



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

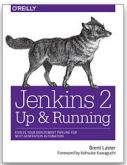
Cross-reference topics for more details



[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

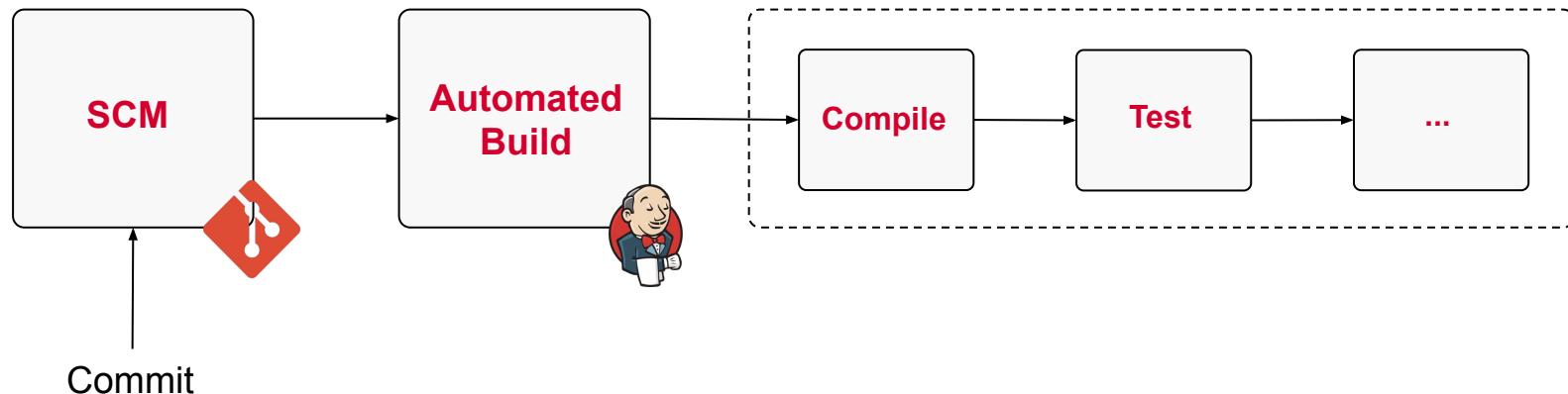


Terminology and Jenkins Installation

Key Concepts, Getting Started

Continuous Integration (CI)

Run an automated build for every commit to SCM



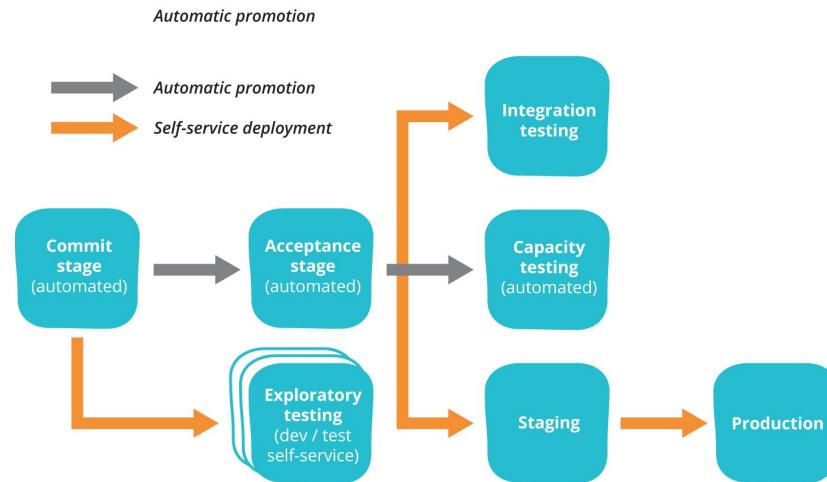
CI Best Practices

- 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/>



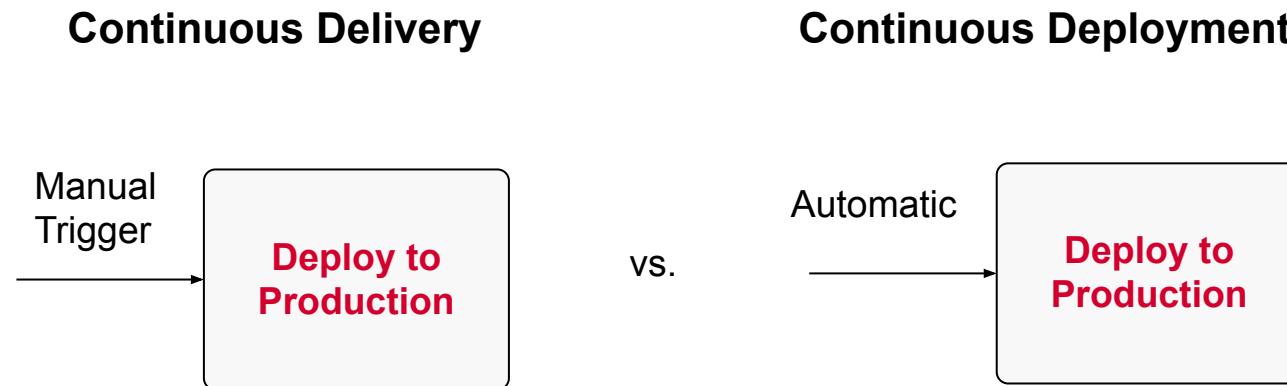
CD Best Practices

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



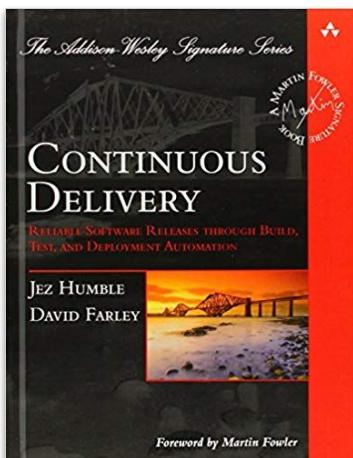
Continuous Deployment (CDE)

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

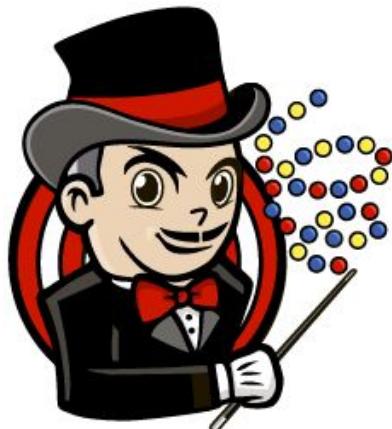


- Installation of JDK 11 or higher
- Set JAVA_HOME as environment variable
- Container image already contains JDK



Installation Wizard

Guides through installation process



Accessible through web browser

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

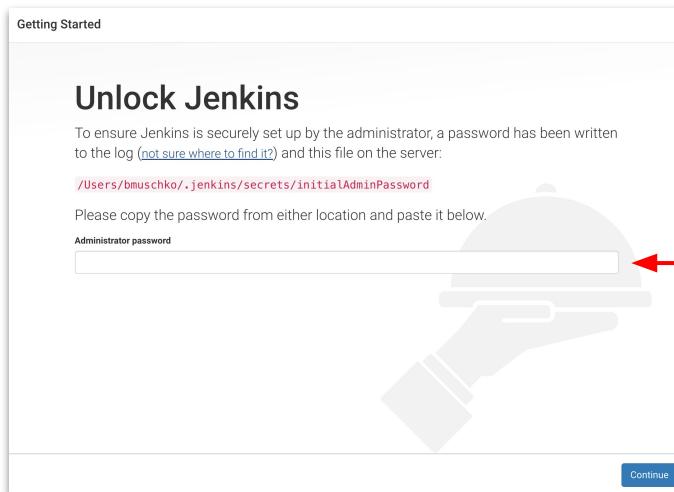
cc creative
commons

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



Unlocking Jenkins

Auto-generated password on local disk

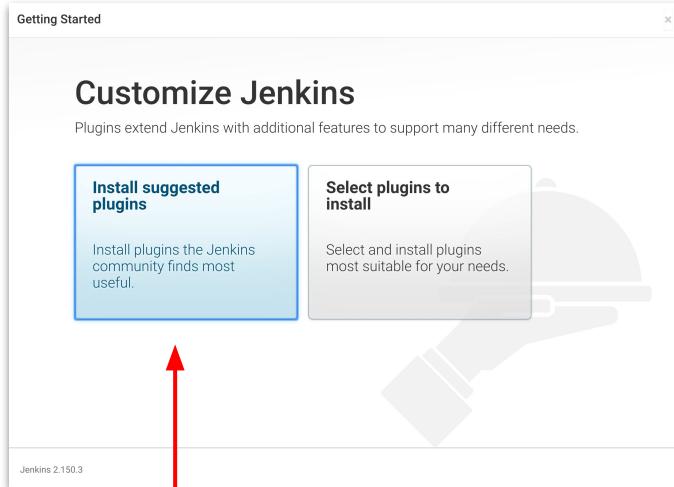


```
$ cat /Users/bmuschko/.jenkins/secrets/initialAdminPassword  
6ce47a19e64045bc87d33c3843f2254b
```



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
** 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

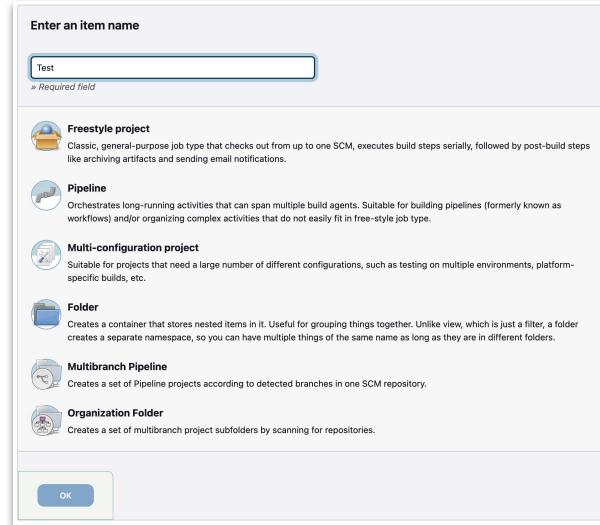
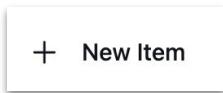
- Compiling a Java-based project
- Executing different types of tests
- Deploying an artifact to a target environment



Different Types of Jobs

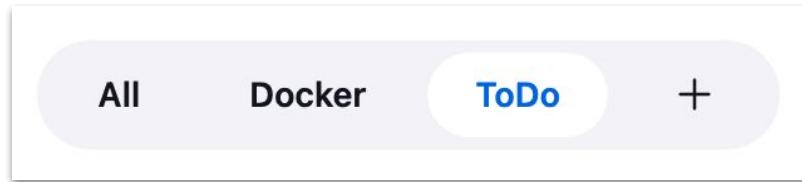
Selected at the time of creation, not changeable later

From Dashboard, click...

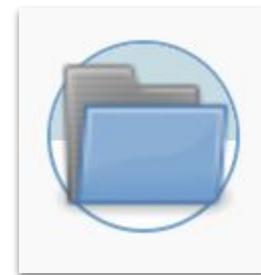


Organizing Jobs

“Grouping of jobs by teams or projects”



VS.



Views

Folders



Using Views

Global and personalized flat lists

Button for creating a new view

The screenshot shows a user interface for managing build jobs. At the top, there are tabs for 'All', 'Docker', 'ToDo', and a '+' button. A red arrow points from the text 'Button for creating a new view' to the '+' button. To the right of the tabs is a 'Add description' link. Below the tabs is a table with columns: S, W, Name (with a sort arrow), Last Success, Last Failure, and Last Duration. The table contains three rows of data:

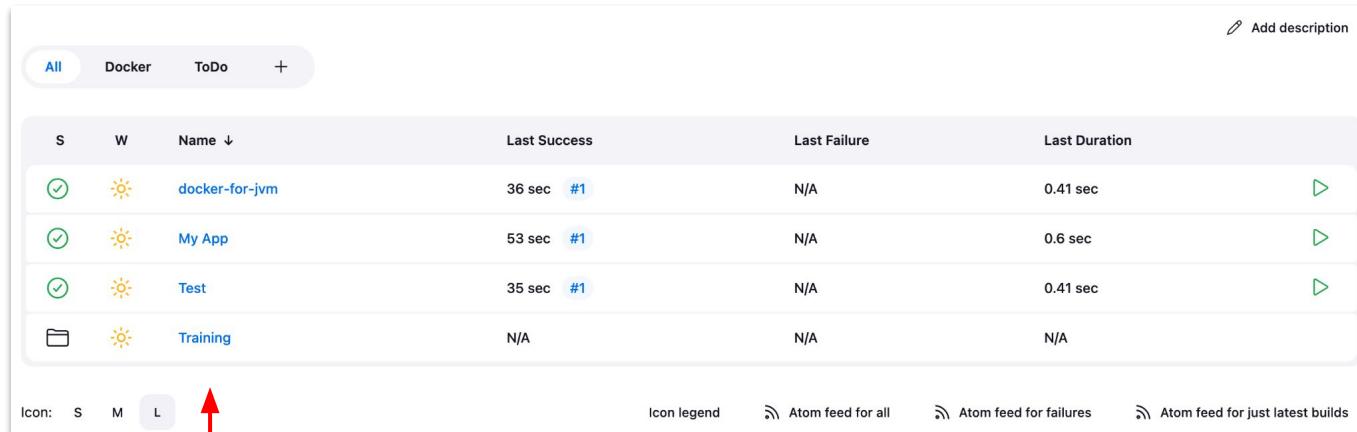
S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	☀️	docker-for-jvm	4 min 7 sec #1	N/A	0.41 sec
✓	☀️	My App	4 min 24 sec #1	N/A	0.6 sec
✓	☀️	Test	4 min 6 sec #1	N/A	0.41 sec

At the bottom, there are icons for 'Icon: S M L', 'Icon legend', and three 'Atom feed' links: 'Atom feed for all', 'Atom feed for failures', and 'Atom feed for just latest builds'. A large red 'O' logo is in the bottom right corner.



