### Run Jenkins in Docker

In this tutorial, you’ll be running Jenkins as a Docker container from the [jenkinsci/blueocean](https://hub.docker.com/r/jenkinsci/blueocean/) Docker image.

To run Jenkins in Docker, follow the relevant instructions below for either [macOS and Linux](https://jenkins.io/doc/tutorials/build-a-java-app-with-maven/" \l "on-macos-and-linux) or [Windows](https://jenkins.io/doc/tutorials/build-a-java-app-with-maven/#on-windows).

You can read more about Docker container and image concepts in the [Docker](https://jenkins.io/doc/book/installing" \l "docker) and [Downloading and running Jenkins in Docker](https://jenkins.io/doc/book/installing#downloading-and-running-jenkins-in-docker)sections of the [Installing Jenkins](https://jenkins.io/doc/book/installing) 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](https://docs.docker.com/engine/reference/commandline/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 "$HOME":/home jenkinsci/blueocean

|  |  |
| --- | --- |
|  | Maps the /var/jenkins\_home directory in the container to the Docker [volume](https://docs.docker.com/engine/admin/volumes/volumes/) with the name jenkins-data. If this volume does not exist, then this docker run command will automatically create the volume for you. |
|  | 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

1. Proceed to the [Setup wizard](https://jenkins.io/doc/tutorials/build-a-java-app-with-maven/#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](https://docs.docker.com/engine/reference/commandline/run/) command (bearing in mind that this command automatically downloads the image if this hasn’t been done):
3. docker run ^
4. --rm ^
5. -u root ^
6. -p 8080:8080 ^
7. -v jenkins-data:/var/jenkins\_home ^
8. -v /var/run/docker.sock:/var/run/docker.sock ^
9. -v "%HOMEPATH%":/home ^

jenkinsci/blueocean

For an explanation of these options, refer to the [macOS and Linux](https://jenkins.io/doc/tutorials/build-a-java-app-with-maven/" \l "on-macos-and-linux) instructions above.

1. Proceed to the [Setup wizard](https://jenkins.io/doc/tutorials/build-a-java-app-with-maven/#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](https://docs.docker.com/engine/reference/commandline/exec/) command, you can add an option like --name jenkins-tutorials (with the [docker run](https://docs.docker.com/engine/reference/commandline/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.



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



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

##### **Customizing Jenkins with plugins**

After [unlocking Jenkins](https://jenkins.io/doc/tutorials/build-a-java-app-with-maven/#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](https://jenkins.io/doc/tutorials/build-a-java-app-with-maven/#run-jenkins-in-docker).

To restart the Jenkins/Blue Ocean Docker container:

1. Run the same docker run …​ command you ran for [macOS, Linux](https://jenkins.io/doc/tutorials/build-a-java-app-with-maven/" \l "on-macos-and-linux) or [Windows](https://jenkins.io/doc/tutorials/build-a-java-app-with-maven/#on-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.

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](https://github.com/).
2. Fork the [simple-java-maven-app](https://github.com/jenkins-docs/simple-java-maven-app) on GitHub into your local GitHub account. If you need help with this process, refer to the [Fork A Repo](https://help.github.com/articles/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:
     1. In GitHub, click the green **Clone or download** button on your forked repository, then **Open in Desktop**.
     2. 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:
     1. 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)
     2. 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](https://jenkins.io/doc/tutorials/build-a-java-app-with-maven/#fork-sample-repository), 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](https://jenkins.io/doc/book/pipeline) and [Using a Jenkinsfile](https://jenkins.io/doc/book/pipeline/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'

args '-v /root/.m2:/root/.m2'

}

}

stages {

stage('Build') {

steps {

sh 'mvn -B -DskipTests clean package'

