

Continuous Delivery with the CloudBees Jenkins Platform

User training Labs

Table of Contents

Introduction	1
Lab 1: Creating a Freestyle Build Job	1
Goal	1
Pre-Step: Configure Jenkins for Maven	1
Step 1. Create a New Build Job	2
Step 2. Configure the SCM Details	3
Step 3. Configure the Build Triggers	4
Step 4. Configure the Build Goals	5
Lab 2: Creating a Maven build job	8
Goal	8
Step 1. Create a new build job	8
Step 2. Configure the SCM details	9
Step 3. Configure the Build Triggers	10
Step 4. Configure the build goals	11
Lab 3: Displaying test results	12
Goal	12
Step 1. Triggering a failed build	13
Step 2. Look at the failed build in Jenkins	18
Step 3. Fix the build	19
Lab 4: Creating an integration tests build	19
Goal	19
Step 1	20
Step 2	21
Lab 5: Integrating with a source repository browser	22
Goal	22
Step 1. Integrate with Gogs	22
Step 2. Making a change	22
Lab 6: Code quality metrics	26
Goal	26
Step 1. Generate the code quality metrics	26
Lab 7: Code coverage metrics	31
Goal	31
Step 1. Set up code coverage reporting	31
Lab 8: Parameterized builds	35
Goal	35
Step 1. Check installation state of the Parameterized Triggers plugin	35

Step 2. Configuring the Parameterized Triggers plugin	35
Lab 9. Automatic deployments to Tomcat	37
Goal.....	37
Step 1: Automatically deploy a SNAPSHOT web application to Tomcat	38
Lab 10: Job Organization and Security with Folders	41
Goal.....	41
Step 1. Securing with RBAC plugin	42
Step 2. Organizing with Folders Plus Plugin.....	50
Lab 11: Validated merge	54
Goal.....	54
Step 1: Fork project in Gogs as "dev" user.....	54
Step 2: Global Jenkins Administration Setup	57
Step 3: Setup a Jenkins job for CI build & validated merge.....	59
Step 4: Edit the code and test the change.....	62
Step 5: Verify there is no regression	67
Step 6: Fixing the regression	68
Step 7: Verifying the change pushed back	69
Lab 12. Pipeline.....	70
Experiment with DSL Syntax	70
Pipeline Control flows	73
Concurrent execution.....	74
Pipeline Stages	75
Lab 13: Templates	78
Builder template.....	78
Job template	81
Pipeline Templates	85
Lab 14: CloudBees Support	93
Goal.....	93
Step 1: Verify / Install the Support plugin	93
Step 2: Generate a bundle.....	94

Introduction

This workbook is designed to supplement your CloudBees Jenkins training. It consists of a sequence of lab exercises that will introduce continuous integration concepts and Jenkins best practices.

These lab exercises should be completed in sequence and in the presence of an instructor. The instructor will distribute the software and code necessary to complete these exercises.

The "Install Lab Environment" document will cover the setup of all the required software.

Your instructor will tell you if this has already been done on your machine, or if you need to do so yourself.

Most of the plugins required for this course are bundled with this installation, the Lab steps have additional information on which plugins are used and how to go about installing them if they were needed.

IMPORTANT Never try to upgrade plugins of the Lab's Jenkins instance !

If you encounter problems with your software installation or if you do not understand any of the instructions, please ask your instructor for help.

Lab 1: Creating a Freestyle Build Job

Goal

The aim of this lab exercise is to configure a simple freestyle build job on your Jenkins server.

Pre-Step: Configure Jenkins for Maven

Start by accessing Jenkins configuration:

- Click on the **Manage Jenkins** menu item.
- Then, click on **Configure System**.

We need to configure a Maven installation, that will be used for building the example applications:

- On the **Maven** section, click on the **Maven installations...** button.
- Then, configure the new Maven Installation with this settings:

Name: maven-3

- Install automatically:** Checked
- Use **Install from Apache**, using the latest 3.3.x available



Figure 1. Maven Configuration

Finally, save this configuration by clicking on the **Save** button at the bottom of the screen.

Step 1. Create a New Build Job

Click on the **New Item** menu on the home page, and choose **Build a free-style software project**.

Name this project `gameoflife-freestyle` and click **OK**:

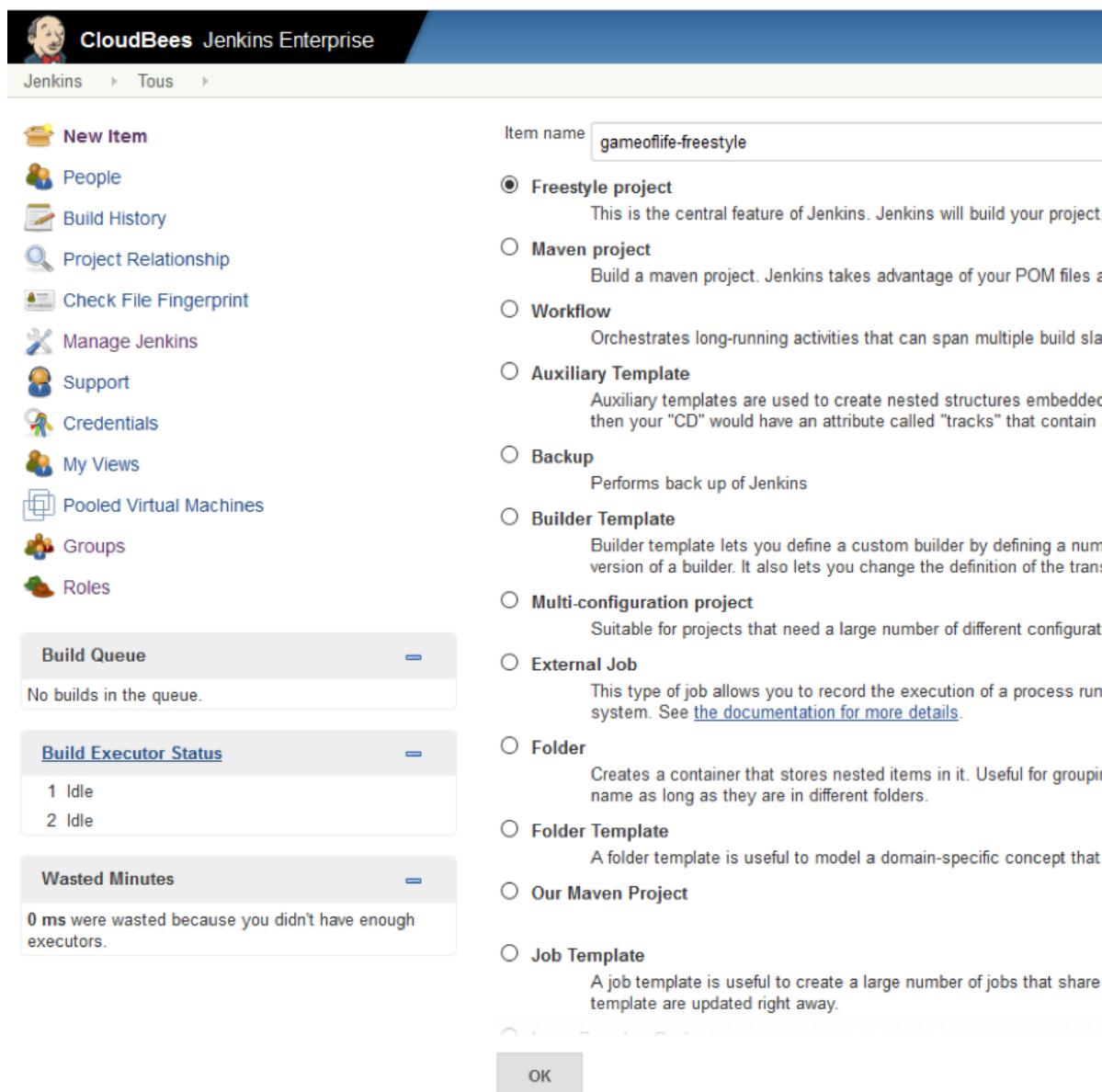


Figure 2. New Job Form of kind Freestyle

Step 2. Configure the SCM Details

For this build job, you need to configure:

- First, choose **Git** in the **Source Code Management** section.
- Then, in the **Repository URL** field, enter your GIT repository address:

```
http://localhost:5000/gitserver/harry/gameoflife.git
```

- Since this repository need authentication to be accessed, you will need to add a new credential with those properties:

☒ Kind: **Username with password**

- ☒ Scope: **Global** (to allow reusing in other jobs)
 - ☒ Username is **harry**
 - ☒ Password is the **same** value as the username
- Then ensure your newly created credential is selected:

Source Code Management

None Git

Repositories

Repository URL: http://54.183.97.218/git/harry/gameoflife.git

Credentials: harry/*****

Add Repository

Branches to build

Branch Specifier (blank for 'any'): */master

Add Branch

Repository browser: (Auto)

Additional Behaviours: Add

Figure 3. SCM Configuration for Job

Step 3. Configure the Build Triggers

Next, set up the build triggers. We will be polling the Git repository every minute, so in the **Poll SCM** section, enter:

```
* * * * *
```

Build Triggers

- Trigger builds remotely (e.g., from scripts)
- Build pull requests to the repository
- Build when a change is pushed to GitHub
- Build when another project is promoted
- Build after other projects are built
- Build periodically
- Monitor Docker Hub for image changes
- Poll SCM

Schedule

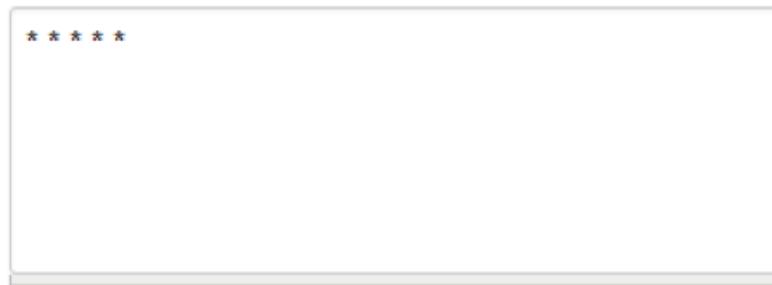


Figure 4. Build Triggers Configuration for Freestyle Job

Note that a warning (Yellow text) may appear proposing to use:

TIP

H * * * *

instead. Do **not** take it in account for now.

Step 4. Configure the Build Goals

Finally, you need to configure the Maven build goals.

First, add a new **Invoke top-level Maven targets** build step:

Build

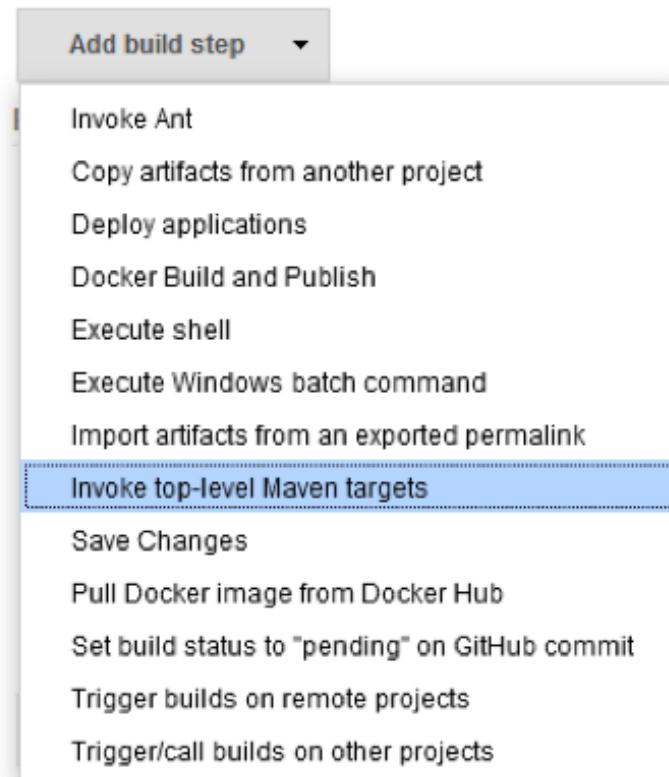


Figure 5. Adding a Maven Build Step

Then:

- Select your **maven-3** installation as **Maven Version**
- Add the following content in the **Goals** field:

```
clean install -B -U
```

Build

The image shows the configuration for the 'Invoke top-level Maven targets' build step. It includes fields for 'Maven Version' (set to 'maven3') and 'Goals' (set to 'install'). At the bottom right are 'Advanced...' and 'Delete' buttons. Below the main configuration is a 'Add build step' dropdown menu.

Figure 6. Maven Build Step Configuration

- Next, click on the **Advanced** button to enter some extra property values to pass to the build:

```
surefire.useFile=false
webdriver.port=9091
jetty.stop.port=9998
```

Build

Invoke top-level Maven targets

Maven Version	maven3
Goals	install
POM	
Properties	<pre>surefire.useFile=false webdriver.port=9091 jetty.stop.port=9998</pre>
JVM Options	
Use private Maven repository	<input type="checkbox"/>
Settings file	Use default maven settings
Global Settings file	Use default maven global settings

Delete

Figure 7. Maven Build Step Advanced Configuration

This will compile, test and package the application.

The `webdriver.port` and `jetty.stop.port` options are parameters used in the "Game of Life" project, that tell Maven which port to run the test application on during the web tests.

- Finally, you need to tell Jenkins where to find the test results:
 - ☒ Go to the **Post-build Actions** and tick the **Publish JUnit test result report**.
 - ☒ Enter the following in the **Test report XMLs** field:

```
**/target/surefire-reports/*.xml
```

Post-build Actions

Publish JUnit test result report

Test report XMLs: **/target/surefire-reports/*.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

Delete

Add post-build action ▾

Figure 8. Post Build Step: Publishing JUnit Reports

Now save this project. A build should kick off almost immediately. Go back to the dashboard to watch the build in progress.

Go back to slides

Lab 2: Creating a Maven build job

Goal

The aim of this lab exercise is to configure a simple Maven build job on your Jenkins server.

Step 1. Create a new build job

Start by clicking on the **New Item** menu item on the home page, and choose **Maven project**.

Call this project gameoflife-default:

New Item

Item name gameoflife-default

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

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.

Workflow
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines and/or organizing complex activities that do not easily fit in free-style job type.

Auxiliary Template
Auxiliary templates are used to create nested structures embedded within other templates as attribute values. For example, if you are modeling a CD, you'd define an auxiliary template called "Track", and then your "CD" would have an attribute called "tracks" that contain a list of "Track"s.

Backup
Performs back up of Jenkins

Builder Template
Builder template lets you define a custom builder by defining a number of attributes and describing how it translates to the configuration of another existing builder. This allows you to create a locked down version of a builder. It also lets you change the definition of the translation without redoing all the use of the template.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See [the documentation for more details](#).

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.

Folder Template
A folder template is useful to model a domain-specific concept that contains other "stuff" in it.

Our Maven Project

Job Template
A job template is useful to create a large number of jobs that share a similar configuration. A designer of a template can enforce uniformity, and when a template changes, all the jobs created from the template are updated right away.

Long-Running Project
A project which runs a special build step detached from Jenkins.

Publisher Template
This publisher template lets some Jenkins or maven publisher has attributes as members of attributes and describes how it translates to the configuration of another publisher

OK

Figure 9. New Job Form of Maven kind

Step 2. Configure the SCM details

For this build job, you need to configure those SCM settings:

- As for the Lab 2, choose **Git** in the **Source Code Management** section
- In the **Repository URL** field, enter your GIT repository address:

```
http://localhost:5000/gitserver/harry/gameoflife.git
```

- Re-use the Credentials you configured in Lab 2

Source Code Management

None
 Git

Repositories

Repository URL

Credentials

Branches to build

Branch Specifier (blank for 'any')

Repository browser

Additional Behaviours

Mercurial
 Subversion

Figure 10. SCM Configuration for Job

Step 3. Configure the Build Triggers

As for the Lab 2, Build Triggers have to be configured as below:

- Polling the SCM repository every minute. In the **Poll SCM** section, enter:

```
* * * * *
```

Build Triggers

- Trigger builds remotely (e.g., from scripts)
- Build pull requests to the repository
- Build when a change is pushed to GitHub
- Build when another project is promoted
- Build after other projects are built
- Build periodically
- Monitor Docker Hub for image changes
- Poll SCM

Schedule

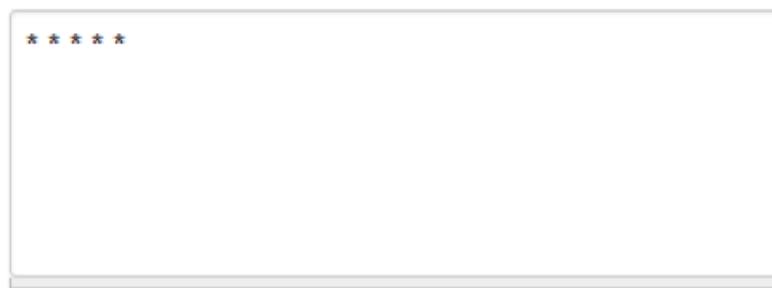


Figure 11. Build Triggers Configuration for Job

Step 4. Configure the build goals

Finally, you need to configure the Maven build goals, in the **Goals and options** field:

```
clean install -B -U -Dsurefire.useFile=false
```

Build	
Maven Version	maven-3.3.3
Root POM	pom.xml
Goals and options	clean install -B -U -Dsurefire.useFile=false
Advanced...	

Figure 12. Maven Job Build Configuration

This will compile, test and package the application, and **deploy** the latest snapshots to the local repository.

Now save this project. A build should kick off almost immediately.

Go back to the dashboard to watch the build in progress.

[Go back to slides](#)

Lab 3: Displaying test results

Goal

In this lab, you will see how Jenkins displays failing tests. You will need to modify the source code of the application.

For this, we provide you an online IDE, hosted on the machine.

Access the IDE with this URL, and authenticate as **harry**:

```
http://localhost:5000/webide
```

TIP **harry**'s password is also **harry**

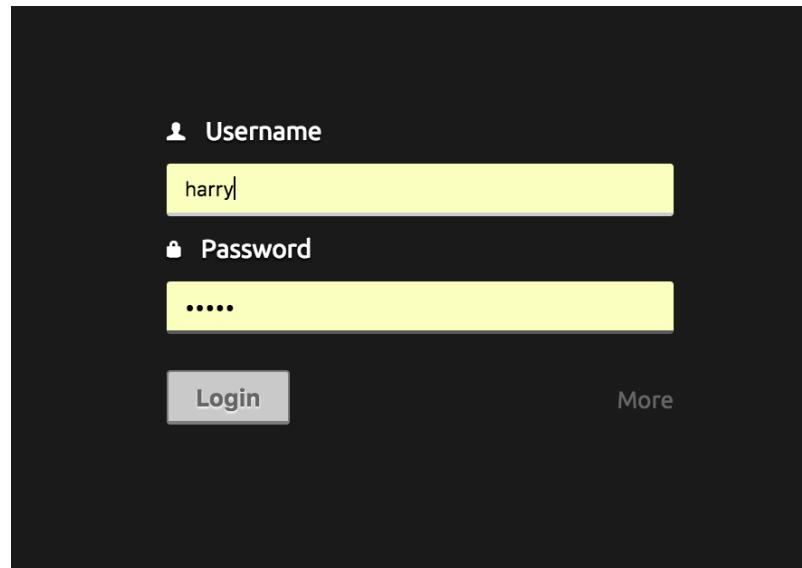


Figure 13. Authenticating on the Web IDE

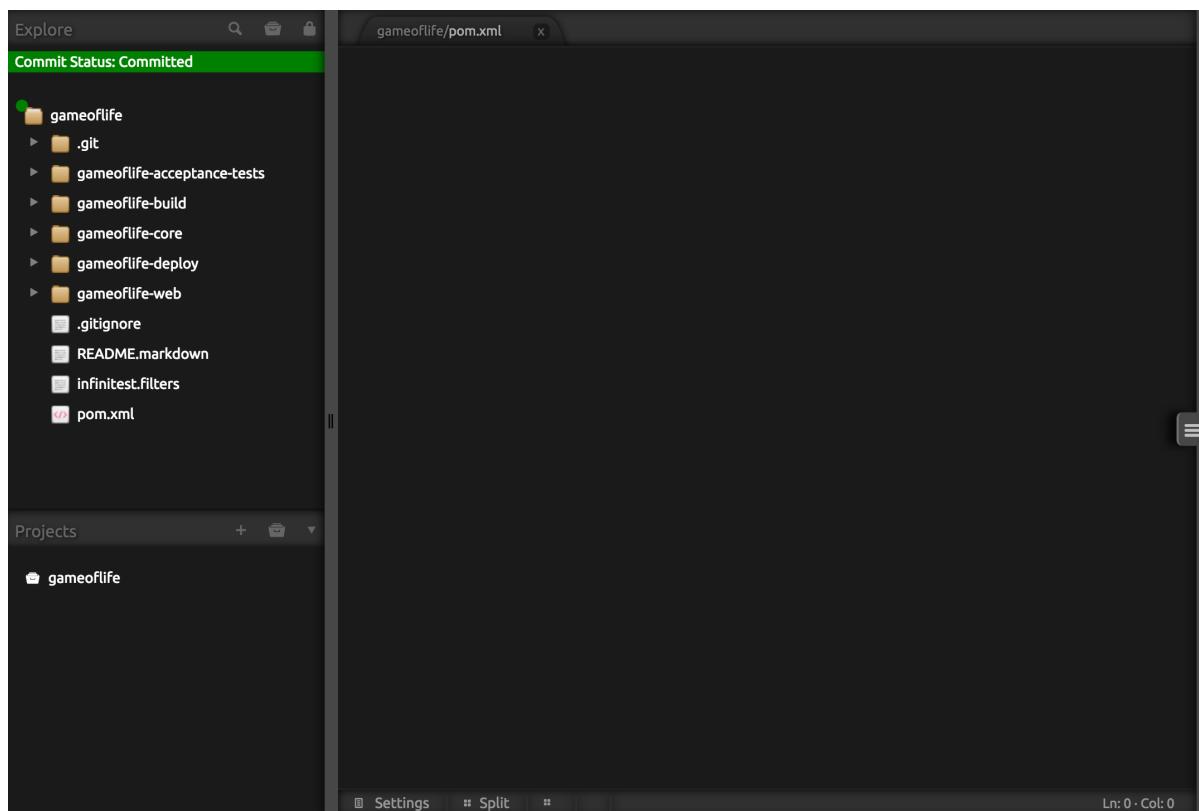


Figure 14. Main Page of the Web IDE

Step 1. Triggering a failed build

On the **Projects** section (bottom right panel), click on the **+** button to initialize our project.

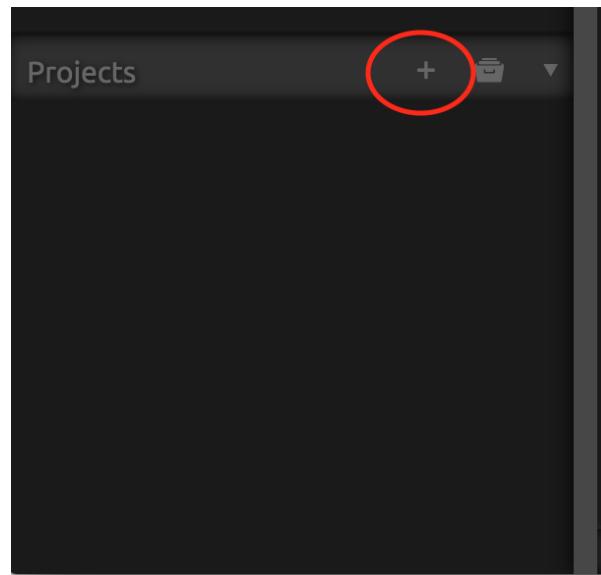


Figure 15. Create project in the Web IDE

Then extend the form by clicking the **...from Git Repo** button, and fill it with those settings:

- **Project Name:** gameoflife

- **Folder Name or Absolute Path:** gameoflife
- **Git Repository:** <http://localhost:5000/gitserver/harry/gameoflife.git>
- **Branch:** master

Then launch the initialization by clicking the **Create Project** button:

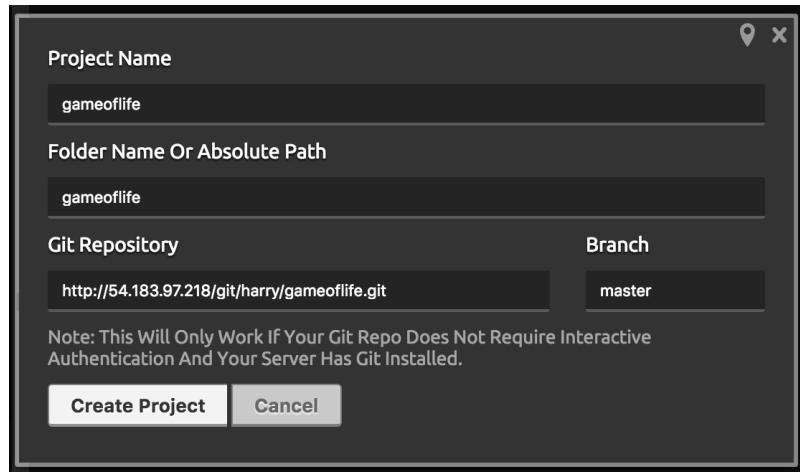


Figure 16. New project form in the Web IDE

TIP Don't forget to watch the notification in the bottom right corner !

To see how Jenkins reacts to a build failure, we are going to introduce an error into the source code.

One way to break the build is to change the symbol representing a dead cell from a period (".") to something else (say, a dash ("-")).

You can do this by modifying the `Cell.java` class, which is located in:

```
gameoflife-core/src/main/java/com/wakaleo/gameoflife/domain/
```

```
neoflife/domain/Cell.java x
1 package com.wakaleo.gameoflife.domain;
2
3 public enum Cell {
4     LIVE_CELL('*'), DEAD_CELL('-');
5
6     private String symbol;
7
8     private Cell(final String initialSymbol) {
9         this.symbol = initialSymbol;
10    }
11
12    @Override
13    public String toString() {
14        return symbol;
15    }
16
17    static Cell fromSymbol(final String symbol) {
18        Cell cellRepresentedBySymbol = null;
19        for (Cell cell : Cell.values()) {
20            if (cell.symbol.equals(symbol)) {
21                cellRepresentedBySymbol = cell;
22                break;
23            }
24        }
25        return cellRepresentedBySymbol;
26    }
27
28    public String getSymbol() {
29        return symbol;
30    }
31 }
32
```

File saved

Figure 17. Editing Cell.java Class

Now we have to commit these changes to the **local** Git repository, and synchronize the change to the **remote** Git repository.

