

DevOps in Action

SCM Hands-on

Github - Create an account

- Go to <https://github.com/join> in a web browser.

Create your personal account

Username *

✓

This will be your username. You can add the name of your organization later.

Email address *

✓

We'll occasionally send updates about your account to this inbox. We'll never share your email address with anyone.

Password *

Make sure it's at least 15 characters OR at least 8 characters including a number and a lowercase letter. [Learn more.](#)

Verify account

✓

wikiHow to Create an Account on GitHub

Github - Create an account

- **Enter your personal details**
- **Click the "Create an account" button.**
- Complete the CAPTCHA puzzle.
- Click the "Verify email address" button in the message from GitHub.
- Select your preferences and click Submit.
- Open Inbox and search for email from - noreply@github.com
 - Click - "Verify Email Address"

GitHub Repositories

- Contain all the repositories on which the user is working.

Github - fork your application code

- Visit - <https://github.com/atingupta2005/hello-world-maven>
- A fork is a copy of a repository.
- Forking a repository allows you to freely experiment with changes without affecting the original project.

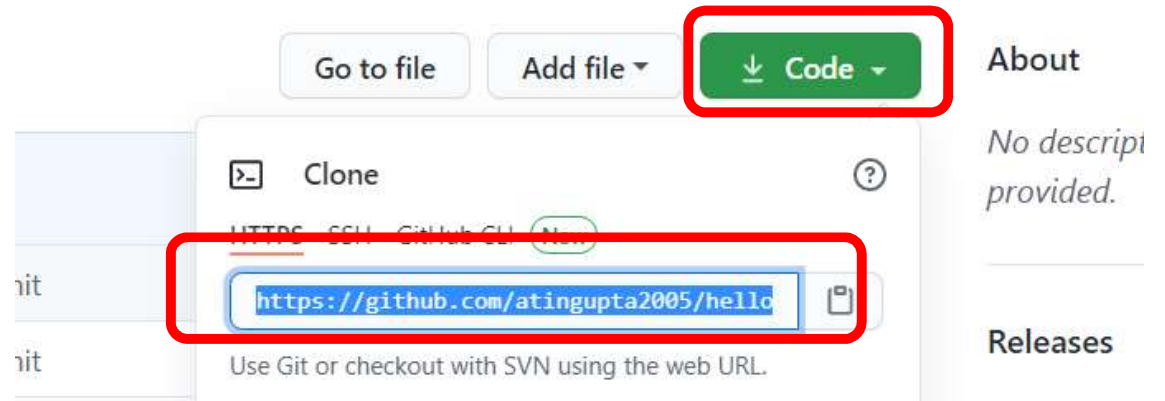
The screenshot shows the GitHub interface for the repository `atingupta2005 / hello-world-maven`. At the top right, there are buttons for `Watch` (1), `Star` (0), and `Fork` (0). The `Fork` button is highlighted with a red rectangular box. Below these buttons is a navigation bar with links for `Code`, `Issues`, `Pull requests`, `Actions`, `Projects`, `Wiki`, `Security`, and `Insights`. Below the navigation bar, there are buttons for `master` (1 branch), `0 tags`, `Go to file`, `Add file`, and a green `Code` button. Below these buttons, there is a table of commits. The first commit is by `atingupta2005` with the message `first commit`, hash `ae8a60e`, and timestamp `2 hours ago`. It has `1 commit` and a file `src` is listed. Below this, there is a `Releases` section with the text `No description, website, or topics provided.`

Commit	Hash	Time	Commits
atingupta2005 first commit	ae8a60e	2 hours ago	1 commit

File	Commit	Time
src	first commit	2 hours ago

Git clone the github code

- Cloning a repository pulls down a full copy of all the repository data that GitHub has at that point in time
- The git clone command is used to create a copy of a specific repository or branch within a repository.



Use maven to compile & package
java source code

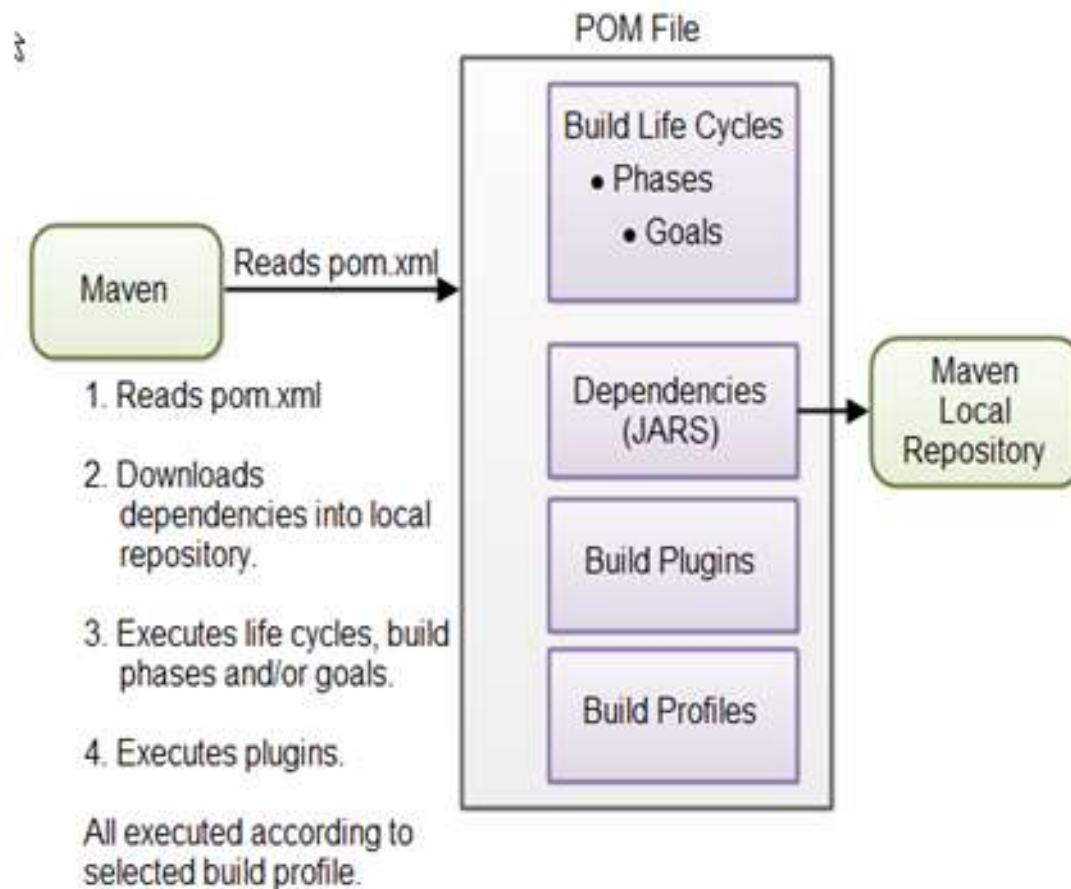
What is Maven?

- A powerful project management tool that is based on POM (Project Object Model)
- It is used for project build, dependency and documentation
- It can be used for building and managing any Java-based project

What can Maven do?

- Can easily build a project
- Can add jars, plugins and other dependencies based on needs
- Helpful in updating central repository of JAR's and other dependencies
- Can build any number of projects into packaging as JAR, WAR etc

Overview of Maven core concepts



Steps

- Install JDK 8
- Install Maven
- Set up the project
- Write a Test
- Go through the source code of maven project
- Define a simple Maven build - pom.xml
- Declare Dependencies
- Build Java code

Commands - Install maven

- `sudo apt -y update`
- `# Check if java already installed?`
- `java --version`

- `#Run below steps if Java not installed`
- `sudo apt install -y openjdk-8-jdk`

- `# Install maven`
- `sudo apt install -y maven`

Commands - Install jdk

- `git clone https://github.com/atingupta2005/hello-world-maven.git`
- `cd hello-world-maven/`
- `mvn compile`

Understanding Builds

- The process of translating source code into an executable application is called a build.

Build Tools

- Apache Maven: allows building application written in Java
- Gradle
- Ant
- NAnt
- MsBuild

Deploy .jar file manually

- # Make sure to cd into the project directory:
- pwd
 - /home/atingupta2005/hello-world-maven
- mvn package
- java -jar target/gs-maven-0.1.0.jar

Maven build lifecycle

- 1. Compile
 - Source code is compiled.
- 2. Test
 - Launches the unit test placed at src/test/java folder
- 3. Validate
 - To validate that the POM is correctly formed according to model version definition.
- 4. Package
 - To group the compiled code in the specified distributable format (jar, war, etc.).
- 5. Install
 - Installs packaged project into local repository. Then, it can be used by other projects.
- 6. Deploy
 - Similar to install
 - Puts the final package on the shared repository

Continuous Integration Tool

Jenkins - Deploy Jenkins on Ubuntu server

- Step 1: Install Java
- Step 2: Add the Jenkins Repository
- Step 3: Install Jenkins
- Step 4: Modify Firewall to Allow Jenkins
- Step 5: Set up Jenkins

Jenkins - Step 1: Install Java

- `java --version`
- #Run below steps if Java not installed
- `sudo apt update`
- `sudo apt install -y openjdk-8-jdk`

Jenkins - Step 2: Add the Jenkins Repository

- `wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -`
- `sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'`

Jenkins - Step 3: Install Jenkins

- `sudo apt -y update`
- `sudo apt install -y jenkins`
- `systemctl status jenkins`

Jenkins - Step 4: Modify Firewall to Allow Jenkins

- `sudo ufw allow 8080`
- `sudo ufw status`
- # Also make sure to add the port # 8080 in Inbound Rules in Azure/AWS Portal