}

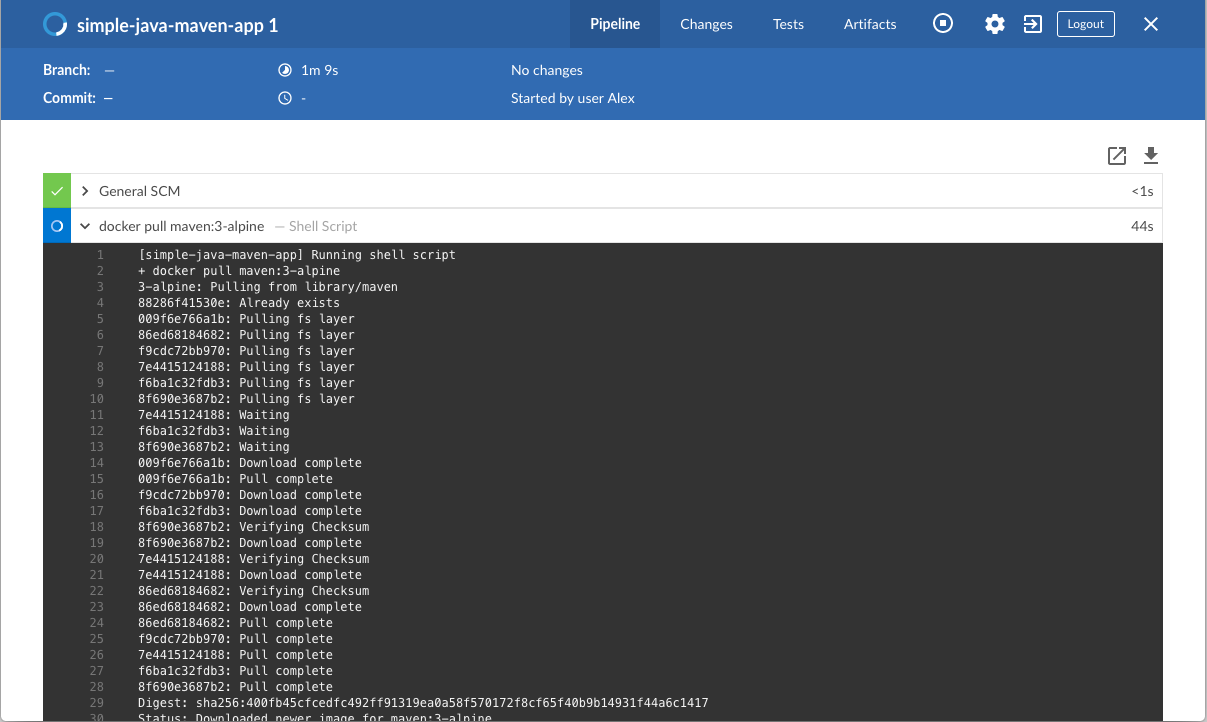
}

}

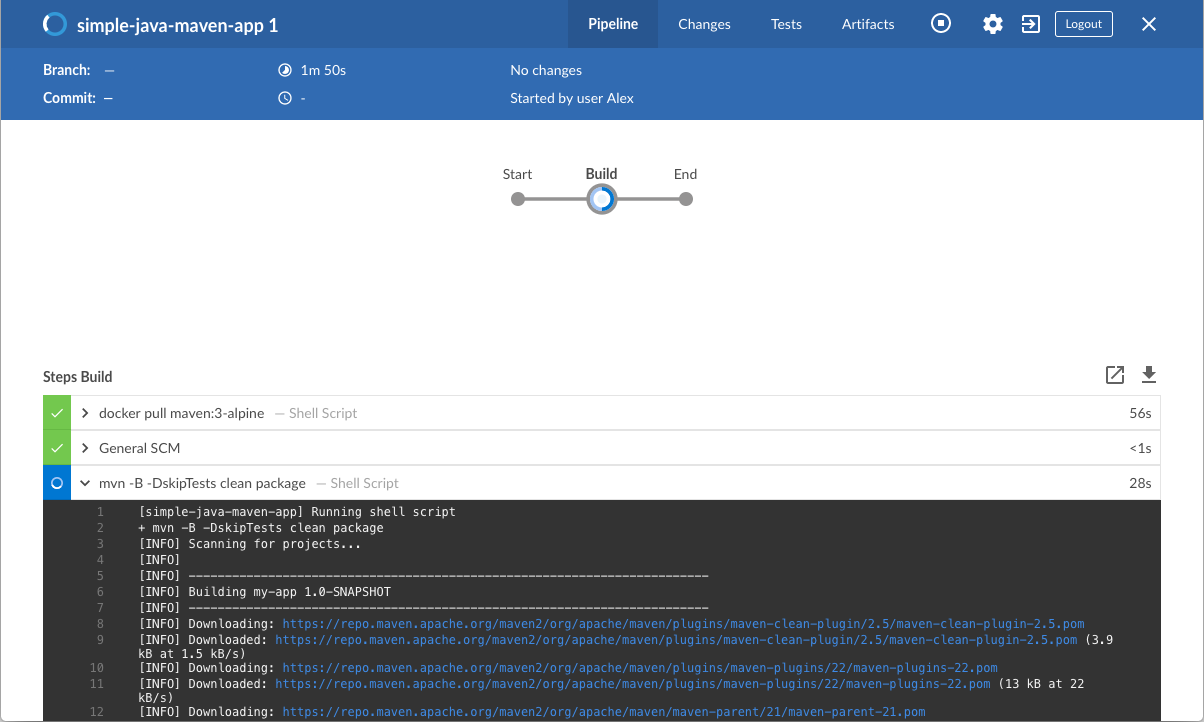
}

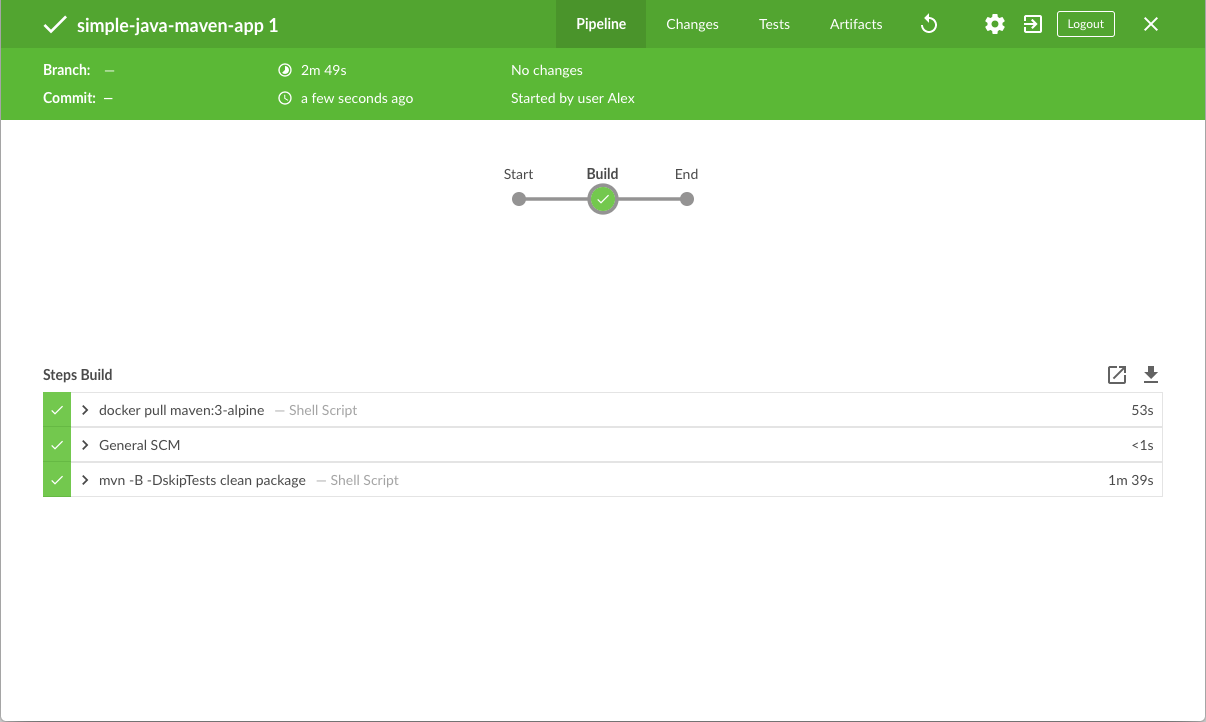
|  |  |
| --- | --- |
|  | This image parameter (of the [agent](https://jenkins.io/doc/book/pipeline/syntax#agent) section’s docker parameter) downloads the [maven:3-apline Docker image](https://hub.docker.com/_/maven/) (if it’s not already available on your machine) and runs this image as a separate container. This means that:   * + You’ll have separate Jenkins and Maven containers running locally in Docker.   + The Maven container becomes the [agent](https://jenkins.io/doc/book/glossary/#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 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](https://jenkins.io/doc/tutorials/build-a-java-app-with-maven/#download-and-run-jenkins-in-docker), 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. |
|  | Defines a [stage](https://jenkins.io/doc/book/pipeline/syntax/#stage) (directive) called Build that appears on the Jenkins UI. |
|  | This [sh](https://jenkins.io/doc/pipeline/steps/workflow-durable-task-step/" \l "code-sh-code-shell-script) step (of the [steps](https://jenkins.io/doc/book/pipeline/syntax/#steps) section) runs the Maven command to cleanly build your Java application (without running any tests). |

