O'REILLY®

Certified Jenkins Engineer (CJE) Crash Course

About the trainer

bmuschko

bmuschko

bmuschko.com

AUTOMATED ASCENT

automatedascent.com

PRACTICE EXAMS

https://www.udemy.com/course/certified-jenkins-engineer-practice-exam

Exam Details and Resources

Objectives, Exam Environment, Preparation

DISCUSSION

What’s your main learning objective?

Exam Objectives

“Design and build Continuous Delivery solutions with open source Jenkins”

For Jenkins engineers, earning certification helps you prove a level of Jenkins proficiency and skill.

🔗 https://www.cloudbees.com/cloudbees-university/training-certifications/jenkins

The Curriculum

- Jenkins Fundamentals (17%)
- Jenkins Administration (42%)
- Jenkins Build Technologies (30%)
- Jenkins Advanced Pipelines (11%)

🔗 https://university.cloudbees.com/jenkins-cje-cert-info/1747434

Candidate Skills

Jenkins

Concepts & Hands-On Expertise

Language Fundamentals

Underlying Concepts

Exam Environment

Proctored online, multiple choice exam

Computer with webcam required

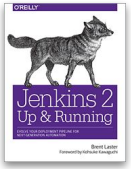
Access to internet or but no documentation

Preparation Resources

Cross-reference topics for more details

Certification exam guide

Self-paced courses on CloudBees University



Jenkins 2: Up and Running, O'Reilly Media

How to Prepare

Internalize by hands-on practice

Score easy points by understanding concepts

Practice with local Jenkins installation

Cross-reference resources in study guide

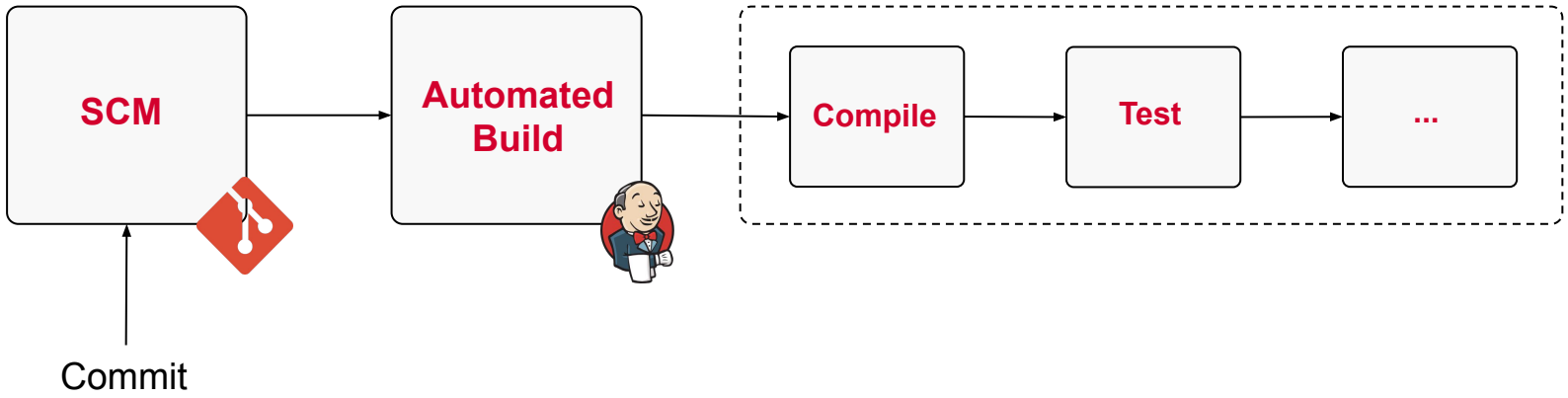
Q & A

Terminology and Jenkins Installation

Key Concepts, Getting Started

Continuous Integration (CI)

Run an automated build for every commit to SCM

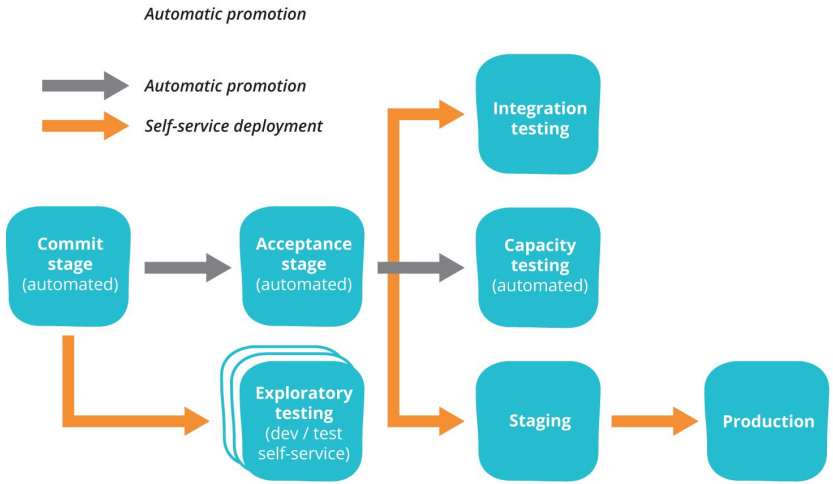


CI Best Practices

- Frequent commits
- No feature/bugfix long-running branches
- Automated test execution
- Fix broken builds
- Automate as much as possible

Continuous Delivery (CD)

Ensure that code is in a deployable state



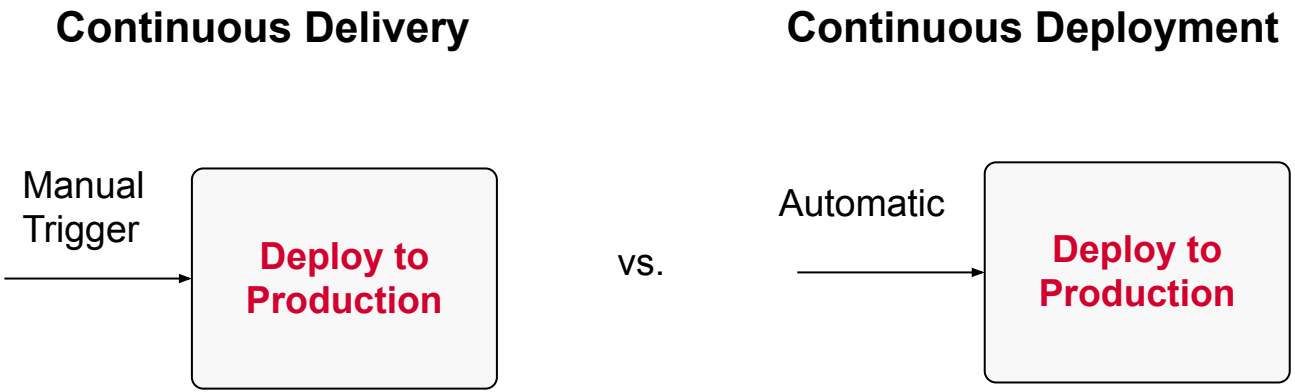
https://continuousdelivery.com/implementing/patterns/

CD Best Practices

- Version control all configuration
- Stop the line immediately for a failure
- Build your binaries only once
- Deploy the same way to all environments

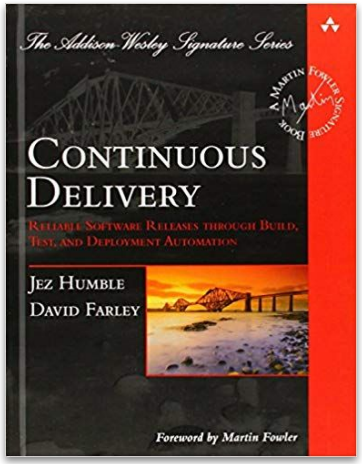
Continuous Deployment (CDE)

Automatic deployment to production



Additional Resources

THE reference guide on CI/CD/CDE



Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation (Addison-Wesley)

https://learning.oreilly.com/library/view/continuous-delivery-reliable/9780321670250/

Distribution Options

Pick the appropriate distribution for your needs

- OS-specific installer
- Executable WAR file
- Container image e.g. on Docker Engine

https://jenkins.io/doc/book/installing/

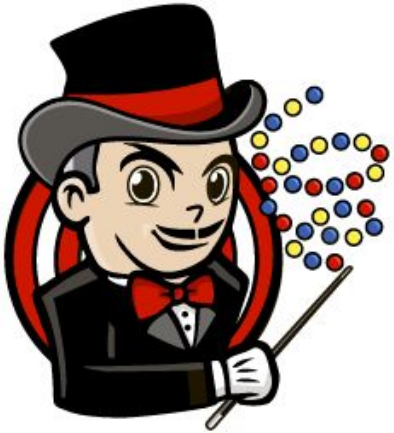
Prerequisites

Jenkins's runtime environment is the JVM

- Installation of JDK 11 or higher
- Set `JAVA_HOME` as environment variable
- Container image already contains JDK

Installation Wizard

Guides through installation process



creative commons

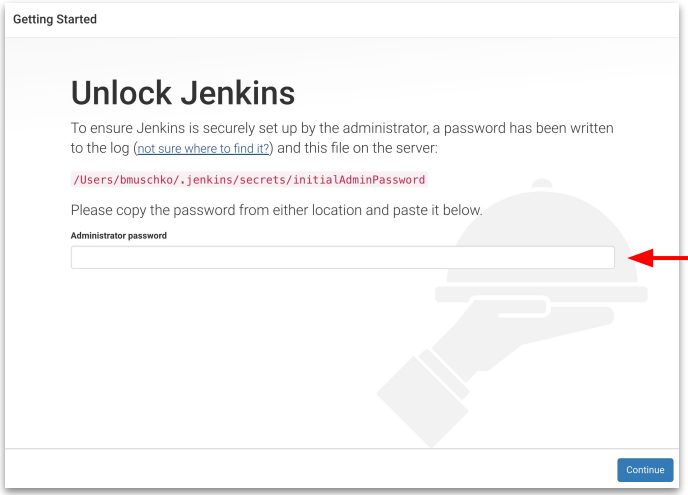
https://jenkins.io/artwork/

Accessible through web browser

http://<server-domain>:8080/

Unlocking Jenkins

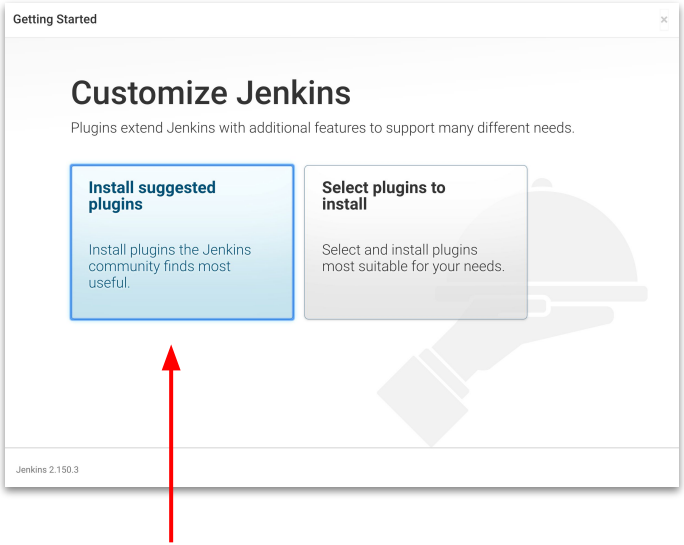
Auto-generated password on local disk

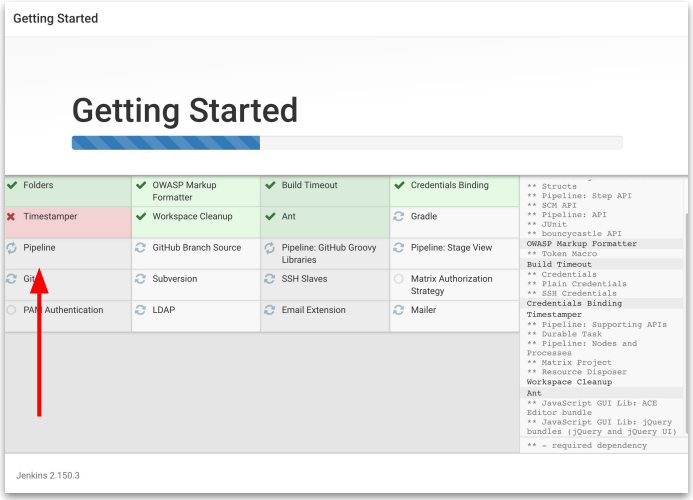



```
$ cat /Users/bmuschko/.jenkins/secrets/↵
initialAdminPassword
6ce47a19e64045bc87d33c3843f2254b
```

Initial Plugin Installation

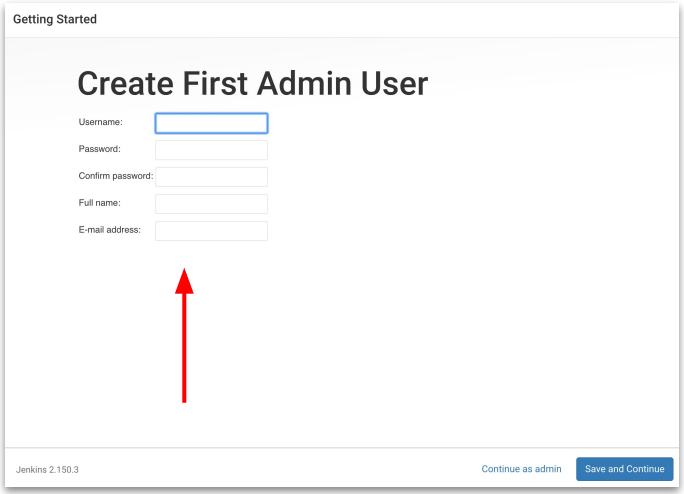
Suggested plugins include the Pipeline functionality

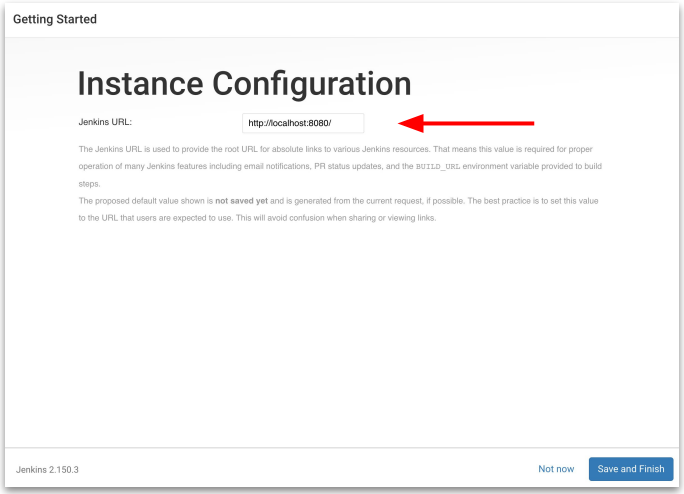




Optional Steps

Creating customized admin user and Jenkins URL





EXERCISE

Installing Jenkins and Exploring the Dashboard

Q & A

BREAK

Jobs, Builds and SCM Configuration

Definition, Usage and Options

What is a Job (aka Project)?

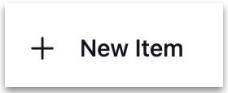
Definition of an executable task or organizational structure

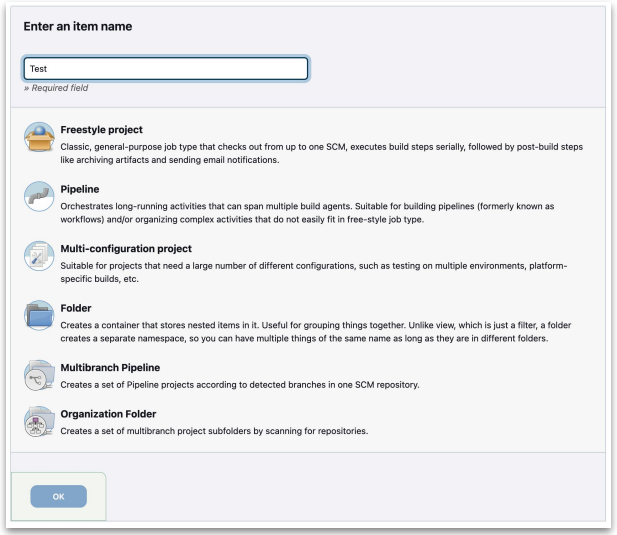
- Compiling a Java-based project
- Executing different types of tests
- Deploying an artifact to a target environment

Different Types of Jobs

Selected at the time of creation, not changeable later

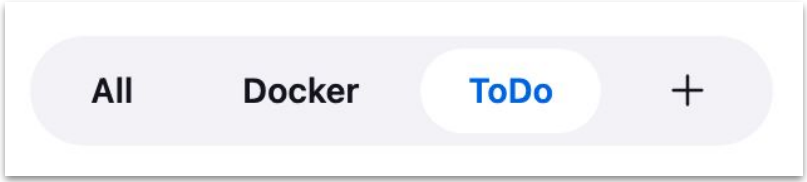
From Dashboard, click...



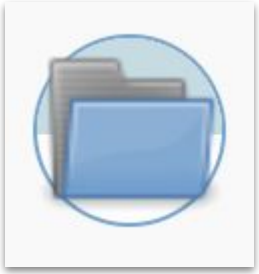


Organizing Jobs

“Grouping of jobs by teams or projects”



VS.

