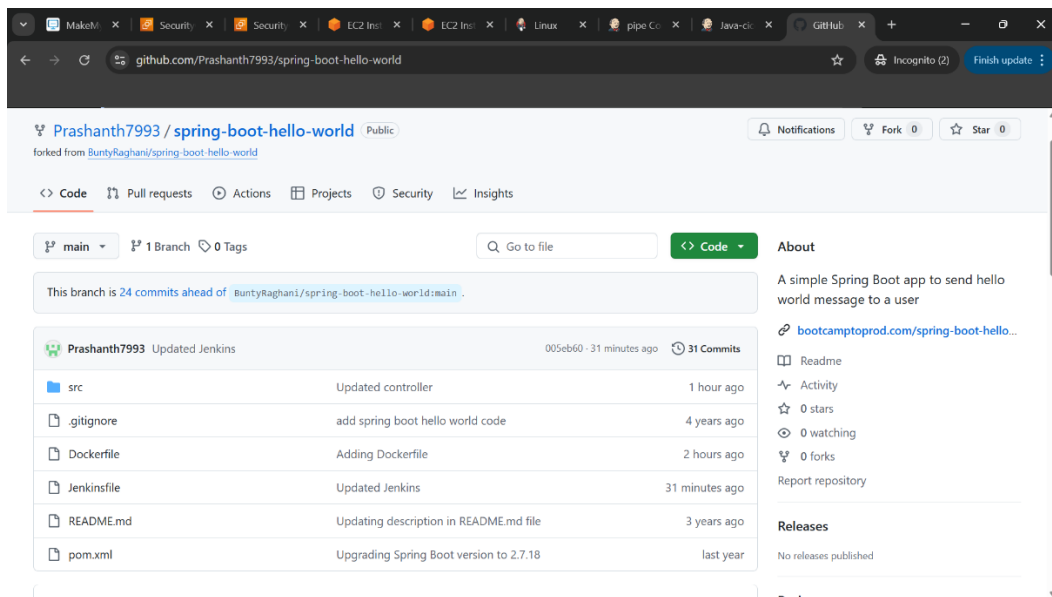


UID: 291226

Name: Prashanth J

## Task 1: Fork a Java Repository

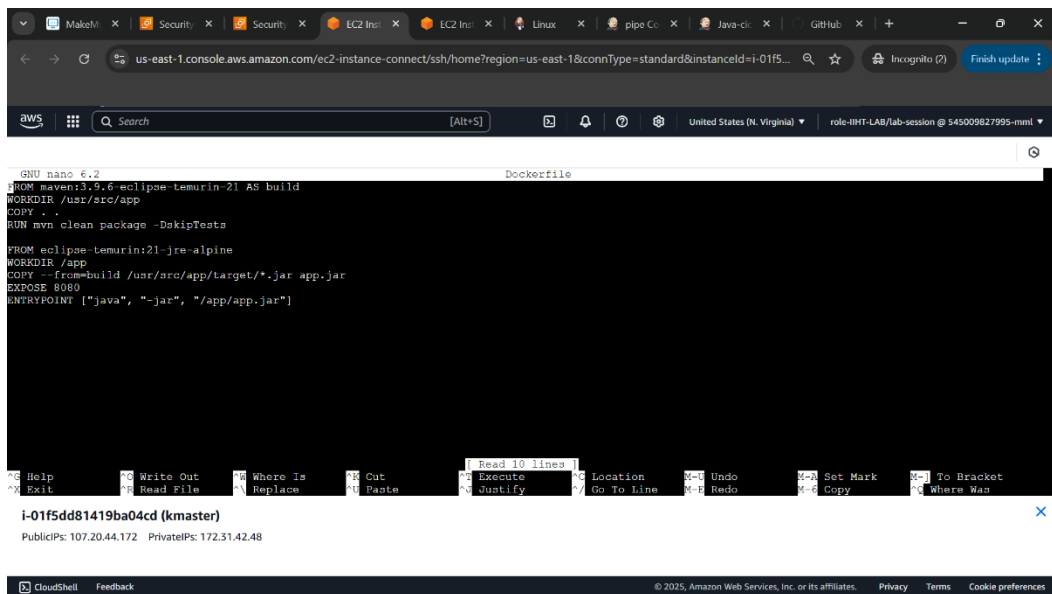
1. Fork the given Java HelloWorld Git repository from GitHub.
2. Clone it into your local machine.