- First, navigate to the **CodeGit**, by using the right panel:

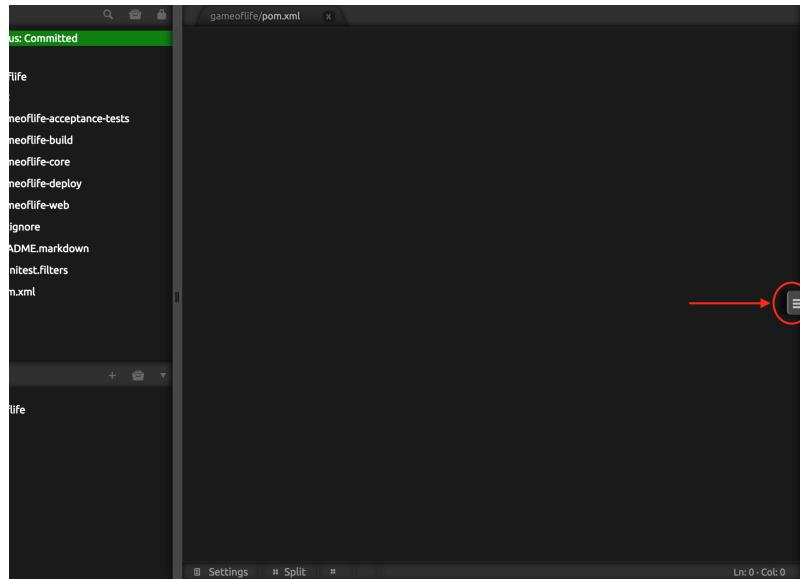


Figure 18. Accessing the right panel

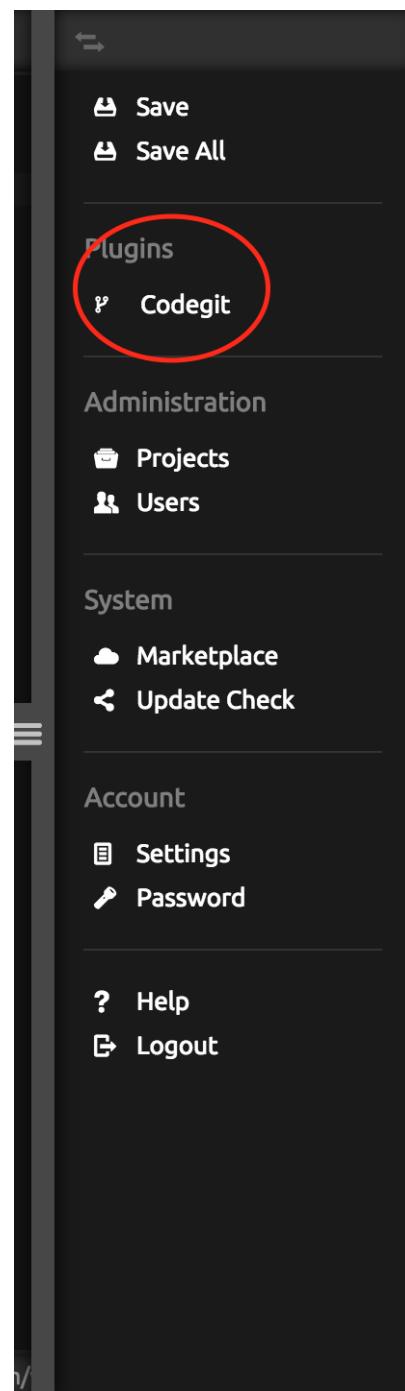


Figure 19. Accessing CodeGit

- In the GitLab view, select the change you've made on Cell class and click the **Commit** button:



Figure 20. CodeGit main view

- Then, enter a commit message like: Changing Cell symbol and click the **Commit** button:



Figure 21. Commit message for Cell class

TIP A green "Change committed" notification will popup on the bottom right

- We have to "push" the committed changes to the **remote** repository. For this, click the **Pull/Push** button in the main view to access the related view:

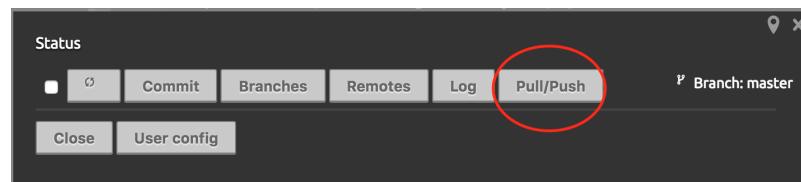


Figure 22. Acces to Push/Pull section

- On the Pull/Push view, fill the form and click to the **Push** button:

Remote: origin

Branch: master

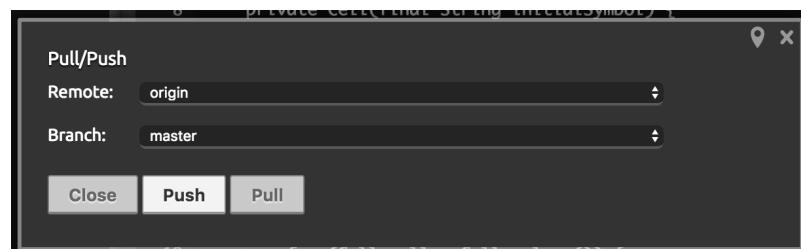


Figure 23. Push/Pull view

- When pushing your changes, you will be prompted for a username and password. Just use **harry** for both username and password.



Figure 24. Authenticating against Remote Git Repository

Step 2. Look at the failed build in Jenkins

Now go to Jenkins and investigate how Jenkins displays the build failure.

Drill down and study the details in some of the test failures.

Also investigate the **Trend** link to see how long the tests take to run over time.

CloudBees Jenkins Enterprise

search harry | logout

Jenkins > gameoflife-default > #2

ENABLE AUTO REFRESH

Back to Project

Status

Changes

Console Output

Edit Build Information

Delete Build

Deploy Now

Tag this build

Redeploy Artifacts

Test Result

See Fingerprints

Previous Build

Next Build

Build #2 (Mar 17, 2016 3:12:24 PM)

Started 1 hr 38 min ago
Took 26 sec

[add description](#)

Revision: 54
Changes

1. failing build ([detail](#)/Sventon 2.x)

Started by user harry

This run spent:

- 16 ms waiting in the queue;
- 26 sec building on an executor;
- 26 sec total from scheduled to completion.

[Test Result](#) (50 failures / +50)
[Show all failed tests >>>](#)

Module Builds

gameoflife	0,64 sec
gameoflife-build	1,3 sec
gameoflife-cli	0,28 sec
gameoflife-core	3,4 sec
gameoflife-web	15 sec
gameoflife-webservice	0,37 sec

Figure 25. An unstable Build Overview

Step 3. Fix the build

Now fix the error, commit and push the changes to Git again.

Study how Jenkins displays the record of the test failures in the build history.

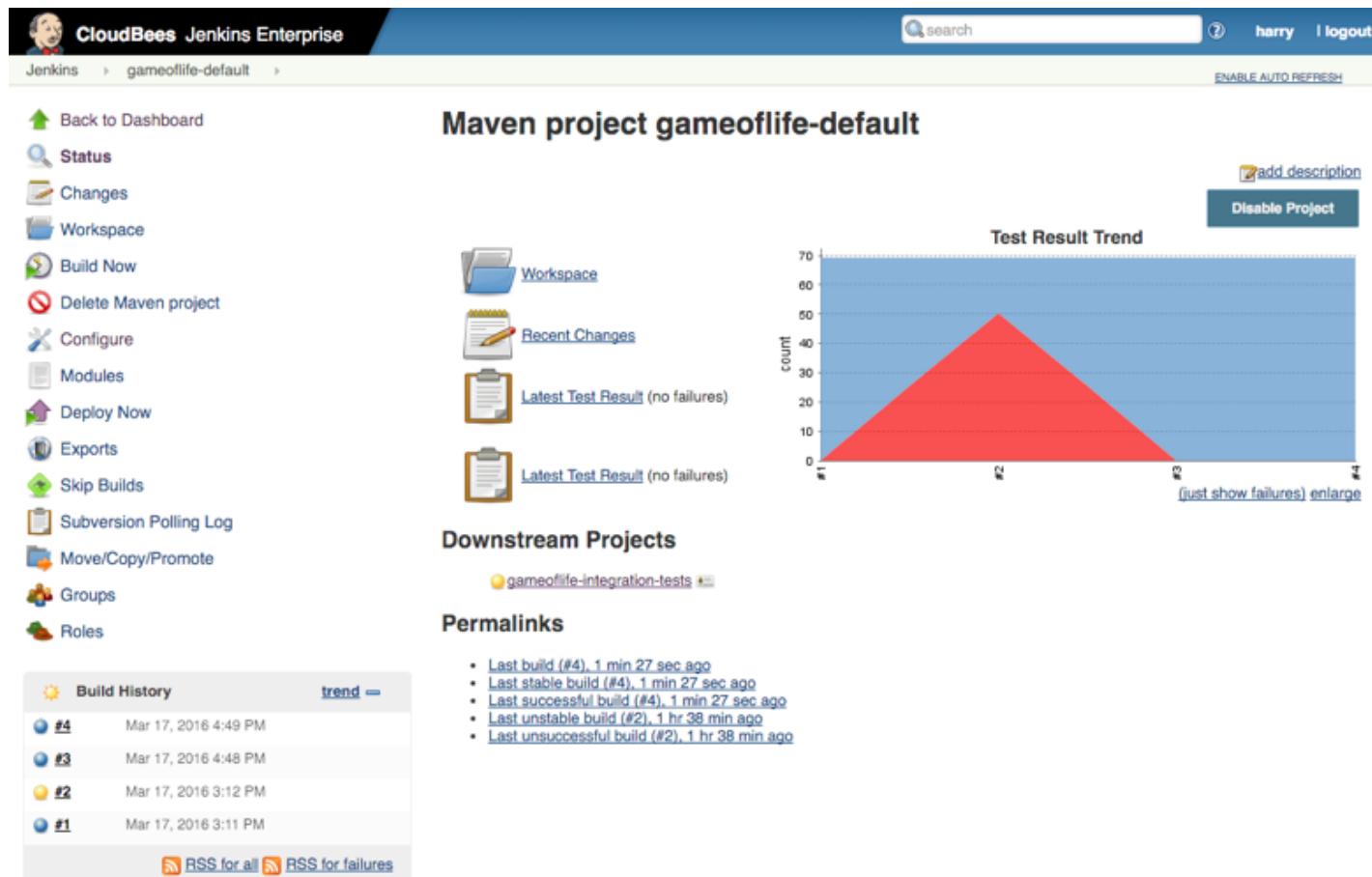


Figure 26. A fixed Build

Go back to slides

Lab 4: Creating an integration tests build

Goal

In this lab, you will learn how to create a second build job that is invoked after an initial one.

This build job will run the web tests in a Firefox browser, rather than with HTMLUnit as is done in the previous labs.

Step 1

In this lab, you will create a job called `gameoflife-integration-tests`. This build will run the `verify` Maven goal to generate a set of HTML reports about the project.

To make things simple, you will just copy your existing `gameoflife-default` project and tweak it:

The screenshot shows the Jenkins interface for creating a new item. On the left, there's a sidebar with various Jenkins management links like New Item, People, Build History, etc. The main area is titled 'New Item' and has a 'Item name' field containing 'gameoflife-integration-tests'. A list of job types is shown with radio buttons: Freestyle project (selected), Maven project, Workflow, Auxiliary Template, Backup, Builder Template, External Job, Folder, Folder Template, Job Template, Long-Running Project, Multi-configuration project, and Publisher Template. Below this is a 'Copy existing Item' section with a radio button selected and a dropdown 'Copy from' set to 'gameoflife-default'. At the bottom right is an 'OK' button.

New Item

Item name `gameoflife-integration-tests`

Freestyle project
This is the central feature of Jenkins. Jenkins will build your project and run tests.

Maven project
Build a maven project. Jenkins takes advantage of your POM file to determine what to build.

Workflow
Orchestrates long-running activities that can span multiple build steps.

Auxiliary Template
Auxiliary templates are used to create nested structures embedded within a job. For example, a "CD" would have an attribute called "tracks" that contain a list of auxiliary templates.

Backup
Performs back up of Jenkins.

Builder Template
Builder template lets you define a custom builder by defining a class that extends the AbstractBuilder class. It also lets you change the definition of the transient properties.

External Job
This type of job allows you to record the execution of a process outside Jenkins. See [the documentation for more details](#).

Folder
Creates a container that stores nested items in it. Useful for organizing items that are in different folders.

Folder Template
A folder template is useful to model a domain-specific concept.

Job Template
A job template is useful to create a large number of jobs that are similar and need to be updated right away.

Long-Running Project
A project which runs a special build step detached from Jenkins.

Multi-configuration project
Suitable for projects that need a large number of different configurations.

Publisher Template
The publisher template lets you define a custom publisher by defining a class that extends the AbstractPublisher class. It also lets you change the definition of the transient properties.

Copy existing Item

Copy from `gameoflife-default`

OK

Figure 27. New Job by Copy Form

Step 2

This build will be triggered immediately after the default build job. In the Build Triggers section:

- Untick **all** the checkboxes of this section
- Select the **Build after other projects are built** checkbox and enter `gameoflife-default` in the corresponding field.

Build Triggers

- Build whenever a SNAPSHOT dependency is built ?
- Trigger builds remotely (e.g., from scripts) ?
- Build pull requests to the repository
- Build when a change is pushed to GitHub
- Build when another project is promoted
- Build after other projects are built ?

Projects to watch

Trigger only if build is stable ?

Trigger even if the build is unstable ?

Trigger even if the build fails ?

- Build periodically ?
- Monitor Docker Hub for image changes ?
- Poll SCM ?

Figure 28. Trigger Build after Another Job

- Change the goals the Maven goal in the **Goals and options** field with the following values (in one line):

```
clean verify -B -U -pl gameoflife-web -Dsurefire.useFile=false  
-Dwebdriver.driver=firefox -Dwebdriver.port=9092 -Djetty.stop.port=9997
```

TIP In this test, we will run the integration tests using Firefox in Jenkins.

Build

Root POM ?

Goals and options ?

Advanced...

Figure 29. Maven Configuration

Now trigger the `gameoflife-default` build manually. This should trigger this new build job, and

result in web tests being executed automatically in Firefox.

Web browser window instances will pop and close, due to the integration test build.

Go back to slides

Lab 5: Integrating with a source repository browser

Goal

In this lab, you will see how Jenkins can integrate with a source code repository browser.

Step 1. Integrate with Gogs

One of the tools installed for this workshop is the Gogs, "A painless self-hosted Git service".

It features a web GUI aimed at browsing repositories.

You can consult this server on the following URL: <http://localhost:5000/gitserver>

TIP

You can configure Jenkins to integrate with Gogs, as well as with other source code repository browsers such like Fisheye, Trac, gitweb, Gitlab, GitHub...

In the **Source Code Management** of the configuration screen for the **gameoflife-default** project, configure the **Repository Browser list** by selecting **Go Git Service (Gogs)**.

Then configure the Repository browser with this URL:

```
http://localhost:5000/gitserver/harry/gameoflife
```

Repository browser Go Git Service (Gogs)

URL http://54.183.97.218/git/harry/gameoflife

Additional Behaviours Add

Figure 30. Configuration for Gogs Repository Browser

Step 2. Making a change

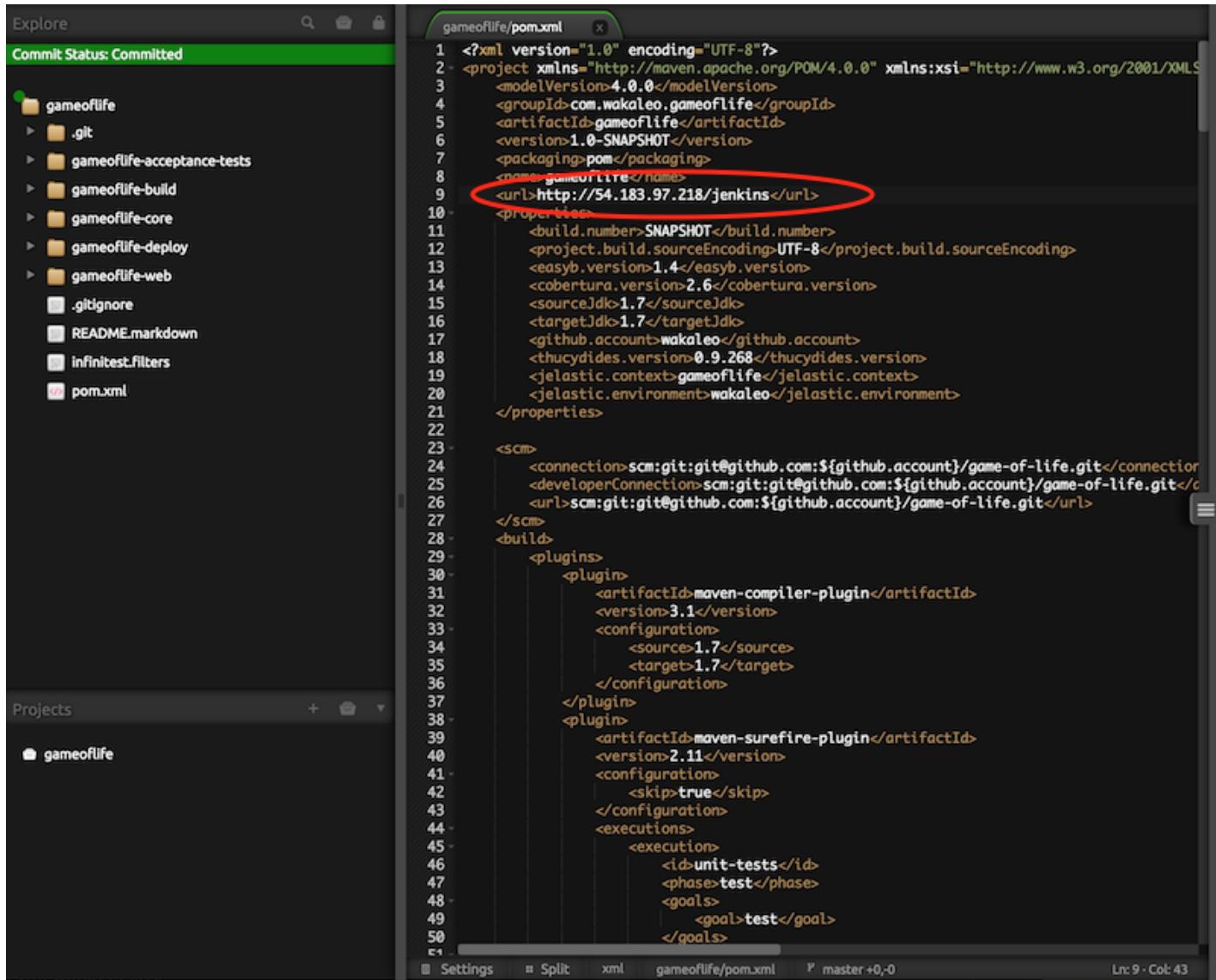
Now make a change to the source code in the gameoflife project and commit your change to your

local Git repository.

Open up Codiad and make one simple (and **harmless**) change. One place where you can do this is in the pom.xml file in the gameoflife directory.

Change the <url> element to point to your Jenkins project, i.e.

<http://localhost:5000/jenkins/gameoflife-default>:



```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.wakaleo.gameoflife</groupId>
  <artifactId>gameoflife</artifactId>
  <version>1.0-SNAPSHOT</version>
  <packaging>pom</packaging>
  <name>gameoflife</name>
  <url>http://54.183.97.218/jenkins</url>
  <properties>
    <build.number>SNAPSHOT</build.number>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <easyb.version>1.4</easyb.version>
    <cobertura.version>2.6</cobertura.version>
    <sourceJdk>1.7</sourceJdk>
    <targetJdk>1.7</targetJdk>
    <github.account>wakaleo</github.account>
    <thucydides.version>0.9.268</thucydides.version>
    <jelastic.context>gameoflife</jelastic.context>
    <jelastic.environment>wakaleo</jelastic.environment>
  </properties>
  <scm>
    <connection>scm:git:git@github.com:${github.account}/game-of-life.git</connection>
    <developerConnection>scm:git:git@github.com:${github.account}/game-of-life.git</developerConnection>
    <url>scm:git:git@github.com:${github.account}/game-of-life.git</url>
  </scm>
  <build>
    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <version>3.1</version>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
        </configuration>
      </plugin>
      <plugin>
        <artifactId>maven-surefire-plugin</artifactId>
        <version>2.11</version>
        <configuration>
          <skip>true</skip>
        </configuration>
        <executions>
          <execution>
            <id>unit-tests</id>
            <phase>test</phase>
            <goals>
              <goal>test</goal>
            </goals>
          </execution>
        </executions>
      </plugin>
    </plugins>
  </build>

```

Figure 31. Editing pom.xml

Save your changes, commit and don't forget to push to the remote Git repository.

TIP If prompted, use harry as both the username and password.

Now watch the build kick off, and click on the **gameoflife-default** build job from the Jenkins home screen to go to the job's details page.

TIP

Direct URL access: <http://localhost:5000/jenkins/gameoflife-default>

From here, click on the **Changes** link in the left-hand menu (3rd item in the menu):

The screenshot shows the Jenkins interface for the 'gameoflife-default' project. The left sidebar has a red circle around the 'Changes' link. The main content area shows a chart titled 'Test Result Trend' with a red triangle pointing upwards, indicating increasing test results over four builds. Below the chart are sections for 'Downstream Projects' (listing 'gameoflife-integration-tests') and 'Permalinks' (listing five build links). A 'Build History' table shows four builds: #4 (Mar 17, 2016 4:49 PM), #3 (Mar 17, 2016 4:48 PM), #2 (Mar 17, 2016 3:12 PM), and #1 (Mar 17, 2016 3:11 PM). RSS feeds for all and failures are also present.

Figure 32. Link to Changes from Job Page

This will take you to a page where you can see the list of changes for each build:

The screenshot shows the Jenkins interface for the 'gameoflife-default' project. On the left is a sidebar with various Jenkins management links. The main content area is titled 'Changes' and displays a list of recent commits. The first commit, '#4 (Jul 20, 2016 3:08:08 PM)', is shown with the message 'harmless pom change' by 'harry / Go Git Service (Gogs)'. Below it are two more commits: '#3 (Jul 20, 2016 2:58:07 PM)' and '#2 (Jul 20, 2016 2:35:07 PM)', each with its own commit message.

- Back to Dashboard
- Status
- Changes**
- Workspace
- Build Now
- Delete Maven project
- Configure
- Modules
- Deploy Now
- Exports
- Skip Builds
- Git Polling Log
- Move/Copy/Promote

Changes

#4 (Jul 20, 2016 3:08:08 PM)

1. harmless pom change — [harry / Go Git Service \(Gogs\)](#)

#3 (Jul 20, 2016 2:58:07 PM)

1. Revert back Cell symbol — [harry / Go Git Service \(Gogs\)](#)

#2 (Jul 20, 2016 2:35:07 PM)

1. Changing Cell symbol — [harry / Go Git Service \(Gogs\)](#)

Figure 33. Changes Page

From here you can:

- Click on a build title to see the Jenkins "diff summary view":

The screenshot shows the Jenkins interface for the 'gameoflife-default' project, specifically the summary view for build #4. The sidebar on the left is identical to Figure 33. The main content area is titled 'Changes' and has a large blue circular icon. Below it is a section titled 'Summary' which lists the commit details for build #4. The commit message 'harmless pom change' is shown with a link to 'details'. A callout box highlights the commit information: 'Commit f59d5288cd71f072b2ca6cad28d8f86060851cc3 by harry' and 'harmless pom change'. Below the summary are links to 'pom.xml' and other Jenkins management links.

- Back to Project
- Status
- Changes**
- Console Output
- Edit Build Information
- Delete Build
- Polling Log
- Deploy Now
- Git Build Data
- No Tags
- Redeploy Artifacts
- Test Result
- See Fingerprints
- Previous Build

Changes

Summary

1. harmless pom change ([details](#))

Commit `f59d5288cd71f072b2ca6cad28d8f86060851cc3` by [harry](#)
harmless pom change

[pom.xml](#)

Figure 34. Summary view

- Or access directly the diff within Gogs, using the "Repository Browser" capability, by clicking the **Go Git Service (Gogs)** or the changeset link in the summary view:

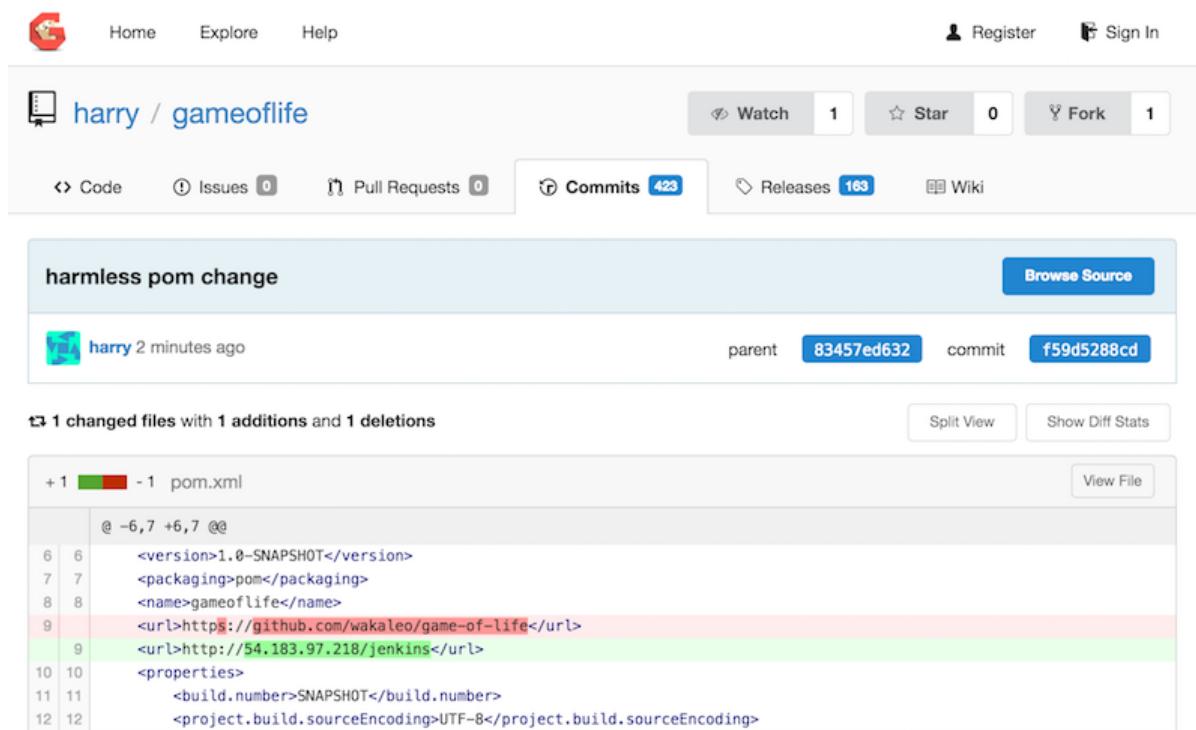


Figure 35. Diff view in Gogs

Go back to slides

Lab 6: Code quality metrics

Goal

In this lab, you will learn how to integrate code quality metrics such as Checkstyle, PMD and FindBugs into your Jenkins builds.