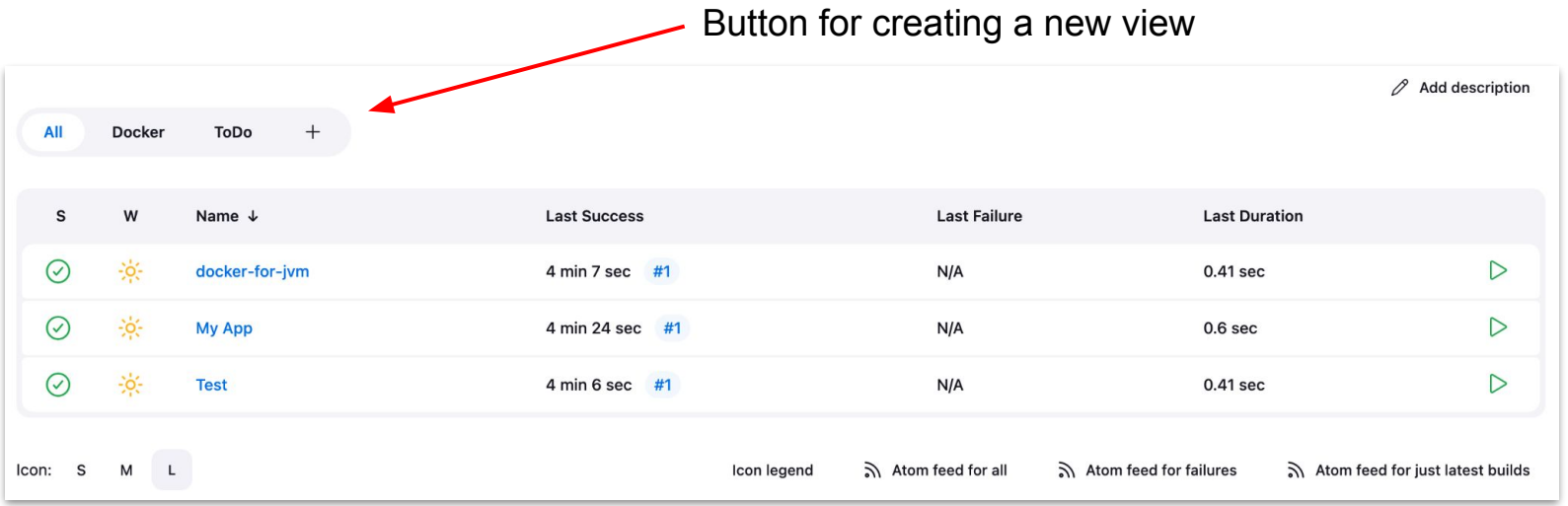


Views

Folders

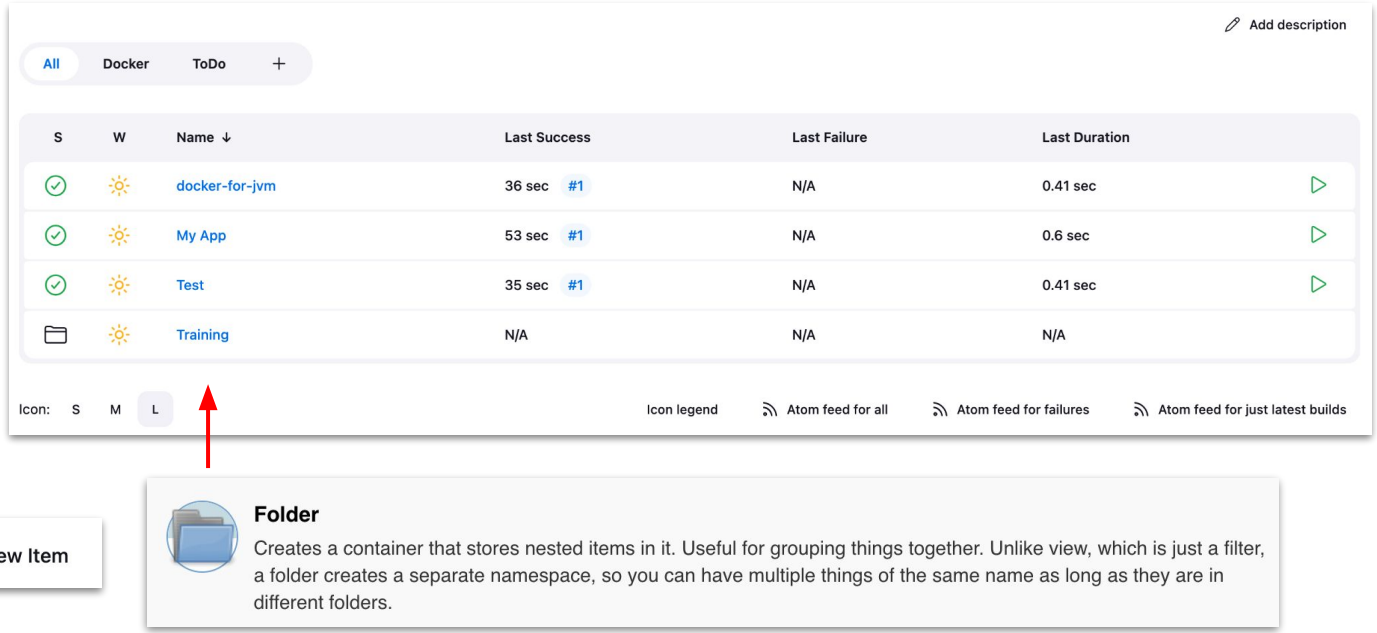
Using Views

Global and personalized flat lists



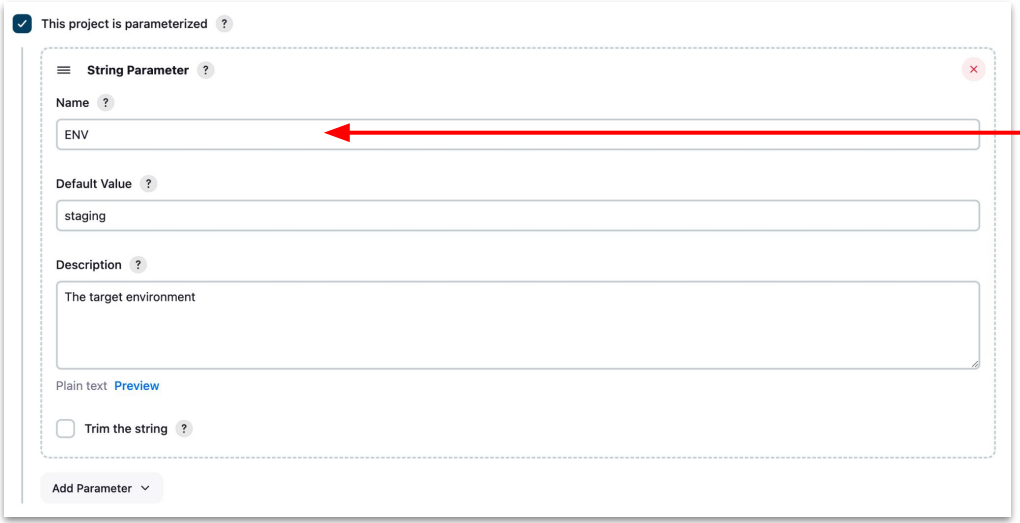
Using Folders

Nested organizational structure



Parameterized Jobs

Pass parameters to control runtime behavior



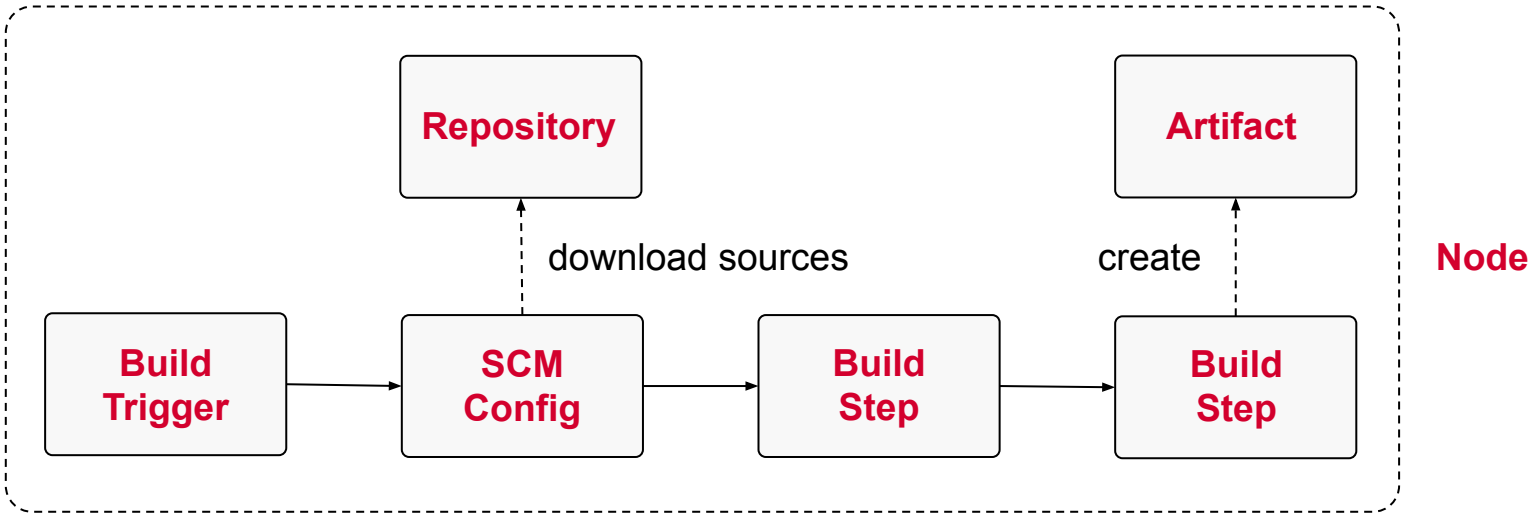
Accessible as environment variable named `ENV`

EXERCISE

Defining, Configuring
and Organizing a Job

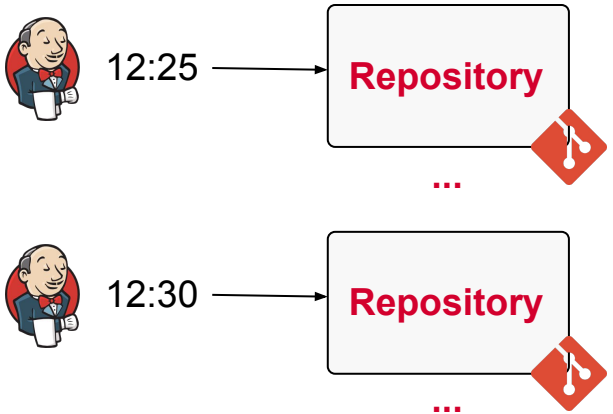
What is a Build?

Execution of an automation task



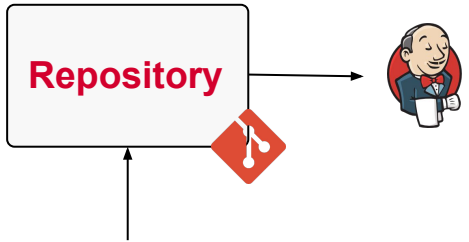
Defining Build Triggers

Periodic checking or direct notification upon a change



Pull every 5 mins

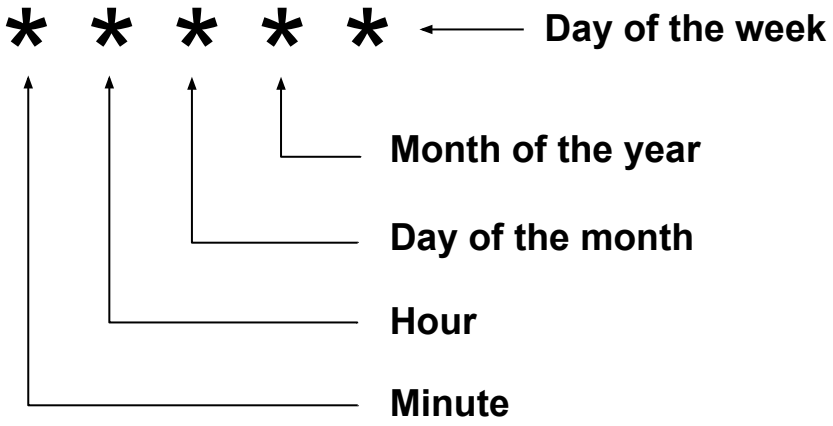
VS.



Push upon commit

Polling Definition

Follows typical Unix cron format (for the most part)



Every 5 minutes

```
*/5 * * * *
```

At minute 5

```
5 * * * *
```

Special Symbol “H”

Indicates hashed value for random value in range

```
H/5 * * * *
```

Every 5 minutes starting at random minute (e.g. :04, :09, :14…)

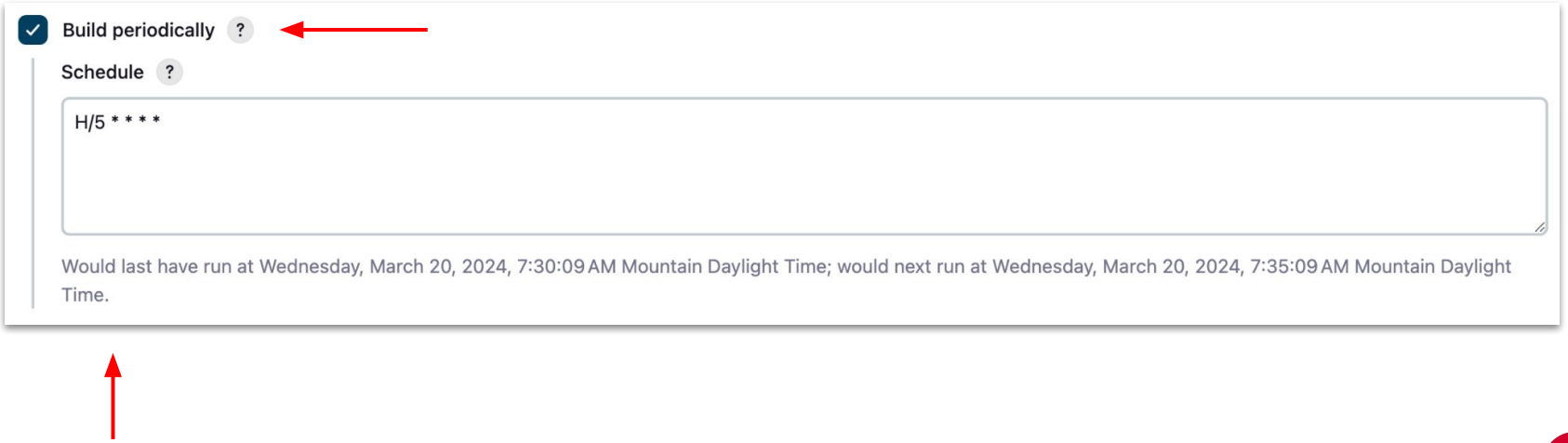
Distribute load without polling spikes a common intervals

Cron Alias Notations

Alias notations for cron definition	
`@yearly, @annually`	Every year
`@monthly`	Every month
`@weekly`	Every week
`@daily`	Every day
`@hourly`	Every hour
`@midnight`	At midnight

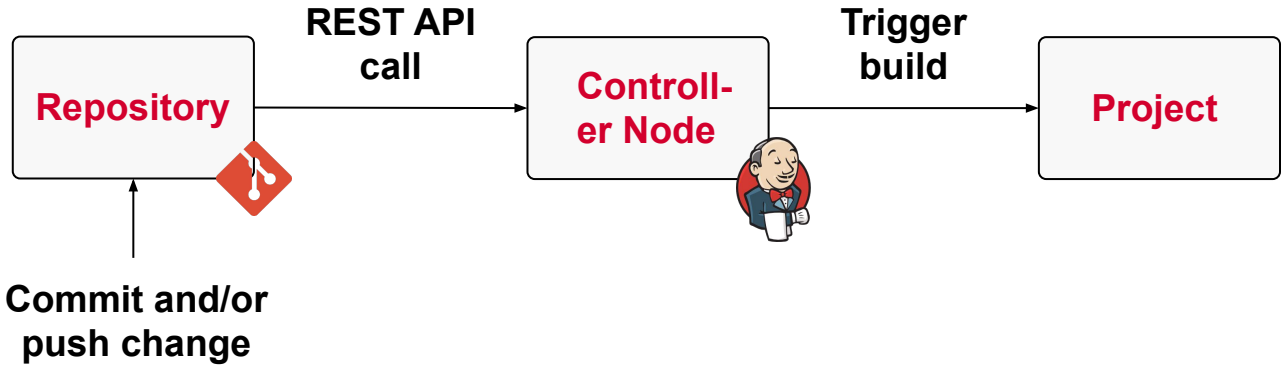
Defining a Periodic Build Trigger

Configure > Build Triggers > Build Periodically



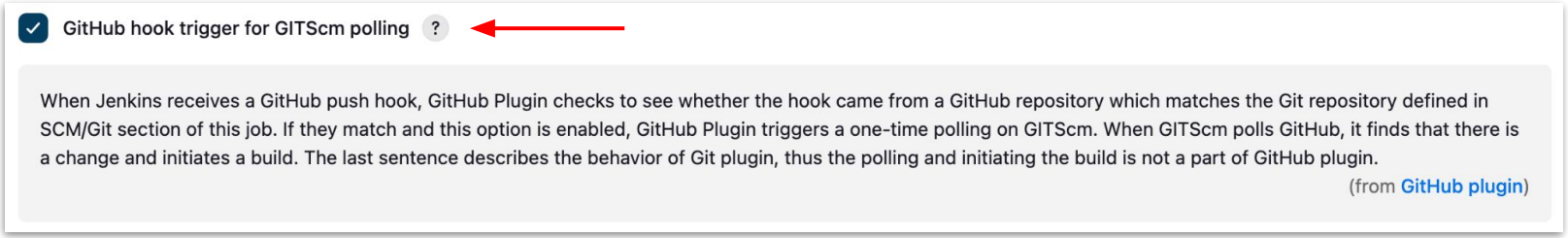
How Does Pushing Work?

Communication via Jenkins REST API



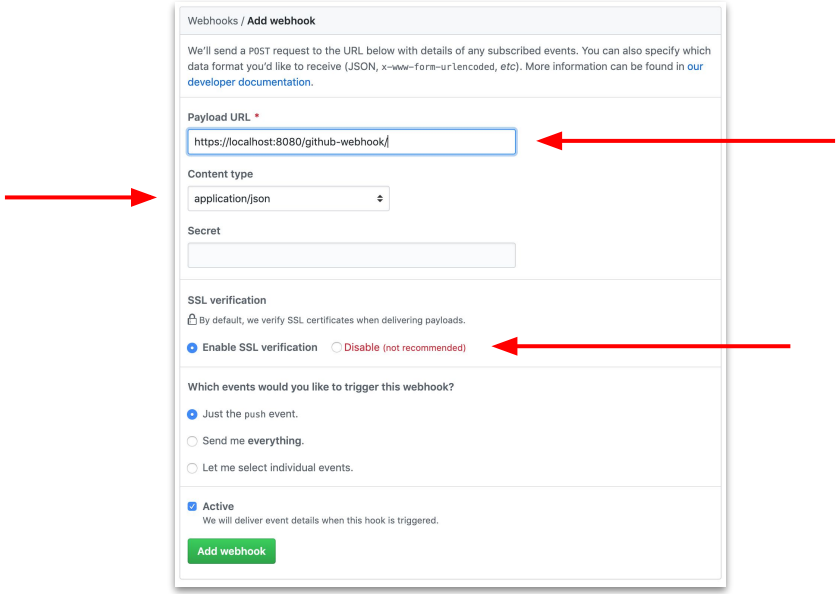
Configuring a GitHub Hook

Configure > Build Triggers > GitHub Hook Trigger...



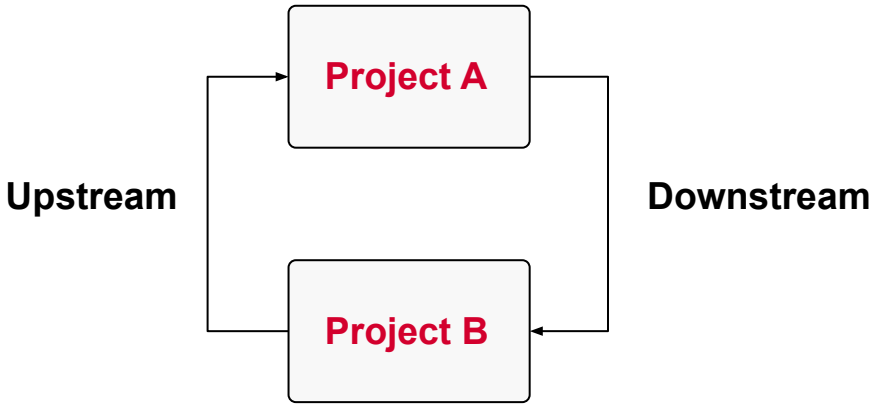
GitHub Webhook Configuration

Settings > Webhooks > Add webhook



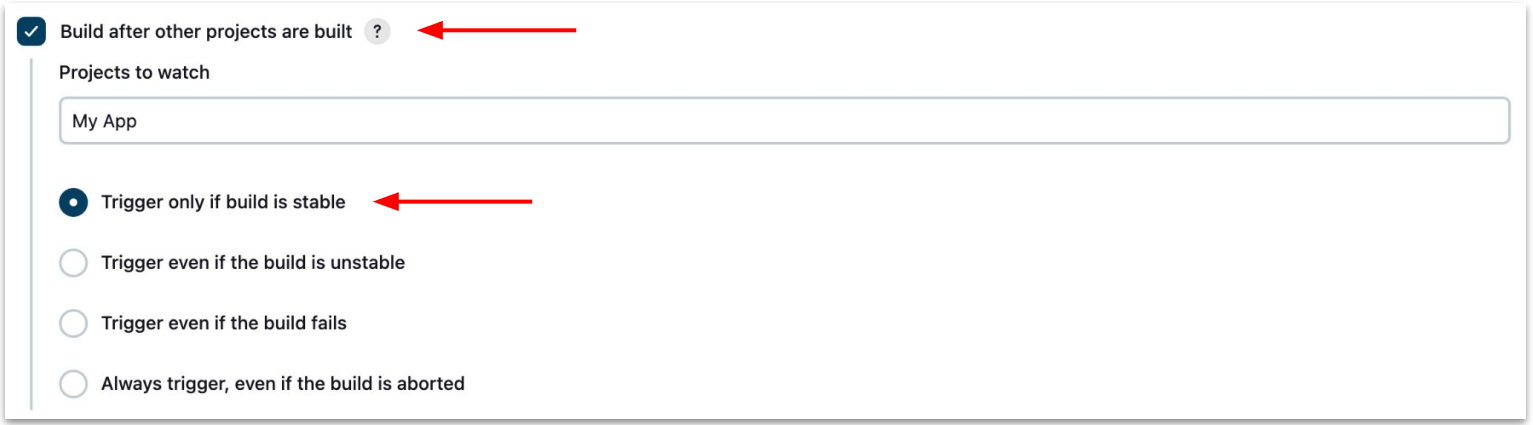
Relationship Between Projects

Sometimes projects do not live in isolation



Triggering Another Project

Configure > Build triggers > Build after other projects...