## Task 2: Define a Dockerfile

Create a `Dockerfile` to containerize the Java application. It should:

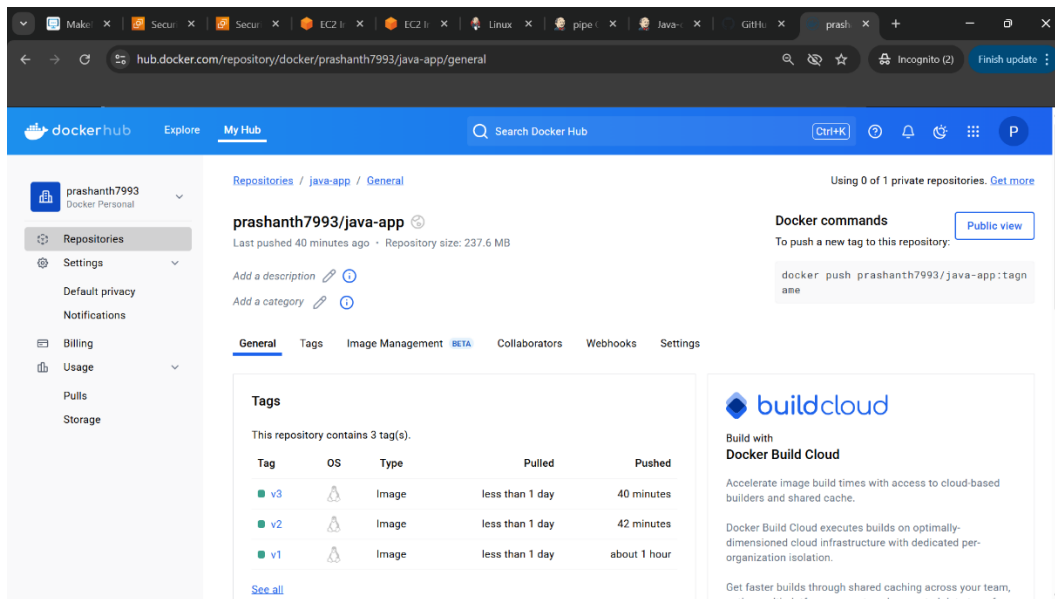
- Use an appropriate base image (e.g., OpenJDK).
- Copy the Java files into the container.
- Compile and execute the Java program.



