



## Certified Jenkins Engineer (CJE) Crash Course



# About the trainer



**bmuschko**



**bmuschko**



**bmuschko.com**



 **AUTOMATED  
ASCENT**  
[automatedascent.com](http://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/jenkins/certification>



# The Curriculum

Section Title	# Questions	% of total exam
<b>Jenkins Fundamentals</b>	10	17%
<b>Jenkins Administration</b>	25	42%
<b>Jenkins Build Technologies: Pipeline</b>	18	30%
<b>Jenkins Build Technologies: Freestyle</b>	7	11%
Total Number of Questions		60 (90 minutes)
Total		100%



---

# 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

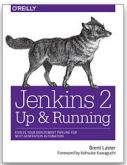
*Cross-reference topics for more details*



[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

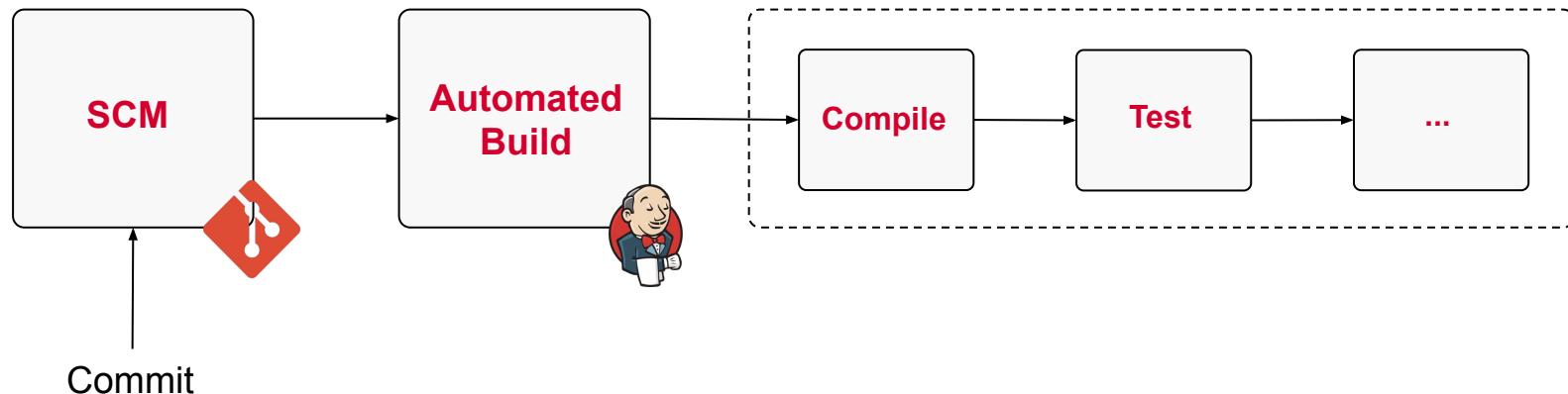


# Terminology and Jenkins Installation

Key Concepts, Getting Started

# Continuous Integration (CI)

*Run an automated build for every commit to SCM*



---

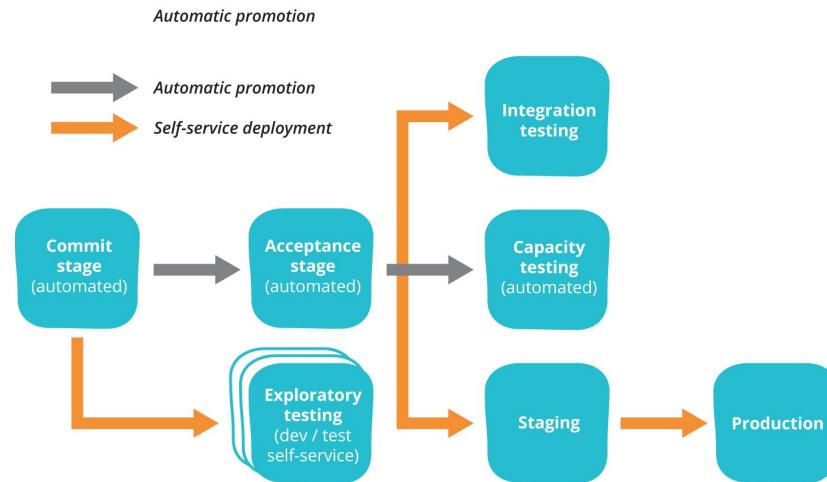
# CI Best Practices

- Frequent commits
- No 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*

**Continuous Delivery**

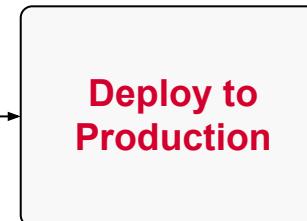
Manual  
Trigger



**Continuous Deployment**

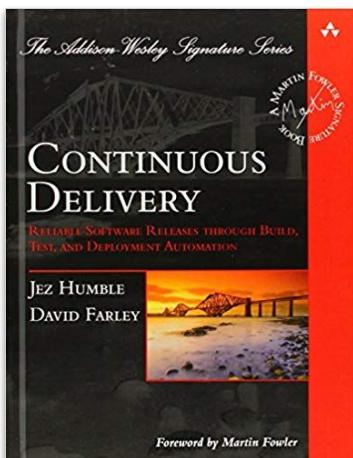
Automatic

vs.



# 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*

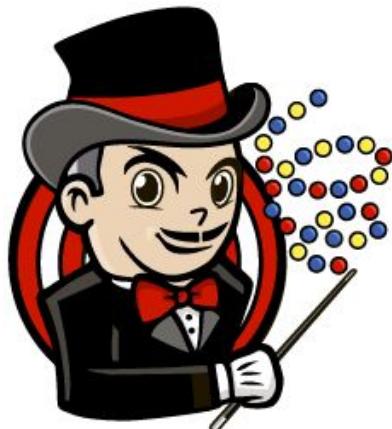


- Installation of JDK 8 or higher
- Set JAVA\_HOME as environment variable
- Container image already contains JDK



# Installation Wizard

*Guides through installation process*



Accessible through web browser

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

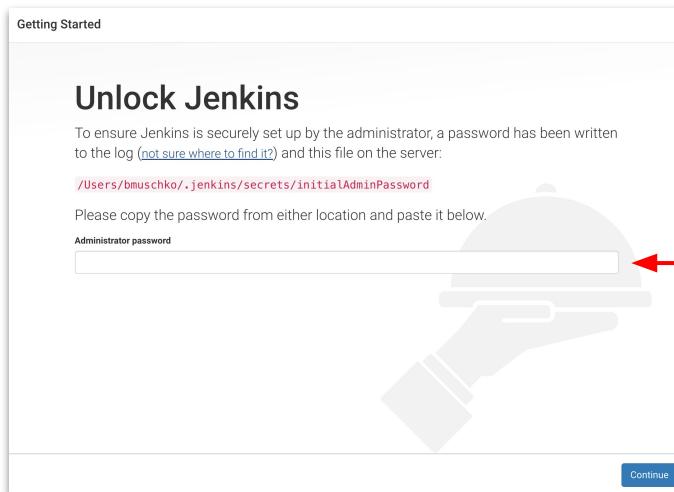
cc creative  
commons

<https://jenkins.io/artwork/>



# Unlocking Jenkins

*Auto-generated password on local disk*

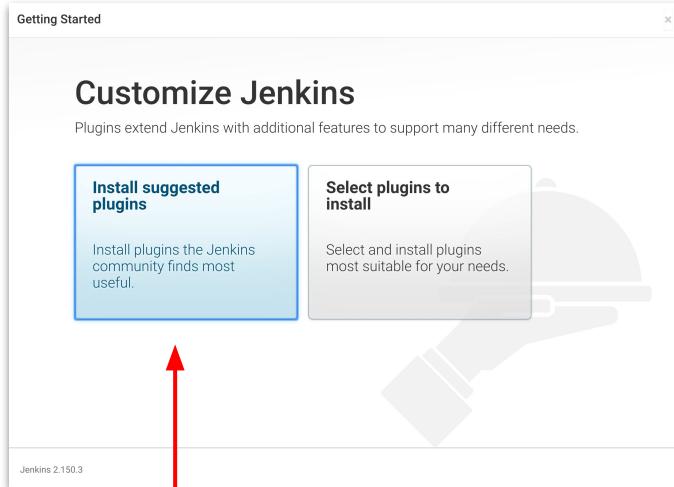


```
$ cat /Users/bmuschko/.jenkins/secrets/initialAdminPassword  
6ce47a19e64045bc87d33c3843f2254b
```



# Initial Plugin Installation

*Suggested plugins include the Pipeline functionality*



The screenshot shows the Jenkins 'Getting Started' page with the 'Select plugins to install' section open. A red arrow points to the 'Pipeline' plugin, which is highlighted with a pink background. The Pipeline plugin is listed under the 'Folders' category. The right side of the screen displays a detailed list of Jenkins features and their descriptions.

Getting Started

## Getting Started

✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding
✗ Timestamper	✓ Workspace Cleanup	✓ Ant	✗ Gradle
✗ Pipeline	✗ GitHub Branch Source	✗ Pipeline: Github Groovy Libraries	✗ Pipeline: Stage View
✗ Git	✗ Subversion	✗ SSH Slaves	✗ Matrix Authorization Strategy
✗ PAI Authentication	✗ LDAP	✗ Email Extension	✗ Mailer

Structs  
\*\* Pipeline: Step API  
\*\* SCM API  
\*\* Pipeline: API  
\*\* DSL API  
\*\* bouncycastle API  
OWASP Markup Formatter  
Build Timeout  
Credentials  
\*\* Basic Credentials  
\*\* SSH Credentials  
Credentials Binding  
Timestamper  
\*\* Pipeline Supporting APIs  
\*\* Durable Task  
\*\* Pipeline Nodes and Processes  
\*\* Matrix Project  
\*\* Resource Disposer  
Workspace Cleanup  
HTML  
\*\* JavaScript GUI Lib: ACE  
\*\* Editor bundle  
\*\* JavaScript GUI Libs: jQuery bundles (jQuery and jQuery UI)  
\*\* - require dependency

Jenkins 2.150.3



# Optional Steps

*Creating customized admin user and Jenkins URL*

Getting Started

## Create First Admin User

Username:

Password:

Confirm password:

Full name:

E-mail address:



Jenkins 2.150.3 Continue as admin Save and Continue

Getting Started

## Instance Configuration

Jenkins URL:  

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `BUILD_URL` environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

Jenkins 2.150.3 Not now Save and Finish



---

# 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*

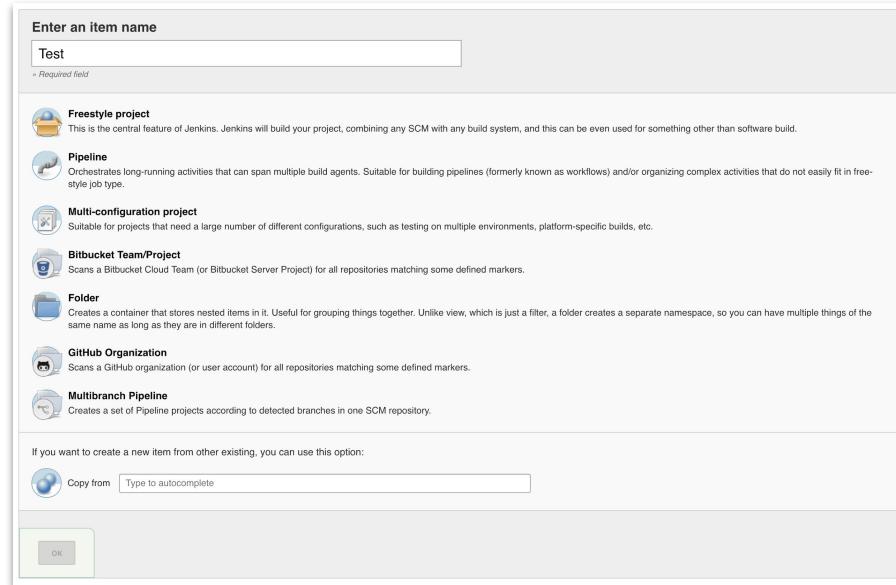
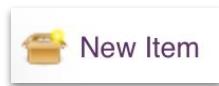
- Compiling a Java-based project
- Executing different types of tests
- Deploying an artifact to a target environment



# Different Types of Jobs

*Selected at the time of creation, not changeable later*

From Dashboard, click...



The screenshot shows the Jenkins "New Item" creation dialog. At the top, there is a search bar labeled "Enter an item name" containing the text "Test". Below the search bar, a note says "Required field". A list of job types is displayed:

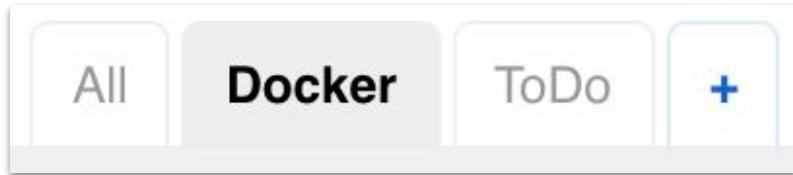
- Freestyle project**: This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
- Pipeline**: Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.
- Multi-configuration project**: Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
- Bitbucket Team/Project**: Scans a Bitbucket Cloud Team (or Bitbucket Server Project) for all repositories matching some defined markers.
- Folder**: Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.
- Github Organization**: Scans a GitHub organization (or user account) for all repositories matching some defined markers.
- Multibranch Pipeline**: Creates a set of Pipeline projects according to detected branches in one SCM repository.

At the bottom of the dialog, there is a note: "If you want to create a new item from other existing, you can use this option:" followed by a "Copy from" button and a "Type to autocomplete" input field. Finally, there is an "OK" button at the bottom left.

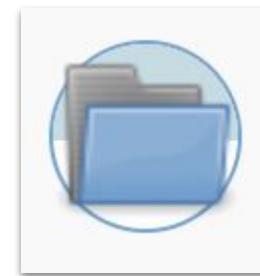


# Organizing Jobs

*“Grouping of jobs by teams or projects”*



vs.