Jenkins - Step 5: Set up Jenkins

- Visit:
 - http://your_ip_or_domain:8080
- Take password:
 - `sudo cat /var/lib/jenkins/secrets/initialAdminPassword`
- Copy the password from your terminal, paste it into the Administrator password field and click Continue.
- Click on the Install suggested plugins box
- Once the plugins are installed, you will be prompted to set up the first admin user
- Fill out all required information and click Save and Continue

Continuous Integration setup - Jenkins and Github

Introduction

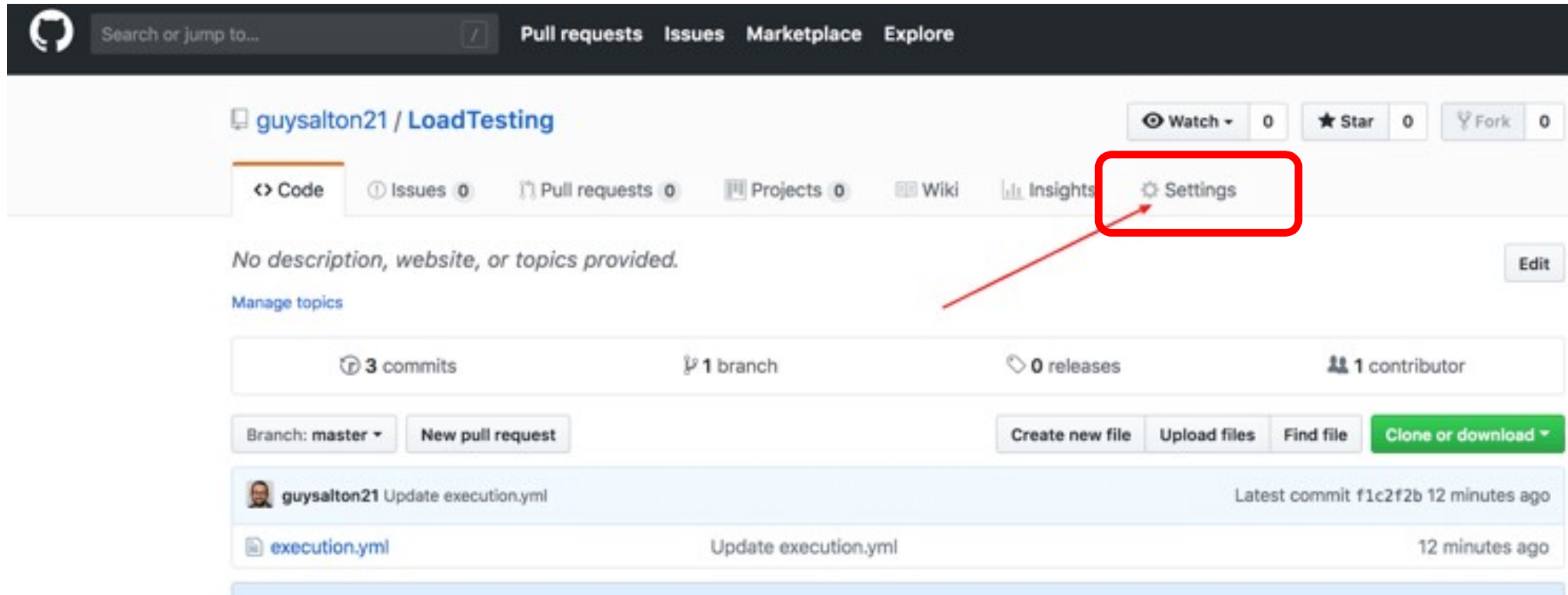
- One of the basic steps of implementing CI/CD is integrating your SCM (Source Control Management) tool with your CI tool.
- This saves you time and keeps your project updated all the time.
- One of the most popular and valuable SCM tools is GitHub.
- We will:
 - Schedule build
 - Pull code and data files from your GitHub repository to Jenkins machine
 - Automatically trigger each build on the Jenkins server, after each Commit on Git repository

Github Repo

- Fork Github repo:
 - <https://github.com/atingupta2005/hello-world-maven>

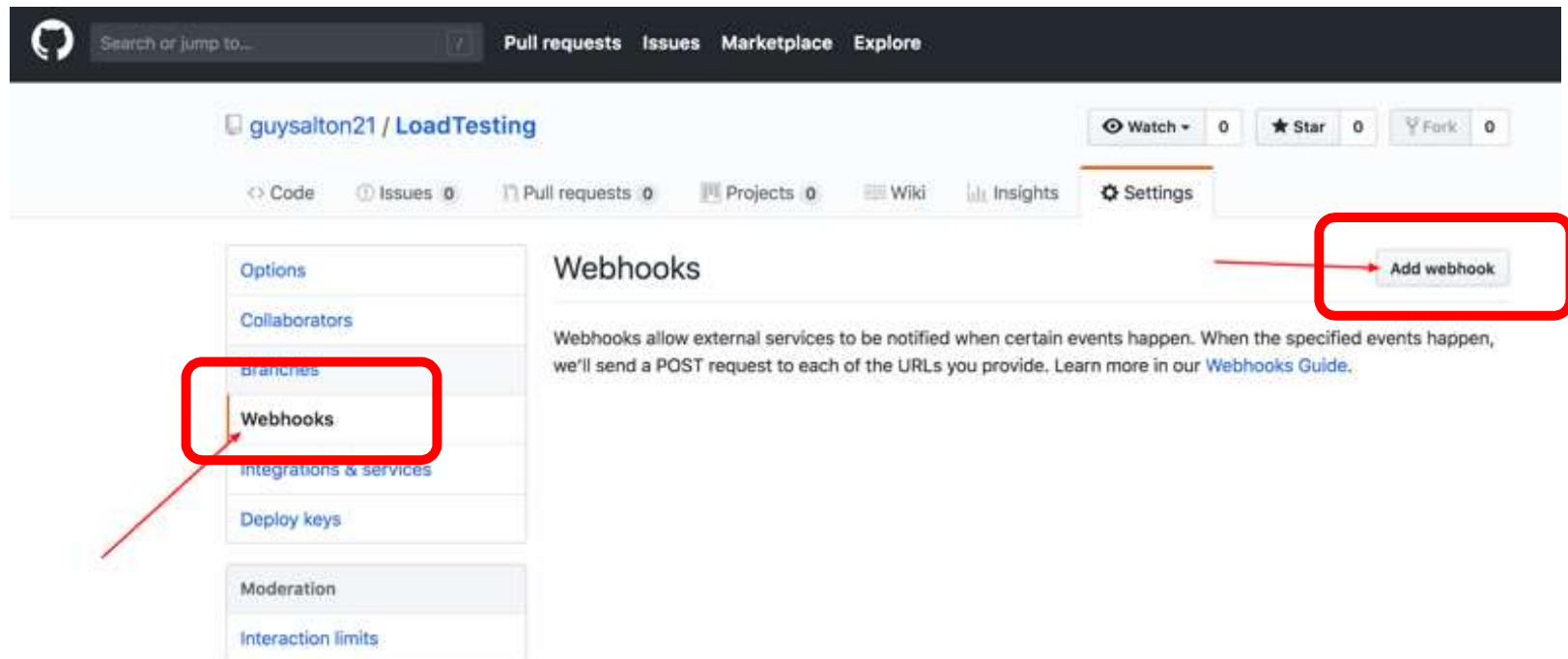
Configuring GitHub

- Go to your GitHub repository and click on 'Settings'.



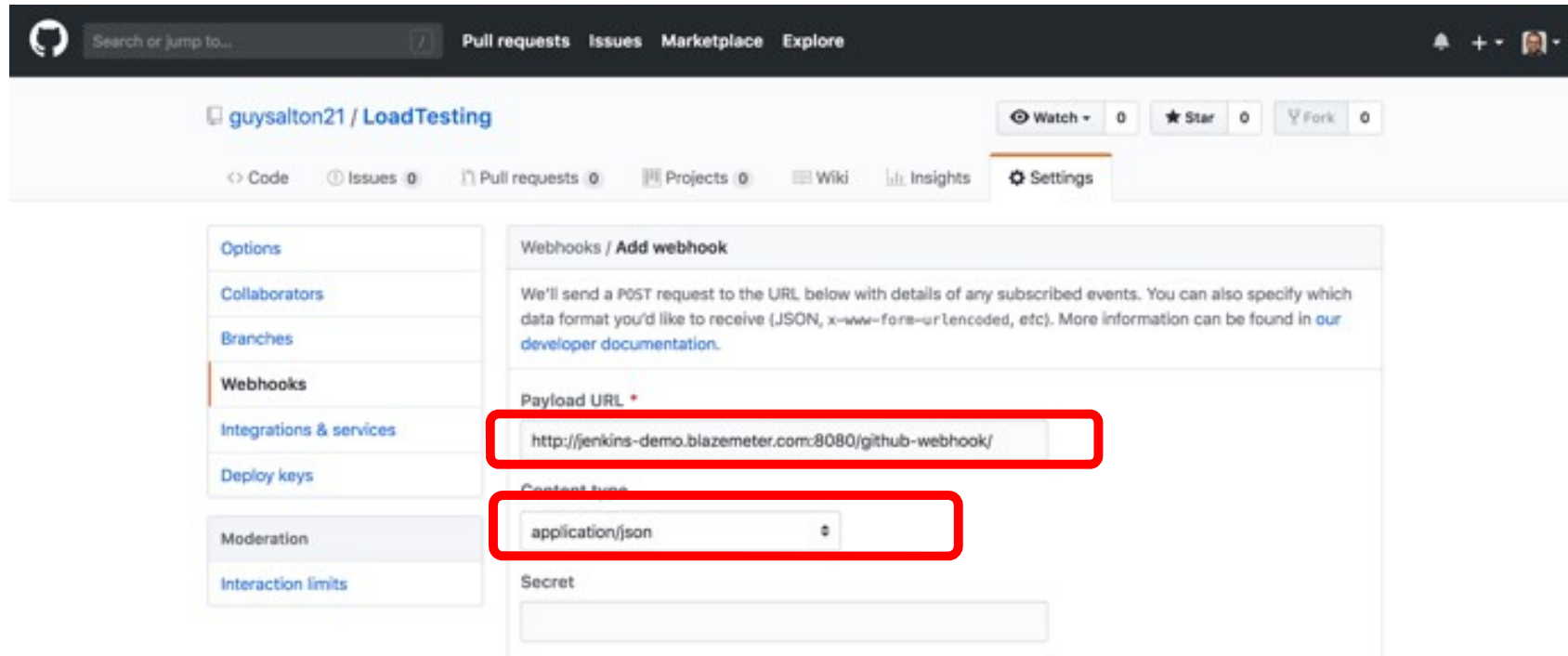
Configuring GitHub

- Step 2: Click on Webhooks and then click on 'Add webhook'.



