

[illegible]

Name	DevOps Pipeline: Java WebApp
URL	attackdefense.com/challengedetails?cid=2064
Type	Pipeline Basics: Web Applications

Important Note: This document illustrates all the important steps required to complete this lab. This is by no means a comprehensive step-by-step solution for this exercise. This is only provided as a reference to various commands needed to complete this exercise and for your further research on this topic. Also, note that the IP addresses and domain names might be different in your lab.

Challenge Description

DevOps practices are to combine software development (Dev) and IT operations (Ops) in order to improve the delivery process. DevOps pipelines are chained tasks and components that run in a sequence to cover different phases of software compilation, packaging, automated testing, and test deployment.

In this lab, we have a simple DevOps pipeline for a sample Java-based web application. The pipeline consists of the following components (and tasks):

- Kali machine (For pulling, modifying, and pushing the code)
- GitLab server (For hosting code)
- Jenkins server (For integrating all parts: building/testing project with maven, deploying with Ansible, and dynamic testing with Selenium)
- Test server (For test deployment)

Objective: Run the pipeline and observe/understand the DevOps process!

Instructions:

- The GitLab server is reachable with the name 'gitlab'
- Gitlab credentials:

Username	Password
root	welcome123

- The Jenkins server is reachable with the name 'jenkins'
- Jenkins credentials:

Username	Password
admin	welcome123

- The test deployment server is reachable by the name "test-server"
- Test server SSH credentials:

Username	Password
tomcat	password1

Lab Setup

On starting the lab, the following interface will be accessible to the user.



Kali Jenkins GitLab Test Server

Kali

Jenkins

GitLab

Test Server

On choosing (clicking the text in the center) top left panel, **KALI CLI** will open in a new tab

```
root@kali-cli:~#
```

Similarly on selecting the top right panel, a web UI of **Jenkins** will open in a new tab.



Welcome to Jenkins!

Sign in

☐ Keep me signed in

On selecting the bottom left panel, a web UI of **Gitlab** will open in a new tab.



GitLab Community Edition

Open source software to collaborate on code

Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

Sign in	Register
Username or email	
<input type="text"/>	
Password	
<input type="password"/>	
<input type="checkbox"/> Remember me	Forgot your password?
<p>Sign in</p>	

And on selecting the bottom right panel, a web UI of **Test Server** will open in a new tab.

A banner for Pentester Academy with a teal background and faint, stylized graphics of a skull and a gear. The text "PENTESTER ACADEMY" is prominently displayed in the center.

PENTESTER ACADEMY

WebApp will appear once deployed!

This page refreshes automatically.

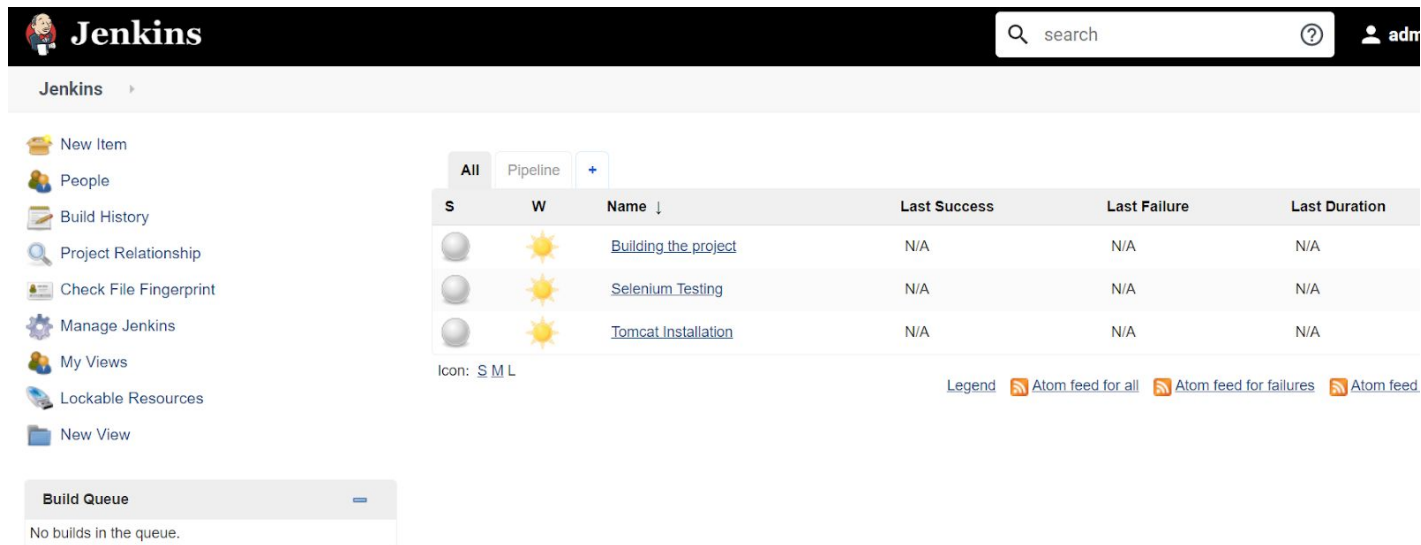
The page will reload until the test-server has started running the web service at port 8080

Solution

Step 1: Login into the Jenkins, The credentials are provided in the challenge description.

Credentials:

- **Username:** admin
- **Password:** welcome123



The screenshot shows the Jenkins dashboard. The top navigation bar includes the Jenkins logo, a search bar, and a user profile icon labeled 'adm'. The left sidebar contains a list of links: New Item, People, Build History, Project Relationship, Check File Fingerprint, Manage Jenkins, My Views, Lockable Resources, and New View. The main content area displays a table of jobs under the 'All' tab. The table has columns for status (S), icon (W), name, last success, last failure, and last duration. Three jobs are listed: 'Building the project', 'Selenium Testing', and 'Tomcat Installation', all with a status of 'Success' (represented by a sun icon) and 'N/A' for last success and last failure. Below the table, there is a legend for Atom feeds and a 'Build Queue' section showing 'No builds in the queue.'