**Views**

**Folders**



# Using Views

*Global and personalized flat lists*

Button for creating a new view

All	Docker	ToDo	+			
S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav
		<a href="#">docker-for-jym</a>	3 mo 14 days - #1	N/A	1 min 32 sec	
		<a href="#">docker-training</a>	3 mo 15 days - #1	N/A	57 sec	
		<a href="#">docker-training-today</a>	1 mo 10 days - #1	N/A	1 min 40 sec	

Icon: [S](#) [M](#) [L](#)

[Legend](#) [RSS for all](#) [RSS for failures](#) [RSS for just latest builds](#)



# Using Folders

*Nested organizational structure*

All	Docker	ToDo	+	S	W	Name ↓	Last Success	Last Failure	Last Duration	Fav
						<a href="#">docker-for-jvm</a>	3 mo 14 days - #1	N/A	1 min 32 sec	
						<a href="#">docker-training</a>	3 mo 15 days - #1	N/A	57 sec	
						<a href="#">docker-training-today</a>	1 mo 10 days - #1	N/A	1 min 40 sec	
						<a href="#">Training</a>	N/A	N/A	N/A	

Icon: [S](#) [M](#) [L](#)

[Legend](#) [RSS for all](#) [RSS for failures](#) [RSS for just latest builds](#)



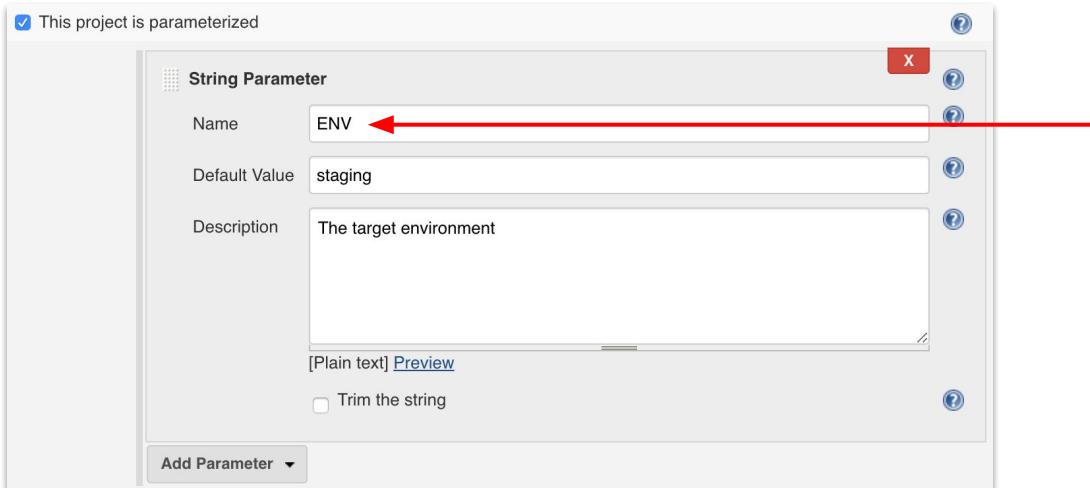
## Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.



# Parameterized Jobs

*Pass parameters to control runtime behavior*



Accessible as  
environment variable  
named ENV



---

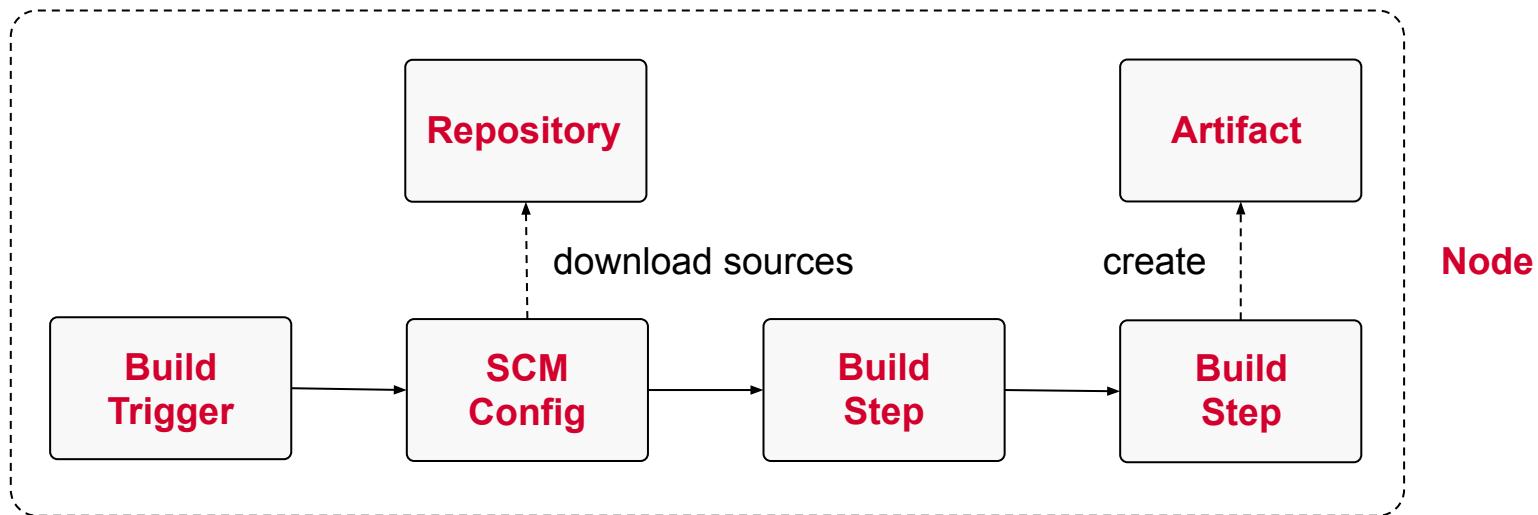
# EXERCISE

Defining, Configuring  
and Organizing a Job



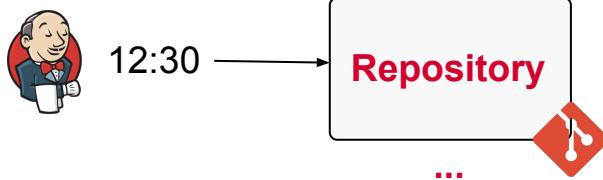
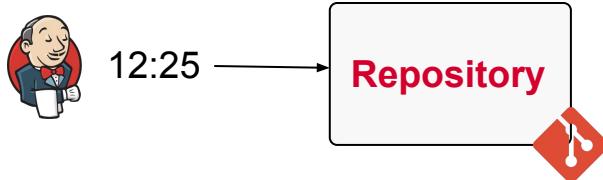
# What is a Build?

*Execution of an automation task*



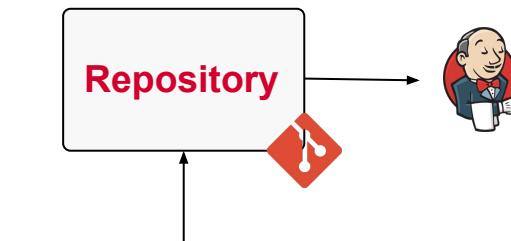
# Defining Build Triggers

*Periodic checking or direct notification upon a change*



**Pull every 5 mins**

VS.

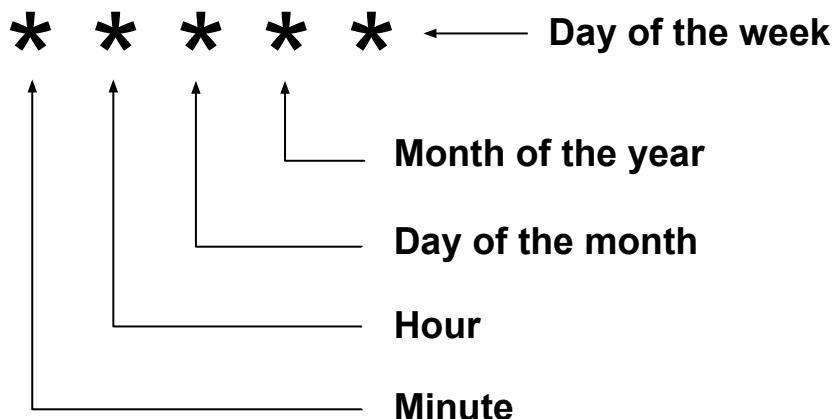


**Push upon commit**



# 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

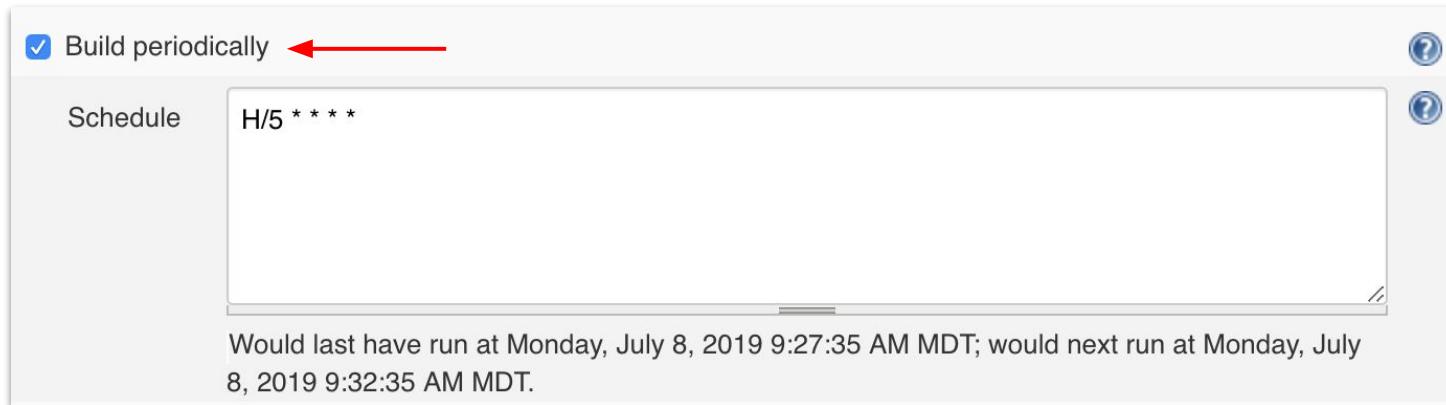
**@hourly** Every hour

**@midnight** At midnight



# Defining a Periodic Build Trigger

*Configure > Build Triggers > Build Periodically*

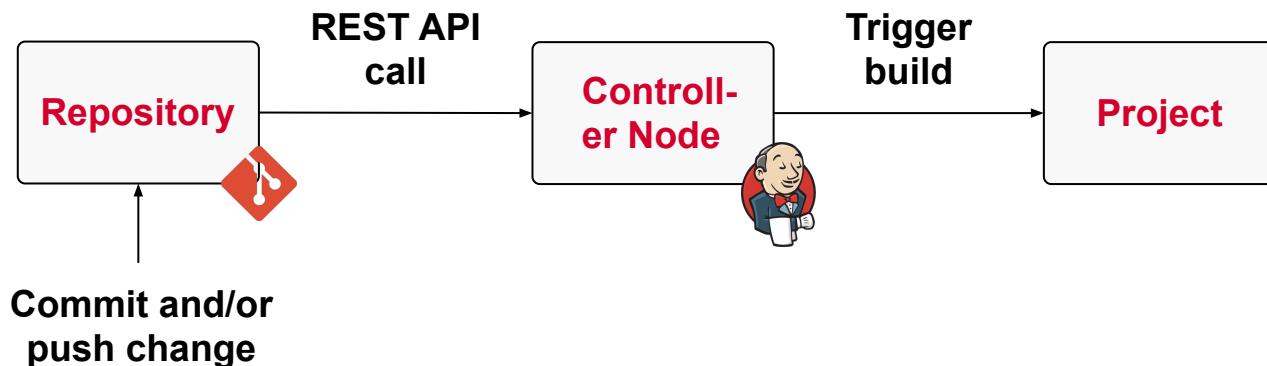


The screenshot shows a configuration dialog for a build trigger. At the top left is a checked checkbox labeled "Build periodically". A red arrow points to this checkbox. To its right is a "Schedule" field containing the cron expression "H/5 \* \* \* \*". Below the schedule is a note: "Would last have run at Monday, July 8, 2019 9:27:35 AM MDT; would next run at Monday, July 8, 2019 9:32:35 AM MDT." There are two question mark icons in the top right corner of the dialog.



# How Does Pushing Work?

*Communication via Jenkins REST API*



# Configuring a GitHub Hook

*Configure > Build Triggers > GitHub Hook Trigger...*

GitHub hook trigger for GITScm polling 

If Jenkins will receive PUSH GitHub hook from repo defined in Git SCM section it will trigger Git SCM polling logic.  
So polling logic in fact belongs to Git SCM.

(from [GitHub plugin](#))



# GitHub Webhook Configuration

*Settings > Webhooks > Add webhook*

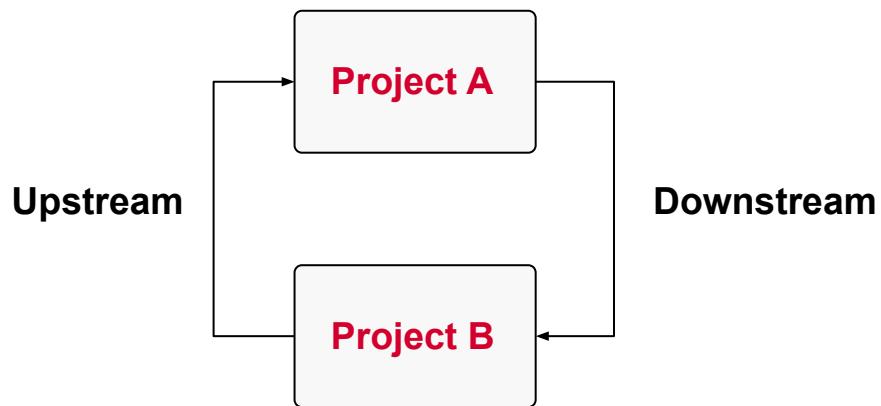
The screenshot shows the 'Webhooks / Add webhook' configuration page on GitHub. The page has the following fields:

- Payload URL \***: A text input field containing "https://localhost:8080/github-webhook/". A red arrow points to this field.
- Content type**: A dropdown menu set to "application/json". A red arrow points to this field.
- Secret**: An empty text input field.
- SSL verification**: A section with a note about verifying SSL certificates. It contains two radio buttons: "Enable SSL verification" (selected) and "Disable (not recommended)". A red arrow points to this section.
- Which events would you like to trigger this webhook?**: A section with three radio button options:
  - Just the push event.
  - Send me everything.
  - Let me select individual events.
- Active**: A checkbox labeled "We will deliver event details when this hook is triggered." It is checked. A red arrow points to this section.
- Add webhook**: A green button at the bottom.