Configuring GitHub

- Step 3: in the 'Payload URL' field, paste your Jenkins environment URL.
- At the end of this URL add /github-webhook/
- In the 'Content type' select 'application/json' and leave the 'Secret' field empty.



The screenshot shows the GitHub 'Add webhook' configuration page for the repository 'guysalton21 / LoadTesting'. The left sidebar contains a list of settings: Options, Collaborators, Branches, Webhooks (selected), Integrations & services, Deploy keys, Moderation, and Interaction limits. The main content area is titled 'Webhooks / Add webhook' and includes a description: 'We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in our [developer documentation](#).' Below this, there are three fields: 'Payload URL *' with the value 'http://jenkins-demo.blazemeter.com:8080/github-webhook/' (highlighted with a red box), 'Content type' with a dropdown menu showing 'application/json' (also highlighted with a red box), and 'Secret' which is an empty text field.

Configuring GitHub

- Step 4: in the 'Which events would you like to trigger this webhook?' 'Let me select individual events.'
- Then, check 'Pull Requests' and 'Push'
- At the end of this option, make sure webhook'.

Which events would you like to trigger this webhook?

☐ Just the push event.

☐ Send me everything.

☒ Let me select individual events.

<input type="checkbox"/> Check runs Check run is created, requested, rerequested, or completed.	<input type="checkbox"/> Check suites Check suite is requested, rerequested, or completed.
<input type="checkbox"/> Commit comments Commit or diff commented on.	<input type="checkbox"/> Branch or tag creation Branch or tag created.
<input type="checkbox"/> Branch or tag deletion Branch or tag deleted.	<input type="checkbox"/> Deployments Repository deployed.
<input type="checkbox"/> Deployment statuses Deployment status updated from the API.	<input type="checkbox"/> Forks Repository forked.
<input type="checkbox"/> Wiki Wiki page updated.	<input type="checkbox"/> Issue comments Issue comment created, edited, or deleted.
<input type="checkbox"/> Issues Issue opened, edited, deleted, transferred, closed, reopened, assigned, unassigned, labeled, unlabeled, milestoned, or demilestoned.	<input type="checkbox"/> Labels Label created, edited or deleted.
<input type="checkbox"/> Collaborator add, remove, or changed Collaborator added to, removed from, or has changed permissions for a repository.	<input type="checkbox"/> Milestones Milestone created, closed, opened, edited, or deleted.

Configuring GitHub

The screenshot shows the GitHub webhook configuration page. It features a list of event types on the left and a list of selected events on the right. Red boxes and arrows highlight specific elements: 'Pull requests' and 'Pushes' are boxed in red on the right, with arrows pointing to them from the left. 'Active' is boxed in red at the bottom, with an arrow pointing to it from the left. The 'Add webhook' button is also highlighted with a red arrow pointing to it from the left.

☐ Visibility changes
Repository changes from private to public.

☐ Pull request reviews
Pull request review submitted, edited, or dismissed.

☒ **Pushes**
Git push to a repository.

☐ Repositories
Repository created, deleted, archived, unarchived, publicized, or privatized.

☐ Repository vulnerability alerts
Vulnerability alert created, resolved, or dismissed on a repository.

☐ Team adds
Team added or modified on a repository.

☒ **Pull requests**
Pull request opened, closed, reopened, edited, assigned, unassigned, review requested, review request removed, labeled, unlabeled, or synchronized.

☐ Pull request review comments
Pull request diff comment created, edited, or deleted.

☐ Releases
Release published in a repository.

☐ Repository imports
Repository import succeeded, failed, or cancelled.

☐ Statuses
Commit status updated from the API.

☐ Watches
User stars a repository.

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

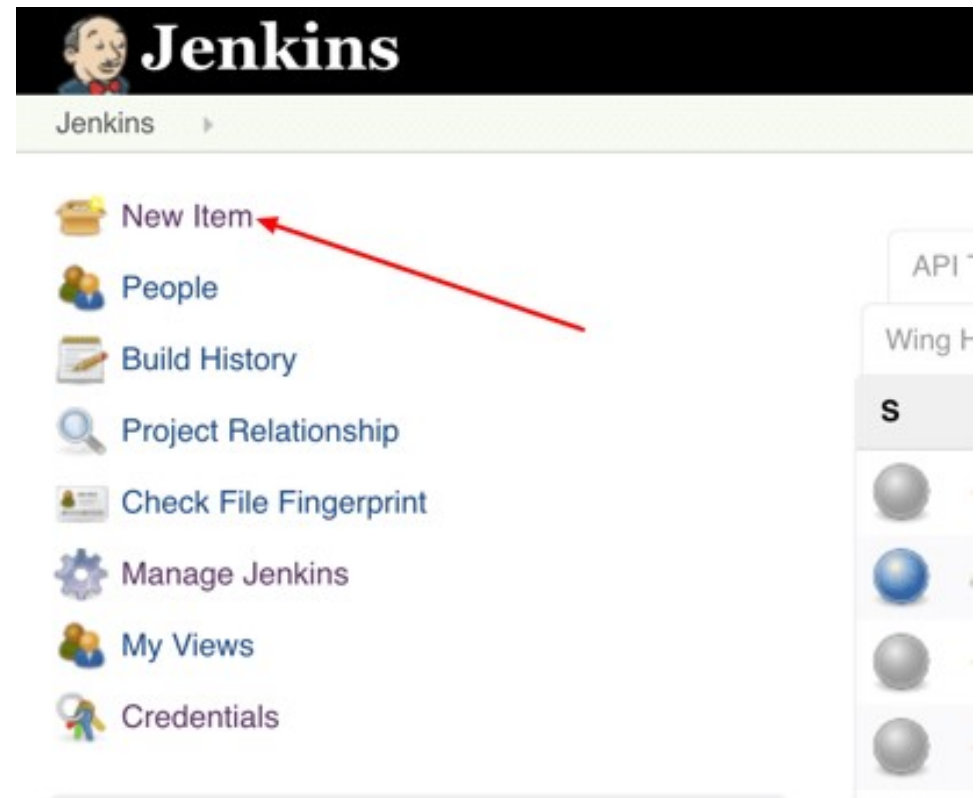
Add webhook

Configuring GitHub

- We're done with the configuration on GitHub's side! Now let's move on to Jenkins.

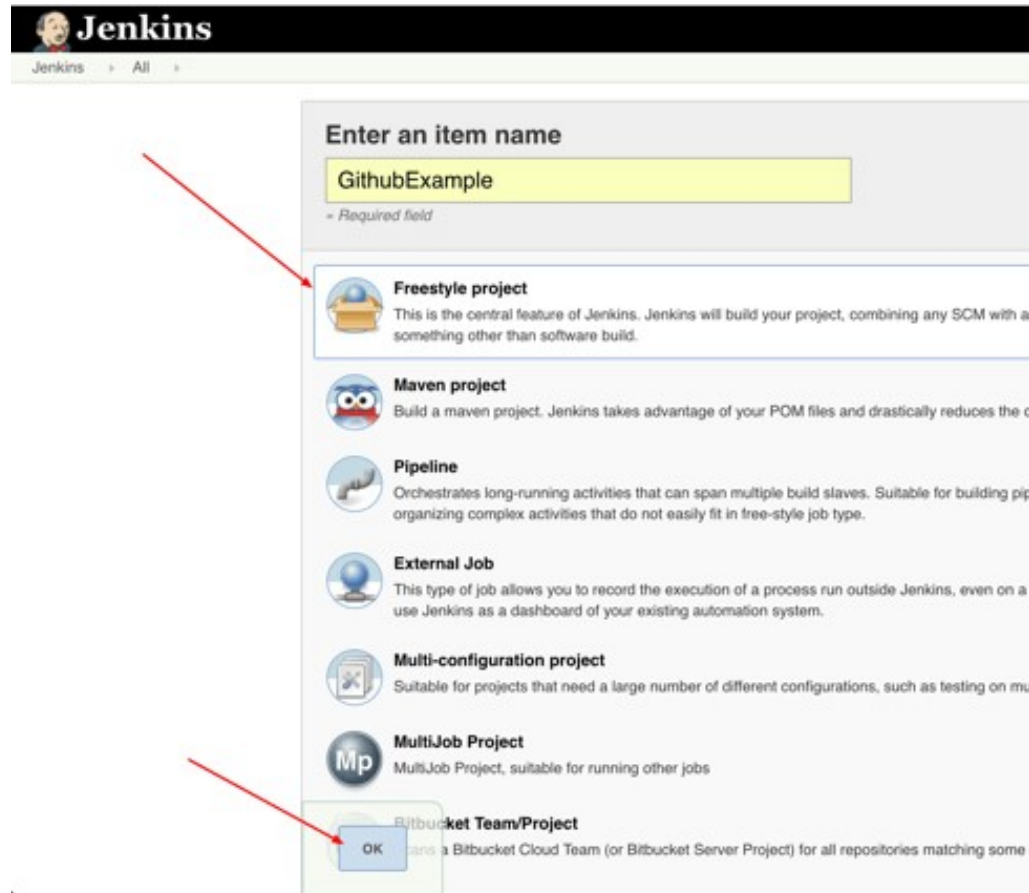
Configuring Jenkins

- Step 5: In Jenkins, click on 'New Item' to create a new project.



