**PROJECT SPECIFIC: Stackfusion - Assignment.**

Launched 5 EC2 Instances : Amaznon Linux 2.

1.Jenkins\_Master - jenkins\_Master.

2.Jenkins\_Slave - app\_build\_server.

3.KuberMaster - kube\_master.

4.Kubernetes WN1 - kube\_worknode1.

5.Monitoring server - monitoring\_server.

Phase I :

Launched 5 these EC2 Instances : Amaznon Linux 2 using Terraform.

Installed & Config required tools in all these servers.

Using Ansible Configuration Management: Ansible:

Tools installed –

1.Jenkins\_Master - jenkins\_Master - git,jdk,jenkins

2.Jenkins\_Slave - app\_build\_server - git,jdk,maven,docker, valid user

3.KuberMaster - kube\_master -used configuration file to install k8s.

4.Kubernetes WN1 - kube\_worknode1 - used configuration file to install k8s.

5.Monitoring server - monitoring\_server – Prometheus , Grafana & Node Exporters.

Phase II :

Create CI/CD Pipeline : End-to-End Pipeline

stage1 – **SCM Checkout &** **Maven Build**

stage2 – **Docker build, Login to DockerHub, Push to DockerHub**

stage3 – **Deployment to Kubernetes Cluster.**

stage5 – **Monitor the servers using Monitoring Tools.**

GITHUB REPOSITORY FOR INSURANCE DOMIAN: https://github.com/balcha95/StackFusion\_DevOps-Engineer-Assignment-Hello\_world.git

**Task 1: Infrastructure as Code (IaC)**

Using a configuration management tool of your choice (such as Ansible, Puppet, or Chef), create an Infrastructure as

Code (IaC) solution to provision a web server. The web server should meet the following requirements:

