

# Build a Java app with Maven

## Table of Contents

- [Prerequisites](#)
- [Run Jenkins in Docker](#)
  - [On macOS and Linux](#)
  - [On Windows](#)
  - [Accessing the Jenkins/Blue Ocean Docker container](#)
  - [Setup wizard](#)
  - [Stopping and restarting Jenkins](#)
- [Fork and clone the sample repository on GitHub](#)
- [Create your Pipeline project in Jenkins](#)
- [Create your initial Pipeline as a Jenkinsfile](#)
- [Add a test stage to your Pipeline](#)
- [Add a final deliver stage to your Pipeline](#)
- [Wrapping up](#)

This tutorial shows you how to use Jenkins to orchestrate building a simple Java application with [Maven](#).

If you are a Java developer who uses Maven and who is new to CI/CD concepts, or you might be familiar with these concepts but don't know how to implement building your application using Jenkins, then this tutorial is for you.

The simple Java application (which you'll obtain from a sample repository on GitHub) outputs the string "Hello world!" and is accompanied by a couple of unit tests to check that the main application works as expected. The results of these tests are saved to a JUnit XML report.

**Duration:** This tutorial takes 20-40 minutes to complete (assuming you've already met the [prerequisites](#) below). The exact duration will depend on the speed of your machine and whether or not you've already [run Jenkins in Docker](#) from [another tutorial](#).

You can stop this tutorial at any point in time and continue from where you left off.

If you've already run through [another tutorial](#), you can skip the [Prerequisites](#) and [Run Jenkins in Docker](#) sections below and proceed on to [forking the sample repository](#). (Just ensure you have [Git](#) installed locally.) If you need to restart Jenkins, simply follow the restart instructions in [Stopping and restarting Jenkins](#) and then proceed on.

## Prerequisites

For this tutorial, you will require:

- A macOS, Linux or Windows machine with:
  - 256 MB of RAM, although more than 512MB is recommended.
  - 10 GB of drive space for Jenkins and your Docker images and containers.

- The following software installed:
  - [Docker](#) - Read more about installing Docker in the [Installing Docker](#) section of the [Installing Jenkins](#) page.  
**Note:** If you use Linux, this tutorial assumes that you are not running Docker commands as the root user, but instead with a single user account that also has access to the other tools used throughout this tutorial.
  - [Git](#) and optionally [GitHub Desktop](#).

## Run Jenkins in Docker

In this tutorial, you'll be running Jenkins as a Docker container from the [jenkinsci/blueocean](#) Docker image.

To run Jenkins in Docker, follow the relevant instructions below for either [macOS and Linux](#) or [Windows](#).

You can read more about Docker container and image concepts in the [Docker](#) and [Downloading and running Jenkins in Docker](#) sections of the [Installing Jenkins](#) page.

### On macOS and Linux

1. Open up a terminal window.
2. Run the jenkinsci/blueocean image as a container in Docker using the following [docker run](#) command (bearing in mind that this command automatically downloads the image if this hasn't been done):

```
docker run \
  --rm \
  -u root \
  -p 8080:8080 \
  -v jenkins-data:/var/jenkins_home \ (1)
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v "$HOME"/:/home \ (2)
  jenkinsci/blueocean
```

Maps the /var/jenkins\_home directory in the container to the Docker [volume](#) with the **1** name jenkins-data. If this volume does not exist, then this docker run command will automatically create the volume for you.

**2** Maps the \$HOME directory on the host (i.e. your local) machine (usually the /Users/<your-username> directory) to the /home directory in the container.

**Note:** If copying and pasting the command snippet above doesn't work, try copying and pasting this annotation-free version here:

```
docker run \
  --rm \
  -u root \
  -p 8080:8080 \
  -v jenkins-data:/var/jenkins_home \
  -v /var/run/docker.sock:/var/run/docker.sock \
  -v "$HOME"/:/home \
  jenkinsci/blueocean
```

3. Proceed to the [Setup wizard](#).

## On Windows

1. Open up a command prompt window.
2. Run the jenkinsci/blueocean image as a container in Docker using the following [docker run](#) command (bearing in mind that this command automatically downloads the image if this hasn't been done):

```
docker run ^  
  --rm ^  
  -u root ^  
  -p 8080:8080 ^  
  -v jenkins-data:/var/jenkins_home ^  
  -v /var/run/docker.sock:/var/run/docker.sock ^  
  -v "%HOMEPATH%":"/home ^  
  jenkinsci/blueocean
```

For an explanation of these options, refer to the [macOS and Linux](#) instructions above.