# Relationship Between Projects

*Sometimes projects do not live in isolation*



# Triggering Another Project

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

The screenshot shows a configuration dialog for "Build after other projects". At the top left is a checked checkbox labeled "Build after other projects are built". A red arrow points to this checkbox. Below it is a section titled "Projects to watch" containing a text input field with the value "todo-web-app" and a help icon. Underneath is a list of three trigger options, each preceded by a radio button. The first option, "Trigger only if build is stable", has a blue dot indicating it is selected. A red arrow points to this radio button. The other two options are "Trigger even if the build is unstable" and "Trigger even if the build fails", both with white circles.

Build after other projects are built

Projects to watch todo-web-app

Trigger only if build is stable

Trigger even if the build is unstable

Trigger even if the build fails



---

# 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*

The image shows the Jenkins configuration interface for adding build steps. On the left, a sidebar lists various build step options. A red arrow points from the 'Execute Windows batch command' option to a detailed configuration window on the right. Another red arrow points from the 'Invoke top-level Maven targets' option to another configuration window below it.

**Build**

- Add build step ▾ (highlighted by a red arrow)
- Execute Windows batch command (selected, highlighted by a blue box)
- Execute shell
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets
- JIRA: Add related environment variables to build
- JIRA: Create new version
- JIRA: Issue custom field updater
- JIRA: Mark a version as Released
- JIRA: Progress issues by workflow action
- Run with timeout
- Set build status to "pending" on GitHub commit

**Execute shell**

Command: `echo "Hello World!"`

See [the list of available environment variables](#)

Advanced...

Add build step ▾

**Invoke top-level Maven targets**

Goals: `clean install`

Advanced...

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 Global Tools

*Manage Jenkins > Global Tools Configuration*

The image shows the Jenkins Global Tool Configuration interface. On the left, under 'Maven' installations, a new Maven configuration is being added with the name 'Maven 3.6.1'. A red arrow points to this name field. Below it, there's an option to 'Install automatically' and a dropdown for 'Install from Apache' set to version 3.6.1. On the right, the 'Global Tool Configuration' page is shown with a title 'Configure tools, their locations and automatic installers.' Below it, the 'Build' section is open, showing a step to 'Invoke top-level Maven targets' with the 'Maven Version' set to 'Maven 3.6.1'. A red arrow points to this 'Maven Version' field. The 'Goals' field is empty, and there's an 'Advanced...' button. At the bottom, there's a 'Add build step' button.

Maven

Maven installations

Add Maven

Maven

Name **Maven 3.6.1** ←

Install automatically

Install from Apache

Version **3.6.1** ←

Global Tool Configuration

Configure tools, their locations and automatic installers.

Build

Invoke top-level Maven targets

Maven Version **Maven 3.6.1** ←

Goals

Advanced...

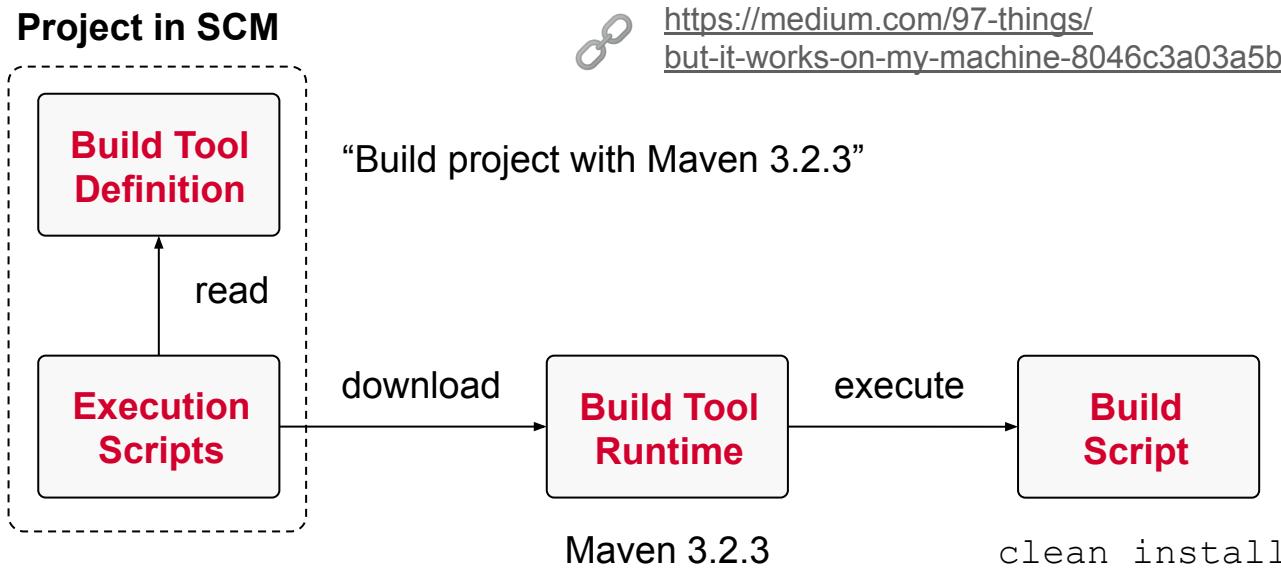
Add build step

Auto-install tool and  
make it available on  
Jenkins nodes



# Using a Build Tool Wrapper

*Project-specific definition of build tool runtime*



---

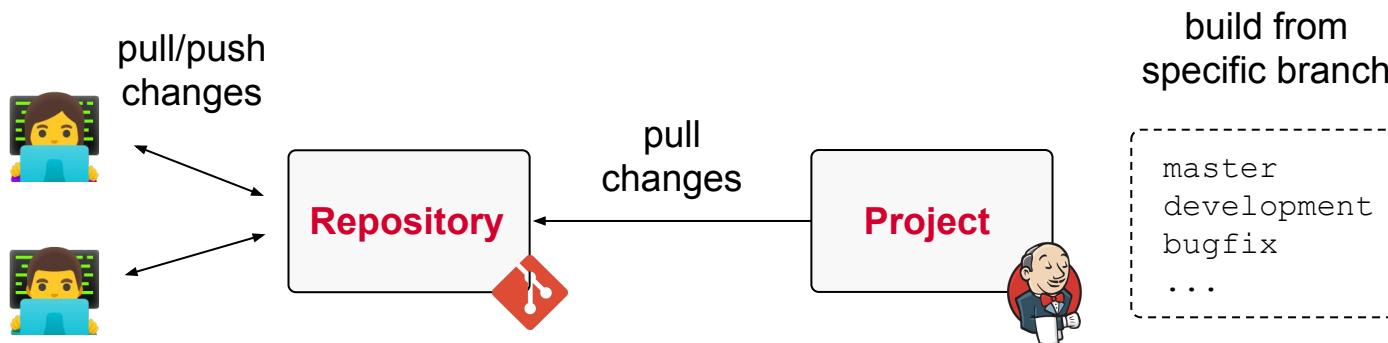
# EXERCISE

Configuring Build  
Triggers and Steps for  
a Job



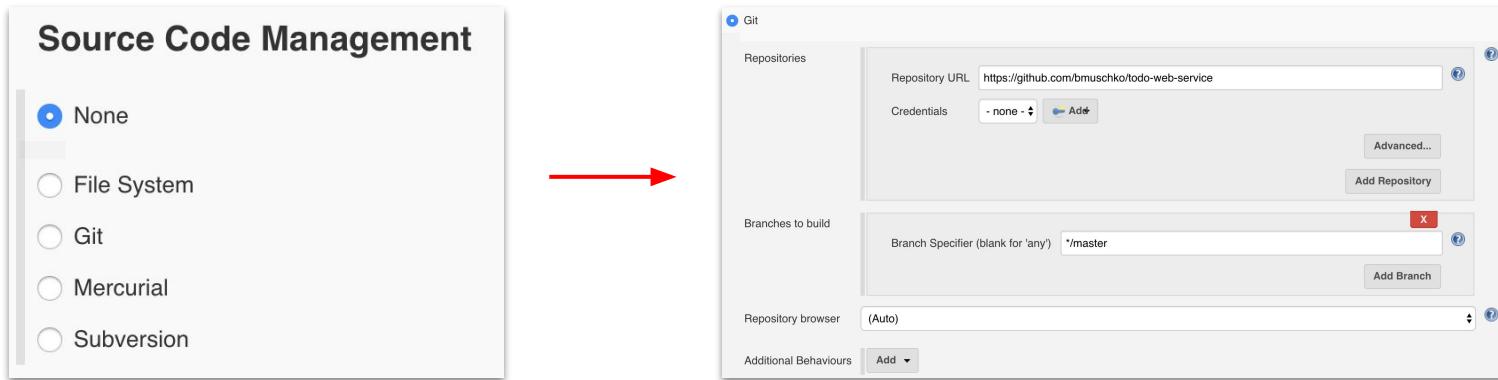
# What's the Purpose of a SCM?

*Central “source of truth” for source code & configuration*



# Supported SCM Products

*Configure > Source Code Management*



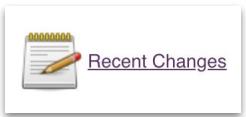
Additional Cloud-based SCMs available e.g. GitHub, BitBucket



# Viewing the Changelog

*“Show me the commits pulled from SCM”*

From Project, click...



**Changes**

**#8 (Jan 22, 2019 9:00:46 AM)**

- 1. Add release step — [Benjamin Muschko / githubweb](#)

**#7 (Jan 22, 2019 8:58:40 AM)**

- 1. Add code analysis stage — [Benjamin Muschko / githubweb](#)

**#6 (Jan 22, 2019 8:56:52 AM)**

- 1. Fix command — [Benjamin Muschko / githubweb](#)

**#5 (Jan 22, 2019 8:56:04 AM)**

- 1. Use correct ID — [Benjamin Muschko / githubweb](#)

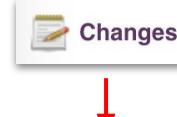
**#4 (Jan 22, 2019 8:55:12 AM)**

- 1. SetCodecov environment variable — [Benjamin Muschko / githubweb](#)

**#3 (Jan 22, 2019 8:53:25 AM)**

- 1. Add test stage — [Benjamin Muschko / githubweb](#)
- 2. Add missing steps — [Benjamin Muschko / githubweb](#)

From specific Build, click...



**Changes**

**Summary**

1. Add release step ([details](#))

Commit [912762e8a856ffe7f4bbc5a5b3b5dbc1255f1732](#) by [Benjamin Muschko](#)  
Add release step

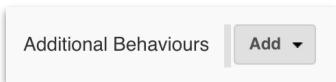
[Jenkinsfile \(diff\)](#)



# Incremental & Clean Check Out

*Trade off: Checkout speed vs. clean workspace*

From SCM configuration, click...



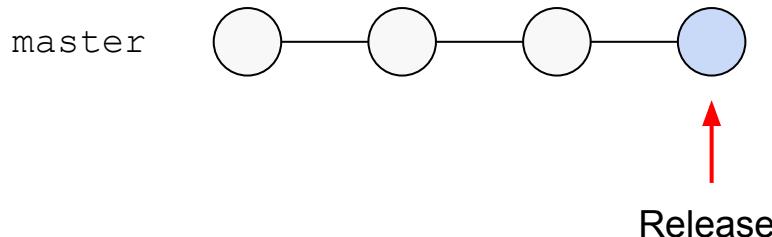
- Advanced checkout behaviours
- Advanced clone behaviours
- Advanced sub-modules behaviours
- Calculate changelog against a specific branch
- Check out to a sub-directory
- Check out to specific local branch
- Clean after checkout
- Clean before checkout
- Create a tag for every build
- Custom SCM name
- Custom user name/e-mail address
- Don't trigger a build on commit notifications
- Force polling using workspace
- Git LFS pull after checkout
- Merge before build
- Polling ignores commits from certain users
- Polling ignores commits in certain paths
- Polling ignores commits with certain messages
- Prune stale remote-tracking branches
- Sparse Checkout paths
- Strategy for choosing what to build
- Use commit author in changelog
- Wipe out repository & force clone



# Branch and Merge Strategies

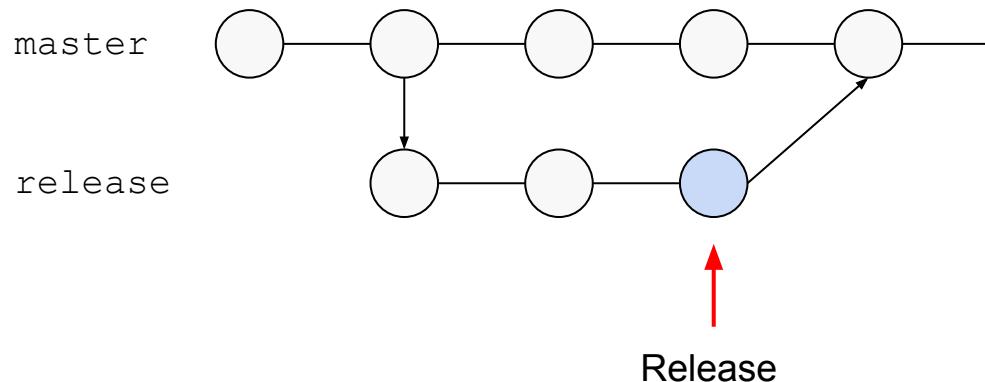
*Pick the most suitable for your team, workflow & culture*