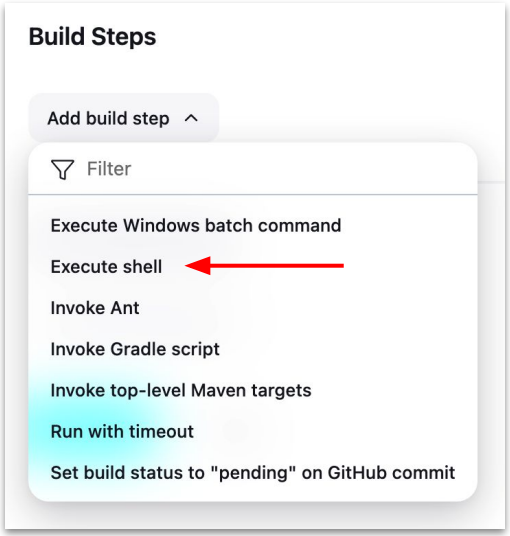
Examples for Build Steps

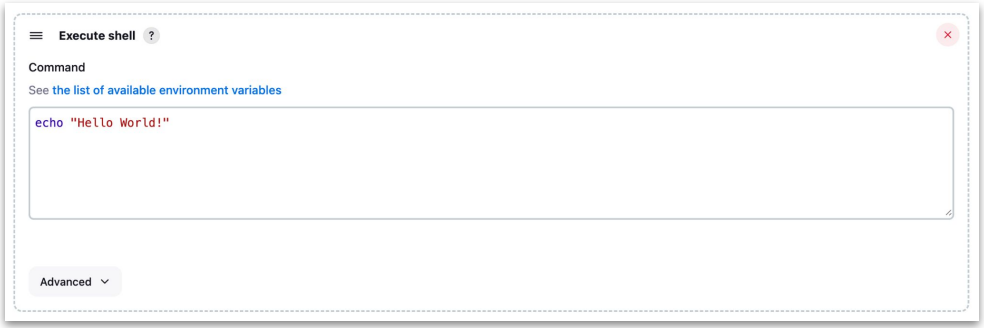
Executable actions upon triggering the build

- Run a build tool (e.g. Ant/Maven/Gradle)
- Execute a shell or batch command
- Running different types of tests

Configuring Build Steps

Configure > Add build step





Using Built-In Env. Variables

Available to all builds and usable through variable

Overview:

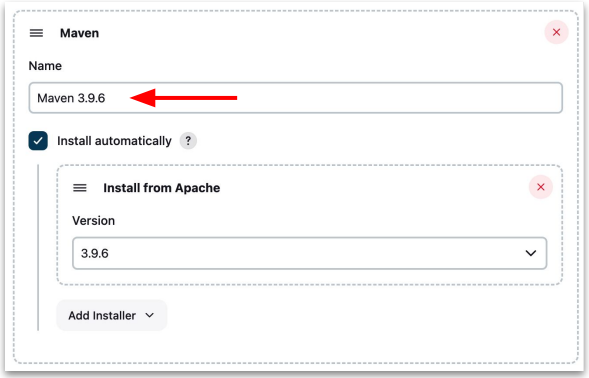
http://<jenkins-domain>:<port>/env-vars.html

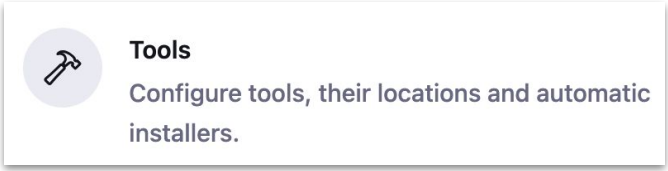
Usage in build step:

```
echo "Current build number: $BUILD_NUMBER"
```

Configuring Tools

Manage Jenkins > Tools

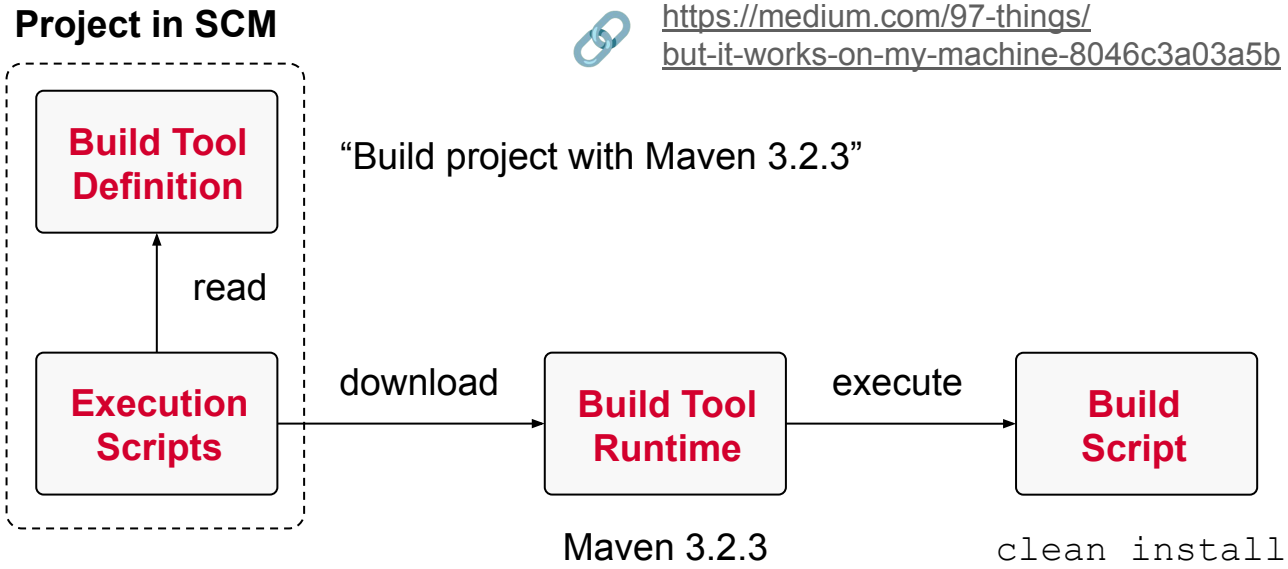




Auto-install tool and make it available on Jenkins nodes

Using a Build Tool Wrapper

Project-specific definition of build tool runtime

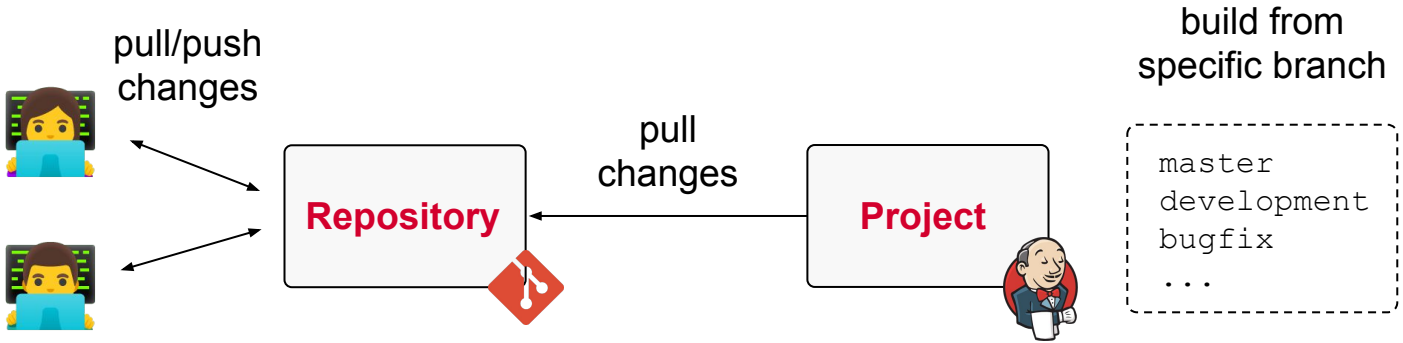


EXERCISE

Configuring Build Triggers and Steps for a Job

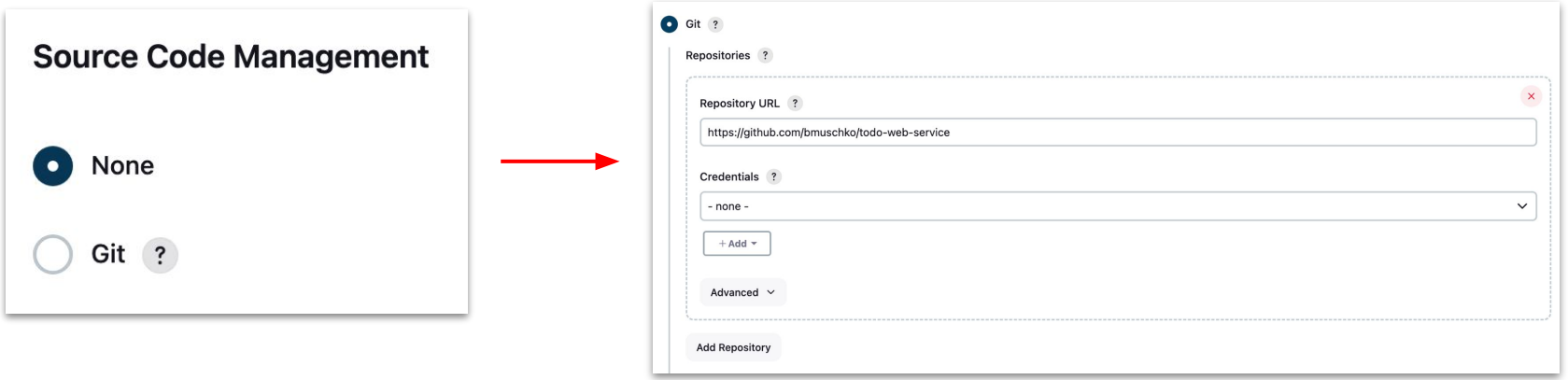
What's the Purpose of a SCM?

Central "source of truth" for source code & configuration



Supported SCM Products

Configure > Source Code Management



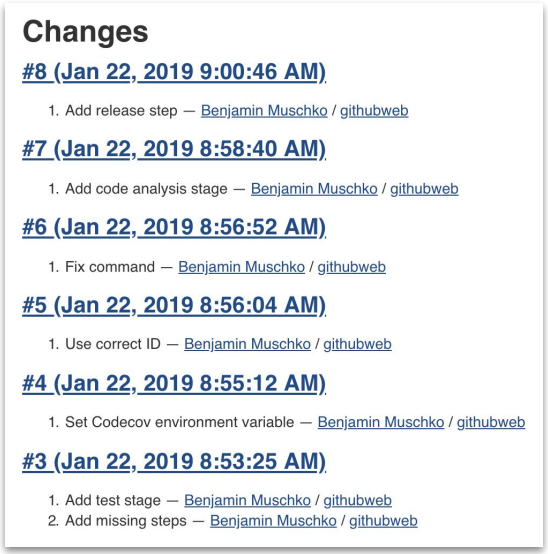
Additional Cloud-based SCMs available e.g. GitHub, BitBucket

Viewing the Changelog

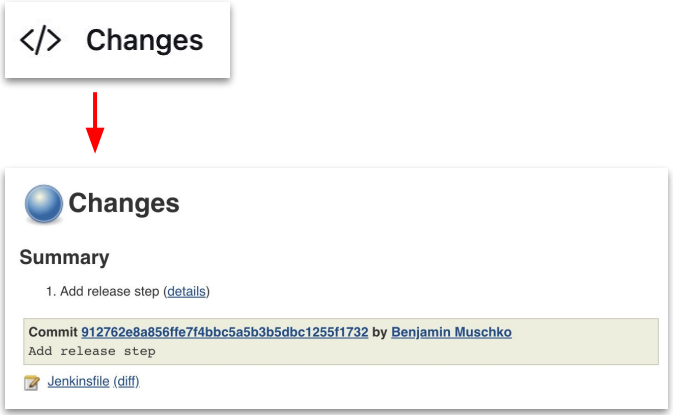
“Show me the commits pulled from SCM”

From Project, click...

</> Changes



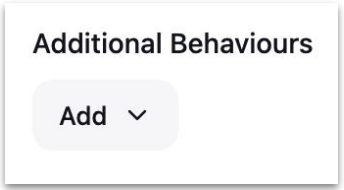
From specific Build, click...

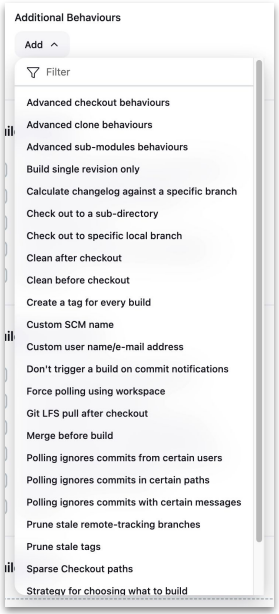


Incremental & Clean Check Out

Trade off: Checkout speed vs. clean workspace

From SCM configuration, click...

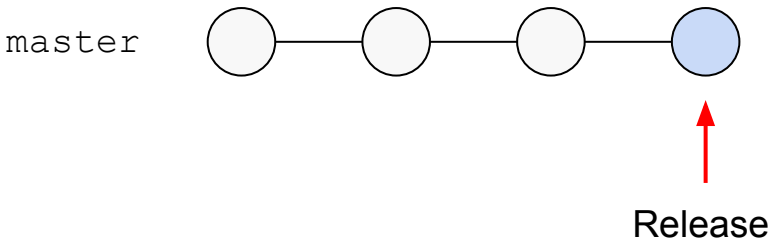




Branch and Merge Strategies

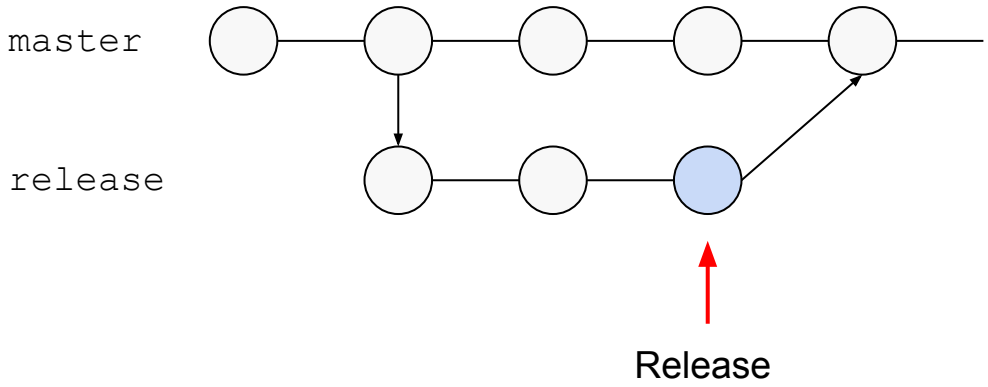
Pick the most suitable for your team, workflow & culture

Only commit to master branch, release from master



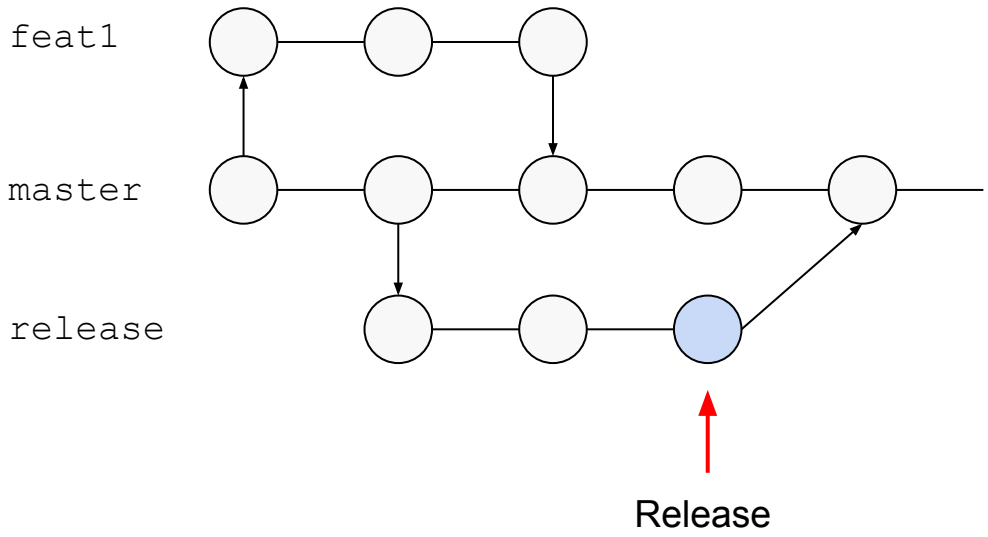
Release Branching

Create branch late in development cycle



Feature Branching

New features are developed on a dedicated branch



Branching Challenges

Branching comes at a cost...

- Avoid long-running branches
- Merge back to mainline early and often
- Employ CI validation for every branch

EXERCISE

Configuring a GitHub SCM

Q & A

BREAK

Testing, Notifications and Artifacts

Definition, Usage and Options

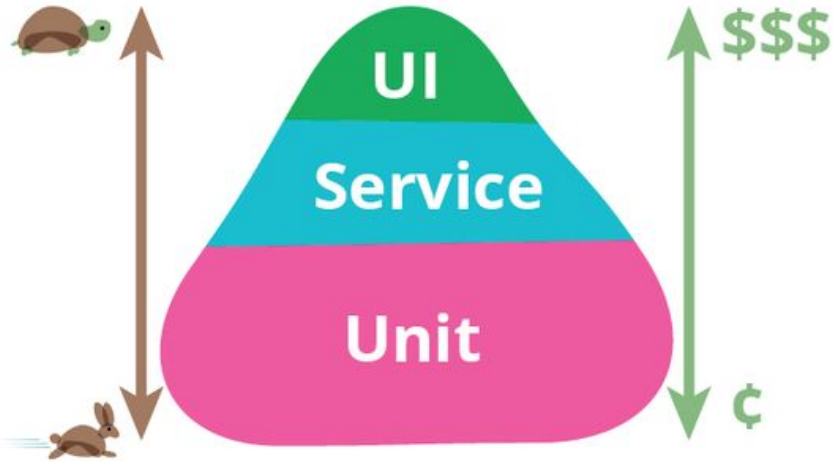
Importance of Testing

Deliver product with acceptable quality

- Functional requirements have been met
- Lower future maintenance cost
- Avoid bugs in production as much as possible
- Foundational for CI/CD

Different Types of Tests

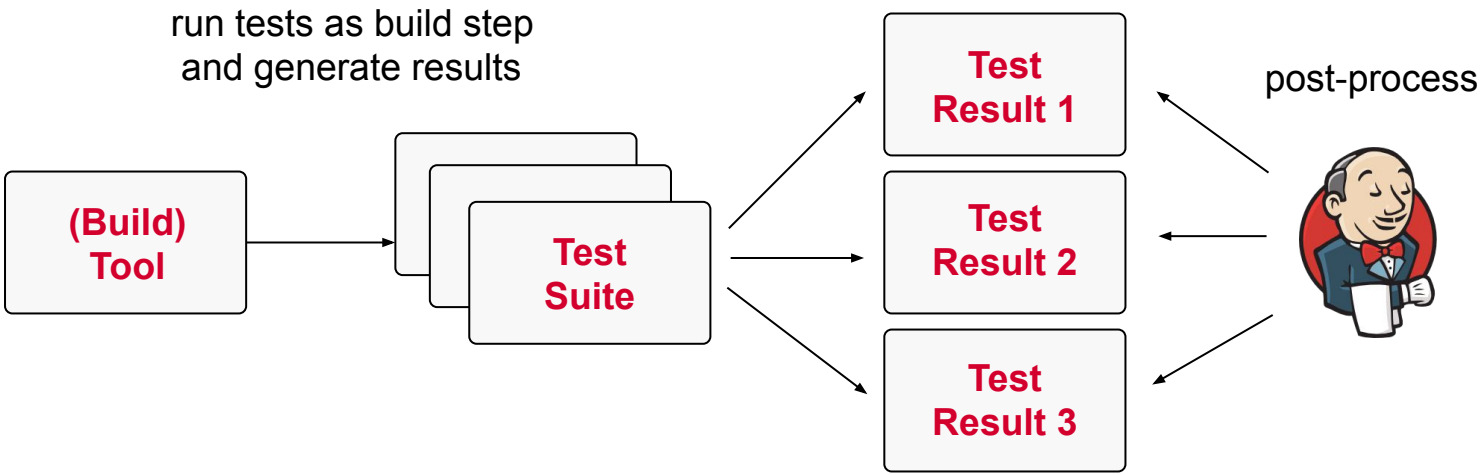
Execution speed vs. cost of maintenance



Source: https://martinfowler.com/bliki/TestPyramid.html

Generating Test Results

Tools run generation, Jenkins just post-processes results



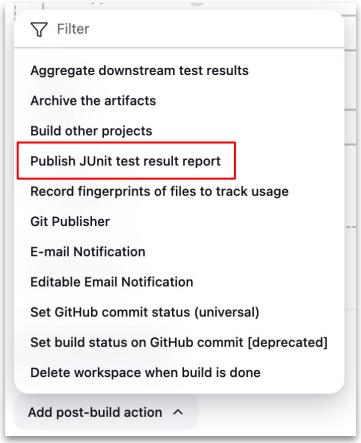
Post-Processing Options

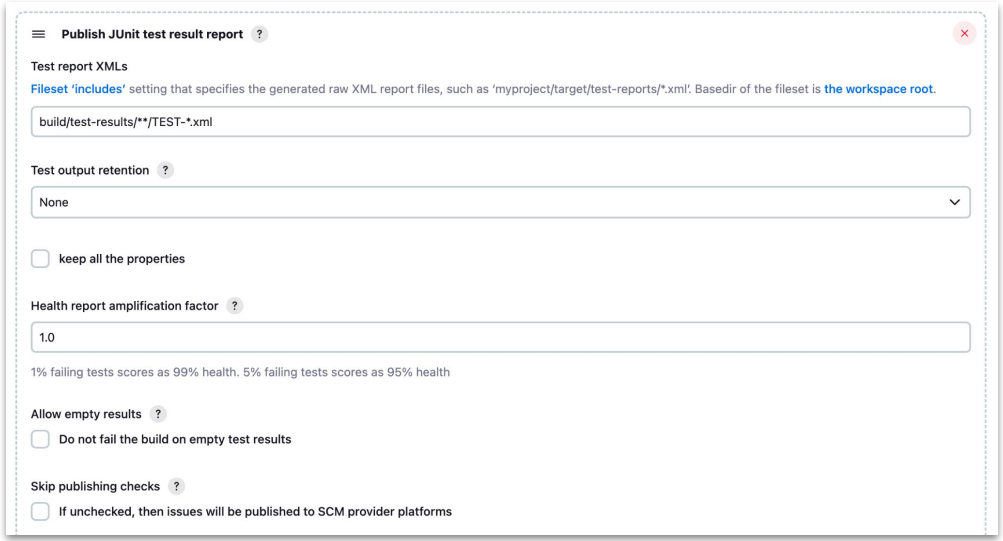
Processed & visualized on Jenkins or external platform