S	W	Name ↓	Last Success	Last Failure	Last Duration
Success	Sun	Building the project	N/A	N/A	N/A
Success	Sun	Selenium Testing	N/A	N/A	N/A
Success	Sun	Tomcat Installation	N/A	N/A	N/A

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

Legend [Atom feed for all](#) [Atom feed for failures](#) [Atom feed](#)

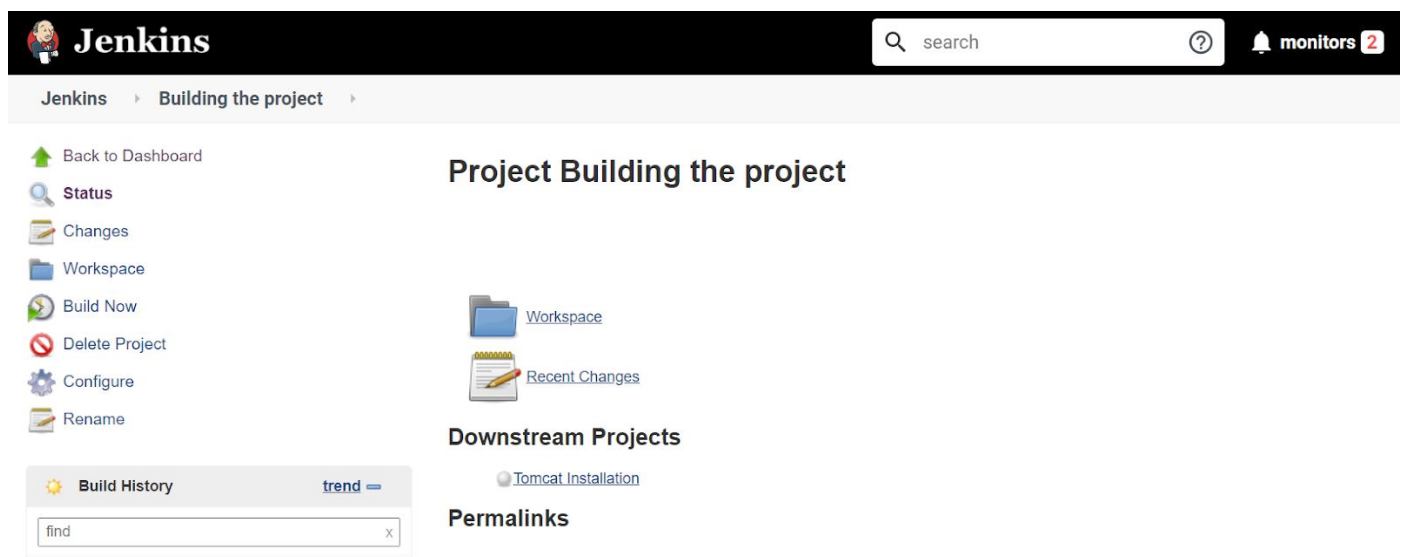
Build Queue

No builds in the queue.

There are 3 jobs present in the Jenkins Interface, We will take one job at a time to study.

Job 1: Building the Project

Step 1: Click on the “Building the project” job.



The screenshot shows the Jenkins job view for 'Building the project'. The top navigation bar includes the Jenkins logo, a search bar, and a 'monitors 2' notification. The left sidebar contains a list of links: Back to Dashboard, Status, Changes, Workspace, Build Now, Delete Project, Configure, and Rename. The main content area displays the job title 'Project Building the project' and a list of downstream projects: 'Tomcat Installation'. There is also a 'Permalinks' section. The 'Build History' section shows a search bar with the text 'find' and a 'trend' button.

Project Building the project

[Workspace](#)

[Recent Changes](#)

Downstream Projects

[Tomcat Installation](#)

Permalinks

Build History [trend](#)

find X

This page is for “Building the Project” job, from here the user can configure, Build, Delete the Job.

The workspace is a personal directory of this job where all the files which get downloaded during the build phase will reside.

Step 2: Click on the “Configure” option to check the configuration of the Job.

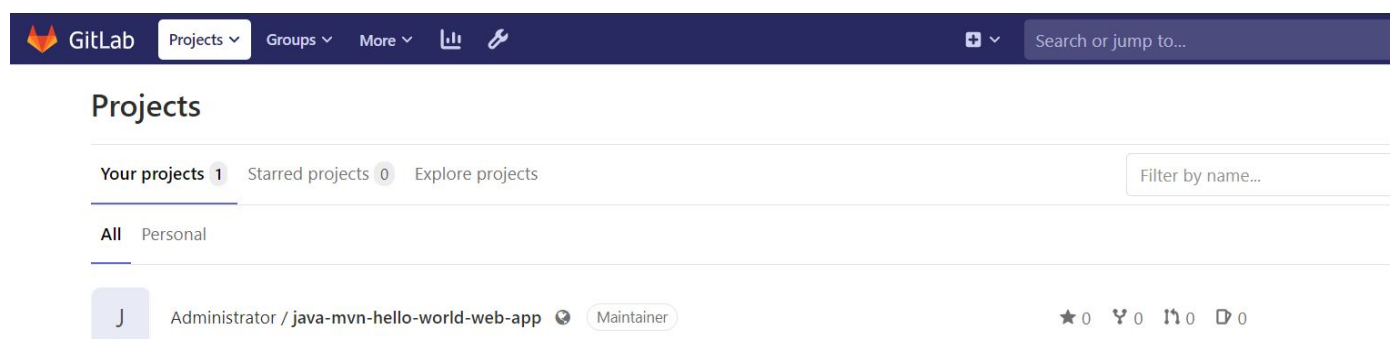
The screenshot displays the Jenkins web interface for configuring a job named "Building the project". The top navigation bar includes the Jenkins logo, a search bar, and user information (admin). The breadcrumb trail shows "Jenkins > Building the project".