3. Proceed to the [Setup wizard](#).

## Accessing the Jenkins/Blue Ocean Docker container

If you have some experience with Docker and you wish or need to access the Jenkins/Blue Ocean Docker container through a terminal/command prompt using the [docker exec](#) command, you can add an option like `--name jenkins-tutorials` (with the [docker run](#) above), which would give the Jenkins/Blue Ocean Docker container the name "jenkins-tutorials".

This means you could access the Jenkins/Blue Ocean container (through a separate terminal/command prompt window) with a `docker exec` command like:

```
docker exec -it jenkins-tutorials bash
```

## Setup wizard

Before you can access Jenkins, there are a few quick "one-off" steps you'll need to perform.

### Unlocking Jenkins

When you first access a new Jenkins instance, you are asked to unlock it using an automatically-generated password.

1. After the 2 sets of asterisks appear in the terminal/command prompt window, browse to `http://localhost:8080` and wait until the **Unlock Jenkins** page appears.

# Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log (not sure where to find it?) and this file on the server:

`/var/jenkins_home/secrets/initialAdminPassword`

Please copy the password from either location and paste it below.

Administrator password

Continue

2. From your terminal/command prompt window again, copy the automatically-generated alphanumeric password (between the 2 sets of asterisks).

```
INFO: Pre-instantiating singletons in org.springframework.beans.factory.support.DefaultListableBeanFactory@24cf7404: defining beans [filter,legacy]; root of factory hierarchy
Sep 30, 2017 7:18:39 AM jenkins.install.SetupWizard init
INFO:

*****
*****
*****

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

3f064d3663814887964b662940572567

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*****
*****
*****

--> setting agent port for jnlp
--> setting agent port for jnlp... done
Sep 30, 2017 7:18:51 AM hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
Sep 30, 2017 7:18:52 AM hudson.model.UpdateSite updateData
INFO: Obtained the latest update center data file for UpdateSource default
Sep 30, 2017 7:18:52 AM hudson.WebAppMain$3 run
INFO: Jenkins is fully up and running
Sep 30, 2017 7:18:52 AM hudson.model.DownloadService$Downloadable load
INFO: Obtained the updated data file for hudson.tasks.Maven.MavenInstaller
Sep 30, 2017 7:18:58 AM hudson.model.DownloadService$Downloadable load
INFO: Obtained the updated data file for hudson.tools.JDKInstaller
Sep 30, 2017 7:18:59 AM hudson.model.AsyncPeriodicWork$1 run
INFO: Finished Download metadata. 25,543 ms
```

3. On the **Unlock Jenkins** page, paste this password into the **Administrator password** field and click **Continue**.

## Customizing Jenkins with plugins

After [unlocking Jenkins](#), the **Customize Jenkins** page appears.

On this page, click **Install suggested plugins**.

The setup wizard shows the progression of Jenkins being configured and the suggested plugins being installed. This process may take a few minutes.

### Creating the first administrator user

Finally, Jenkins asks you to create your first administrator user.

1. When the **Create First Admin User** page appears, specify your details in the respective fields and click **Save and Finish**.
2. When the **Jenkins is ready** page appears, click **Start using Jenkins**.

#### Notes:

- This page may indicate **Jenkins is almost ready!** instead and if so, click **Restart**.
  - If the page doesn't automatically refresh after a minute, use your web browser to refresh the page manually.
3. If required, log in to Jenkins with the credentials of the user you just created and you're ready to start using Jenkins!

### Stopping and restarting Jenkins

Throughout the remainder of this tutorial, you can stop the Jenkins/Blue Ocean Docker container by typing `Ctrl-C` in the terminal/command prompt window from which you ran the `docker run ...` command [above](#).