UID: 291226  
Name: Prashanth J

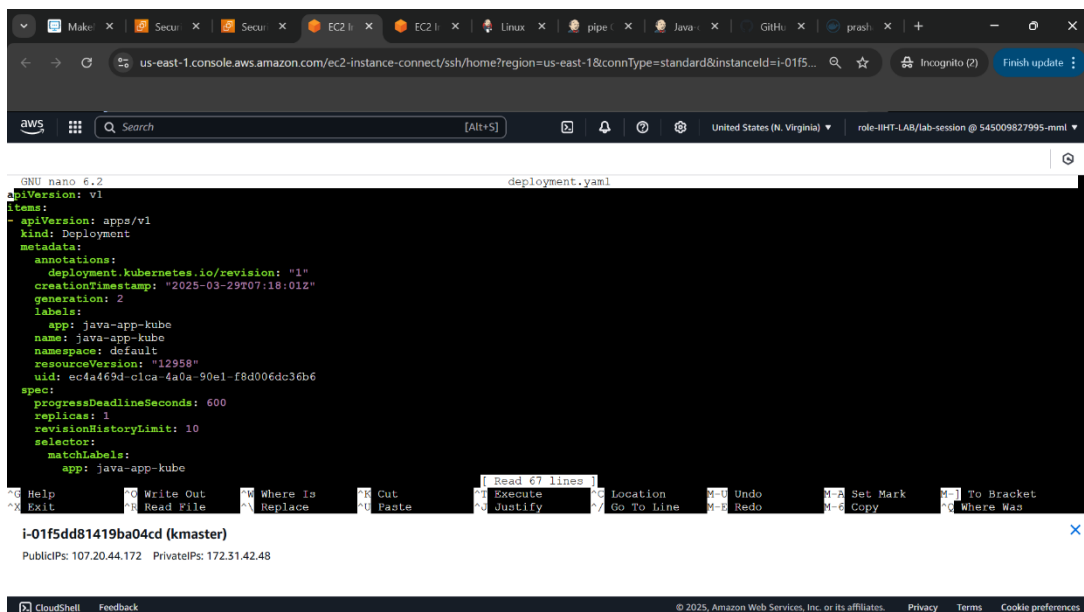
## Task 3: Build and Push Docker Image

1. Build a Docker image from the Dockerfile.



## Task 4: Create a Kubernetes Deployment

Define a Deployment YAML file ('deployment.yaml') for your application and apply it using 'kubectl apply -f deployment.yaml'.



UID: 291226  
Name: Prashanth J

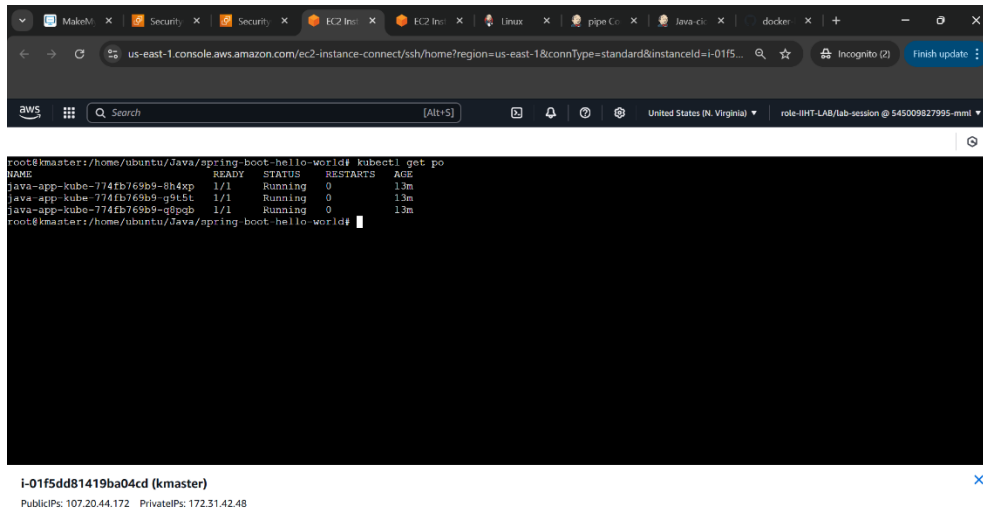
### Task 5: Scale Up and Scale Down

Scale out to 3 replicas:

```
kubectl scale deployment helloworld-deployment --replicas=3
```

Scale down to 1 replica:

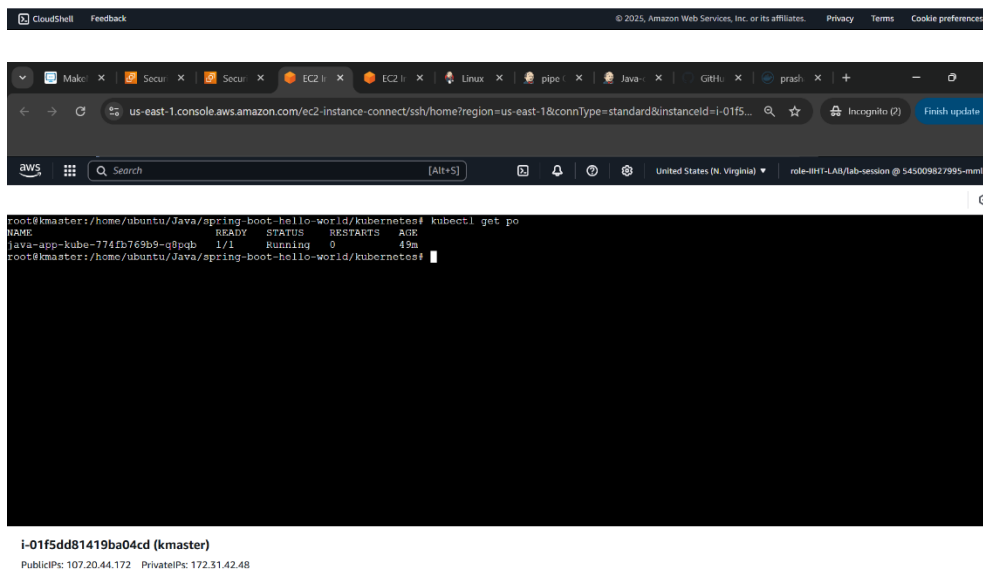
```
kubectl scale deployment helloworld-deployment --replicas=1
```



The screenshot shows a terminal window with the following output:

```
root@kmaster:/home/ubuntu/Java/spring-boot-hello-world# kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
java-app-kube-774fb769b9-8h4xp      1/1     Running   0           13m
java-app-kube-774fb769b9-g9t5t      1/1     Running   0           13m
java-app-kube-774fb769b9-g8pqb      1/1     Running   0           13m
root@kmaster:/home/ubuntu/Java/spring-boot-hello-world#
```

Below the terminal output, the instance ID is shown: **i-01f5dd81419ba04cd (kmaster)**. Public IP: 107.20.44.172, Private IP: 172.31.42.48.



The screenshot shows a terminal window with the following output:

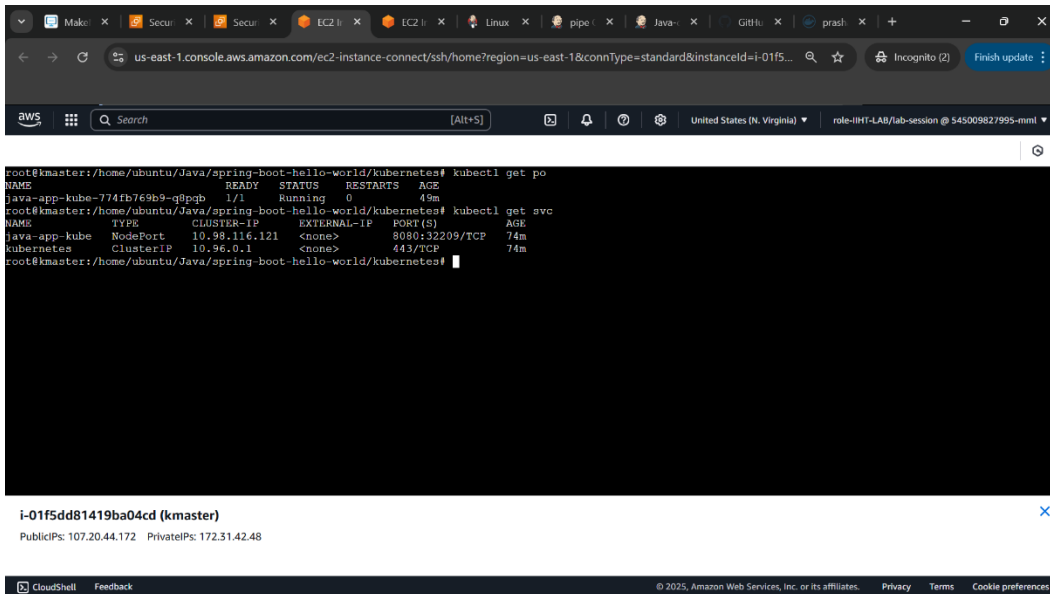
```
root@kmaster:/home/ubuntu/Java/spring-boot-hello-world/kubernetes# kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
java-app-kube-774fb769b9-g8pqb      1/1     Running   0           49m
root@kmaster:/home/ubuntu/Java/spring-boot-hello-world/kubernetes#
```