The configuration page is divided into two main sections:

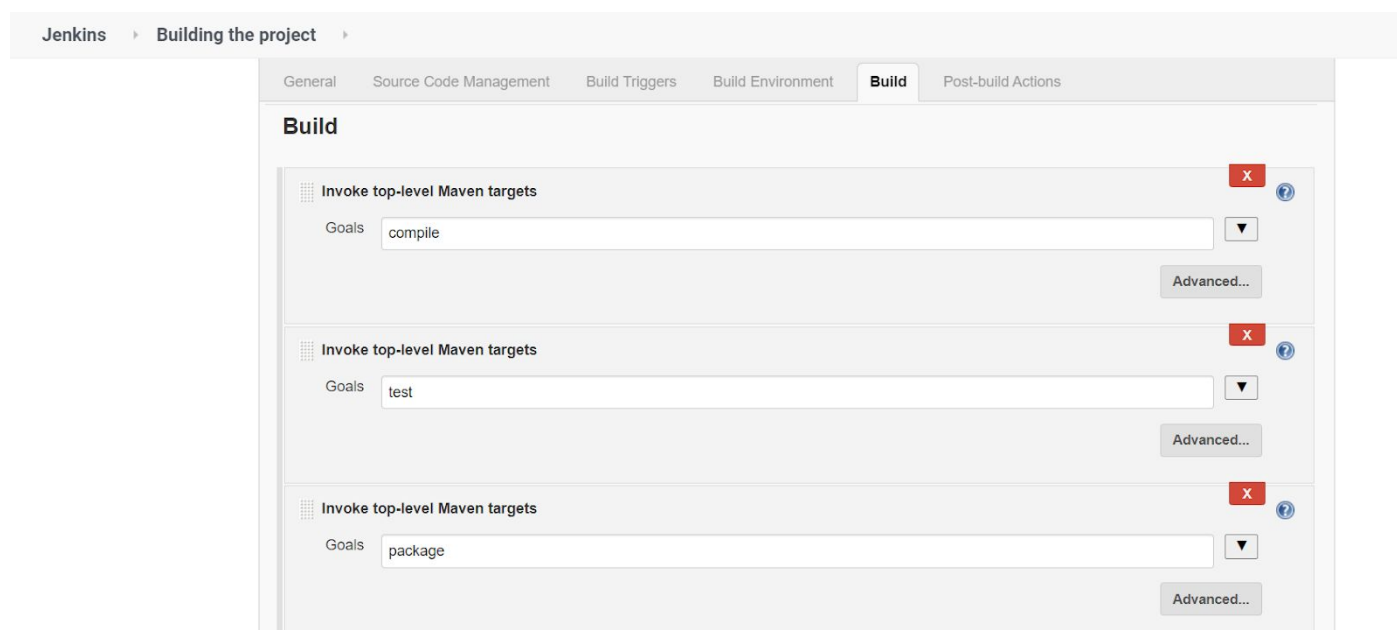
- General:** This section contains a "Description" text area, a "[Plain text] Preview" link, and several checkboxes for build options: "Discard old builds", "GitHub project", "This build requires lockable resources", "This project is parameterized", "Throttle builds", "Disable this project", and "Execute concurrent builds if necessary". An "Advanced..." button is located at the bottom right of this section.
- Source Code Management:** This section shows the selected source code management system as "Git". It includes fields for "Repository URL" (http://gitlab/root/java-mvn-hello-world-web-app.git), "Credentials" (set to "- none -"), and "Branches to build" (with a branch specifier of "*/master"). There are also buttons for "Add Repository" and "Add Branch".

The Source Code Management (SCM) is the repository where the source code of the application is stored. This option will tell Jenkins to pull the code from the provided repository,

Which in this case is “http://gitlab/root/java-mvn-hello-world-web-app.git”, This is the repository which is stored in the Gitlab.



Step 3: Scroll down to check more options for the Job configuration



This phase is the “Build” phase where the application gets compiled or packages. In this case, we are using maven for compilation hence the maven plugin is installed and Goals are passed.

Goals are the tasks which are used to build and manage an application. Here, we have assigned 3 tasks such as compile, test, package.

Tasks:

- **Compile:** To compile the source code of the application
- **Test:** Testing the compiled source code using unit testing frameworks.
- **Package:** Package the compiled source code into a distributable format like JAR, WAR

Step 4: Checking the Shell commands and post-build actions of the job.