Using Folders

Nested organizational structure



The screenshot shows a web-based CI/CD pipeline interface. At the top, there are tabs: All (selected), Docker, ToDo, and a plus sign. Below the tabs is a table with columns: S, W, Name (sorted by name), Last Success, Last Failure, and Last Duration. The table contains four rows:

S	W	Name ↓	Last Success	Last Failure	Last Duration
✓	☀️	docker-for-jvm	36 sec #1	N/A	0.41 sec
✓	☀️	My App	53 sec #1	N/A	0.6 sec
✓	☀️	Test	35 sec #1	N/A	0.41 sec

Below the table, there is a section for icons: S, M, L. An arrow points from this section down to a detailed description of the 'Folder' icon. At the bottom right of the main interface, there are links for Atom feed for all, Atom feed for failures, and Atom feed for just latest builds.

+ New Item



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



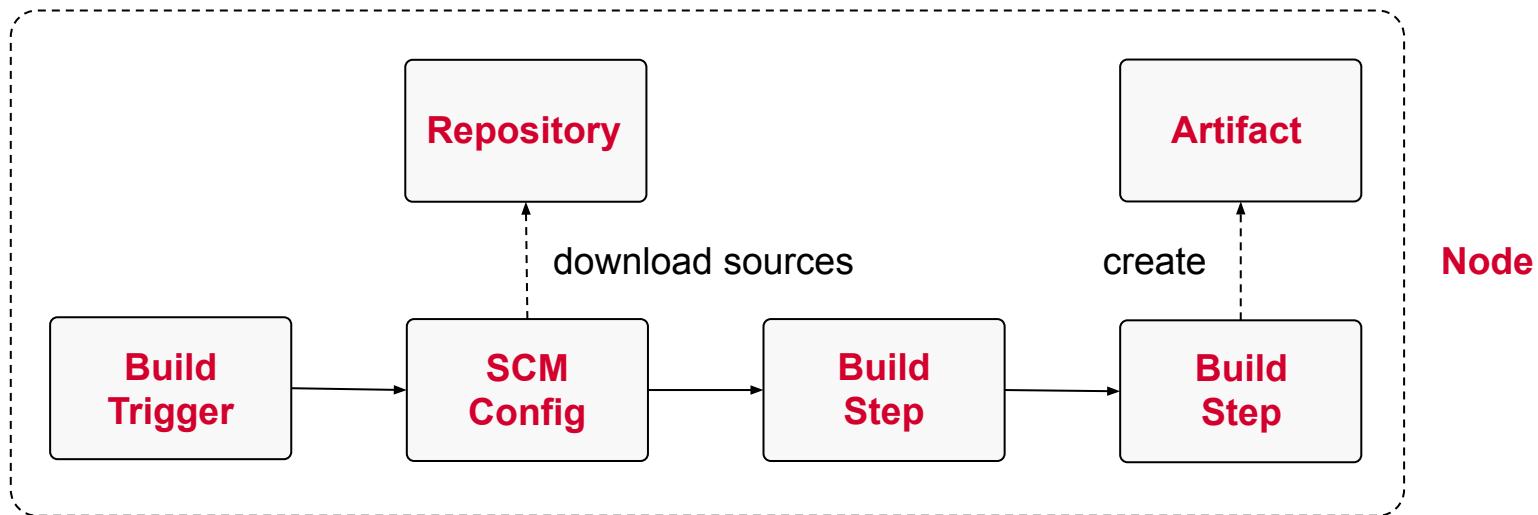
EXERCISE

Defining, Configuring
and Organizing a Job



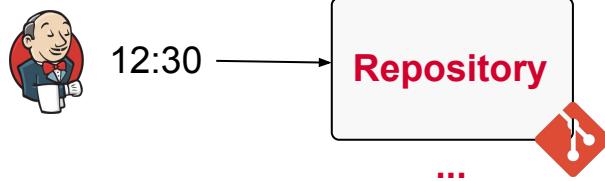
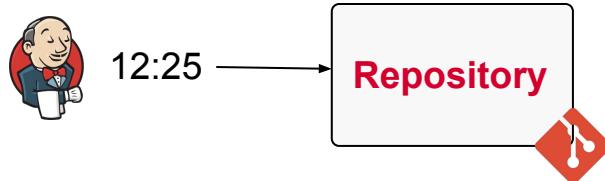
What is a Build?

Execution of an automation task



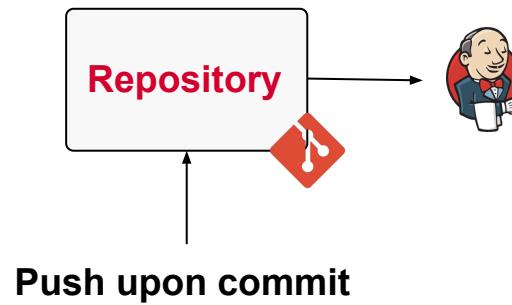
Defining Build Triggers

Periodic checking or direct notification upon a change



Pull every 5 mins

VS.

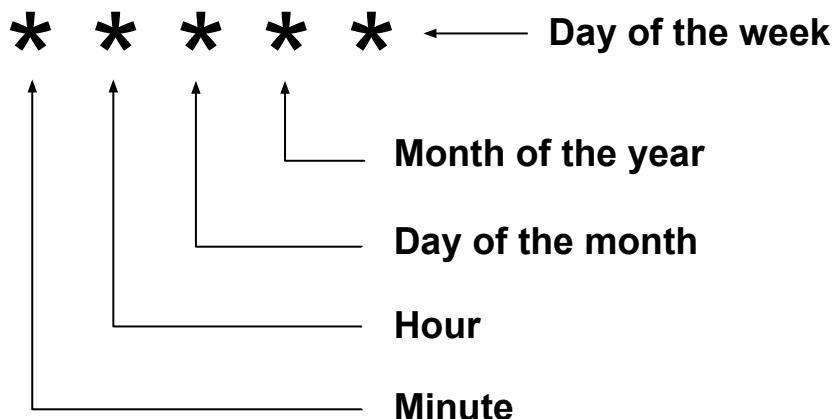


Push upon commit



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

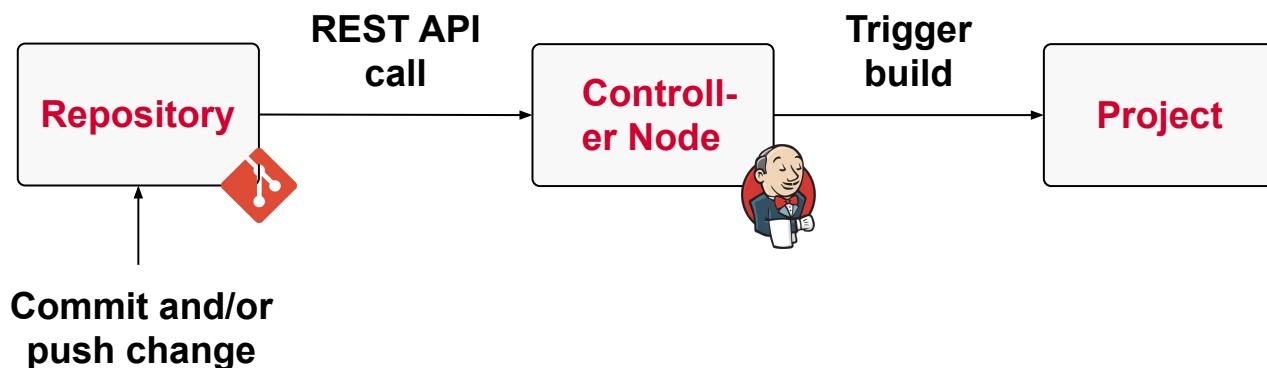
Configure > Build Triggers > Build Periodically

The screenshot shows a configuration interface for a build trigger. At the top left is a checked checkbox labeled "Build periodically". To its right is a question mark icon. A red arrow points from the text above to this checkbox. Below the checkbox is a section titled "Schedule" with a question mark icon. A second red arrow points from the text above to this section. The "Schedule" input field contains the cron expression "H/5 * * * *". At the bottom of the window, a message states: "Would last have run at Wednesday, March 20, 2024, 7:30:09 AM Mountain Daylight Time; would next run at Wednesday, March 20, 2024, 7:35:09 AM Mountain Daylight Time."



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  

When Jenkins receives a GitHub push hook, GitHub Plugin checks to see whether the hook came from a GitHub repository which matches the Git repository defined in SCM/Git section of this job. If they match and this option is enabled, GitHub Plugin triggers a one-time polling on GITScm. When GITScm polls GitHub, it finds that there is a change and initiates a build. The last sentence describes the behavior of Git plugin, thus the polling and initiating the build is not a part of GitHub plugin.

(from [GitHub plugin](#))



GitHub Webhook Configuration

Settings > Webhooks > Add webhook

Webhooks / Add webhook

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our [developer documentation](#).

Payload URL *

←

Content type

→

Secret

SSL verification

By default, we verify SSL certificates when delivering payloads.

Enable SSL verification Disable (not recommended) ←

Which events would you like to trigger this webhook?

Just the push event.
 Send me everything.
 Let me select individual events.

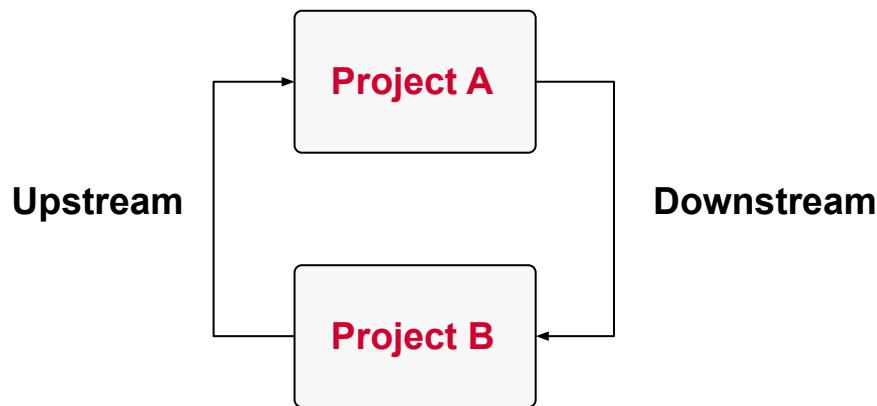
Active
We will deliver event details when this hook is triggered.

Add webhook



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 interface for build triggers. At the top left, there is a checked checkbox labeled "Build after other projects are built" with a question mark icon. A red arrow points to the right of this checkbox. Below it, there is a section titled "Projects to watch" containing a single entry: "My App". Underneath this, there is a list of trigger options, each preceded by a radio button. The first option, "Trigger only if build is stable", has a red arrow pointing to its radio button, indicating it is the selected option. The other three options are: "Trigger even if the build is unstable", "Trigger even if the build fails", and "Always trigger, even if the build is aborted".

Build after other projects are built ?

Projects to watch

My App

Trigger only if build is stable

Trigger even if the build is unstable

Trigger even if the build fails

Always trigger, even if the build is aborted



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 diagram illustrates the configuration of build steps in a Continuous Integration pipeline. It shows three main components:

- Build Steps Selection:** A sidebar titled "Build Steps" with a "Add build step" button. A red arrow points from this button to the "Execute shell" option in the list below, which is highlighted in blue.
- Execute shell Configuration:** A detailed configuration panel for the "Execute shell" step. It includes a "Command" section with the text "echo \"Hello World!\"", an "Advanced" dropdown, and a red "X" button in the top right corner.
- Invoke top-level Maven targets Configuration:** A detailed configuration panel for the "Invoke top-level Maven targets" step. It includes a "Goals" section with the text "clean install", an "Advanced" dropdown, and a red "X" button in the top right corner.

Red arrows indicate the flow from the "Add build step" button to the "Execute shell" option, and from the "Execute shell" configuration panel to the "Invoke top-level Maven targets" configuration panel, suggesting a sequence or dependency in the pipeline setup.