- Results produced by JUnit-compatible test frameworks
- Any kind of arbitrary HTML report
- Support for other results via third party-plugins
- Sending test results to external platform e.g. Coveralls, SonarQube

Processing JUnit Test Results

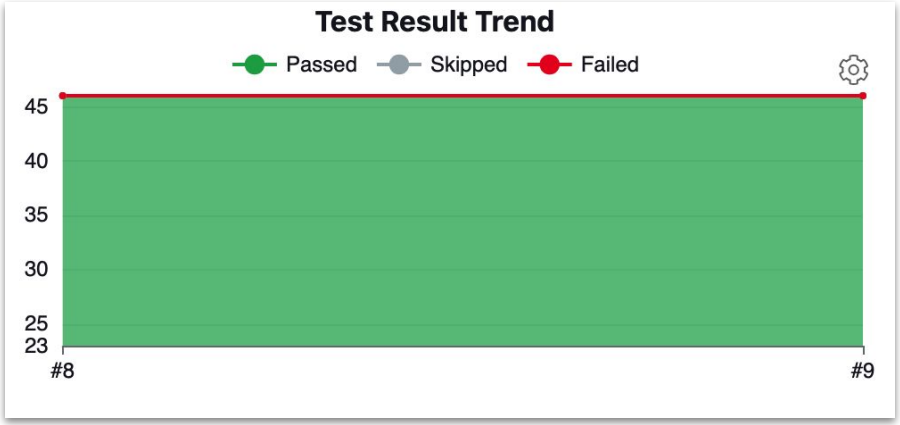
Configure > Add post-build action > *Publish JUnit...*

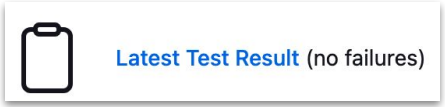


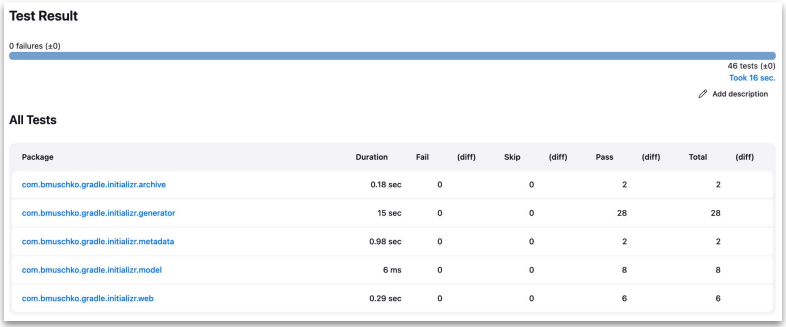


Test Trending & Reporting

“Give me a historical and detailed test overview”



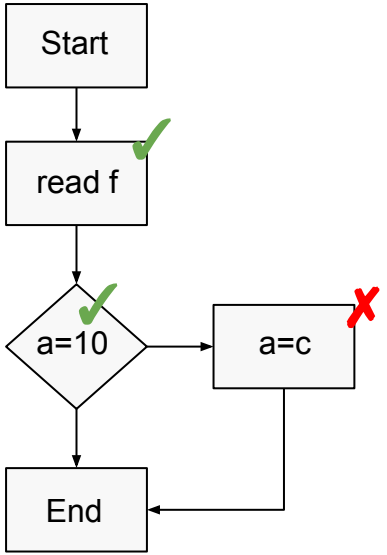




Code Coverage Metrics

“Which portions of my code have been covered by tests?”

- Statement/line coverage
- Branch coverage
- Condition coverage

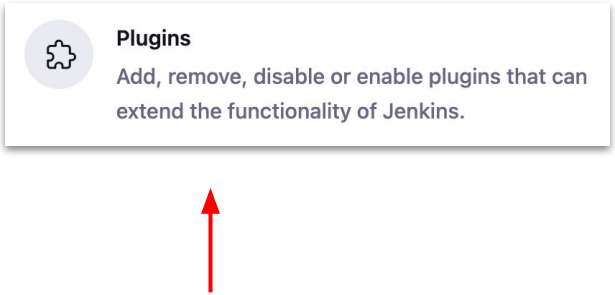


Code Coverage Tools

Tooling choice depends on your programming language

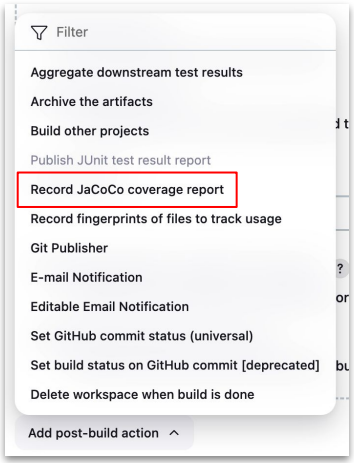
- JaCoCo
- Cobertura
- OpenClover

Support for some tools can be installed as Jenkins plugin



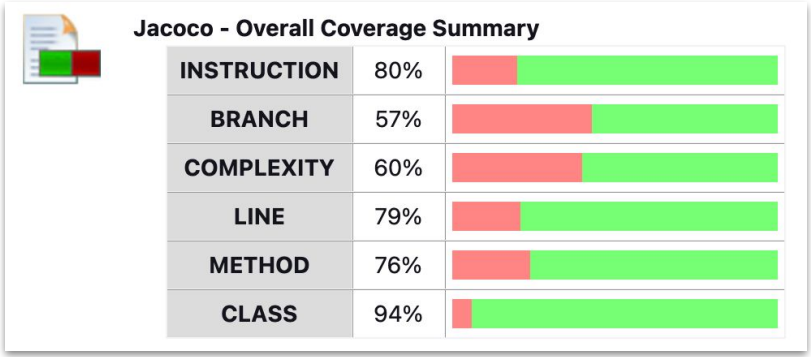
Processing JaCoCo Metrics

Configure > Add post-build action > *Record JaCoCo...*



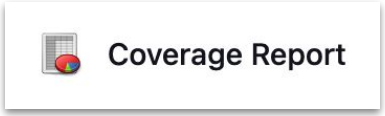
Coverage Trending & Reporting

“Give me a historical and detailed coverage overview”

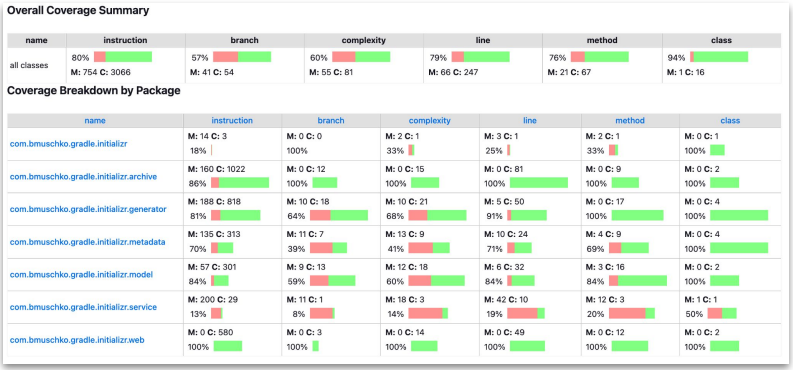


Jacoco - Overall Coverage Summary

INSTRUCTION	80%
BRANCH	57%
COMPLEXITY	60%
LINE	79%
METHOD	76%
CLASS	94%



Coverage Report



Overall Coverage Summary

name	instruction	branch	complexity	line	method	class
all classes	80% M: 754 C: 3066	57% M: 41 C: 54	60% M: 55 C: 81	79% M: 66 C: 247	76% M: 21 C: 67	94% M: 1 C: 16

Coverage Breakdown by Package

name	instruction	branch	complexity	line	method	class
com.bmuschko.gradle.initializr	M: 14 C: 3 18%	M: 0 C: 0 100%	M: 2 C: 1 33%	M: 3 C: 1 25%	M: 2 C: 1 33%	M: 0 C: 1 100%
com.bmuschko.gradle.initializr.archive	M: 160 C: 1022 86%	M: 0 C: 12 100%	M: 0 C: 15 100%	M: 0 C: 81 100%	M: 0 C: 9 100%	M: 0 C: 2 100%
com.bmuschko.gradle.initializr.generator	M: 188 C: 818 81%	M: 10 C: 18 64%	M: 10 C: 21 68%	M: 5 C: 50 91%	M: 0 C: 17 100%	M: 0 C: 4 100%
com.bmuschko.gradle.initializr.metadata	M: 135 C: 313 70%	M: 11 C: 7 39%	M: 13 C: 9 41%	M: 10 C: 24 71%	M: 4 C: 9 69%	M: 0 C: 4 100%
com.bmuschko.gradle.initializr.model	M: 57 C: 301 84%	M: 9 C: 13 59%	M: 12 C: 18 60%	M: 6 C: 32 84%	M: 3 C: 16 84%	M: 0 C: 2 100%
com.bmuschko.gradle.initializr.service	M: 200 C: 29 13%	M: 11 C: 1 8%	M: 18 C: 3 14%	M: 42 C: 10 19%	M: 12 C: 3 20%	M: 1 C: 1 50%
com.bmuschko.gradle.initializr.web	M: 0 C: 580 100%	M: 0 C: 3 100%	M: 0 C: 14 100%	M: 0 C: 49 100%	M: 0 C: 12 100%	M: 0 C: 2 100%

EXERCISE

Displaying JUnit and
JaCoCo Test Results

What's a Notification?

Inform team members about events in build

- Alert developers who broke the build to fix it
- Minimize the number of notifications
- Required for successful adoption of CI/CD
- Use channel most suitable to team

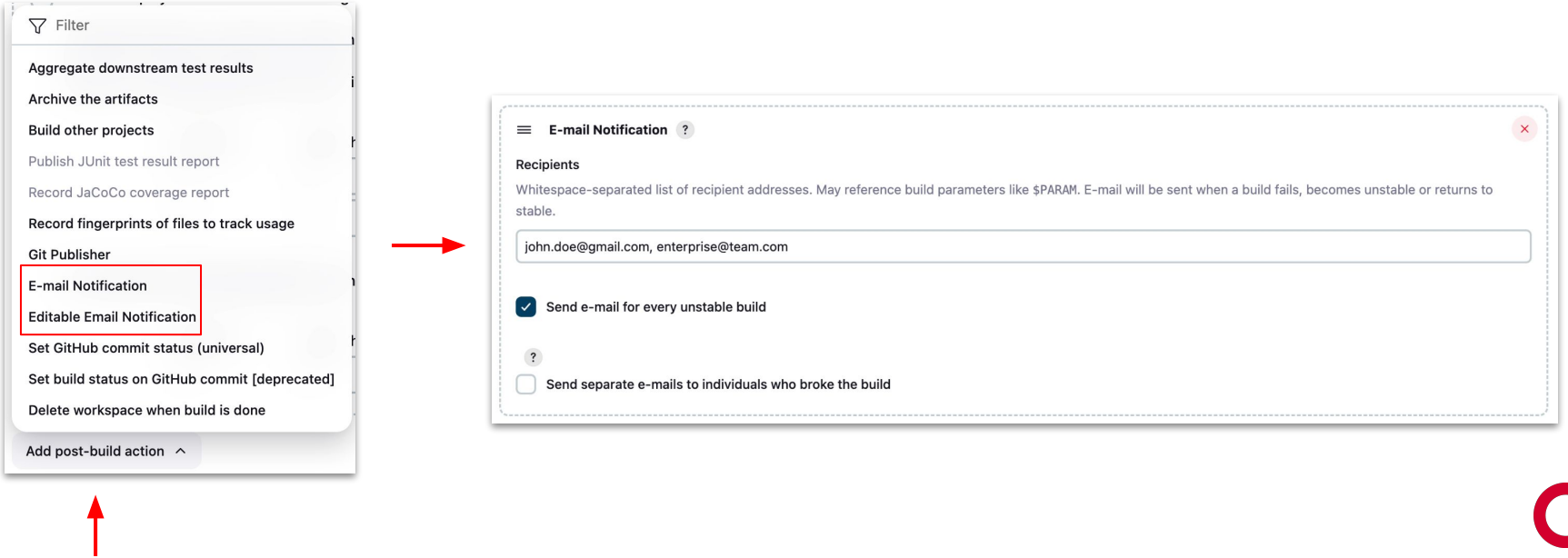
Types of Notifications

Standard is email, others can be added via plugins

- Email with or without customized message
- Team collaboration tools like Slack and HipChat
- Messengers like Google Chat, Jabber or SMS

Configuring Email Notification

Configure > Add post-build action > E-mail Notification

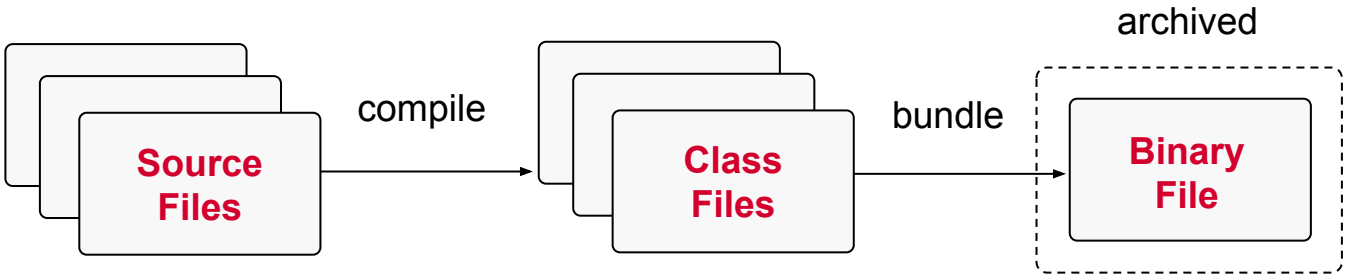


EXERCISE

Notifying the Team
Upon a Broken Build

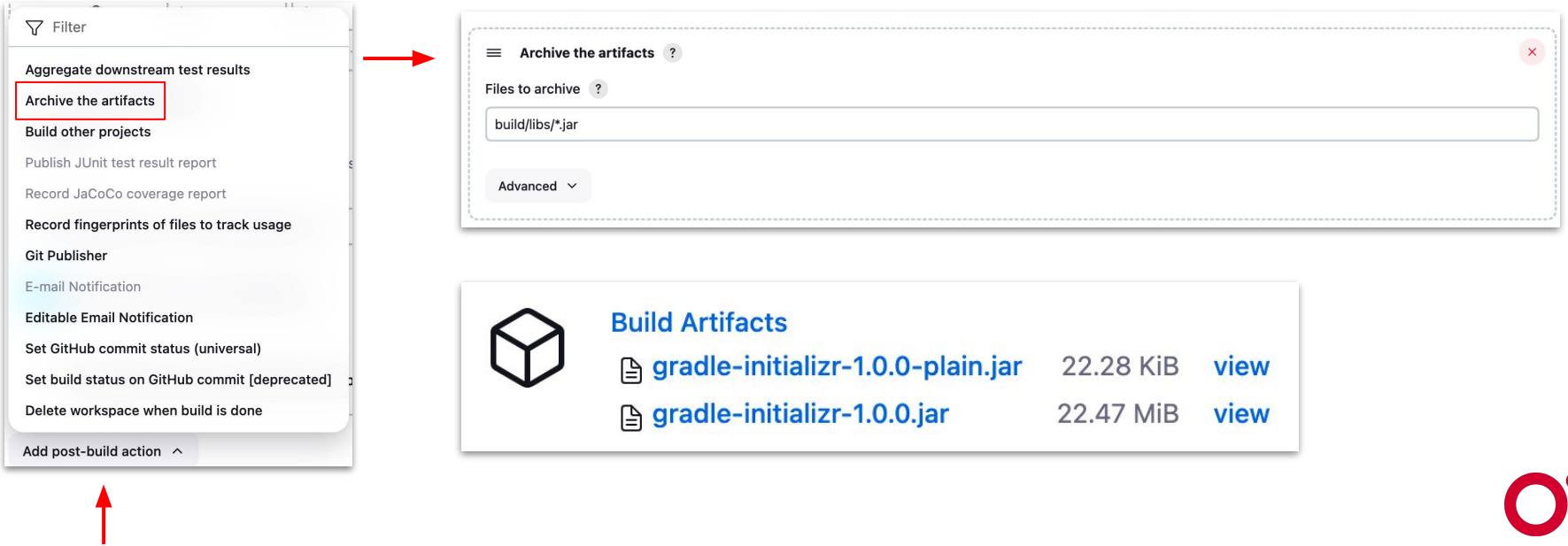
What's are Artifacts?

Immutable files produced and archived by build



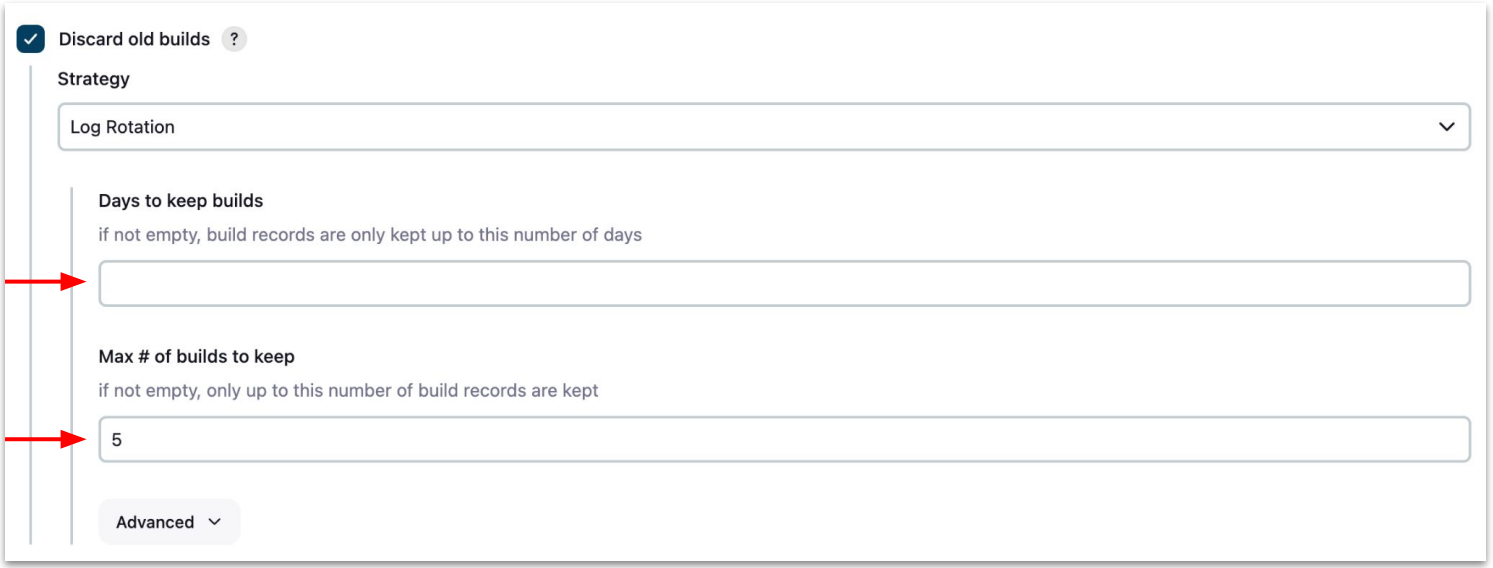
Archiving an Artifact

Configure > Add post-build action > Archive the artifacts



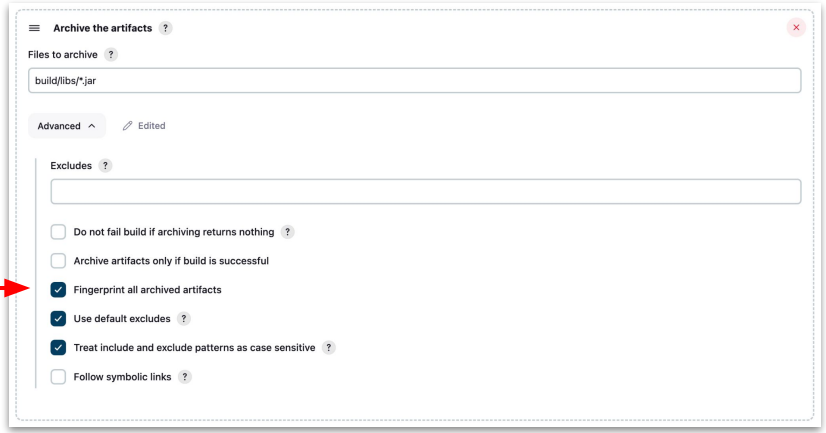
Artifact Retention Policy

“I want to limit the number of archived artifacts”