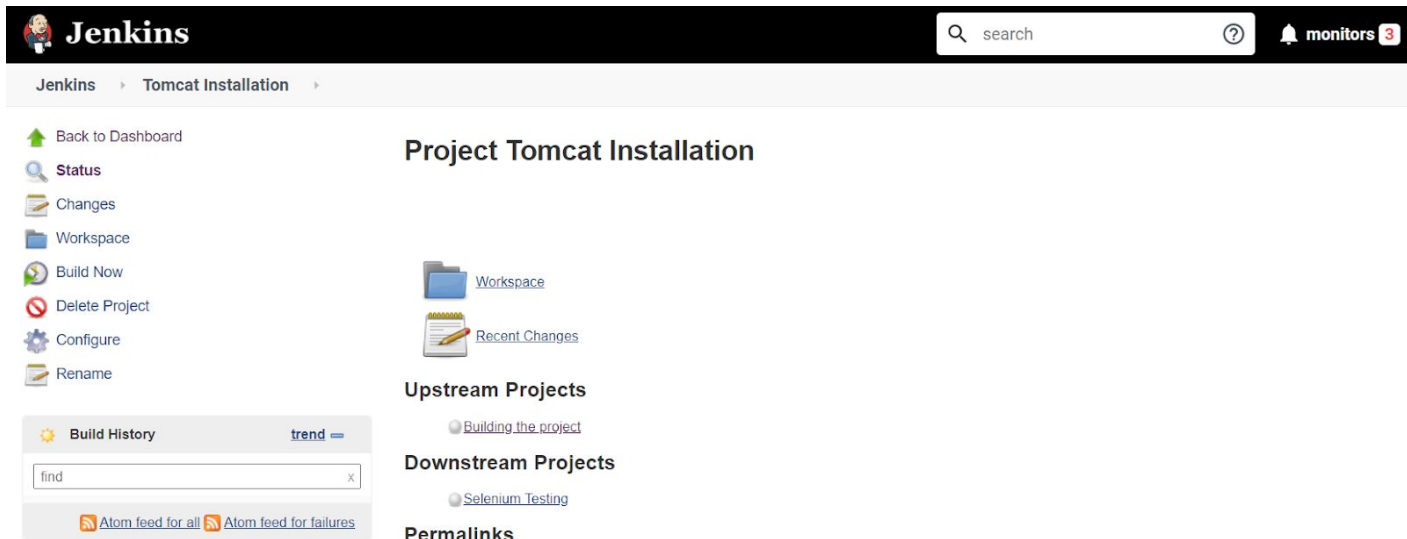
The screenshot shows the Jenkins configuration interface for a job named 'Building the project'. The 'Build' tab is selected, displaying the 'Execute shell' section with the command `mv target/*.war /tmp/`. Below this, the 'Post-build Actions' section is visible, showing 'Build other projects' with 'Tomcat Installation' selected. The page has tabs for General, Source Code Management, Build Triggers, Build Environment, Build, and Post-build Actions.

The “Execute Shell” section let the Jenkins execute certain sets of commands while building the project. In this case, The war file which is packaged by the maven will be moved to the temp (/tmp) directory.

Post Build Actions: These are the steps which get executed after performing the Build phase. Here, the action is to execute another Job (Tomcat Installation) after finishing this one.

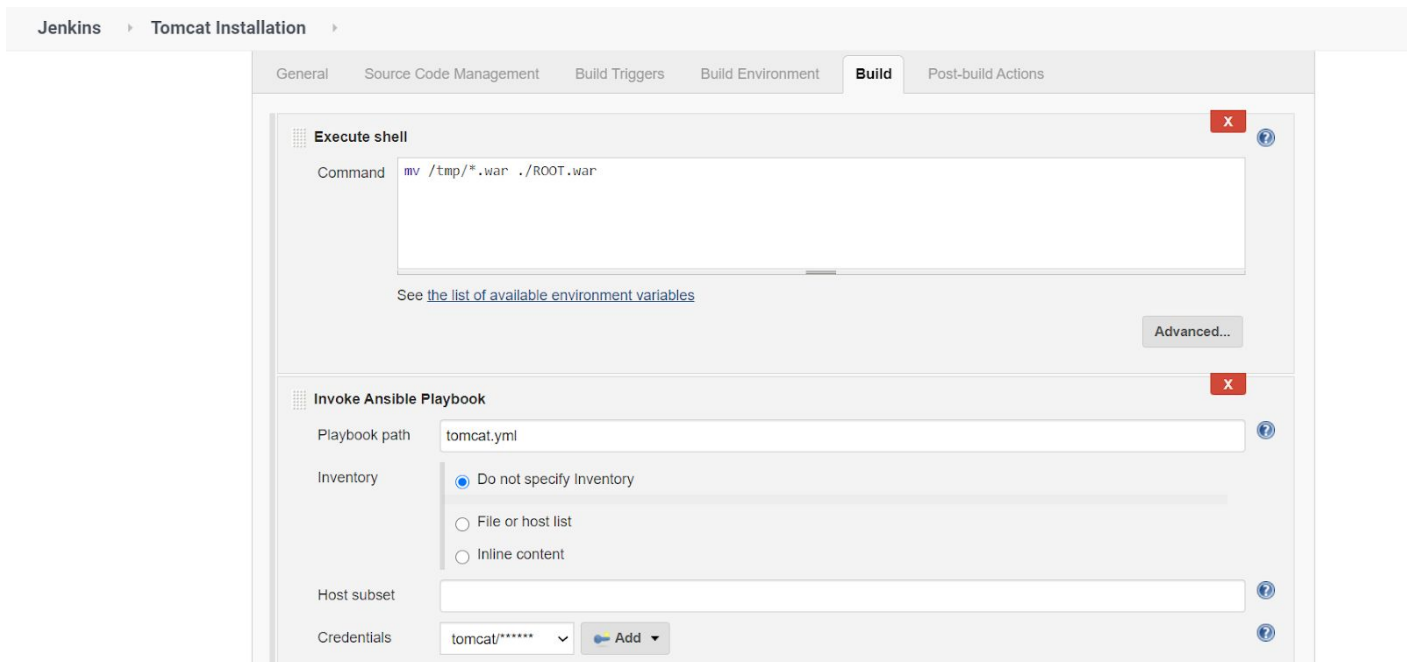
Job 2: Tomcat Installation

Step 1: Click on the “Tomcat Installation” job.