1. Use a Linux-based operating system (e.g., Ubuntu, CentOS) of your choice.

2. Install and configure web server software (e.g., Apache, Nginx).

3. Secure the web server with SSL/TLS encryption (using self-signed certificates is acceptable).

4. Configure firewall rules to allow HTTP (80) and HTTPS (443) traffic.

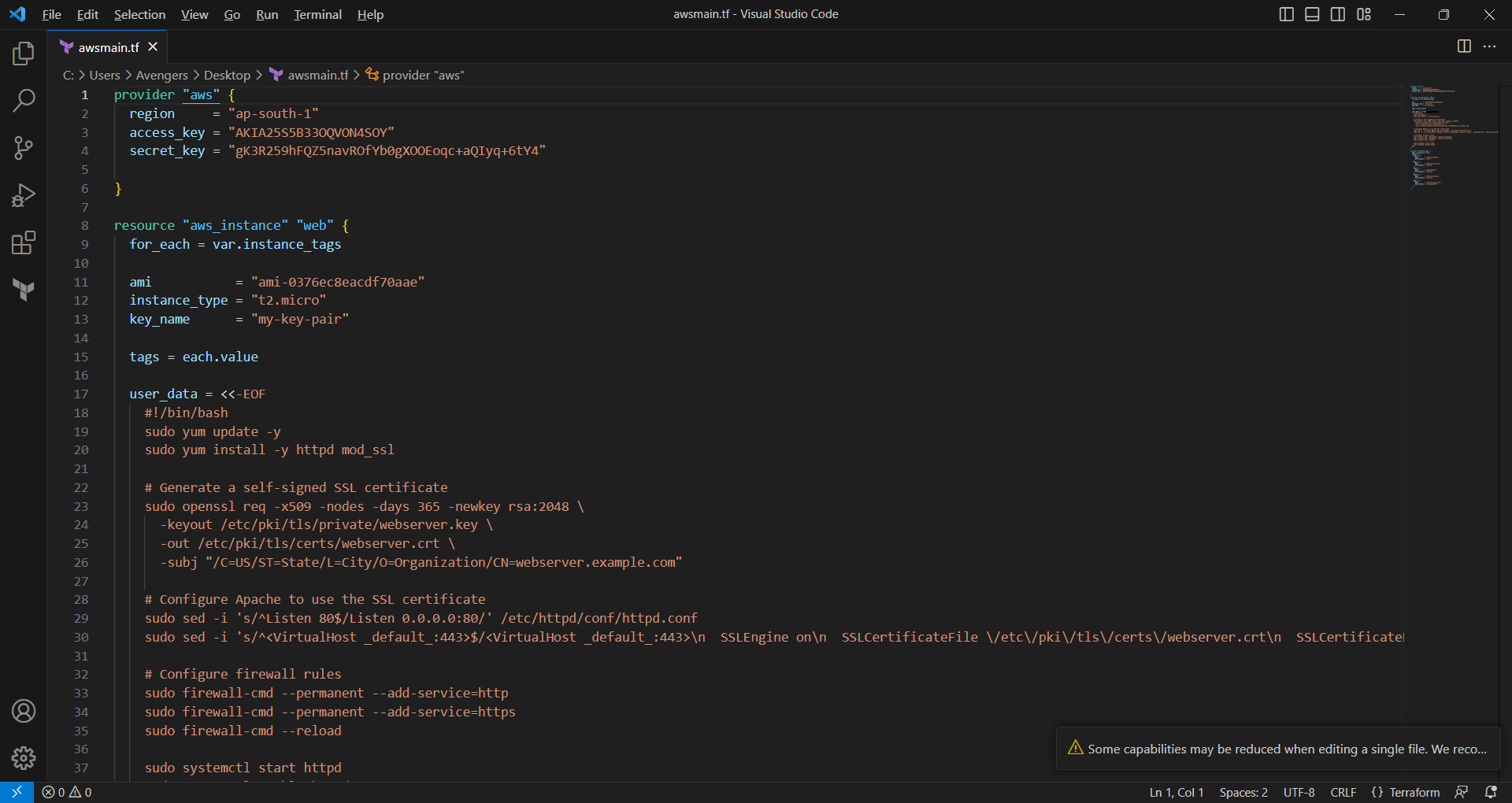
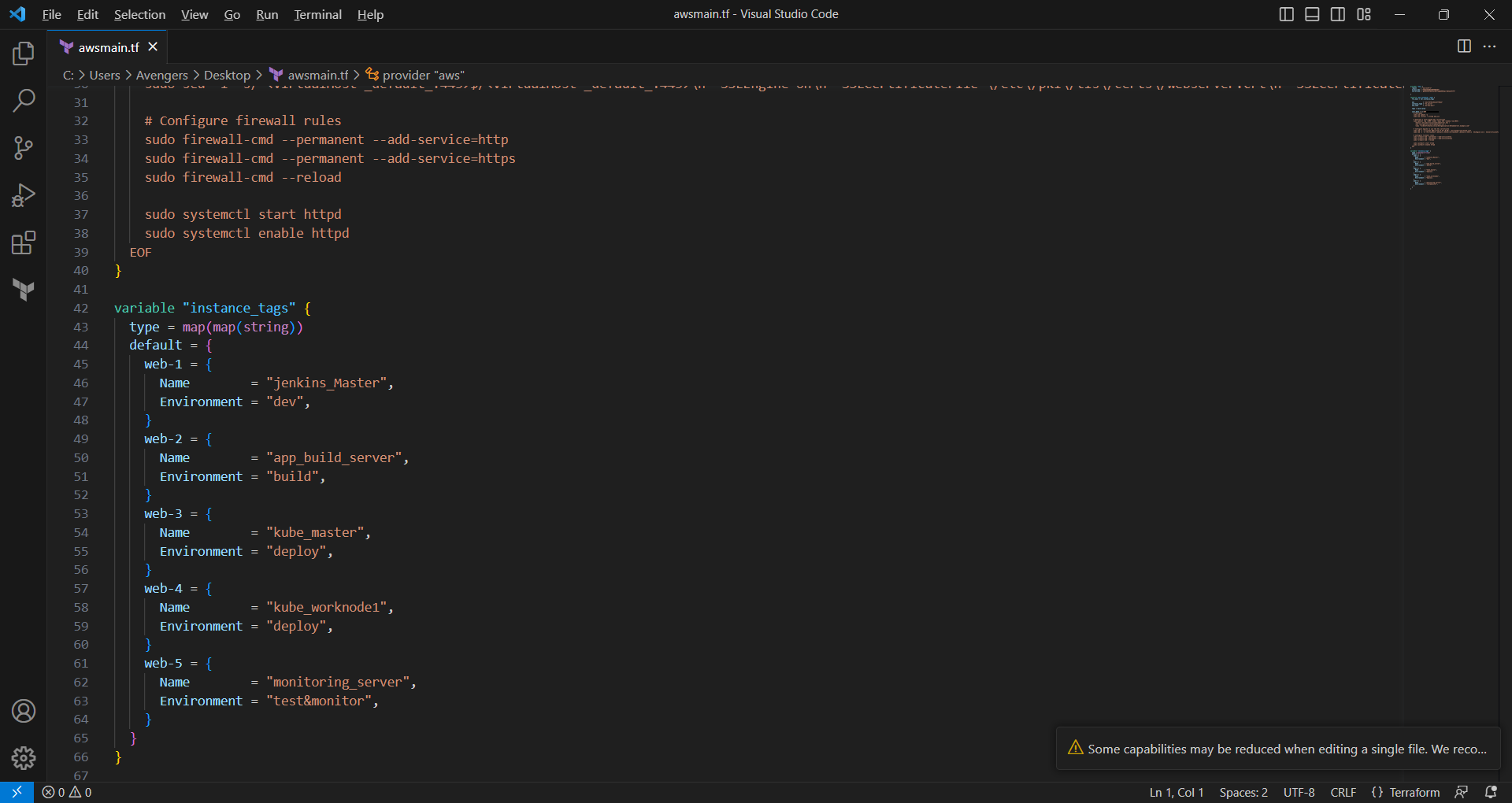
5. Automate the deployment and configuration of the web server using your chosen tool.