To restart the Jenkins/Blue Ocean Docker container:

1. Run the same `docker run ...` command you ran for [macOS](#), [Linux](#) or [Windows](#) above.  
**Note:** This process also updates the `jenkinsci/blueocean` Docker image, if an updated one is available.
2. Browse to `http://localhost:8080`.
3. Wait until the log in page appears and log in.

### Fork and clone the sample repository on GitHub

Obtain the simple "Hello world!" Java application from GitHub, by forking the sample repository of the application's source code into your own GitHub account and then cloning this fork locally.

1. Ensure you are signed in to your GitHub account. If you don't yet have a GitHub account, sign up for a free one on the [GitHub website](#).
2. Fork the [simple-java-maven-app](#) on GitHub into your local GitHub account. If you need help with this process, refer to the [Fork A Repo](#) documentation on the GitHub website for more information.
3. Clone your forked `simple-java-maven-app` repository (on GitHub) locally to your machine. To begin this process, do either of the following (where `<your-username>` is the name of your user account on your operating system):
  - If you have the GitHub Desktop app installed on your machine:

- a. In GitHub, click the green **Clone or download** button on your forked repository, then **Open in Desktop**.
- b. In GitHub Desktop, before clicking **Clone** on the **Clone a Repository** dialog box, ensure **Local Path** for:
  - macOS is `/Users/<your-username>/Documents/GitHub/simple-java-maven-app`
  - Linux is `/home/<your-username>/GitHub/simple-java-maven-app`
  - Windows is `C:\Users\<your-username>\Documents\GitHub\simple-java-maven-app`
- Otherwise:
  - a. Open up a terminal/command line prompt and cd to the appropriate directory on:
    - macOS - `/Users/<your-username>/Documents/GitHub/`
    - Linux - `/home/<your-username>/GitHub/`
    - Windows - `C:\Users\<your-username>\Documents\GitHub\` (although use a Git bash command line window as opposed to the usual Microsoft command prompt)
  - b. Run the following command to continue/complete cloning your forked repo:
 

```
git clone https://github.com/YOUR-GITHUB-ACCOUNT-NAME/simple-java-maven-app
```

 where YOUR-GITHUB-ACCOUNT-NAME is the name of your GitHub account.

## Create your Pipeline project in Jenkins

1. Go back to Jenkins, log in again if necessary and click **create new jobs** under **Welcome to Jenkins!**  
**Note:** If you don't see this, click **New Item** at the top left.
2. In the **Enter an item name** field, specify the name for your new Pipeline project (e.g. `simple-java-maven-app`).
3. Scroll down and click **Pipeline**, then click **OK** at the end of the page.
4. ( *Optional* ) On the next page, specify a brief description for your Pipeline in the **Description** field (e.g. An entry-level Pipeline demonstrating how to use Jenkins to build a simple Java application with Maven.)
5. Click the **Pipeline** tab at the top of the page to scroll down to the **Pipeline** section.
6. From the **Definition** field, choose the **Pipeline script from SCM** option. This option instructs Jenkins to obtain your Pipeline from Source Control Management (SCM), which will be your locally cloned Git repository.
7. From the **SCM** field, choose **Git**.

8. In the **Repository URL** field, specify the directory path of your locally cloned repository [above](#), which is from your user account/home directory on your host machine, mapped to the /home directory of the Jenkins container - i.e.
  - For macOS - /home/Documents/GitHub/simple-java-maven-app
  - For Linux - /home/GitHub/simple-java-maven-app
  - For Windows - /home/Documents/GitHub/simple-java-maven-app
9. Click **Save** to save your new Pipeline project. You're now ready to begin creating your Jenkinsfile, which you'll be checking into your locally cloned Git repository.

## Create your initial Pipeline as a Jenkinsfile

You're now ready to create your Pipeline that will automate building your Java application with Maven in Jenkins. Your Pipeline will be created as a Jenkinsfile, which will be committed to your locally cloned Git repository (simple-java-maven-app).