Configuring Jenkins

- Step 6: Give your project a name, then choose 'Freestyle project' and finally click on 'OK'.



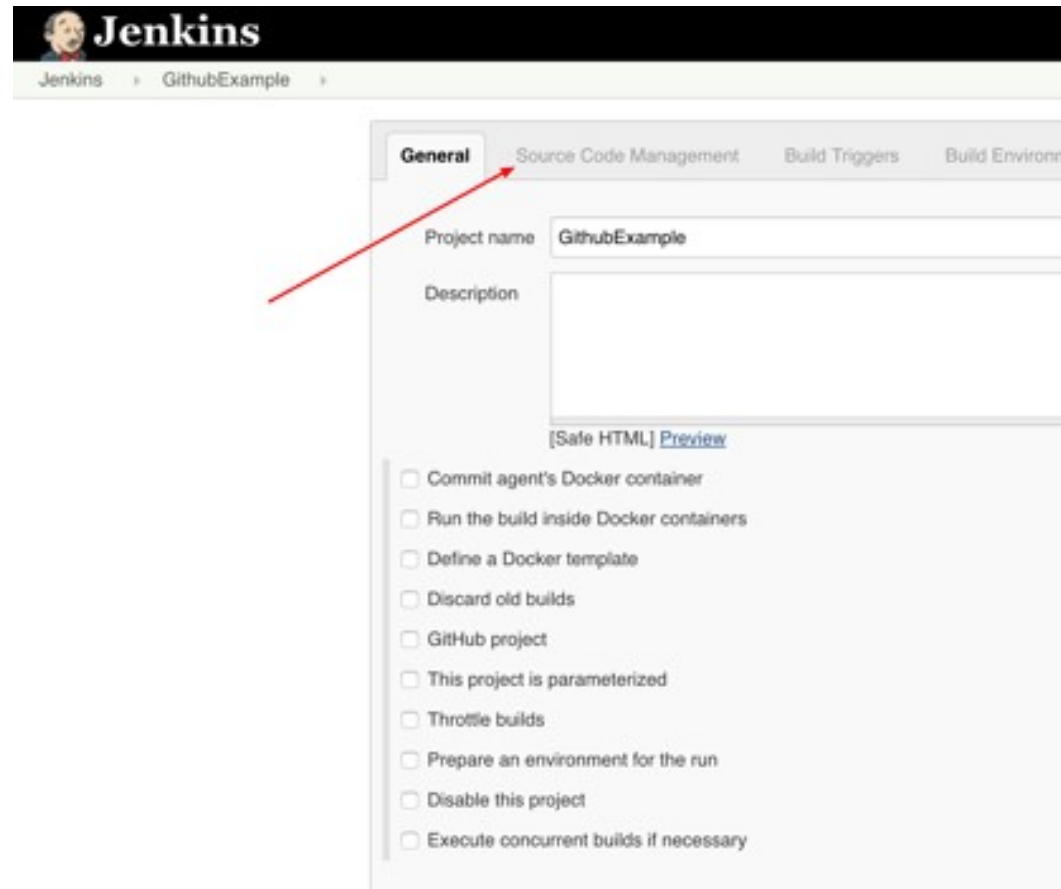
The screenshot shows the Jenkins 'New Item' configuration page. At the top, the Jenkins logo and 'Jenkins > All >' breadcrumb are visible. The main section is titled 'Enter an item name' and contains a text input field with the value 'GithubExample'. Below this, a list of project types is shown, each with an icon and a description:

- Freestyle project**: This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with anything else other than software build. (This option is selected, indicated by a red arrow.)
- Maven project**: Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the complexity of building.
- Pipeline**: Orchestrates long-running activities that can span multiple build slaves. Suitable for building pipelines organizing complex activities that do not easily fit in free-style job type.
- External Job**: This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. Use Jenkins as a dashboard of your existing automation system.
- Multi-configuration project**: Suitable for projects that need a large number of different configurations, such as testing on multiple platforms.
- MultiJob Project**: MultiJob Project, suitable for running other jobs.
- Bitbucket Team/Project**: This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. Use Jenkins as a dashboard of your existing automation system.

At the bottom of the list, there is an 'OK' button, which is highlighted with a red arrow.

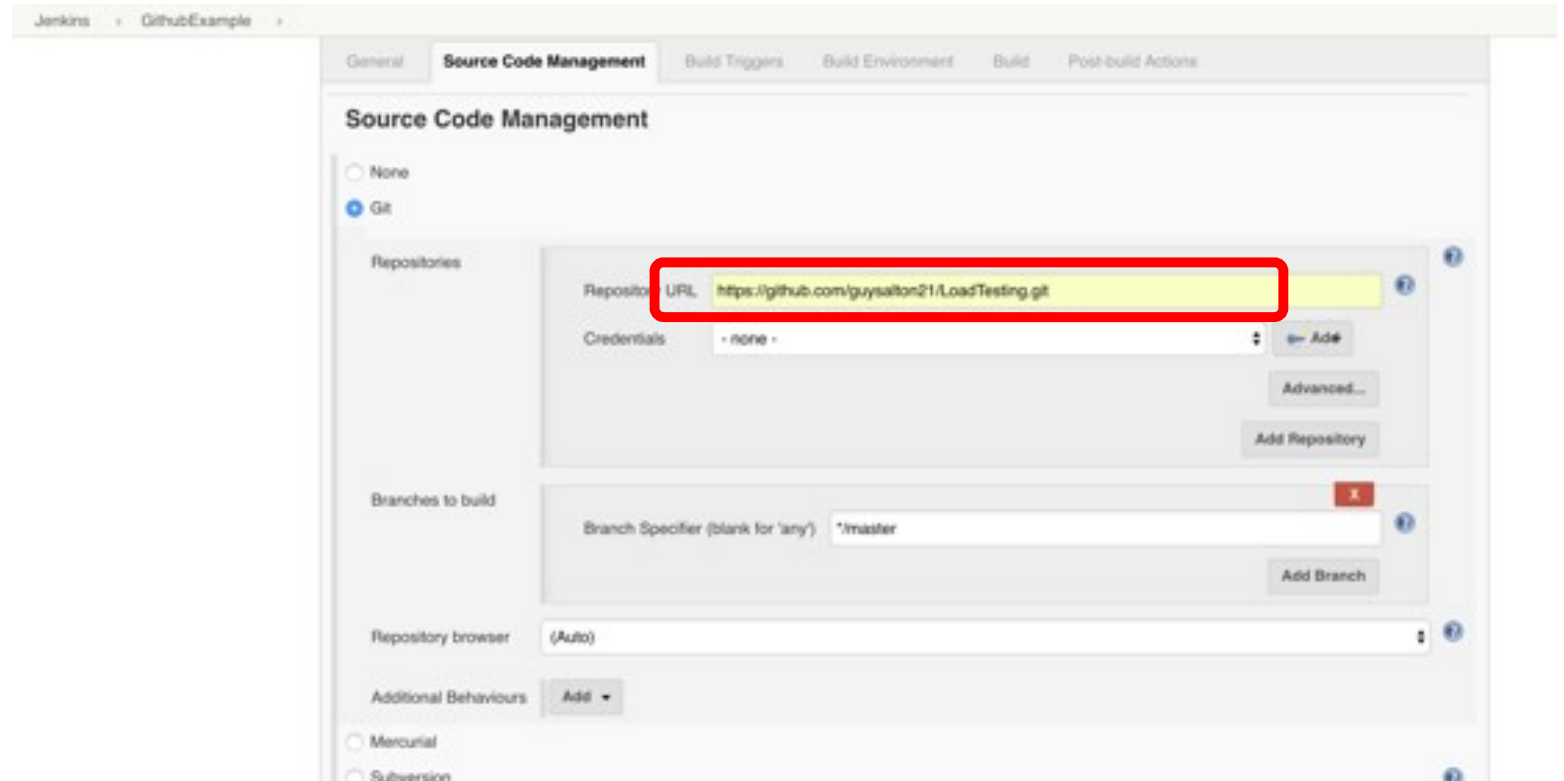
Configuring Jenkins

- Step 7: Click on the 'Source Code Management' tab.



Configuring Jenkins

- Step 8: Click on Git and paste your forked GitHub repository URL in the 'Repository URL' field.



The screenshot shows the Jenkins configuration interface for a project named 'GithubExample'. The 'Source Code Management' tab is selected, and the 'Git' option is chosen. The 'Repository URL' field is highlighted with a red rectangle and contains the text 'https://github.com/guysalton21/LoadTesting.git'. The 'Credentials' field is set to 'none'. The 'Branches to build' section shows a 'Branch Specifier (blank for 'any')' set to '*/*master'. The 'Repository browser' is set to '(Auto)'. The 'Additional Behaviours' section has an 'Add' button. The 'None' option is also visible at the top of the Source Code Management section.

Configuring Jenkins

- Step 9: Click on the 'Build Triggers' tab and then on the 'GitHub hook trigger for GITScm polling'. Or, choose the trigger of your choice.

The screenshot shows the Jenkins configuration interface for a project named 'GithubExample'. The 'Build Triggers' tab is active, displaying various trigger options. The 'GitHub hook trigger for GITScm polling' option is selected, indicated by a blue checkmark and a red rectangular highlight. Other visible options include 'Mercurial', 'Subversion', 'Trigger builds remotely (e.g., from scripts)', 'Build after other projects are built', 'Build periodically', 'Build when a change is pushed to BitBucket', 'GitHub Branches', 'GitHub Pull Requests', and 'Poll SCM'. The 'Build Environment' section is partially visible at the bottom.