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:

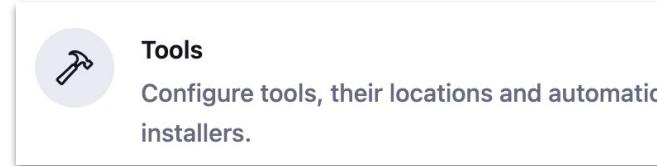
```
echo "Current build number: $BUILD_NUMBER"
```



Configuring Tools

Manage Jenkins > Tools

The screenshot shows the Jenkins 'Tools' configuration page. Under the 'Maven' section, there is a 'Name' field containing 'Maven 3.9.6' with a red arrow pointing to it. Below it is a checked checkbox for 'Install automatically'. A sub-section titled 'Install from Apache' shows a 'Version' dropdown set to '3.9.6'. At the bottom left is a 'Add Installer' button.



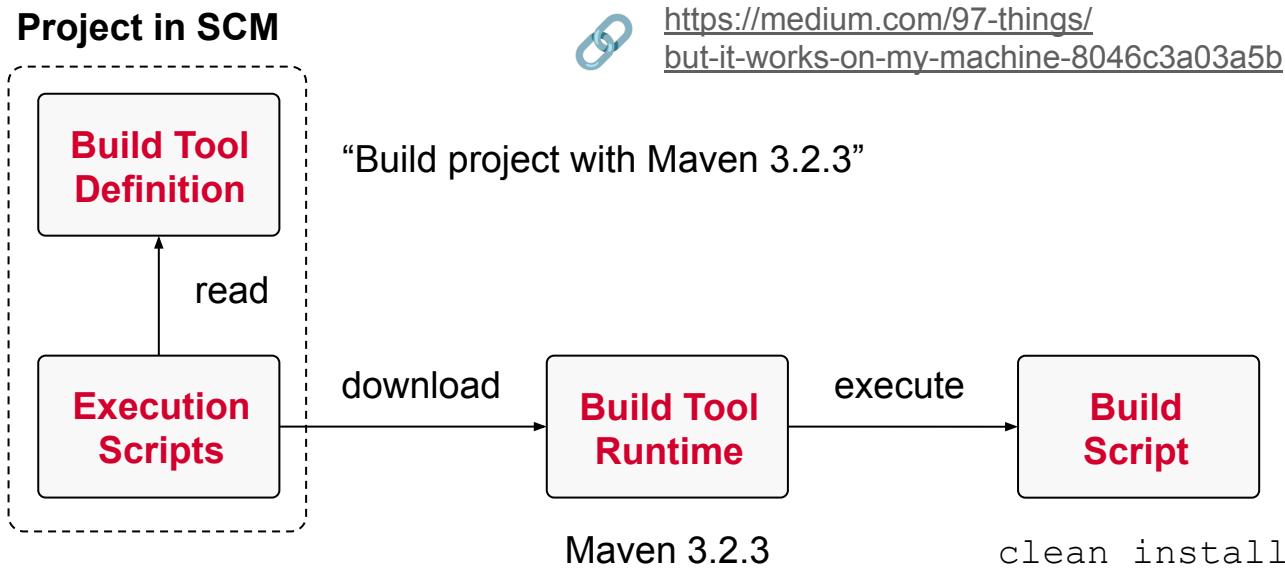
The screenshot shows the Jenkins 'Build Steps' configuration page. Under the 'Invoke top-level Maven targets' section, there is a 'Maven Version' dropdown containing 'Maven 3.9.6' with a red arrow pointing to it. Below it is a 'Goals' dropdown containing 'clean install'. At the bottom left is an 'Advanced' button.

Auto-install tool and make it available on Jenkins nodes



Using a Build Tool Wrapper

Project-specific definition of build tool runtime



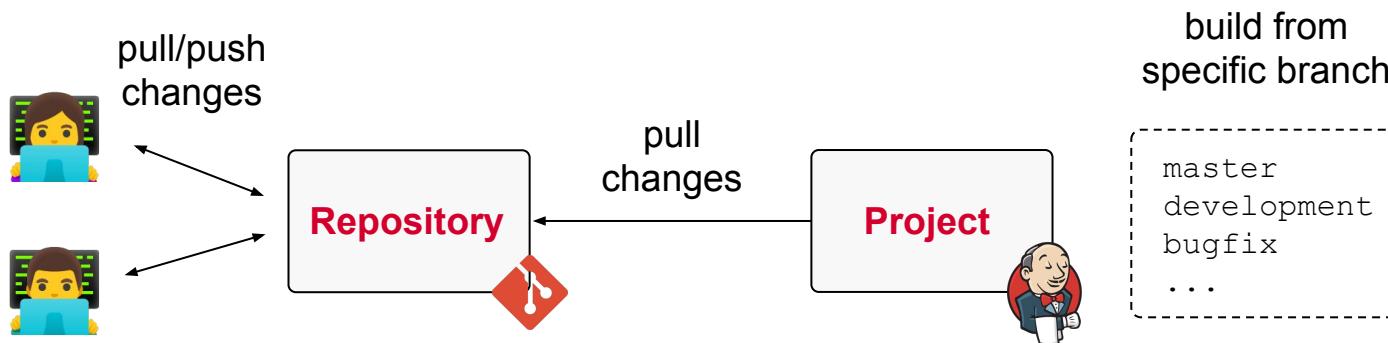
EXERCISE

Configuring Build
Triggers and Steps for
a Job



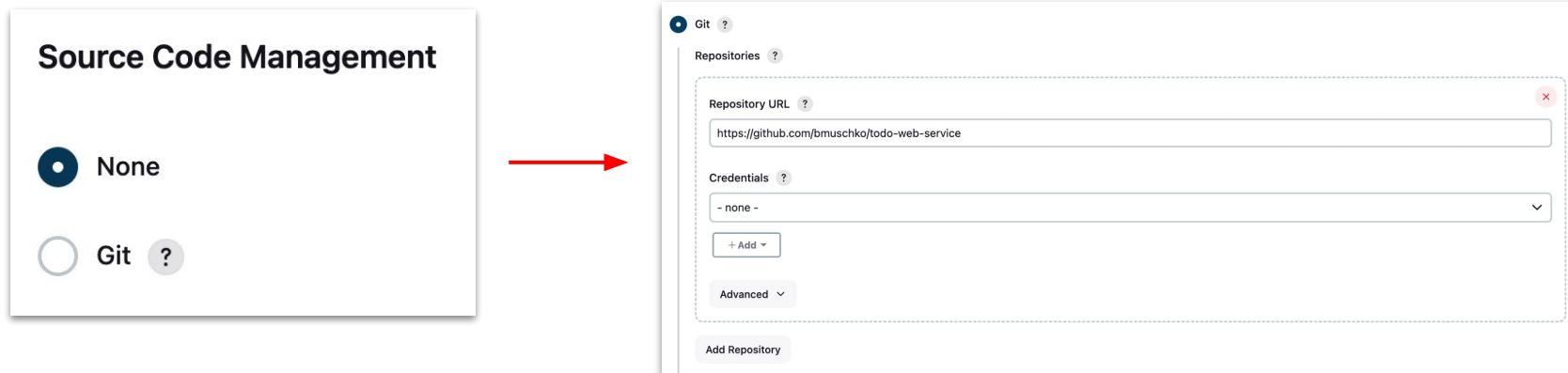
What's the Purpose of a SCM?

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



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



Changes

Summary

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

Commit [912762e8a856ffe7f4bbc5a5b3b5dbc1255f1732](#) by [Benjamin Muschko](#)

Add release step

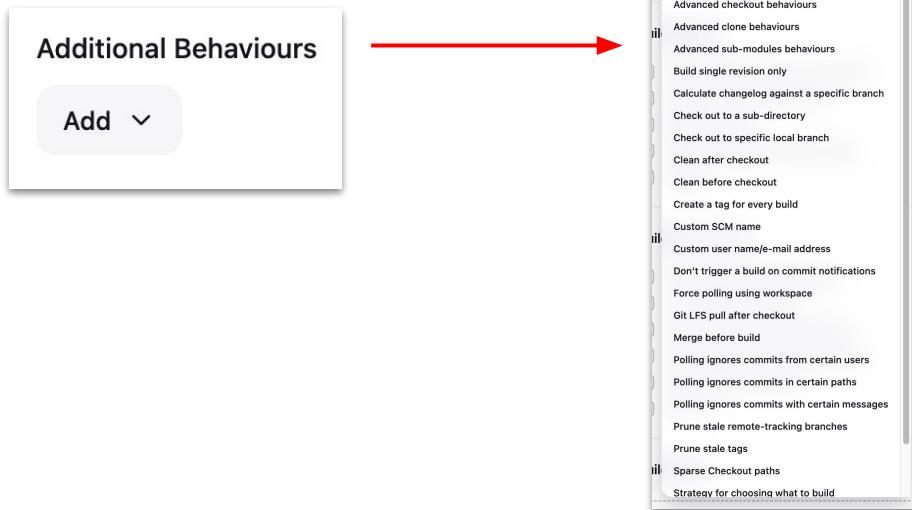
 [Jenkinsfile \(diff\)](#)



Incremental & Clean Check Out

Trade off: Checkout speed vs. clean workspace

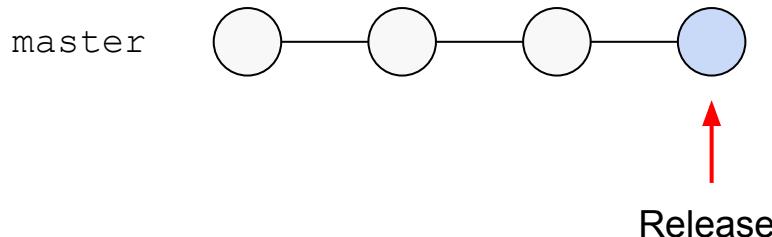
From SCM configuration, click...



Branch and Merge Strategies

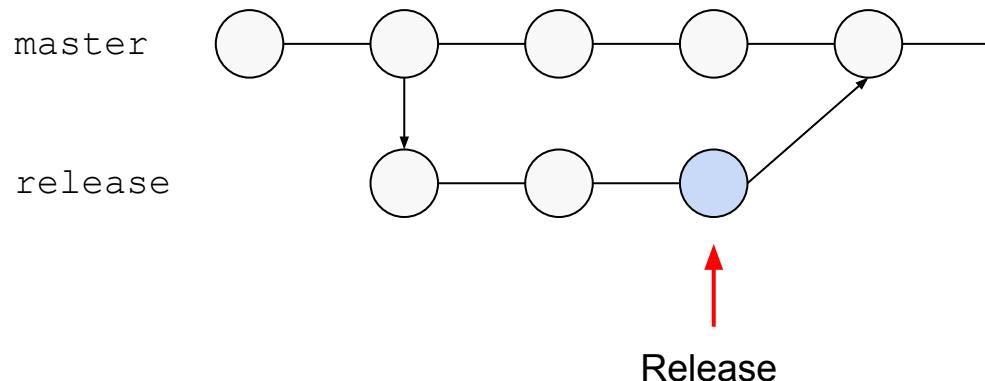
Pick the most suitable for your team, workflow & culture