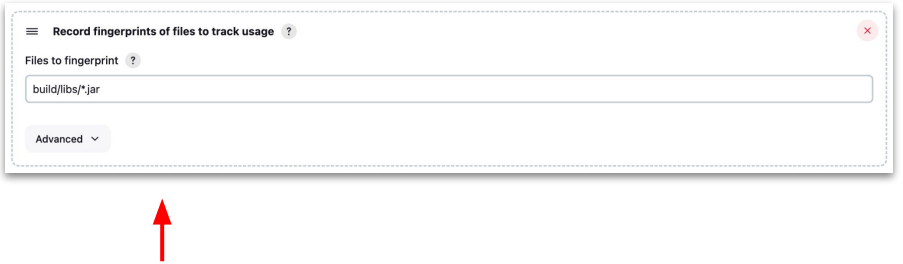


Fingerprinting Artifacts

Track a particular artifact back to the producing build



or

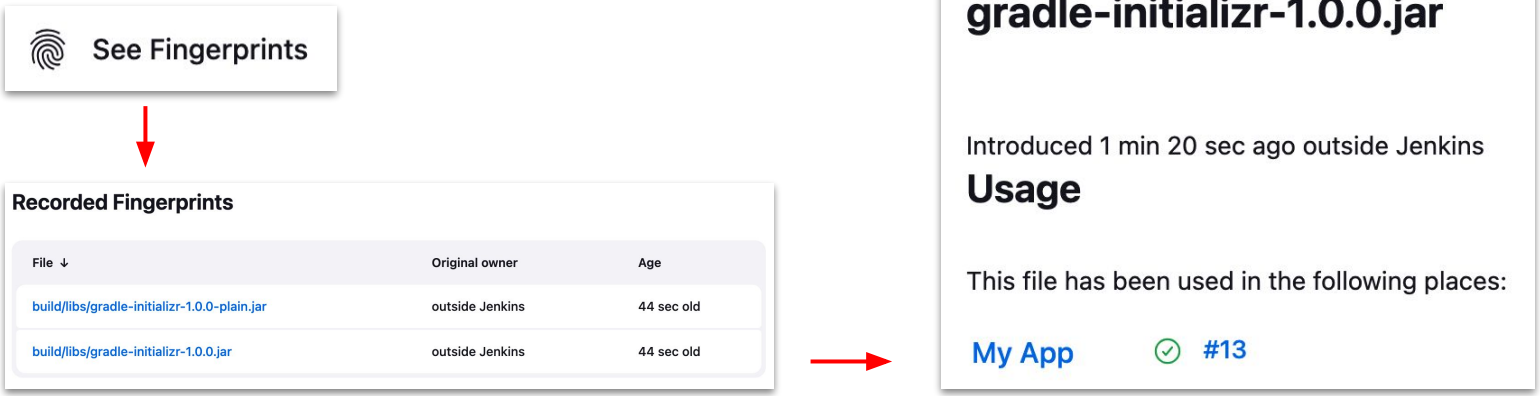


Important for interdependent projects setups

Viewing Artifact Usage

“Show me the originating build and the artifact’s MD5 hash”

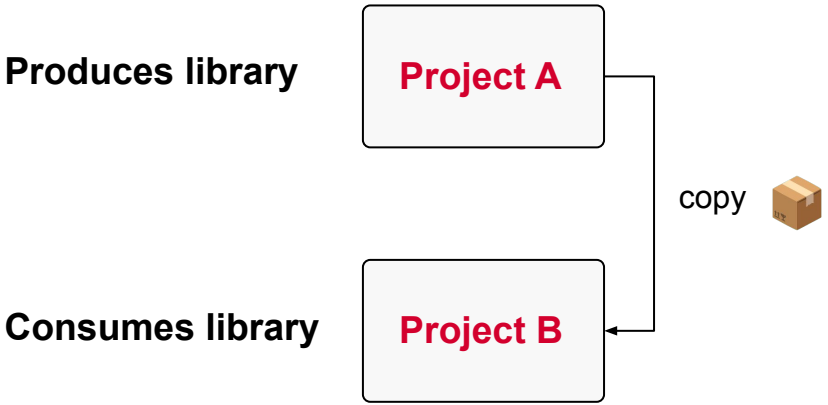
From specific Build, click...



MD5: b25c3d16fe8b44525bca8f6e9867f997

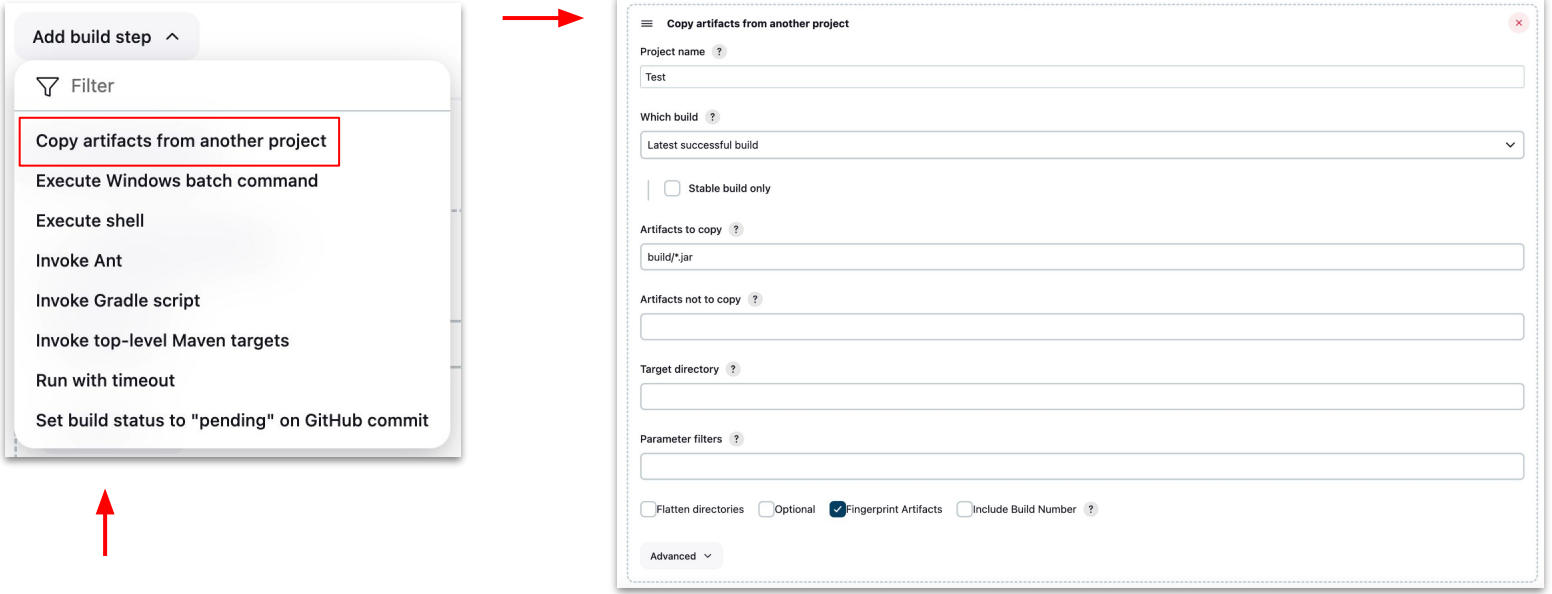
Cross-Project Artifacts

Make the output of a build available to another project



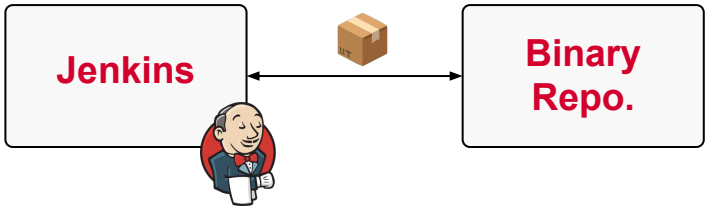
Copying Artifacts

Functionality only available through “Copy Artifact” plugin



Using Binary Repositories

Products offer features more suitable to enterprise projects



JFrog ARTIFACTORY

sonatype

EXERCISE

Storing and
Fingerprinting Artifacts

Q & A

Jenkins Administration

Security, REST API, Distributed Builds

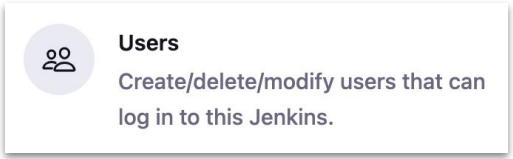
Authentication vs. Authorization

Access Control is primary mechanism for securing Jenkins

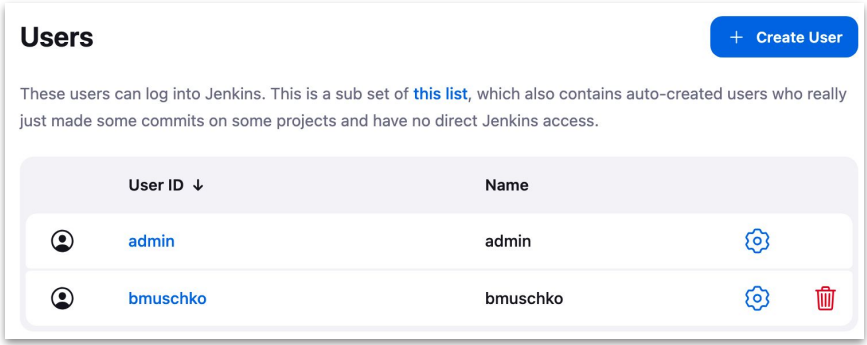
- **Authentication:** “Validating that users are who they claim to be”
- **Authorization:** “Process of giving the user permission to access a specific resource or function”

Managing Users

Manage Jenkins > Users

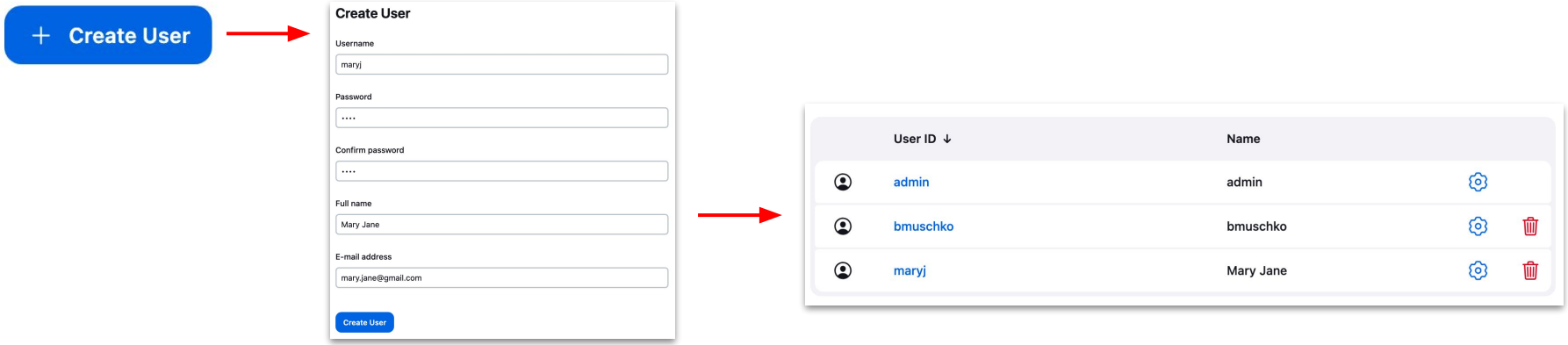


You can’t delete logged-in user



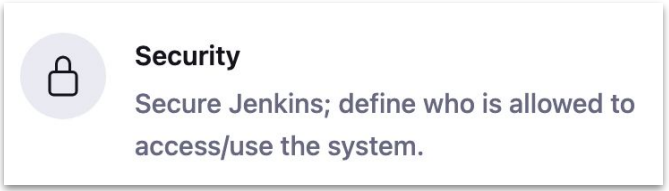
Creating Users

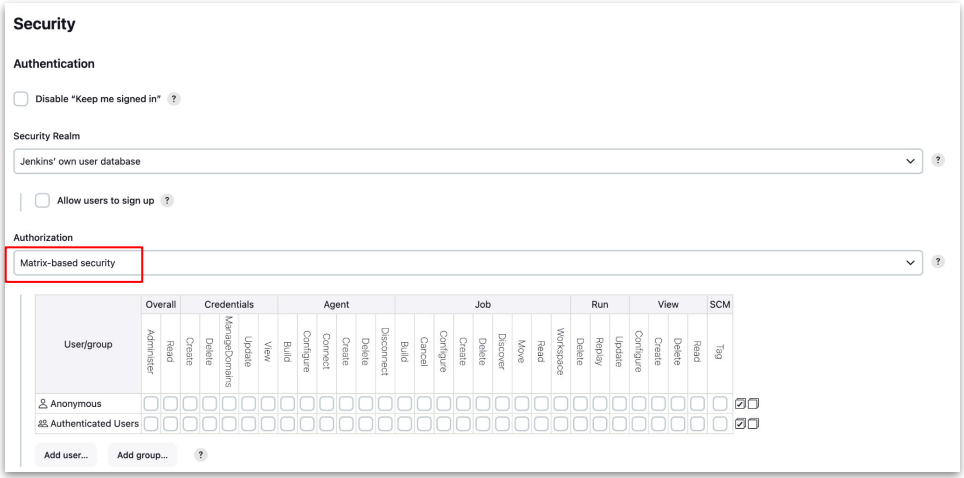
Manage Jenkins > Manage Users > Create User



Enabling Matrix Security

Manage Jenkins > Security

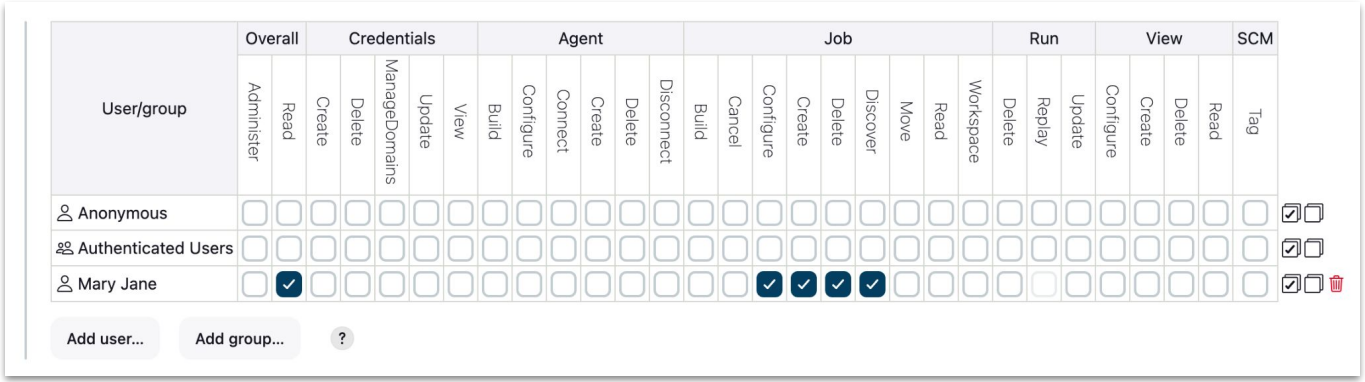




Access Control is the primary mechanism for securing a Jenkins environment

Configuring Matrix Security

Pick appropriate permissions for users or groups



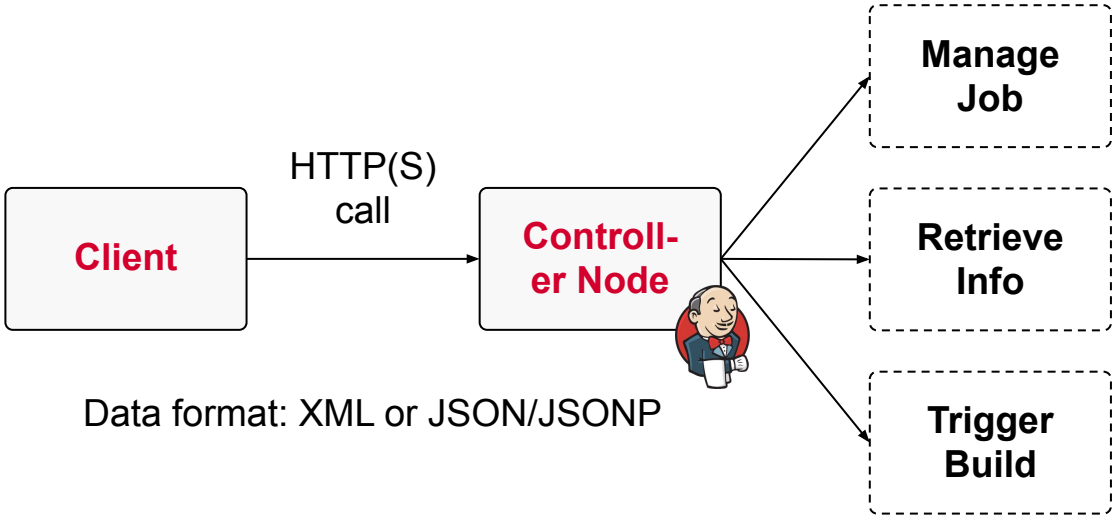
Group support is only usable when integrating Jenkins with LDAP or Active Directory

EXERCISE

Creating a User and
Setting Permissions

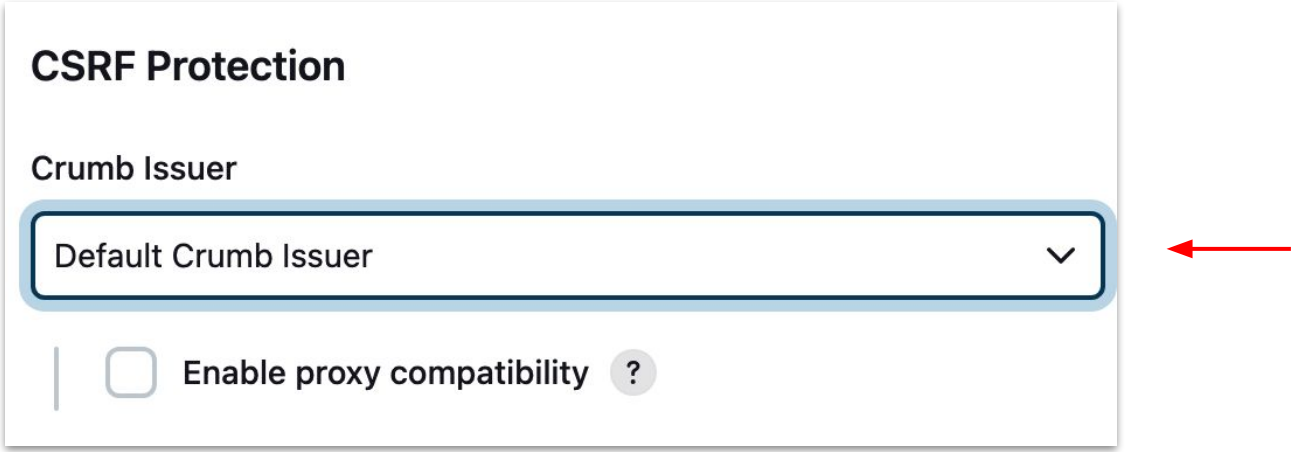
Jenkins REST API in a Nutshell

Remote access for triggering operations via HTTP(S) call



Prevent CSRF Exploits

Manage Jenkins > Security



Enabled by default

Generating a Crumb

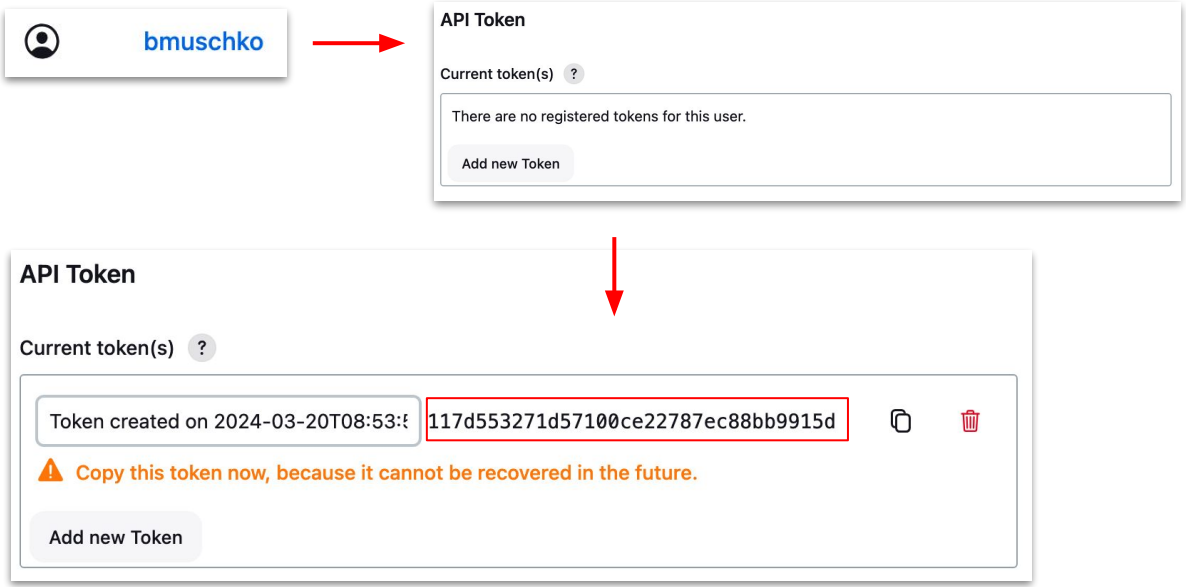
Requires sending credentials for user

```
$ curl -u "bmuschko:password"
'http://localhost:8080/crumbIssuer/api/xml?xp
ath=concat(//crumbRequestField,":",//crumb)'

Jenkins-Crumb:3e1a6ffc32e811b46299e28c40f16e50
```