Only commit to master branch, release from master



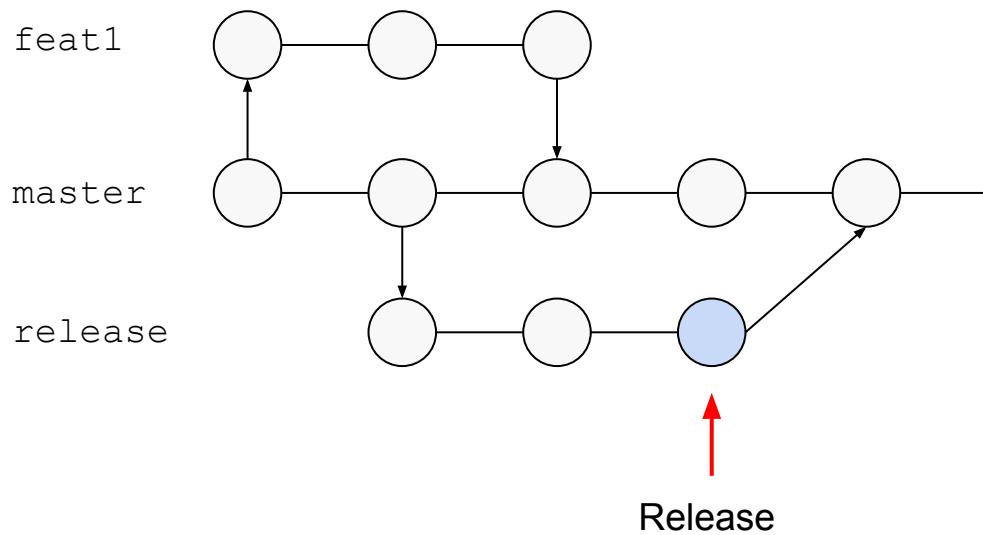
# Release Branching

*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

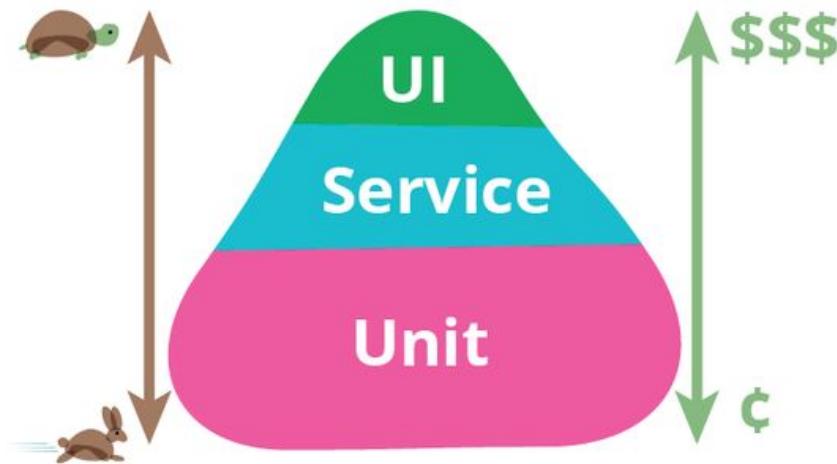
*Deliver product with acceptable quality*

- 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*

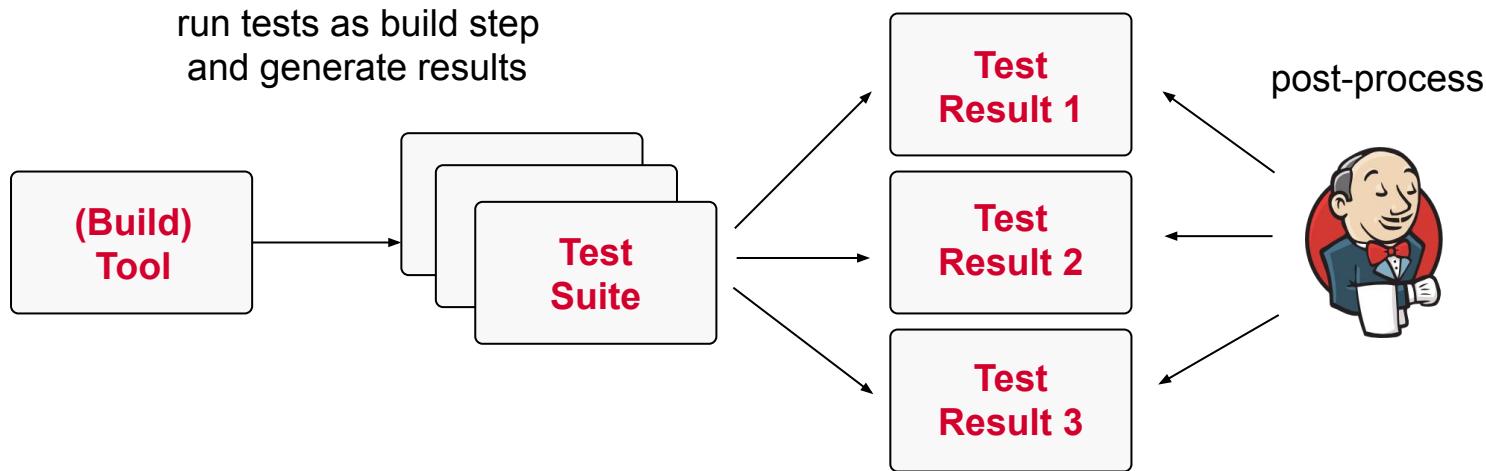


Source: <https://martinfowler.com/bliki/TestPyramid.html>



# 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



# Processing JUnit Test Results

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

The screenshot shows the Jenkins configuration interface. On the left, a sidebar lists various post-build actions. The 'Publish JUnit test result report' option is highlighted with a blue selection bar and has a red arrow pointing to it from the bottom-left. The main panel displays the configuration for the selected action. It includes fields for 'Test report XMLs' (set to 'build/test-results/TEST-\*.xml'), a note about 'Fileset "includes"', and options for 'Retain long standard output/error', 'Health report amplification factor' (set to 1.0), 'Allow empty results', and 'Do not fail the build on empty test results'. At the bottom, there is a 'Add post-build action ▾' button.

Aggregate downstream test results

Archive the artifacts

Build other projects

JIRA: Update relevant issues

Publish HTML reports

**Publish JUnit test result report**

Record fingerprints of files to track usage

Git Publisher

E-mail Notification

Editable Email Notification

JIRA: Create issue

JIRA: Create new version

JIRA: Mark a version as Released

JIRA: Move issues matching JQL to the specified version

Set GitHub commit status (universal)

Set build status on GitHub commit [deprecated]

Delete workspace when build is done

Add post-build action ▾

**Publish JUnit test result report**

Test report XMLs

build/test-results/TEST-\*.xml

Fileset 'includes' setting that specifies the generated raw XML report files, such as 'myproject/target/test-reports/\*.xml'. Basedir of the fileset is [the workspace root](#).

Retain long standard output/error

Health report amplification factor

1.0

1% failing tests scores as 99% health. 5% failing tests scores as 95% health

Allow empty results

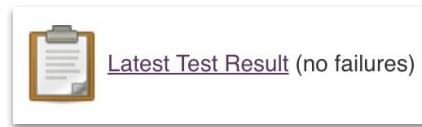
Do not fail the build on empty test results

Add post-build action ▾



# Test Trending & Reporting

*“Give me a historical and detailed test overview”*



Latest Test Result (no failures)

### Test Result

0 failures ( $\pm 0$ )

15 tests ( $\pm 0$ )  
Took 1.3 sec.  
[add description](#)

### All Tests

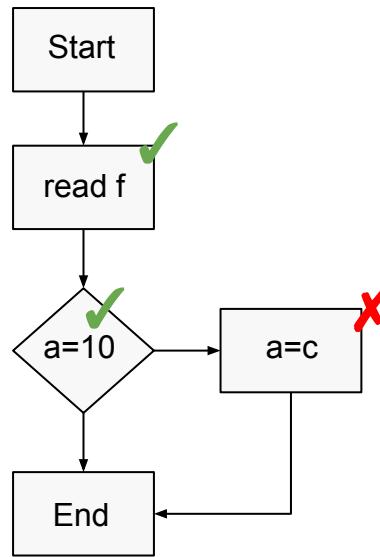
Package	Duration	Fail (diff)	Skip (diff)	Pass (diff)	Total (diff)
com.bmuschko.todo.webservice	0.76 sec	0	0	1	1
com.bmuschko.todo.webservice.model	11 ms	0	0	2	2
com.bmuschko.todo.webservice.model.controller	0.21 sec	0	0	6	6
com.bmuschko.todo.webservice.model.repository	0.35 sec	0	0	6	6



# 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      Support for some tools can be installed as Jenkins plugin
- Cobertura
- OpenClover



## Manage Plugins

Add, remove, disable or enable plugins that can extend the functionality of Jenkins.



# Processing JaCoCo Metrics

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

The image shows the Jenkins configuration interface for adding a post-build action. A red arrow points from the 'Record JaCoCo coverage report' option in the left sidebar to the detailed configuration screen on the right.

**Left Sidebar (Available Actions):**

- Aggregate downstream test results
- Archive the artifacts
- Build other projects
- JIRA: Update relevant issues
- Publish HTML reports
- Publish JUnit test result report
- Record JaCoCo coverage report** (highlighted with a blue box)
- Record fingerprints of files to track usage
- Git Publisher
- E-mail Notification
- Editable Email Notification
- JIRA: Create issue
- JIRA: Create new version
- JIRA: Mark a version as Released
- JIRA: Move issues matching JQL to the specified version
- Set GitHub commit status (universal)
- Set build status on GitHub commit [deprecated]
- Delete workspace when build is done

**Add post-build action ▾**

**Right Panel (Record JaCoCo coverage report Configuration):**

Path to exec files (e.g.: \*\*/target/\*\*.exec, \*\*/jacoco.exec)  
build/jacoco/test.exec

Inclusions (e.g.: \*\*/\*.class)  
Exclusions (e.g.: \*\*/\*Test\*.class)

Path to class directories (e.g.: \*\*/target/classDir, \*\*/classes)  
\*\*/build/classes

Path to source directories (e.g.: \*\*/mySourceFiles) Inclusions (e.g.: \*\*/\*.java, \*\*/\*.groovy, \*\*/\*.gs)  
\*\*/src/main/java      \*\*/\*.java

Exclusions (e.g.: generated/\*\*/\*.java)

Disable display of source files for coverage

Change build status according the thresholds

Instruction	% Branch	% Complexity	% Line	% Method	% Class
0	0	0	0	0	0
0	0	0	0	0	0

Fail the build if coverage degrades more than the delta thresholds

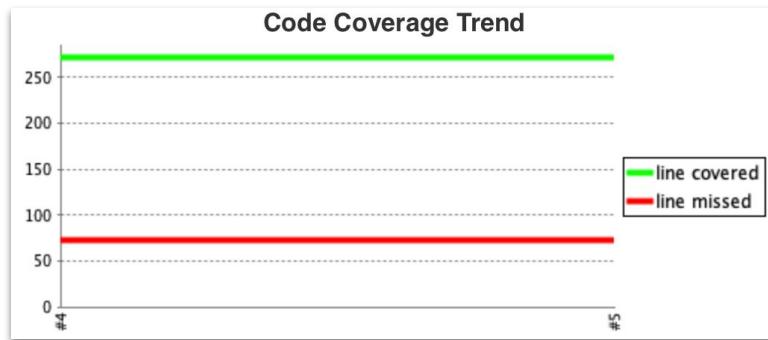
Instruction	% Branch	% Complexity	% Line	% Method	% Class
0	0	0	0	0	0

**Add post-build action ▾**



# Coverage Trending & Reporting

*“Give me a historical and detailed coverage overview”*



---

# 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

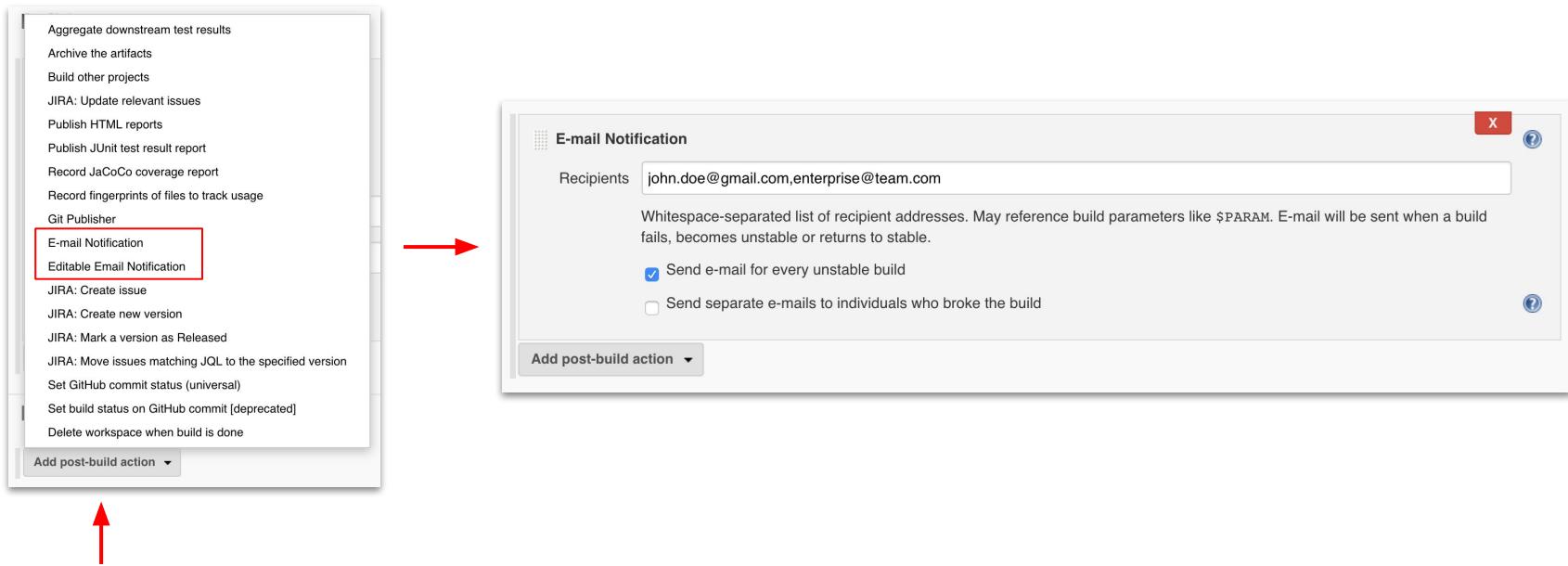
*Standard is email, others can be added via plugins*

- 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*



---

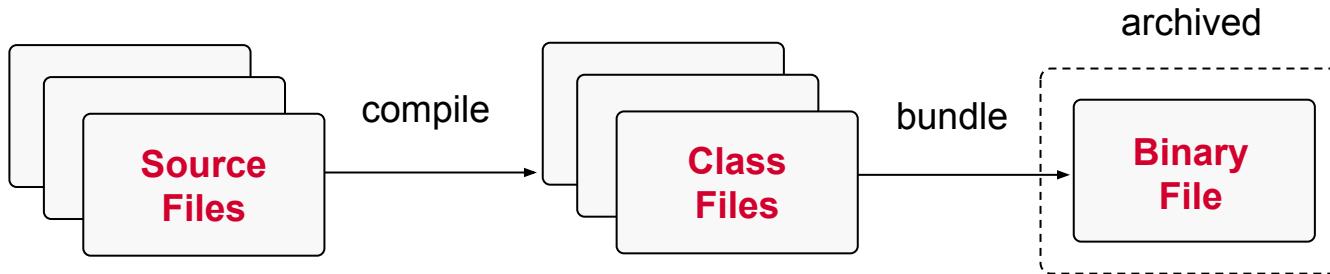
# EXERCISE

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*

The image shows the Jenkins configuration interface. On the left, a sidebar lists various post-build actions. The 'Archive the artifacts' option is highlighted with a blue selection bar and has a red arrow pointing to the right. To the right, a detailed configuration window for 'Archive the artifacts' is displayed. It shows the 'Files to archive' field containing 'build/libs/\*.war'. Below this window is another button labeled 'Add post-build action ▾'. At the bottom, there is a summary of the last successful artifact: 'Last Successful Artifacts' with a file icon, the filename 'todo-1.0.0.war', its size '30.86 MB', and a 'view' link.