The screenshot shows the Jenkins web interface. At the top, there's a black header with the Jenkins logo, a search bar, and a 'monitors 3' notification. Below the header, a breadcrumb trail shows 'Jenkins > Tomcat Installation'. On the left sidebar, there are links for 'Back to Dashboard', 'Status', 'Changes', 'Workspace', 'Build Now', 'Delete Project', 'Configure', and 'Rename'. The main content area is titled 'Project Tomcat Installation'. It features a 'Workspace' folder icon and a 'Recent Changes' icon. Below these, there are sections for 'Upstream Projects' (with a link to 'Building the project') and 'Downstream Projects' (with a link to 'Selenium Testing'). At the bottom, there are 'Permalinks' and a 'Build History' section with a search bar and links to 'Atom feed for all' and 'Atom feed for failures'.

Step 2: Click on the “Configure” option to check the configuration of the Job.

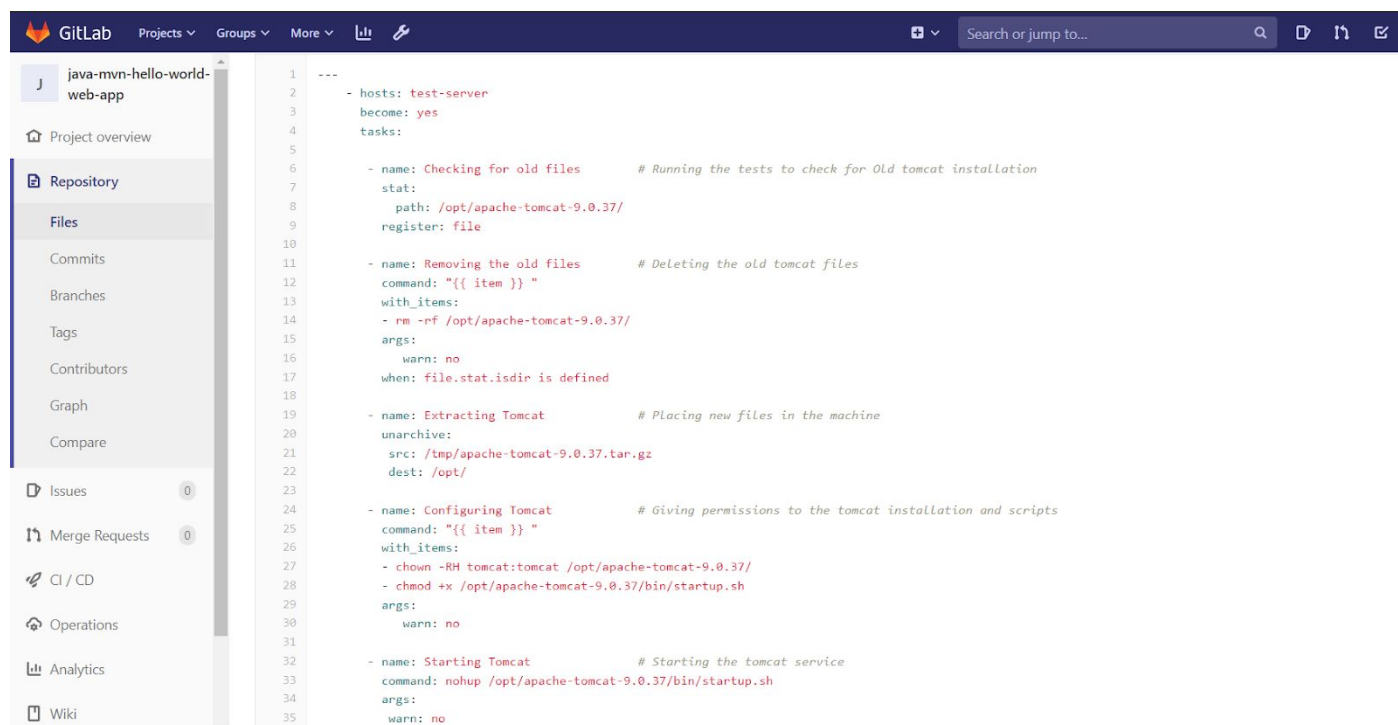


The screenshot shows the 'Configure' page for the 'Tomcat Installation' job. The breadcrumb trail is 'Jenkins > Tomcat Installation'. The page has tabs for 'General', 'Source Code Management', 'Build Triggers', 'Build Environment', 'Build', and 'Post-build Actions'. The 'Build' tab is selected. Under the 'Execute shell' section, the command is 'mv /tmp/*.war ./ROOT.war'. Below this, there's a link to 'See the list of available environment variables' and an 'Advanced...' button. Under the 'Invoke Ansible Playbook' section, the 'Playbook path' is 'tomcat.yml'. The 'Inventory' section has three options: 'Do not specify inventory' (selected), 'File or host list', and 'Inline content'. The 'Host subset' field is empty. The 'Credentials' section shows 'tomcat/*****' with an 'Add' button.

The Shell command will rename any file ending with “.war” extension to ROOT.war.

The Ansible Plugin will load the “tomcat.yml” as the configuration to install tomcat on test-server

Step 3: Check the tomcat.yml file on Github repository.