Only commit to master branch, release from master



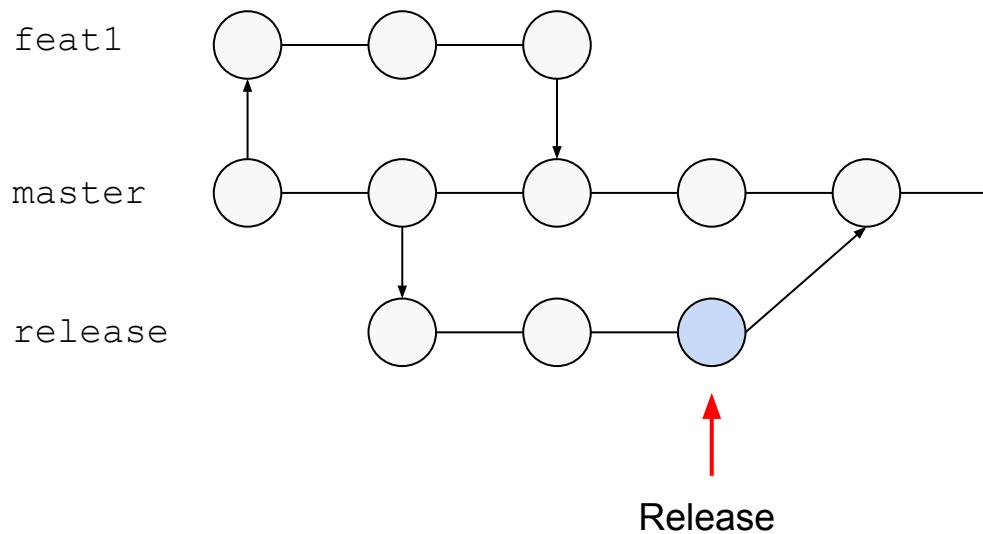
Release Branching

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

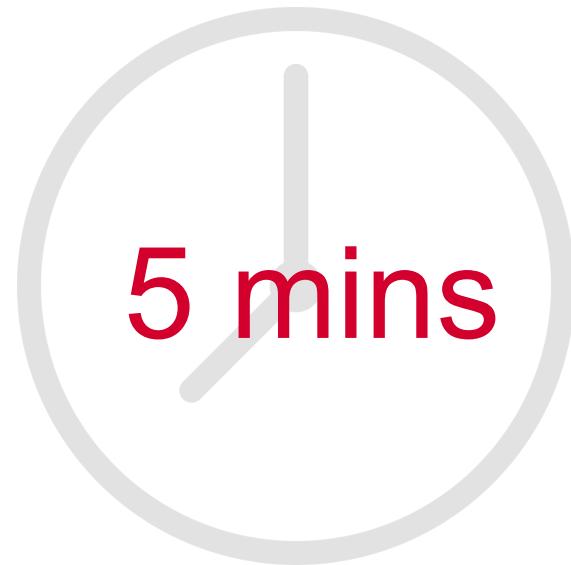
Configuring a GitHub
SCM



Q & A



BREAK



Testing, Notifications and Artifacts

Definition, Usage and Options

Importance of Testing

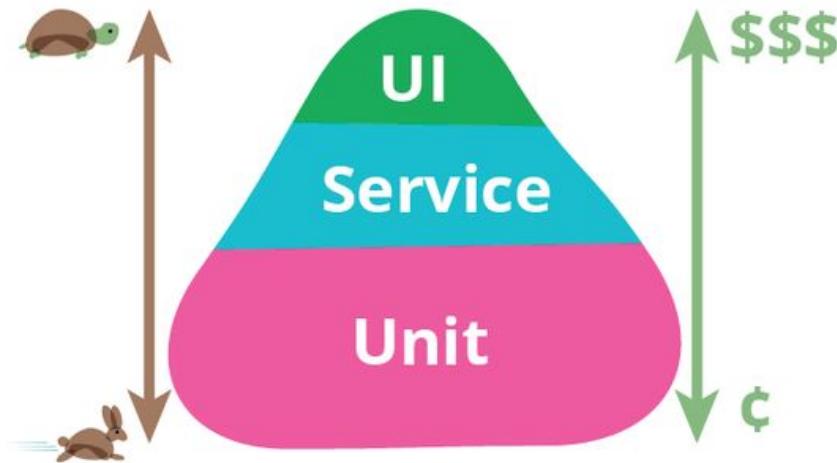
Deliver product with acceptable quality

- 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

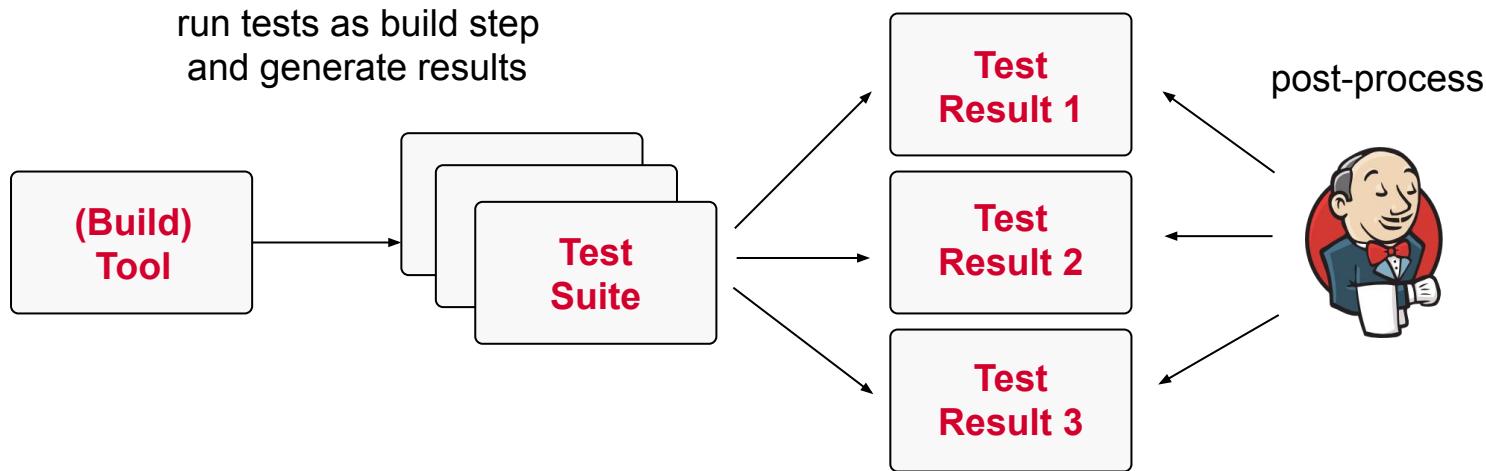


Source: <https://martinfowler.com/bliki/TestPyramid.html>



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...*

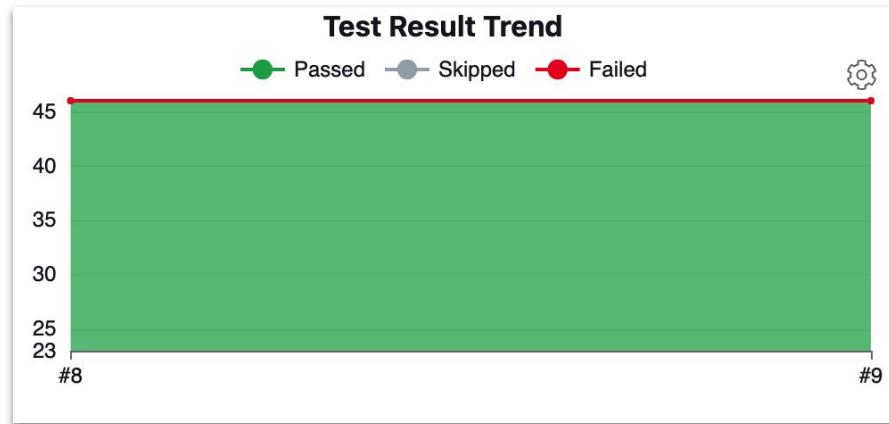
The image shows the Jenkins configuration interface. On the left, a sidebar lists various post-build actions. The 'Publish JUnit test result report' option is highlighted with a red box and a red arrow points from it to the main configuration window on the right. The main window is titled 'Publish JUnit test result report'. It contains several configuration fields:

- Test report XMLs:** A text input field containing the value "build/test-results/**/TEST-*.xml".
- Test output retention:** A dropdown menu set to "None".
- keep all the properties:** A checkbox that is unchecked.
- Health report amplification factor:** A text input field containing the value "1.0". Below it, a note states: "1% failing tests scores as 99% health, 5% failing tests scores as 95% health".
- Allow empty results:** A checkbox that is unchecked.
- Do not fail the build on empty test results:** A sub-option under 'Allow empty results'.
- Skip publishing checks:** A checkbox that is unchecked.
- If unchecked, then issues will be published to SCM provider platforms:** A sub-option under 'Skip publishing checks'.



Test Trending & Reporting

“Give me a historical and detailed test overview”



Latest Test Result (no failures)

Test Result
0 failures (x0) 46 tests (x0)
Took 16 sec. Add description

All Tests

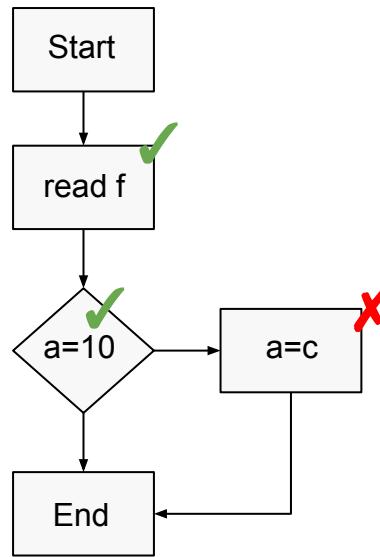
Package	Duration	Fail	(diff)	Skip	(diff)	Pass	(diff)	Total	(diff)
com.bmuschko.gradle.initializr.archive	0.18 sec	0		0		2		2	
com.bmuschko.gradle.initializr.generator	15 sec	0		0		28		28	
com.bmuschko.gradle.initializr.metadata	0.98 sec	0		0		2		2	
com.bmuschko.gradle.initializr.model	6 ms	0		0		8		8	
com.bmuschko.gradle.initializr.web	0.29 sec	0		0		6		6	



Code Coverage Metrics

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

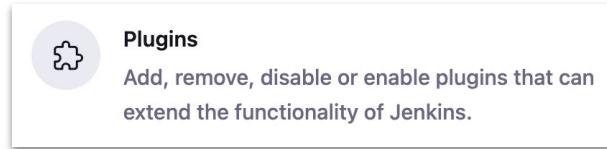
- Statement/line coverage
- Branch coverage
- Condition coverage



Code Coverage Tools

Tooling choice depends on your programming language

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



Processing JaCoCo Metrics

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

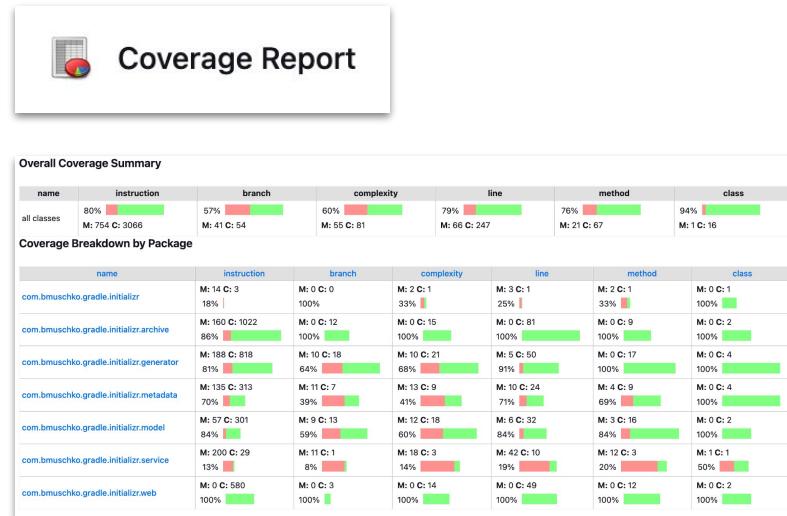
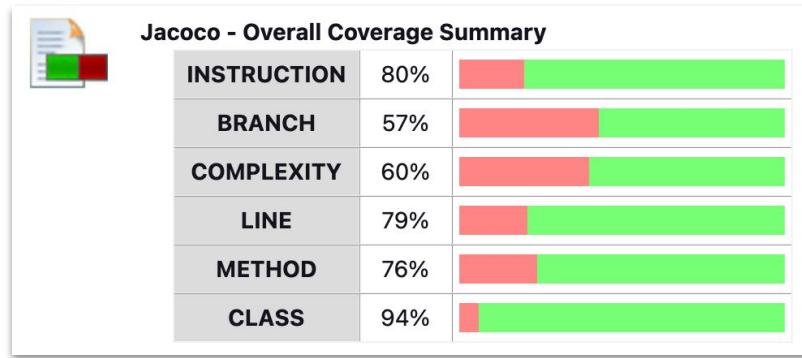
The screenshot shows the Jenkins configuration interface for a job. On the left, a sidebar lists various post-build actions. The 'Record JaCoCo coverage report' option is highlighted with a red box and has a red arrow pointing to it from the left. To the right, a detailed configuration dialog for 'Record JaCoCo coverage report' is open. It contains fields for 'Path to exec files' (set to '**/*.exec), 'Path to class directories' (set to '**/classes), and 'Path to source directories' (set to '**/src/main/java). There are also sections for 'Inclusions' and 'Exclusions'. At the bottom, there are three checkboxes: 'Disable display of source files for coverage', 'Change build status according to the defined thresholds', and 'Always run coverage collection, even if build is FAILED or ABORTED'. Below these checkboxes is a table showing JaCoCo metrics:

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



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

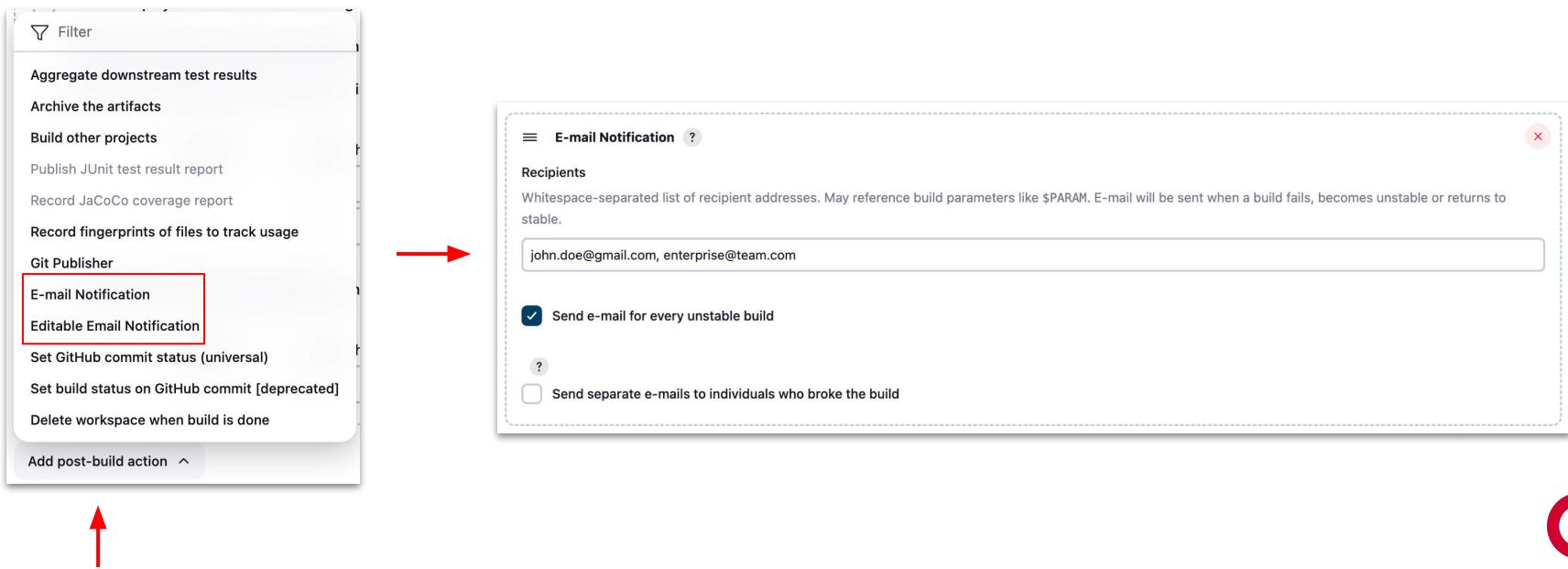
Standard is email, others can be added via plugins

- 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



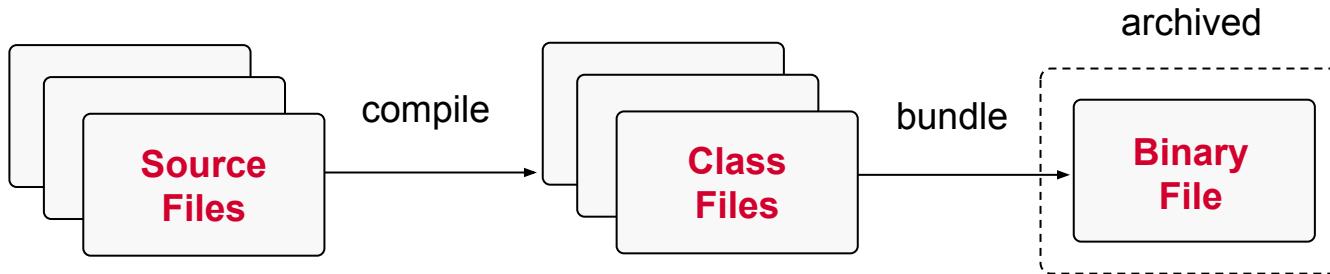
EXERCISE

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 a user interface for configuring a post-build action. On the left, a sidebar lists various actions: 'Aggregate downstream test results', 'Archive the artifacts' (which is highlighted with a red box), 'Build other projects', 'Publish JUnit test result report', 'Record JaCoCo coverage report', 'Record fingerprints of files to track usage', 'Git Publisher', 'E-mail Notification', 'Editable Email Notification', 'Set GitHub commit status (universal)', 'Set build status on GitHub commit [deprecated]', 'Delete workspace when build is done', and 'Add post-build action ^'. A red arrow points from the 'Archive the artifacts' option in the sidebar to a detailed configuration window on the right. This window has a title 'Archive the artifacts' and a sub-section 'Files to archive' containing the pattern 'build/libs/*.jar'. Below this is an 'Advanced' dropdown menu. On the far right of the configuration window is a close button (red circle with an 'X'). At the bottom of the configuration window is a dashed line. On the far left of the configuration window is a vertical scroll bar. In the bottom right corner of the image is a red circular logo with a white dot.

Filter

Aggregate downstream test results

Archive the artifacts

Build other projects

Publish JUnit test result report

Record JaCoCo coverage report

Record fingerprints of files to track usage

Git Publisher

E-mail Notification

Editable Email Notification

Set GitHub commit status (universal)

Set build status on GitHub commit [deprecated]

Delete workspace when build is done

Add post-build action ^

Archive the artifacts

Files to archive

build/libs/*.jar

Advanced

Build Artifacts

gradle-initializr-1.0.0-plain.jar 22.28 KiB view

gradle-initializr-1.0.0.jar 22.47 MiB view

Artifact Retention Policy

“I want to limit the number of archived artifacts”

Discard old builds ?

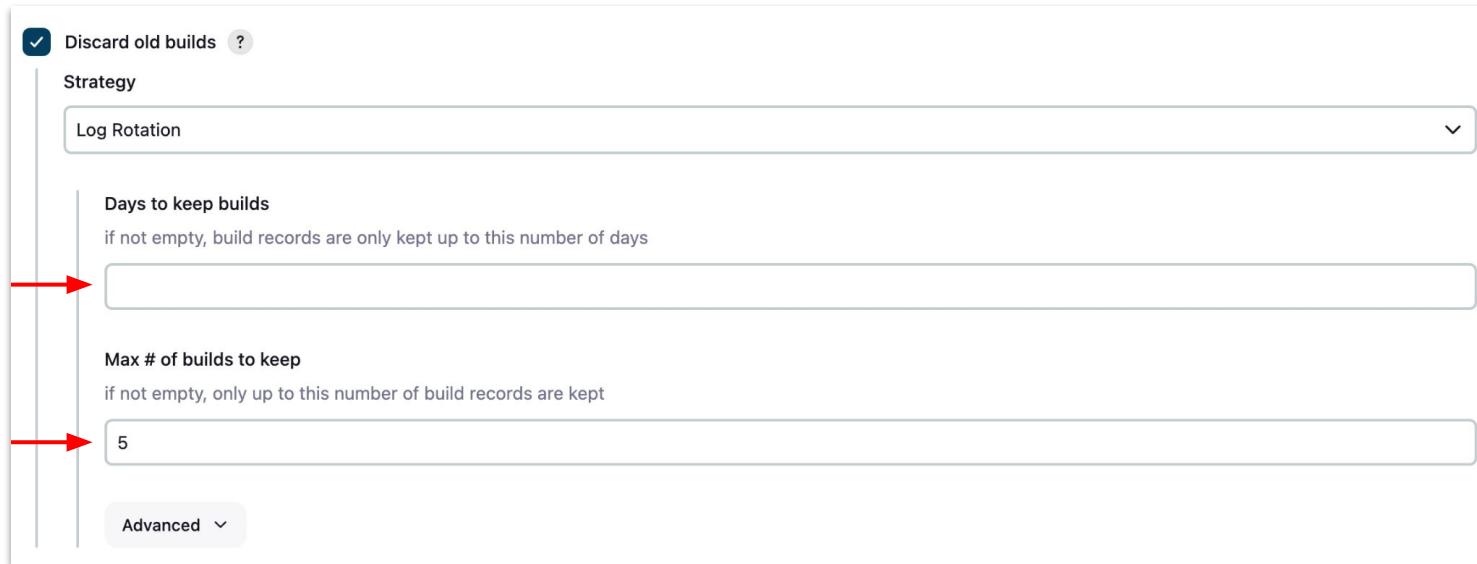
Strategy

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

Advanced ▾



Fingerprinting Artifacts

Track a particular artifact back to the producing build

Archive the artifacts ?

Files to archive ?
build/libs/*.jar

Advanced ^ Edited

Excludes ?
[empty input field]

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 ?

Follow symbolic links ?



or

Record fingerprints of files to track usage ?

Files to fingerprint ?
build/libs/*.jar

Advanced ^



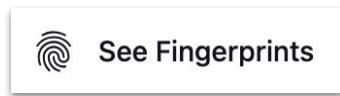
Important for interdependent projects setups



Viewing Artifact Usage

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

From specific Build, click...

 See Fingerprints

↓

Recorded Fingerprints

File ↓	Original owner	Age
build/libs/gradle-initializr-1.0.0-plain.jar	outside Jenkins	44 sec old
build/libs/gradle-initializr-1.0.0.jar	outside Jenkins	44 sec old

gradle-initializr-1.0.0.jar

Introduced 1 min 20 sec ago outside Jenkins

Usage

This file has been used in the following places:

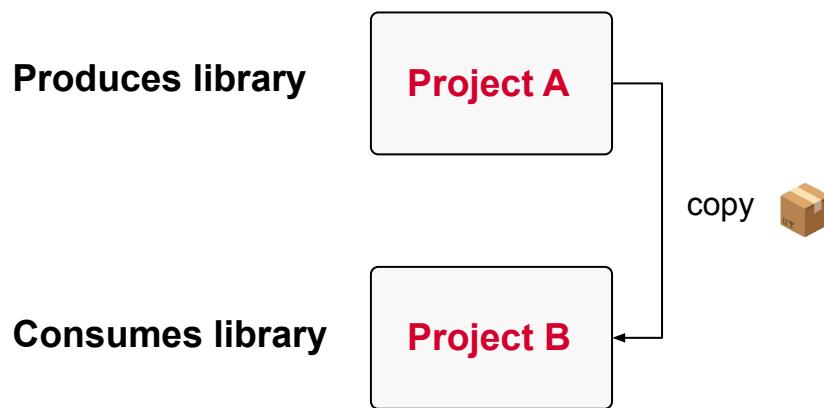
My App  #13

MD5: b25c3d16fe8b44525bca8f6e9867f997



Cross-Project Artifacts

Make the output of a build available to another project



Copying Artifacts

Functionality only available through “Copy Artifact” plugin

The screenshot shows the Jenkins interface for adding a new build step. A red arrow points from the left sidebar to the configuration dialog on the right.

Add build step ^

Filter

- Copy artifacts from another project** (highlighted with a red box)
- Execute Windows batch command
- Execute shell
- Invoke Ant
- Invoke Gradle script
- Invoke top-level Maven targets
- Run with timeout
- Set build status to "pending" on GitHub commit

Copy artifacts from another project

Project name ?
Test

Which build ?
Latest successful build

Stable build only

Artifacts to copy ?
build/*.jar

Artifacts not to copy ?

Target directory ?

Parameter filters ?

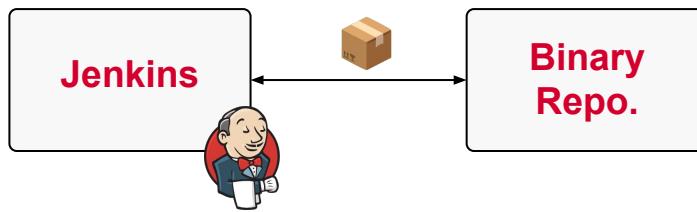
Flatten directories Optional Fingerprint Artifacts Include Build Number ?

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

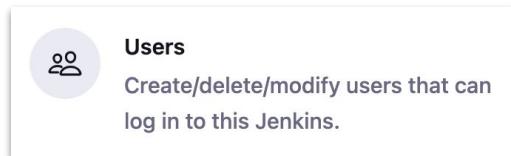
Access Control is primary mechanism for securing Jenkins

- **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 > Users



You can't delete logged-in user

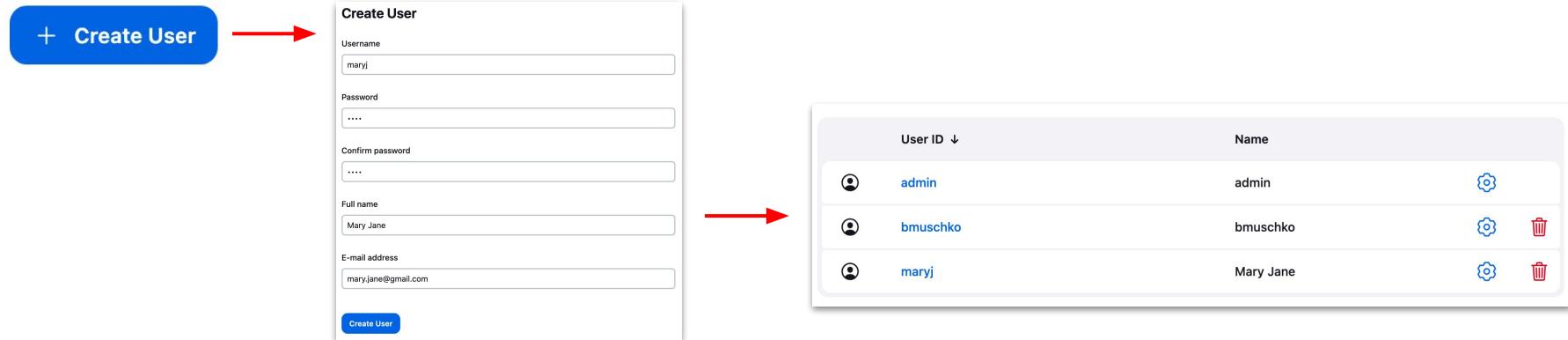


User ID ↓	Name	
	admin	
	bmuschko	



Creating Users

Manage Jenkins > Manage Users > Create User



Enabling Matrix Security

Manage Jenkins > Security

Security

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

Access Control is the primary mechanism for securing a Jenkins environment



Security

Authentication

Disable "Keep me signed in" [?](#)

Security Realm

Jenkins' own user database [?](#)

Allow users to sign up [?](#)

Authorization

Matrix-based security

	Overall	Credentials	Agent	Job	Run	View	SCM	Tag
User/group	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Administrator	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Anonymous	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Authenticated Users	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
Add user...	Add group... ?							