This is the foundation of "Pipeline-as-Code", which treats the continuous delivery pipeline as a part of the application to be versioned and reviewed like any other code. Read more about Pipeline and what a Jenkinsfile is in the [Pipeline](#) and [Using a Jenkinsfile](#) sections of the User Handbook.

First, create an initial Pipeline to download a Maven Docker image and run it as a Docker container (which will build your simple Java application). Also add a "Build" stage to the Pipeline that begins orchestrating this whole process.

1. Using your favorite text editor or IDE, create and save new text file with the name Jenkinsfile at the root of your local simple-java-maven-app Git repository.
2. Copy the following Declarative Pipeline code and paste it into your empty Jenkinsfile:

```
pipeline {
  agent {
    docker {
      image 'maven:3-alpine' (1)
      args '-v /root/.m2:/root/.m2' (2)
    }
  }
  stages {
    stage('Build') { (3)
      steps {
        sh 'mvn -B -DskipTests clean package' (4)
      }
    }
  }
}
```

1 This image parameter (of the [agent](#) section's docker parameter) downloads the [maven:3-apline Docker image](#) (if it's not already available on your machine) and runs this image as a separate container. This means that:

- You'll have separate Jenkins and Maven containers running locally in Docker.

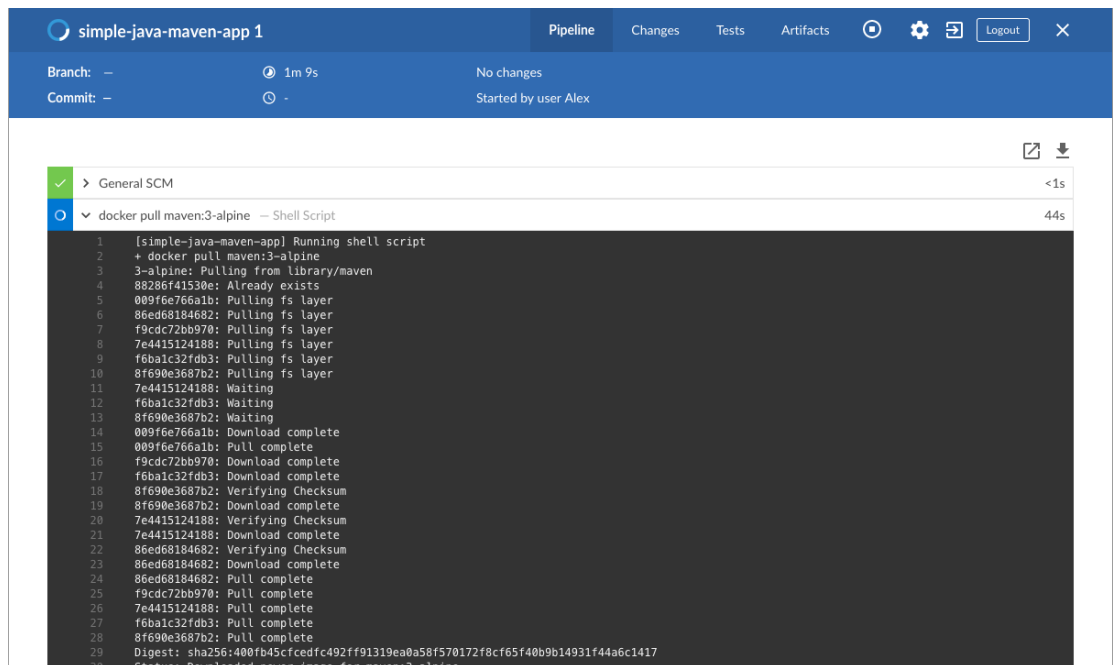
- The Maven container becomes the [agent](#) that Jenkins uses to run your Pipeline project. However, this container is short-lived - its lifespan is only that of the duration of your Pipeline's execution.