Crumb needs to be sent with every request

Creating an API Token

Manage Jenkins > Users > Select user



Constructing an API call

Endpoint + Crumb + Username & API token

```
$ curl -X POST -H
"Jenkins-Crumb:3e1a6ffc32e811b46299e28c40f16e50
"
http://bmuschko:112f8ff3a62211c7db297970de214d1
c51@localhost:8080/job/test/build
```

Crumb

Endpoint

API token

Sending Parameters

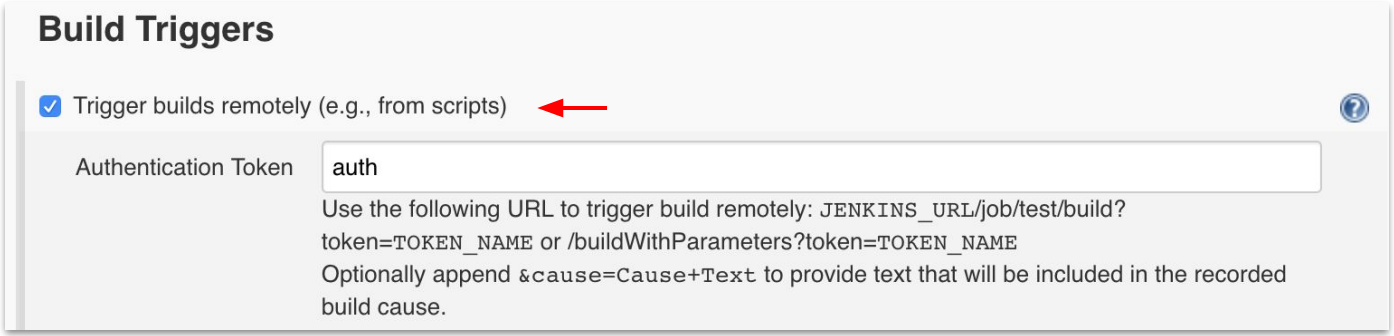
Provide as form data or JSON with the API call

```
$ curl -X POST -H
"Jenkins-Crumb:3e1a6ffc32e811b46299e28c40f16e50
"
http://bmuschko:112ba66030d39aad78178f5f0d2add4
88d@localhost:8080/job/test/description
--data-urlencode description="My awesome job"
```

Data

Optional: Auth. Token for Job

Dashboard > Job > Build Triggers



Adds an additional layer of security

Discovering API Operations

“Easy! Ask the Jenkins server directly.”

http://<jenkins-domain>:<port>/**job**/<name>/**api**/

Examples:

Disable Job: http://localhost:8080/job/test/disable

Fetch configuration: http://localhost:8080/job/test/config.xml

API Client Implementations

Suitable for calling the API from a program

Currently available as wrappers in Python, Ruby, Java

Usage example for Java client:

```
JenkinsServer jenkins = new JenkinsServer(new↵
URI("http://localhost:8080/jenkins"), "admin", "password");
Map<String, Job> jobs = jenkins.getJobs();
JobWithDetails job = jobs.get( "My Job").details();
```

https://github.com/jenkinsci/java-client-api

The Jenkins CLI

A ready-to-use standalone Java-based application

Download:

http://<jenkins-domain>:<port>/jnlpJars/jenkins-cli.jar

Usage:

```
$ java -jar jenkins-cli.jar [-s JENKINS_URL]↵
command ...
```

Example CLI Usage

Authentication via SSH or user credentials

```
$ java -jar jenkins-cli.jar -s http://localhost:8080
-auth bmuschko:1116814fd4a6378d0fcadc01c11126b4e6
version
2.176.1
```

Username:API token

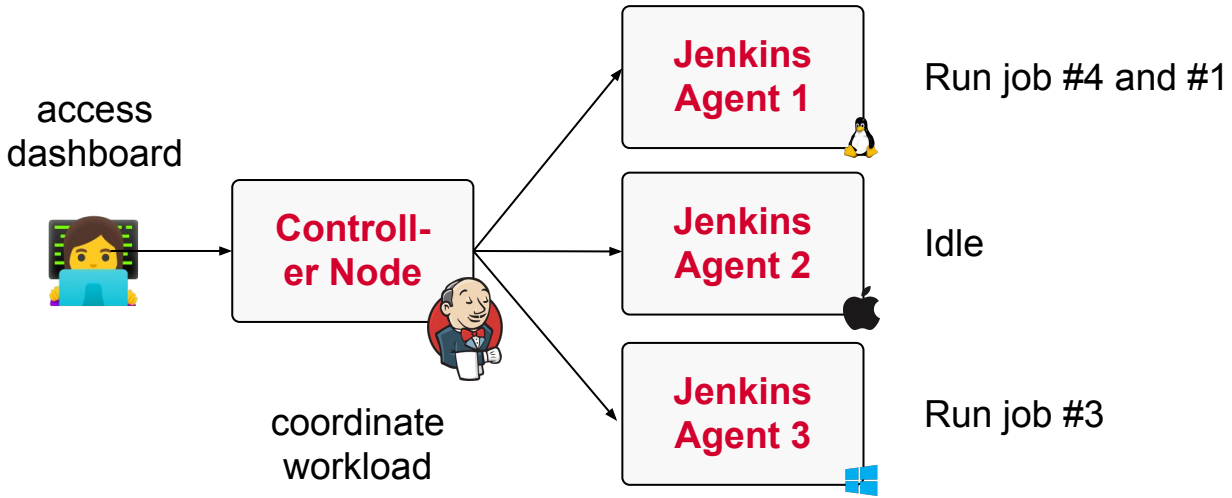
Full reference: http://<jenkins-domain>:<port>/cli/

EXERCISE

Using the REST API for
Common Operations

What's a Distributed Build?

Scaling workload or running job in different environments



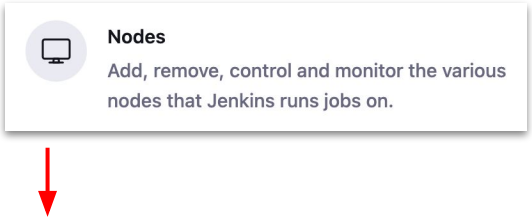
Reasons for Adoption

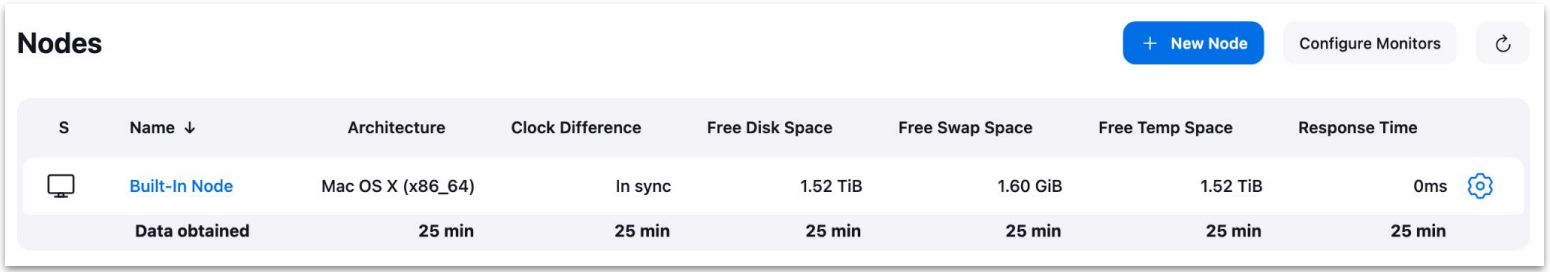
Many benefits for enterprise, heterogeneous projects

- Intelligent job dispatching
- Automatic installation of tools on agents
- Agent health metrics monitoring
- Manual or automatic installation of a agent

Managing Nodes

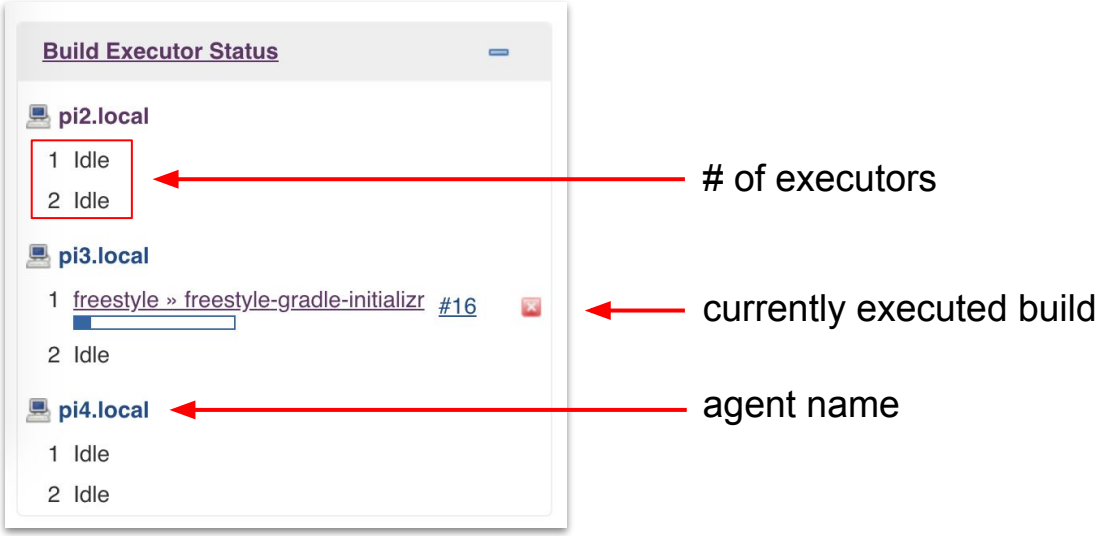
Manage Jenkins > Nodes





Executors Dashboard Overview

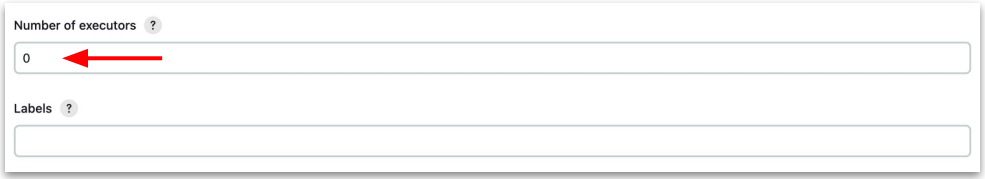
Maximum builds per executor & what is currently running?



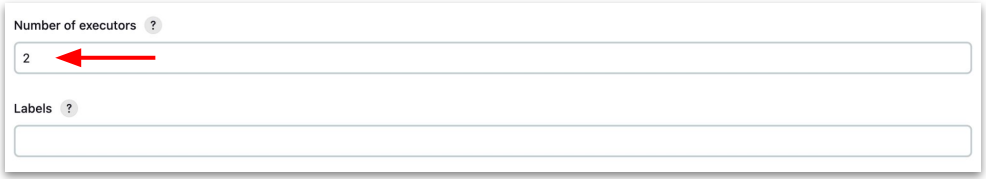
Executor Configuration

Controller node should not execute build workload

Controller configuration

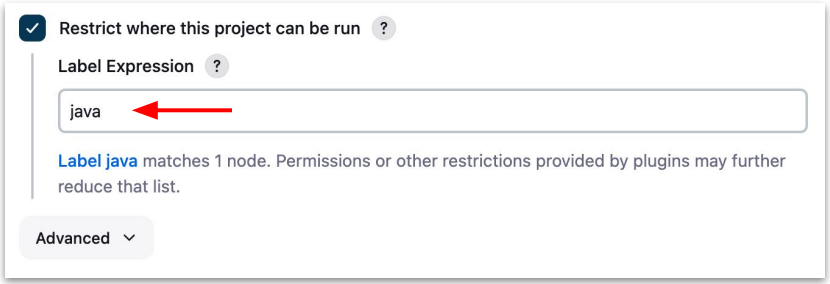


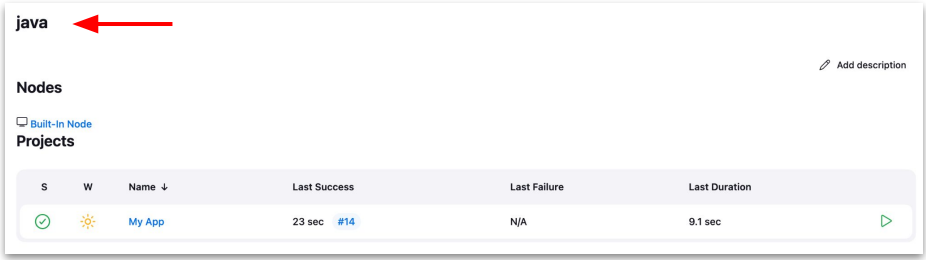
Agent configuration



Job Execution by Label

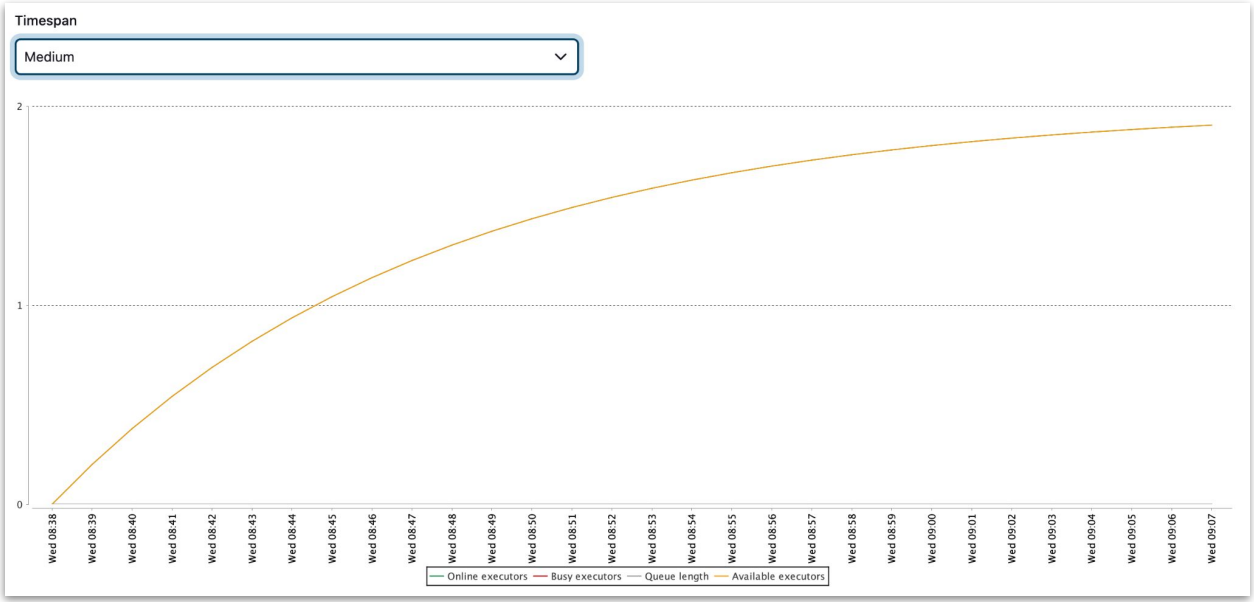
Various plugins provide configuration option(s)





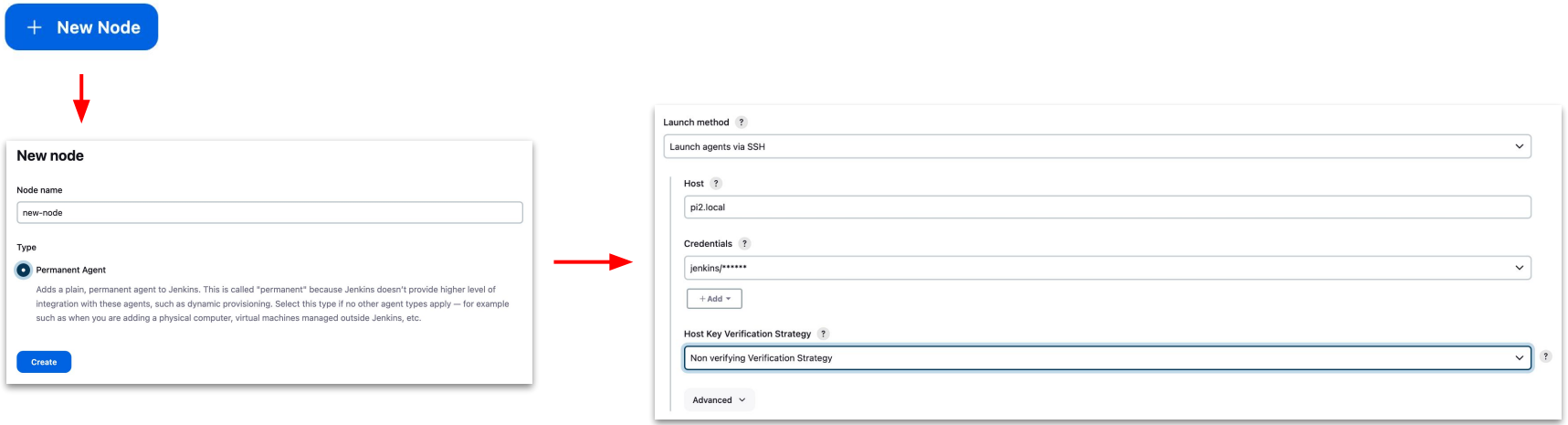
Node Statistics

Keeping track of executors uptime, availability etc.



Adding New Nodes

SSH is the most common way to launch an agent



EXERCISE

Configuring and Executing Jobs in a Distributed Build

Q & A

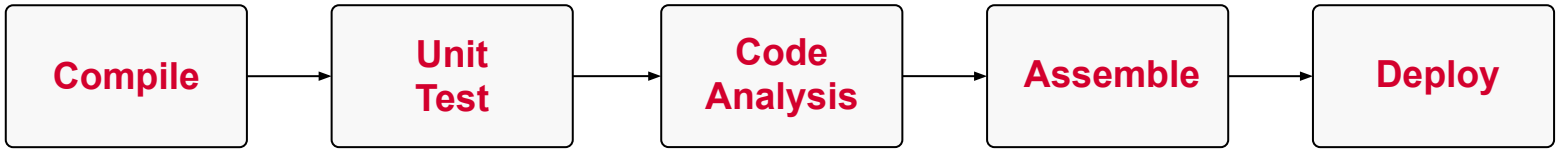
BREAK

Building Continuous Delivery (CD) Pipelines

Pipeline as Code, Pipeline Types and Syntax

Why Model a Pipeline?