Configuring Matrix Security

Pick appropriate permissions for users or groups

User/group	Overall	Credentials		Agent		Job		Run		View		SCM	
		Read	Delete	Create	Configure	Update	Replay	Delete	Read	Move	Discover	Configure	Tag
Anonymous		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Authenticated Users		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
Mary Jane		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>				

[Add user...](#) [Add group...](#) [?](#)



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



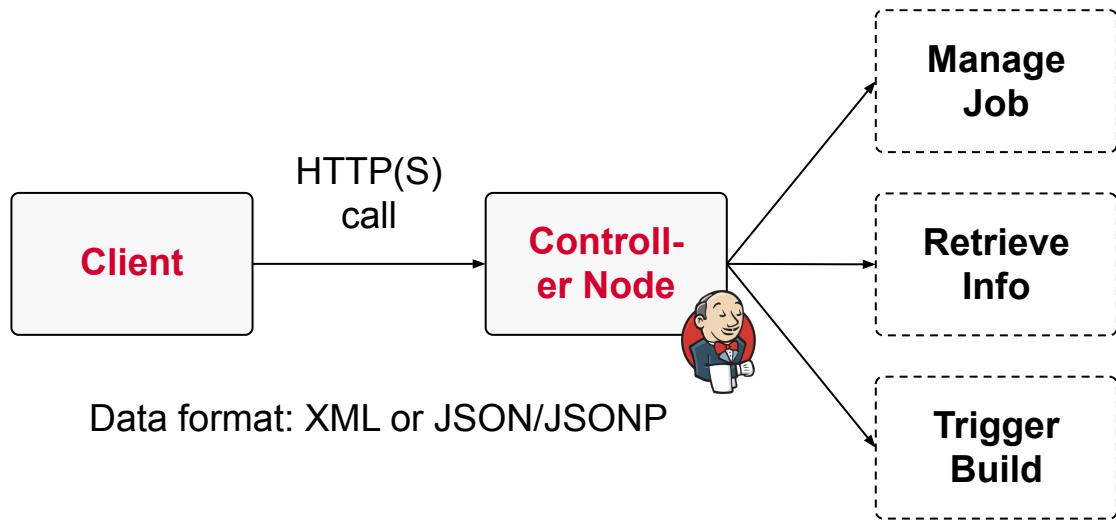
EXERCISE

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

CSRF Protection

Crumb Issuer

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 > Users > Select user

API Token

Current token(s) ?

There are no registered tokens for this user.

Add new Token

API Token

Current token(s) ?

Token created on 2024-03-20T08:53:51Z 117d553271d57100ce22787ec88bb9915d

Copy 117d553271d57100ce22787ec88bb9915d Delete

⚠ Copy this token now, because it cannot be recovered in the future.

Add new Token



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

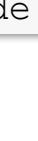
API token



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

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:

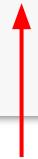
```
$ java -jar jenkins-cli.jar [-s JENKINS_URL] <  
command ...
```



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/>



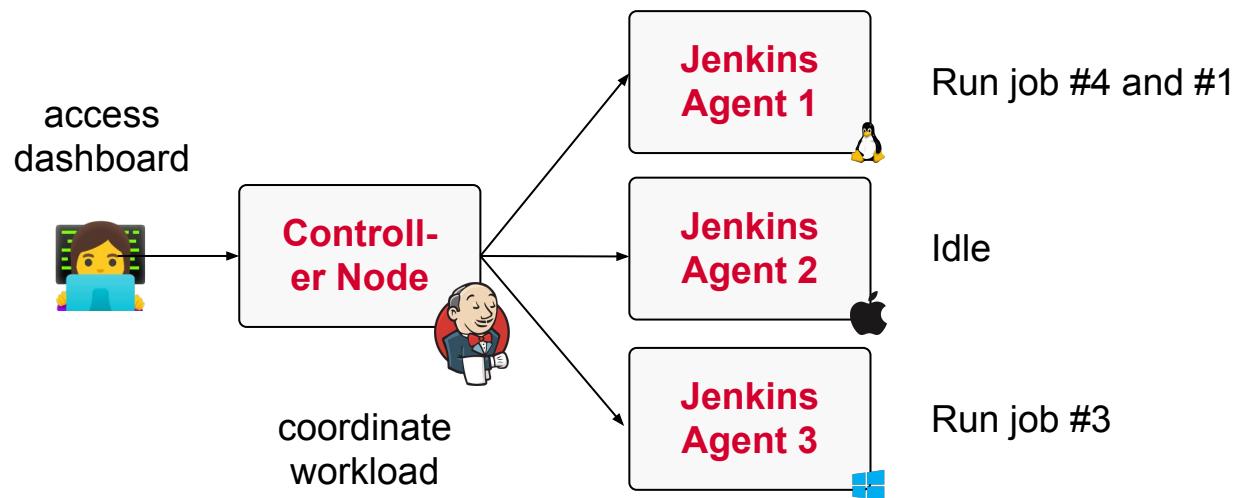
EXERCISE

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 > Nodes



S	Name ↓	Architecture	Clock Difference	Free Disk Space	Free Swap Space	Free Temp Space	Response Time	
	Built-In Node	Mac OS X (x86_64)	In sync	1.52 TiB	1.60 GiB	1.52 TiB	0ms	
	Data obtained	25 min	25 min	25 min	25 min	25 min	25 min	



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 "pi4.local" header, labeled "agent name".



Executor Configuration

Controller node should not execute build workload

Controller configuration

Number of executors ?

Labels ?

Agent configuration

Number of executors ?

Labels ?



Job Execution by Label

Various plugins provide configuration option(s)

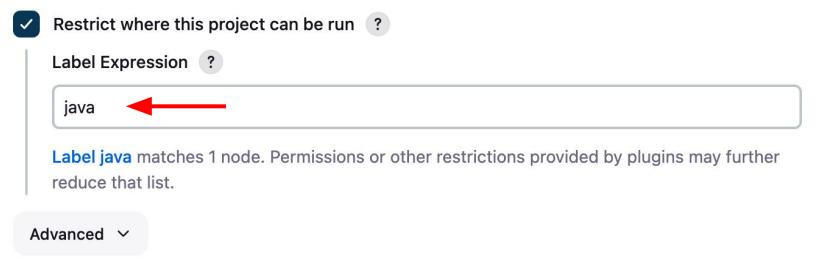
Restrict where this project can be run ?

Label Expression ?

java 

Label **java** matches 1 node. Permissions or other restrictions provided by plugins may further reduce that list.

Advanced ▾



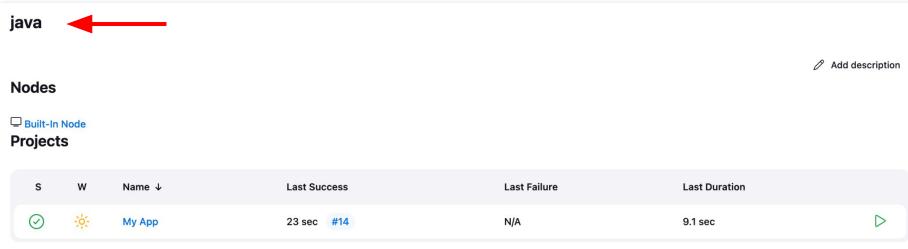
Nodes 

Add description

Built-In Node

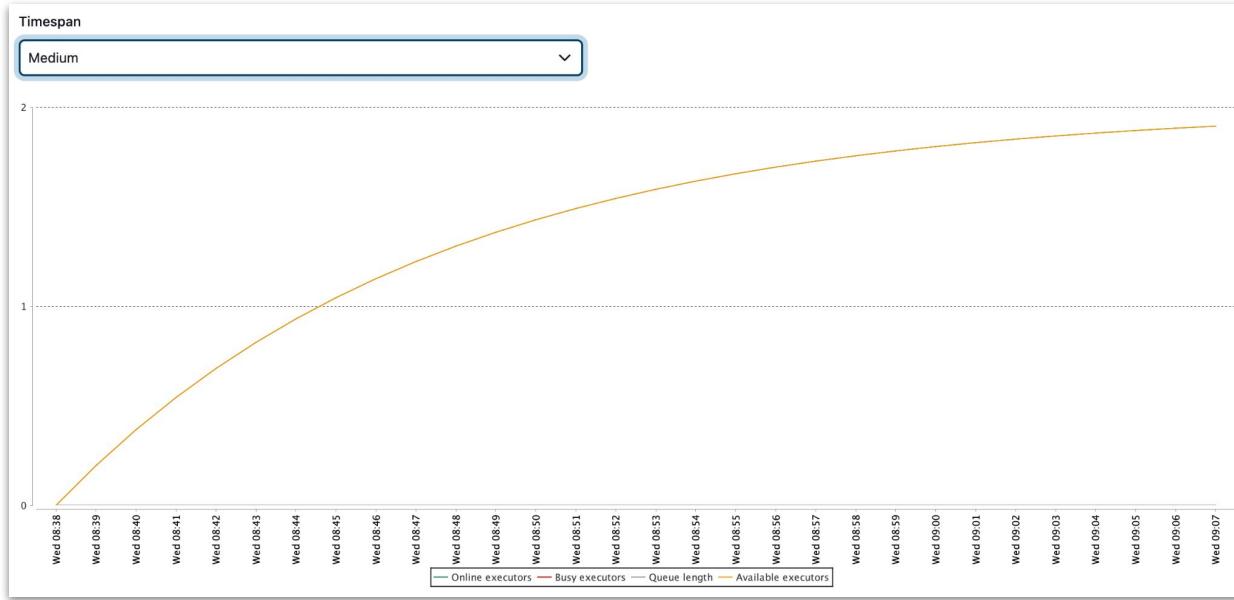
Projects

S	W	Name ↓	Last Success	Last Failure	Last Duration
		My App	23 sec #14	N/A	9.1 sec >



Node Statistics

Keeping track of executors uptime, availability etc.



Adding New Nodes

SSH is the most common way to launch an agent

The image shows two screenshots of the Jenkins 'New node' creation interface. A red arrow points downwards from the 'New Node' button on the left to the 'New node' configuration screen. Another red arrow points from the bottom right of the 'New node' screen to the top right of the 'Advanced' configuration screen.

New node

Node name: new-node

Type: 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.

Create

Advanced

Launch method: Launch agents via SSH

Host: pi2.local

Credentials: jenkins*****

Host Key Verification Strategy: Non verifying 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

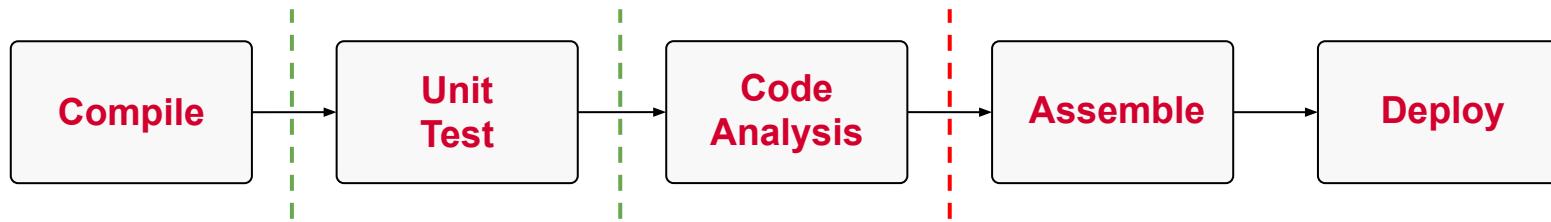


There's no one-size-fits-all pipeline



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?”

