# Building CI/CD pipelines with Jenkins

Building a DevOps pipeline from scratch. The core technology driving that initiative was [Jenkins](https://jenkins.io/" \t "https://opensource.com/article/19/9/_blank), an open source tool to build continuous integration and continuous delivery (CI/CD) pipelines.

At Citi, there was a separate team that provided dedicated Jenkins pipelines with a stable master-slave node setup, but the environment was only used for quality assurance (QA), staging, and production environments. The development environment was still very manual, and our team needed to automate it to gain as much flexibility as possible while accelerating the development effort. This is the reason we decided to build a CI/CD pipeline for DevOps. And the open source version of Jenkins was the obvious choice due to its flexibility, openness, powerful plugin-capabilities, and ease of use.

Step-by-step walkthrough on how you can build a CI/CD pipeline using Jenkins.

## What is a pipeline?

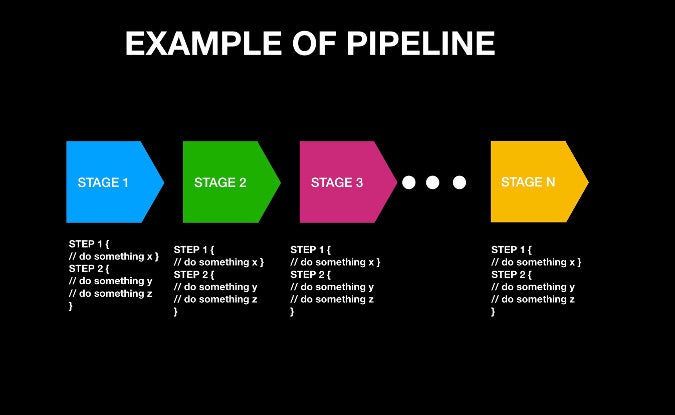
Before jumping into the tutorial, it's helpful to know something about CI/CD pipelines.

To start, it is helpful to know that Jenkins itself is not a pipeline. Just creating a new Jenkins job does not construct a pipeline. Think about Jenkins like a remote control—it's the place you click a button. What happens when you do click a button depends on what the remote is built to control. Jenkins offers a way for other application APIs, software libraries, build tools, etc. to plug into Jenkins, and it executes and automates the tasks. On its own, Jenkins does not perform any functionality but gets more and more powerful as other tools are plugged into it.

A pipeline is a separate concept that refers to the groups of events or jobs that are connected together in a sequence:

A **pipeline** is a sequence of events or jobs that can be executed.

The easiest way to understand a pipeline is to visualize a sequence of stages, like this:



Here, you should see two familiar concepts: Stage and Step.

* **Stage:** A block that contains a series of steps. A stage block can be named anything; it is used to visualize the pipeline process.
* **Step:** A task that says what to do. Steps are defined inside a stage block.

In the example diagram above, Stage 1 can be named "Build," "Gather Information," or whatever, and a similar idea is applied for the other stage blocks. "Step" simply says what to execute, and this can be a simple print command (e.g., **echo "Hello, World"**), a program-execution command (e.g., **java HelloWorld**), a shell-execution command (e.g., **chmod 755 Hello**), or any other command—as long as it is recognized as an executable command through the Jenkins environment.

The Jenkins pipeline is provided as a codified script typically called a **Jenkinsfile**, although the file name can be different. Here is an example of a simple Jenkins pipeline file.

// Example of Jenkins pipeline script

pipeline {

  stages {

    stage("Build") {

       steps {

          // Just print a Hello, Pipeline to the console

          echo "Hello, Pipeline!"

          // Compile a Java file. This requires JDKconfiguration from Jenkins

          javac HelloWorld.java

          // Execute the compiled Java binary called HelloWorld. This requires JDK configuration from Jenkins

          java HelloWorld

          // Executes the Apache Maven commands, clean then package. This requires Apache Maven configuration from Jenkins

          mvn clean package ./HelloPackage

          // List the files in current directory path by executing a default shell command

          sh "ls -ltr"

       }

   }

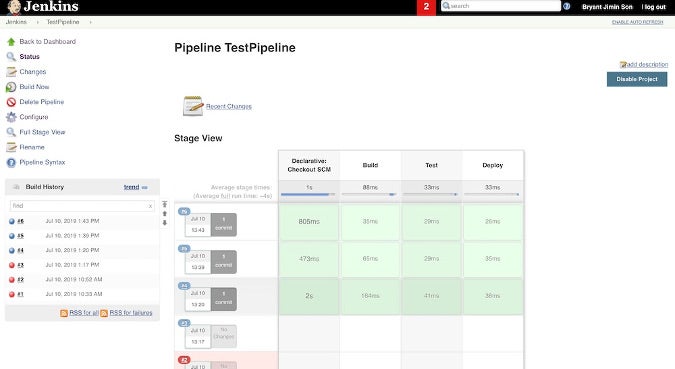
   // And next stages if you want to define further...