Step 1. Generate the code quality metrics

As we did for the integration tests, you'll have to hit the **New Item** left-hand menu item.

Name your project **gameoflife-metrics**, and select the **Copy from existing job** option. Type in **gameoflife-default** to this field:

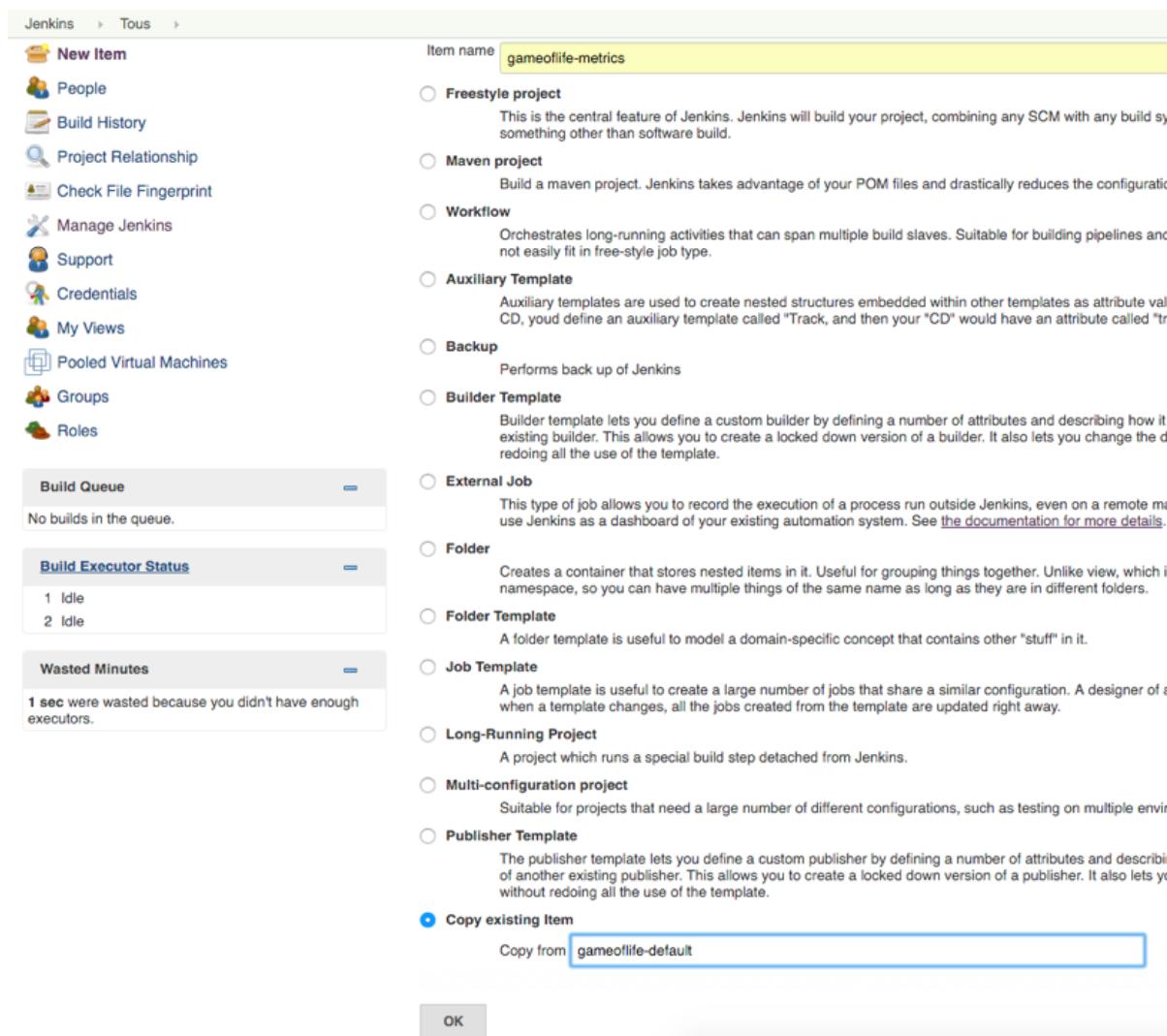


Figure 36. Create Metrics Job by Copying Another

Now we're going to configure this build up to run after the integration tests by configuring the build triggers.

Since this job is copied from the **gameoflife-default** one, you'll first have to delete the **Poll SCM build trigger**.

Now, check the **Build after other projects are built** option and name **gameoflife-integration-tests** as the upstream project:

Build Triggers

- Build whenever a SNAPSHOT dependency is built ?
- Trigger builds remotely (e.g., from scripts) ?
- Build pull requests to the repository ?
- Build when a change is pushed to GitHub ?
- Build when another project is promoted ?
- Build after other projects are built ?

Projects to watch

- Trigger only if build is stable
- Trigger even if the build is unstable
- Trigger even if the build fails

- Build periodically ?
- Monitor Docker Hub for image changes ?
- Poll SCM ?

Figure 37. Job Build Trigger

We want a build job that generates a full set of code quality metrics instead of simply the code coverage ones, so now we'll change the Maven Goal to the following (all on one line):

```
clean install site -Pmetrics -Pcoverage -B -U -Dwebdriver.port=9093  
-Djetty.stop.port=9996
```

IMPORTANT

Code quality metrics need plenty of memory, so click on the **Advanced** button and set MAVEN_OPTS field to -Xmx256m

Build

Root POM pom.xml

Goals and options clean install site -Pmetrics -Pcoverage -B -U -Dwebdriver.port=9093 -Djetty.stop.port=9996

MAVEN_OPTS -Xmx256m

Incremental build - only build changed modules

Disable automatic artifact archiving

Disable automatic site documentation artifact archiving

Disable automatic fingerprinting of consumed and produced artifacts

Enable triggering of downstream projects

Block downstream trigger when building

Build modules in parallel

Use private Maven repository

Resolve Dependencies during Pom parsing

Run Headless

Process Plugins during Pom parsing

Use custom workspace

Maven Validation Level DEFAULT

Settings file Use default maven settings

Global Settings file Use default maven global settings

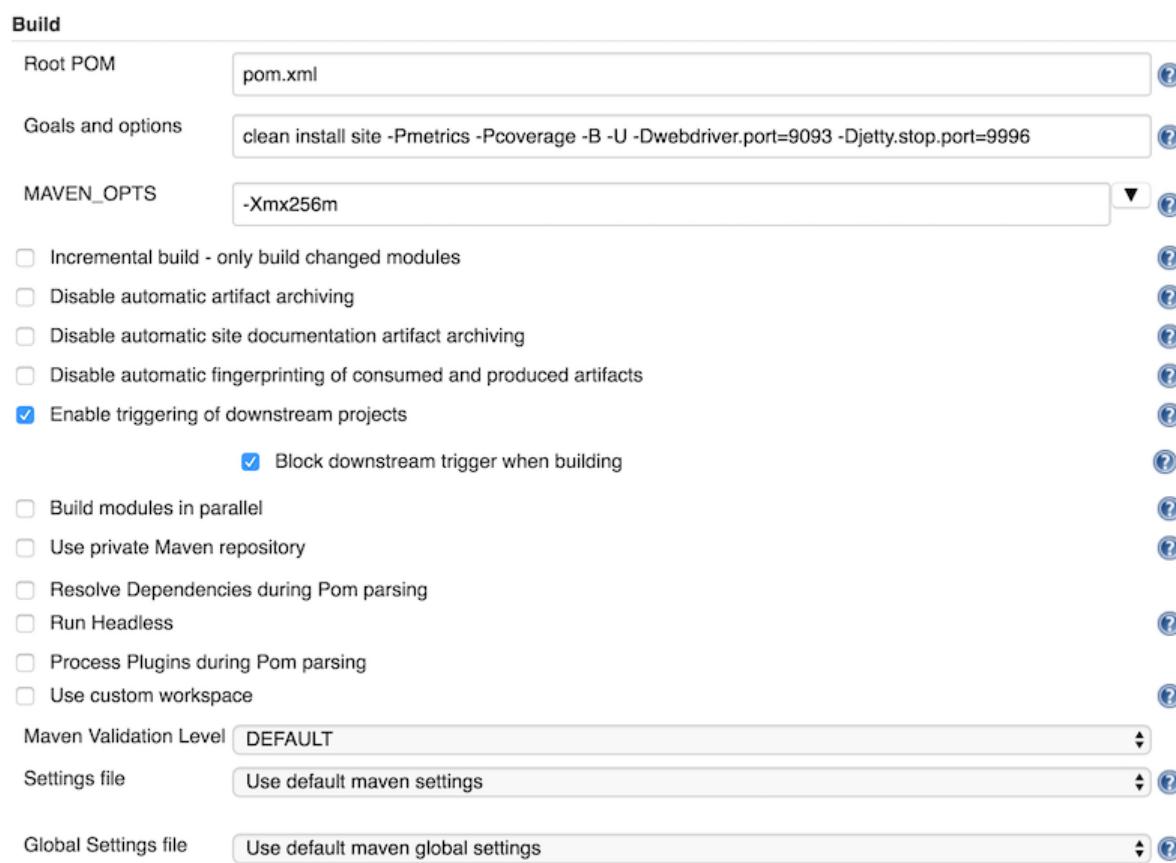


Figure 38. Job Build Configuration

Next, go to the **Build settings** section and tick the **Violations** option. For a Maven project, this plugin knows where to look to find the Checkstyle, PMD and Findbugs XML reports, so you can leave the default settings:

Build Settings

E-mail Notification

violations

☀️ ☔️ ☀️ XML filename pattern

checkstyle	10	999	999	auto	↻	
codenarc	10	999	999	auto	↻	
cpd	10	999	999	auto	↻	
cpplint	10	999	999			
csslint	10	999	999	auto	↻	
findbugs	10	999	999	auto	↻	
fxcop	10	999	999			

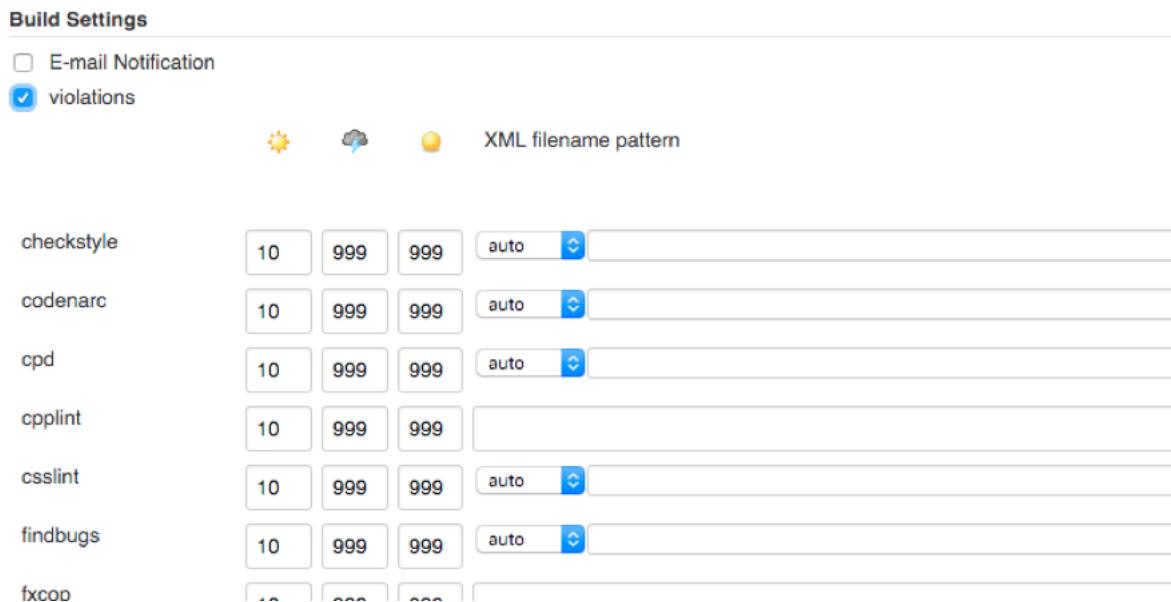


Figure 39. Violations Configuration

The configuration save should have started a first build.

TIP Since we are studying trends, a second build must have been run.

After ensuring that you have 2 builds, whether success or failed, check the project page to see global quality metrics:

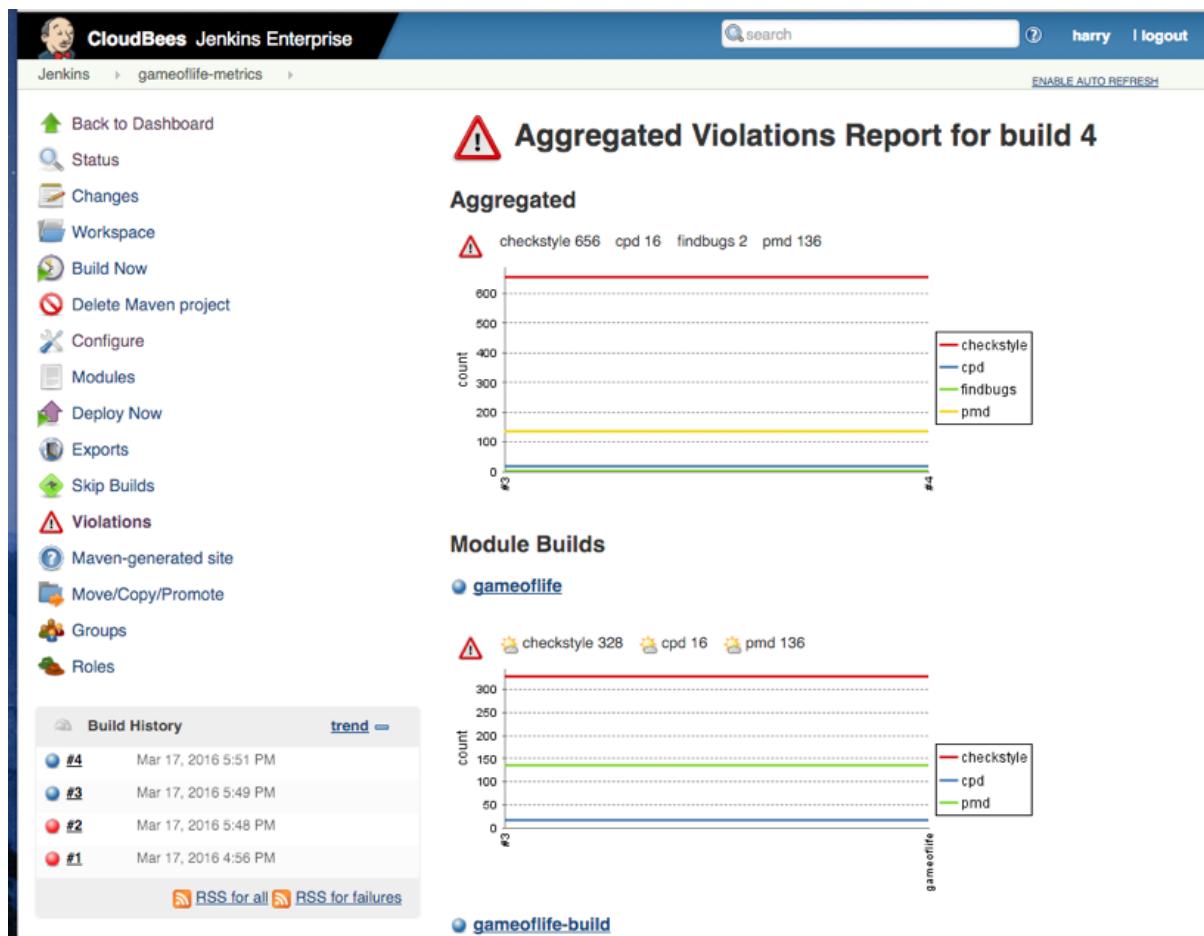


Figure 40. Violations Trend

Drill down into the modules to see how the detailed metrics are reported:

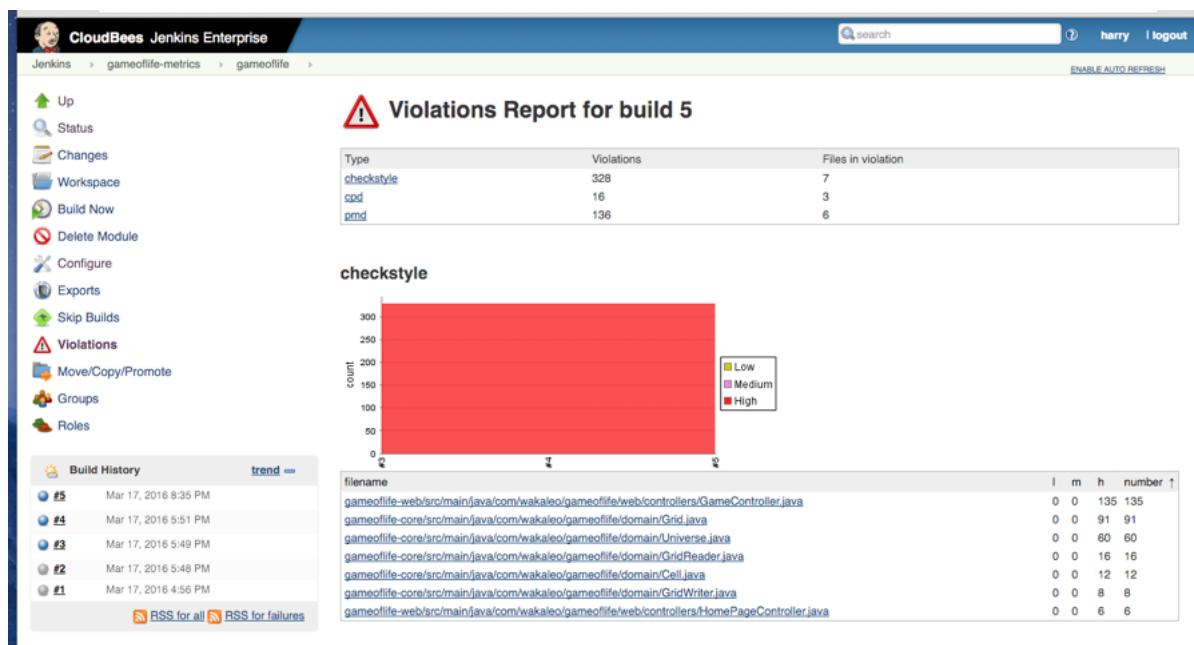


Figure 41. Details metrics

Go back to slides

Lab 7: Code coverage metrics

Goal

In this lab, you will learn how to integrate code coverage into your Jenkins builds.

Step 1. Set up code coverage reporting

We will add code coverage reporting to the **gameoflife-metrics** build job.

The project implements code coverage using JaCoCo (Java Code Coverage), a dedicated library that handles all the data generation and reporting.

IMPORTANT

The JaCoCo plugin is already installed. Do **not** try to update it since latest versions introduced breaking changes.

Triggering the export is done by the `mvn verify` goal, with just a few options, without the `install` nor the `site` goals.

TIP

We will add a post-step Maven invocation, that will clean some compiled classes and then generate the test coverage reports.

Open the build job configuration page, and:

- First, in the section **Post Steps**, add a **post-build step** step:

☒ The type to use is **Invoke top-level Maven targets**.

☒ Select the **maven3** Maven installation

☒ Enter those targets:

```
clean verify -Pmetrics
```

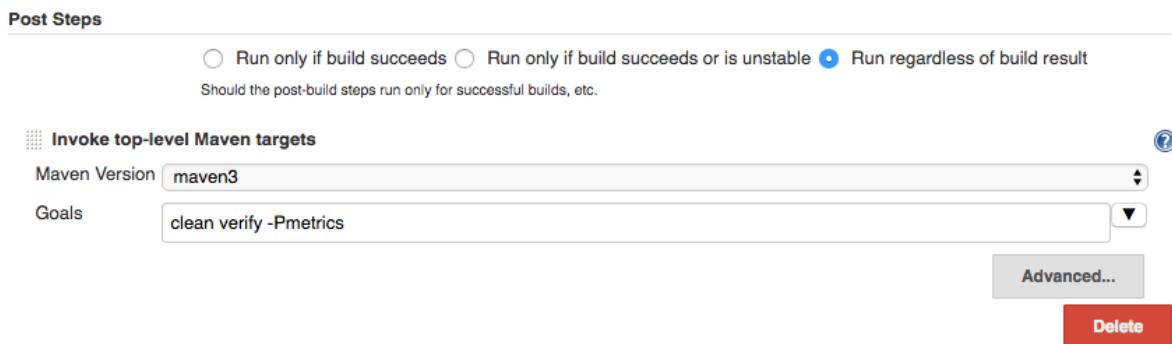


Figure 42. Post-Step Maven for Coverage

TIP

Now, it is time to make Jenkins aware of the generated data to let it publish a report for each build

- Then go to the **Post-build Actions** and select the type **Record JaCoCo coverage report**, with this configuration:

☒ **Path to exec files:** **/target/*.exec

☒ **Path to class directories:** **/target

☒ **Path to source directories:** **/src/main/java

☒ **Inclusions:** **/*.class

☒ **Exclusions:** **/When*

Post-build Actions

Record JaCoCo coverage report

Path to exec files (e.g.: **/target/**.exec, **/jacoco.exec)	Path to class directories (e.g.: **/target/classDir, **/classes)	Path to source directories (e.g.: **/mySourceFiles)
/target/.exec	**/target	**/src/main/java
Inclusions (e.g.: **/*.class)	Exclusions (e.g.: **/*Test*.class)	
**/*.class	**/When*	

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

Change build status according the thresholds

Delete

Figure 43. Post-Build Action Configuration

TIP

This is the JaCoCo settings for our projects. For more information, please read the `pom.xml` file and JaCoCo documentation

Make sure you hit the **Save** button on the bottom of the screen,

Now let the build run, and then consult the new code coverage metrics reports:

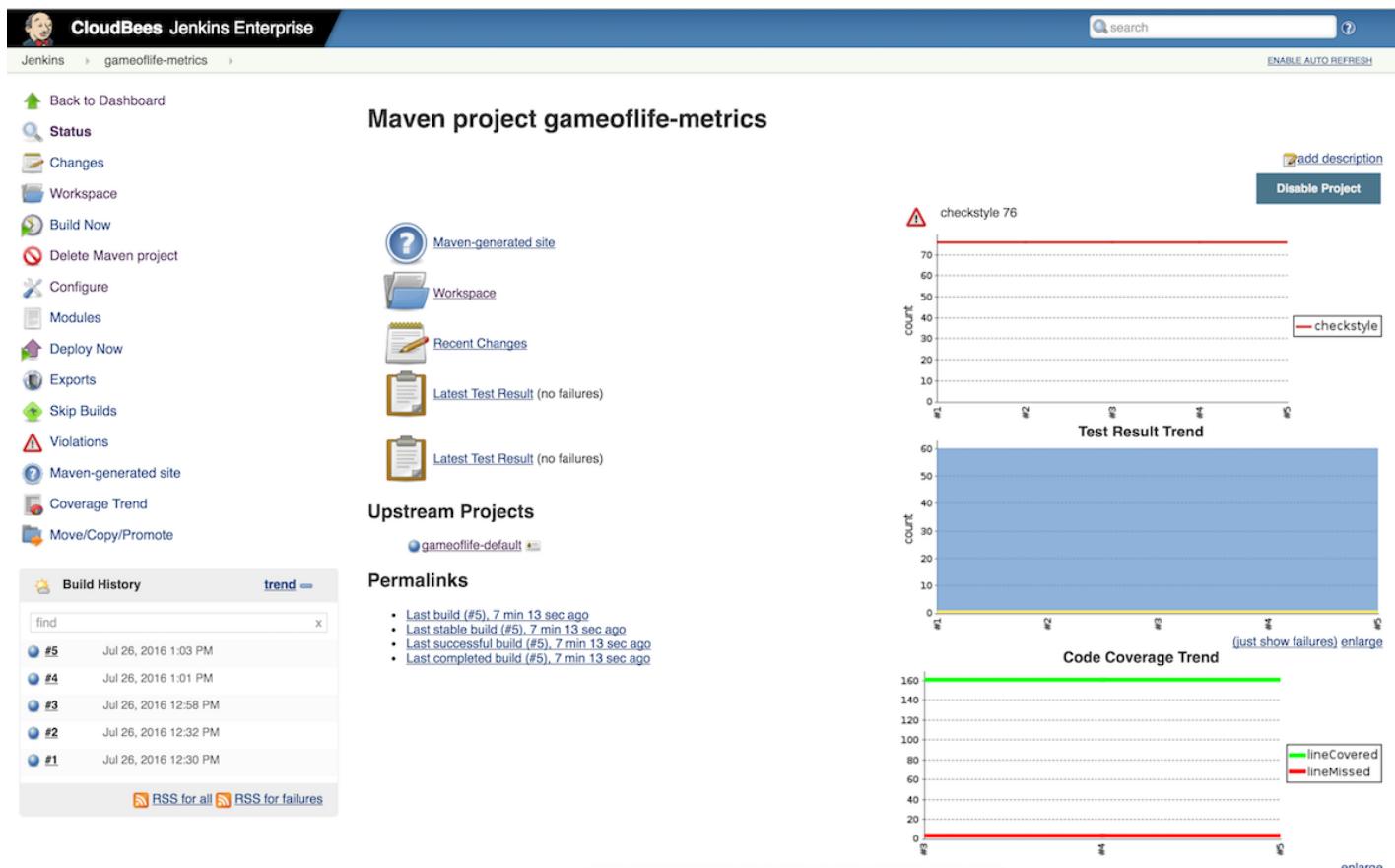
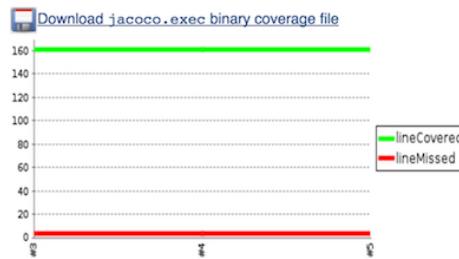


Figure 44. Code Coverage Metrics

Now click on either the graph itself or the **Coverage Report** link on the build's summary page to see the detailed breakdown:

JaCoCo Coverage Report



Overall Coverage Summary

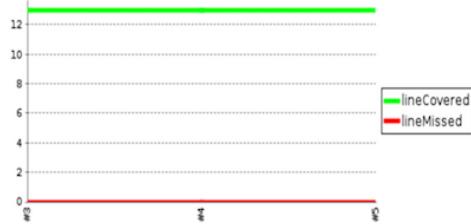
name	instruction	branch	complexity	line	method	class
all classes	98% M: 23 C: 998	94% M: 5 C: 77	95% M: 6 C: 105	98% M: 3 C: 161	99% M: 1 C: 69	100% M: 0 C: 9

Coverage Breakdown by Package

name	instruction	branch	complexity	line	method	class
com.wakaleo.gameoflife.domain	M: 5 C: 624 99%	M: 1 C: 57 98%	M: 2 C: 71 97%	M: 0 C: 122 100%	M: 1 C: 43 98%	M: 0 C: 5 100%
com.wakaleo.gameoflife.webtests.controllers	M: 18 C: 374 95%	M: 4 C: 20 83%	M: 4 C: 34 89%	M: 3 C: 39 93%	M: 0 C: 26 100%	M: 0 C: 4 100%

Figure 45. Code Coverage Metrics Details

Package: Cell



Cell

name	instruction	branch	complexity	line	method
Cell(String, int, String)	M: 0 C: 8 100%	M: 0 C: 0 0%	M: 0 C: 1 100%	M: 0 C: 3 100%	M: 0 C: 1 100%
fromSymbol(String)	M: 0 C: 28 100%	M: 0 C: 4 100%	M: 0 C: 3 100%	M: 0 C: 6 100%	M: 0 C: 1 100%
getSymbol()	M: 0 C: 3 100%	M: 0 C: 0 0%	M: 0 C: 1 100%	M: 0 C: 1 100%	M: 0 C: 1 100%
static (...)	M: 0 C: 26 100%	M: 0 C: 0 0%	M: 0 C: 1 100%	M: 0 C: 2 100%	M: 0 C: 1 100%
toString()	M: 0 C: 3 100%	M: 0 C: 0 0%	M: 0 C: 1 100%	M: 0 C: 1 100%	M: 0 C: 1 100%
valueOf(String)	M: 5 C: 0 0%	M: 0 C: 0 0%	M: 1 C: 0 0%	M: 1 C: 0 0%	M: 1 C: 0 0%
values()	M: 0 C: 4 100%	M: 0 C: 0 0%	M: 0 C: 1 100%	M: 0 C: 1 100%	M: 0 C: 1 100%

Coverage

```

1: package com.wakaleo.gameoflife.domain;
2:
3: public enum Cell {
4:     LIVE_CELL("*"), DEAD_CELL(".");
5:
6:     private String symbol;
7:
8:     private Cell(final String initialSymbol) {
9:         this.symbol = initialSymbol;
10:    }
11:
12:    @Override
    
```

Figure 46. Code Coverage Metrics per Class

[Go back to slides](#)

Lab 8: Parameterized builds

Goal

In this lab, you will learn how to run the integration tests in a different browser, by getting Jenkins to prompt for the browser type when you trigger the build.