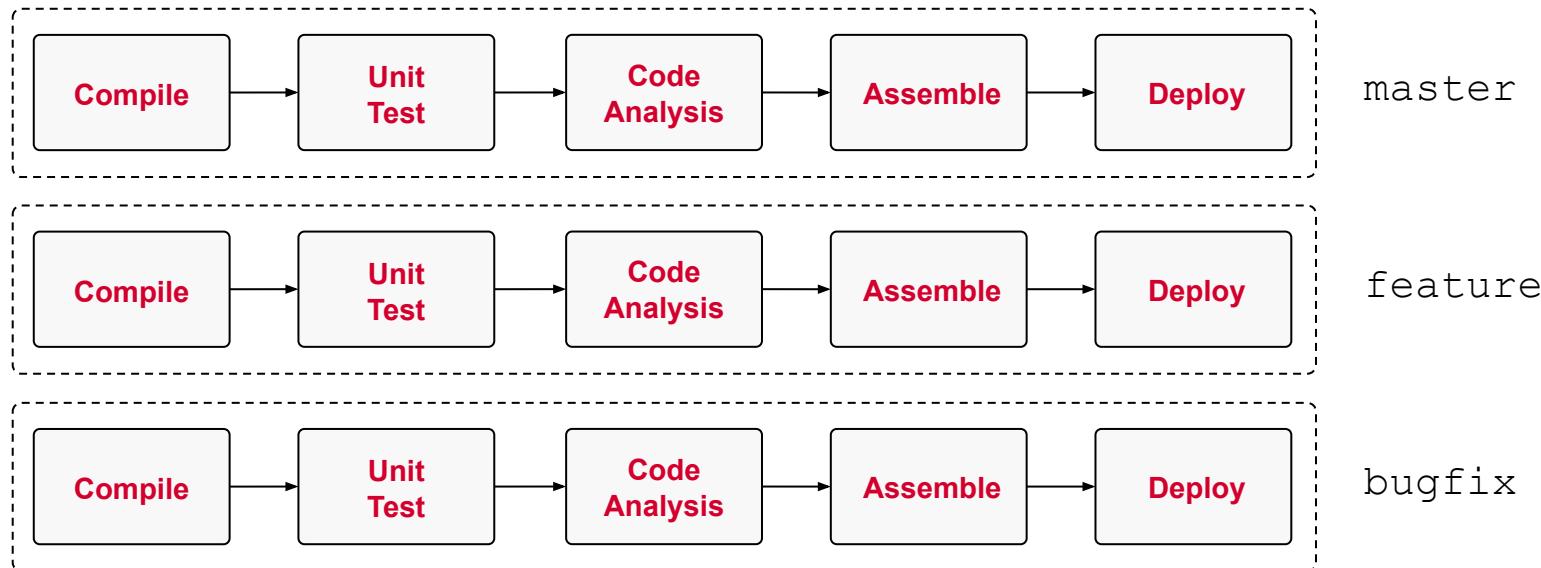


“Do all tests pass and produce
>= 70% code coverage?”



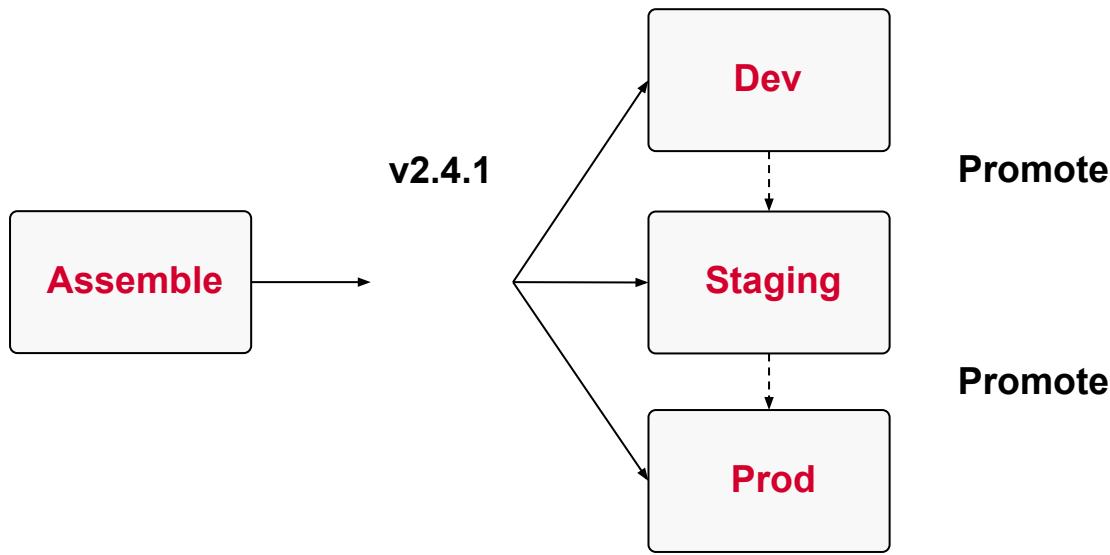
Same Pipeline for Branches

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!'  
            }  
        }  
    }  
}
```

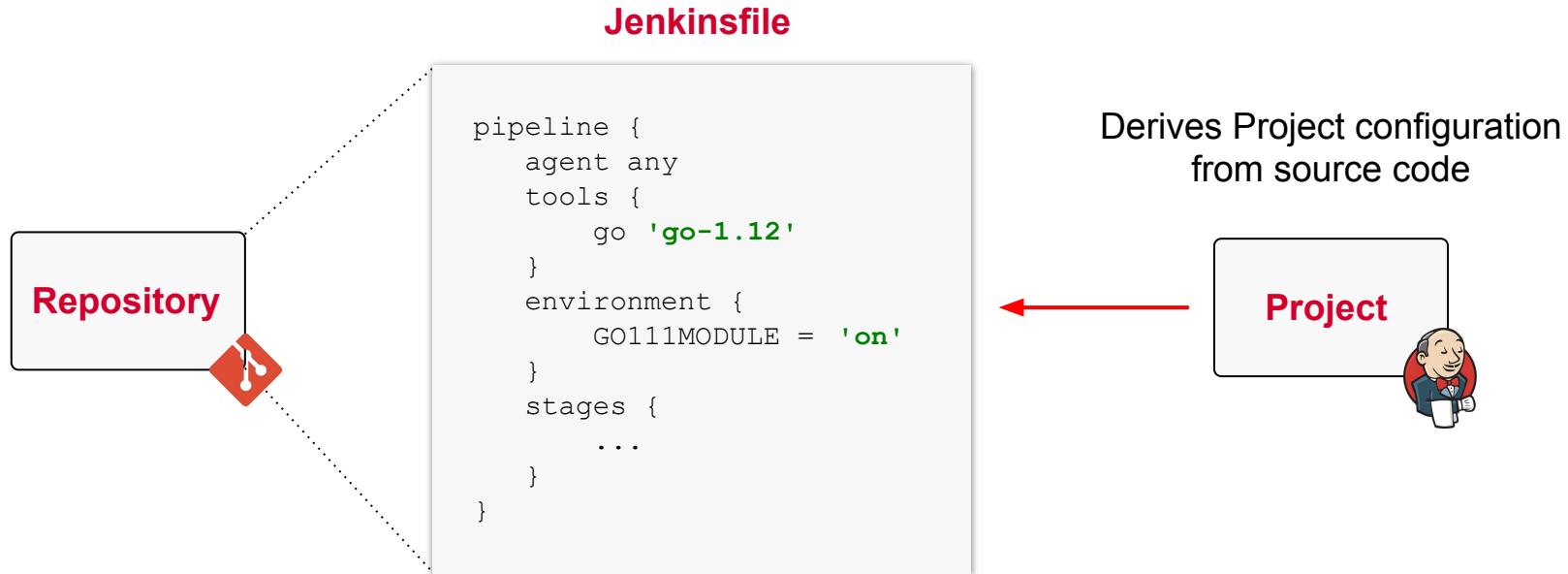
VS.

```
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...

+ New Item

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.

Multibranch Pipeline
Creates a set of Pipeline projects according to detected branches in one SCM repository.



Scripted Pipeline Definition

Scripting is an option but versioned code is preferable

The screenshot shows a 'Pipeline' configuration screen. At the top, there's a dropdown menu labeled 'Definition' with 'Pipeline script' selected. A red arrow points to this dropdown. Below it is a 'Script' section containing a Groovy code snippet:

```
1 -> node {  
2     echo 'Hello World'  
3 }
```

A red arrow points from the text 'easy to edit and try out' down towards the script editor area. At the bottom left, there's a checked checkbox for 'Use Groovy Sandbox'. At the very bottom, there are 'Save' and 'Apply' buttons.



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

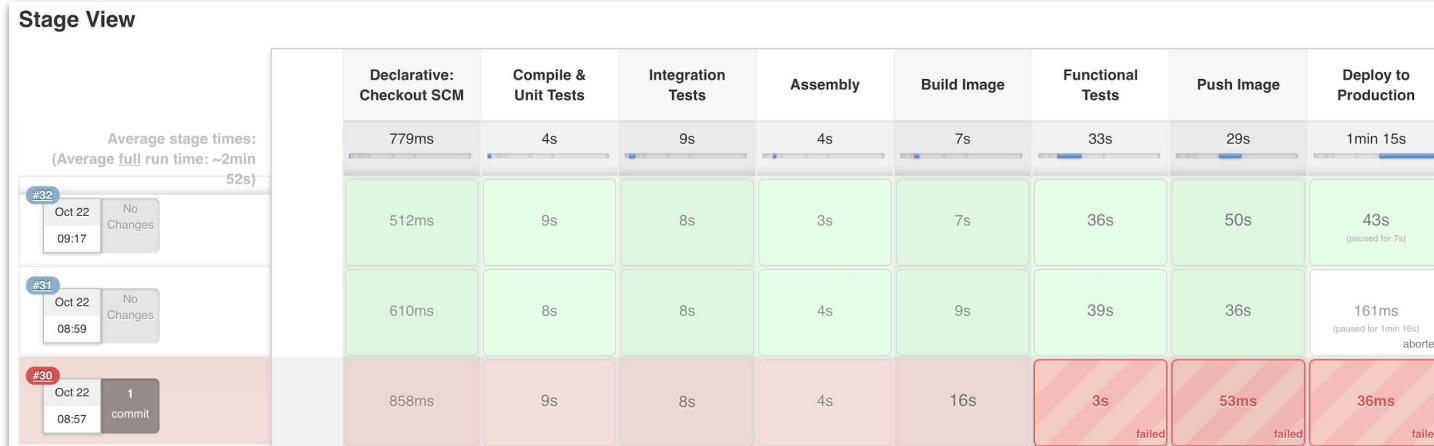


Jenkinsfile language support



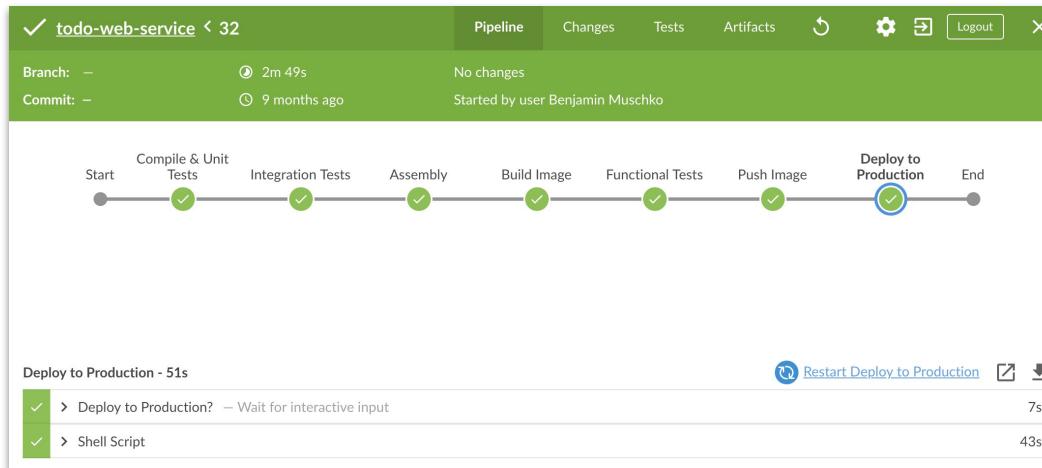
Standard Pipeline Visualization

Only renders pipelines linearly



Using the BlueOcean Plugin

Next generation pipeline visualization

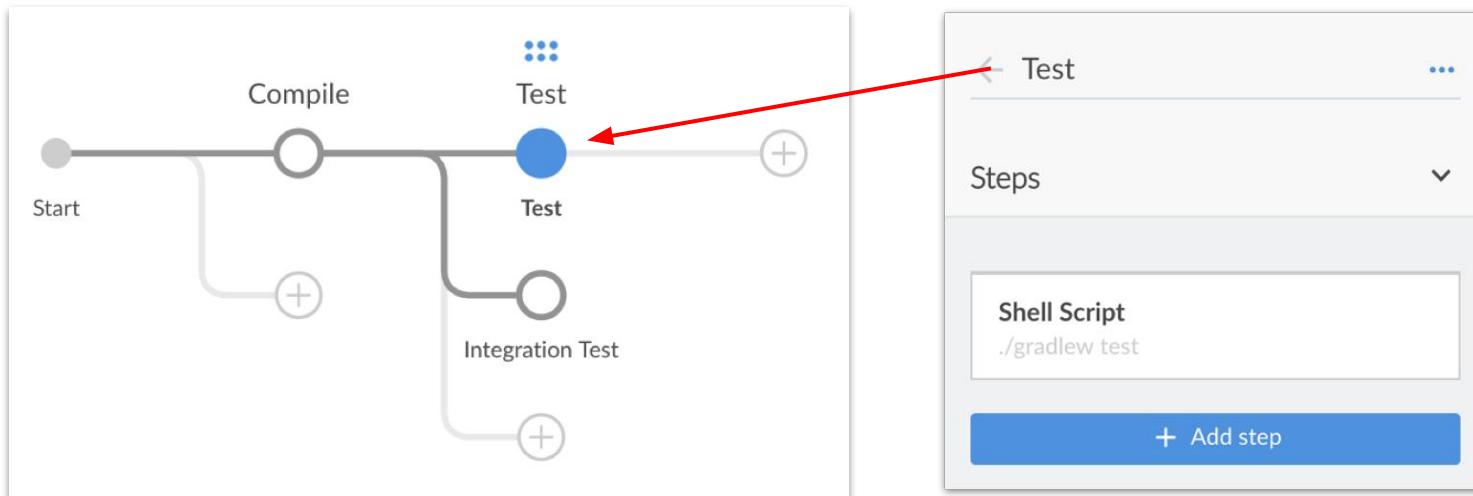


Plugin is not part of standard installation



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

```
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')  
}
```