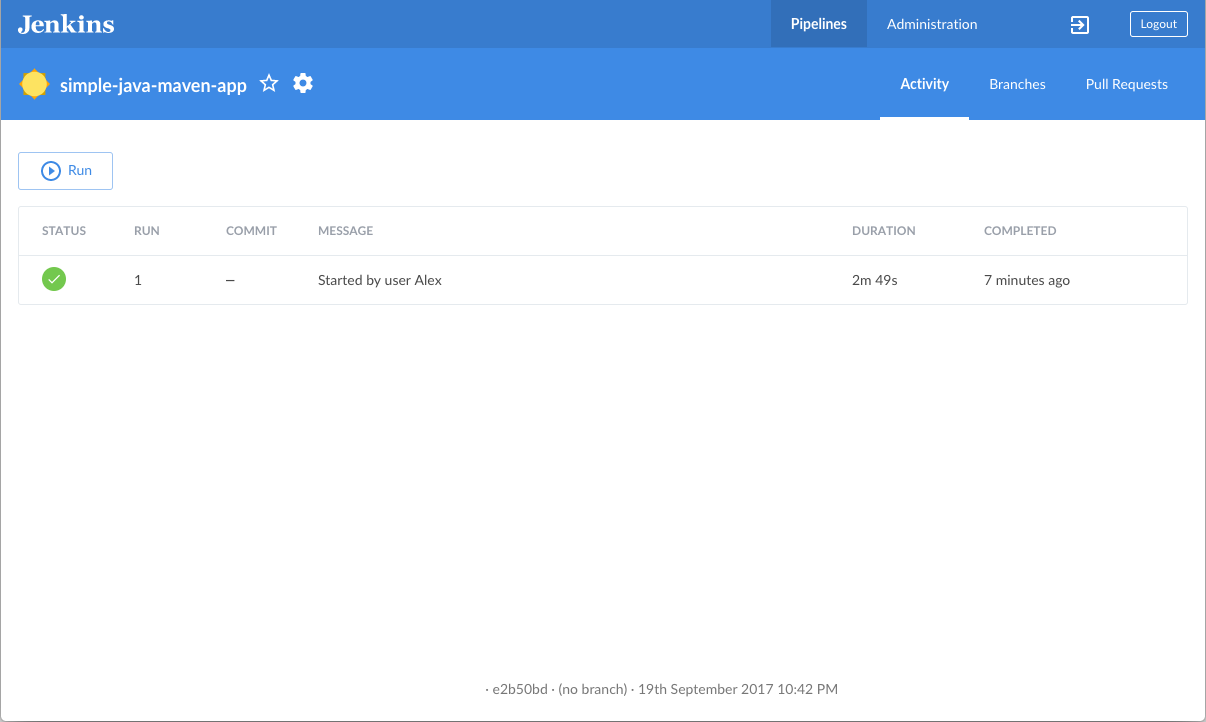
1. Save your edited Jenkinsfile and commit it to your local simple-java-maven-app Git repository. E.g. Within the simple-java-maven-app directory, run the commands:  
   git add .  
   then  
   git commit -m "Add initial Jenkinsfile"
2. Go back to Jenkins again, log in again if necessary and click **Open Blue Ocean** on the left to access Jenkins’s Blue Ocean interface.
3. 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:
   * Initially queues the project to be run on the agent.
   * Downloads the Maven Docker image and runs it in a container on Docker.