- Aggregate downstream test results
- Archive the artifacts**
- Build other projects
- JIRA: Update relevant issues
- Publish HTML reports
- Publish JUnit test result report
- Record JaCoCo coverage report
- Record fingerprints of files to track usage
- Git Publisher
- E-mail Notification
- Editable Email Notification
- JIRA: Create issue
- JIRA: Create new version
- JIRA: Mark a version as Released
- JIRA: Move issues matching JQL to the specified version
- Set GitHub commit status (universal)
- Set build status on GitHub commit [deprecated]
- Slack Notifications
- Delete workspace when build is done

Add post-build action ▾

**Archive the artifacts**

Files to archive: build/libs/\*.war

Advanced...

Add post-build action ▾

Last Successful Artifacts

todo-1.0.0.war 30.86 MB view



# Artifact Retention Policy

*“I want to limit the number of archived artifacts”*

Discard old builds

Strategy Log Rotation

Days to keep builds

if not empty, build records are only kept up to this number of days

Max # of builds to keep

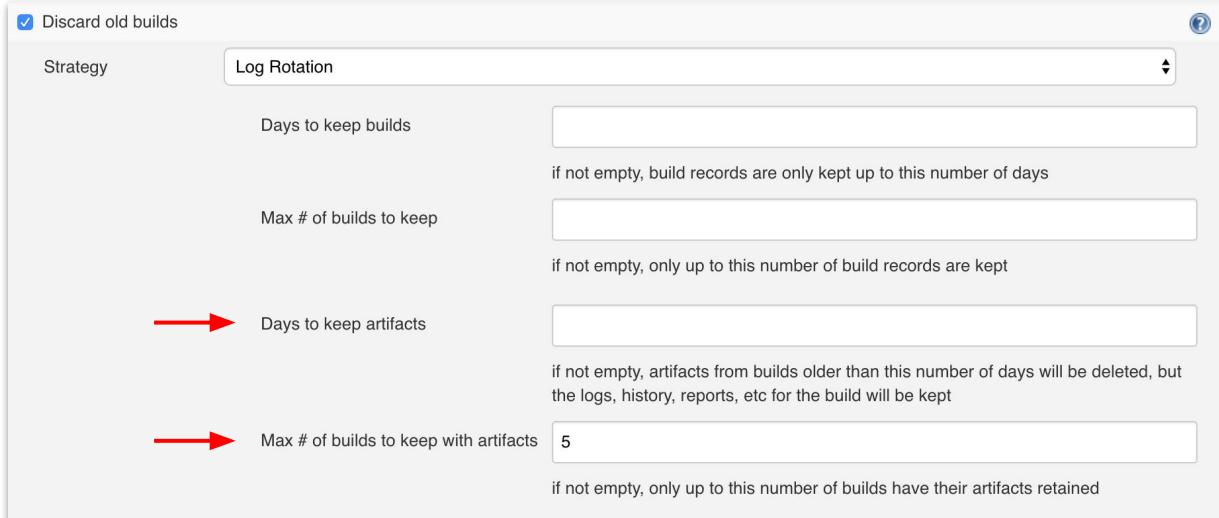
if not empty, only up to this number of build records are kept

Days to keep artifacts

if not empty, artifacts from builds older than this number of days will be deleted, but the logs, history, reports, etc for the build will be kept

Max # of builds to keep with artifacts

if not empty, only up to this number of builds have their artifacts retained



# Fingerprinting Artifacts

*Track a particular artifact back to the producing build*

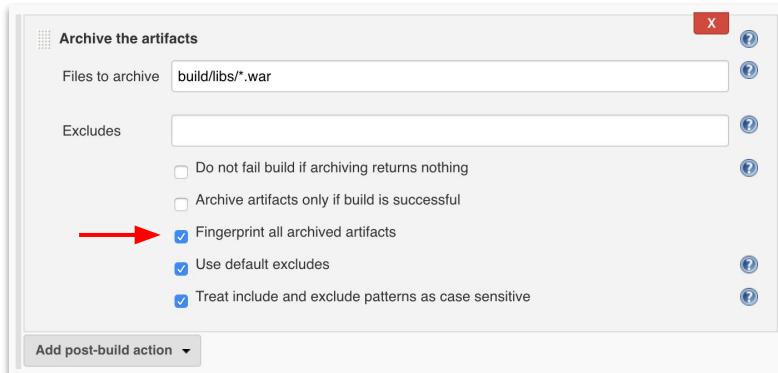
Archive the artifacts

Files to archive: build/libs/\*.war

Excludes:

- Do not fail build if archiving returns nothing
- Archive artifacts only if build is successful
- Fingerprint all archived artifacts
- Use default excludes
- Treat include and exclude patterns as case sensitive

Add post-build action ▾

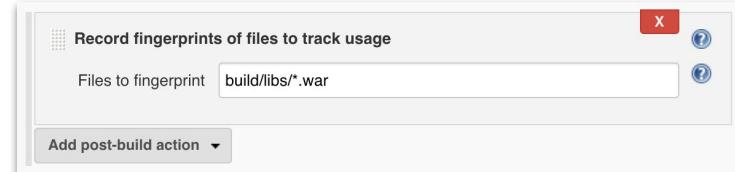


or

Record fingerprints of files to track usage

Files to fingerprint: build/libs/\*.war

Add post-build action ▾



Important for interdependent projects setups



# Viewing Artifact Usage

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

From specific Build, click...



File ↓	Original owner	Age	
build/libs/todo-1.0.0.war	this build	40 sec old	<a href="#">more details</a>

todo-1.0.0.war

Introduced 4 min 10 sec ago [test #5](#)

### Usage

This file has been used in the following places:

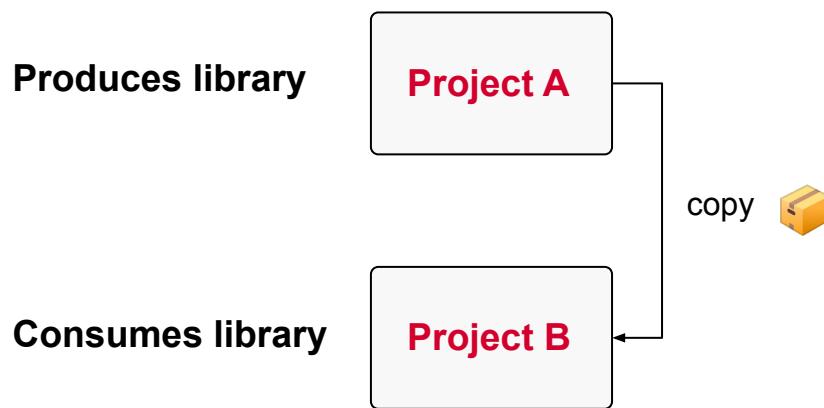
[test](#) [#5](#)

MD5: b25c3d16fe8b44525bca8f6e9867f997



# Cross-Project Artifacts

*Make the output of a build available to another project*



# Copying Artifacts

*Functionality only available through “Copy Artifact” plugin*

- [Copy artifacts from another project](#)
- Execute Windows batch command
- Execute shell
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets
- JIRA: Add related environment variables to build
- JIRA: Create new version
- JIRA: Issue custom field updater
- JIRA: Mark a version as Released
- JIRA: Progress issues by workflow action
- Run with timeout
- Set build status to "pending" on GitHub commit

Add build step ▾



Copy artifacts from another project

Project name: todo-web-service

Which build: Last build with artifacts

Artifacts to copy: build/libs/my-lib-\*.  
jar

Artifacts not to copy:

Target directory: build/libs

Parameter filters:

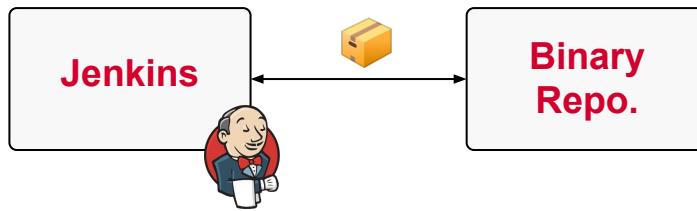
Flatten directories    Optional    Fingerprint Artifacts

Advanced...



# Using Binary Repositories

*Products offer features more suitable to enterprise projects*



---

# 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 > Manage Users*

 **Manage Users**  
Create/delete/modify users that can log in to this Jenkins



*You can't delete logged-in user*

## Users

These users can log into Jenkins. This is a sub set of [this list](#), which also contains auto-created users who really just made some commits on some projects and have no direct Jenkins access.

User ID	Name	
 bmuschko	Benjamin Muschko	



# Creating Users

*Manage Jenkins > Manage Users > Create User*

The diagram illustrates the Jenkins user creation process through three sequential steps:

- Create User**: A button with a user icon and the text "Create User". A red arrow points from this button to the next step.
- Create User**: A form titled "Create User" with fields:
  - Username:
  - Password:
  - Confirm password:
  - Full name:
  - E-mail address:A "Create User" button is at the bottom.
- Users**: A list titled "Users" with the following content:

These users can log into Jenkins. This is a sub set of [this list](#), which also contains auto-created users who really just made some commits on some projects and have no direct Jenkins access.

User ID	Name	Action
bmuschko	<a href="#">Benjamin Muschko</a>	
jdoe	<a href="#">John Doe</a>	



# Enabling Matrix Security

*Manage Jenkins > Configure Global Security*

The screenshot shows the Jenkins 'Configure Global Security' configuration page. On the left, there is a sidebar with a lock icon and the title 'Configure Global Security'. Below it, the text reads 'Secure Jenkins; define who is allowed to access/use the system.' A red arrow points from this sidebar to the main configuration page on the right.

**Configure Global Security**

Secure Jenkins; define who is allowed to access/use the system.

**Configure Global Security**

Enable security  Disable remember me

**Access Control**

Jenkins' own user database  Delegate to servlet container  
 Allow users to sign up  LDAP  
 Unix user/group database

**Security Realm**

Jenkins' own user database  Delegate to servlet container  
 Allow users to sign up  LDAP  
 Unix user/group database

**Authorization**

Logged-in users can do anything  Anyone can do anything  
 Legacy mode  Allow anonymous read access  
 Matrix-based security  Project-based Matrix Authorization Strategy

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*



---

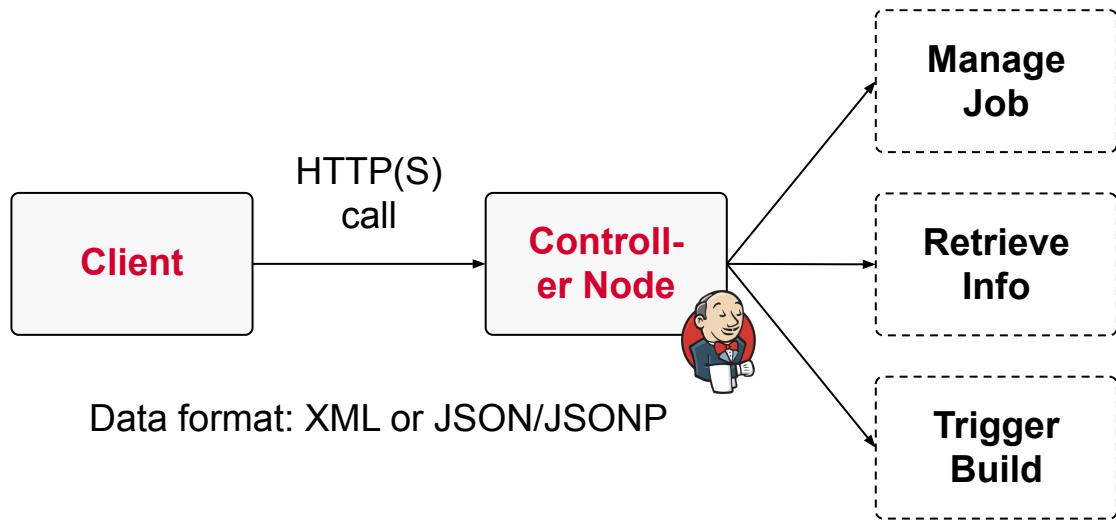
# EXERCISE

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 > Configure Global Security*

**CSRF Protection**

Prevent Cross Site Request Forgery exploits ←

Crumbs

**Crumb Algorithm**

Default Crumb Issuer

Enable proxy compatibility

*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 > Manage Users > Select user*

The screenshot shows the Jenkins 'API Token' creation dialog. On the left, a user card for 'bmuschko' is highlighted with a red arrow pointing to it. The dialog title is 'API Token'. It displays the message 'Current token(s) There are no registered tokens for this user.' Below this are three buttons: 'Default name' (input field), 'Generate' (blue button), and 'Cancel' (red button). A large red arrow points downwards from the 'Generate' button to the second screenshot.

The screenshot shows the Jenkins 'API Token' list. The 'Current token(s)' section displays a single token: 'Token created on 2019-07-10T14:37:53.1 112f8fff3a62211c7db297970de214d1c51'. This token is highlighted with a red box. Below the token, a yellow warning message says '⚠ Copy this token now, because it cannot be recovered in the future.' A red arrow points upwards from this message to the 'Generate' button in the previous screenshot. At the bottom of the list is a 'Add new Token' button.



# 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

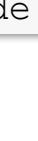
API token



# 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



**Project test**

My awesome job



# Optional: Auth. Token for Job

*Dashboard > Job > Build Triggers*

### Build Triggers

Trigger builds remotely (e.g., from scripts) 



Authentication Token	auth
----------------------	------

Use the following URL to trigger build remotely: JENKINS\_URL/job/test/build?  
token=TOKEN\_NAME or /buildWithParameters?token=TOKEN\_NAME  
Optionally append &cause=Cause+Text to provide text that will be included in the recorded  
build cause.