This `args` parameter creates a reciprocal mapping between the `/root/.m2` (i.e. Maven repository) directories in the short-lived Maven Docker container and that of your Docker host's filesystem. Explaining the details behind this is beyond the scope of this tutorial. However, the main reason for doing this is to ensure that the artifacts necessary to build your Java application (which Maven downloads while your Pipeline is being

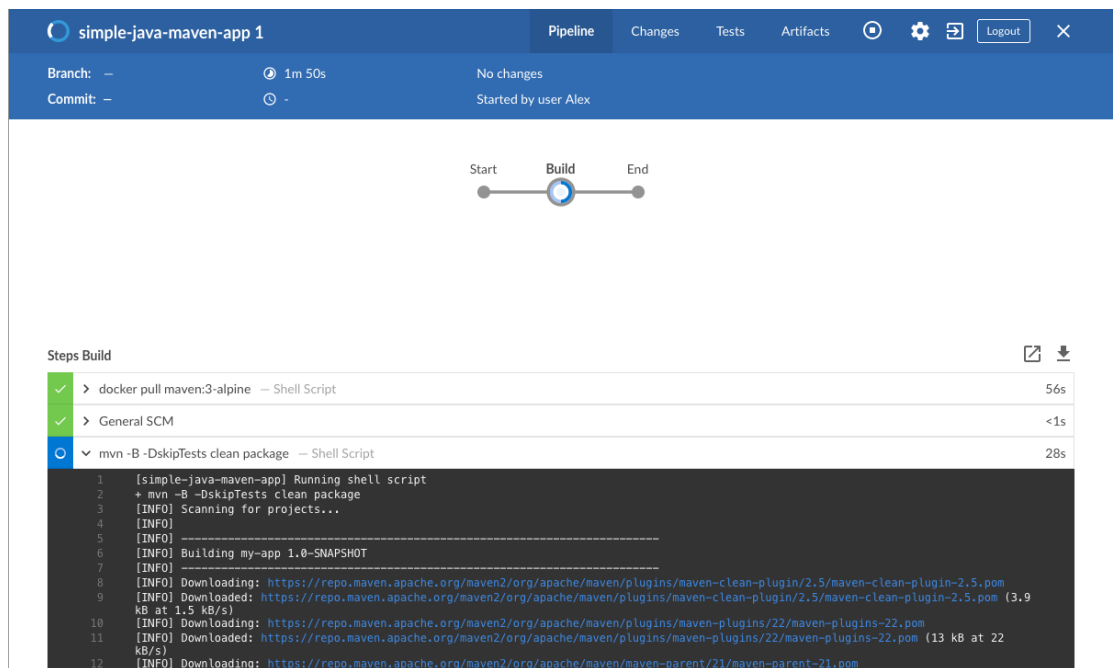
- 2 executed) are retained in the Maven repository beyond the lifespan of the Maven container. This prevents Maven from having to download the same artifacts during successive runs of your Jenkins Pipeline, which you'll be conducting later on. Be aware that unlike the Docker data volume you created for `jenkins-data` [above](#), the Docker host's filesystem is effectively cleared out each time Docker is restarted. This means you'll lose the downloaded Maven repository artifacts each time Docker restarts.
  - 3 Defines a [stage](#) (directive) called `Build` that appears on the Jenkins UI.
  - 4 This [sh](#) step (of the [steps](#) section) runs the Maven command to cleanly build your Java application (without running any tests).
3. Save your edited `Jenkinsfile` and commit it to your local `simple-java-maven-app` Git repository. E.g. Within the `simple-java-maven-app` directory, run the commands:  

```
git add .
then
git commit -m "Add initial Jenkinsfile"
```
  4. Go back to Jenkins again, log in again if necessary and click **Open Blue Ocean** on the left to access Jenkins's Blue Ocean interface.
  5. In the **This job has not been run** message box, click **Run**, then quickly click the **OPEN** link which appears briefly at the lower-right to see Jenkins running your Pipeline project. If you weren't able to click the **OPEN** link, click the row on the main Blue Ocean interface to access this feature.
- Note:** You may need to wait several minutes for this first run to complete. After making a clone of your local `simple-java-maven-app` Git repository itself, Jenkins:
- a. Initially queues the project to be run on the agent.
  - b. Downloads the Maven Docker image and runs it in a container on Docker.