 } // End of stages  
} // End of pipeline

It's easy to see the structure of a Jenkins pipeline from this sample script. Note that some commands, like **java**, **javac**, and **mvn**, are not available by default, and they need to be installed and configured through Jenkins. Therefore:

A **Jenkins pipeline** is the way to execute a Jenkins job sequentially in a defined way by codifying it and structuring it inside multiple blocks that can include multiple steps containing tasks.

OK. Now that you understand what a Jenkins pipeline is, I'll show you how to create and execute a Jenkins pipeline. At the end of the tutorial, you will have built a Jenkins pipeline like this:



## How to build a Jenkins pipeline

To make this tutorial easier to follow, I created a sample [GitHub repository](https://github.com/bryantson/CICDPractice" \t "https://opensource.com/article/19/9/_blank) and a video tutorial.

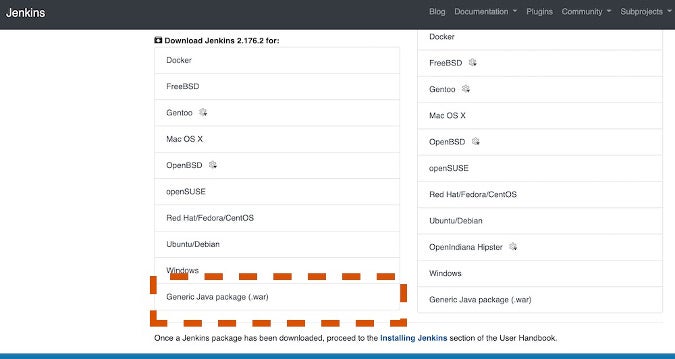
Before starting this tutorial, you'll need:

* **Java Development Kit:** If you don't already have it, install a JDK and add it to the environment path so a Java command (like **java jar**) can be executed through a terminal. This is necessary to leverage the Java Web Archive (WAR) version of Jenkins that is used in this tutorial (although you can use any other distribution).
* **Basic computer operations:** You should know how to type some code, execute basic Linux commands through the shell, and open a browser.

Let's get started.

### Step 1: Download Jenkins

Navigate to the [Jenkins download page](https://jenkins.io/download/" \t "https://opensource.com/article/19/9/_blank). Scroll down to **Generic Java package (.war)** and click on it to download the file; save it someplace where you can locate it easily. (If you choose another Jenkins distribution, the rest of tutorial steps should be pretty much the same, except for Step 2.) The reason to use the WAR file is that it is a one-time executable file that is easily executable and removable.

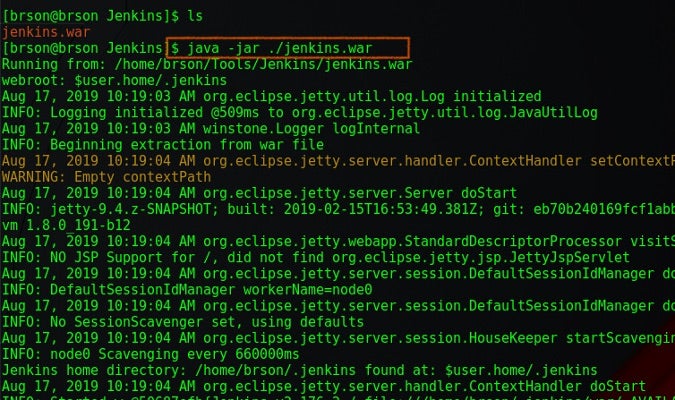


### Step 2: Execute Jenkins as a Java binary

Open a terminal window and enter the directory where you downloaded Jenkins with **cd <your path>**. (Before you proceed, make sure JDK is installed and added to the environment path.) Execute the following command, which will run the WAR file as an executable binary:

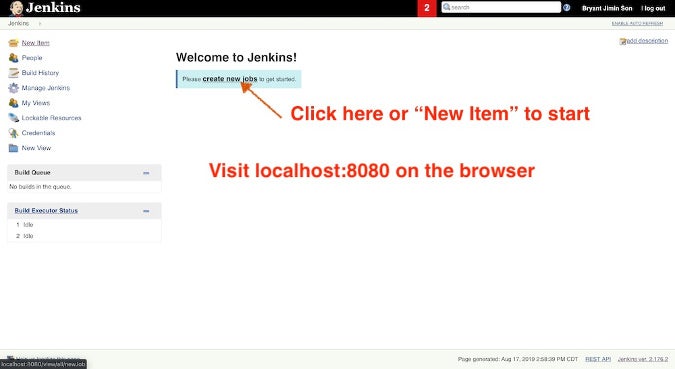
java -jar ./jenkins.war

If everything goes smoothly, Jenkins should be up and running at the default port 8080.



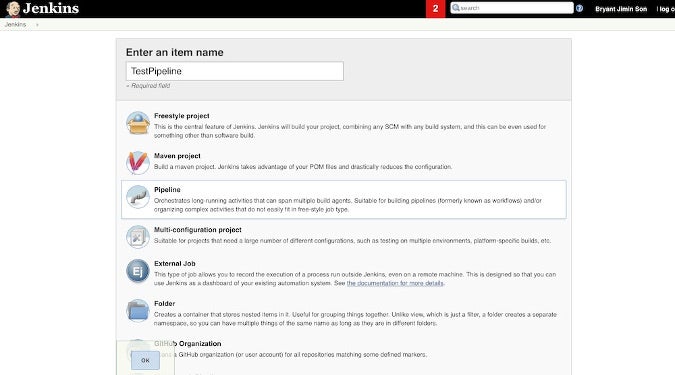
### Step 3: Create a new Jenkins job

Open a web browser and navigate to **localhost:8080**. Unless you have a previous Jenkins installation, it should go straight to the Jenkins dashboard. Click **Create New Jobs**. You can also click **New Item** on the left.



### Step 4: Create a pipeline job

In this step, you can select and define what type of Jenkins job you want to create. Select **Pipeline** and give it a name (e.g., TestPipeline). Click **OK** to create a pipeline job.

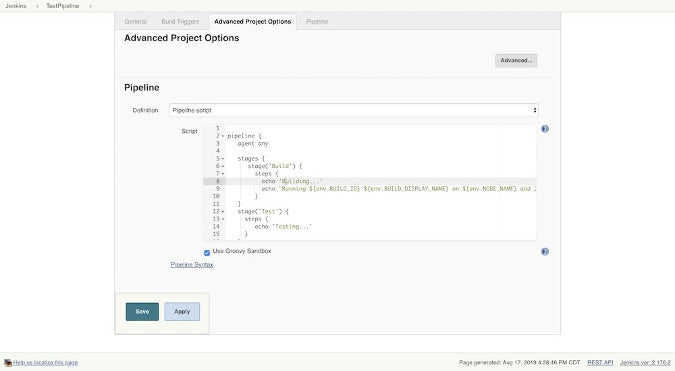


You will see a Jenkins job configuration page. Scroll down to find**Pipeline section**. There are two ways to execute a Jenkins pipeline. One way is by directly writing a pipeline script on Jenkins, and the other way is by retrieving the Jenkins file from SCM (source control management). We will go through both ways in the next two steps.

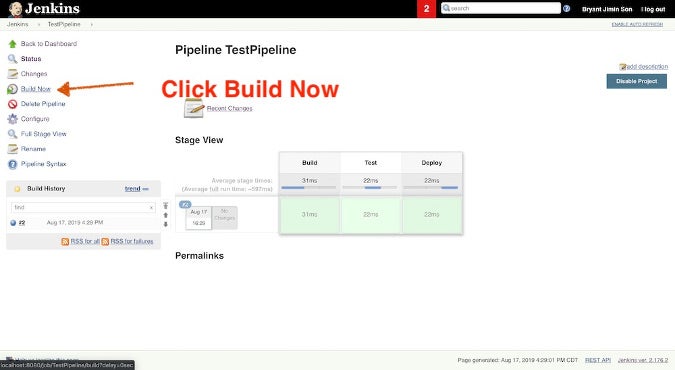
### Step 5: Configure and execute a pipeline job through a direct script

To execute the pipeline with a direct script, begin by copying the contents of the [sample Jenkinsfile](https://github.com/bryantson/CICDPractice" \t "https://opensource.com/article/19/9/_blank) from GitHub. Choose **Pipeline script** as the **Destination** and paste the **Jenkinsfile** contents in **Script**. Spend a little time studying how the Jenkins file is structured. Notice that there are three Stages: Build, Test, and Deploy, which are arbitrary and can be anything. Inside each Stage, there are Steps; in this example, they just print some random messages.