Below the terminal output, the instance ID is shown: **i-01f5dd81419ba04cd (kmaster)**. Public IP: 107.20.44.172, Private IP: 172.31.42.48.

UID: 291226  
Name: Prashanth J

## Task 6: Expose the Service Using NodePort

Create a `service.yaml` file and expose the service on a NodePort. Then, access it using `curl http://<public-ip>:30007`.



The screenshot shows a terminal window in AWS CloudShell. The user is running the following commands:

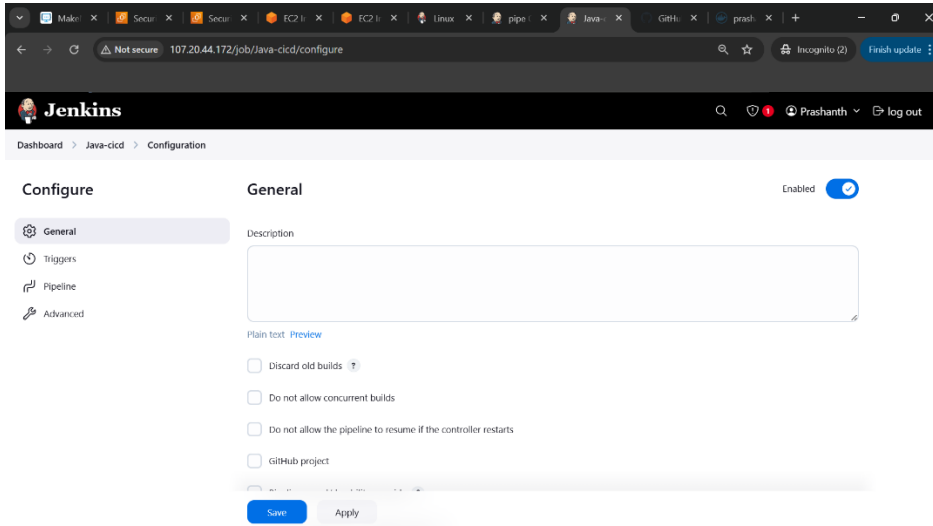
```
root@kmaster:/home/ubuntu/Java/spring-boot-hello-world/kubernetes# kubectl get po
NAME                                READY   STATUS    RESTARTS   AGE
java-app-kube-774fb769b9-g8pgb     1/1     Running   0           49m
root@kmaster:/home/ubuntu/Java/spring-boot-hello-world/kubernetes# kubectl get svc
NAME                                TYPE                CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
java-app-kube                       NodePort            10.98.116.121   <none>         8080:32209/TCP   74m
kubernetes                           ClusterIP           10.96.0.1       <none>         443/TCP          74m
```

Below the terminal output, the instance details for `i-01f5dd81419ba04cd (kmaster)` are shown:

PublicIPs: 107.20.44.172 PrivateIPs: 172.31.42.48

## Task 7: Automate Deployment Using Jenkins

Install Jenkins, create a Freestyle Jenkins Job, and configure GitHub webhooks to automate deployment.