Orchestrates automated build and release workflow



There's no one-size-fits-all pipeline

Pipeline Quality Gates

Stop the pipeline if expected metrics are not met

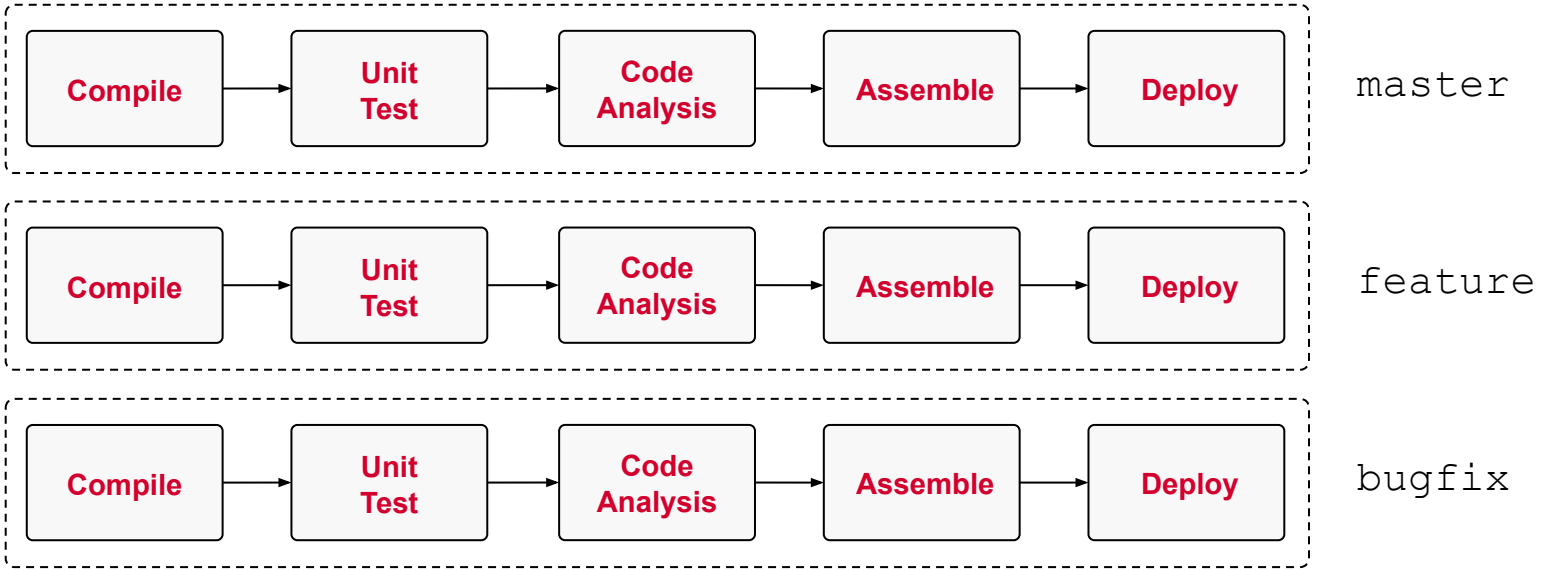
“Does the code compile without issues or warnings?” ✓

“Does the code follow code conventions?” ✗

“Do all tests pass and produce >= 70% code coverage?” ✓

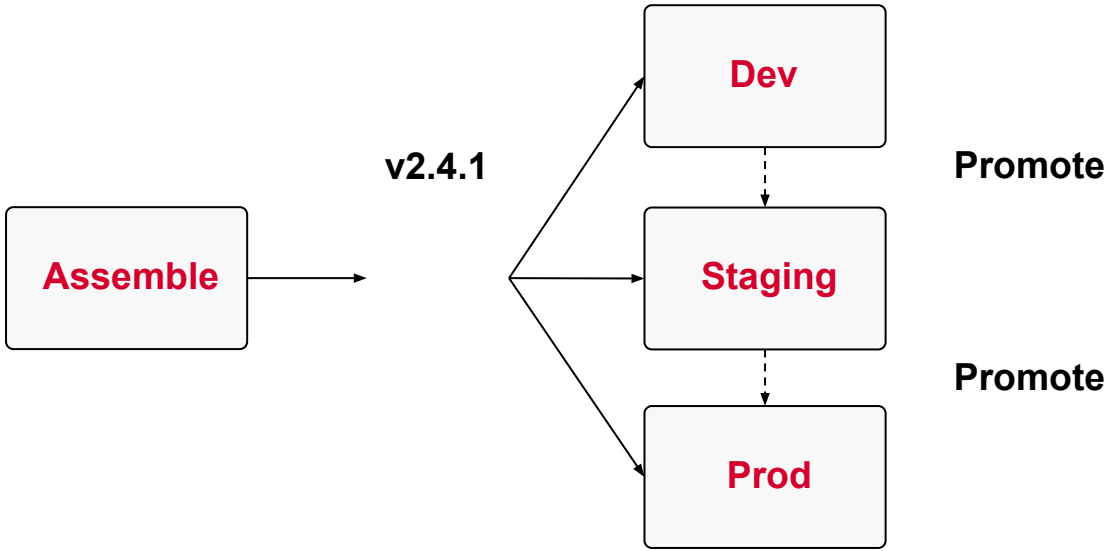
Same Pipeline for Branches

Apply stages, automation and checks consistently



Reusing an Artifact

Build once per commit and version appropriately



Promoting Artifacts

Manually approve or via automated process

- Promoted Builds Plugin is not compatible with pipelines (JENKINS-36089)

- Practical solution
  - Keep logic in build tool definition
  - Store artifact in binary repository, promote across repositories
  - Pipeline functionality offers manual steps

Monitoring the Pipeline

Identify issues early and proactively

- Build your own dashboard or use commercial product
- Find and fix bottlenecks
- Track code quality metrics over time
- Use as communication tool

Modeling a Pipeline in Jenkins

“Pipeline” - a suite of plugins for creating CI/CD pipelines

- Definition of pipeline with complex requirements
- Integration with standard Jenkins features
- Visualization of pipeline in real-time
- Integration with Docker

Pipeline Definition Language

Scripted vs. declarative syntax as Groovy DSL

- **Scripted**
  - More flexible, custom logic
  - Requires deeper knowledge about Groovy

- **Declarative**
  - Higher-level language constructs
  - Preferred syntax
  - Sometimes lacks in features

Pipeline DSL Comparison

Only scripted DSL allows imperative pipeline logic

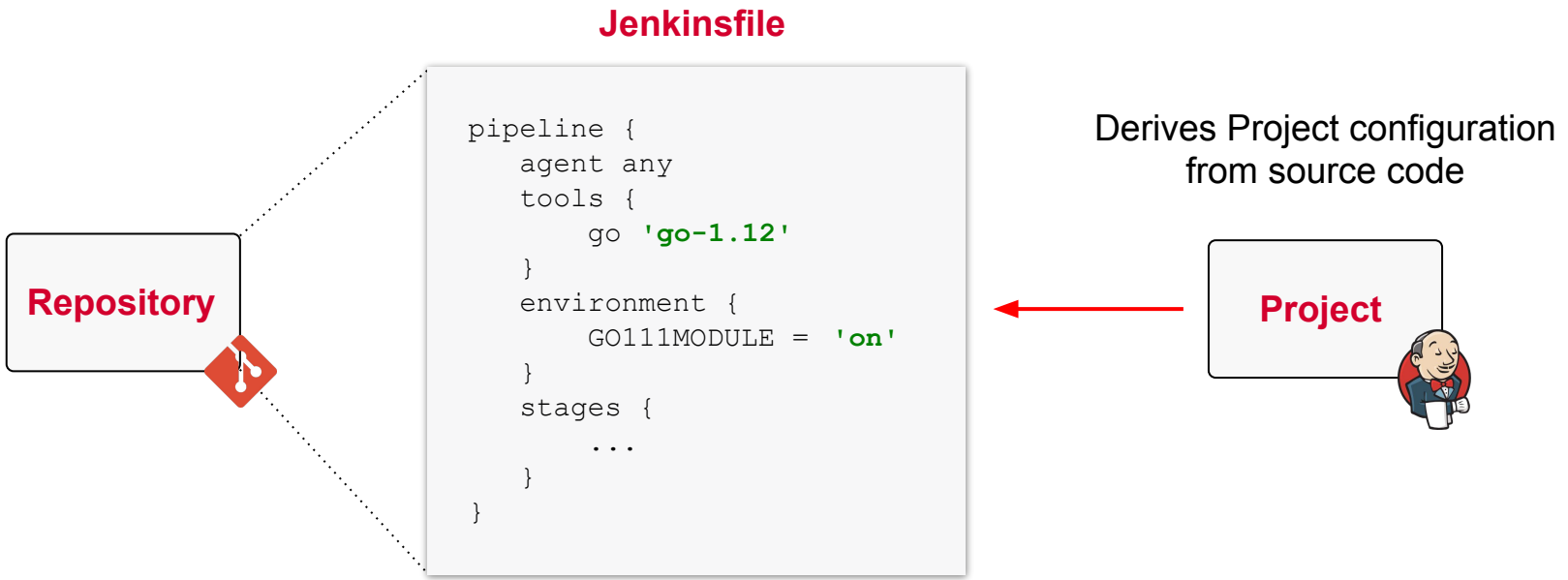
```
pipeline {
    agent any
    stages {
        stage('Message') {
            steps {
                echo 'Hello World!'
            }
        }
    }
}
```

vs.

```
node {
    stage('Message') {
        if (env.BRANCH_NAME
                == 'master') {
            echo 'Hello World!'
        } else {
            echo 'Do nothing...'
        }
    }
}
```

Infrastructure-as-Code

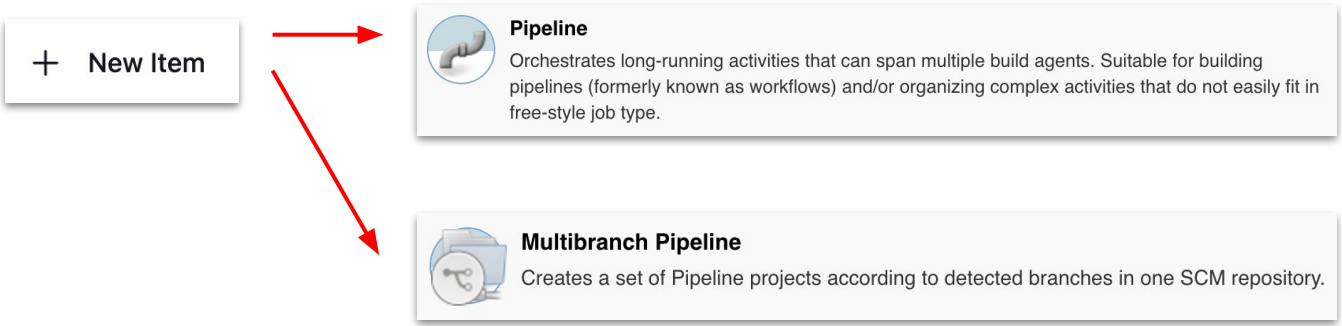
Version controlled infrastructure configuration



Creating a Pipeline Job

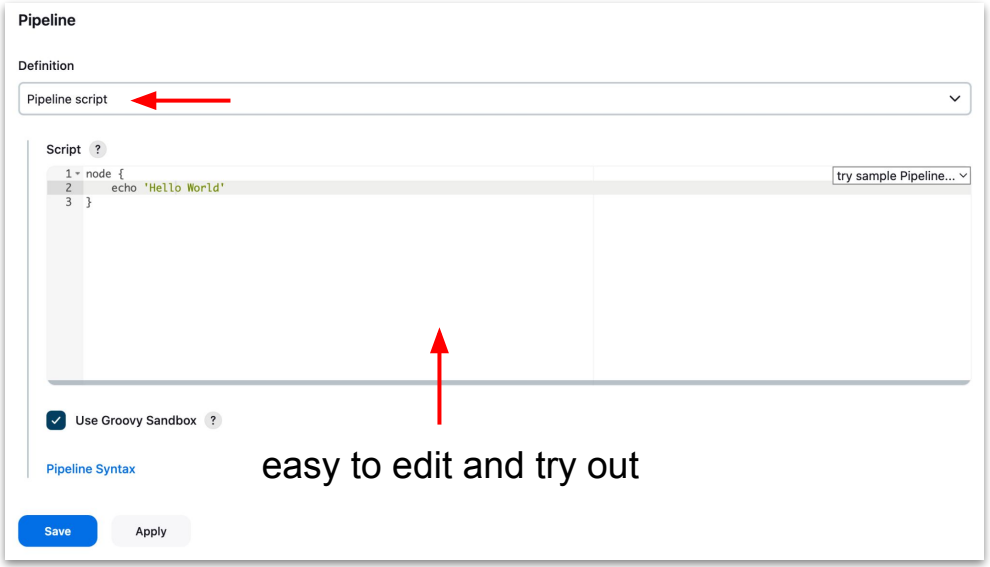
Pipeline jobs are available if plugins have been installed

From Dashboard, click...



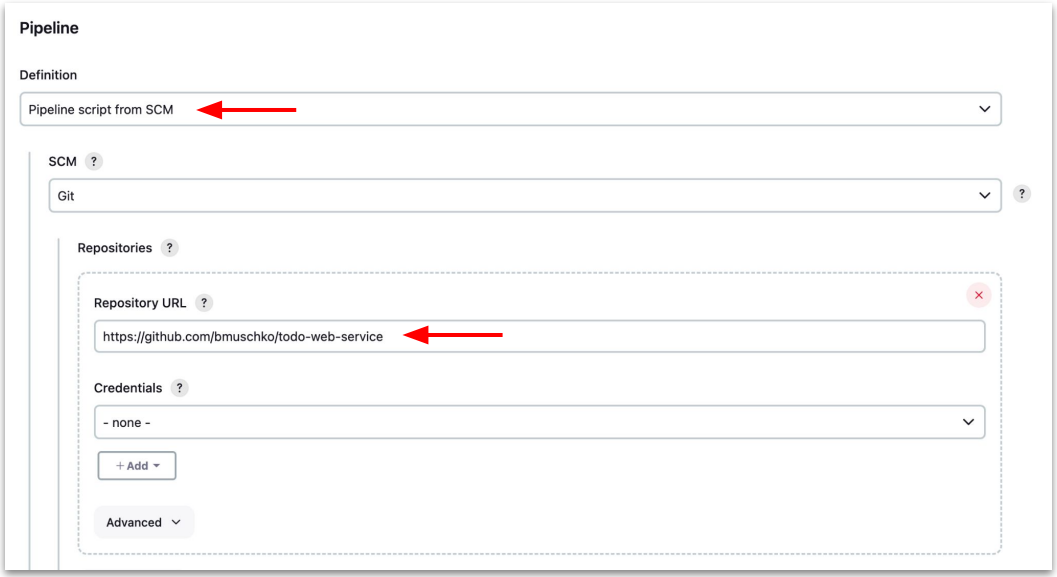
Scripted Pipeline Definition

Scripting is an option but versioned code is preferrable



Versioned Pipeline Definition

Embrace pipeline-as-code approach



Linting in IDE

“Getting the syntax right is hard, use some help!”

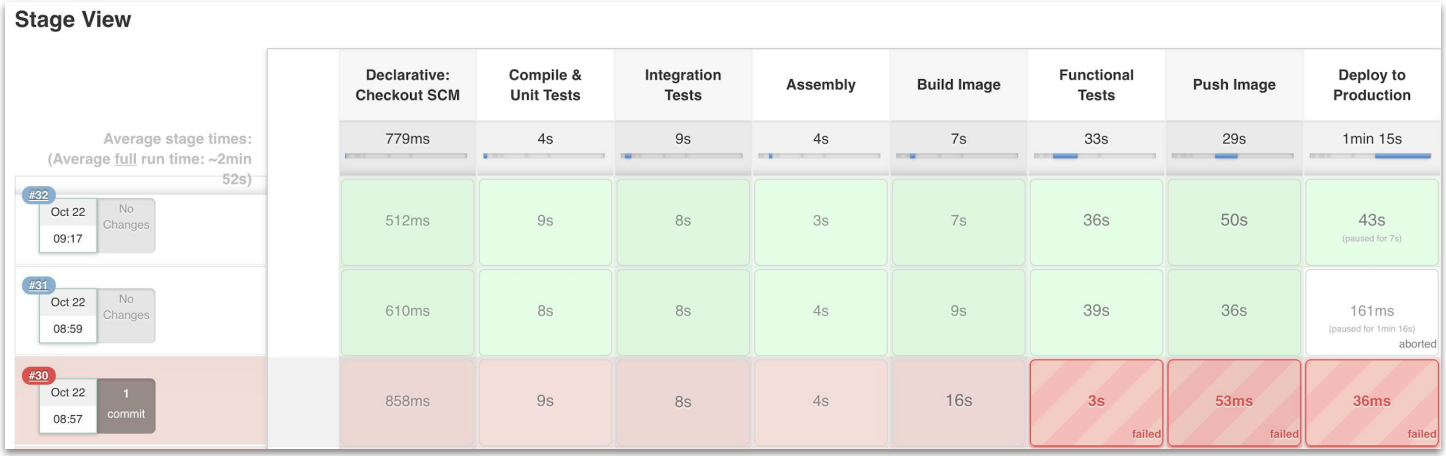
Jenkins Editor Eclipse plugin

Jenkins Pipeline Linter Connector

Jenkinsfile language support

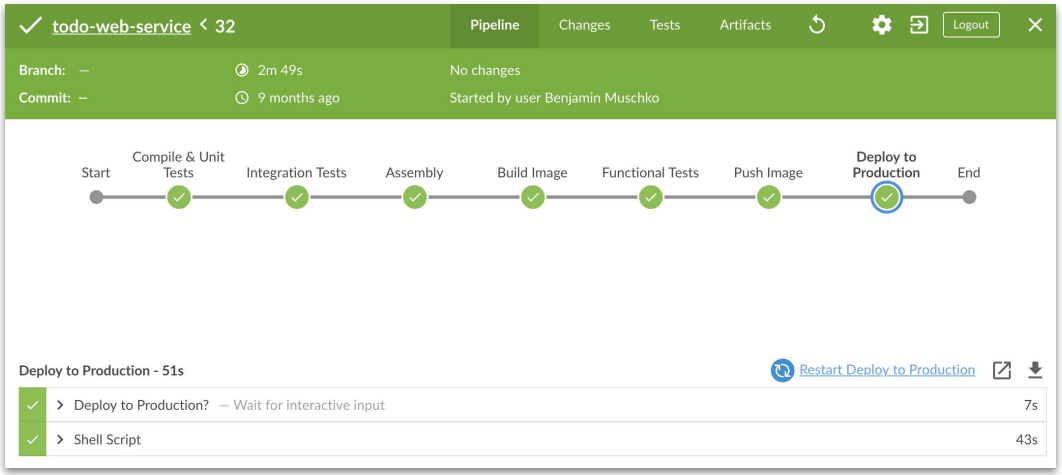
Standard Pipeline Visualization

Only renders pipelines linearly



Using the BlueOcean Plugin

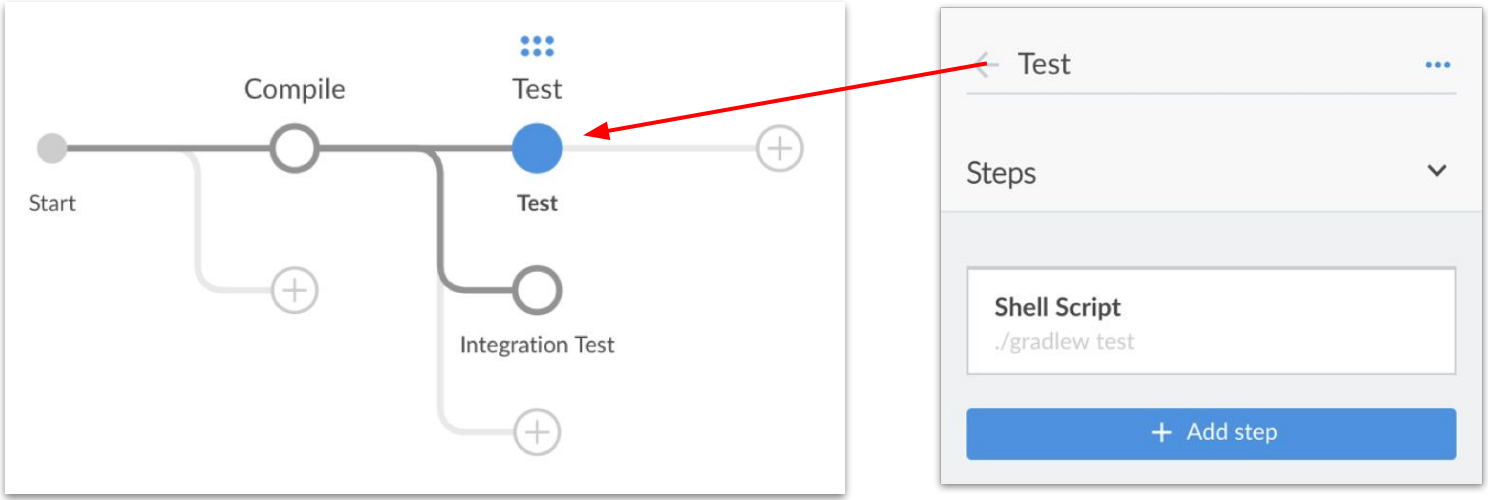
Next generation pipeline visualization



Plugin is not part of standard installation

BlueOcean Pipeline Builder

Good initial step for generating a pipeline definition



EXERCISE

Creating a Pipeline Job

The Declarative DSL

Your number one choice for modeling pipelines

- No scripting allowed
- Very readable and maintainable
- Doesn’t meet all requirements (yet)

https://jenkins.io/doc/book/pipeline/syntax/#declarative-pipeline

The Pipeline Section

Top-level configuration element for a Jenkins pipeline

```
pipeline {
    ...
}
```

Mandatory element, doesn't take any no attributes

The Agent Section

“Where should the pipeline or an individual step execute?”

```
pipeline {
    agent any
}
```

```
pipeline {
    agent {
        label 'java'
    }
}
```

Agent can be specified to run in a Docker container

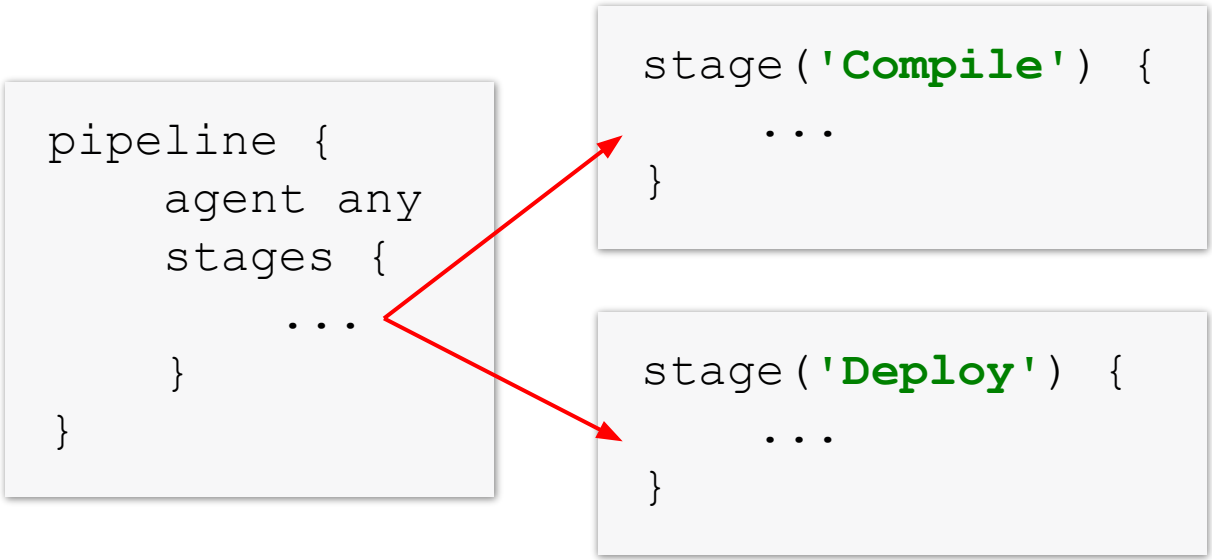
Docker Agent

Run a step or all steps in a container

```
pipeline {
    agent {
        docker { image 'node:20.11.1-alpine3.19' }
    }
}
```

The Stages and Stage Sections

Subset of tasks, visualized in UI



Running Stages in Parallel

Execute stages in parallel, join execution path

```
stages {
    parallel {
        stage('Unit Test') {
            ...
        }
        stage('Integration Test') {
            ...
        }
    }
}
```

The Steps Section

One or many tasks, “what to execute”

```
pipeline {
    agent any
    stages {
        stage('Compile') {
            ...
        }
    }
}
```

```
steps {
    echo 'Compiling...'
    sh './gradlew compileJava'
}
```

The Options Directive

Configures high-level pipeline-specific options

```
options {
    timeout(time: 1, unit: 'HOURS')
    buildDiscarder(logRotator(
        numToKeepStr: '10',
        artifactNumToKeepStr: '5'))
}
```

The Environment Directive

Set custom environment variables for pipeline or stage

```
environment {
    GITHUB_TOKEN = '3ahbfkx'
    PROFILE = 'staging'
}
```

Accessible with `$` notation in other portions of script

The Credentials Type

Access centrally-configured credentials

```
environment {
    GITHUB_TOKEN = credentials('GITHUB_TOKEN')
}
```

The Tools Directive

Reference globally-configured tools in pipeline definition

```
tools {
    maven 'apache-maven-3.0.1'
}
```

Tool is added to `PATH` and can be used by any agent

EXERCISE

Writing a Basic Jenkinsfile

The Triggers Directive

Only pulling is allowed but no support for pushing

```
triggers {
    pollSCM('H */4 * * 1-5')
}
```

Other triggers e.g. push to SCM can be configured

The Parameters Directive

Ask user to enter values to control runtime behavior

```
parameters {
    string(name: 'PROFILE', defaultValue: 'dev',
           description: 'Build for what environment?')
    password(name: 'TARGET_ENV_PWD',
             description: 'Enter deployment password')
}
```

Accessible with `$params.<NAME>` notation

The Input Directive

Valuable for implementing push-button release steps

```
timeout(time: 1, unit: 'DAYS') {
    input {
        message 'Deploy to Production?'
    }
}
```

Use in conjunction with `timeout` to avoid infinite loop

Conditional Stage Execution

Select from wide range of built-in conditions

```
when {
    not {
        branch 'master'
    }
}
```

The Post Directive

“What should happen after completion of pipeline/stage?”