* + 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).



1. The Blue Ocean interface turns green if Jenkins built your Java application successfully.
2. 
3. Click the **X** at the top-right to return to the main Blue Ocean interface.



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:
3. stage('Test') {
4. steps {
5. sh 'mvn test'
6. }
7. post {
8. always {
9. junit 'target/surefire-reports/\*.xml'
10. }
11. }

}

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'

}

}

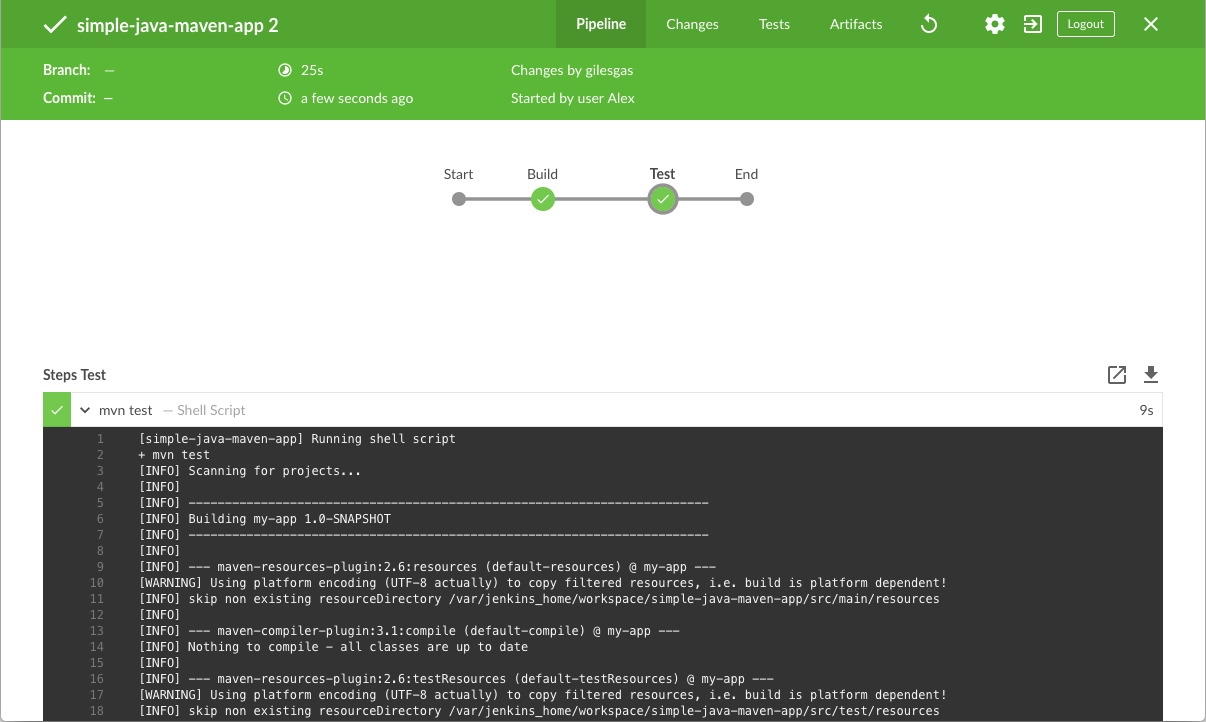
}

}

}

|  |  |
| --- | --- |
|  | Defines a [stage](https://jenkins.io/doc/book/pipeline/syntax/#stage) (directive) called Test that appears on the Jenkins UI. |
|  | This [sh](https://jenkins.io/doc/pipeline/steps/workflow-durable-task-step/" \l "code-sh-code-shell-script) step (of the [steps](https://jenkins.io/doc/book/pipeline/syntax/#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 saved to the target/surefire-reports directory (within the /var/jenkins\_home/workspace/simple-java-maven-app directory in the Jenkins container). |
|  | This [junit](https://jenkins.io/doc/pipeline/steps/junit/" \l "code-junit-code-archive-junit-formatted-test-results) step (provided by the [JUnit Plugin](https://jenkins.io/doc/pipeline/steps/junit)) archives the JUnit XML report (generated by the mvn test command above) and exposes the results through the Jenkins interface. In Blue Ocean, the results are accessible through the **Tests** page of a Pipeline run. The [post](https://jenkins.io/doc/book/pipeline/syntax/#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. |

1. 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"
2. Go back to Jenkins again, log in again if necessary and ensure you’ve accessed Jenkins’s Blue Ocean interface.
3. 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](https://jenkins.io/doc/tutorials/build-a-java-app-with-maven/#create-your-initial-pipeline-as-a-jenkinsfile), 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.



1. 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:
3. stage('Deliver') {
4. steps {
5. sh './jenkins/scripts/deliver.sh'
6. }

}

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') {

steps {

sh './jenkins/scripts/deliver.sh'