- c. Runs the Build stage (defined in the Jenkinsfile) on the Maven container. During this time, Maven downloads many artifacts necessary to build your Java application, which will ultimately be stored in Jenkins's local Maven repository (in the Docker host's filesystem).



The Blue Ocean interface turns green if Jenkins built your Java application successfully.

simple-java-maven-app 1

Branch: —    2m 49s    No changes

Commit: —    a few seconds ago    Started by user Alex

Start    **Build**    End

Steps Build

Step	Duration
> docker pull maven:3-alpine — Shell Script	53s
> General SCM	<1s
> mvn -B -DskipTests clean package — Shell Script	1m 39s

6. Click the **X** at the top-right to return to the main Blue Ocean interface.

Jenkins

simple-java-maven-app

Run

STATUS	RUN	COMMIT	MESSAGE	DURATION	COMPLETED
✓	1	—	Started by user Alex	2m 49s	7 minutes ago

· e2b50bd · (no branch) · 19th September 2017 10:42 PM

## Add a test stage to your Pipeline

1. Go back to your text editor/IDE and ensure your Jenkinsfile is open.
2. Copy and paste the following Declarative Pipeline syntax immediately under the Build stage of your Jenkinsfile:

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

```

        post {
            always {
                junit 'target/surefire-reports/*.xml'
            }
        }
    }
}

```

so that you end up with:

```

pipeline {
    agent {
        docker {
            image 'maven:3-alpine'
            args '-v /root/.m2:/root/.m2'
        }
    }
    stages {
        stage('Build') {
            steps {
                sh 'mvn -B -DskipTests clean package'
            }
        }
        stage('Test') { (1)
            steps {
                sh 'mvn test' (2)
            }
            post {
                always {
                    junit 'target/surefire-reports/*.xml' (3)
                }
            }
        }
    }
}

```

**1** Defines a [stage](#) (directive) called Test that appears on the Jenkins UI.

This [sh](#) step (of the [steps](#) section) executes the Maven command to run the unit test on your simple Java application. This command also generates a JUnit XML report, which is

**2** saved to the target/surefire-reports directory (within the /var/jenkins\_home/workspace/simple-java-maven-app directory in the Jenkins container).

This [junit](#) step (provided by the [JUnit Plugin](#)) archives the JUnit XML report (generated by the mvn test command above) and exposes the results through the Jenkins interface.

**3** In Blue Ocean, the results are accessible through the **Tests** page of a Pipeline run. The [post](#) section's always condition that contains this junit step ensures that the step is *always* executed *at the completion* of the Test stage, regardless of the stage's outcome.

3. Save your edited Jenkinsfile and commit it to your local simple-java-maven-app Git repository. E.g. Within the simple-java-maven-app directory, run the commands:

```

git stage .
then
git commit -m "Add 'Test' stage"

```

4. Go back to Jenkins again, log in again if necessary and ensure you've accessed Jenkins's Blue Ocean interface.

5. Click **Run** at the top left, then quickly click the **OPEN** link which appears briefly at the lower-right to see Jenkins running your amended Pipeline project. If you weren't able to

click the **OPEN** link, click the *top* row on the Blue Ocean interface to access this feature.

**Note:** You'll notice from this run that Jenkins no longer needs to download the Maven Docker image. Instead, Jenkins only needs to run a new container from the Maven image downloaded previously. Also, if Docker had not restarted since you last ran the Pipeline [above](#), then no Maven artifacts need to be downloaded during the "Build" stage. Therefore, running your Pipeline this subsequent time should be much faster.