*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<br>URI("http://localhost:8080/jenkins"), "admin", "password");<br>Map<String, Job> jobs = jenkins.getJobs();<br>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:**

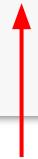
```
$ java -jar jenkins-cli.jar [-s JENKINS_URL] <  
command ...
```



# 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/>



---

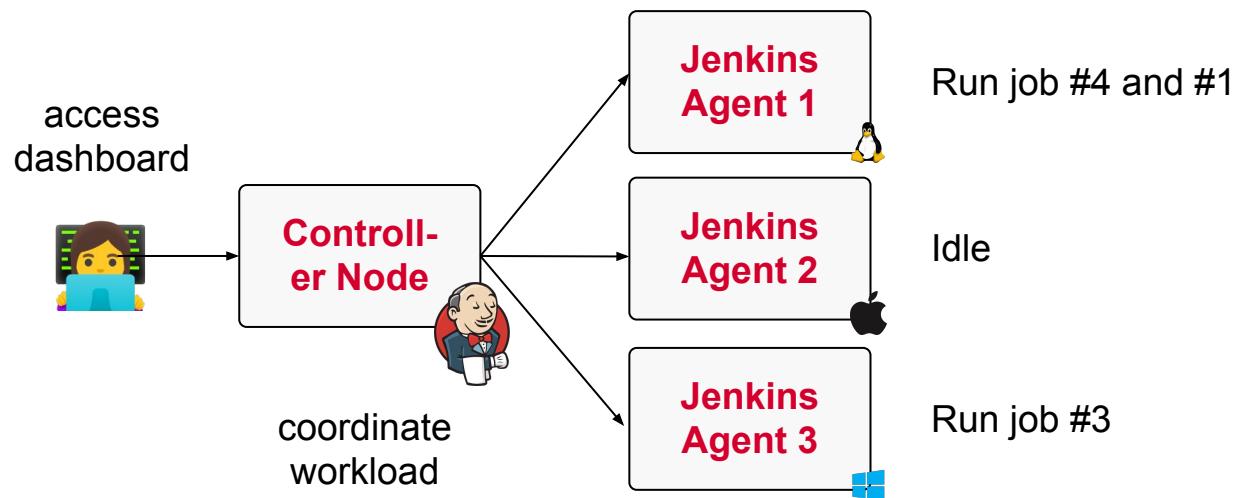
# EXERCISE

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 > Manage Nodes*

 **Manage Nodes**  
Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

↓

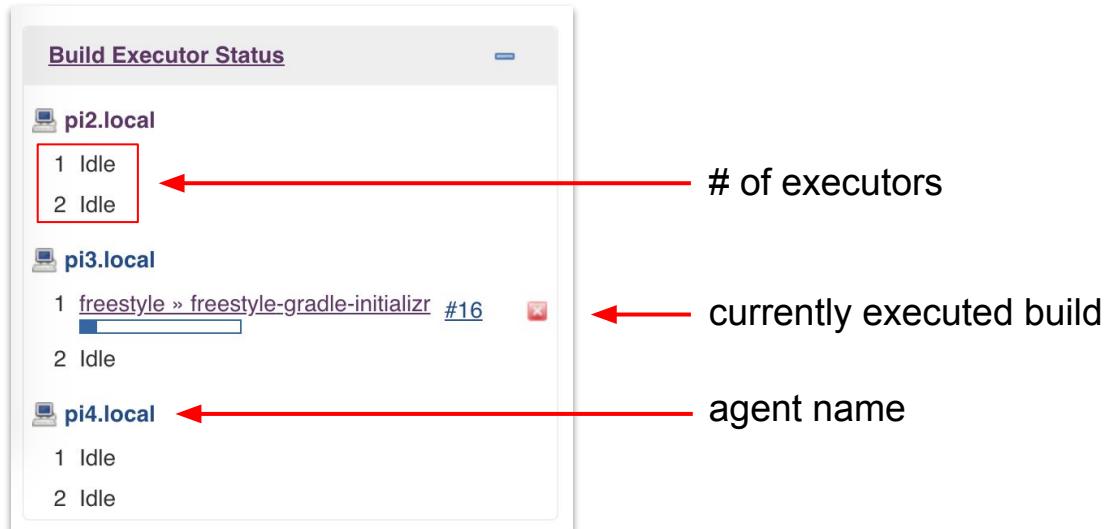
S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time	
	<a href="#">master</a>	Linux (arm)	In sync	11.31 GB	100.00 MB	11.31 GB	0ms	
	<a href="#">pi2.local</a>	Linux (arm)	In sync	10.82 GB	100.00 MB	10.82 GB	299ms	
	<a href="#">pi3.local</a>		N/A	N/A	N/A	N/A	N/A	
	<a href="#">pi4.local</a>	Linux (arm)	In sync	10.90 GB	100.00 MB	10.90 GB	184ms	
Data obtained		78 ms	0.12 sec	0.11 sec	66 ms	0.1 sec	33 ms	

**Refresh status**



# Executors Dashboard Overview

*Maximum builds per executor & what is currently running?*



The screenshot shows the Jenkins Executors Dashboard with three agents listed:

- pi2.local**: 1 Idle, 2 Idle
- pi3.local**: 1 [freestyle » freestyle-gradle-initializr #16](#), 2 Idle
- pi4.local**: 1 Idle, 2 Idle

Annotations with red arrows point to specific elements:

- A red arrow points to the "1 Idle" entry under pi2.local, labeled "# of executors".
- A red arrow points to the build link "#16" under pi3.local, labeled "currently executed build".
- A red arrow points to the "pi3.local" agent name, labeled "agent name".



# Executor Configuration

*Controller node should not execute build workload*

## Controller configuration

# of executors	<input type="text" value="0"/>	
Labels	<input type="text"/>	

## Agent configuration

Name	<input type="text" value="pi2.local"/>	
Description	<input type="text"/>	
# of executors	<input type="text" value="2"/>	



# Job Execution by Label

*Various plugins provide configuration option(s)*

Restrict where this project can be run

Label Expression  

[Label java](#) is serviced by 2 nodes. Permissions or other restrictions provided by plugins may prevent this job from running on those nodes.

[Advanced...](#)



 **java**

[add description](#)

**Nodes**

**Projects**

S	W	Name	Last Success	Last Failure	Last Duration	Fav	
		freestyle - freestyle-gradle-initializer	2 mo 29 days - #13	1 hr 4 min - #16	1 min 57 sec		
		freestyle - freestyle-todo-web-service-exercise	N/A	3 mo 1 day - #11	53 sec		
		my.java-project	1 min 15 sec - #2	N/A	0.26 sec		

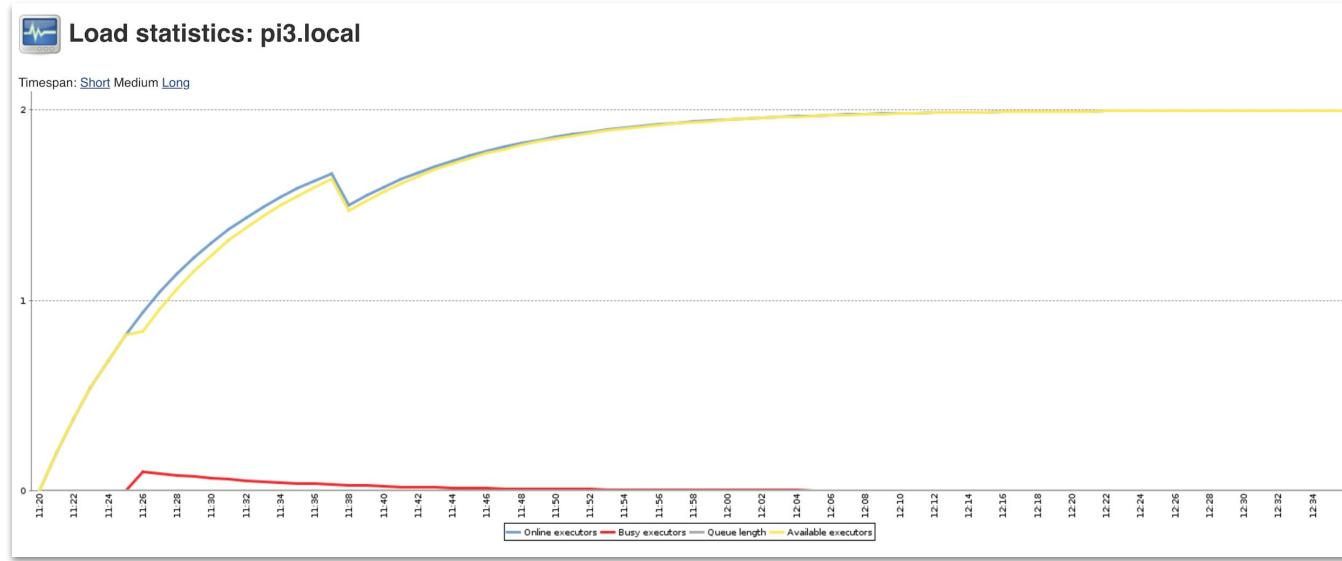
Icon: S M L

Legend: RSS for all RSS for failures RSS for just latest builds



# Node Statistics

*Keeping track of executors uptime, availability etc.*



# Adding New Nodes

*SSH is the most common way to launch an agent*

New Node

Node name

Permanent Agent

Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.

Copy Existing Node

Copy from

Usage

Launch method

Host

Credentials

Host Key Verification Strategy



---

# 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*

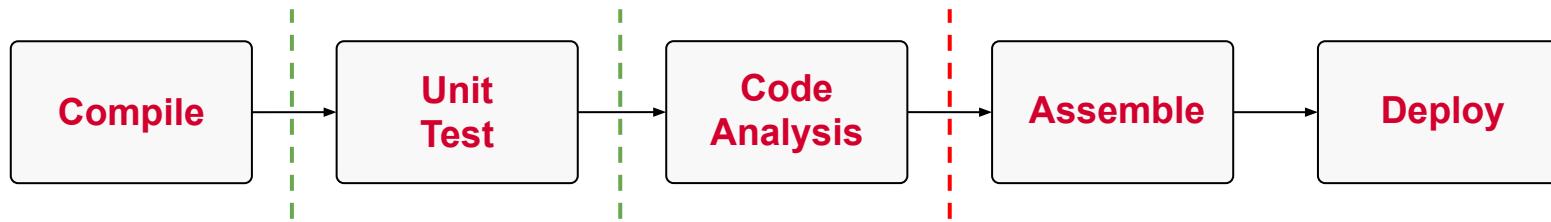


*There's no one-size-fits-all pipeline*



# Pipeline Quality Gates

*Stop the pipeline if expected metrics are not met*



Compile

Unit  
Test

Code  
Analysis

Assemble

Deploy

“Does the code compile  
without issues or warnings?”



“Does the code follow  
code conventions?”

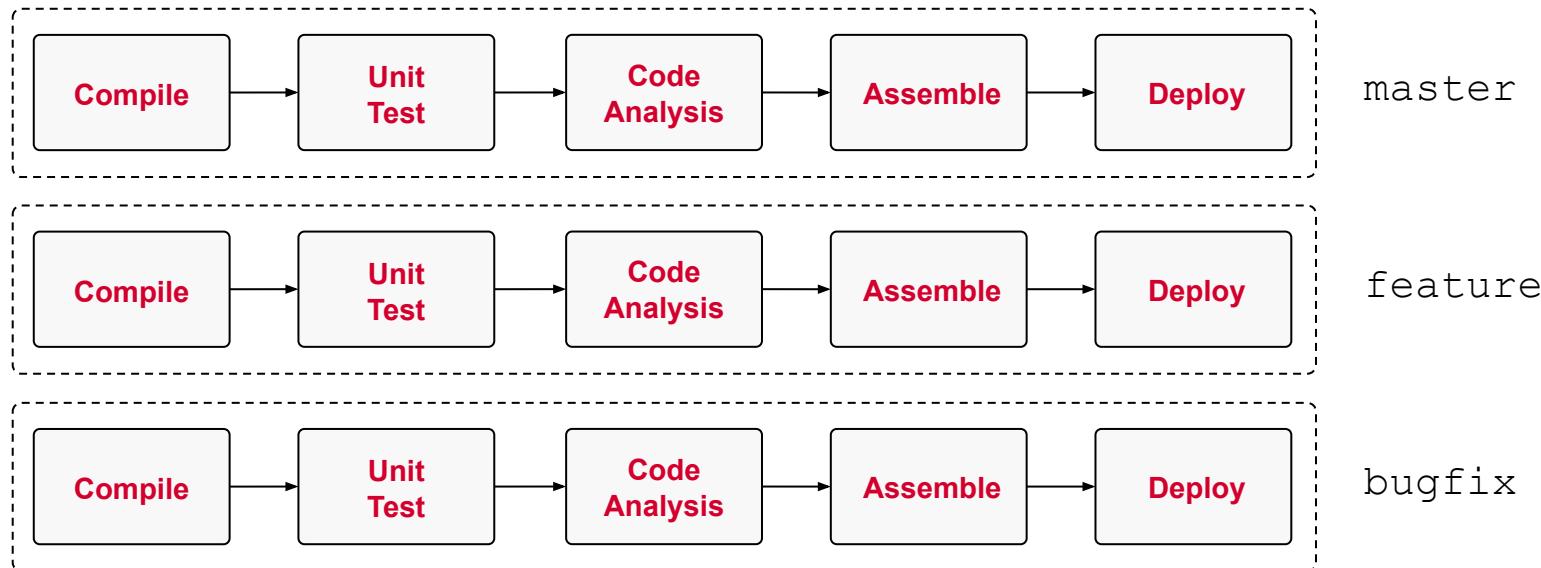


“Do all tests pass and produce  
>= 70% code coverage?”