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

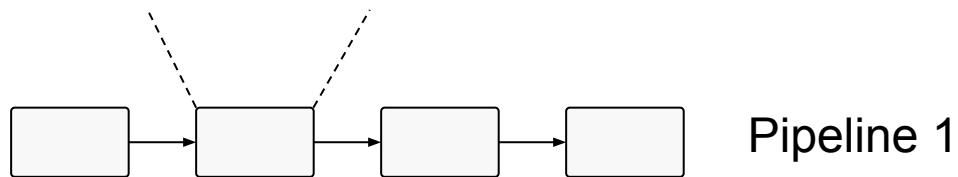
Enhancing a Pipeline
with Advanced Features



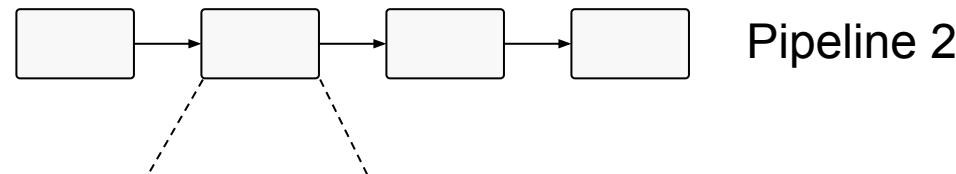
Keeping Pipelines DRY

Avoid copy/pasting pipeline code across multiple projects

Steps Definition



Pipeline 1



Steps Definition



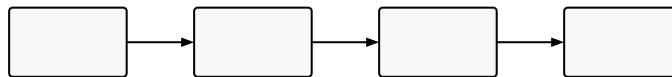
Pipeline Templates

You can even define a “standard” pipeline if needed

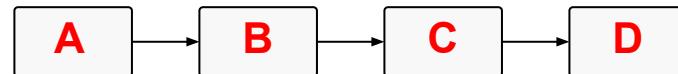
Inject Dynamic Parameter Values



```
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.

 Library	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"/>	
Retrieval method		
<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

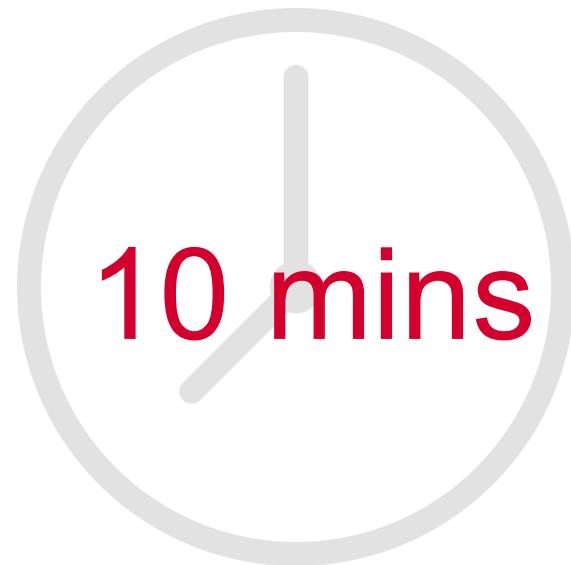
Writing and Using a
Shared Library



Q & A



BREAK

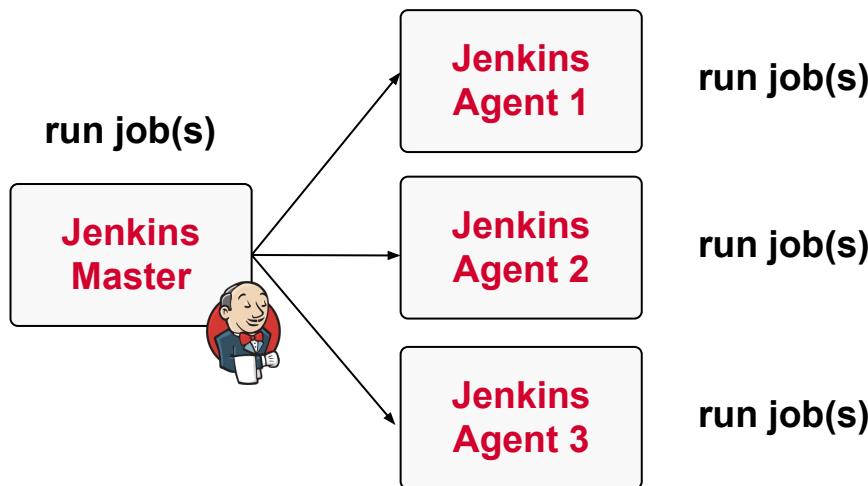


CD-as-Code Best Practices

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



Defining Reusable Agents

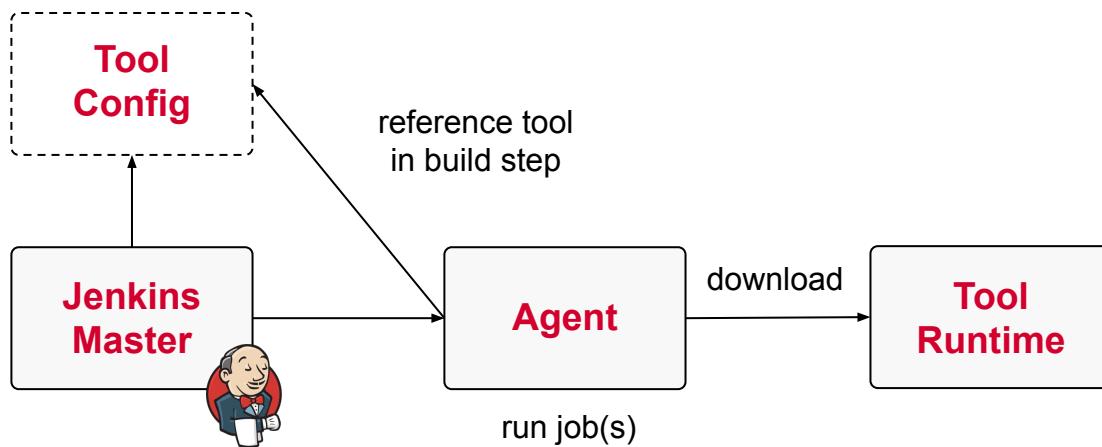
Generalize as much as possible, specialize if needed

- 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

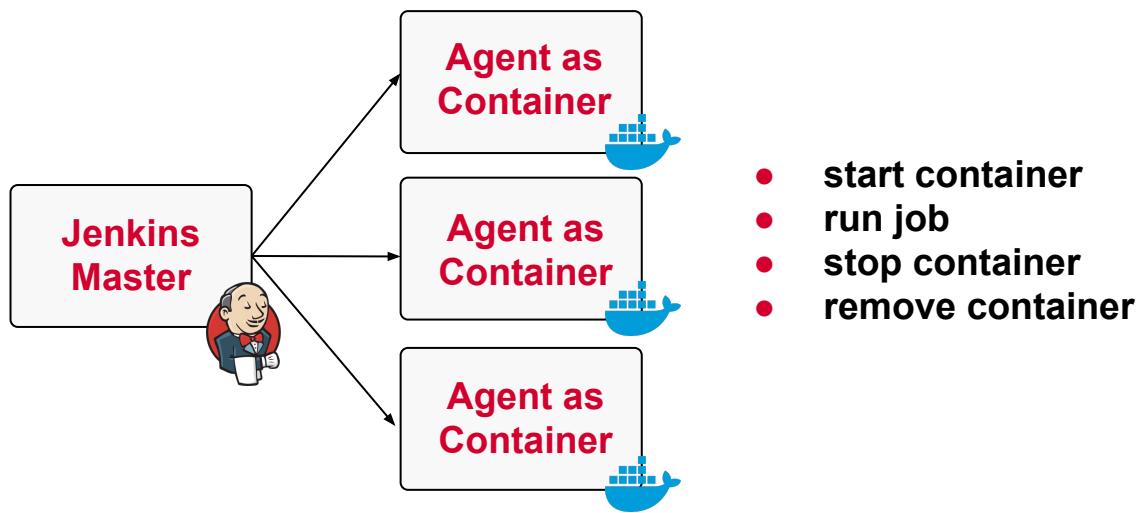
Offload build execution to on-demand agents

- 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



Hardware for Master node

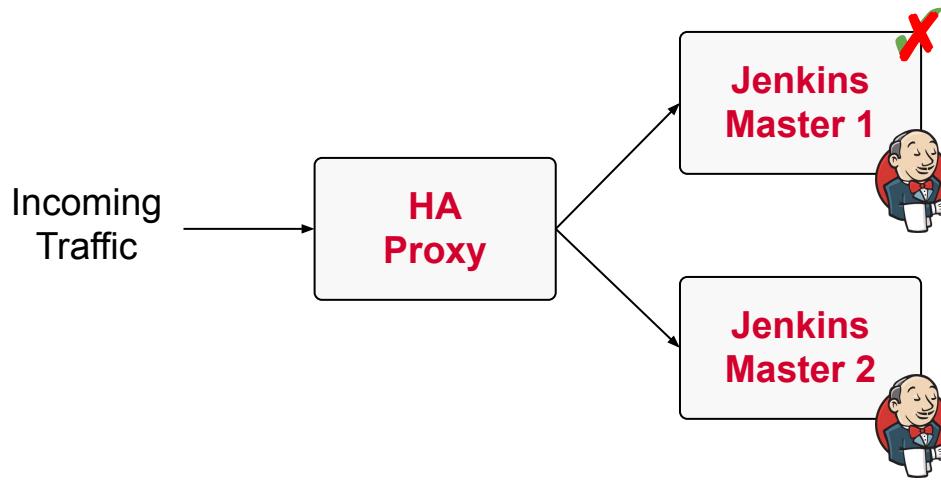
There's no one-size-fits-all solution

- **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



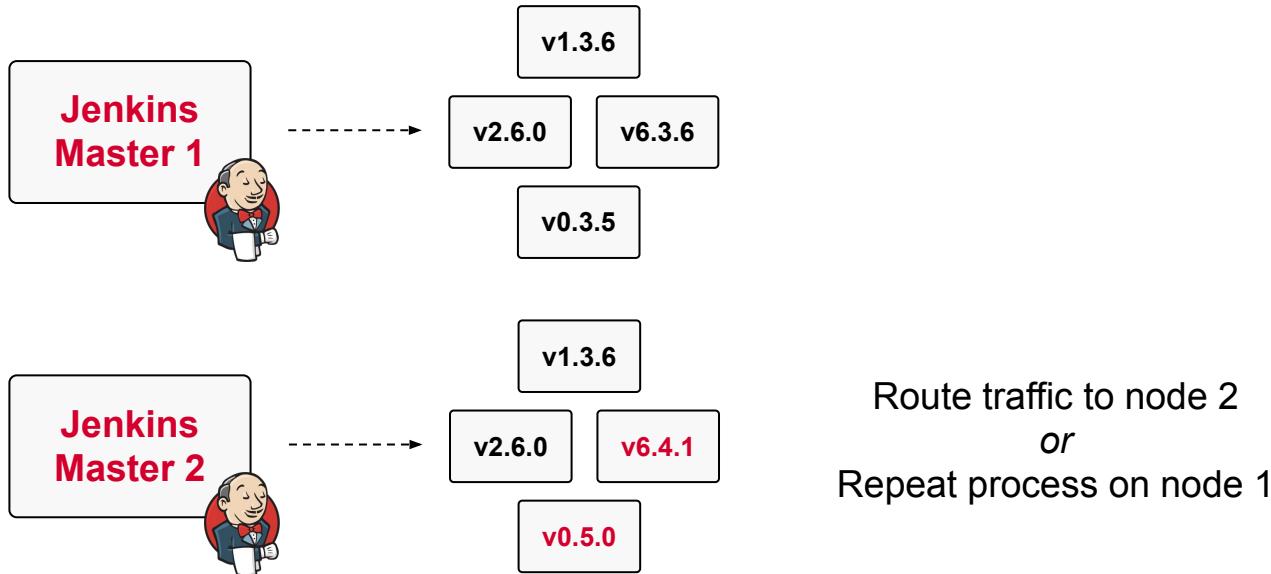
High Availability for Master node

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