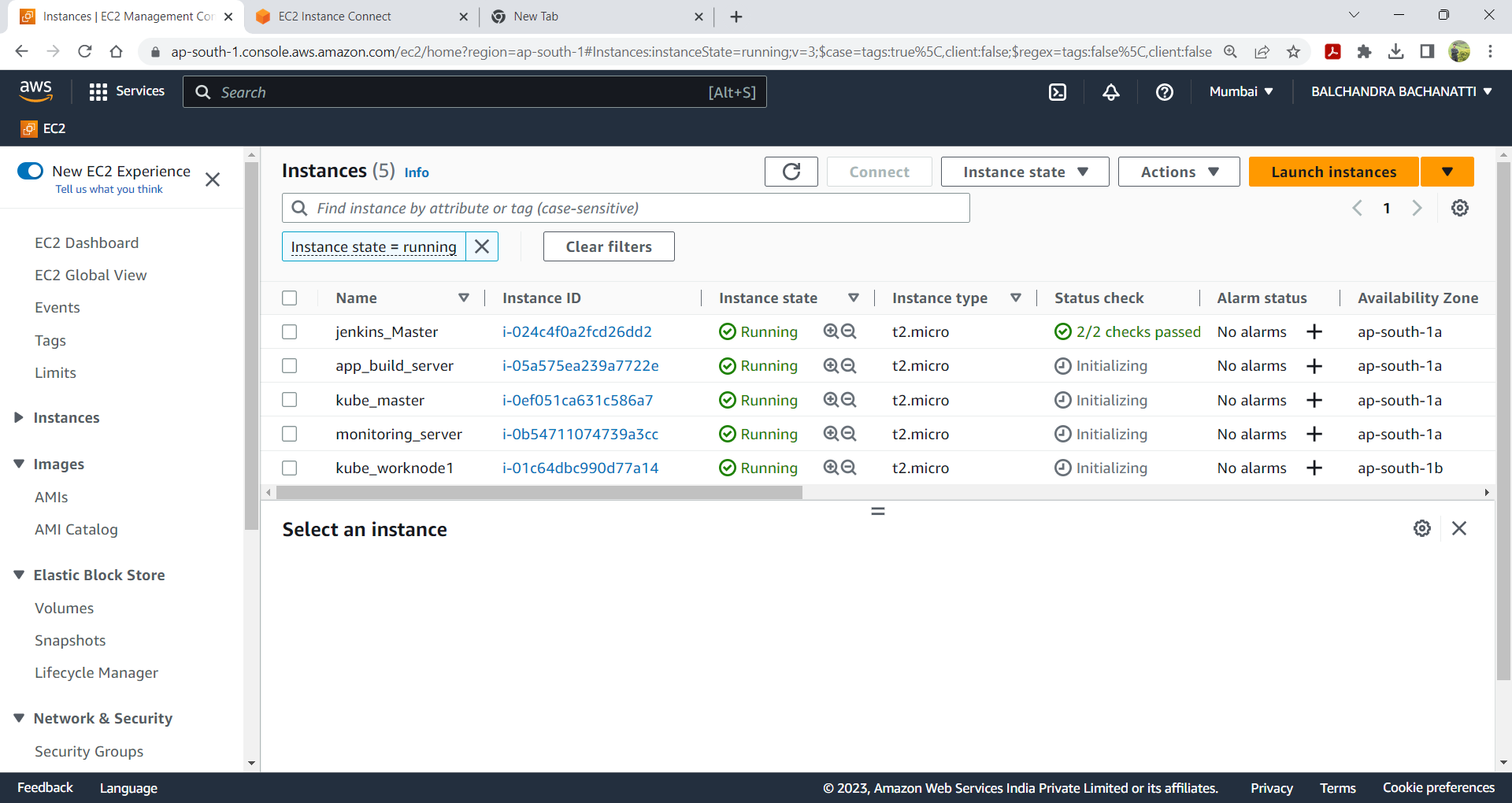
**NOTE:** Provide all the necessary scripts and configuration files required to provision the web server.