The screenshot shows the GitLab web interface for a project named 'java-mvn-hello-world-web-app'. The left sidebar contains navigation links for Project overview, Repository, Files, Commits, Branches, Tags, Contributors, Graph, Compare, Issues (0), Merge Requests (0), CI / CD, Operations, Analytics, and Wiki. The main content area displays the 'tomcat.yml' Ansible playbook. The playbook is written in YAML and includes several tasks: checking for old files, removing old files, extracting Tomcat, configuring Tomcat, and starting Tomcat. The tasks are commented with their purposes in English. The playbook is executed on a host named 'test-server' and becomes the 'yes' user.

```
1 ---
2 - hosts: test-server
3   become: yes
4   tasks:
5
6     - name: Checking for old files      # Running the tests to check for Old tomcat installation
7       stat:
8         path: /opt/apache-tomcat-9.0.37/
9       register: file
10
11    - name: Removing the old files      # Deleting the old tomcat files
12      command: "{{ item }}"
13      with_items:
14        - rm -rf /opt/apache-tomcat-9.0.37/
15      args:
16        warn: no
17      when: file.stat.isdir is defined
18
19    - name: Extracting Tomcat          # Placing new files in the machine
20      unarchive:
21        src: /tmp/apache-tomcat-9.0.37.tar.gz
22        dest: /opt/
23
24    - name: Configuring Tomcat         # Giving permissions to the tomcat installation and scripts
25      command: "{{ item }}"
26      with_items:
27        - chown -RH tomcat:tomcat /opt/apache-tomcat-9.0.37/
28        - chmod +x /opt/apache-tomcat-9.0.37/bin/startup.sh
29      args:
30        warn: no
31
32    - name: Starting Tomcat            # Starting the tomcat service
33      command: nohup /opt/apache-tomcat-9.0.37/bin/startup.sh
34      args:
35        warn: no
```

According to the configuration, These steps will be performed by Ansible:

- Check the old configuration of tomcat
- Remove the old files from the test-server
- Copy and extract the new files from Jenkins server
- Assign permissions to the tomcat directory for “tomcat” user
- Start the tomcat server

Step 4: Check the Post Build Section from the Job configuration

Jenkins > Tomcat Installation >

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

Build other projects

Projects to build Selenium Testing

☒ Trigger only if build is stable

☐ Trigger even if the build is unstable

☐ Trigger even if the build fails

Deploy war/ear to a container

WAR/EAR files **/*.war

Context path

Containers

Tomcat 9.x Remote

Credentials deployer/***** Add

Tomcat URL http://test-server:8080/ Advanced...

Add Container

The “Deployer war/ear to a container” plugin will deploy the ROOT.war to the tomcat server and install the web application. The next Job which gets executed is “Selenium Testing”

Job 3: Selenium Testing

Step 1: Click on the “Selenium Testing” job.