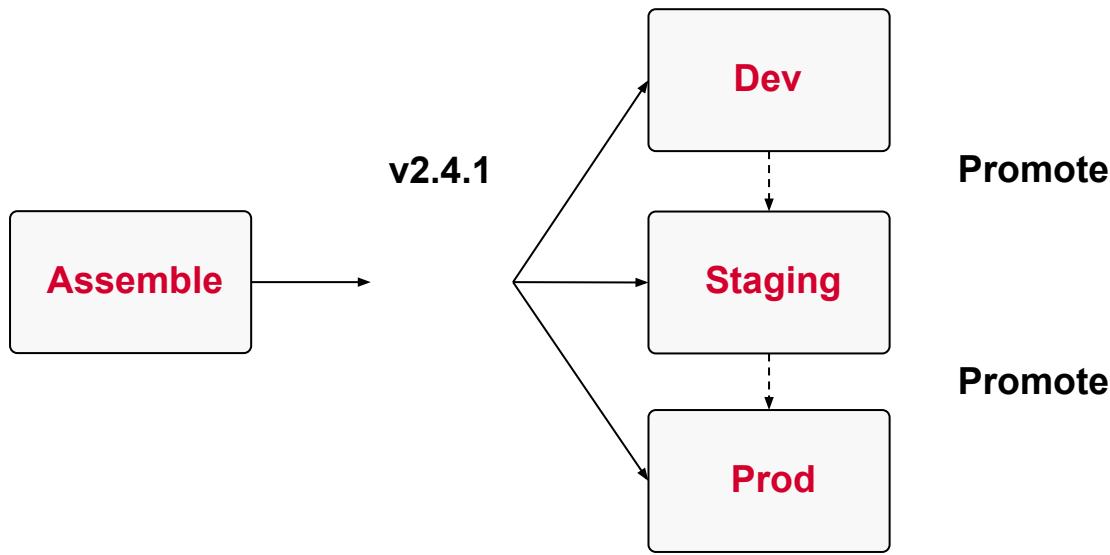
# Same Pipeline for Branches

*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!'  
            }  
        }  
    }  
}
```

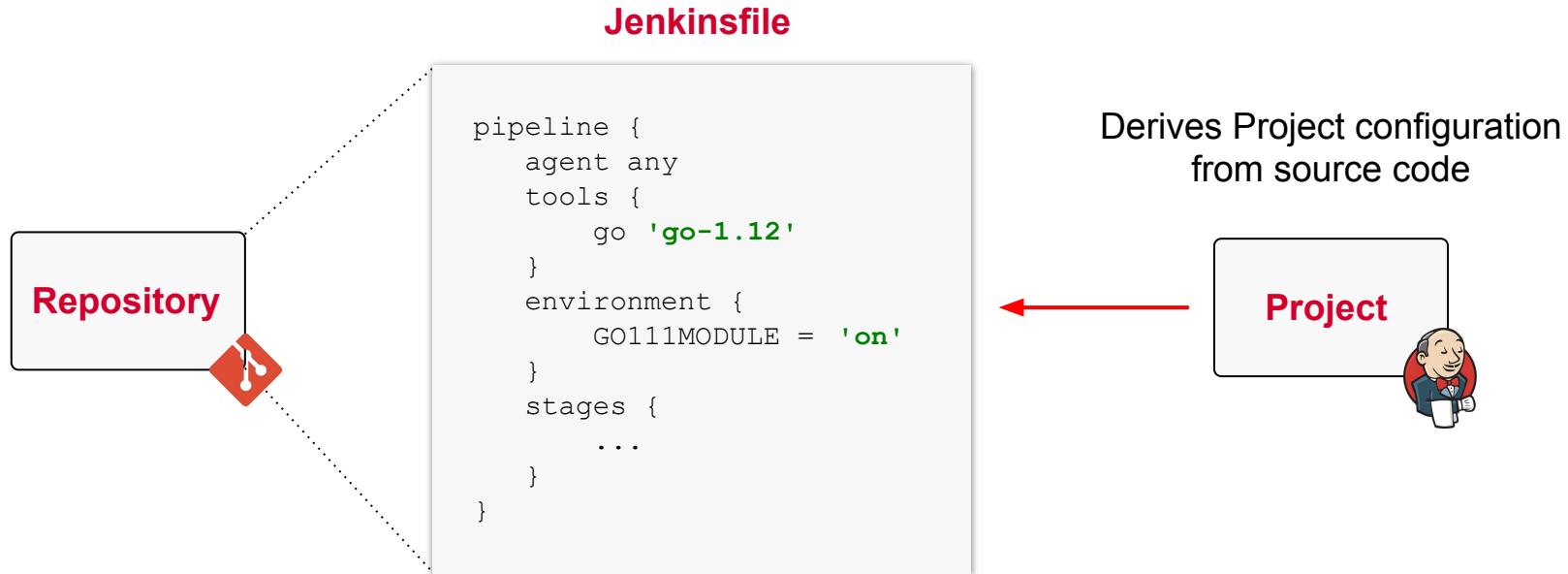
VS.

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



# Infrastructure-as-Code

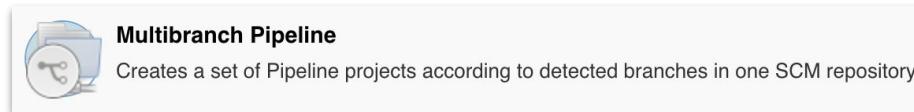
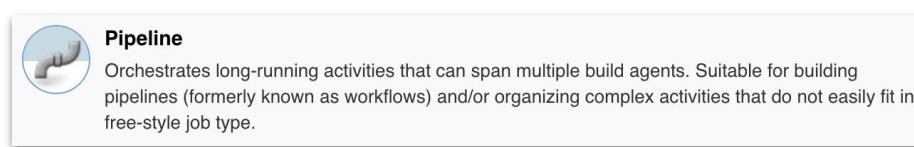
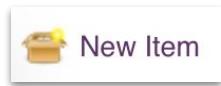
*Version controlled infrastructure configuration*



# Creating a Pipeline Job

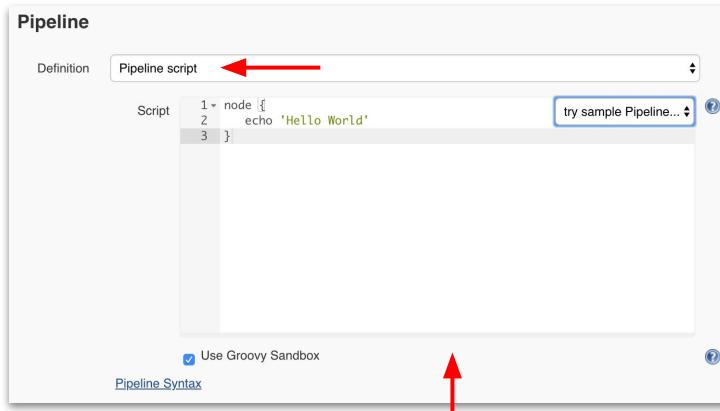
*Pipeline jobs are available if plugins have been installed*

From Dashboard, click...



# Scripted Pipeline Definition

*Scripting is an option but versioned code is preferable*

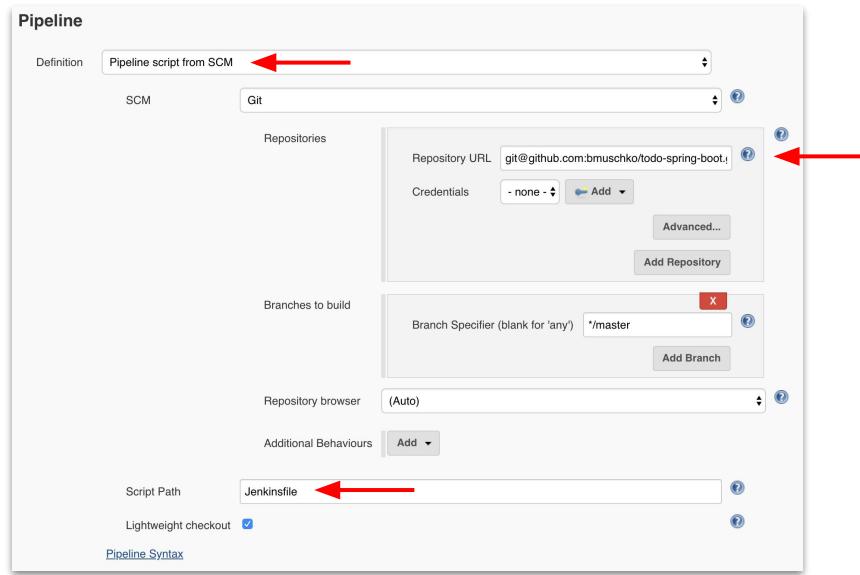


easy to edit and try out



# 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

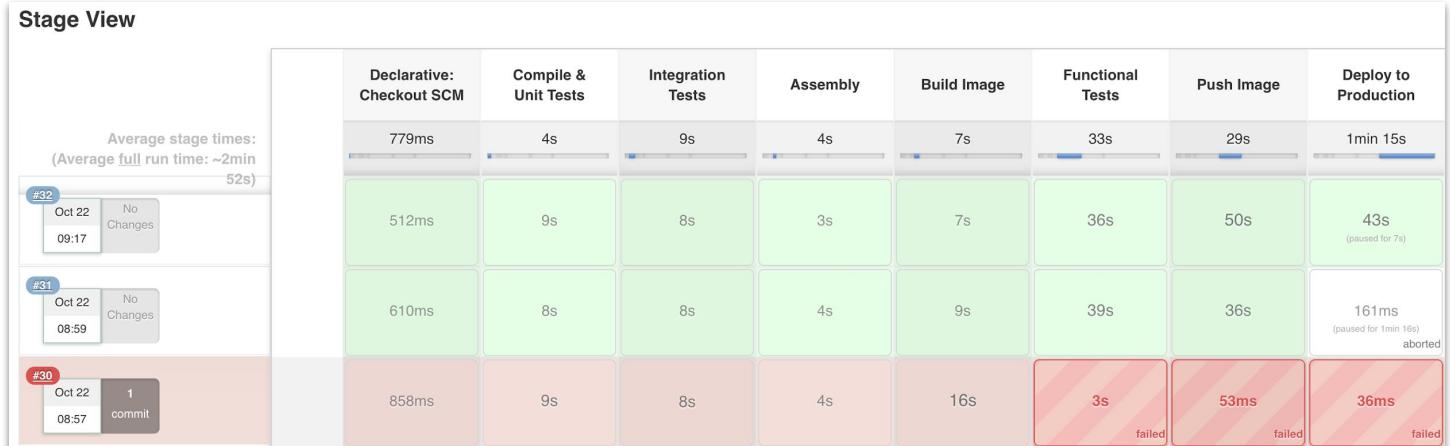


Jenkinsfile language support



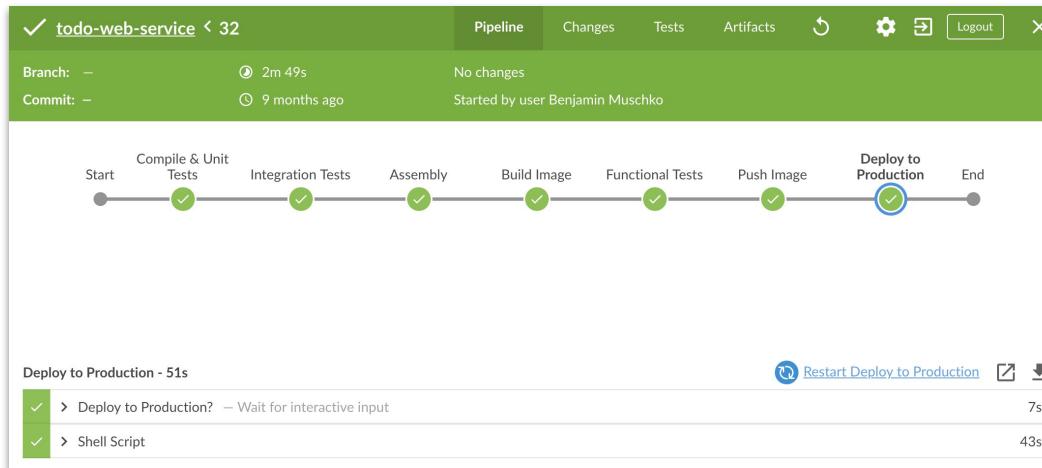
# Standard Pipeline Visualization

*Only renders pipelines linearly*



# Using the BlueOcean Plugin

*Next generation pipeline visualization*

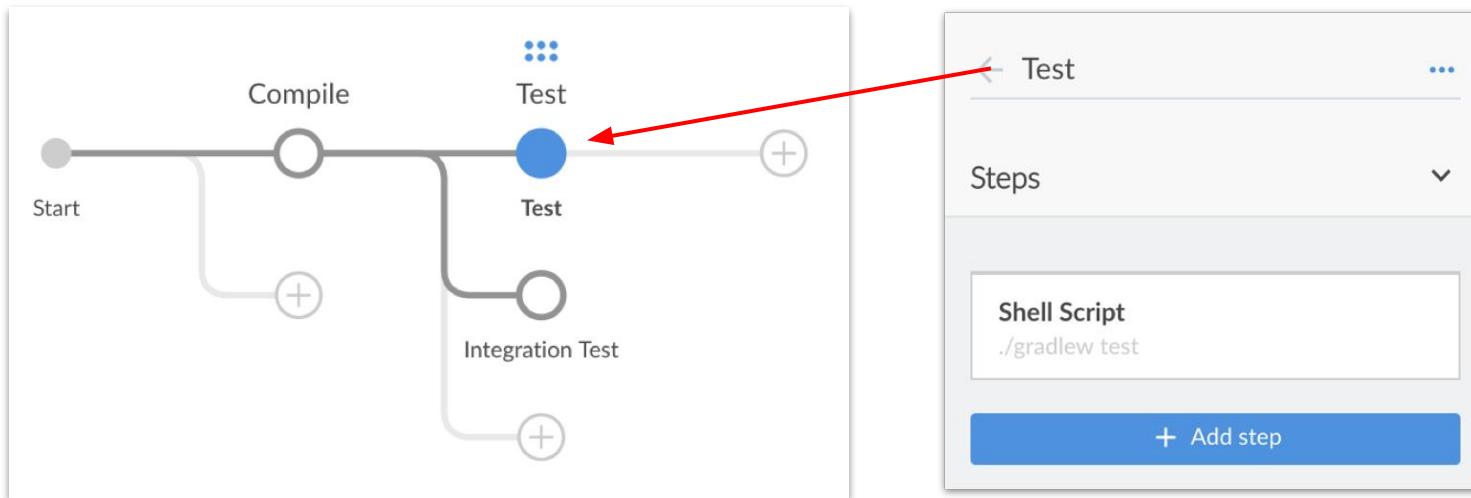


*Plugin is not part of standard installation*



# 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



# The Stages and Stage Sections

*Subset of tasks, visualized in UI*

```
pipeline {  
    agent any  
    stages {  
        ...  
    }  
}
```

```
stage ('Compile') {  
    ...  
}
```

```
stage ('Deploy') {  
    ...  
}
```



---

# 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')  
}
```