**Phase I:**

Launched 5 these EC2 Instances: Amaznon Linux 2 using Terraform.

STEPS - I used TERRAFORM to setup servers: For creating ec2 machines as servers and deploy the web application.

1.2.

3.

**Task 2: Continuous Integration/Continuous Deployment (CI/CD)**

Implement a basic CI/CD pipeline using the version control system (VCS) and CI/CD tool of your choice (e.g., Git,

Jenkins, Travis CI). The pipeline should include the following steps:

1. Whenever a change is pushed to the repository, the pipeline should trigger an automated build process.

2. The build process should include any necessary steps for compiling, testing, and packaging the application

(assume a simple Hello World web application).

3. After a successful build, the application should be deployed to the web server provisioned in Task 1.

4. The pipeline should be configured to run on every push to the main branch.

**NOTE:** Provide all the necessary configuration files and scripts required to set up the CI/CD pipeline.

Phase II:

Create CI/CD Pipeline: End-to-End Pipeline

stage1 – **SCM Checkout &** **Maven Build**

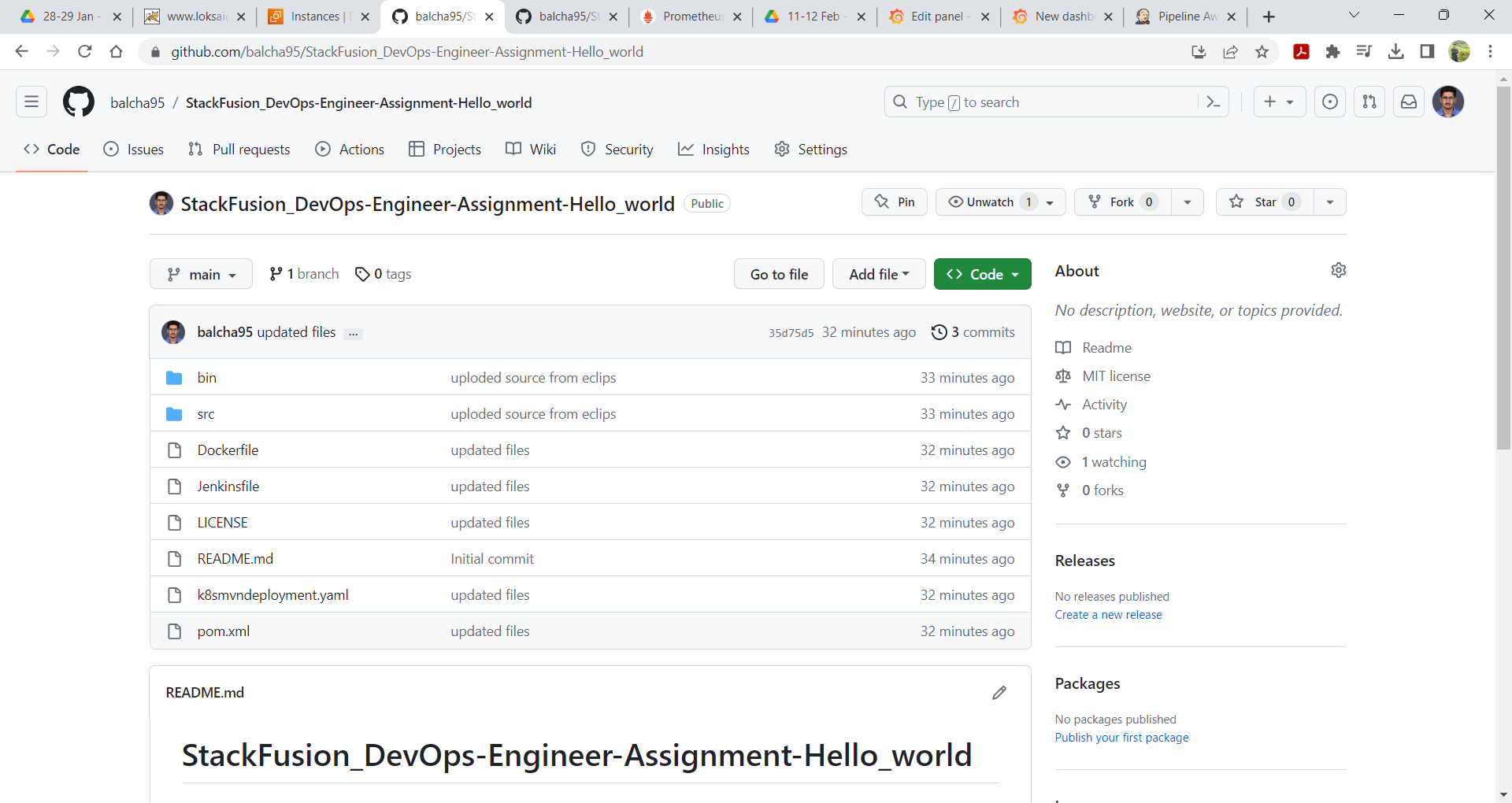
stage2 – **Docker build, Login to DockerHub, Push to DockerHub**