Step 1. Check installation state of the Parameterized Triggers plugin

For this lab, we will be using the **Parameterized Trigger** plugin.

It should be already installed, check it inside the **Installed** tabs of the **Managed plugins** section of the global administration.

If this is not the case, install this plugin (without restarting) from the **Available** tab and go back to the top page of Jenkins.

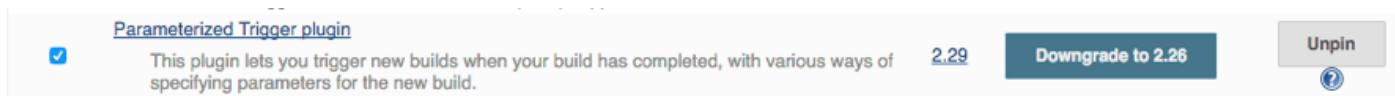


Figure 47. Parameterized Builds Plugin

Step 2. Configuring the Parameterized Triggers plugin

For this lab, we will copy the **gameoflife-integration-tests** build job, and adapt it so that it can be triggered manually. Start by copying this build job. Next, we need to create a parameter for this build job.

Tick on the **This build is parameterized**, and add a Choice parameter as shown here:

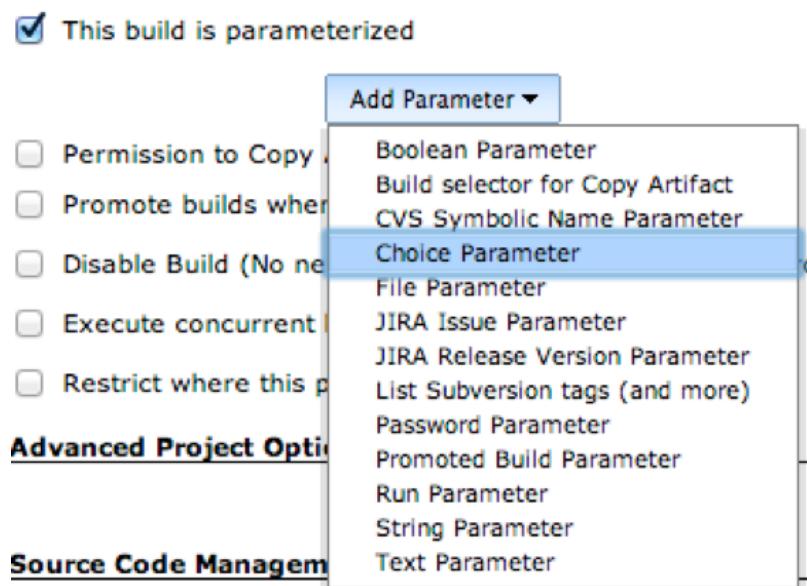


Figure 48. Adding a Choice Parameter

Name it:

Browser

And add those choices:

firefox
htmlunit

This build is parameterized

Choice Parameter

Name	Browser
Choices	firefox htmlunit
Description	The browser the tests will be run in

[Plain text] [Preview](#)

[Delete](#)

Figure 49. Configuring choices for Browser Parameter

Make it a "manual build" by deactivating all of the build triggers:

Build Triggers

- Build whenever a SNAPSHOT dependency is built
- Build after other projects are built
- Poll SCM
- Build periodically



Figure 50. Manual Job Build Triggers

Now pass this parameter into the build as the `webdriver.driver` property, as shown here:

```
clean verify -B -U -pl gameoflife-web -Dwebdriver.driver=${Browser}  
-Dwebdriver.port=9094 -Djetty.stop.port=9996
```

Now start this build manually. Jenkins will prompt for a browser, presenting the list of supported options in a dropdown list:

The screenshot shows the Jenkins interface for a Maven project named "gameoflife-integration-tests". On the left, there's a sidebar with various Jenkins navigation links: Back to Dashboard, Status, Changes, Workspace, Build with Parameters, Delete Maven project, Configure, Modules, and Deploy Now. The main area is titled "Maven project gameoflife-integration-tests". It displays a message: "This build requires parameters:" followed by a dropdown menu labeled "Browser" which has "firefox" selected. Below the dropdown, a smaller note says "The browser the test will be run in". At the bottom right of this section is a large blue "Build" button.

Figure 51. Launch Parameterized Job

The tests should now run on the browser you've chosen.

Go back to slides

Lab 9. Automatic deployments to Tomcat

Goal

In this lab, you will learn how to automatically deploy snapshots to Tomcat.

Step 1: Automatically deploy a SNAPSHOT web application to Tomcat

We are going to modify the **gameoflife-default** build job so that it automatically deploy the gameoflife web application to a Tomcat Application Server after each build.

There is an instance of Tomcat running at <http://localhost:5000/production> which we will use as a test server.

Open the configuration screen of the gameoflife-default build job and go to the **Post-build Actions** section.

Click on the **Add a post-build action** button and then select the **Deploy war/ear to a container** option:

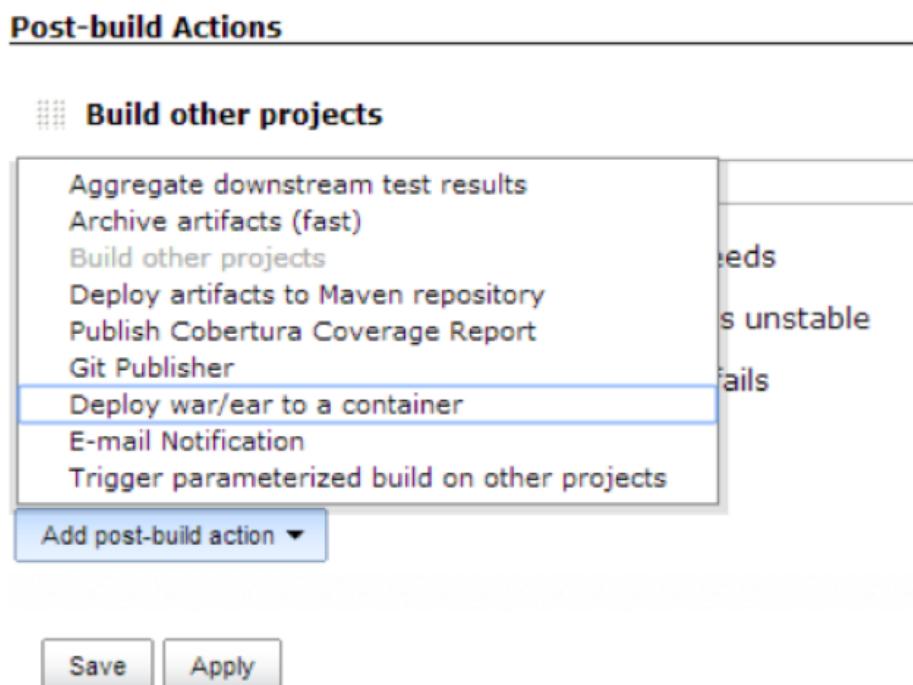


Figure 52. Adding a Post Build Step

Select **Tomcat 7.x** as your container and enter, and type in the **WAR/EAR files** field (see screenshots below):

TIP

If you cannot reach the "Tomcat 7.x", add, drag and drop a 2nd post build action to you will delete afterwards. And check BlueOcean !

```
**/target/*.war
```

TIP

This will deploy any WAR files produced by the builds in any "target" folder.

Now configure the Tomcat instance:

- **URL:** `http://production:8080`
 - ☒ Note this is the *private* address of the Tomcat instance manager
- **Manager username:** admin
- **Manager password:** password

The screenshot shows a configuration form for deploying a WAR file to a Tomcat 6.x container. The fields are as follows:

Deploy war/ear to a container	
WAR/EAR files	<code>**/target/*.war</code>
Context path	
Container	Tomcat 6.x
Manager user name	admin
Manager password
Tomcat URL	<code>http://localhost:8888</code>
Deploy on failure	<input type="checkbox"/>

Figure 53. Deploying WAR to Tomcat - Configuration

TIP

If your build fails and you receive a “401” error in your console output, this is an authentication error and was likely caused by a typo in the password or username fields.

Save the build job and trigger a manual build by hitting the **Build Now** button in the left-hand menu.

Now go to `http://localhost:5000/production/gameoflife/` to see the deployed application:

This is a really cool web version of Conway's famous Game Of Life. The Game of Life is a cellular automaton devised by the British mathematician John Horton Conway way back in 1970.

The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, live or dead. Every cell interacts with its eight neighbors, which are the cells that are directly horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

- Any live cell with fewer than two live neighbours dies, as if caused by underpopulation.
- Any live cell with more than three live neighbours dies, as if by overcrowding.
- Any live cell with two or three live neighbours lives on to the next generation.
- Any dead cell with exactly three live neighbours becomes a live cell.

New Game

Game Of Life version 0.0.68 (build job gameoflife-default - #17)

Figure 54. Application Deployed inside Tomcat

To prove that Jenkins really is deploying the latest snapshot version of the webapp, go and make some cosmetic changes to the **home.jsp** page, that is located within `gameoflife-web/src/main/webapp/WEB-INF/jsp/` folder:

```
ip/WEB-INF/jsp/home.jsp x
1 <%@taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>
2 <%@page import="java.io.InputStream" %>
3 <%@page import="java.io.IOException" %>
4 <%@page import="java.util.Properties" %>
5 <html>
6 <head>
7     <title>The Game Of Life</title>
8     <!-- TODO: Don't inline this -->
9     <style type="text/css">
10        h2 {
11            color: red;
12            font-family: sans-serif;
13        }
14
15        .intro {
16            font-family: sans-serif;
17            background: rgb(200, 200, 200);
18            border: 1px solid black;
19            padding: 8px;
20            margin: 4px;
21            text-align: justify;
22            color: rgb(25, 25, 25)
23        }
24
25        .footer {
26            color: white;
27            font-size: medium;
28            text-align: right;
29            background-color: blue;
30            margin-top: 100px;
31            border-top: thin solid black;
32            padding: 2px;
33            font-family: sans-serif;
34            font-weight: bold;
35        }
36
37        .action-button {
38            border-bottom: 2px solid rgb(100, 100, 100);
39            border-left: 2px solid rgb(100, 100, 100);
40            border-top: 2px solid rgb(150, 150, 150);
41            border-right: 2px solid rgb(150, 150, 150);
42            background: silver;
43            width: 100px;
44            margin: 8px;
45            padding: 4px;
46            text-align: center;
47        }
48
49        a {
50            text-decoration: none;
51        }
52    </style>
53
54    <body>
55        <div class="intro">
56            <h1>The Game Of Life</h1>
57            <p>A Java web application to play Conway's Game of Life</p>
58            <form>
59                <input type="button" value="Play" />
60                <input type="button" value="Reset" />
61            </form>
62        </div>
63
64        <div class="game">
65            <img alt="Game of Life grid visualization" />
66        </div>
67
68        <div class="footer">
69            <a href="#">Home
70            <a href="#">About
71            <a href="#">Contact
72        </div>
73    </body>
74
75</html>
```

Figure 55. Changing Application Index

Save these changes and commit them to Git.

Then wait for the **gameoflife-default** build to complete and refresh the gameoflife URL.

You should now see your cosmetic changes appear.

Go back to slides

Lab 10: Job Organization and Security with Folders

Goal

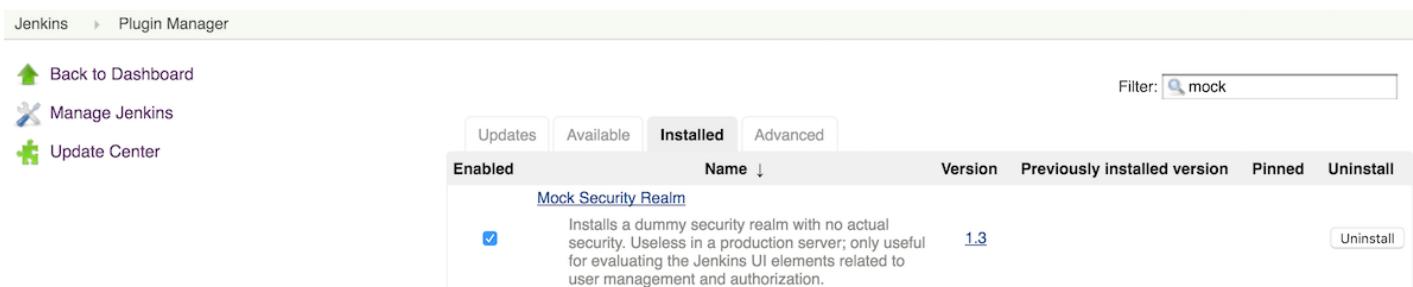
This will cover how to **secure** and **organize** jobs in a Jenkins instance.

Step 1. Securing with RBAC plugin

Step 1.1 Configuring security

We will use the "Mock Security Realm" plugin for this lab. Please ensure that the plugin is installed on your CloudBees Jenkins Enterprise instance, using the "Manage Plugins" option.

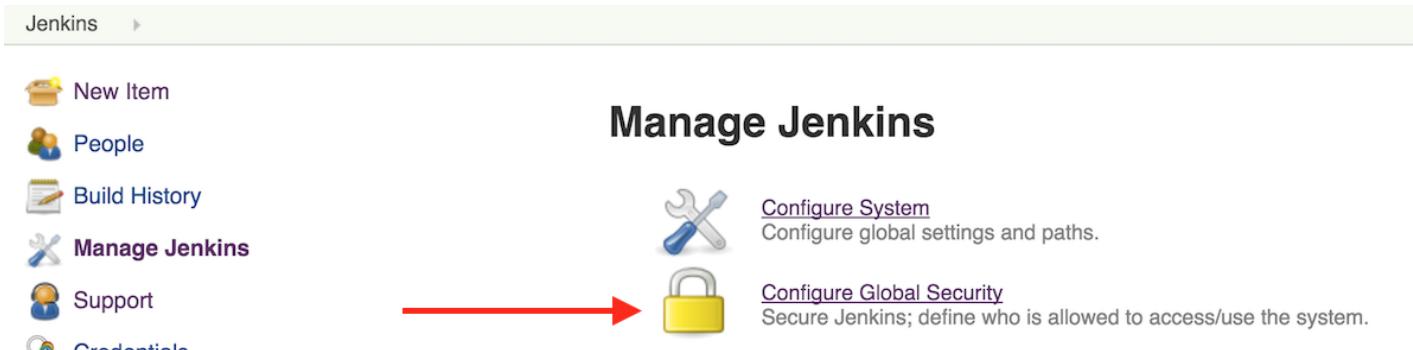
TIP Direct URL access at <http://localhost:5000/jenkins/pluginManager/install>



The screenshot shows the Jenkins Plugin Manager interface. In the top navigation bar, there are links for 'Jenkins' and 'Plugin Manager'. Below the navigation, there are links for 'Back to Dashboard', 'Manage Jenkins', and 'Update Center'. A search bar is present with the text 'mock'. Below the search bar, there are tabs for 'Updates', 'Available', 'Installed' (which is selected), and 'Advanced'. A filter bar shows the text 'mock'. The main table lists the 'Mock Security Realm' plugin, which is enabled. The table columns include 'Enabled', 'Name', 'Version', 'Previously installed version', 'Pinned', and 'Uninstall'. The 'Mock Security Realm' row has a description: 'Installs a dummy security realm with no actual security. Useless in a production server; only useful for evaluating the Jenkins UI elements related to user management and authorization.' An 'Uninstall' button is visible on the right side of the row.

Figure 56. Plugin Mock Security Realm is installed

The first step is to enable security on the CloudBees Jenkins Enterprise master, so click on the "Manage Jenkins" link in the left-hand menu and then on the "Configure Global Security" main menu option.



The screenshot shows the 'Manage Jenkins' page. On the left, there is a sidebar with links: 'New Item', 'People', 'Build History', 'Manage Jenkins' (which is selected and highlighted in blue), 'Support', and 'Credentials'. The main content area is titled 'Manage Jenkins' and contains two items: 'Configure System' (with a wrench icon) and 'Configure Global Security' (with a padlock icon). A red arrow points from the sidebar link 'Manage Jenkins' to the 'Configure Global Security' link in the main content area.

Figure 57. Access to the Configure Global Security

TIP Direct URL access at <http://localhost:5000/jenkins/configureSecurity/>

Configure security with those settings:

- Select the **Enable Security** checkbox
 - Select the **Mock Security Realm** as the security realm.
- ☒ Use this content for the **Users** and **Groups** text-field:

```
harry admin-ext
sally developer-ext
barry
```

IMPORTANT Mock Security Realm will simulate an external LDAP or Active Directory populated with this content. Left column is the username and right column is the group's name.

TIP This will create 3 users: **harry**, **sally** and **barry**. Jenkins will also recognize **admin-ext** and **developer-ext** as "external" groups and the related users' membership

- Select **Role-based Matrix Authorization Strategy** as the authorization policy.

Configure Global Security

Enable security

TCP port for JNLP slave agents Fixed : Random Disable

Disable remember me

Access Control

Security Realm

- Active Directory
- Delegate to servlet container
- Google Apps SSO (with OpenID)
- Jenkins' own user database
- LDAP
- Mock Security Realm

Users and Groups

harry	admin-ext
sally	developer-ext
barry	

Advanced...

Authorization

- Anyone can do anything
- Legacy mode
- Logged-in users can do anything
- Matrix-based security
- Project-based Matrix Authorization Strategy
- Role-based matrix authorization strategy

Import strategy Retain any existing role-based matrix authorization strategy configuration

Figure 58. Global Security configuration

Now save your security settings by clicking the **Save** button and return to the top-level of CloudBees Jenkins Enterprise.

You will now see a **log in** link in the top right-hand corner, so click on that and log in as **harry**.

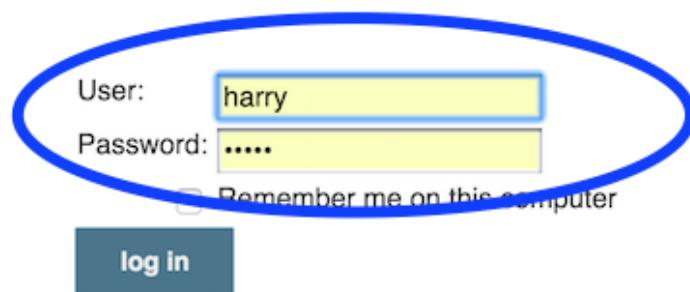


Figure 59. Jenkins Log In

TIP Password have the same value as usernames.

Step 1.2. Creating roles

Now we will need to create roles for our mock imported users. From the top-level screen, click on the **Roles** link in the left-hand menu. Now click on the **Manage** link in the left-hand menu and you will be taken to the "Manage Roles" screen.

TIP This link is now available because you selected the "Role-based Matrix Authorization Strategy" option in the last section.

TIP You have a direct-URL access: <http://localhost:5000/jenkins/plugin/nectar-rbac/manage/>

Figure 60. Accessing Manage Roles page

Our goal here is to create a security set up that has the following characteristics:

- Anonymous users have no access whatsoever, and they need to login first before even seeing the CloudBees Jenkins Enterprise top page.
- Once logged in, users have generic read access.
- Jobs can assign some users to the **reader** role that has read-only access to jobs, their test results, build results, etc.
- Jobs can assign some users to the **developer** role, who can start a new build, configure jobs, and so on.
- A few people will be in the **admin** role, which gets irrevocable full access to Jenkins.

This maps to the **two pre-defined system roles** (anonymous and authenticated) and **three additional roles we had to create** (reader, developer and admin).

To create those additional roles, type in the titles to the "Role to add" field on the "Manage Roles" page and then hit the add button:

The screenshot shows the Jenkins 'Manage Roles' interface. At the top, there's a title 'Manage Roles' with a hat icon. Below it is a table with columns for 'Role' (empty), 'Overall' (checkboxes for Delete, Create, Connect, Configure, Build, RunScripts, UploadPlugins, Administer, Read, Configure, UpdateCenter), and 'Slave' (checkboxes for Administer, Read, Configure, UpdateCenter). Three rows represent existing system roles: 'anonymous' (checked for Delete, Create, Connect, Configure, Build, RunScripts, UploadPlugins, Administer, Read), 'authenticated' (checked for Delete, Create, Connect, Configure, Build, RunScripts, UploadPlugins, Administer, Read, Configure, UpdateCenter), and 'admin' (checked for Delete, Create, Connect, Configure, Build, RunScripts, UploadPlugins, Administer, Read, Configure, UpdateCenter). At the bottom, there's a red oval highlighting a search bar labeled 'Role to add: developer' and a 'Save' button. To the right of the search bar is an 'Add' button.

Figure 61. Quickly add a new role

By default, **filterable** property will be set to true for them all. As such, you'll now need to edit each role's

permissions. Note that the first permissions column shows the name of the **section** the permission is for, while the second column shows the name of the **checkbox** in that section:

Table 1. Roles and Permissions table

Role	Permissions (section, checkbox)		Filterable
anonymous	None	None	Yes
authenticated	Overall	Read	Yes
reader	Overall Job	Read Read	Yes
developer	Job	Read Create Configure Build Workspace	Yes
admin	Overall	Administer	No

Once you are ready to edit the "authenticated role", note that you'll be able to uncheck all boxes at once

by scrolling to the right and clicking on the button.

Role	Filterable	Overall	Slave	Metrics	Support	Group	Role	Run	Credentials	Job	Alerts	SCM	View	Client/Managed Master	Update center	VMWare Pools
anonymous	<input checked="" type="checkbox"/>															
authenticated	<input checked="" type="checkbox"/>															
reader	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" 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 type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
admin	<input type="checkbox"/>	<input checked="" 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 checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>
developer	<input checked="" 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 type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Figure 62. Resulting Permission Matrix

Try to save your configuration.

Note that the "Anti-lockout" guard will kick in and change our configuration, giving "authenticated" overall Administer permissions.



Manage Roles

 **A configuration was submitted which would have resulted in no users with administrative permissions.**

In order to prevent such an administrative lockout, the configuration was modified to that below. This warning will remain until either:

- the instance is restarted, or
- a configuration which does not result in an administrative lockout is submitted.

IMPORTANT

To avoid an administrative lockout, you must ensure that either:

- One of the system roles (anonymous or authenticated) has the **Overall Administer** permission; or
- A [root group](#) with at least one member has been assigned at least one role which has the **Overall Administer** permission.

Note: it is not possible to determine if the members of such a group are valid users, so the check that triggered this warning cannot prevent all administrative lockouts, just the obvious ones.

This is because the "admin" role is not yet associated with a group that contains at least one member. As such, saving the configuration as-is would have locked you out of the instance.

Now let's save these altered roles "as-it" by clicking the "Save" button at the bottom of the screen.

Step 1.3. Creating groups

The next step is to create groups that connect users to the roles, and populate these groups. We will create 3 groups:

- Developers
- Administrators
- Browsers

Click on the "Groups" link in the left-hand menu, which will take you to the "Groups" list. Then, click on the "add some?" link to start creating groups (see below for settings).

The screenshot shows the Jenkins interface with the 'Groups' option selected from the left sidebar. The main content area is titled 'Defined in Jenkins' and contains a table with two columns: 'Name' and 'Membership'. A message at the bottom of the table says 'None defined yet. Want to [add some?](#)' with a red oval circling the 'add some?' link. Below the table, it says 'Icon: S M L'.

- New Item
- People
- Build History
- Manage Jenkins
- Support
- Credentials
- My Views
- Groups**
- New Group
- New Quick Group
- Roles
- Pooled Virtual Machines
- Cluster Operations

Figure 63. Access to group creation

We want to create groups to represent each role we have created, and we also want to add both our imported users *and* our imported groups as members of each group.