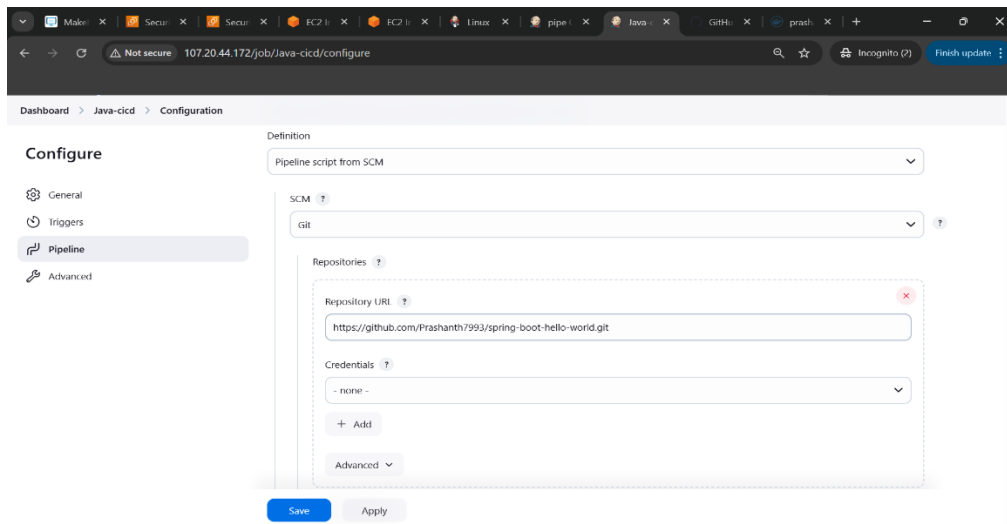


The screenshot shows the Jenkins Configuration page for a Freestyle Job. The "General" tab is selected, and the job is "Enabled". The configuration options are:

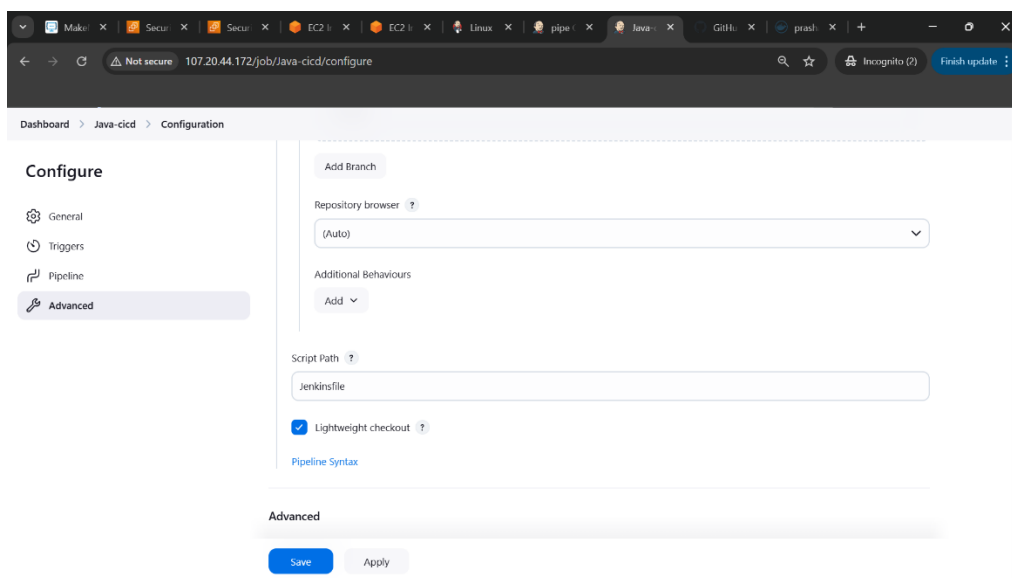
- Description: (Empty text area)
- Plain text: [Preview](#)
- ☐ Discard old builds
- ☐ Do not allow concurrent builds
- ☐ Do not allow the pipeline to resume if the controller restarts
- ☐ GitHub project

At the bottom, there are "Save" and "Apply" buttons.