If your amended Pipeline ran successfully, here's what the Blue Ocean interface should look like. Notice the additional "Test" stage. You can click on the previous "Build" stage circle to access the output from that stage.

simple-java-maven-app 2

Branch: — 25s Changes by giles gas  
Commit: — a few seconds ago Started by user Alex

Start Build Test End

Steps Test

✓ mvn test — Shell Script 9s

```
1 [simple-java-maven-app] Running shell script
2 + mvn test
3 [INFO] Scanning for projects...
4 [INFO]
5 [INFO] -----
6 [INFO] Building my-app 1.0-SNAPSHOT
7 [INFO] -----
8 [INFO]
9 [INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ my-app ---
10 [WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
11 [INFO] skip non existing resourceDirectory /var/jenkins_home/workspace/simple-java-maven-app/src/main/resources
12 [INFO]
13 [INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ my-app ---
14 [INFO] Nothing to compile - all classes are up to date
15 [INFO]
16 [INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ my-app ---
17 [WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
18 [INFO] skip non existing resourceDirectory /var/jenkins_home/workspace/simple-java-maven-app/src/test/resources
```

6. Click the X at the top-right to return to the main Blue Ocean interface.

## Add a final deliver stage to your Pipeline

1. Go back to your text editor/IDE and ensure your Jenkinsfile is open.
2. Copy and paste the following Declarative Pipeline syntax immediately under the Test stage of your Jenkinsfile:

```
stage('Deliver') {
    steps {
        sh './jenkins/scripts/deliver.sh'
    }
}
```

so that you end up with:

```
pipeline {
    agent {
        docker {
            image 'maven:3-alpine'
            args '-v /root/.m2:/root/.m2'
        }
    }
    stages {
```

```

stage('Build') {
    steps {
        sh 'mvn -B -DskipTests clean package'
    }
}
stage('Test') {
    steps {
        sh 'mvn test'
    }
    post {
        always {
            junit 'target/surefire-reports/*.xml'
        }
    }
}
stage('Deliver') { (1)
    steps {
        sh './jenkins/scripts/deliver.sh' (2)
    }
}
}

```

1 Defines a new stage called Deliver that appears on the Jenkins UI.

This [sh](#) step (of the [steps](#) section) runs the shell script `deliver.sh` located in the `jenkins/scripts` directory from the root of the `simple-java-maven-app` repository.

2 Explanations about what this script does are covered in the `deliver.sh` file itself. As a general principle, it's a good idea to keep your Pipeline code (i.e. the Jenkinsfile) as tidy as possible and place more complex build steps (particularly for stages consisting of 2 or more steps) into separate shell script files like the `deliver.sh` file. This ultimately makes maintaining your Pipeline code easier, especially if your Pipeline gains more complexity.

3. Save your edited Jenkinsfile and commit it to your local `simple-java-maven-app` Git repository. E.g. Within the `simple-java-maven-app` directory, run the commands:

```

git stage .
then
git commit -m "Add 'Deliver' stage"

```

4. Go back to Jenkins again, log in again if necessary and ensure you've accessed Jenkins's Blue Ocean interface.

5. Click **Run** at the top left, then quickly click the **OPEN** link which appears briefly at the lower-right to see Jenkins running your amended Pipeline project. If you weren't able to click the **OPEN** link, click the *top* row on the Blue Ocean interface to access this feature. If your amended Pipeline ran successfully, here's what the Blue Ocean interface should look like. Notice the additional "Deliver" stage. Click on the previous "Test" and "Build" stage circles to access the outputs from those stages.

✓ simple-java-maven-app 3

PipelineChangesTestsArtifactsLogout

Branch: — 1m 3s Changes by gilesgas  
Commit: — a few seconds ago Started by user Alex