Table 2. Groups final configuration

Group Name	Roles	Propagated	Members
Browsers	reader	Yes	barry, harry, sally
Developers	developer	Yes	sally, developer-ext
Administrators	admin	Yes	harry, admin-ext

For example, "Developer" would be configured like this:

Developers

 [add description](#)

Roles

[developer](#)

Members

Name	Actions
 developer-ext	
 sally	

Icon: [S](#) [M](#) [L](#) [Add user/group](#)

Permissions

Permission	On Jenkins	On children	On grandchildren	Beyond
Job / Build	Granted	Propagates	Propagates	Propagates
Job / Cancel	Granted	Propagates	Propagates	Propagates
Job / Configure	Granted	Propagates	Propagates	Propagates
Job / Create	Granted	Propagates	Propagates	Propagates
Job / Discover	Granted	Propagates	Propagates	Propagates
Job / Read	Granted	Propagates	Propagates	Propagates
Job / Request	Granted	Propagates	Propagates	Propagates
Job / Workspace	Granted	Propagates	Propagates	Propagates
Client/Managed Master / Configure	Granted	Propagates	Propagates	Propagates
Client/Managed Master / Lifecycle	Granted	Propagates	Propagates	Propagates

Figure 64. Developers Group configuration

TIP

Once you have created a group, use the [Add user/group](#) button to add users and sub-groups to the current group.

If you want to create another group, hit the [Back to Groups](#) button in the left-hand menu and then hit the "New Group" button

Repeat this process until all 3 groups have been created

Step 1.4. Completing configuration of the roles

Now that we have a group with the Overall/Administer permission, we can complete the configuration of the roles.

Go back to the **Manage Roles** screen (**Manage Jenkins , Manage Roles**). De-select the Overall/Admin permission for the **authenticated** role and ensure that the Overall/Read permission is still on selected.

Now save the configuration. Since the **admin** role is now associated with a populated group, you should not receive an **Anti-lockout** warning.

Step 1.5. Single sign-on

Ensure that the users **harry** and **sally** exist and that the current logged user is **harry**.

Step 2. Organizing with Folders Plus Plugin

Step 2.1. Creating Folders

The first step is to create a folder, which is a dashboard-level object in Jenkins.

To start, log into CloudBees Jenkins Enterprise and click on the **New Item** link in the left-hand menu.

Now select the **Folder** type for the job and name the folder **Harry's Project**:

Item name	Harry's Project
<input type="radio"/> 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.
<input type="radio"/> Maven project	Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
<input type="radio"/> Workflow	Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines and/or organizing complex activities that do not easily fit in free-style job type.
<input type="radio"/> Auxiliary Template	Auxiliary templates are used to create nested structures embedded within other templates as attribute values. For example, if you are modeling a CD, you'd define an auxiliary template called "Track, and then your "CD" would have an attribute called "tracks" that contain a list of "Track"s.
<input type="radio"/> Backup	Performs back up of Jenkins
<input type="radio"/> Builder Template	Builder template lets you define a custom builder by defining a number of attributes and describing how it translates to the configuration of another existing builder. This allows you to create a locked down version of a builder. It also lets you change the definition of the translation without redoing all the use of the template.
<input type="radio"/> External Job	This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See the documentation for more details .
<input checked="" type="radio"/> 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.

Figure 65. Creating a New Folder

Once this is done, let the default configuration fields and click **Ok** to save.

Now at the top page or dashboard-level of Jenkins, you should see the **Harry's Project** folder in the default view, along with any other configured jobs:

All	W	Name ↓	Last Success
		fast-archive-job	3 days 16 hr - #8
		Harry's Project	N/A
		testJob	15 days - #1

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

Figure 66. Harry's Project in Default View

To edit the configuration of this folder, either click on the folder and the **Configure** link in the left-hand menu or click on the black arrow to the right of the folder's name and the **Configure** link in that drop-down menu:

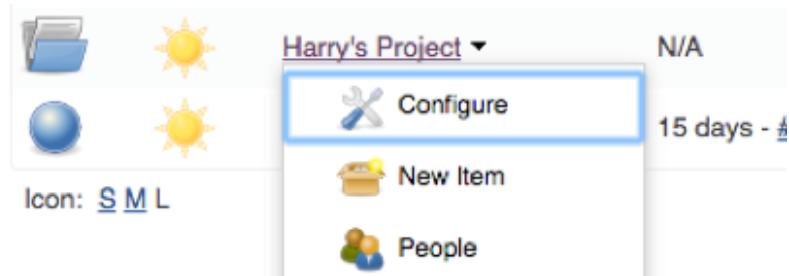


Figure 67. Configure Harry's Project from Default View

The Folders plugin allows users to organize jobs into folders, but the **CloudBees Folders Plus** plugin expands this functionality to also allow folders to:

1. control agents
2. show an overall health report for jobs in a folder
3. display custom folder icons
4. contain environment variables, which are accessible to jobs
5. restrict the content of folders to certain item types
6. inherit global roles and group permissions, or have a custom group with folder-specific permissions

Step 2.2. Create security on folders

To create a custom set of permissions only on this folder, we will need to create a new group for this folder.

To do this, click the **Groups** link in the left-hand menu of the **Harry's Project** folder configuration screen:

Name	Membership
None defined yet. Want to add some?	

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

Figure 68. Harry's Project Group Configuration

As you can see, there are not yet any groups configured on this folder, but the **Folders Plus** plugin is giving us the option of creating one. In the **Inherited from Jenkins** section below, you will also see the names and members of all groups configured at the root level of Jenkins.

For now, just click either the **Add some?** link in the Groups list of the **New Group** link in the left-hand menu of this group configuration screen.

We are now creating a new group specific to **Harry's Project**, so on the next screen configure the Group name to be **ProjectDevelopers** and click **Ok**.

You'll see a screen prompting for a description and assign to the group any role configured at the root level of Jenkins. Since this is a developers group for this folder, select the **developer** role and click **save** to continue.

On the next page we can add group members, so for now add **harry** and **barry** to this group:

	barry
	harry

Figure 69. Harry and Barry are in the Project

Normally, permissions granted closer to the root gets inherited through to folders and jobs in them. When we activate a **require explicit assignment** filter, we are essentially saying that users explicitly need to be assigned to a group in this folder to be able to get a role.

Now go back to the **Harry's Project** folder and click on the **Roles** link in the left-hand menu, then click the **Filter** link and mark all roles as requiring explicit assignment:

Filter roles inherited from Jenkins

Role	Require explicit assignment
anonymous	<input checked="" type="checkbox"/>
authenticated	<input checked="" type="checkbox"/>
developer	<input checked="" type="checkbox"/>
reader	<input checked="" type="checkbox"/>

Figure 70. Filter Roles

In practice, this means that **sally**, who has the developer role to the root, will not get the developer role in the **Harry's project** folder unless she is explicitly added to the **ProjectDevelopers** group.

barry, who only has the authenticated role in the root has been given explicit permission on the **Harry's Project** folder as a developer, and so should be able to have just as many permissions on this folder as **harry**.

Logout as **harry** and login as **barry**. Verify that you can create and build jobs, but only in the **Harry's Project** folder and that any other jobs in the root level of Jenkins are hidden from view.

Logout as **barry** and login as **sally**. Verify that you cannot see the folder **Harry's Project**. Try to access the project directly using the URL, e.g. <http://localhost:5000/jenkins/job/Harry's%20Project/> and confirm that you get a 404 error page.

Now logout as **sally** and log back in as *harry.

Step 2.3. Move jobs into folders

Since we just created **Harry's Project**, this folder is empty. To populate it with jobs, lets create jobs into the root of Jenkins and move them into this folder.

Go back to the top page of Jenkins, either by click on the CloudBees Jenkins Enterprise banner or the **Jenkins** breadcrumb in the navigation bar.

Now create 3 jobs of any type (freestyle, Maven, Pipeline, etc) and name them anything while leaving their configurations blank.

Once done, click on either the job's name or the black arrow to the right of the job's name.

In either menu, there will be an option called **Move** with a icon of a dolly.

Select the **Move** option and select **Jenkins - > Harry's Project** from the dropdown menu as the destination for your job.

Repeat this for all of the dummy jobs you just created.

Now open up the **Harry's Project** folder: all jobs that you just moved should now be listed there.

Go back to slides 

Lab 11: Validated merge

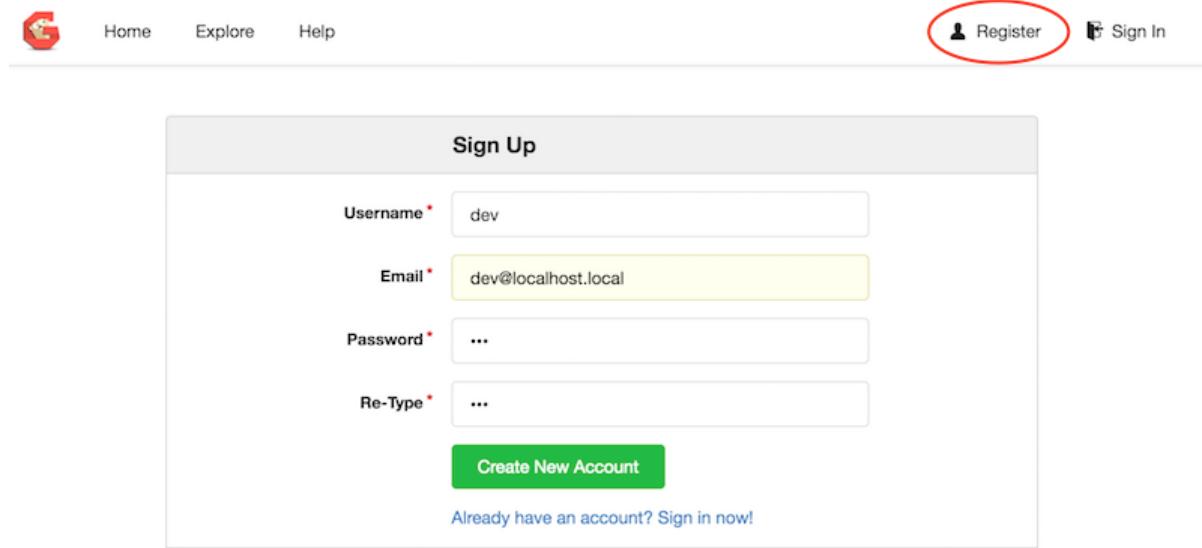
Goal

In this lab, you will learn how to set up Jenkins for validated merge with Git as an administrator, then how to perform actual validated merge as a developer.

Step 1: Fork project in Gogs as "dev" user

We will use the Gogs service for this exercise. A new "user" is required: **dev**, different than **harry**. This new user will fork **harry**'s project and change things inside, to have a different workflow than before:

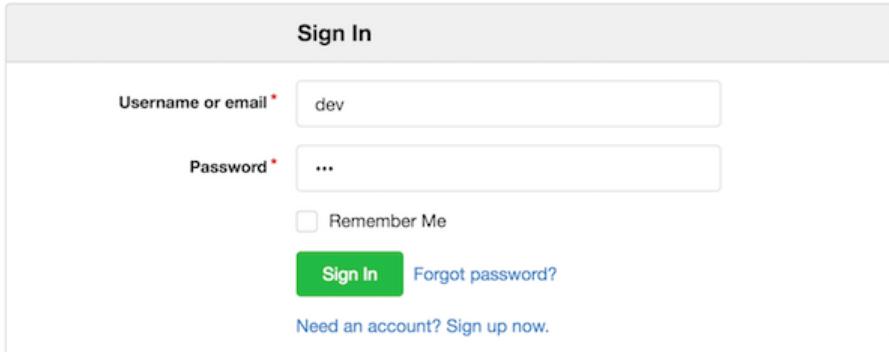
- Navigate to the Gogs service, on <http://localhost:5000/gitserver>
- Create the **dev** user by clicking the **Register** link in the top right, and fill the form like this:
 - ☒ **Username:** dev
 - ☒ **Email:** dev@localhost.local
 - ☒ **Password:** dev



The screenshot shows the Gogs registration interface. At the top, there are links for Home, Explore, and Help. On the right, there are 'Register' and 'Sign In' buttons. The 'Register' button is circled in red. Below the header is a 'Sign Up' form with four fields: 'Username' (dev), 'Email' (dev@localhost.local), 'Password' (redacted), and 'Re-Type' (redacted). A green 'Create New Account' button is at the bottom of the form. A small note at the bottom says 'Already have an account? Sign in now!'

Figure 71. Register as dev

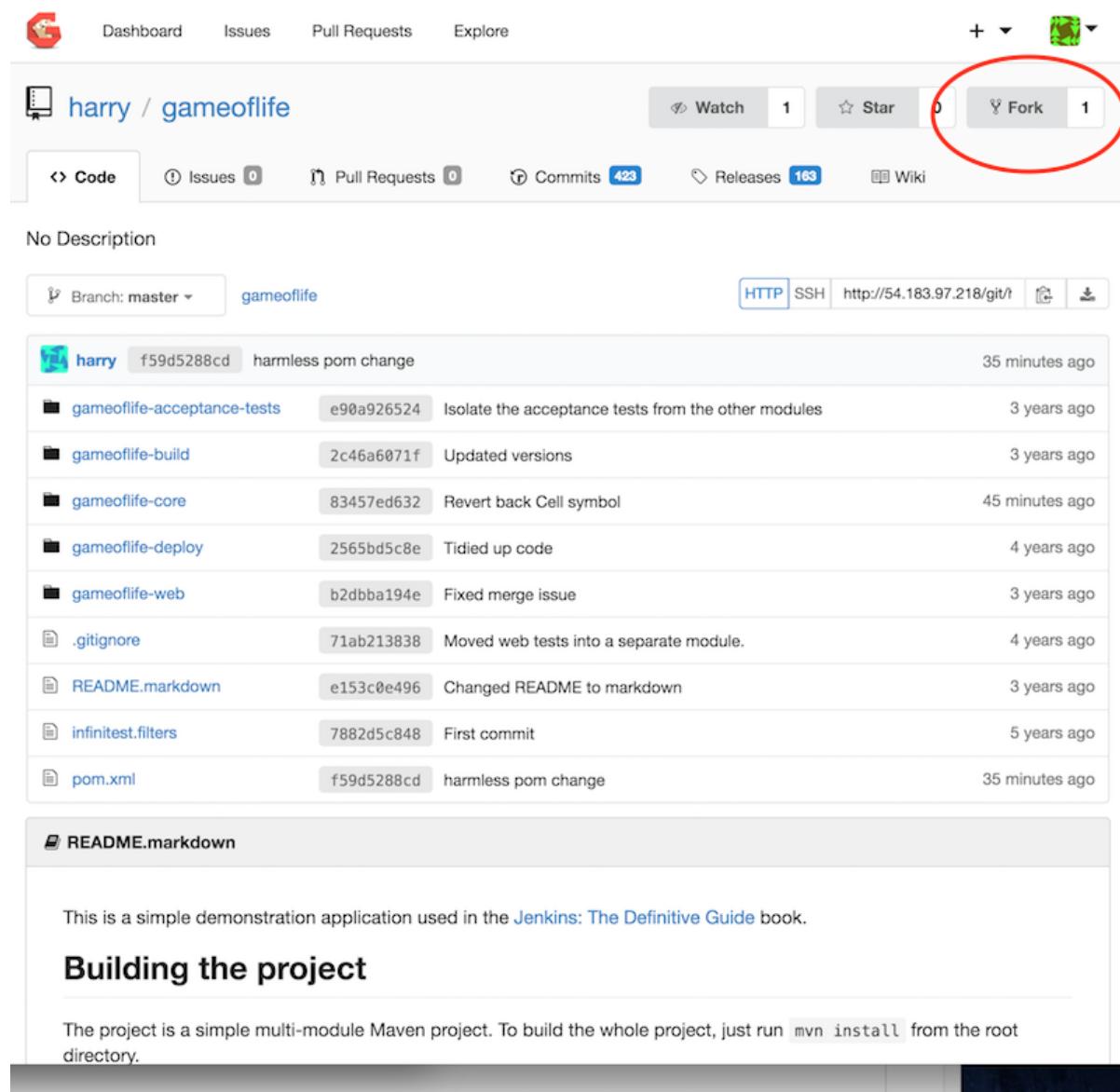
- Then, sign-in as **dev** user, to grant the right of forking the project:



The screenshot shows a 'Sign In' page. At the top, there are navigation links: Home, Explore, Help, Register, and a red-circled 'Sign In' button. The 'Sign In' form itself has fields for 'Username or email*' (containing 'dev') and 'Password*' (containing '...'). There is also a 'Remember Me' checkbox and a 'Sign In' button (which is green). Below the form, there is a link to 'Forgot password?' and a note 'Need an account? Sign up now.'

Figure 72. Signing-in as dev

- Once you are logged-in under your GitHub account, browse to the **harry**'s project:
<http://localhost:5000/gitserver/harry/gameoflife>
- Then, create a fork of this project in your "dev" account namespace, that we will use for this Labs:



The screenshot shows a Gogs project page for 'harry / gameoflife'. At the top, there are navigation links for Dashboard, Issues, Pull Requests, and Explore. On the far right, there are buttons for Watch (1), Star (0), Fork (1), and a user profile icon. The 'Fork' button is circled in red.

Below the header, there's a summary bar with links for Code, Issues (0), Pull Requests (0), Commits (423), Releases (163), and Wiki.

No Description

Branch: master gameoflife

HTTP SSH http://54.183.97.218/git/

Commits (423)

Author	Commit ID	Message	Time Ago
harry	f59d5288cd	harmless pom change	35 minutes ago
gameoflife-acceptance-tests	e90a926524	Isolate the acceptance tests from the other modules	3 years ago
gameoflife-build	2c46a6071f	Updated versions	3 years ago
gameoflife-core	83457ed632	Revert back Cell symbol	45 minutes ago
gameoflife-deploy	2565bd5c8e	Tidied up code	4 years ago
gameoflife-web	b2dbba194e	Fixed merge issue	3 years ago
.gitignore	71ab213838	Moved web tests into a separate module.	4 years ago
README.markdown	e153c0e496	Changed README to markdown	3 years ago
infinitest.filters	7882d5c848	First commit	5 years ago
pom.xml	f59d5288cd	harmless pom change	35 minutes ago

README.markdown

This is a simple demonstration application used in the [Jenkins: The Definitive Guide](#) book.

Building the project

The project is a simple multi-module Maven project. To build the whole project, just run `mvn install` from the root directory.

Figure 73. Fork the Project in Gogs

- When asked, leave the default options:

New Fork Repository

Owner * dev

Fork From harry/gameoflife

Repository Name * gameoflife

Visibility This repository is Private
You cannot alter the visibility of a forked repository.

Description

Fork Repository Cancel

Figure 74. Fork options

- Write down the **HTTP URL** of the newly **forked repository**:

Dashboard Issues Pull Requests Explore + ⌂

dev / gameoflife forked from harry/gameoflife

Code Commits 425 Releases 163 Settings

No Description

Branch: master gameoflife

HTTP SSH http://54.183.97.218/git/c

Red circle highlights the HTTP link: <http://54.183.97.218/git/c>

Commit	Message	Date
dev a7c9e7d809	Going back	2 hours ago
gameoflife-acceptance-tests e90a926524	Isolate the acceptance tests from the other modules	3 years ago
gameoflife-build 2c46a6071f	Updated versions	3 years ago
gameoflife-core 118e088d54	Restored working tests	3 years ago
gameoflife-deploy 2565bd5c8e	Tidied up code	4 years ago

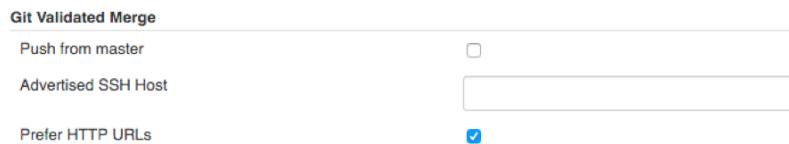
Figure 75. Get the Fork HTTP URL

Step 2: Global Jenkins Administration Setup

Navigate to the Jenkins global settings (**Manage Jenkins - Configure System**) and check those global

settings:

- In the **GitHub Validated Merge** section, enable the **Prefer HTTP URLs** option:



The screenshot shows the 'Git Validated Merge' configuration section. It includes fields for 'Push from master' (unchecked), 'Advertised SSH Host' (empty text field), and 'Prefer HTTP URLs' (checked). A legend at the bottom indicates that checked boxes are blue.

Figure 76. Global Configuration for Validated Merge

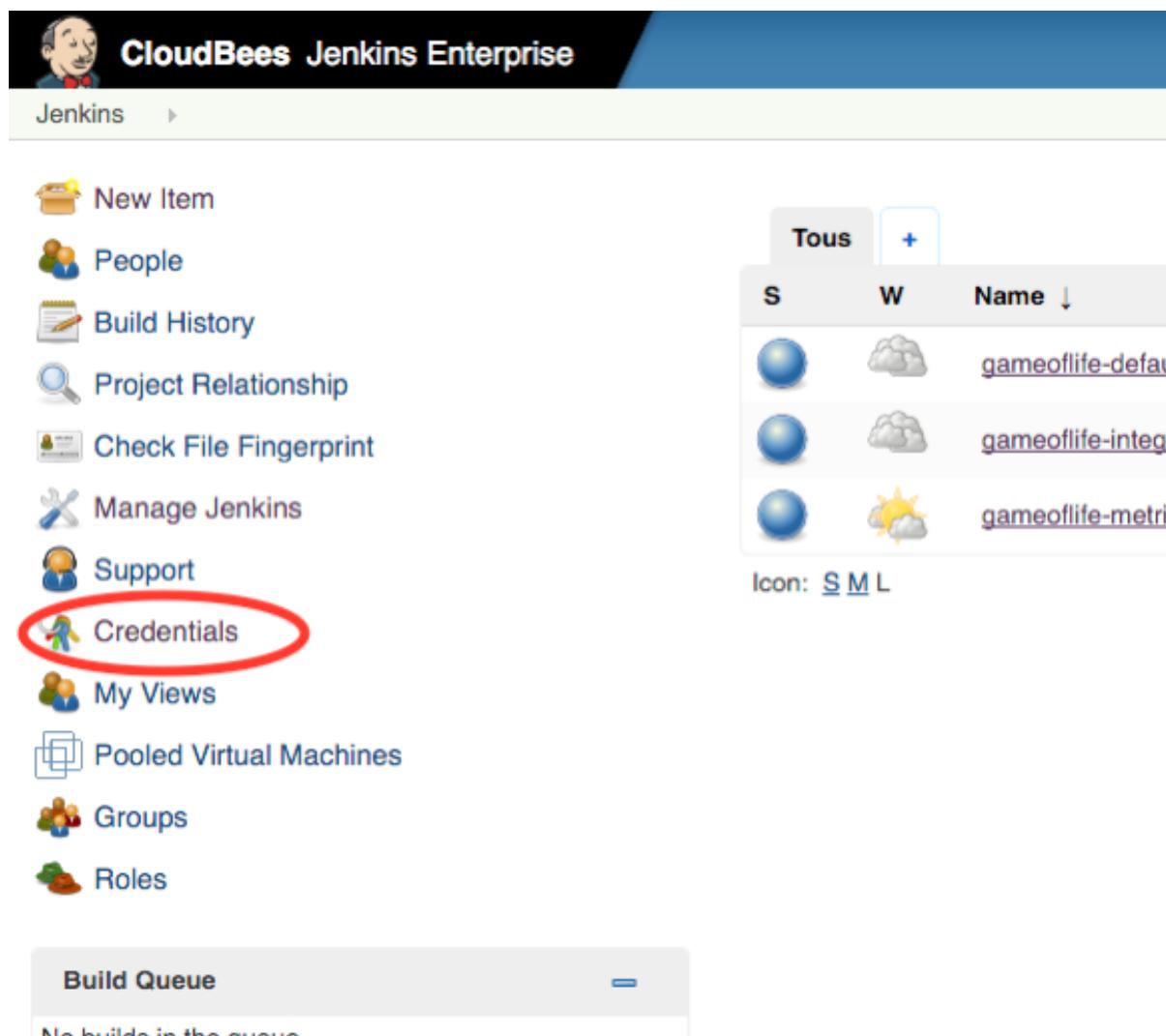
- Finally, configure the internal SSH server to use a fixed port. We propose to use **8000**:



The screenshot shows the 'SSH Server' configuration section. It has a 'SSHD Port' field containing '8000' with a radio button labeled 'Fixed'. Other options include 'Random' and 'Disable'.

Figure 77. SSH Configuration for Validated Merge

Save those configuration, and navigate to the Jenkins global credentials (**Credentials** item in the left menu):



The screenshot shows the Jenkins main page. On the left, a sidebar lists various Jenkins management options: New Item, People, Build History, Project Relationship, Check File Fingerprint, Manage Jenkins, Support, **Credentials** (which is circled in red), My Views, Pooled Virtual Machines, Groups, and Roles. To the right, there is a table titled 'Tous' showing global credentials. The table has columns for Status (S), Weather (W), and Name. Three entries are listed: 'gameoflife-default' (cloud icon, grey), 'gameoflife-integration' (cloud icon, grey), and 'gameoflife-metrics' (sun icon, yellow).

S	W	Name ↓
		gameoflife-default
		gameoflife-integration
		gameoflife-metrics

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

Figure 78. Access Credentials from Main Page

Browse to the **Global Credentials** domain:

The screenshot shows the Jenkins interface with the title "CloudBees Jenkins Enterprise". The left sidebar has links for "Up", "Credentials", and "Add domain". The main content area is titled "Credentials" with a magnifying glass icon. A table titled "Domain" lists "Global credentials (unrestricted)" with an icon of a castle tower. Below the table, it says "Icon: S M L".

Figure 79. Global Credentials

And add a new credentials for the **dev** user.

TIP The password for dev user is dev. Simple isn't it ?

Step 3: Setup a Jenkins job for CI build & validated merge

Let's pretend that you are a team lead and set up a Jenkins job that does validated merge.

Go to the Jenkins top page, click on the **New Item** menu item, and choose "Freestyle project".