Jenkins > GithubExample >

General Source Code Management **Build Triggers** Build Environment Build Post-build Actions

Additional Behaviours Add ▼

☐ Mercurial
☐ Subversion

Build Triggers

☐ Trigger builds remotely (e.g., from scripts)
☐ Build after other projects are built
☐ Build periodically
☐ Build when a change is pushed to BitBucket
☐ GitHub Branches
☐ GitHub Pull Requests
☒ GitHub hook trigger for GITScm polling
☐ Poll SCM

Build Environment

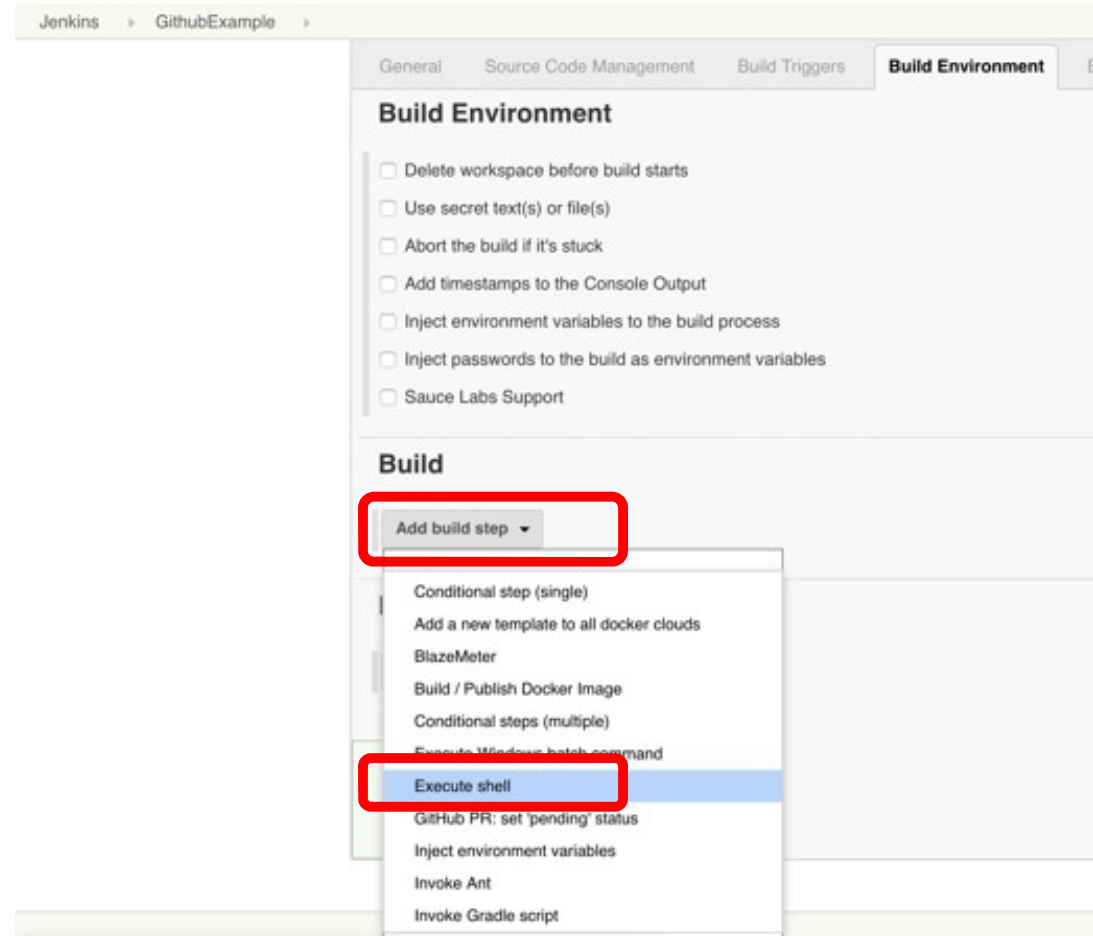
Configuring Jenkins

- Your GitHub repository is integrated with your Jenkins project.
- You can now use any of the files found in the GitHub repository and trigger the Jenkins job to run with every code commit.

Triggering the Jenkins Job to Run with Every Code Commit

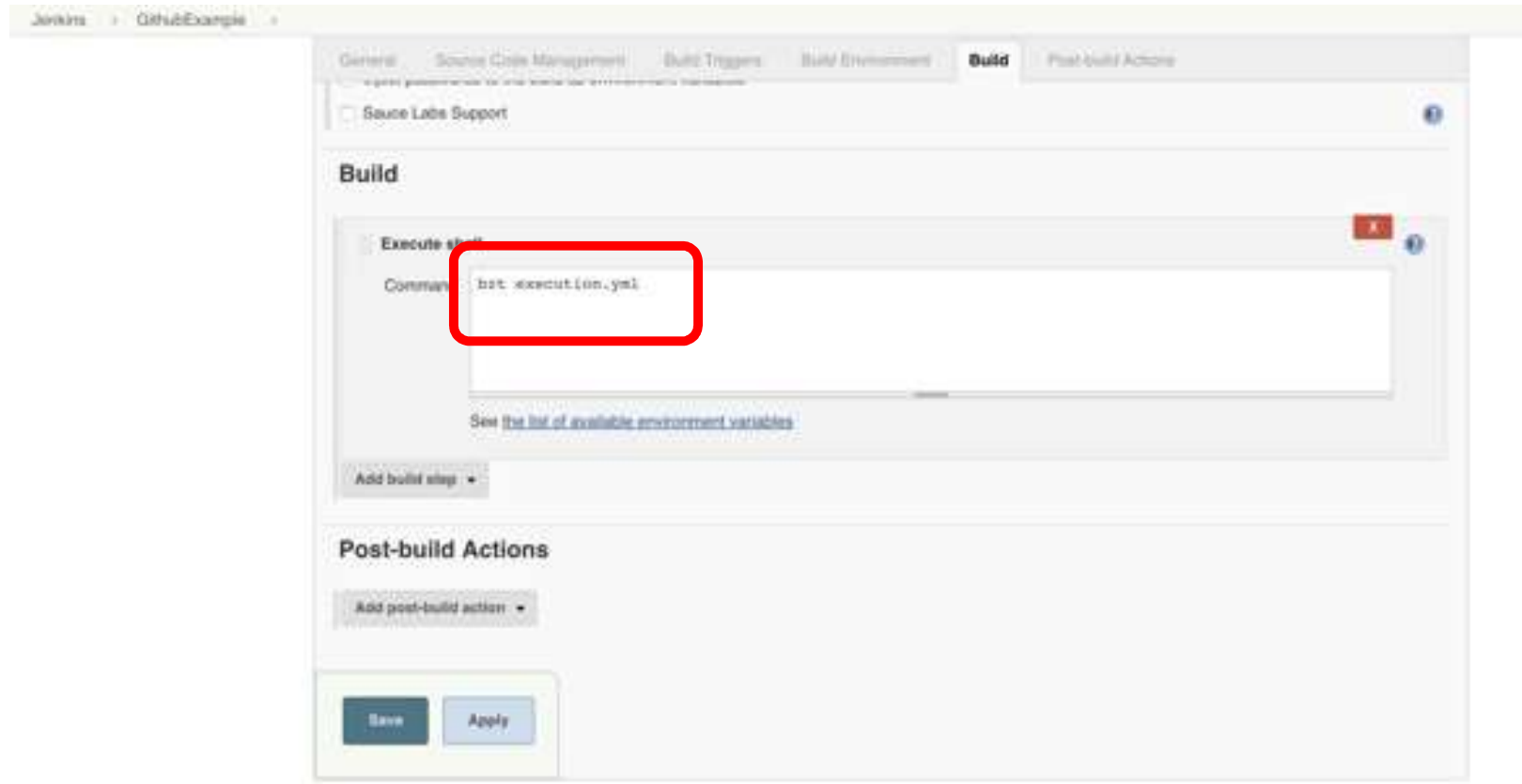
Triggering the Jenkins Job to Run

- Step 10: Click on the 'Build' tab,
- Then click on 'Add build step' and
- Choose 'Execute shell'.