*Pushing SCM changes requires falling back to scripted script*



# 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

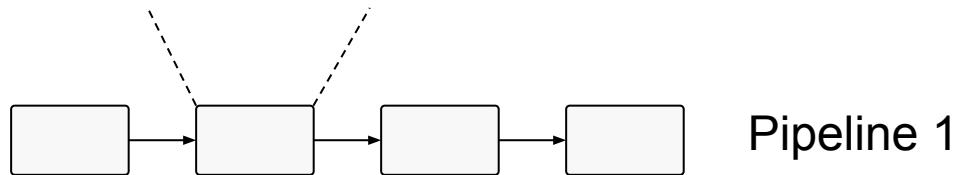
Enhancing a Pipeline  
with Advanced Features



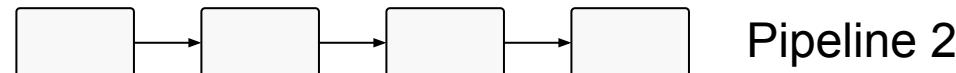
# Keeping Pipelines DRY

*Avoid copy/pasting pipeline code across multiple projects*

**Steps Definition**



Pipeline 1



Pipeline 2

**Steps Definition**



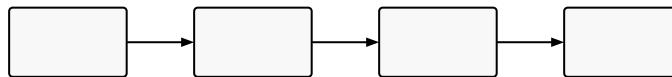
# Pipeline Templates

*You can even define a “standard” pipeline if needed*

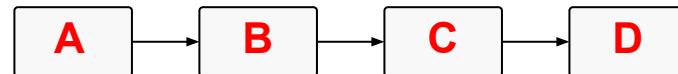
**Inject Dynamic Parameter Values**



```
stageNames = [ 'stage1': 'A', 'stage2': 'B', ... ]
```



Pipeline Template



Generated Pipeline



---

# 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
```

**Build Step**

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

**Unix/Linux/MacOSX:**

```
./gradlew clean install
```

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*

**Global Pipeline Libraries**

Sharable libraries available to any Pipeline jobs running on this system. These libraries will be trusted, meaning they run without "sandbox" restrictions and may use @Grab.

	<b>Library</b>
Name	gradle-shared-lib
Default version	master
Currently maps to revision: 3b549123622c48ece44f48d825499554fee60716	
Load implicitly	<input type="checkbox"/>
Allow default version to be overridden	<input checked="" type="checkbox"/>
Include @Library changes in job recent changes	<input checked="" type="checkbox"/>
<b>Retrieval method</b>	
<input checked="" type="radio"/> Modern SCM	



# 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') ← Branch  
@Library('gradle-shared-lib@2.1') ← Tag  
@Library('gradle-shared-lib@712e341aead532...') ← Commit hash
```

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



---

# EXERCISE

Writing and Using a  
Shared Library



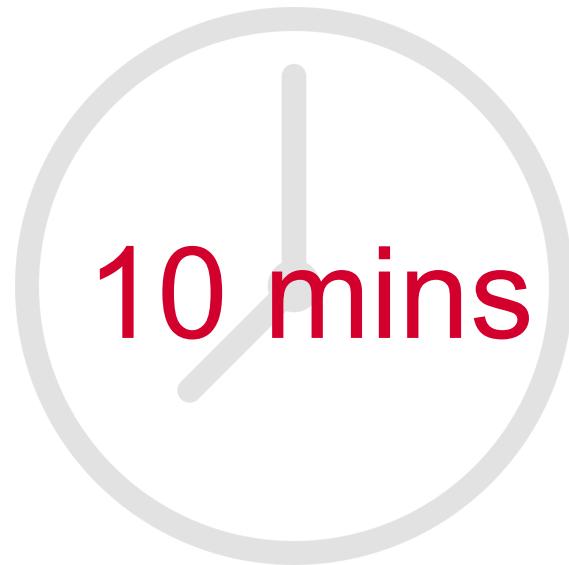
---

# Q & A



---

# BREAK

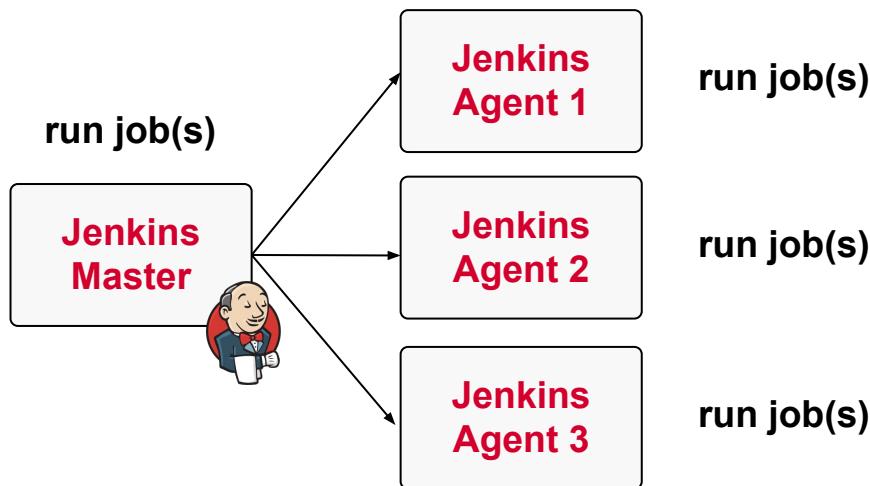


# CD-as-Code Best Practices

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*



# Configuring Master Node

*Set # of executors to zero to avoid running jobs*

## Master configuration

# of executors  ←

Labels

Build Executor Status

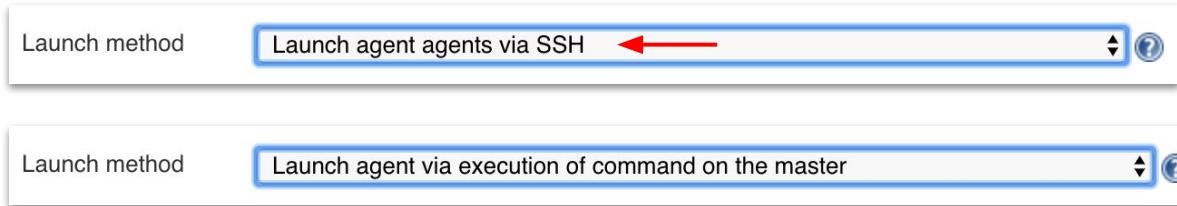
Node	Status	Idle
pi2.local	Idle	1
pi2.local	Idle	2
pi3.local	Idle	1
pi3.local	Running	2
pi4.local	Idle	1
pi4.local	Idle	2

**Master node doesn't even  
show up in executor overview**



# Communication Protocols

*SSH launch method is preferred way*



*JNLP-TCP connector, JNLP-HTTP connector (Java Web Start) are not supported anymore*



---

# Defining Reusable Agents

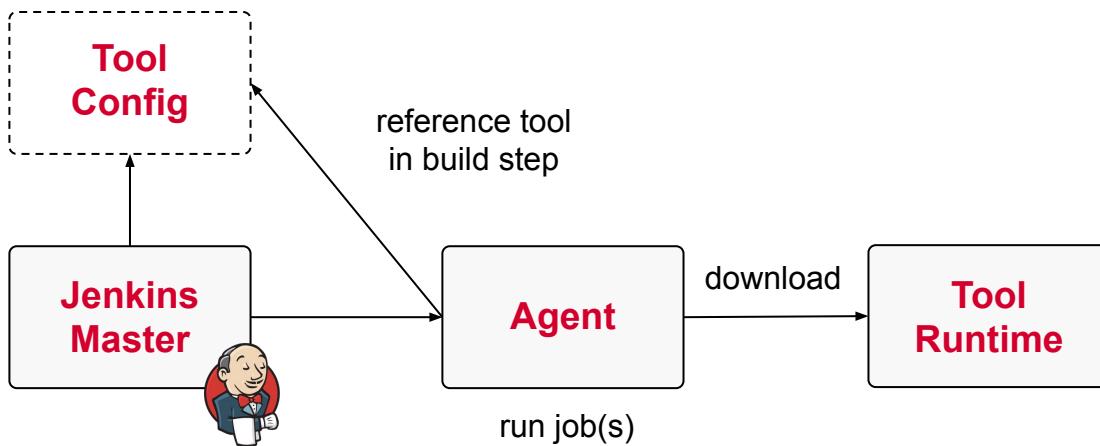
*Generalize as much as possible, specialize if needed*

- 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

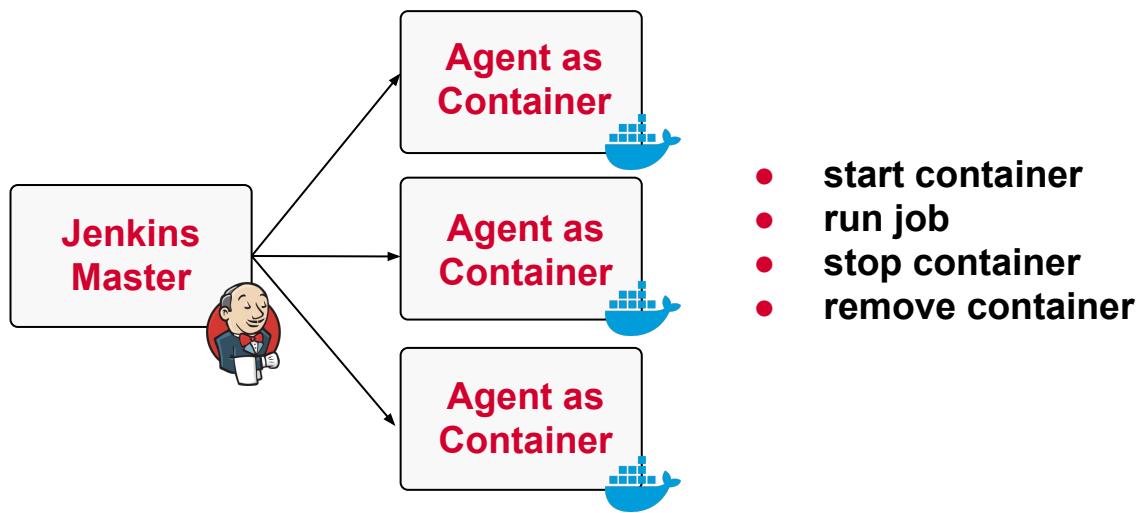
*Offload build execution to on-demand agents*

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



# Running Container Agents

*On-demand, temporary Docker containers to run jobs*



---

# Hardware for Master node

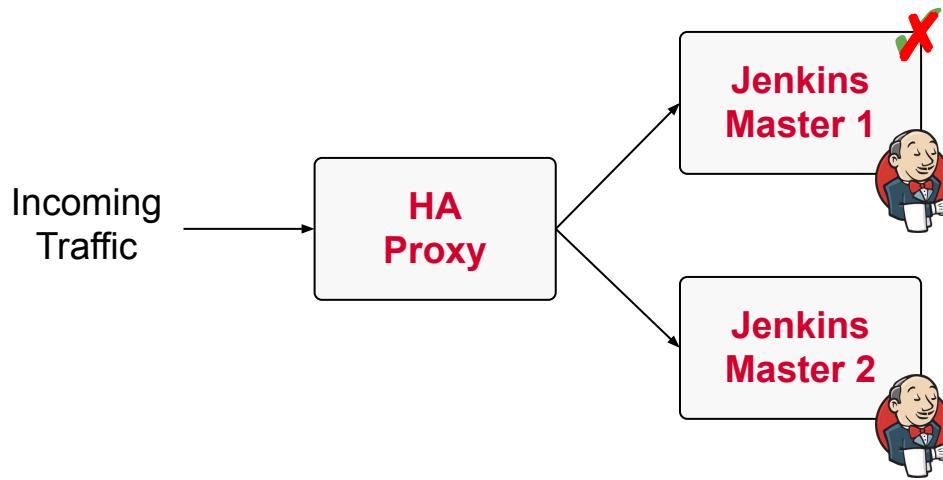
*There's no one-size-fits-all solution*

- **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



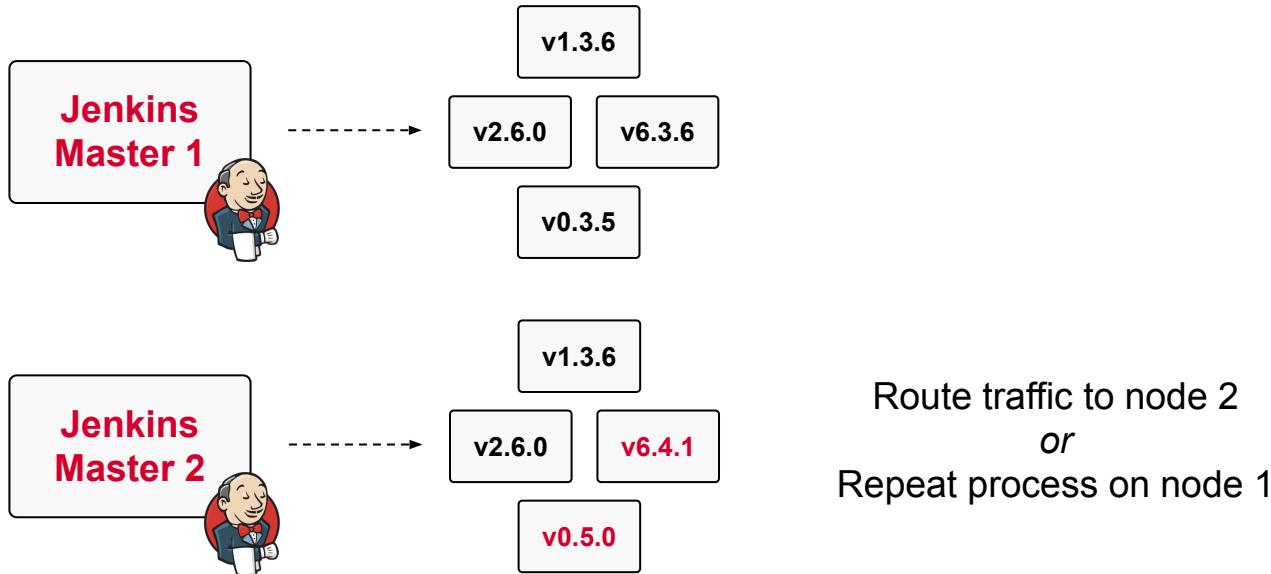
# High Availability for Master node

*High Availability plugin provides failover to backup*



# Testing Plugin Updates

*Incompatible changes can bring down Master node*



---

# 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...



Thank you