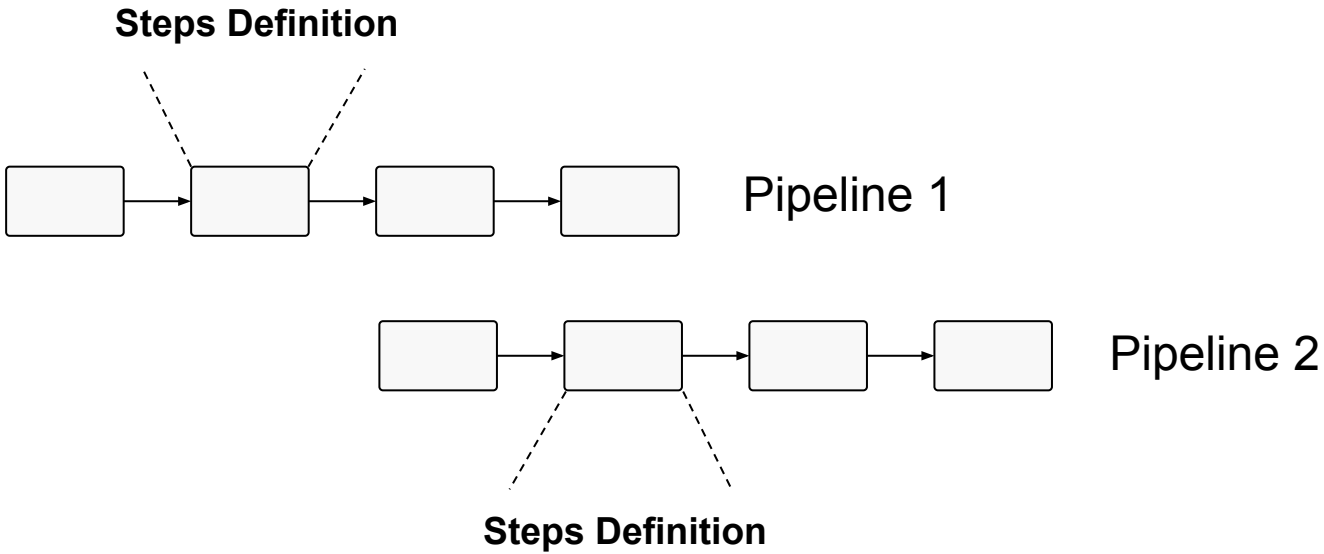
```
post {
    failure {
        mail to: 'benjamin.muschko@gmail.com',
             subject: 'Build failed', body: 'Please fix!'
    }
    success {
        archiveArtifacts artifacts: 'build/libs/**/*.jar',
                         fingerprint: true
        junit 'build/reports/**/*.xml'
    }
}
```

EXERCISE

Enhancing a Pipeline
with Advanced Features

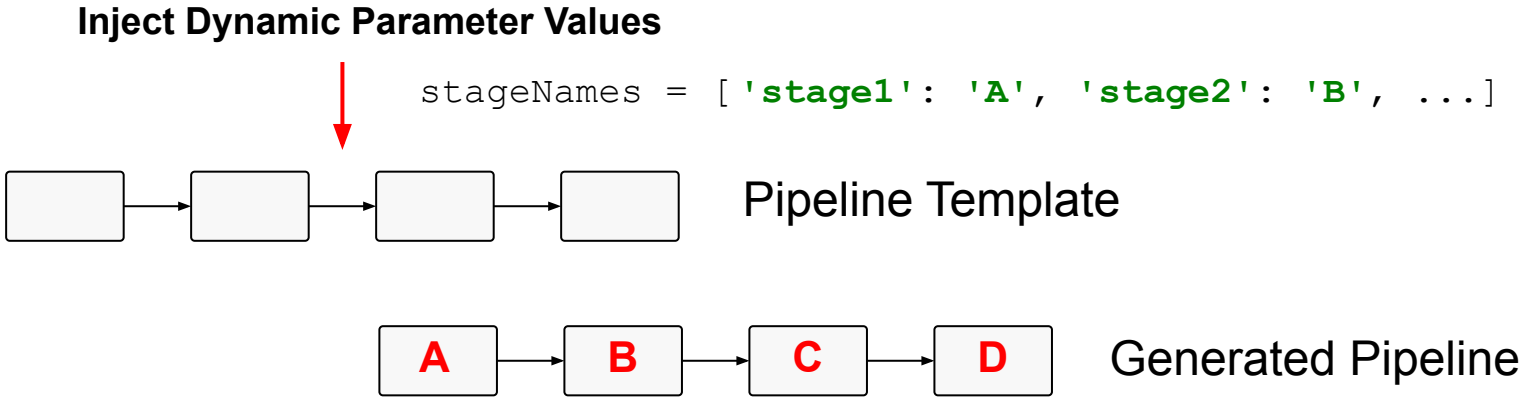
Keeping Pipelines DRY

Avoid copy/pasting pipeline code across multiple projects



Pipeline Templates

You can even define a “standard” pipeline if needed



Implementing Shared Libraries

Write once, reuse across multiple projects

- Written in Groovy
- Hosted in dedicated SCM repository
- Should follow versioning scheme

Repository Structure

Requires following the conventions

```
(root)
+- src                     # Groovy source files
|   +- org
|       +- foo
|           +- Bar.groovy  # for org.foo.Bar class
+- vars
|   +- foo.groovy          # for global 'foo' variable
|   +- foo.txt             # help for 'foo' variable
+- resources               # resource files (external libraries only)
    +- org
        +- foo
            +- bar.json    # static helper data for org.foo.Bar
```

Example of a Shared Library

Simplifying the usage of the Gradle Wrapper in build step

Windows:

```
gradlew.bat clean install
```

Unix/Linux/MacOSX:

```
./gradlew clean install
```

Build Step

```
if (OS == Windows) {
    ...
} else {
    ...
}
```

Source Code: https://github.com/bmuschko/jenkins-shared-lib-gradle

Implementing a Global Variable

vars/gradlew.groovy

```
def call(String... args) {
    if (isUnix()) {
        sh "./gradlew ${args.join(' ')} -s"
    } else {
        bat "gradlew.bat ${args.join(' ')} -s"
    }
}
```

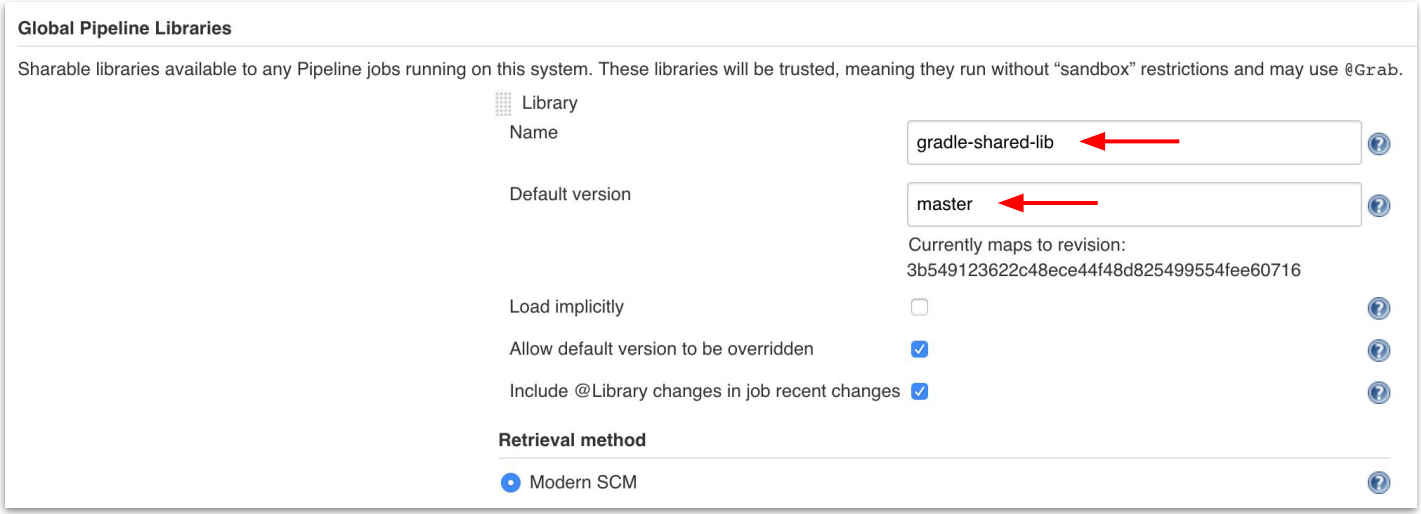
Implementing a Class

src/com/bmuschko/jenkins/Gradle.groovy

```
package com.bmuschko.jenkins

import org.apache.commons.lang3.SystemUtils

@Grab('org.apache.commons:commons-lang3:3.8.1')
class Gradle implements Serializable {
   // inject steps into constructor

   def wrapper(String... args) {
       if (!SystemUtils.IS_OS_WINDOWS) {
           steps.sh "./gradlew ${args.join(' ')} -s"
       } else {
           steps.bat "gradlew.bat ${args.join(' ')} -s"
       }
   }
}
```

Configuring a Shared Library

Manage Jenkins > Configure System



Using a Global Variable

Global variable invoked as built-in step

```
@Library('gradle-shared-lib') _

pipeline {
    agent any
    stages {
        stage('Gradle version with global var') {
            steps {
                gradlew '-v'
            }
        }
    }
}
```

The Libraries Directive

Declarative pipeline syntax provides shortcut notation

```
pipeline {
    agent any
    libraries {
        lib('gradle-shared-lib')
    }
    stages {
        ...
    }
}
```

Using a Class Implementation

Calls a Groovy method invoked as built-in step

```
@Library('gradle-shared-lib') import com.bmuschko.jenkins.Gradle

def gradle = new Gradle(this)

pipeline {
    agent any
    stages {
        stage('Gradle version with class') {
            steps {
                script {
                    gradle.wrapper '-v'
                }
            }
        }
    }
}
```

Requesting a Library Version

Add version to name with `@` notation

```
@Library('gradle-shared-lib@master')
@Library('gradle-shared-lib@2.1')
@Library('gradle-shared-lib@712e341aead532...')
```

Branch

Tag

Commit hash

General pattern: `<libname>@<version>`

EXERCISE

Writing and Using a
Shared Library

Q & A

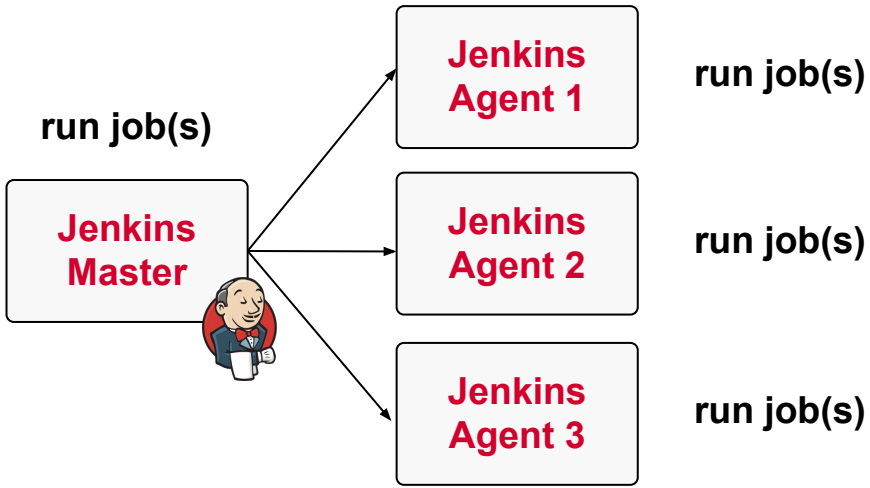
BREAK

CD-as-Code Best Practices

Distributed Architecture, High Availability, Traceability

Avoid running jobs on Master(s)

If workload increases, Master will likely run out of resources



Potential Security Issue

Malicious script has full permissions on Master

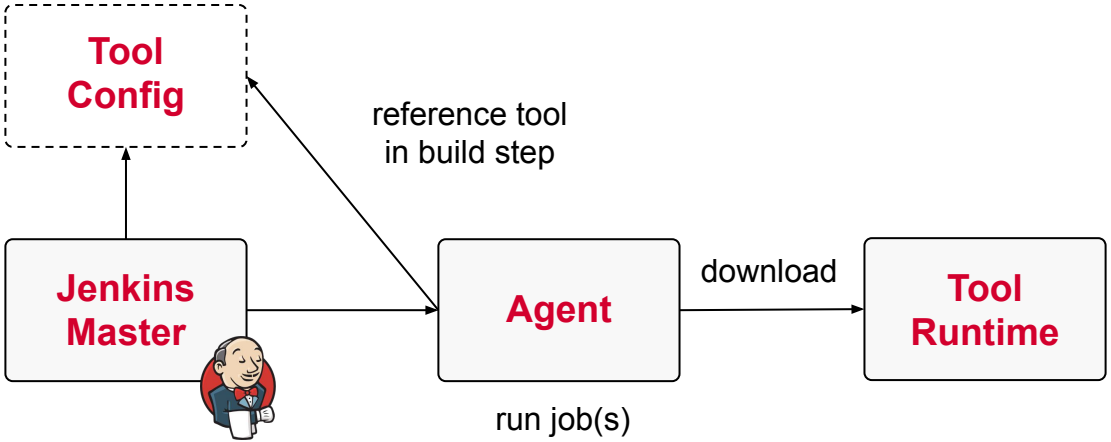
Defining Reusable Agents

Generalize as much as possible, specialize if needed

- Create agents with infrastructure tools e.g. Chef, Ansible
- Allow agents to be shared across teams
- Selected specialized agents via labels
- Prefer running build steps in Docker

Using Tools on Agents

Reference globally configured tools configuration



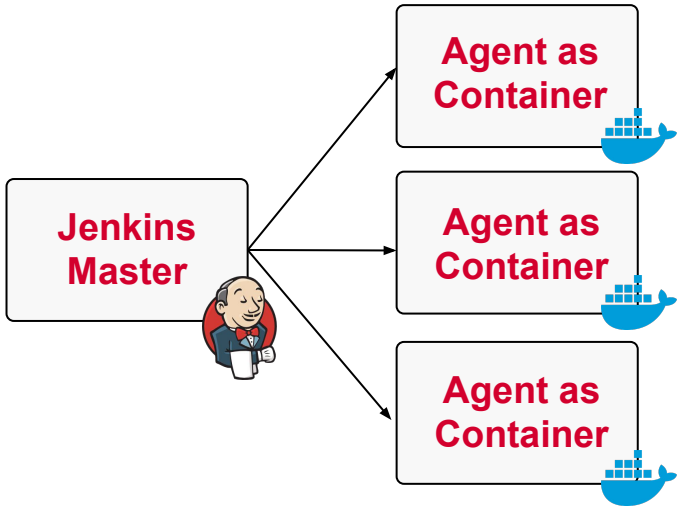
Setting Up Cloud Agents

Offload build execution to on-demand agents

- **Amazon EC2 Plugin:** AWS EC2 on-premise instances
- **JClouds Plugin:** Any cloud provider supported by JCloud libraries

Running Container Agents

On-demand, temporary Docker containers to run jobs



- start container
- run job
- stop container
- remove container

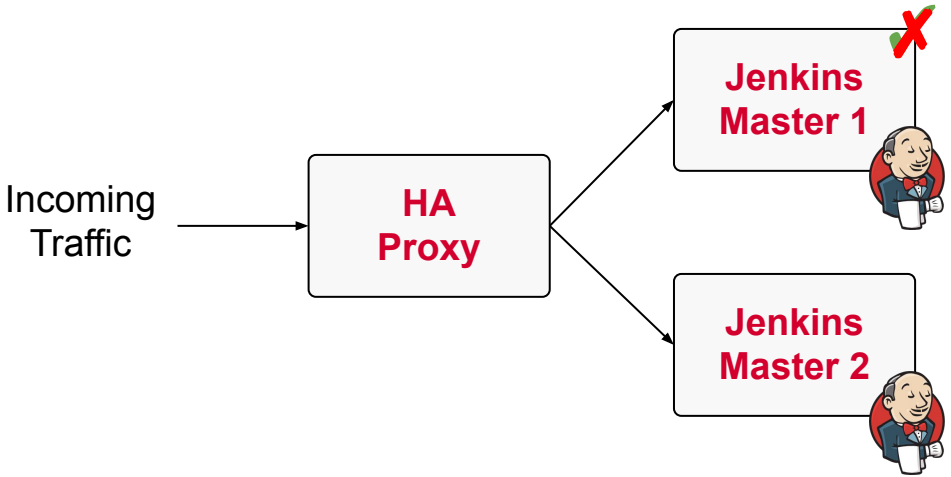
Hardware for Master node

There's no one-size-fits-all solution

- **CPU/Memory Requirements:** Depend on usage
- **Memory Guideline:** Each build node connection will take 2-3 threads, which equals about 2 MB or more of memory

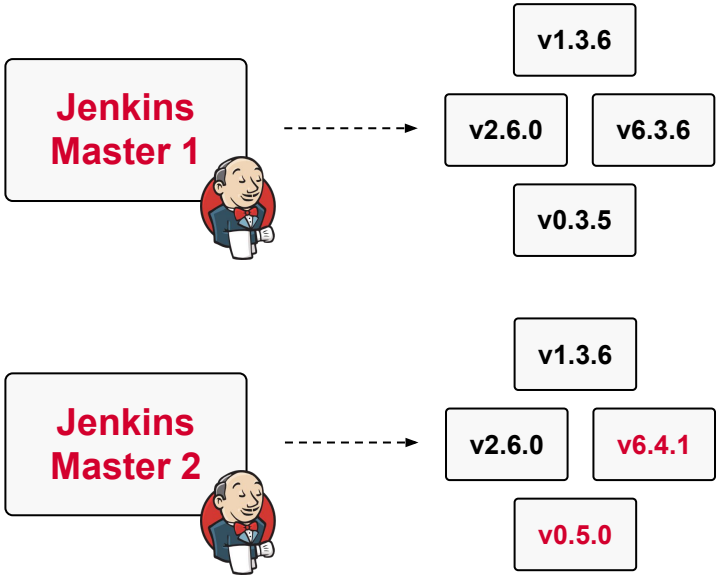
High Availability for Master node

High Availability plugin provides failover to backup



Testing Plugin Updates

Incompatible changes can bring down Master node



Route traffic to node 2

or

Repeat process on node 1

Traceability

Needed for audit requirements or for impact analysis

- **Artifacts:** Trace code from commit to deployment with built-in Fingerprinting
- **Docker images:** Trace the build and deployment history of Docker images with the CloudBees Docker Traceability plugin

Q & A

Summary & Wrap Up

Last words of advice...

O'REILLY®

Thank you