Jenkins

Jenkins > Selenium Testing >

Back to Dashboard

Status

Changes

Workspace

Build Now

Delete Project

Configure

Rename

Build History trend

find

Project Selenium Testing

Workspace

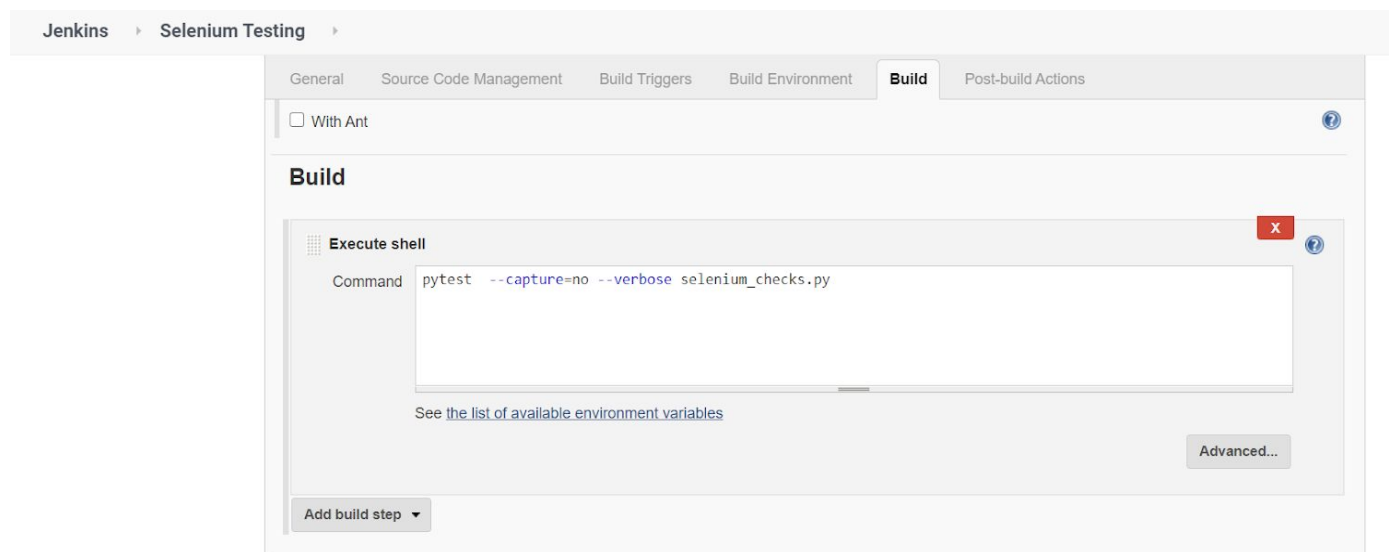
Recent Changes

Upstream Projects

Tomcat Installation

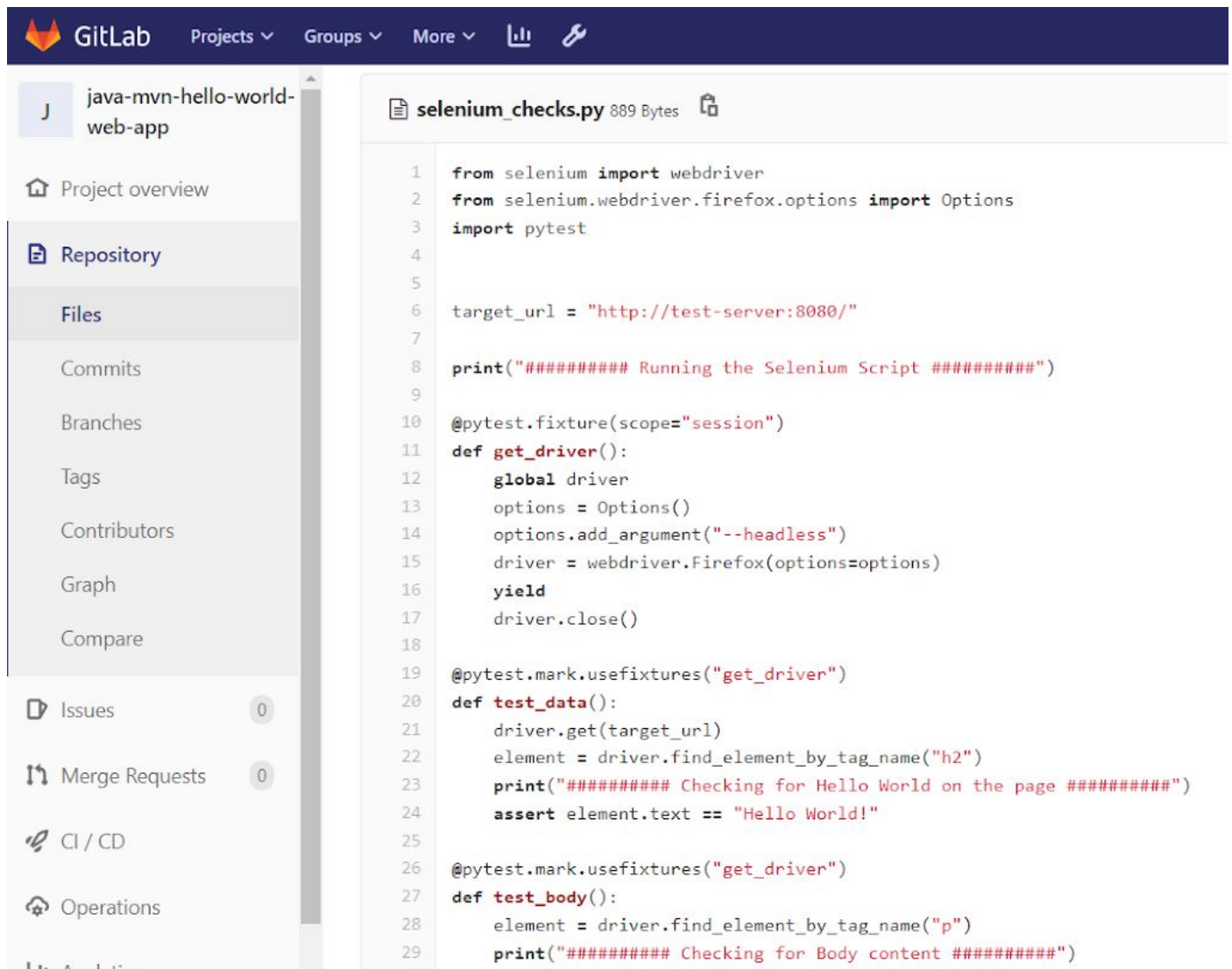
Permalinks

Step 2: Click on the “Configure” option to check the configuration of the Job.



The “selenium_checks.py” contains test cases which will perform tests on the web application.

Step 3: Check the selenium_checks.py file in the gitlab instance



The screenshot shows the GitLab interface for a project named 'java-mvn-hello-world-web-app'. The left sidebar contains navigation links: Project overview, Repository, Files, Commits, Branches, Tags, Contributors, Graph, Compare, Issues (0), Merge Requests (0), CI / CD, and Operations. The main content area displays the file 'selenium_checks.py' (889 Bytes). The code is a Python script using Selenium and pytest to automate browser tests.

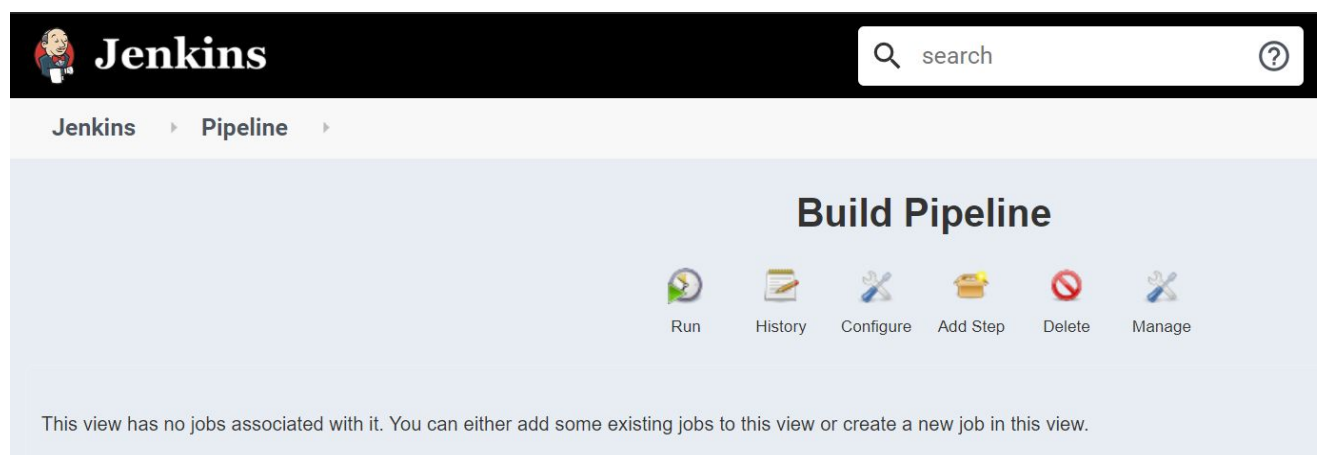
```
1 from selenium import webdriver
2 from selenium.webdriver.firefox.options import Options
3 import pytest
4
5
6 target_url = "http://test-server:8080/"
7
8 print("##### Running the Selenium Script #####")
9
10 @pytest.fixture(scope="session")
11 def get_driver():
12     global driver
13     options = Options()
14     options.add_argument("--headless")
15     driver = webdriver.Firefox(options=options)
16     yield
17     driver.close()
18
19 @pytest.mark.usefixtures("get_driver")
20 def test_data():
21     driver.get(target_url)
22     element = driver.find_element_by_tag_name("h2")
23     print("##### Checking for Hello World on the page #####")
24     assert element.text == "Hello World!"
25
26 @pytest.mark.usefixtures("get_driver")
27 def test_body():
28     element = driver.find_element_by_tag_name("p")
29     print("##### Checking for Body content #####")
```