Call this project **gameoflife-personal** and click **OK**:

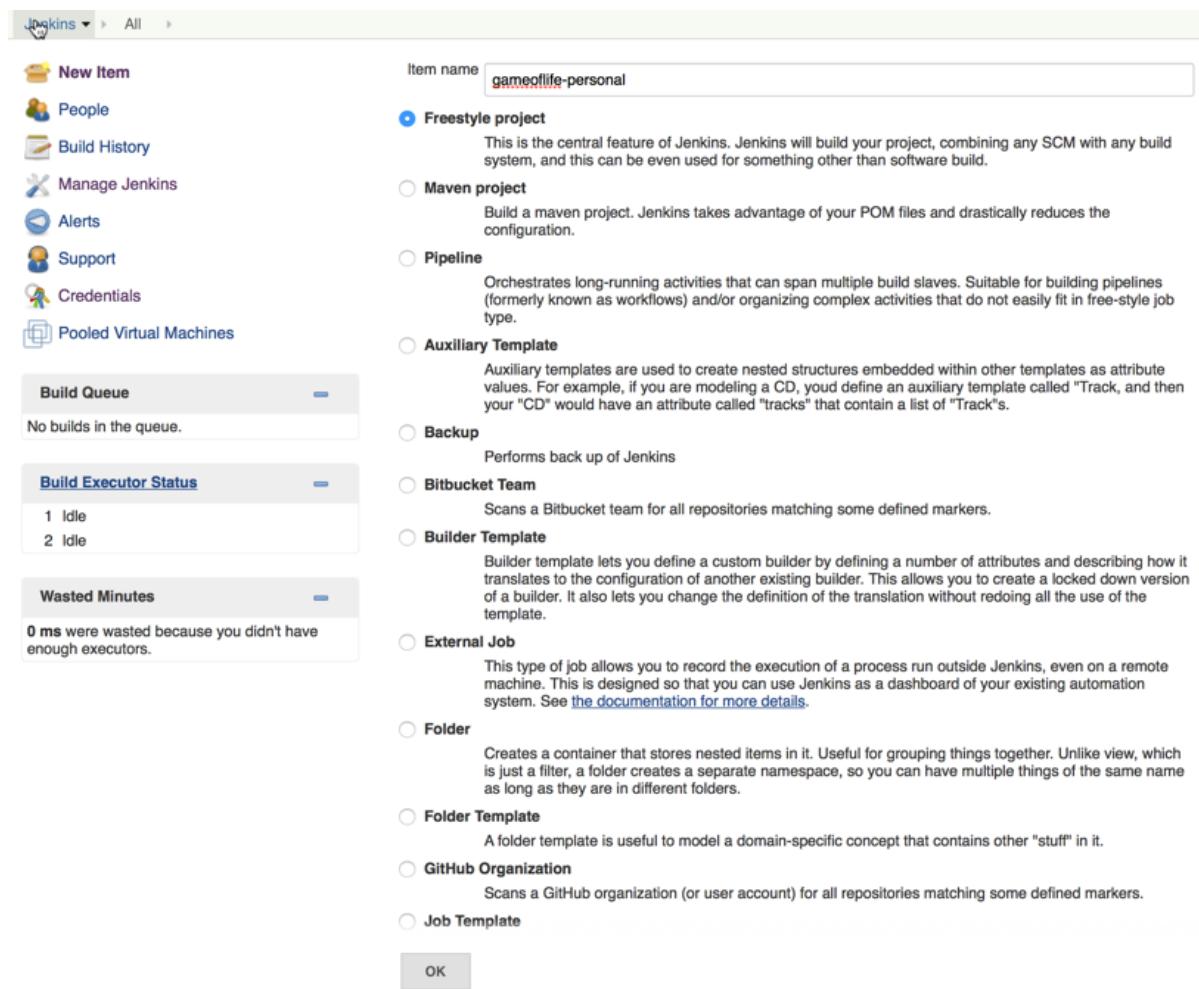


Figure 80. Create a new Freestyle Job

In the job configuration screen :

- Select **Enable Git validated merge support** to enable the feature, with those settings:
 - ☒ Select you git credentials (dev user) from the list
 - ☒ Specify an **Alternate Push URL**: `http://gitserver:3000/dev/gameoflife.git`

TIP The alternate URL use the "internal" address of the git server.

<input checked="" type="checkbox"/> Enable Git validated merge support	<small>?</small>	
If post-build push fails	<input type="checkbox"/> Fail the build if push fails	<small>?</small>
Cause the build to fail, thereby leaving it up to the submitter to deal with the situation.		
Alternate push URL	<input type="text" value="http://gitserver:3000/dev/gameoflife.git"/>	<small>?</small>
Credentials	<input type="text" value="dev/*****"/>	<small>?</small>

Figure 81. Job configuration for validated merge

- Specify the repository URL in the Source Code Management section as usual:
 - ☒ URL of the repo: `http://localhost:5000/gitserver/harry/gameoflife.git`

- ☒ Don't forget to use the new **dev** credential !

Source Code Management

None

Git

Repositories

Repository URL

Credentials

Branches to build

Branch Specifier (blank for 'any')

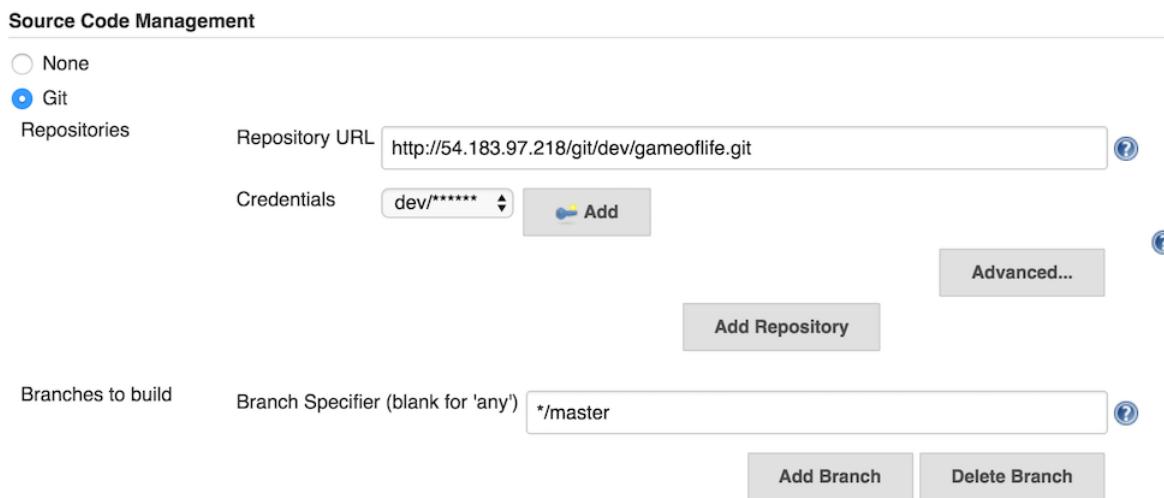


Figure 82. SCM Configuration

- In the main build section of the configuration page, add **Invoke top-level Maven targets** and then type in **install** as the Maven goal:

Build

Invoke top-level Maven targets

Maven Version

Goals



Figure 83. Maven Configuration

Click the **Save** button to apply this configuration. You can review the configuration is valid by checking that your project is now linked to the **Git Repository for Validated Merged**:

The screenshot shows the Jenkins interface for the 'gameoflife-personal' project. The left sidebar contains various Jenkins management options: Back to Dashboard, Status, Changes, Workspace, Build Now, Delete Project, Configure, Git Repository for Validated Merge (which is circled in red), Exports, Skip Builds, and Move/Copy/Promote. The main content area is titled 'Project gameoflife-personal' and includes links for 'Workspace' and 'Recent Changes'. Below this is a section titled 'Permalinks' with a list of four recent builds.

- [Last build \(#3\), 2 hr 52 min ago](#)
- [Last stable build \(#3\), 2 hr 52 min ago](#)
- [Last successful build \(#3\), 2 hr 52 min ago](#)
- [Last failed build \(#2\), 2 hr 53 min ago](#)

Figure 84. Validated Merge options for our Job

If you'd like, you can click **Build Now** to make sure this set up is correct. That completes the team lead's work.

Step 4: Edit the code and test the change

As a developer, let's make changes to the source code.

- First, let's login again in the WebIDE as **dev** user (instead of **harry**), using the right panel to logout:

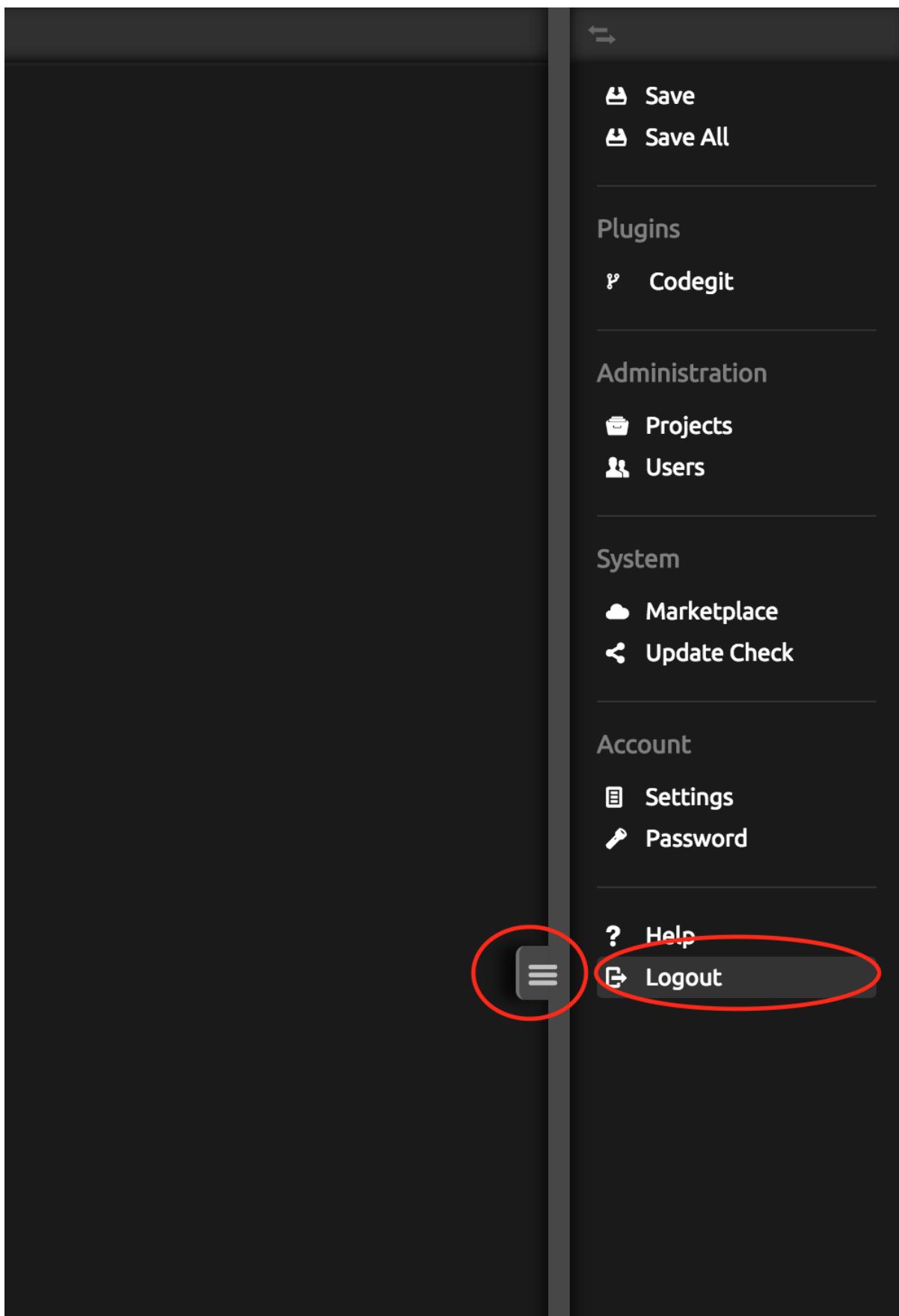


Figure 85. Logging out of the Web IDE

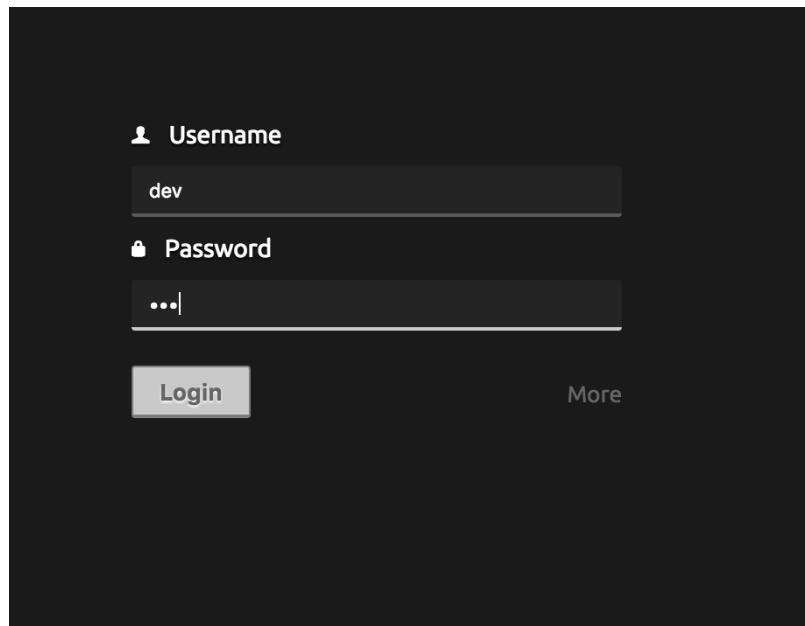


Figure 86. Logging in as dev user in the Web IDE

Then, checkout the code of the fork in the Web IDE:

TIP

Use the repository URL that you copied from Gogs at the end of Step 1 It should be an URL like this: <http://localhost:5000/gitserver/dev/gameoflife.git>

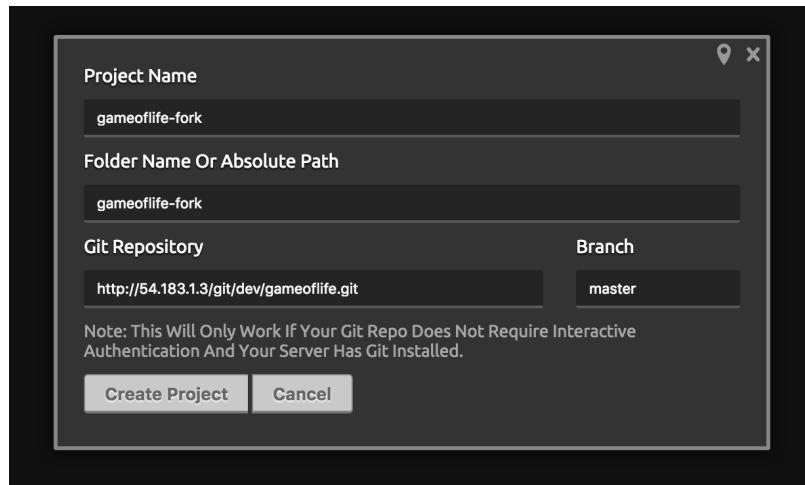


Figure 87. Checkout the code of the fork

Let's play a scenario where **dev** unknowingly breaking a build by a change. Any chance would do, but for example, edit

```
gameoflife-core/src/main/java/com/wakaleo/gameoflife/domain/Cell.java
```

and make it un-compilable by adding garbage (say ! ! ! `) at the top of the file. Commit this change as follows.

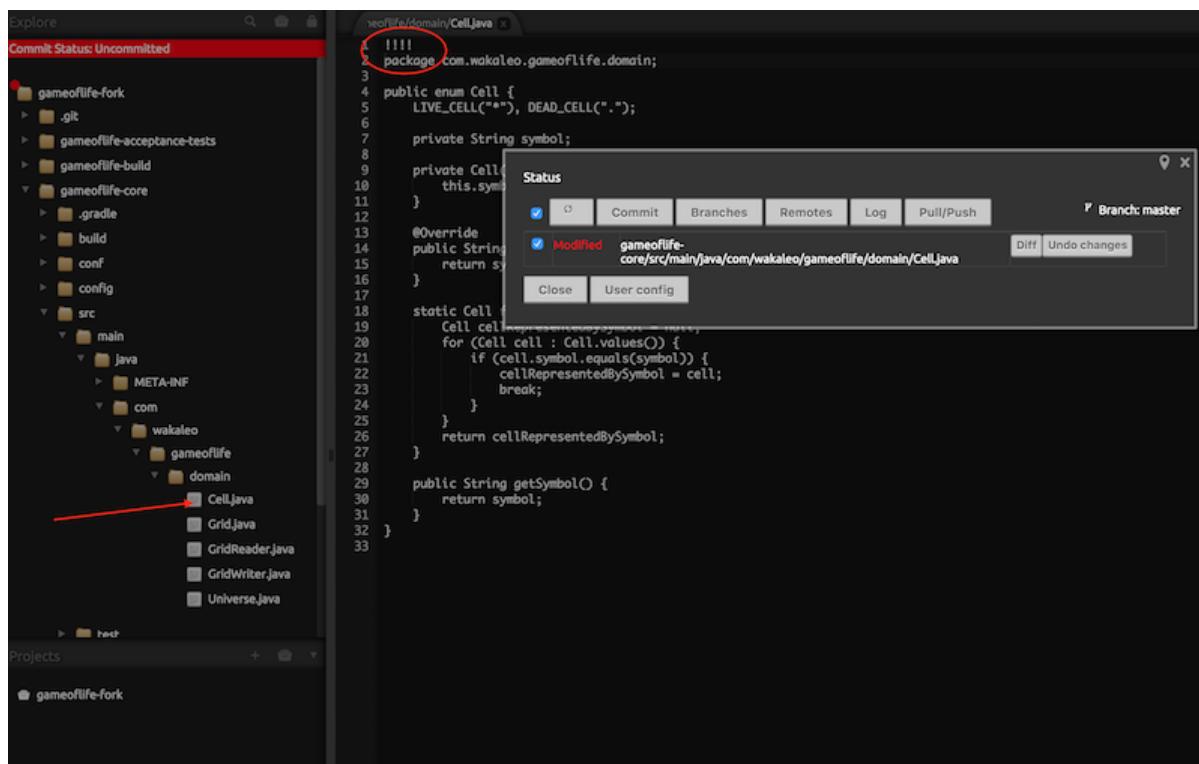


Figure 88. Breaking the Cell class



Figure 89. Committing the breaking change

IMPORTANT

Because Git is distributed, at this point this change is not exposed to the team repository yet.

To have this change tested by Jenkins, click the **Git repository for validated merge** menu item from the **gameoflife-personal** project page, then click the clipboard icon on the right to copy the command line that needs to be executed:

Git Repository for Validated Merge

This project implements [the "validated merge" feature](#), which lets you push your untested changes to have Jenkins build/test them. If the build/test is successful, your changes will be pushed to the upstream repository.

The following commands show a typical workflow to do this:

```
git remote add jenkins http://localhost:8080/git-validated-merge-repos/harry/gameoflife-personal/lcc9c0c16b0095059984733bf6ba32df73e196f08e250eaf3b65b6709c1f0be4/repo.git
git checkout branch
... make changes and commit changes ...
git push jenkins HEAD
... keep on hacking and check back Jenkins later ...
```

Figure 90. Validated Merge Build Log

We have to add a git "Remote" pointing to this **Jenkins-hosted** repository.

For this, use the **CodeGit** view from the right panel, and click the "Remote" button:

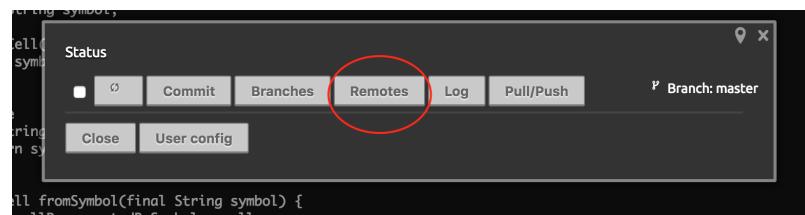


Figure 91. Accessing the CodeGit's Remote view

You see that the remote named **origin** points to your fork. Click the **New** button to add a new one:

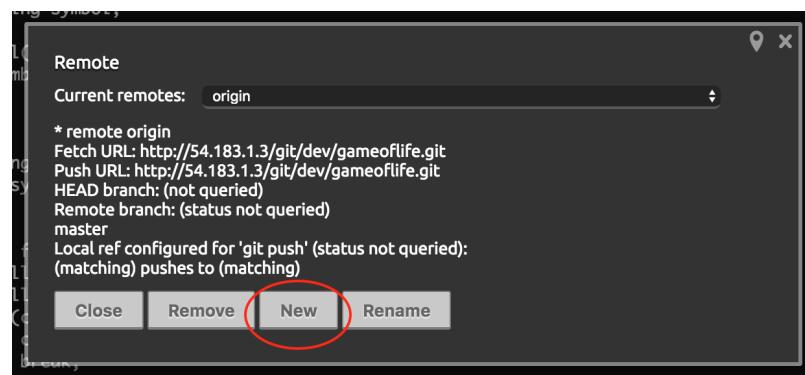


Figure 92. Accessing the new Remote view

Then fill the "New Remote" form with those settings:

- Name: **jenkins**
- Address: <The address you copied from your Job Validated Merge view>

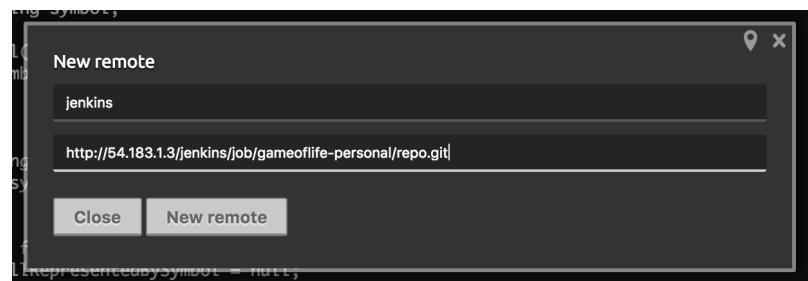


Figure 93. New Remote form pointing to Jenkins

This will register Jenkins as one of the "Remote" upstream repository.

Next, push your committed change to this remote, using the "Push/Pull" section of CodeGit view:

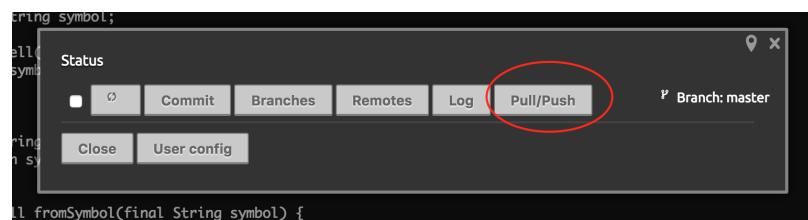


Figure 94. Accessing the CodeGit's Push/Pull view

and Push to the **jenkins** remote repository:



Figure 95. Pushing to jenkins remote

Switch back to Jenkins and wait for the build to take place. Look around in the Jenkins UI to make sure that:

- Change log correctly reports your personal change
- Build has actually picked up your change
- UI in the build top page that shows how others can retrieve this change.

Step 5: Verify there is no regression

Navigate on the Gogs repository and validate that the change has not pushed by Jenkins since the build failed.

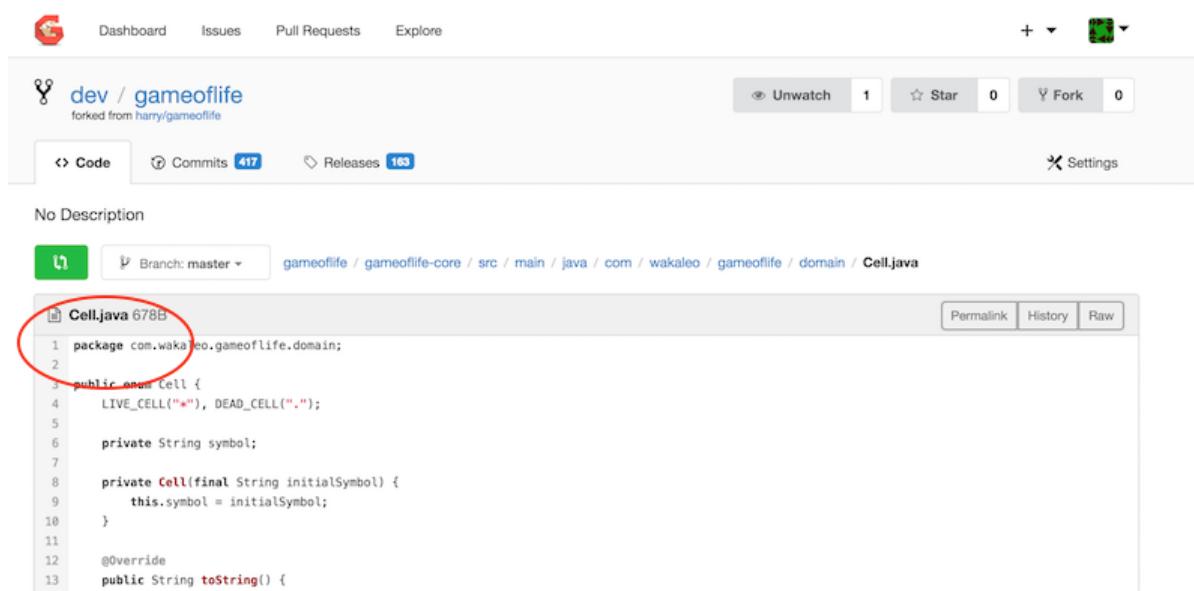


Figure 96. Fork is not changed

Step 6: Fixing the regression

Going back to the WebIDE, just make the !!! a comment:

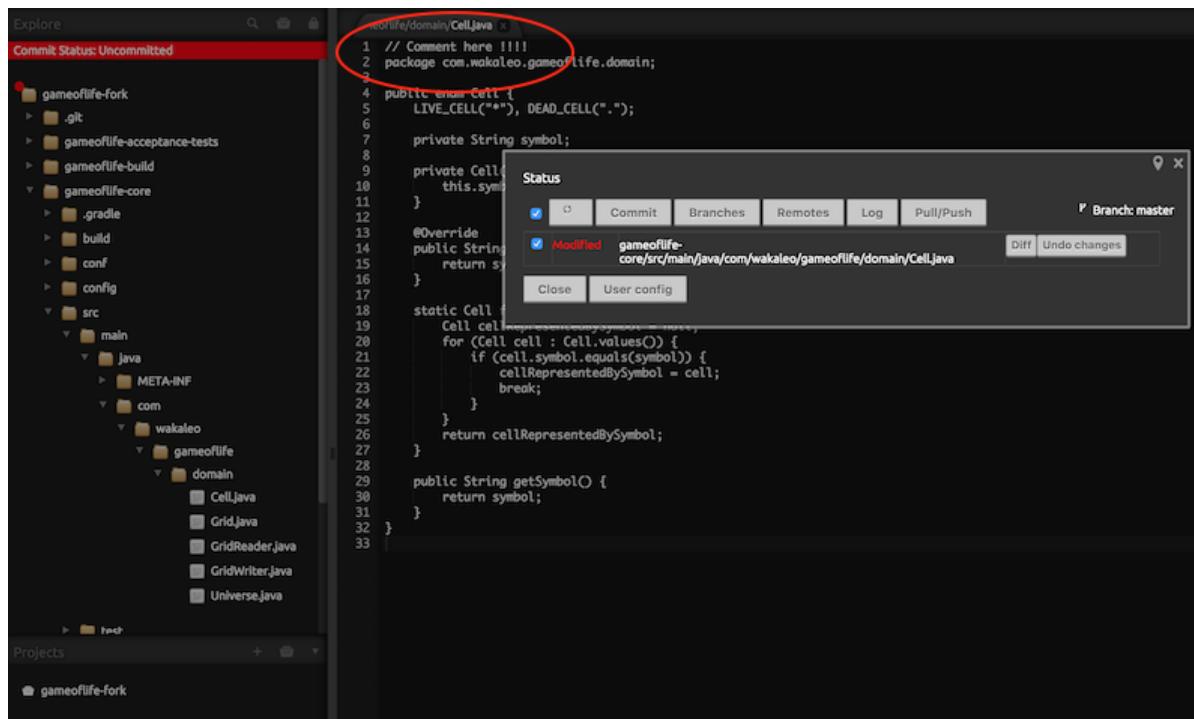


Figure 97. Commenting out the breaking change

IMPORTANT

Don't forget to commit the change, and push it to the **jenkins** remote again.