Triggering the Jenkins Job to Run

- Step 11: To run sample commands - echo "Building Project"; echo "\$(pwd)"

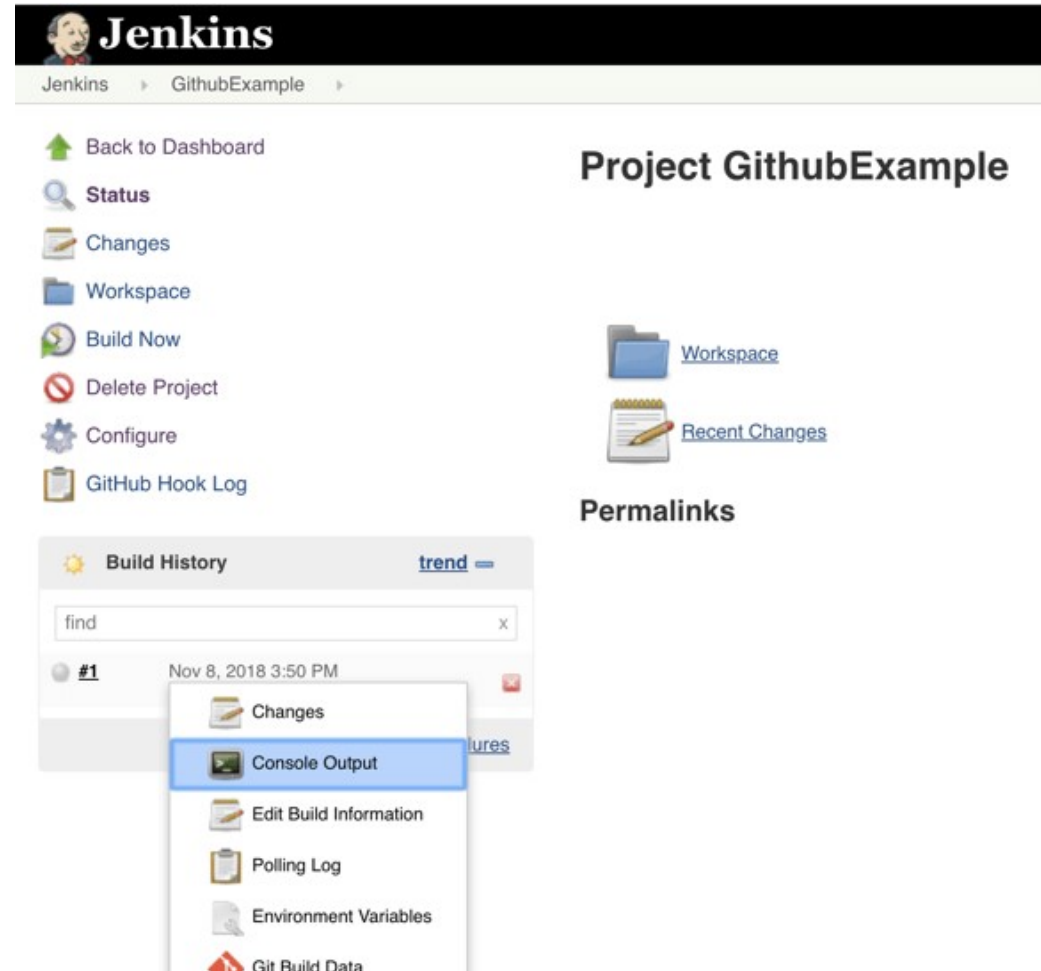


Triggering the Jenkins Job to Run

- Step 12: Go back to your GitHub repository, edit the code and commit the changes.
 - We will now see how Jenkins ran the script after the commit.

Triggering the Jenkins Job to Run

- Step 13: Go back to your Jenkins project and you'll see that a new job was triggered automatically from the commit we made at the previous step.
- Click on the little arrow next to the job and choose 'Console Output'.



Triggering the Jenkins Job to Run

- Step 14: You can see that Jenkins was able to pull the latest code and run it!
- Every time you publish your changes to Github, GitHub will trigger your new Jenkins job.

Code Packaging automation

Automation Maven test, Compile and Package

Continuous Delivery Pipeline Using Jenkins

- Fetching the code from GitHub
- Compiling the source code
- Unit testing and generating the JUnit test reports
- Packaging the application into a WAR file and deploying it on the Tomcat server



Source Code on Github

- <https://github.com/atingupta2005/java-servlet-hello>
- Clone
 - cd
 - git clone <https://github.com/atingupta2005/java-servlet-hello>
 - cd java-servlet-hello
- Compile app
 - mvn clean install
- Package App
 - mvn clean package

Step 1 — Compiling the Source Code

- Let's begin by first creating a Freestyle project in Jenkins
- Use Project Name – “Compile”
- When you scroll down you will find an option to add source code repository, select "git" and add the repository URL
- In that repository, there is a pom.xml file which we will use to build our project
- Consider the below screenshot:

Step 1 — Compiling the Source Code

General **Source Code Management** Build Triggers Build Environment Build Post-build Actions

Source Code Management

☐ None
☒ Git

Repositories

Repository URL

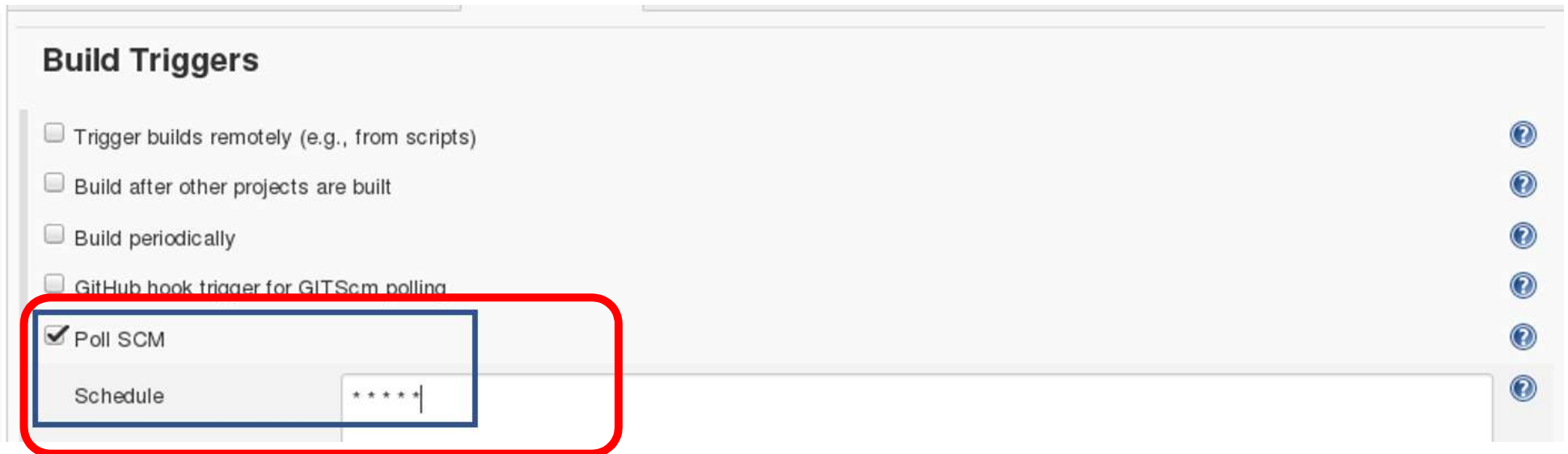
Credentials

Branches to build

Branch Specifier (blank for 'any')

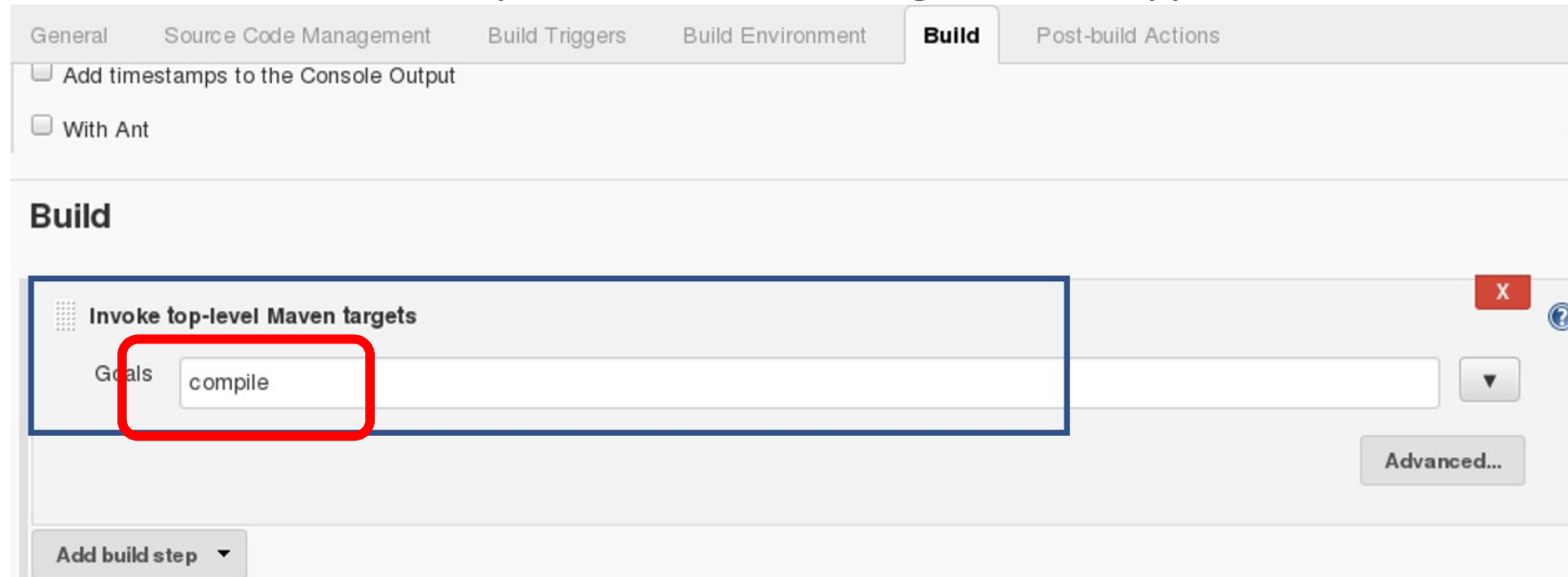
Step 1 — Compiling the Source Code

- Now we will add a Build Trigger
- Pick the poll SCM option
 - Basically, we will configure Jenkins to poll the GitHub repository after every 5 minutes for changes in the code
- Consider the below screenshot:



Step 1 — Compiling the Source Code

- In the build tab, click on invoke top level maven targets and type the below command:
 - compile



- This will pull source code from the GitHub repository and will also compile it.
- Click on Save and run the project.
- Now, click on the console output to see the result.