stage3 – **Deployment to Kubernetes Cluster.**

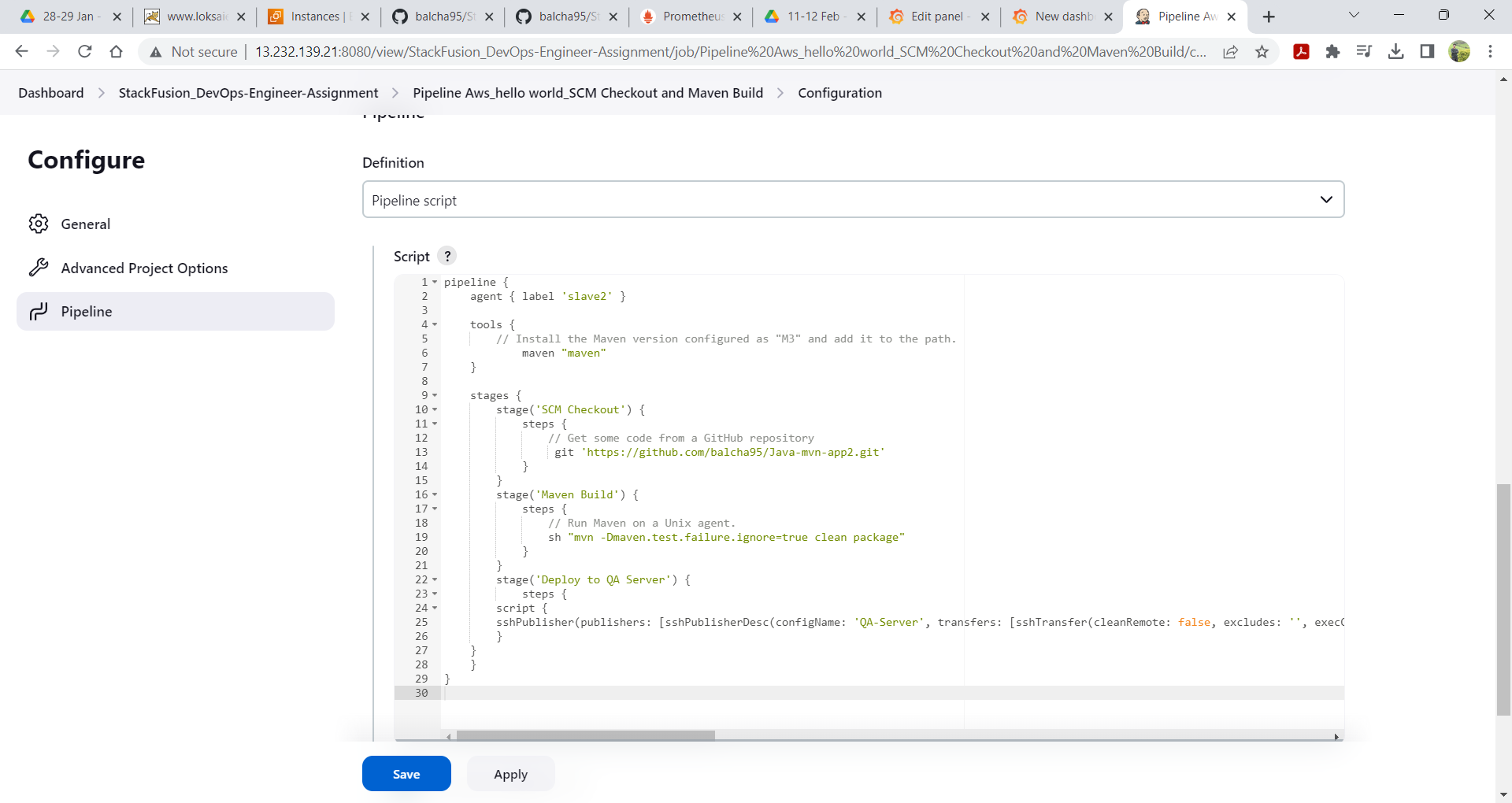
Stage4 – **Monitor the servers using Monitoring Tools.**