StartBuildTestDeliverEnd

Steps Deliver

✓ > ./jenkins/scripts/deliver.sh — Shell Script 43s

Here's what the output of the "Deliver" stage should look like, showing you the execution results of your Java application at the end.

```
10 [INFO] -----
11 [INFO] Building my-app 1.0-SNAPSHOT
12 [INFO] -----
13 [INFO] --- maven-jar-plugin:3.0.2:jar (default-cli) @ my-app ---
14 [INFO] --- maven-install-plugin:2.4:install (default-cli) @ my-app ---
15 [INFO] Installing /var/jenkins_home/workspace/simple-java-maven-app/target/my-app-1.0-SNAPSHOT.jar to
16 /root/.m2/repository/com/mycompany/app/my-app/1.0-SNAPSHOT/my-app-1.0-SNAPSHOT.jar
17 [INFO] Installing /var/jenkins_home/workspace/simple-java-maven-app/pom.xml to /root/.m2/repository/com/mycompany/app/my-app/1.0-
18 SNAPSHOT/my-app-1.0-SNAPSHOT.pom
19 [INFO] -----
20 [INFO] Building my-app 1.0-SNAPSHOT
21 [INFO] -----
22 [INFO] --- maven-help-plugin:2.2:evaluate (default-cli) @ my-app ---
23 [INFO] No artifact parameter specified, using 'com.mycompany.app:my-app:jar:1.0-SNAPSHOT' as project.
24 [INFO] -----
25 my-app
26 [INFO] -----
27 [INFO] BUILD SUCCESS
28 [INFO] -----
29 [INFO] Total time: 1.870 s
30 [INFO] Finished at: 2017-10-01T13:20:22Z
31 [INFO] Final Memory: 18M/96M
32 [INFO] -----
33 + set +x
34 The following complex command extracts the value of the <name/> element
35 within <project/> of your Java/Maven projects "pom.xml" file.
36 ++ mvn help:evaluate -Dexpression=project.name
37 ++ grep '^[\[]{'
38 + NAME=my-app
39 + set +x
40 The following complex command behaves similarly to the previous one but
41 extracts the value of the <version/> element within <project/> instead.
42 ++ mvn help:evaluate -Dexpression=project.version
43 ++ grep '^[\[]{'
44 + VERSION=1.0-SNAPSHOT
45 + set +x
46 The following command runs and outputs the execution of your Java
47 application (which Jenkins built using Maven) to the Jenkins UI.
48 + java -jar target/my-app-1.0-SNAPSHOT.jar
49 Hello World!
```

- Click the **X** at the top-right to return to the main Blue Ocean interface, which lists your previous Pipeline runs in reverse chronological order.

Jenkins

PipelinesAdministration

simple-java-maven-app

ActivityBranchesPull Requests

Run

STATUS	RUN	COMMIT	MESSAGE	DURATION	COMPLETED
✓	3	—	Add 'Deliver' stage	1m 3s	5 minutes ago
✓	2	—	Add 'Test' stage	25s	10 minutes ago
✓	1	—	Started by user Alex	2m 49s	25 minutes ago

e2b50bd · (no branch) · 19th September 2017 10:42 PM

## Wrapping up

Well done! You’ve just used Jenkins to build a simple Java application with Maven!

The "Build", "Test" and "Deliver" stages you created above are the basis for building more complex Java applications with Maven in Jenkins, as well as Java and Maven applications that integrate with other technology stacks.

Because Jenkins is extremely extensible, it can be modified and configured to handle practically any aspect of build orchestration and automation.

To learn more about what Jenkins can do, check out:

- The [Tutorials overview](#) page for other introductory tutorials.
- The [User Handbook](#) for more detailed information about using Jenkins, such as [Pipelines](#) (in particular [Pipeline syntax](#)) and the [Blue Ocean](#) interface.
- The [Jenkins blog](#) for the latest events, other tutorials and updates.

---

### [Was this page helpful?](#)

Please submit your feedback about this page through this [quick form](#).

Alternatively, if you don't wish to complete the quick form, you can simply indicate if you found this page helpful?

☐ Yes    ☐ No

Type the answer to 4 plus 2 before clicking "Submit" below.

Submit

See existing feedback [here](#).

[Improve this page](#) | [Page history](#)



The content driving this site is licensed under the Creative Commons Attribution-ShareAlike 4.0 license.

## Resources

- [Events](#)
- [Documentation](#)
- [Blog](#)

## Solutions

- [Android](#)
- [C/C++](#)
- [Docker](#)
- [Embedded](#)
- [GitHub](#)
- [Java](#)
- [PHP](#)
- [Continuous Delivery](#)
- [Python](#)
- [Ruby](#)

## Project

- [Issue tracker](#)
- [Wiki](#)
- [GitHub](#)
- [Jenkins on Jenkins](#)

## Community

- [Users mailing list](#)
- [Developers mailing list](#)
- [Special Interest Groups](#)
- [Twitter](#)
- [Reddit](#)
- [Merchandise](#)
- [Awards](#)