Step 2 — Test the Source Code

- Now we will create one more Freestyle Project for unit testing.
- Project Name - **Test**
- Add the same repository URL in the source code management tab, like we did in the previous job.
- Now, in the "Build Trigger" tab click on the
 - "build after other projects are built".
- In the Build tab, click on invoke top level maven targets and use the below command:
 - `test`

Build Triggers

☐ Trigger builds remotely (e.g., from scripts)

☒ Build after other projects are built

Projects to watch **Compile,**

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

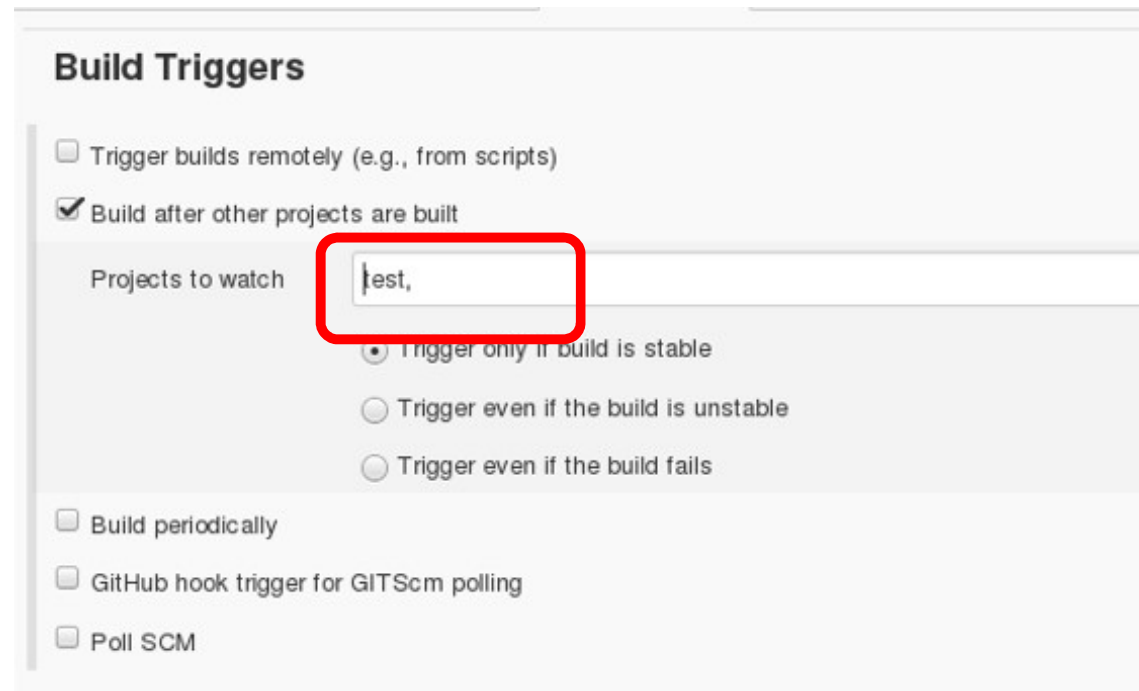
☐ Build periodically

☐ GitHub hook trigger for GITScm polling

☐ Poll SCM

Step 3 — Creating a JAR File and Deploying

- Create one more freestyle project and add the source code repository URL.
- Then in the build trigger tab, select build when other projects are built, consider the below screenshot:
- Project Name – “Create Jar”



Build Triggers

☐ Trigger builds remotely (e.g., from scripts)

☒ 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

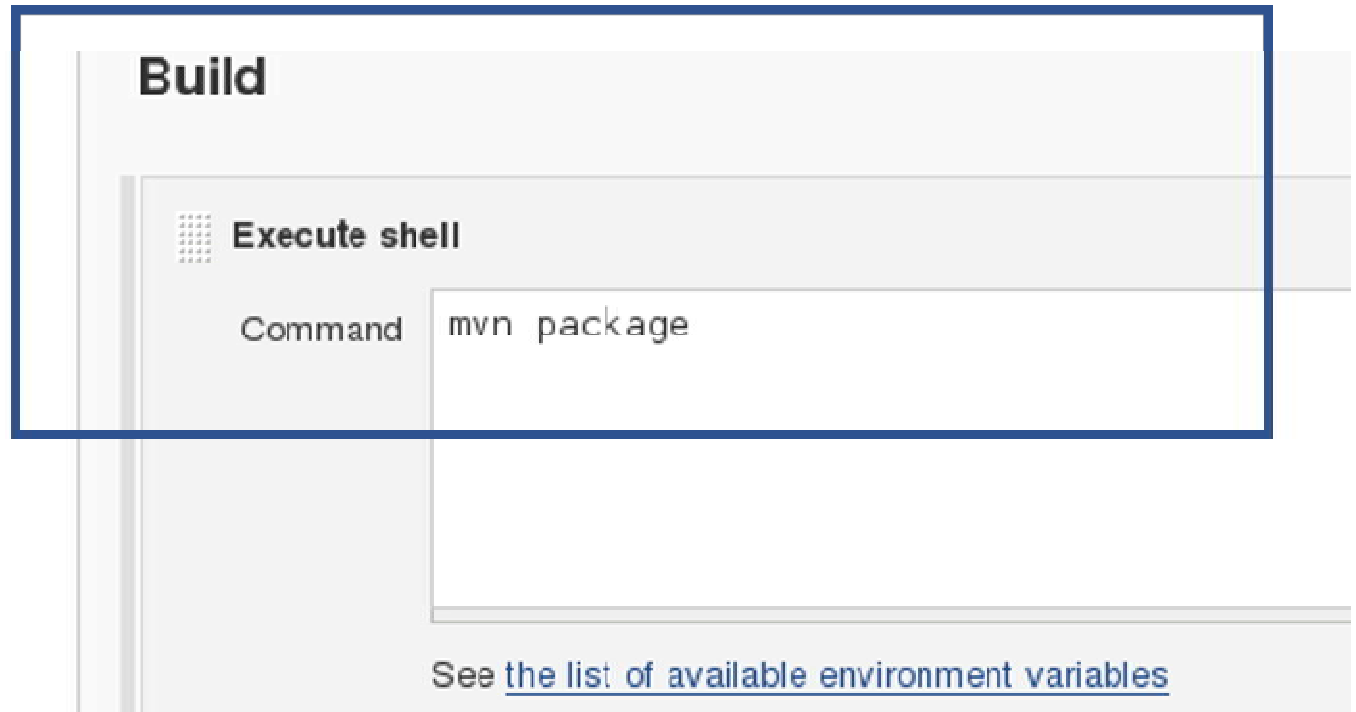
☐ Build periodically

☐ GitHub hook trigger for GITScm polling

☐ Poll SCM

Step 3 — Creating a JAR File and Deploying

- In the build tab, select shell script. Type the below command to package the application:
 - `mvn package`



Add slave nodes to Jenkins

Add slave nodes to Jenkins

- There are two ways of authentication for setting up the Jenkins slaves.
 - Using username and password
 - Using ssh keys.

Jenkins Slave Prerequisites

- Java should be installed on your slave machine.
- We should have a valid user id on slave machine using which we can perform the required tasks

Create a Ubuntu Machine

- `sudo docker run -dit --name my_jenkins_slave --privileged=true -p 8889:8080 -p 222:22 atingupta2005/tomcat_jenkins_ubuntu`
- `sudo docker exec -it my_jenkins_slave bash`
 - `# service ssh start`
 - `exit`
- `ssh root@localhost -p 222 # password 123456`

Setting Up Oracle and Maven on Slave

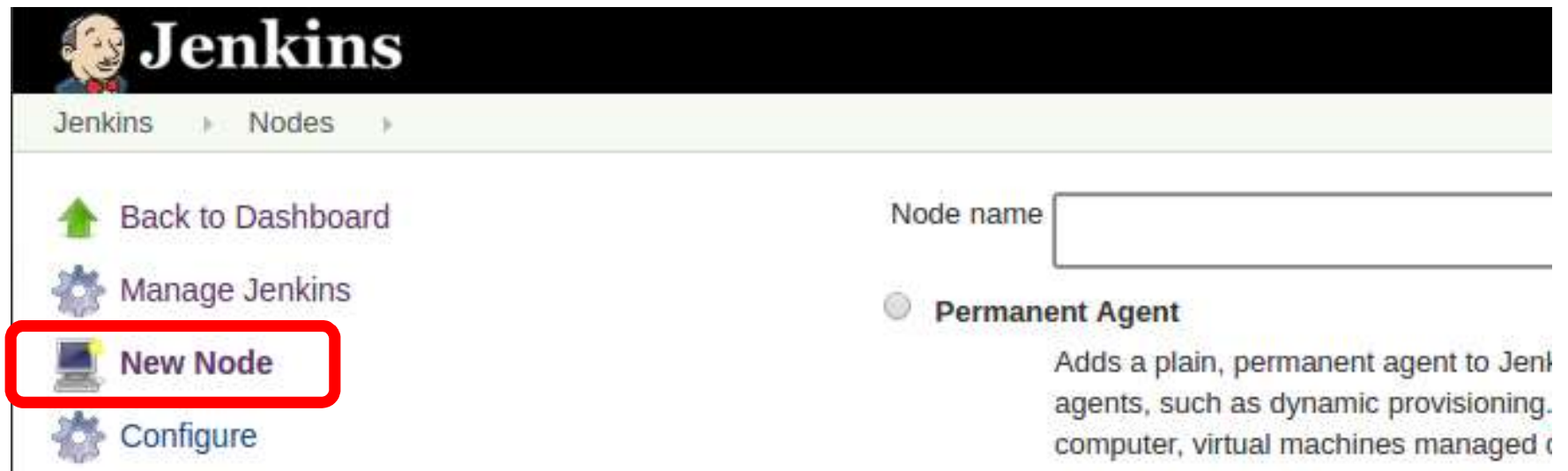
- Login to Docker container at port 222
- # If required switch to root:
 - su
- sudo apt -y update
- # Check if Java and maven are already installed?
- java -version
- mvn -v
- #Only if needed
- apt install -y maven
- apt install -y openjdk-8-jdk

Create a Jenkins User

- It is recommended to execute all Jenkins jobs as jenkins user on the slave nodes.
- On Docker Container Slave, create a jenkins user and a password using the following command.
 - `adduser jenkins --shell /bin/bash`
- Now, login as jenkins user.
 - `su jenkins`
- Create a “jenkins_slave” directory under /home/jenkins.
 - `mkdir /home/jenkins/jenkins_slave`

Setting up Jenkins slaves using username/password

- Head over to Jenkins dashboard -> Manage Jenkins -> Manage Nodes
- Select new node option.