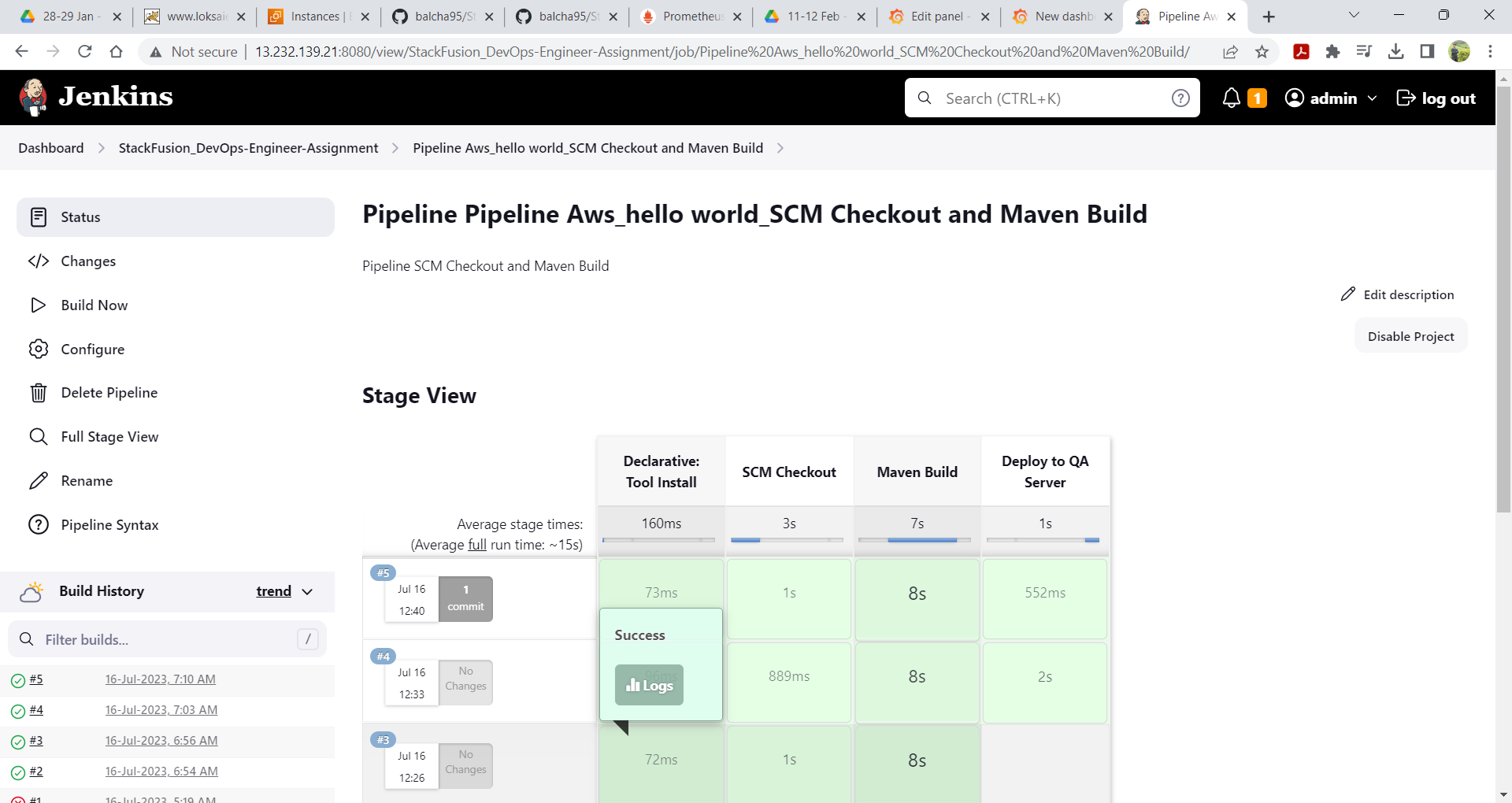
1.Create CI/CD Pipeline: End-to-End Pipeline

