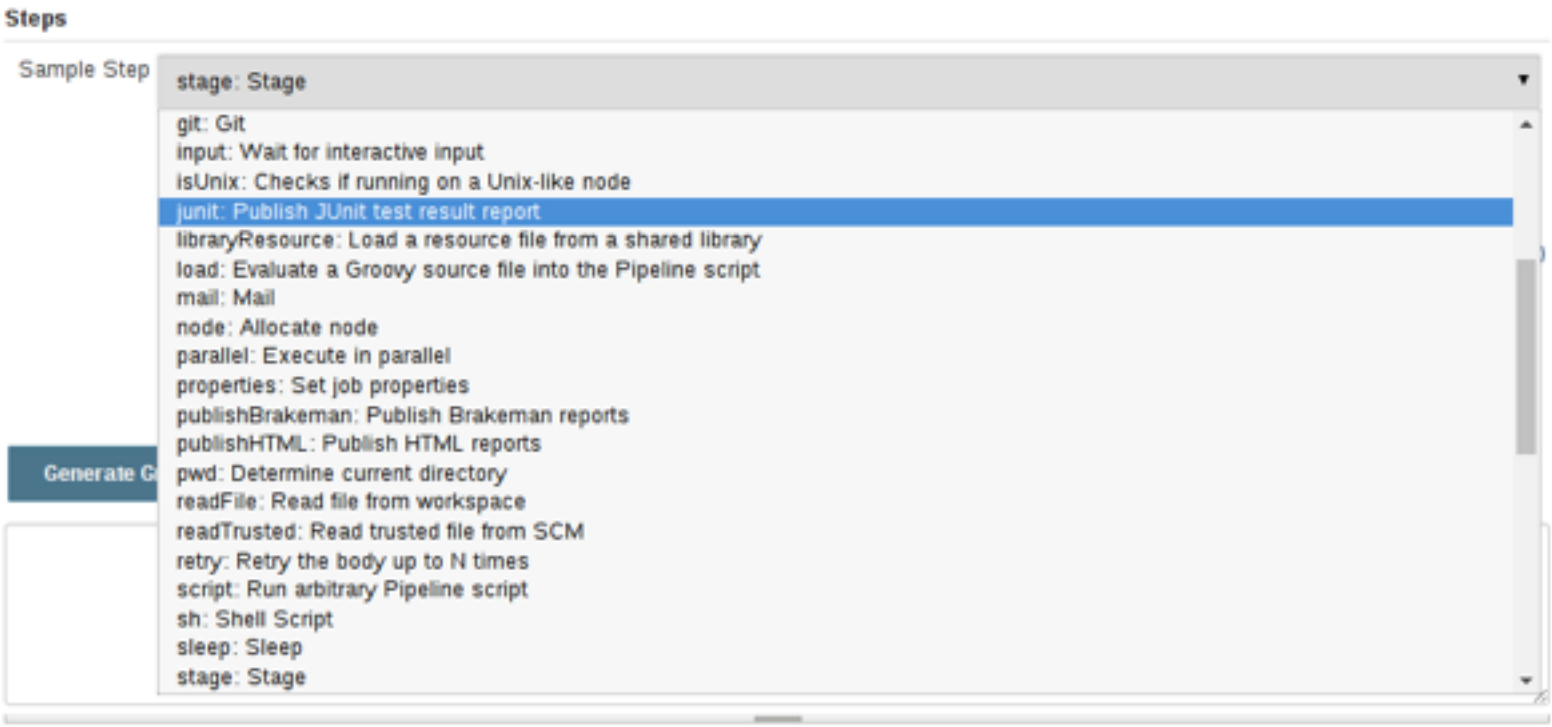
Includes

Excludes

Use Default Ant Excludes ☑

Generate Groovy

Snippet Generator

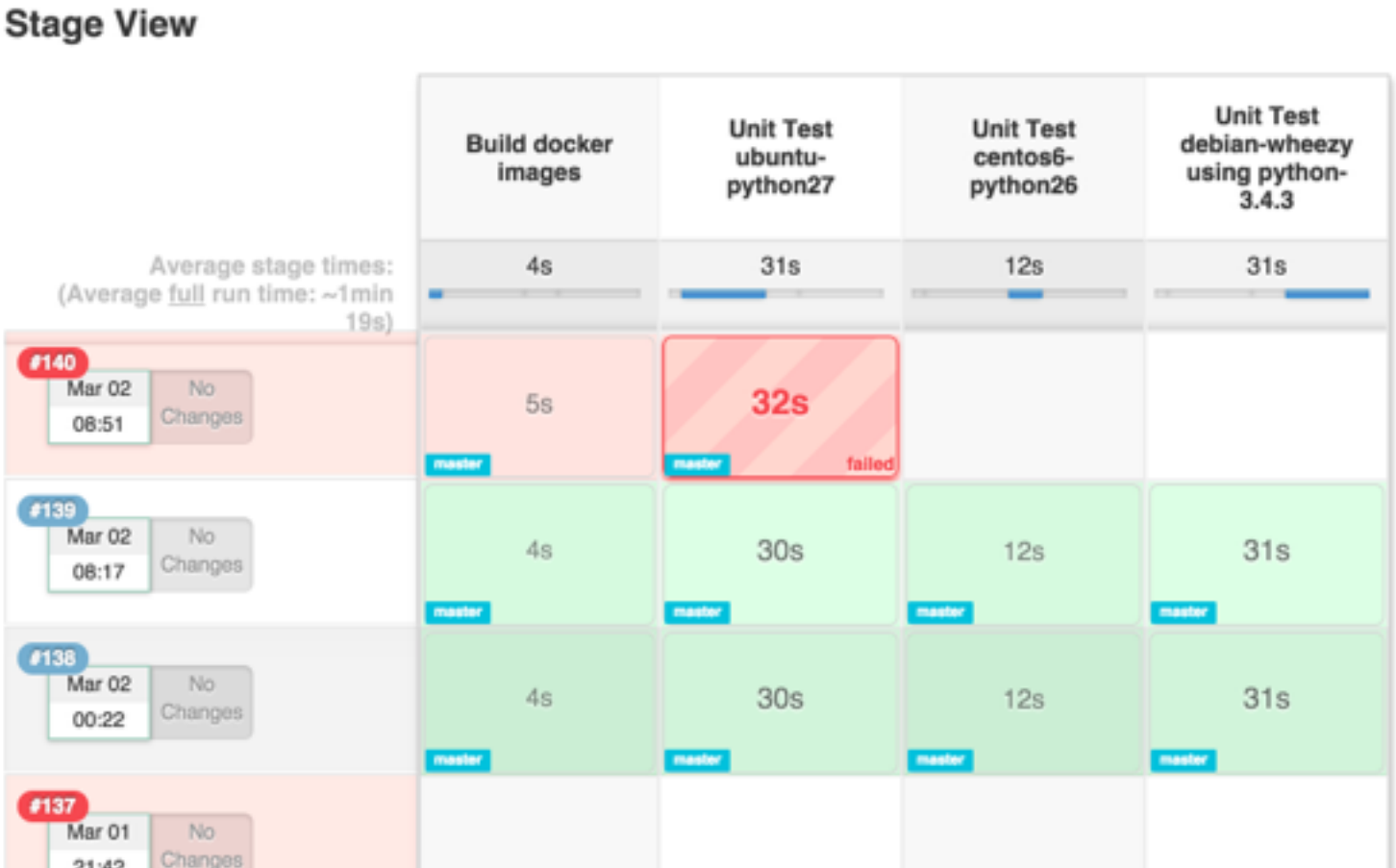


Pipeline Plugins

Pipeline Plugins

Extending Pipeline

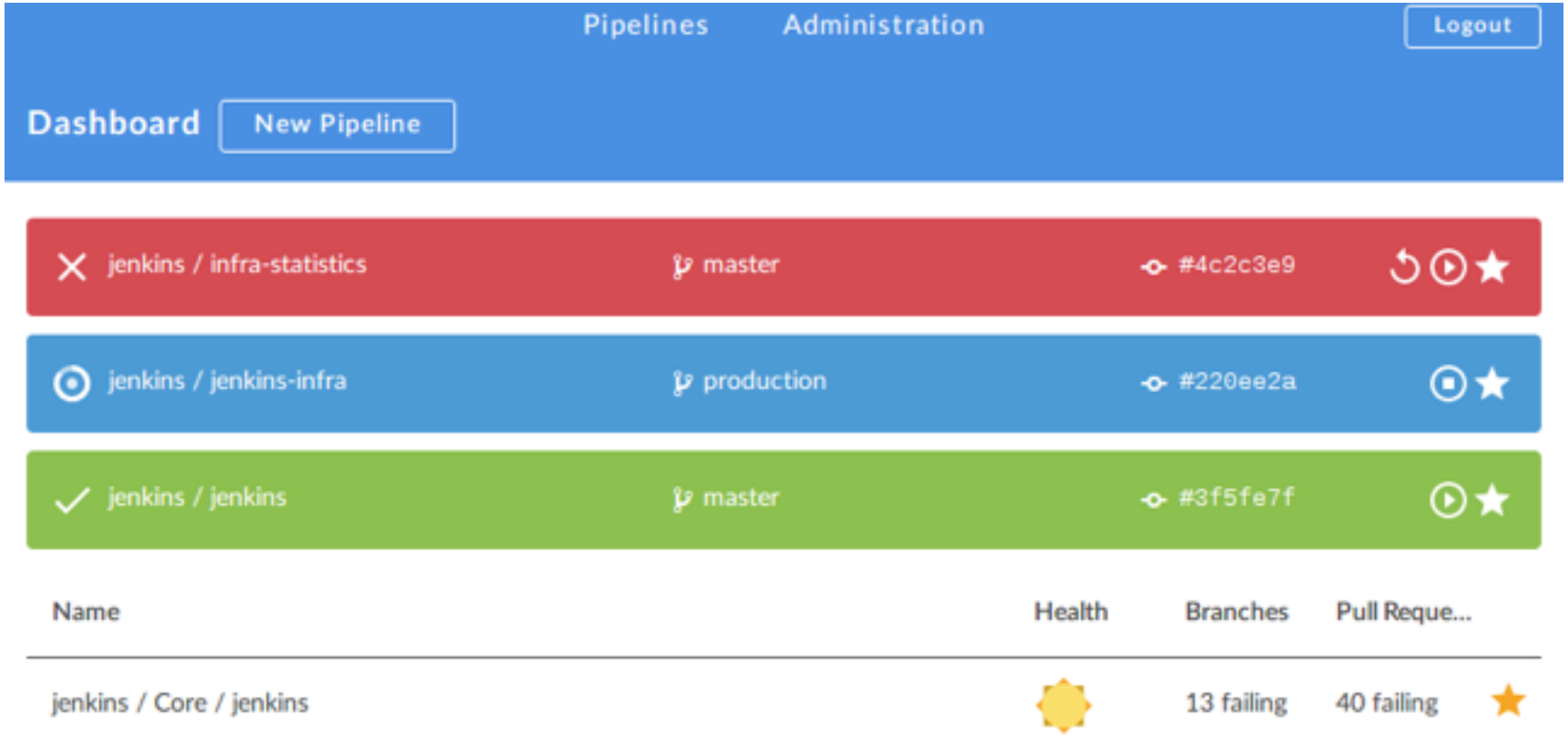
Pipeline Stage View plugin



Pipeline Multibranch plugin

- Represent pipelines for multiple branches in one "Folder"
- Automatically scan a repository for each branch which contains a `Jenkinsfile`

Blue Ocean


Pipelines
Administration
Logout
Dashboard
New Pipeline
jenkins / infra-statistics
master
#4c2c3e9
jenkins / jenkins-infra
production
#220ee2a
jenkins / jenkins
master
#3f5fe7f
Name
Health
Branches
Pull Reque...
jenkins / Core / jenkins
13 failing
40 failing

Plugins that support pipeline