The tests will do the following tasks:

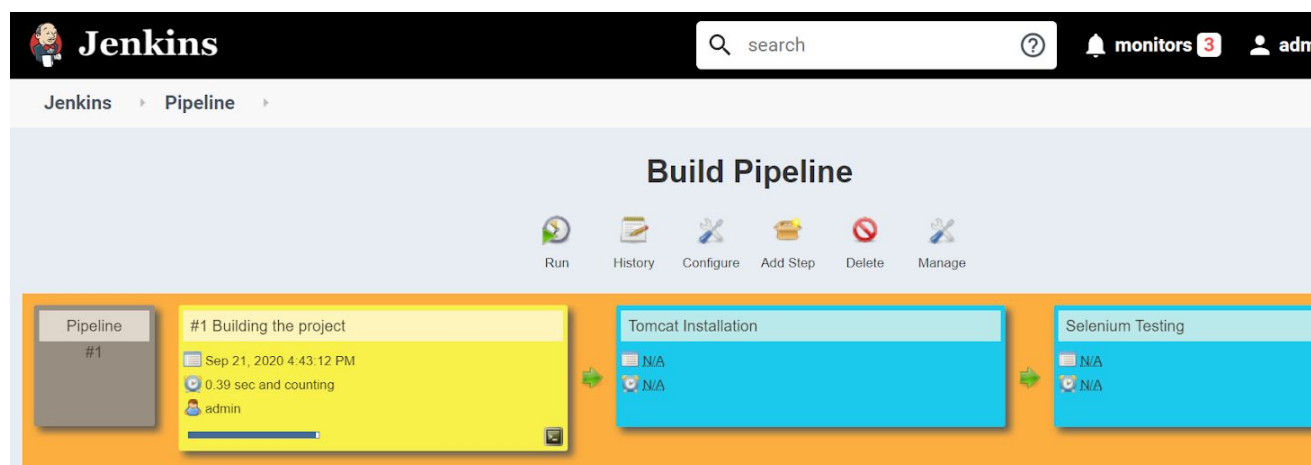
- Check if the “Hello World!” text exists in the h2 tag.
- Check if the “This is a sample application” exists under the ‘p’ tag

Pipeline Execution

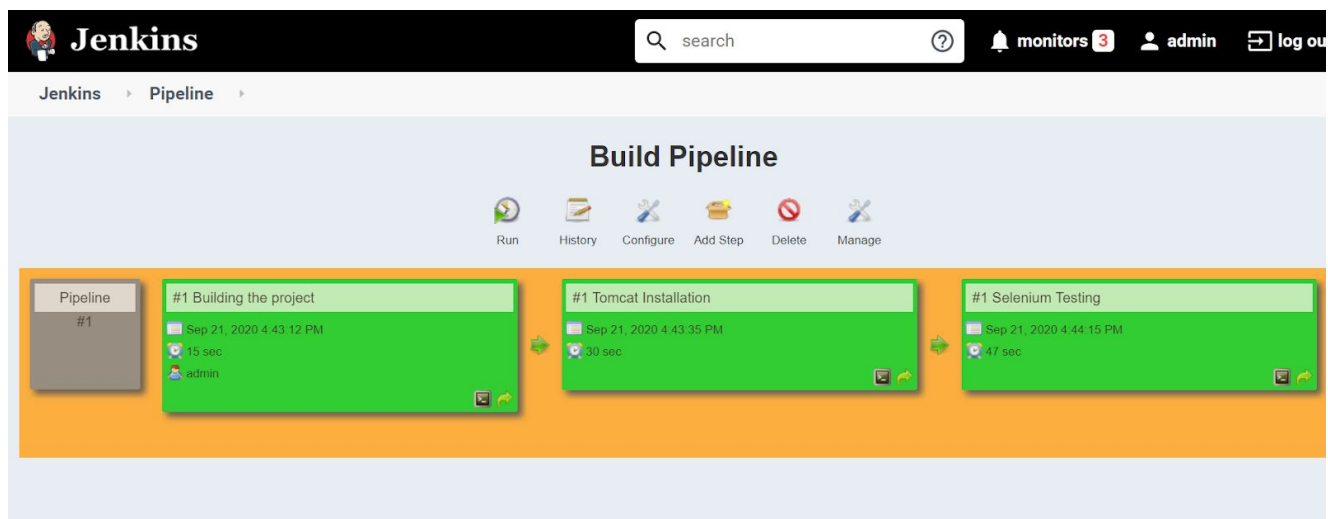
Step 1: Navigate to the Pipeline tab.



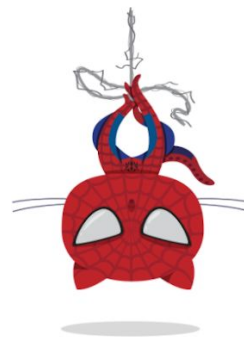
Step 2: Click on the Run button to start the Pipeline.



Reload the page to see the recent changes in the pipeline



Step 3: Navigate to the deployed website.



Hello World!

This is a sample application

Learning

Working of a simple DevOps pipeline consisting of different components.