1. Git - For version control for tracking changes in the code files.

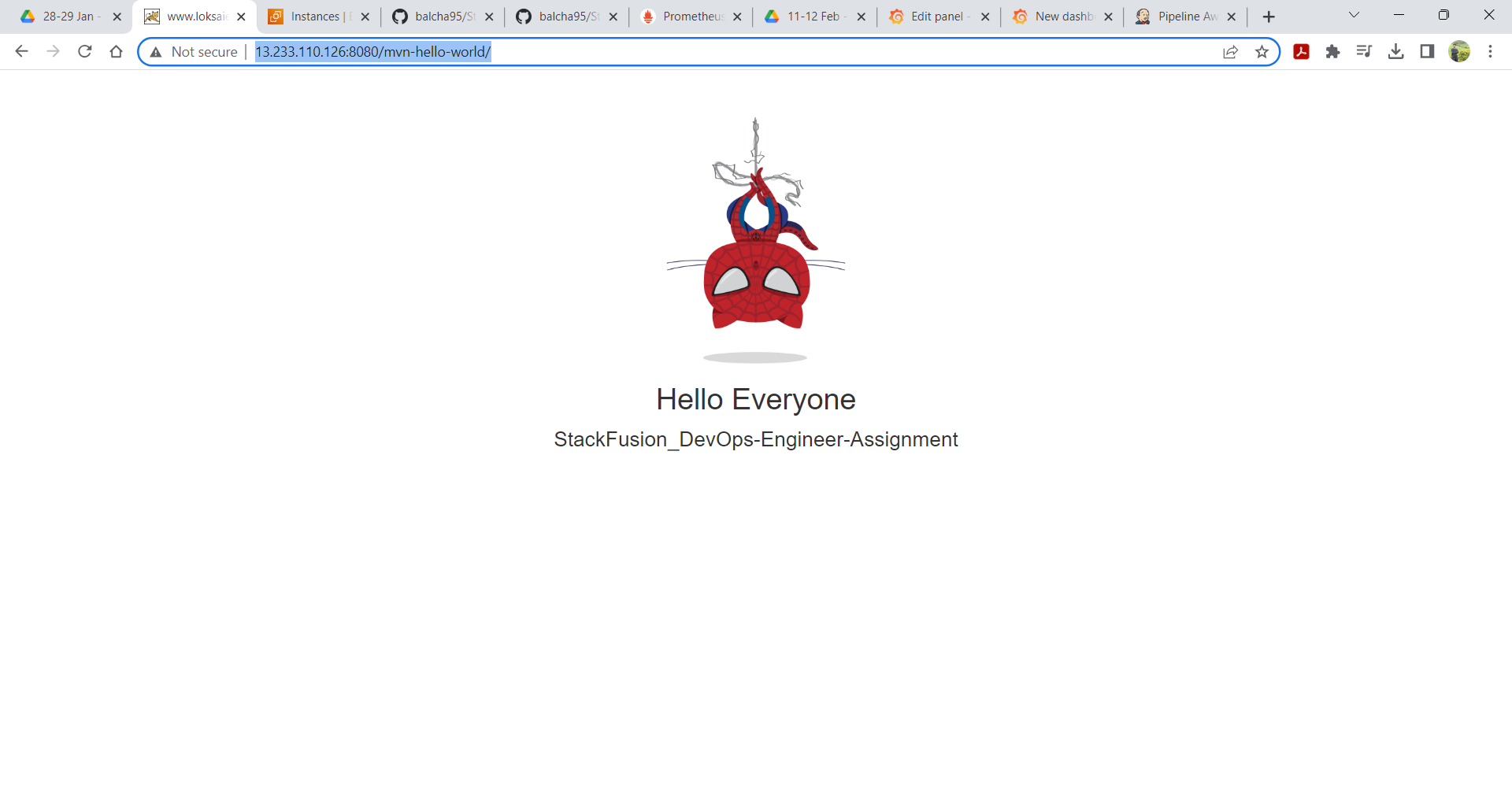


2. stage1 – **SCM Checkout &** **Maven Build & deployed application on** **Apache Tommcat.**