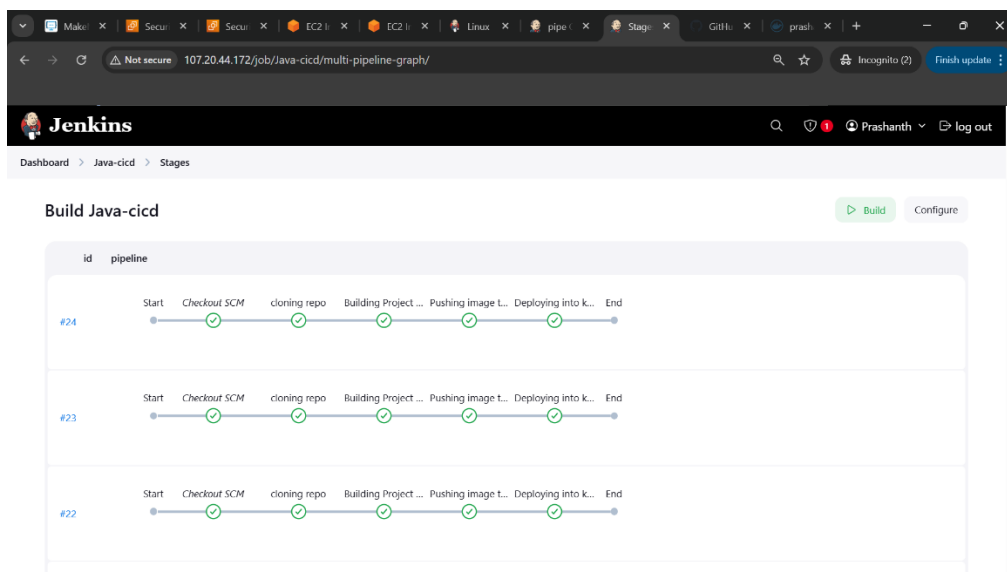
UID: 291226  
Name: Prashanth J



The screenshot shows the Jenkins Pipeline Configuration page for a job named 'Java-cicd'. The left sidebar contains links for 'General', 'Triggers', 'Pipeline', and 'Advanced', with 'Pipeline' currently selected. The main area is titled 'Configure' and shows the 'Definition' section with 'Pipeline script from SCM' selected. Below this, the 'SCM' is set to 'Git'. A 'Repositories' section contains a single entry with 'Repository URL' as 'https://github.com/Prashanth7993/spring-boot-hello-world.git' and 'Credentials' set to '- none -'. At the bottom, there are 'Save' and 'Apply' buttons.



This screenshot shows the 'Advanced' section of the Jenkins Pipeline Configuration page. The 'Add Branch' button is visible at the top. The 'Repository browser' is set to '(Auto)'. Under 'Additional Behaviours', there is an 'Add' button. The 'Script Path' is set to 'Jenkinsfile'. The 'Lightweight checkout' checkbox is checked. A 'Pipeline Syntax' link is present. At the bottom, 'Save' and 'Apply' buttons are visible.



UID: 291226  
Name: Prashanth J

The screenshot shows the Jenkins web interface in a browser. The address bar indicates the URL is 107.20.44.172/job/Java-cicd/. The Jenkins logo and name are at the top left. A navigation bar on the left lists various actions: Status, Changes, Build Now, Configure, Delete Pipeline, Stages, Rename, Pipeline Syntax, and GitHub Hook Log. The main content area displays the job name 'Java-cicd' with a green status icon and an 'Add description' button. Below this, a 'Permalinks' section lists several build links with their respective times. At the bottom left, a 'Builds' section shows a list of builds, with the most recent one being #24, completed today at 7:17 AM.

Dashboard > Java-cicd >

**Status** ✓ **Java-cicd** [Add description](#)

**Permalinks**

- [Last build \(#24\)](#), 53 min ago
- [Last stable build \(#24\)](#), 53 min ago
- [Last successful build \(#24\)](#), 53 min ago
- [Last failed build \(#21\)](#), 1 hr 2 min ago
- [Last unsuccessful build \(#21\)](#), 1 hr 2 min ago
- [Last completed build \(#24\)](#), 53 min ago

**Builds**

Filter

Today

- ✓ #24 7:17 AM