Jenkins will kick-off a new build, that should be a success:

The screenshot shows the Jenkins interface for the 'gameoflife-personal' project. On the left, a sidebar lists various project management options like Back to Dashboard, Status, Changes, Workspace, Build Now, Delete Project, Configure, Git Repository for Validated Merge, Exports, Skip Builds, and Move/Copy/Promote. The main content area is titled 'Project gameoflife-personal'. It features two large icons: 'Workspace' (a folder icon) and 'Recent Changes' (a document icon). Below these are 'Permalinks' and a bulleted list of recent builds. The 'Build History' section contains two entries:

#	Date	Status
#2	Jul 22, 2016 7:59 AM	Validating push from anonymous
#1	Jul 22, 2016 7:42 AM	Validating push from anonymous

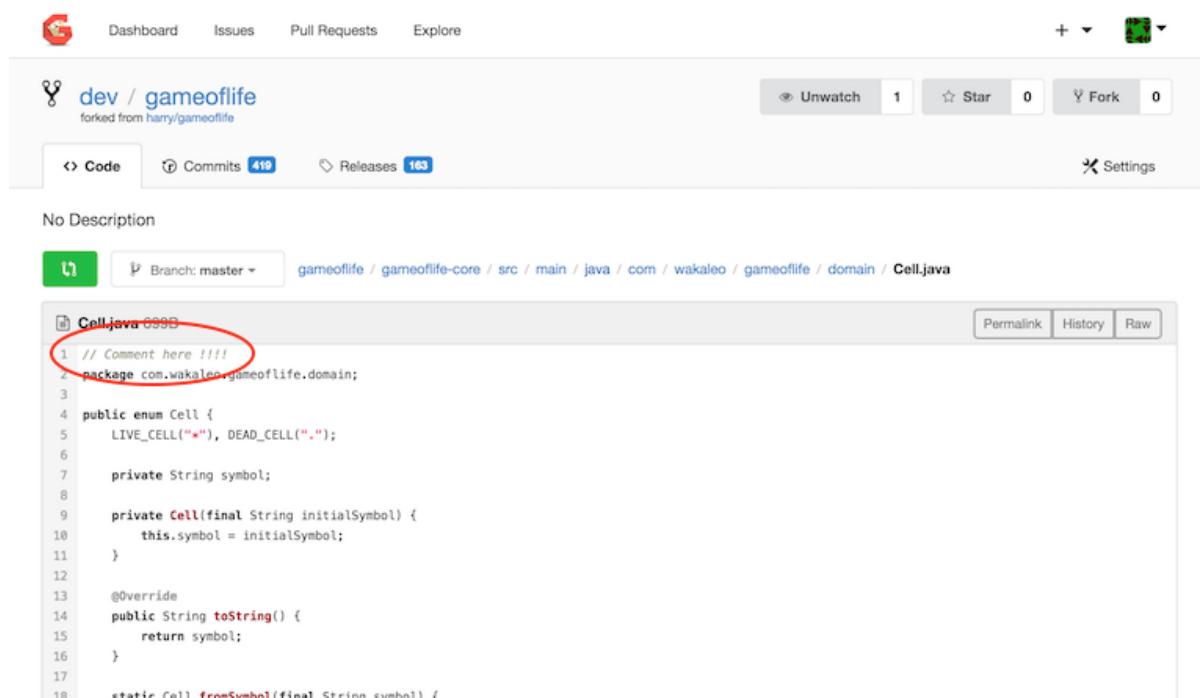
At the bottom of the history section are links for 'RSS for all' and 'RSS for failures'. A red circle highlights the '#2' entry in the build history.

Figure 98. Jenkins jobs in success state again

Step 7: Verifying the change pushed back

Since the build has been a success, Jenkins should have pushed automatically the change to the **origin** repository.

You can check this by navigating again to the Cell class on the Gogs repository, to check that your commented line is here:



The screenshot shows a GitHub repository page for 'dev / gameoflife'. The 'Code' tab is selected, displaying the contents of 'Cell.java'. A line of code containing a comment // Comment here !!!! is highlighted and circled in red. The code listing includes imports, an enum definition, constructor, overridden toString method, and a static factory method.

```
// Comment here !!!!
package com.wakaleo.gameoflife.domain;

public enum Cell {
    LIVE_CELL("x"), DEAD_CELL(".");
}

private String symbol;

private Cell(final String initialSymbol) {
    this.symbol = initialSymbol;
}

@Override
public String toString() {
    return symbol;
}

static Cell fromSymbol(final String symbol) {
}
```

Figure 99. Verifying the change has been pushed to origin

[Go back to slides](#)

Lab 12. Pipeline

The following lab exercises complements the pipeline core training.

It consists of a sequence of lab exercises that will introduce the Pipeline plugin and its DSL syntax.

Most of them are open to experimentation, and only introduce the initial usage.

Feel free to experiment further with your own ideas and ask the training instructor for advice and/or assistance.

Experiment with DSL Syntax

The aim of this initial Lab is to get familiar with Pipeline plugin and DSL syntax basis.

Step 1. Create a Pipeline Project

With Pipeline plugin installed, a new Job type is offered creating an item on Jenkins web UI : **Pipeline**.

Create a new item of the type **Pipeline** and name it **pipeline-lab**.

The screenshot shows the Jenkins dashboard with a sidebar on the left containing links like 'New Item', 'People', 'Build History', etc. The main area is titled 'Create New Item' with a search bar and user 'harry'. A form is open for creating a new item named 'pipeline-lab'. The 'Pipeline' option is selected, and its description is visible. Other project types like 'Freestyle project', 'Maven project', 'Auxiliary Template', 'Backup', 'Builder Template', 'Multi-configuration project', 'External Job', and 'Folder' are listed with their descriptions. On the left, there are sections for 'Build Queue' (empty), 'Build Executor Status' (1 Idle, 2 Idle), and 'Wasted Minutes' (0 ms). A large 'OK' button is at the bottom right of the form.

Figure 100. Creating a Pipeline Job

Compared to job types you might be used to manage, a pipeline has a very limited set of options.

You can define general :

- Job description
- Data persistence options
- Build triggers
- And pipeline script

That's it.

Everything you used to setup using form controls in UI is **replaced** in a pipeline with **DSL script**.

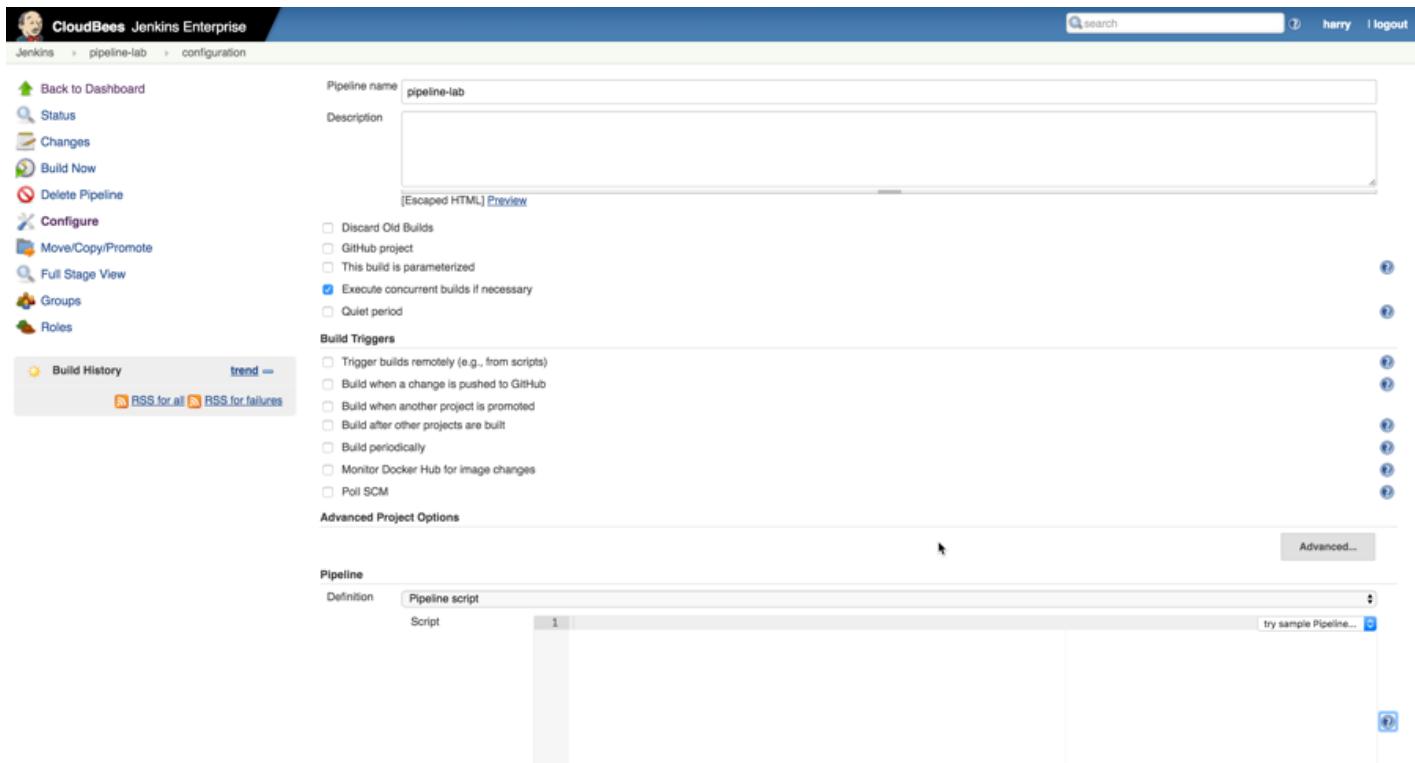


Figure 101. Overview of the Configuration

Step 2. Basic syntax

Let's start with the canonical **hello world** exercise.

TIP

Pipeline DSL is based on Groovy but you actually don't need to know anything about Groovy for simplest usages. Just rely on the **Snippet generator** to learn the DSL syntax.

Select a **Print Message** step, configure attributes and let the snippet generator generate the adequate DSL script:

The screenshot shows the Pipeline Snippet Generator interface. It has a 'Steps' section where 'echo: Print Message' is selected from a dropdown. Below it, a 'Message' field contains the text 'Hello World'. A large blue button at the bottom left says 'Generate Groovy'. To the right of the button is a code editor window showing the generated Groovy script: 'echo \'Hello World\''. The entire interface has a clean, modern design with a light blue header and a white background.

Figure 102. The Pipeline Snippet Generator

Copy-Paste to pipeline DSL script, save the job configuration (using the **Apply** button) and run it.

Pipeline is executed and you can check build console to confirm message has been printed as expected.

Step 3. Shell script

For this exercise we will introduce an execution step to run on build node, defining command to run from pipeline script.

For this purpose, we use a shell script build step:

```
node {  
    git 'https://github.com/jenkinsci/docker'  
    sh 'ls -l'  
}
```

Such a shell script let you execute arbitrary tools within your pipeline. For non-trivial shell scripts, you will need more than just a single command line.

TIP You can use Groovy multi-line syntax for this purpose.

Groovy let you define a long, multi-line string, delimited by 3 double quotes:

```
node {  
    git 'https://github.com/jenkinsci/docker'  
    sh """  
        ls -l  
        cat README.md  
    """  
}
```

Pipeline Control flows

During this exercise we will experiment with pipeline DSL control flow syntax (based on Groovy).

Step 1. Control flow syntax

Let's parameterize the pipeline job with a boolean parameter.

As already done in the Lab 9 (Parameterized builds), add a new parameter to your job's config.

Our goal is to have a distinct execution path depending on this parameter value:



Figure 103. Define a Parameter

Snippet generator does not assist you in writing control flow syntax, for this one you will need to learn some basic of Groovy language.

Anyway this is the easiest part and comparable to many other popular languages.

The **Script** text area features a Groovy text editor that will assists (syntax color, autocomplete, etc.).

```
node {  
    git 'https://github.com/jenkinsci/docker'  
    echo "foo is $foo"  
    if (foo == 'true') {  
        echo "foo is true"  
        sh 'ls -l'  
    } else {  
        echo "foo is false"  
        sh 'cat docker-compose.yml'  
    }  
}
```

Step 2. Interact with humans

Add an **input** step to your pipeline to ask administrator for pipeline continuation. Such a manual step is useful to confirm some status that you can't automate, like checking some web UI.

TIP Can also be required when you need a set of authorized people to approve deployment.

```
input message: 'Is it ok to push to server ?',ok: 'Yes, please do'
```

Experiment with human interaction, and also check what happens when you restart Jenkins while a pipeline is waiting for human input.

Concurrent execution

This exercise aims to demonstrate concurrent execution within a single pipeline. For this purpose we will have two concurrent branches running loops

Step 1. Create a loop

Use Groovy syntax to create a loop, and pipeline's **sleep** and **echo** steps within this loop to fake some long running build activity.

TIP Use the snippet generator to discover those steps DSL syntax.

```
for (def i = 0; i < 20; i++) {  
    echo "building... $i"  
    sleep 1  
}
```

Step 2. Create a second loop to run in parallel

Use the parallel sub-pipeline snippet generator to discover parallel execution syntax.

Copy the build loop as a parallel branch, and create a second one with distinct output:

```
parallel firstBranch: {  
    for (def i = 0; i < 20; i++) {  
        echo "building... $i"  
        sleep 1  
    }  
, secondBranch: {  
    for (def i = 0; i < 10; i++) {  
        echo "building in parallel ... $i"  
        sleep 2  
    }  
}
```

Run the pipeline and check build log to confirm parallel execution.

Pipeline Stages

Goal:

The goal of this lab will be to create a complete pipeline which is composed of a set of sub-steps.

Step 1. Define stages

Let's introduce **Stages** in the pipeline, to make our process explicit:

```
stage "build"
echo "step 1"
echo "step 2"

stage "test"
echo "step 3"
echo "step 4"

stage "package"
echo "step 5"
echo "step 6"

stage "acceptance-test"
echo "step 7"
echo "step 8"

stage "deploy"
echo "step 9"
```

Run this pipeline and check the job index page for visualization

Step 2. Use stages concurrency

Configure the **acceptance-test** stage so it only support a single concurrent execution.

This will match a scenario where your test infrastructure only can run a **single** test suite, for licensing or resource consumption reasons.

```
stage concurrency: 1, name: "acceptance-test"
echo "step 7"
sleep 30
echo "step 8"
```

IMPORTANT

Do not forget to disable the Groovy sandbox since this capability (concurrency) needs low level rights.

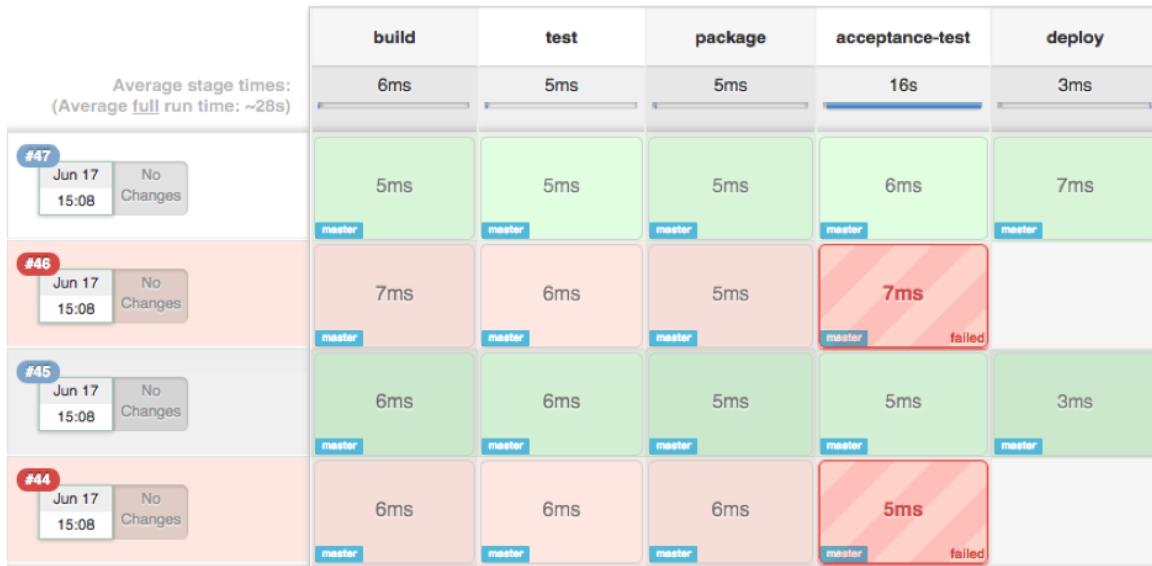
Trigger pipeline job multiple times and check what's happening on stage execution.

Step 3. Use checkpoints

Declare a checkpoint after the **package** stage completion, and introduce a failure in acceptance-test stage:

```
stage "package"
echo "step 5"
echo "step 6"

stage "acceptance-test"
echo "step 7"
echo "step 8"
```

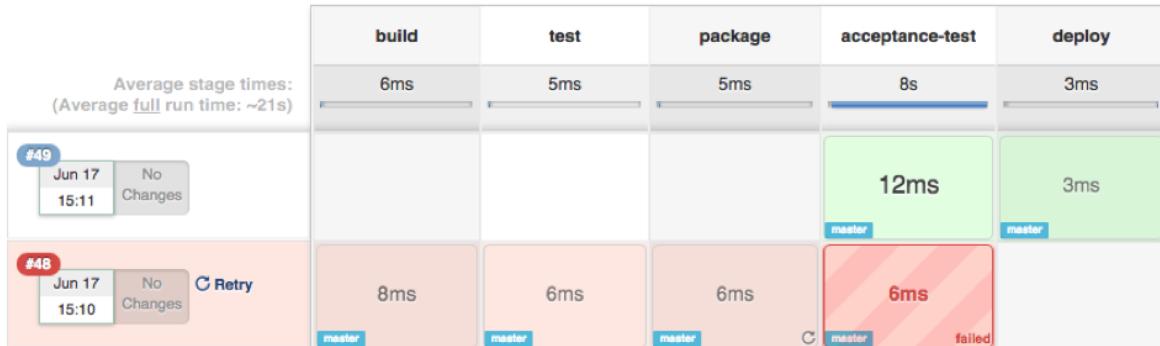
*Figure 104. Introducing a Failure*

Introduce a checkpoint just before the acceptance-test stage:

```
checkpoint 'packaged'
stage "acceptance-test"
```

Run a new build and navigate the build index page. On side-panel, click the Checkpoints link.

A **packaged** checkpoint has been recorded, and can be used to restart the pipeline from saved state:

*Figure 105. Checkpoint on Stage*

Go back to slides

Lab 13: Templates

Builder template

Goal

In this lab, you'll learn how to create a builder template. A builder template allows you to create custom build steps that can be used many times over in different Freestyle jobs.

Step 1. Create a builder template

First, we'll create a builder template. When executed, this builder template produces a text file in the workspace that lists all the changes that went into this build.

In Jenkins, click **New Item** and select **Builder Template**. Give it the name `Save Changes`.

Click **OK** to create a new empty builder template.

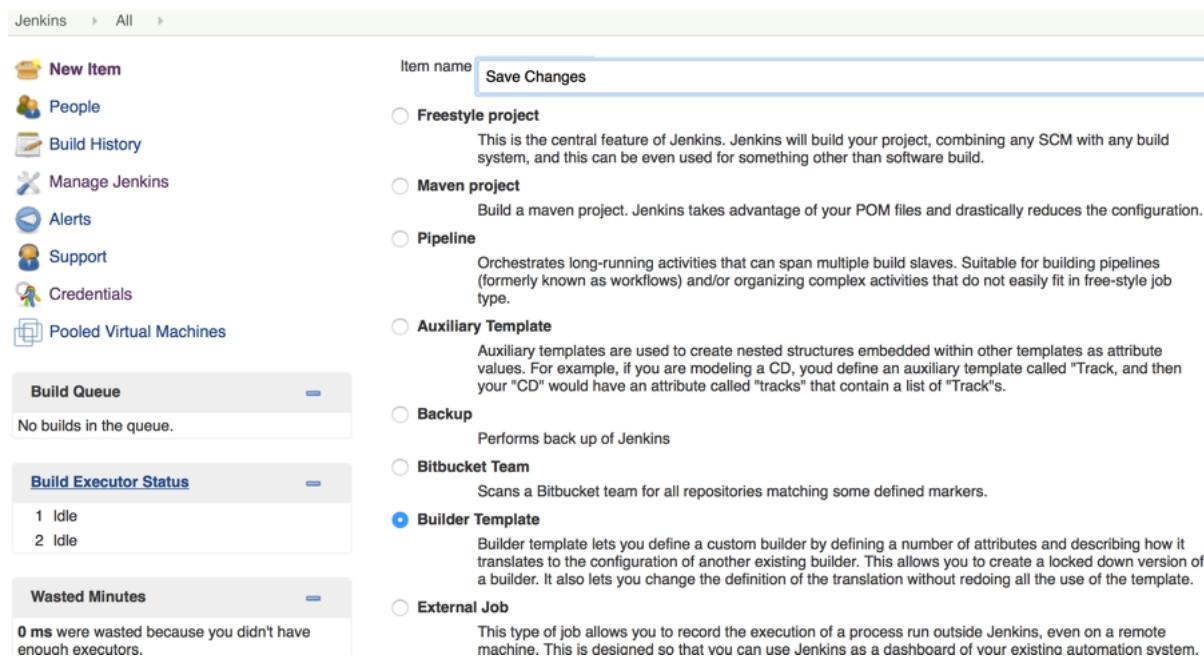


Figure 106. Creating a new Builder Template

You are now in the template configuration page.

In the **Attribute** section, click **Add Attribute**, and configure as below:

- **ID:** changelog
- **Display name:** Change Log File

- **Type:** Text field (should be default)
- **Inline Help:** A short description of the field, like This is where the changelog entries are written to

In the **Transformer** section:

- Make sure that the **Type** field is set to **Sets some variables using Groovy** (should be by default)
- **Groovy Script** field: enter the code below:

```
build.workspace.child(changelog).  
write(build.changeSet.items*.msg.join('\r\n'), null)
```

- Set the **Shell/Batch Script** field as follows, so that the build console will indicate that the file has been written:

```
echo "Wrote changelog"
```



Figure 107. Configuration of Builder Template

Click **Save** to complete the setup of this template.

How did you come up with that Groovy script?

The “build” variable refers to the instance of the `AbstractBuild` class that represents the build currently in progress.

TIP

For other exposed variables, see the “Variable Bindings” section in <http://jenkins-enterprise.cloudbees.com>.

For methods of `AbstractBuild`, see <http://javadoc.jenkins-ci.org/byShortName/AbstractBuild>.

Step 2. Use a builder template

Open the **gameoflife-freestyle** project from Lab 2, and then click the **Configure** link from the left.

Turn off **Poll SCM** because we want to make several commits before firing a new build to test our new builder template, and polling might automatically start a new build when we don't want one.

Click **Add build step**, then select **Save Changes** to insert a new build step. Enter `changes .txt` for the **Change Log File** field.

Grab the **Save Changes** header of this build step, then drag this to the top of the **Build** section.

Click **Add post-build action**, select **Archive artifacts**, and enter `changes .txt` for the **Files to archive** field.

This lets us see the generated changelog file from Jenkins web UI:

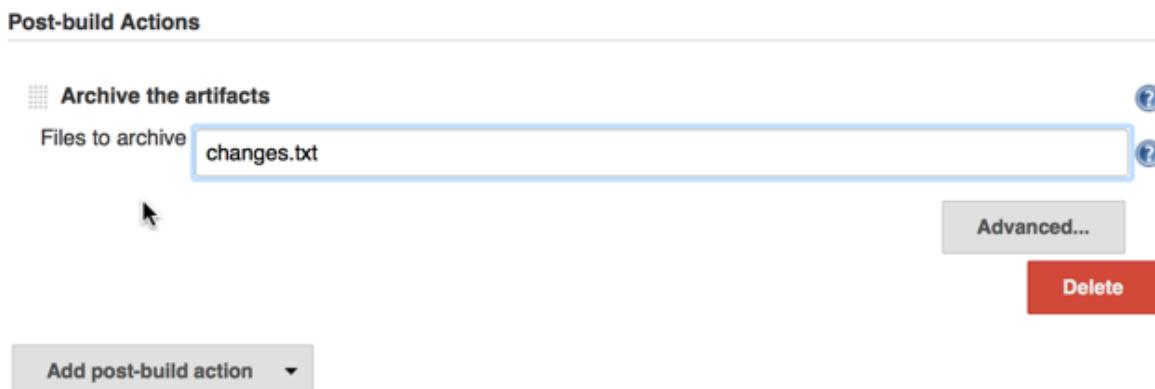


Figure 108. Post Build Step with Builder Template

Click **Save** to apply this configuration to Jenkins.

Step 3. See the builder in action

Now open Codiad, and edit any file in this project (e.g. add some spaces or add a comment to pom.xml).

Save your changes, commit and don't forget to push to the remote Git repository.

TIP If prompted, use harry as both the username and password.

Do this again to make a second commit (or more), but make sure to use different commit messages so that we can tell them apart!

Click **Build Now** in the Jenkins project to schedule a build. When it has finished, click on the newly completed build.