Setting up Jenkins slaves using username/password

- Give it a name, select the “permanent agent” option and click ok.



The screenshot shows the 'Add Node' dialog box in Jenkins. It has a title bar at the top. Below the title bar, there is a text input field labeled 'Node name' containing the text 'ubuntu-slave1'. Below this, there is a radio button selected next to the label 'Permanent Agent'. To the right of the radio button, there is a descriptive text: 'Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't manage agents, such as dynamic provisioning. Select this type if no other agent types apply — for example, computer, virtual machines managed outside Jenkins, etc.' At the bottom left of the dialog, there is a grey button labeled 'OK'.

Node name

☒ **Permanent Agent**

Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't manage agents, such as dynamic provisioning. Select this type if no other agent types apply — for example, computer, virtual machines managed outside Jenkins, etc.

OK

Setting up Jenkins slaves using username/password

- Enter the details as shown in the image below and save it
- For credential box, click the add button and enter the slaves jenkins username and password
- Click on Save

# of executors	<input type="text" value="2"/>	Host	<input type="text" value="localhost"/>
Remote root directory	<input type="text" value="/home/jenkins/jenkins_slave"/>	Credentials	<input type="text" value="jenkins/*****"/> <input type="button" value="Add"/>
Labels	<input type="text" value="ubuntu_slave1"/>	Host Key Verification Strategy	<input type="text" value="Non verifying Verification Strategy"/>
Usage	<input type="text" value="Only build jobs with label expressions matching this node"/>	Port	<input type="text" value="222"/>
Launch method	<input type="text" value="Launch agents via SSH"/>		

Preparing Slave Nodes to Perform Build

The screenshot shows the Jenkins configuration interface for a project named 'slave-test'. The 'General' tab is selected, and the 'Restrict where this project can be run' checkbox is checked and highlighted with a red box. The 'Label Expression' is set to 'ubuntu-slave1', and a message indicates that the label is serviced by 1 node. Other options like 'Discard old builds', 'GitHub project', 'This project is parameterised', 'Throttle builds', 'Disable this project', and 'Execute concurrent builds if necessary' are unchecked. The 'Advanced...' button is visible at the bottom right.

General Source Code Management Build Triggers Build Environment Build Post-build Actions

Project name

Description

[Plain text] [Preview](#)

- ☐ Discard old builds
- ☐ GitHub project
- ☐ This project is parameterised
- ☐ Throttle builds
- ☐ Disable this project
- ☐ Execute concurrent builds if necessary
- ☒ Restrict where this project can be run

Label Expression

[Label](#) is serviced by 1 node

[Advanced...](#)

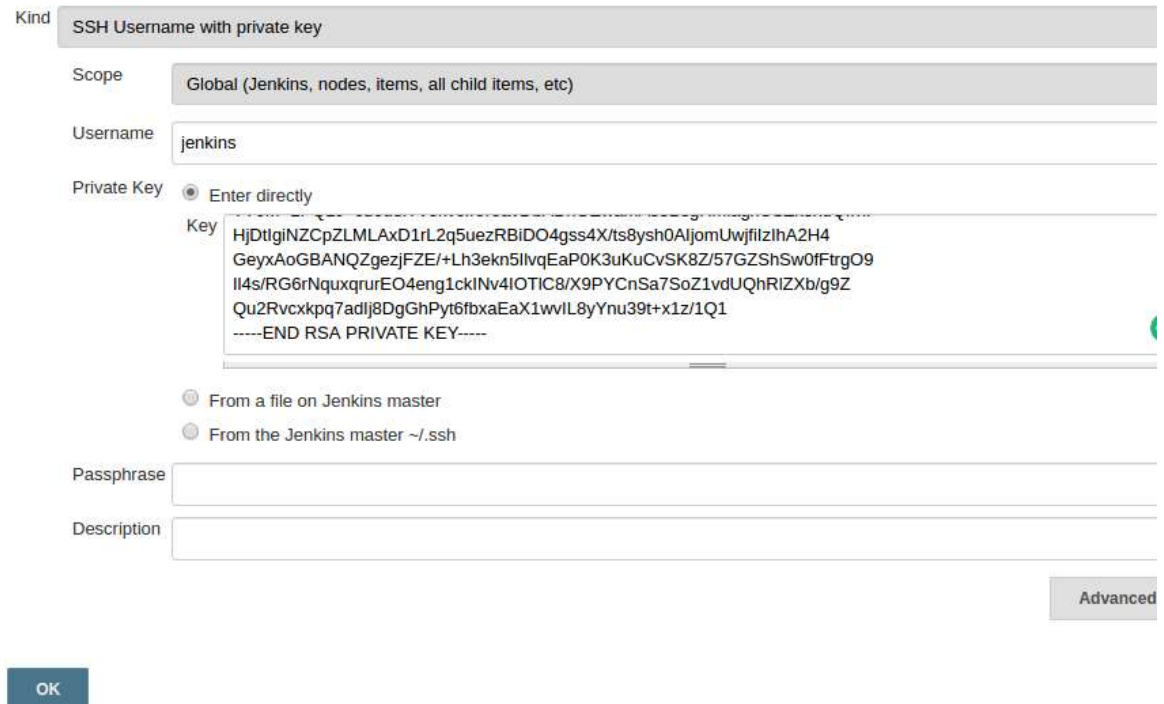
[Save](#) [Apply](#) [Cancel](#)

Setting up Jenkins slaves using ssh keys

- Login to the slave server as a jenkins user.
 - `ssh jenkins@localhost -p 222`
- Create a `.ssh` directory and `cd` into the directory.
 - `mkdir ~/.ssh && cd ~/.ssh`
- Create an ssh key pair using the following command. Press enter for all the defaults when prompted.
 - `ssh-keygen -t rsa -C "The access key for Jenkins slaves "`
- Add the public to `authorized_keys` file using the following command.
 - `cat id_rsa.pub > ~/.ssh/authorized_keys`
- Now, copy the contents of the private key to the clipboard.
 - `cat id_rsa`

Add the private key to Jenkins credential list

- Go to jenkins - Dashboard -> manage jenkins -> manage credentials -> Global credentials -> add credentials
 - Sample URL: http://13.90.41.2:8080/credentials/store/system/domain/_/
- Select and enter all the credentials as shown



The screenshot shows the 'Add Credentials' form in Jenkins. The 'Kind' is set to 'SSH Username with private key'. The 'Scope' is 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' is 'jenkins'. Under 'Private Key', the 'Enter directly' radio button is selected. The 'Key' field contains a long base64-encoded string followed by '-----END RSA PRIVATE KEY-----'. Below this, there are two unselected radio buttons: 'From a file on Jenkins master' and 'From the Jenkins master ~/.ssh'. The 'Passphrase' and 'Description' fields are empty. At the bottom right is an 'Advanced' button, and at the bottom left is an 'OK' button.

Kind: SSH Username with private key

Scope: Global (Jenkins, nodes, items, all child items, etc)

Username: jenkins

Private Key: ☒ Enter directly

Key: HjdTigiNZCpZLMLAxD1rL2q5uezRbiDO4gss4X/ts8ysh0AljomUwjfilzlhA2H4
GeyxAoGBANQZgezjFZE/+Lh3ekn5llvqEaP0K3uKuCvSK8Z/57GZShSw0fFtrgO9
ll4s/RG6rNquxqrurEO4eng1ckINv4IOTIC8/X9PYCnSa7SoZ1vdUQHrIZXb/g9Z
Qu2Rvcxkpq7adlj8DgGhPyt6fbaEaX1wvIL8yYnu39t+x1z/1Q1
-----END RSA PRIVATE KEY-----

☐ From a file on Jenkins master

☐ From the Jenkins master ~/.ssh

Passphrase:

Description:

Advanced

OK

Setup slaves from Jenkins master

- Follow the first 3 steps we did for slave configuration using username and password.
- Follow all the configuration in the 4th step. But this time, for the launch method, select the credential you created with the ssh key.

Test the slaves

- To test the slave, create a sample project and select the option as shown
- You need to select the node using the label option
- If you start to type the letter the node list will show up

The screenshot shows the Jenkins 'General' configuration page for a project named 'slave-test'. The 'Project name' field is filled with 'slave-test'. Below it is a large 'Description' text area. A list of checkboxes is visible, including 'Discard old builds', 'GitHub project', 'This project is parameterised', 'Throttle builds', 'Disable this project', 'Execute concurrent builds if necessary', and 'Restrict where this project can be run'. The 'Restrict where this project can be run' checkbox is checked and highlighted with a red rectangle. Below this checkbox is a 'Label Expression' field containing 'ubuntu-slave1'. A message below the field states 'Label is serviced by 1 node'. At the bottom left are 'Save' and 'Apply' buttons. At the bottom right is an 'Advanced...' button. The top navigation bar includes tabs for 'General', 'Source Code Management', 'Build Triggers', 'Build Environment', 'Build', and 'Post-build Actions'.

Important URLs

- GitHub projects:
 - <https://github.com/atingupta2005/hello-world-maven>
 - Console Based – To build jar file
 - <https://github.com/atingupta2005/java-servlet-hello>
 - Web Based to build war file

Thanks