- A few plugins which provide custom steps for Pipeline scripts:
  - Tooling: Credentials binding, SSH Agent, Parallel Test
  - Reporters: JUnit, Brakeman, HTML Publisher, Cucumber Test Result
  - Notifiers: Slack, HipChat, email-ext
  - Wrappers: Timestamper
- Plugins work within Pipelines
  - SCM: Git, SVN, Mercurial, P4, CVS
  - Wrappers: Ansi Color, NodeJS
  - Build steps: Ant, etc
- More defined in COMPATIBILITY.md

Build Steps

Steps

Sample Step: step: General Build Step

Build Step: Publish Javadoc

Javadoc directory: docs

Directory relative to the root of the workspace, such as 'myproject/build/javadoc'

☑ Retain Javadoc for each successful build

Generate Groovy

```
step([$class: 'JavadocArchiver', javadocDir: 'docs', keepAll: true])
```

Build Wrappers

Steps

Sample Step: wrap: General Build Wrapper

Build Wrapper: Add timestamps to the Console Output

Generate Groovy

```
wrap([$class: 'TimestamperBuildWrapper']) {
    // some block
}
```

Abstracting Pipeline

Sharing code, best practices and templates for success

Shared Library

A shared library is a collection of independent Groovy scripts which you pull into your Jenkinsfile at runtime.

First you create your Groovy scripts (see below for details on this), and add them into your Git repository.

Then, you add your Shared Library into Jenkins from the Manage Jenkins screen.

Finally, you pull the Shared Library into your pipeline using this annotation (usually at the top of your Jenkinsfile):

```
@Library('your-library-name')
```

THANK YOU!

training@laksans.com