Build Artifacts should list `changes .txt`.

Click this link and you should see your commit messages.

TIP Use the Back button to return to Jenkins.

Job template

Goal

In this lab, you will learn how to create a job template. Job templates are useful when you would like to create multiple jobs that are very similar to each other with only a few parameters that are different between jobs. An example would be defining a Job template for a build job where the build process is the same for different projects but may only differ by the repository to be used. As with all of the templates, when changes are made to the template all child jobs using the template are also affected.

Step 1. Create a job template

Now we will turn the **gameoflife-freestyle** project into a job template.

In Jenkins, click **New Item** and select **Job Template**. Give it the name `Our Maven Project`.

Click **OK** to create a new empty job template.

Select **Groovy template transformation**.

In the **Load Prototype Job** field enter the job name **gameoflife-freestyle** and click **Load**. This will load the `config.xml` file in the **Script** field. The `config.xml` is the backend definition that Jenkins uses to read the settings of the job. The `config.xml` will define what is the 'same' between the different child jobs.

You can drag the thick bar at the bottom of the text area down to see more than a few lines of the script at once.

Step 2. Define attributes

Let's define some attributes:

We'll use an attribute to let users specify where to check out the source code:

Click **Add Attribute** and specify:

- **ID** field: the value `repoLocation`
- **Display name** field: `SCM Checkout Location`

- **Type** field: Ensure that **Text-field** is selected (should be default)

We'll use another to allow users to specify extra Maven properties:

Add another attribute and specify:

- **ID** field: **properties**
- **Display name** field: **Extra Maven Properties**
- Set the **Type** field to **Text area**.

The screenshot shows the Jenkins Job Template configuration interface. It displays two attribute definitions side-by-side.

Attribute 1 (Top):

- ID: repoLocation
- Display Name: SCM Checkout Location
- Type: Text-field

Attribute 2 (Bottom):

- ID: properties
- Display Name: Extra Maven Properties
- Type: Text area

Both attributes have "Inline Help" sections below them, each with a "Plain text" and "Preview" link. A red "Delete" button is located at the bottom right of each attribute section. At the very bottom left, there is a grey "Add" button.

Figure 109. Attributes for the Job Template

Next, we'll tweak the **Transformers** to use these attributes:

Look in the **Script** text area of the **Transformer** section and replace:

- `http://localhost:5000/gitserver/harry/gameoflife.git by ${repoLocation}`
- `webdriver.port=9091<linebreak>jetty.stop.port=9998 with ${properties}`

IMPORTANT Don't delete nor remove `surefire.useFile=false`

Groovy template transformation

Script

```
<hosts>
<com.cloudbees.plugins.deployer.impl.amazon.HostImpl plugin="cloudbees-aws-deployer@1.15">
<@sets class="java.util.Collections\$UnmodifiableRandomAccessList" resolves-to="java.util.Collections\$UnmodifiableList">
<c class="list />
<list reference=".c"/>
</targets>
<credentialsId></credentialsId>
</com.cloudbees.plugins.deployer.impl.amazon.HostImpl>
</hosts>
</com.cloudbees.plugins.deployer.DeployNowJobProperty>
</properties>
<scm class="hudson.plugins.git.GitSCM" plugin="git@2.5.2">
<configVersion>2</configVersion>
<userRemoteConfigs>
<hudson.plugins.git.UserRemoteConfig>
<url>$repoLocation</url>
<credentialsId>a2062d14-4197-461b-a8a6-5d5879959481</credentialsId>
</hudson.plugins.git.UserRemoteConfig>
</userRemoteConfigs>
</branches>
<hudson.plugins.git.BranchSpec>
<name>/master</name>
</hudson.plugins.git.BranchSpec>
</branches>
<doGenerateSubmoduleConfigurations>false</doGenerateSubmoduleConfigurations>
<browser class="me.renann.gogsrepositoryviewer.GogsGitRepositoryBrowser" plugin="gogs-repository-viewer@1.0.1.Alfa">
<url></url>
</browser>
<submoduleCfg class="list"/>
<extensions/>
<scm>
<canRoam>true</canRoam>
<disabled>false</disabled>
<blockBuildWhenDownstreamBuilding>false</blockBuildWhenDownstreamBuilding>
<blockBuildWhenUpstreamBuilding>false</blockBuildWhenUpstreamBuilding>
<triggers/>
<concurrentBuild>false</concurrentBuild>
<builders>
<com.cloudbees.hudson.plugins.modeling.impl.builder.BuilderImpl plugin="cloudbees-template@4.21">
<model>Save Changes</model>
<values>
<entry>
<string>changelog</string>
<string>changes.txt</string>
</entry>
</values>
</com.cloudbees.hudson.plugins.modeling.impl.builder.BuilderImpl>
<hudson.tasks.Maven>
<targets>clean install -B -U</targets>
<mavenName>mvn3</mavenName>
<properties>surefire.useFile=true
${properties}</properties>
<usePrivateRepository>false</usePrivateRepository>
<settings class="jenkins.mvn.DefaultSettingsProvider"/>
<globalSettings class="jenkins.mvn.DefaultGlobalSettingsProvider"/>
</hudson.tasks.Maven>
```

Figure 110. Final Transformer for Job Template

Finally, click **Save** to define this new job template.

Step 3. Instantiate this template

From the breadcrumb menu, select **Jenkins**, click **New Item** and select **Our Maven Project**.

Name it **gameoflife-freestyle-templatized**, then click **OK** to create a new job:

Item name

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.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Workflow
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines and/or organizing complex activities that do not easily fit in free-style job type.

Auxiliary Template
Auxiliary templates are used to create nested structures embedded within other templates as attribute values. For example, if you are modeling a CD, you'd define an auxiliary template called "Track, and then your "CD" would have an attribute called "tracks" that contain a list of "Track's.

Backup
Performs back up of Jenkins

Builder Template
Builder template lets you define a custom builder by defining a number of attributes and describing how it translates to the configuration of another existing builder. This allows you to create a locked down version of a builder. It also lets you change the definition of the translation without redoing all the use of the template.

External Job
This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See [the documentation for more details](#).

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.

Folder Template
A folder template is useful to model a domain-specific concept that contains other "stuff" in it.

Job Template
A job template is useful to create a large number of jobs that share a similar configuration. A designer of a template can enforce uniformity, and when a template changes, all the jobs created from the template are updated right away.

Long-Running Project
A project which runs a special build step detached from Jenkins.

Multi-configuration project
Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.

Our Maven Project

Publisher Template
The publisher template lets you define a custom publisher by defining a number of attributes and describing how it translates to the configuration of another existing publisher. This allows you to create a locked down version of a publisher. It also lets you change the definition of the translation without redoing all the use of the template.

Figure 111. New item from Job Template

You should be now taken to the job configuration page. Fill the attributes as it:

- **SCM Checkout Location**: `http://localhost:5000/gitserver/harry/gameoflife.git`
- **Extra Maven Properties**, in two lines:

```
webdriver.port=9091
jetty.stop.port=9998
```

Click **Save** to complete changes.

If you now look at `http://localhost:5000/jenkins/job/gameoflife-freestyle-templatized/config.xml`, you should see very similar content that you saw earlier, except for a

```
<com.cloudbees.hudson.plugins.modeling.impl.jobTemplate.JobPropertyImpl>
```

with attribute properties for the instance.

Click **Build Now** and you should see similar results as when building the original job.

Step 4. Applying a template change

To verify that the template is a template and not just a wizard that creates a job, let's update the definition of this job template and make sure it gets reflected right away.

Go back to the template definition, and look for in the **Script** field, this is how we invoke Maven:

```
clean install -B -U
```

Change this, by removing the `clean` target, then save this configuration:

```
install -B -U
```

Go back to the **gameoflife-freestyle-templated** job. Without touching the configuration, simply click the **Build Now** link to start a new build, then observe that the build runs **without** the clean goal in Maven now.

Pipeline Templates

Goal

In this lab, you will learn how to create a Pipeline template.

Pipelines allow you to define and visualize the whole application lifecycle. With Templates you can standardize pipelines, quicken the onboarding process for new projects, and make changes globally to your templated jobs.

Step 1. Create a Pipeline template

In Jenkins, click **New Item** and select **Job Template**. Give it the name `Pipeline Template`.

Click **OK** to create a new empty job template.

Select **Groovy template for Pipeline** in the **Transformer** section.

In the **Pipeline Script** section, paste the following pipeline script:

```
node {  
    stage 'build'  
    git 'https://github.com/jenkinsci/docker'  
  
    echo "jdk8 is $jdk8"  
  
    if (jdk8 == true) {  
        echo "Building in jdk8"  
    } else {  
        echo "Building in jdk7"  
    }  
}
```

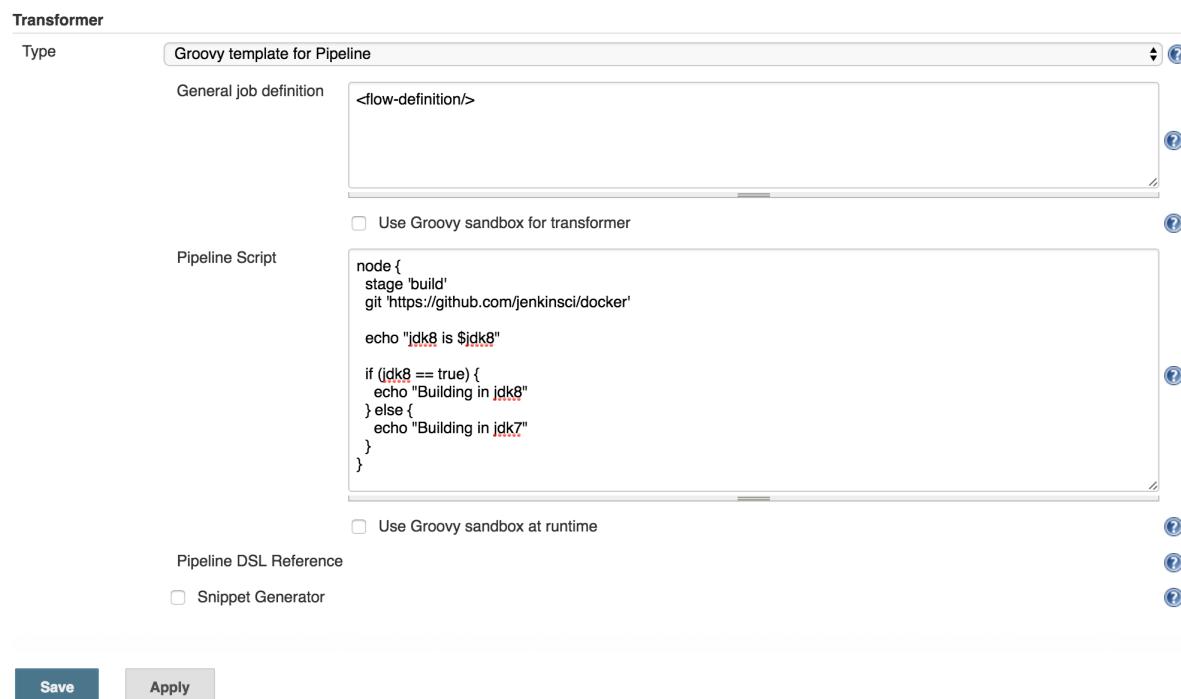


Figure 112. Pipeline Script

Step 2. Define attributes

Let's define some attributes:

We'll use an attribute to let users specify where to check out the source code:

Click **Add Attribute** and specify:

- **ID field:** the value `repoLocation`
- **Display name field:** Repository Checkout Location
- **Type field:** Ensure that **Text-field** is selected (should be default)

We'll use another to allow users to specify whether to build the project with jdk8 or not.

- **ID** field: jdk8
- **Display name** field: Use JDK8 ?
- Set the **Type** field to **Checkbox**.

The screenshot shows the 'Pipeline Template Attributes' section of the Jenkins UI. It displays two attribute definitions, each with fields for ID, Display Name, Type, and an optional Inline Help text area. The first attribute, 'repoLocation', has an ID of 'repoLocation', a display name of 'Repository Checkout Location', and a type of 'Text-field'. The second attribute, 'jdk8', has an ID of 'jdk8', a display name of 'JDK8?', and a type of 'Checkbox'. The 'checkbox' type is selected. Below the checkbox type, there is a 'Default Value' section with radio buttons for 'Checked' (selected) and 'Unchecked'. Each attribute entry includes a 'Delete' button in the top right corner and a 'Plain text' / 'Preview' link at the bottom. An 'Add' button is located at the bottom left of the list.

ID	Display Name	Type
repoLocation	Repository Checkout Location	Text-field
jdk8	JDK8?	Checkbox

Figure 113. Pipeline Template Attributes

Next, we'll tweak the **Transformers** to use these attributes:

Look in the **Pipeline Script** text area of the **Transformer** section and replace:

- `git 'https://github.com/jenkinsci/docker'` with `git repoLocation`

The `jdk8` variable is already referenced in the script.

Transformer

Type: Groovy template for Pipeline

General job definition: <flow-definition/>

Use Groovy sandbox for transformer

Pipeline Script:

```
node {  
    stage "build"  
    git repoLocation  
  
    echo "jdk8 is $jdk8"  
  
    if (jdk8 == true) {  
        echo "Building in jdk8"  
    } else {  
        echo "Building in jdk7"  
    }  
}
```

Use Groovy sandbox at runtime

Pipeline DSL Reference

Snippet Generator

Figure 114. Attributes added to Pipeline Script

Finally, click **Save** to define this new job template.

Step 3. Instantiate this template

From the breadcrumb menu, select **Jenkins**, click **New Item** and select **Pipeline Template**.

Name it `gameoflife-pipeline-templatized`, then click **OK** to create a new job.

Item name

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.

Maven project
Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.

Pipeline
Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

Auxiliary Template
Auxiliary templates are used to create nested structures embedded within other templates as attribute values. For example, if you are modeling a CD, you'd define an auxiliary template called "Track, and then your "CD" would have an attribute called "tracks" that contain a list of "Track"s.

Backup
Performs back up of Jenkins

Pipeline Template

Publisher Template
The publisher template lets you define a custom publisher by defining a number of attributes and describing how it translates to the configuration of another existing publisher. This allows you to create a locked down version of a publisher. It also lets you change the definition of the translation without redoing all the use of the template.

Copy existing Item
Copy from

Figure 115. Create a new job from the Pipeline Template

You should be now taken to the job configuration page. Fill the attributes as specified below:

• **Repository Checkout Location :**

`http://localhost:5000/gitserver/harry/gameoflife.git`

• **Use JDK8?** should be checked

The screenshot shows the Jenkins job configuration interface for a templated pipeline. The top navigation bar includes the CloudBees logo and the Jenkins URL. The left sidebar lists various Jenkins management options like Back to Dashboard, Status, Changes, Build Now, Delete Pipeline, Configure, Move/Copy/Promote, Template: Pipeline Template, and Full Stage View. The main configuration area has fields for Name (set to 'gameoflife-pipeline-templatized'), Repository Checkout Location (set to 'http://54.193.97.154/git/harry/gameoflife.git'), and a checked checkbox for 'JDK8?'. Below these are Save and Apply buttons. At the bottom, there's a 'Build History' section with a search bar and RSS links for all and failures.

Figure 116. Templatized Job Configuration

Click **Save** to complete changes.

Click **Build Now**.

Note in the console output that the job checks out the specified directory and uses jdk8.

Console Output

```
Started by user anonymous
[Pipeline] Allocate node : Start
Running on master in /var/jenkins_home/workspace/gameoflife-pipeline-templatized
[Pipeline] node {
[Pipeline] stage (build)
Entering stage build
Proceeding
[Pipeline] git
> git rev-parse --is-inside-work-tree # timeout=10
Fetching changes from the remote Git repository
> git config remote.origin.url http://54.193.97.154/git/harry/gameoflife.git # timeout=10
Fetching upstream changes from http://54.193.97.154/git/harry/gameoflife.git
> git --version # timeout=10
> git -c core.askpass=true fetch --tags --progress http://54.193.97.154/git/harry/gameoflife.git
+refs/heads/*:refs/remotes/origin/*
> git rev-parse refs/remotes/origin/master^{commit} # timeout=10
> git rev-parse refs/remotes/origin/origin/master^{commit} # timeout=10
Checking out Revision ea982fe5bc1d1c5a77a2e548bbd9f75ac76017cf (refs/remotes/origin/master)
> git config core.sparsecheckout # timeout=10
> git checkout -f ea982fe5bc1d1c5a77a2e548bbd9f75ac76017cf # timeout=10
> git branch -a -v --no-abbrev # timeout=10
> git branch -D master # timeout=10
> git checkout -b master ea982fe5bc1d1c5a77a2e548bbd9f75ac76017cf
> git rev-list ea982fe5bc1d1c5a77a2e548bbd9f75ac76017cf # timeout=10
[Pipeline] echo
jdk8 is true
[Pipeline] echo
Building in jdk8
[Pipeline] f //node
[Pipeline] Allocate node : End
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Figure 117. Console Output for Templatized Job

Step 4. Applying a template change

To verify that the template is a template and not just a wizard that creates a job, let's add another stage to the Pipeline template and make sure that it gets reflected in the child job.

Select the Pipeline Template job and click **Configure**.

Now go back to the **Pipeline Script** section, paste the following:

```
node {  
    stage 'build'  
    git repoLocation  
  
    echo "jdk8 is $jdk8"  
  
    if (jdk8 == true) {  
        echo "Building in jdk8"  
    } else {  
        echo "Building in jdk7"  
    }  
  
    stage 'Deploy'  
    input 'deploy?'  
  
    echo "deployed application!"  
}
```

Notice that it is the same script but we added a new Deploy stage to the template.

Click **Save**.

Go back to the `gameoflife-pipeline-templatized` job. Without touching the configuration, simply click the **Build Now** link to start a new build. Observe that the job now asks for an approval after the build completes and will subsequently deploy the application if the approval is given.

CloudBees Jenkins Enterprise

Jenkins > gameoflife-pipeline-templated >

[Back to Dashboard](#) [Status](#) [Changes](#) [Build Now](#) [Delete Pipeline](#) [Configure](#) [Move/Copy/Promote](#) [Template: Pipeline Template](#) [Full Stage View](#)

[Recent Changes](#)

Pipeline gameoflife-pipeline-templated

Stage View

Average stage times:
(Average full run time: ~1s)

build	Deploy
1s	0ms

#2 Aug 18 13:54 Ch deploy? [Proceed](#) [Abort](#)

#1 Aug 18 13:53 No Changes almost complete 1s master

0ms (paused for 1ms) paused

Build History

find

#2 Aug 18, 2016 5:54 PM

#1 Aug 18, 2016 5:53 PM

[RSS for all](#) [RSS for failures](#)

Permalinks

- [Last build \(#1\), 36 sec ago](#)
- [Last stable build \(#1\), 36 sec ago](#)
- [Last successful build \(#1\), 36 sec ago](#)
- [Last completed build \(#1\), 36 sec ago](#)

Figure 118. Deploy Stage added to Template

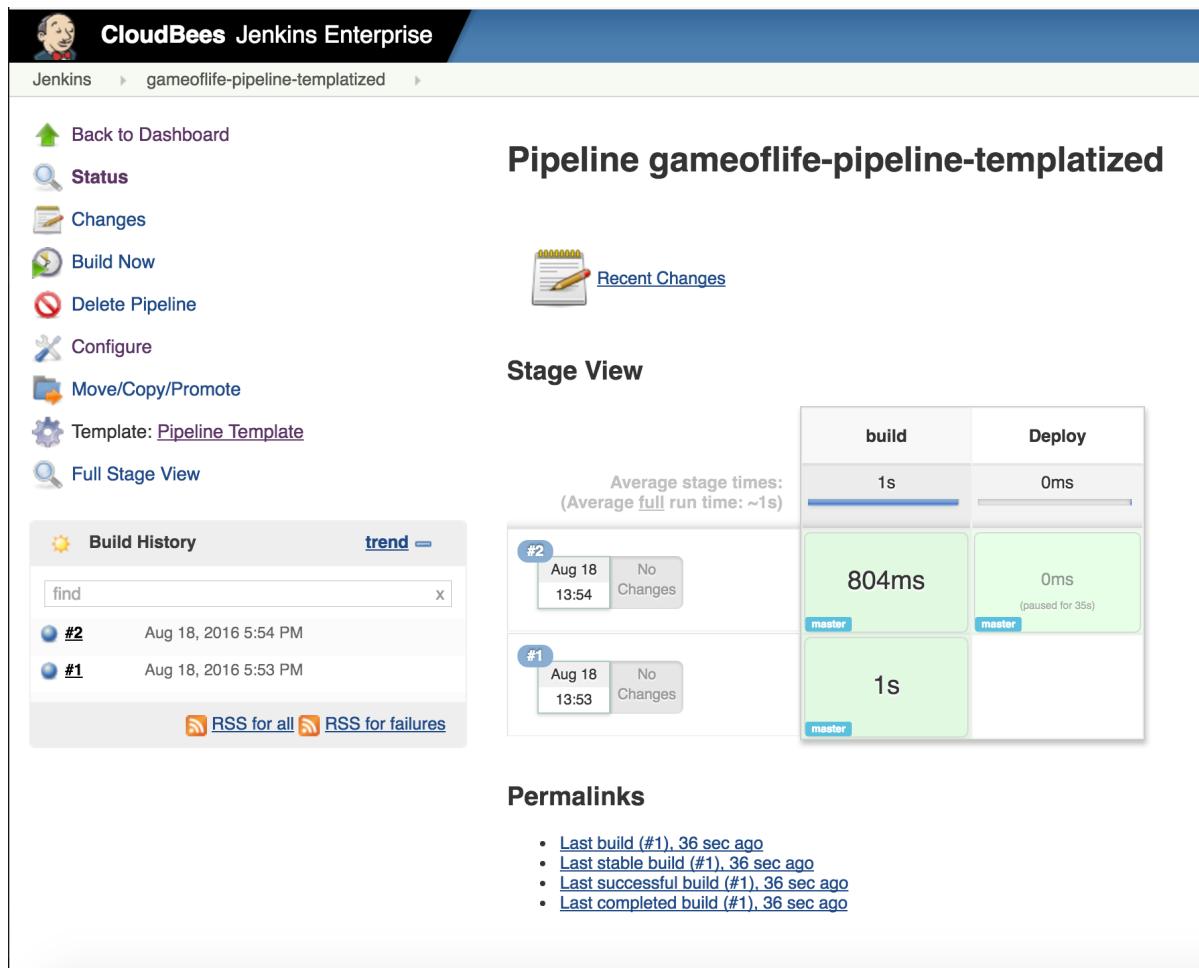


Figure 119. Deploy stage ran after approval

Go back to slides

Lab 14: CloudBees Support

Goal

This lab will teach users the basics of CloudBees support plugin. This plugin provides the ability to generate a bundle of all the commonly requested information used by CloudBees when resolving support issues.

Step 1: Verify / Install the Support plugin

Since some CloudBees customers purchase support contracts without being licensed for CloudBees Jenkins Enterprise, this plugin is also available for download and installation into any Jenkins installation.

Simply go to the related release page, pick the latest version and use the download from the **Download** section.

You can then install the downloaded plugin using **Manage Jenkins , Manage Plugins , Advanced , Upload Plugin**, and select the downloaded ".hpi" file.

The screenshot shows the Jenkins Plugin Manager interface. At the top, there are links for 'Jenkins' and 'Plugin Manager'. Below that, there are buttons for 'Back to Dashboard' and 'Manage Jenkins'. A search bar is labeled 'Filter: CloudBees Support'. The main area has tabs for 'Updates', 'Available', 'Installed' (which is selected), and 'Advanced'. Under the 'Enabled' section, there is a table with columns for 'Name', 'Version', 'Previously installed version', 'Pinned', and 'Uninstall'. The first row in the table is the 'CloudBees Support Plugin', which is checked and has a detailed description below it. Buttons for 'Downgrade to 3.6' and 'Unpin' are also present.

Figure 120. Plugin installed by default in CJE

Step 2: Generate a bundle

To generate a bundle, simply go to the CloudBees Support link on the Jenkins instance and Generate the Bundle with default options checked.

TIP Direct URL access from <http://localhost:5000/jenkins/support/>

The screenshot shows the 'CloudBees Support' page. On the left is a sidebar with icons for 'New Item', 'People', 'Build History', 'Manage Jenkins', 'Alerts', 'Support' (which is selected), 'Credentials', 'My Views', 'Pooled Virtual Machines', 'Groups', 'Roles', and 'Cluster Operations'. The main content area has sections for 'Build Queue' (No builds in the queue) and 'Build Executor Status' (1 Idle, 2 Idle). The 'Support' section title is 'CloudBees Support'. Below it, text says: 'In order to assist CloudBees Technical Support in providing you with an efficient response to your support requests please generate a bundle of the commonly requested support information and attach this bundle to your support ticket.' It then explains: 'It is best to include all of the following information in the support bundle, but if you are unable to provide some of the information due to corporate policy, you can either deselect the information to exclude prior to generating the bundle or edit the generated bundle to remove the specific information that you must exclude. Each bundle is a simple ZIP file containing mostly plain text files and you can examine and/or modify the bundle prior to sharing the bundle if you have any concerns about the information contained within.' A large list of items is shown with checkboxes, most of which are checked by default: Log Recorders, About browser, About Jenkins, About user (basic authentication details only), Administrative monitors, Build queue, Dump slave export tables (could reveal some memory leaks), Environment variables, File descriptors (Unix only), JVM process system metrics (Linux only), Load Statistics, All loggers currently enabled., Metrics, Networking Interface, Root CAs, System configuration (Linux only), System properties, Update Center, Slow Request Records, Deadlock Records, and Thread dumps. A 'Generate Bundle' button is at the bottom.

Figure 121. CloudBees Support Bundle page

The bundle will be generated and downloaded as a ZIP archive:

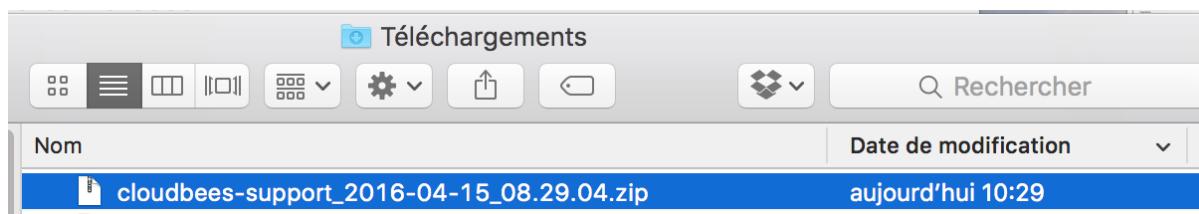


Figure 122. CloudBees Support Bundle ZIP