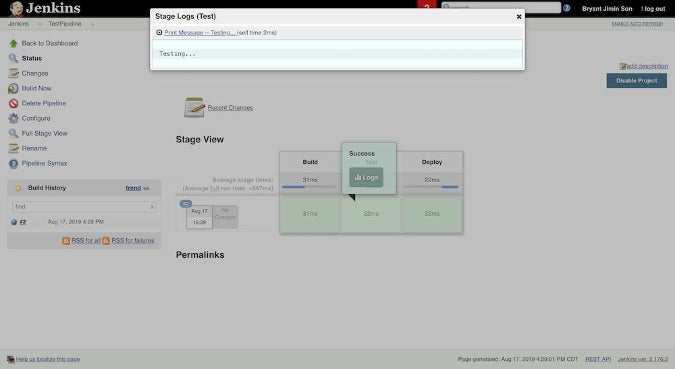
Click **Save** to keep the changes, and it should automatically take you back to the Job Overview.



To start the process to build the pipeline, click **Build Now**. If everything works, you will see your first pipeline (like the one below).

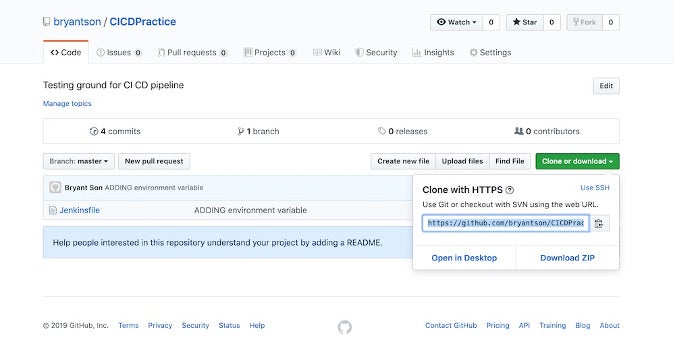


To see the output from the pipeline script build, click any of the Stages and click **Log**. You will see a message like this.

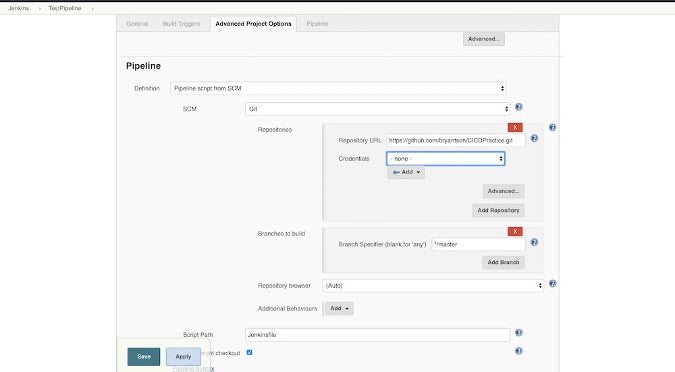


### Step 6: Configure and execute a pipeline job with SCM

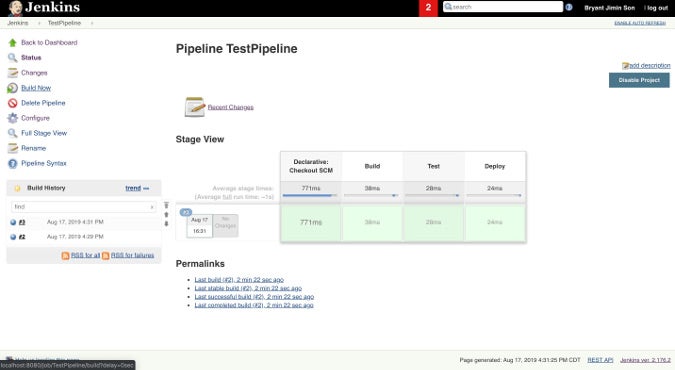
Now, switch gears: In this step, you will Deploy the same Jenkins job by copying the **Jenkinsfile** from a source-controlled GitHub. In the same [GitHub repository](https://github.com/bryantson/CICDPractice" \t "https://opensource.com/article/19/9/_blank), pick up the repository URL by clicking **Clone or download** and copying its URL.