}

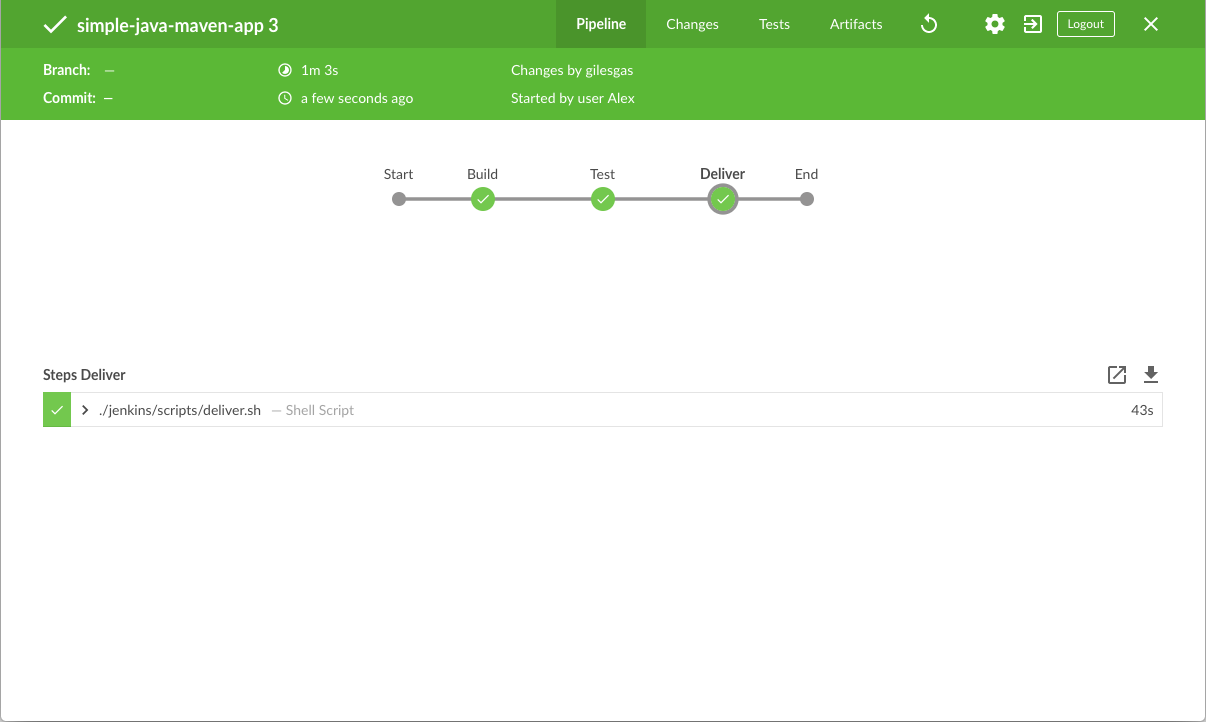
}

}

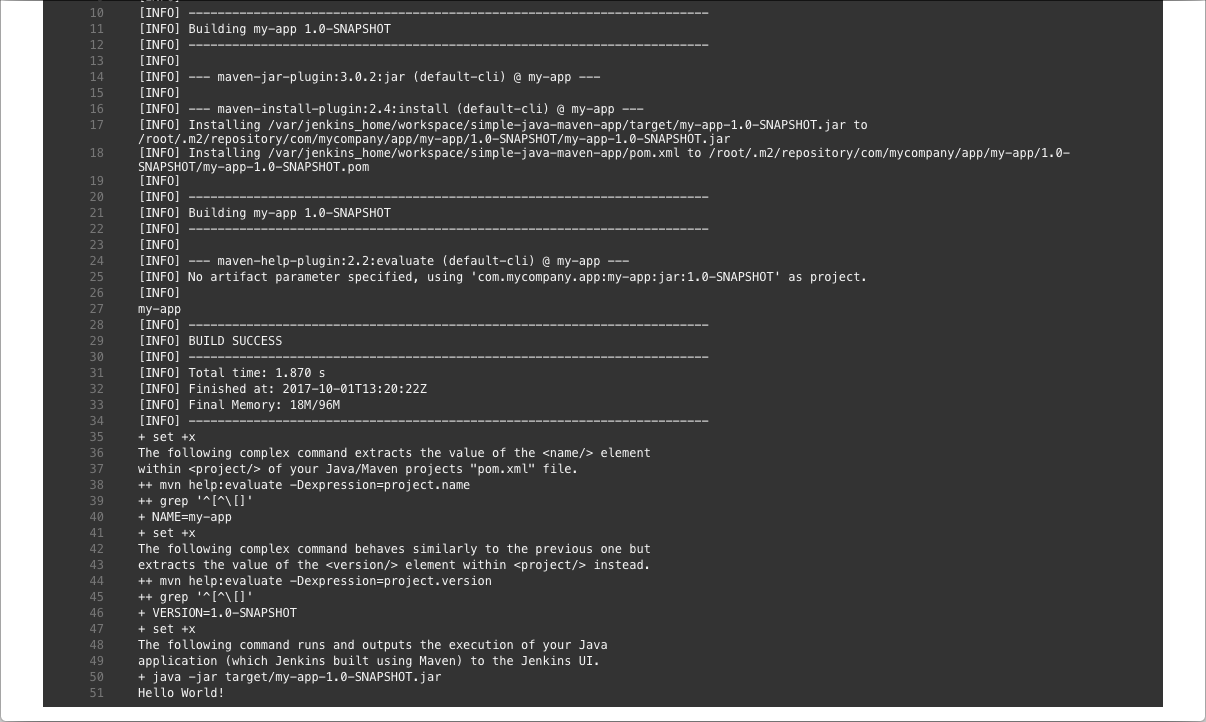
}

|  |  |
| --- | --- |
|  | Defines a new stage called Deliver that appears on the Jenkins UI. |
|  | This [sh](https://jenkins.io/doc/pipeline/steps/workflow-durable-task-step/" \l "code-sh-code-shell-script) step (of the [steps](https://jenkins.io/doc/book/pipeline/syntax/#steps) section) runs the shell script deliver.sh located in the jenkins/scripts directory from the root of the simple-java-maven-app repository. 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. |

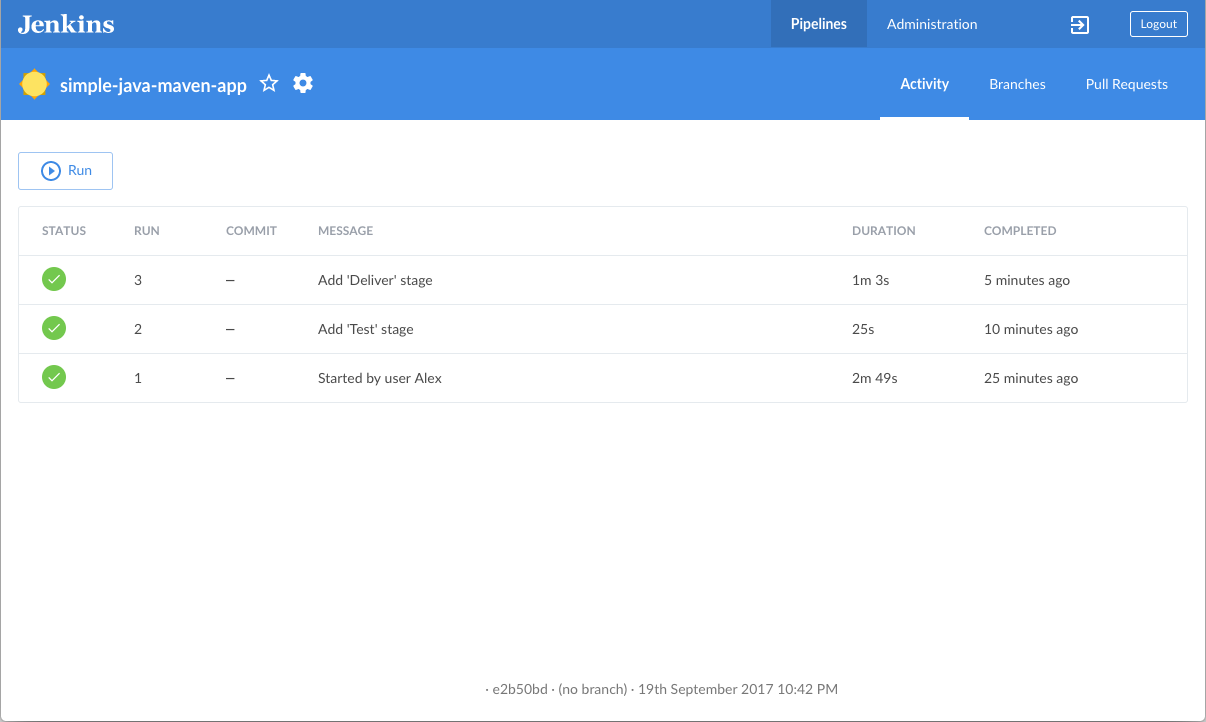
1. Save your edited Jenkinsfile and commit it to your local simple-java-maven-app Git repository. E.g. Within the simple-java-maven-app directory, run the commands:  
   git stage .  
   then  
   git commit -m "Add 'Deliver' stage"
2. Go back to Jenkins again, log in again if necessary and ensure you’ve accessed Jenkins’s Blue Ocean interface.
3. 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.



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



1. 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.



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.