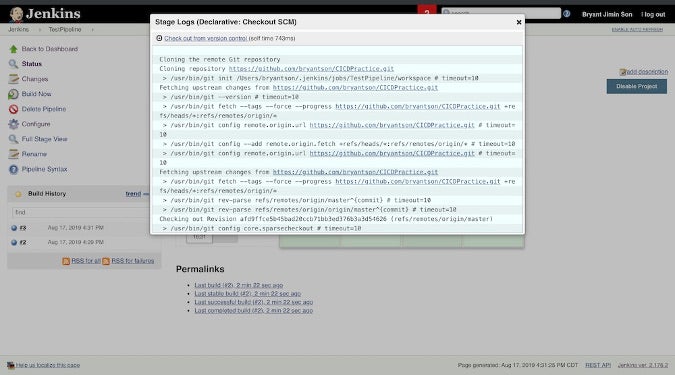
Click **Configure** to modify the existing job. Scroll to the **Advanced Project Options** setting, but this time, select the **Pipeline script from SCM** option in the **Destination** dropdown. Paste the GitHub repo's URL in the **Repository URL**, and type **Jenkinsfile** in the **Script Path**. Save by clicking the **Save** button.



To build the pipeline, once you are back to the Task Overview page, click **Build Now** to execute the job again. The result will be the same as before, except you have one additional stage called **Declaration: Checkout SCM**.



To see the pipeline's output from the SCM build, click the Stage and view the **Log** to check how the source control cloning process went.



## Do more than print messages

Congratulations! You've built your first Jenkins pipeline!

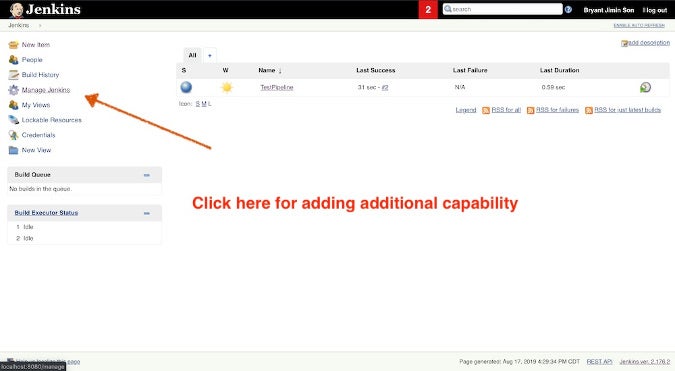
"But wait," you say, "this is very limited. I cannot really do anything with it except print dummy messages." That is OK. So far, this tutorial provided just a glimpse of what a Jenkins pipeline can do, but you can extend its capabilities by integrating it with other tools. Here are a few ideas for your next project:

* Build a multi-staged Java build pipeline that takes from the phases of pulling dependencies from JAR repositories like Nexus or Artifactory, compiling Java codes, running the unit tests, packaging into a JAR/WAR file, and deploying to a cloud server.
* Implement the advanced code testing dashboard that will report back the health of the project based on the unit test, load test, and automated user interface test with Selenium.
* Construct a multi-pipeline or multi-user pipeline automating the tasks of executing Ansible playbooks while allowing for authorized users to respond to task in progress.
* Design a complete end-to-end DevOps pipeline that pulls the infrastructure resource files and configuration files stored in SCM like GitHub and executing the scripts through various runtime programs.

Follow any of the tutorials at the end of this article to get into these more advanced cases.

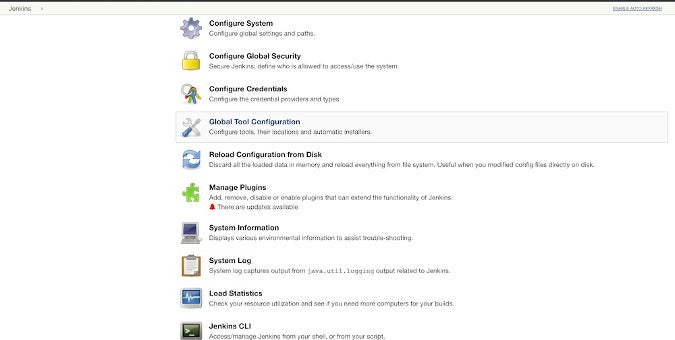
### Manage Jenkins

From the main Jenkins dashboard, click **Manage Jenkins**.



### Global tool configuration

There are many options available, including managing plugins, viewing the system log, etc. Click **Global Tool Configuration**.



### Add additional capabilities

Here, you can add the JDK path, Git, Gradle, and so much more. After you configure a tool, it is just a matter of adding the command into your Jenkinsfile or executing it through your Jenkins script.