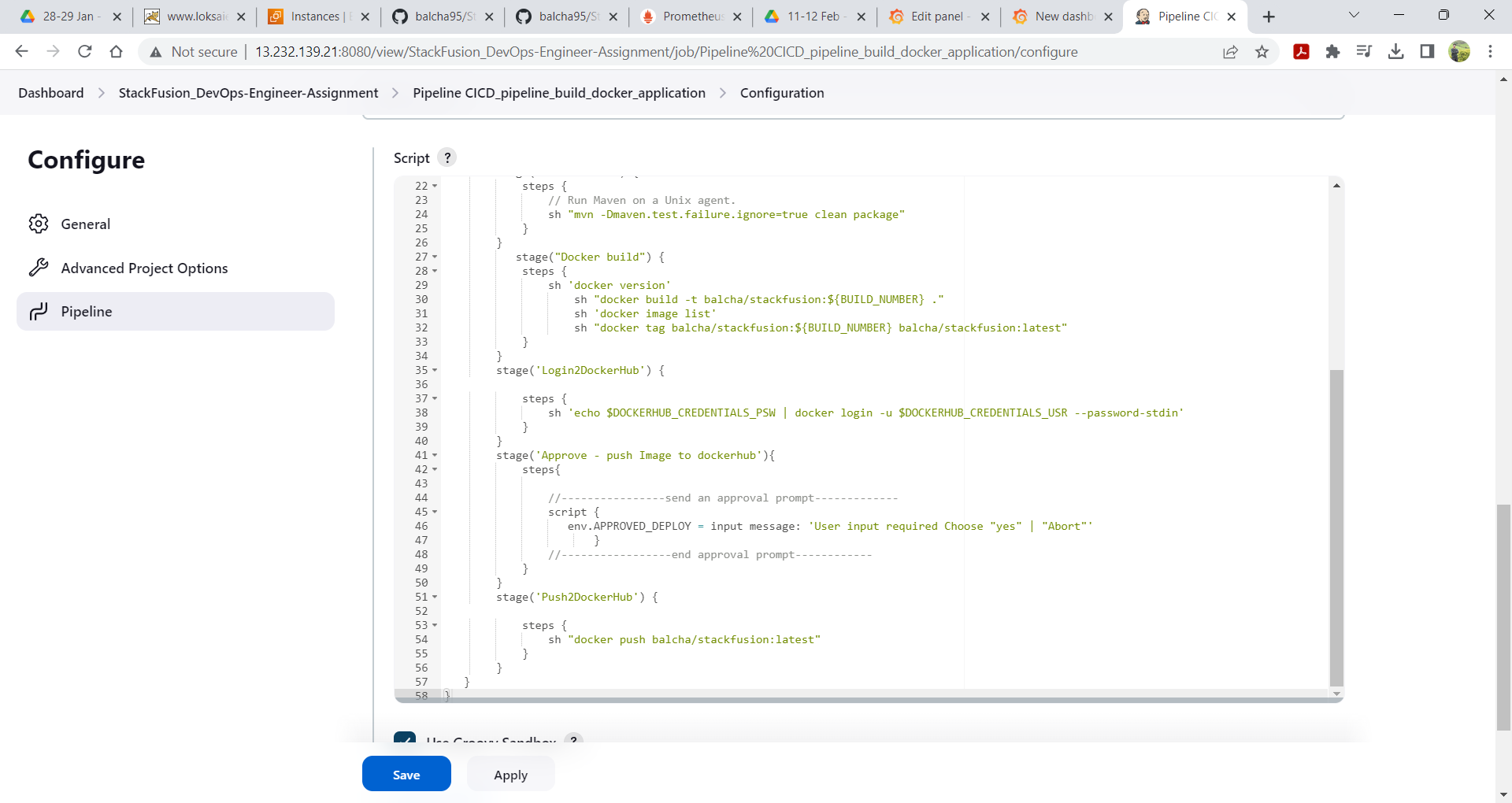
3

**4.Deployed application on** **Apache Tommcat.**

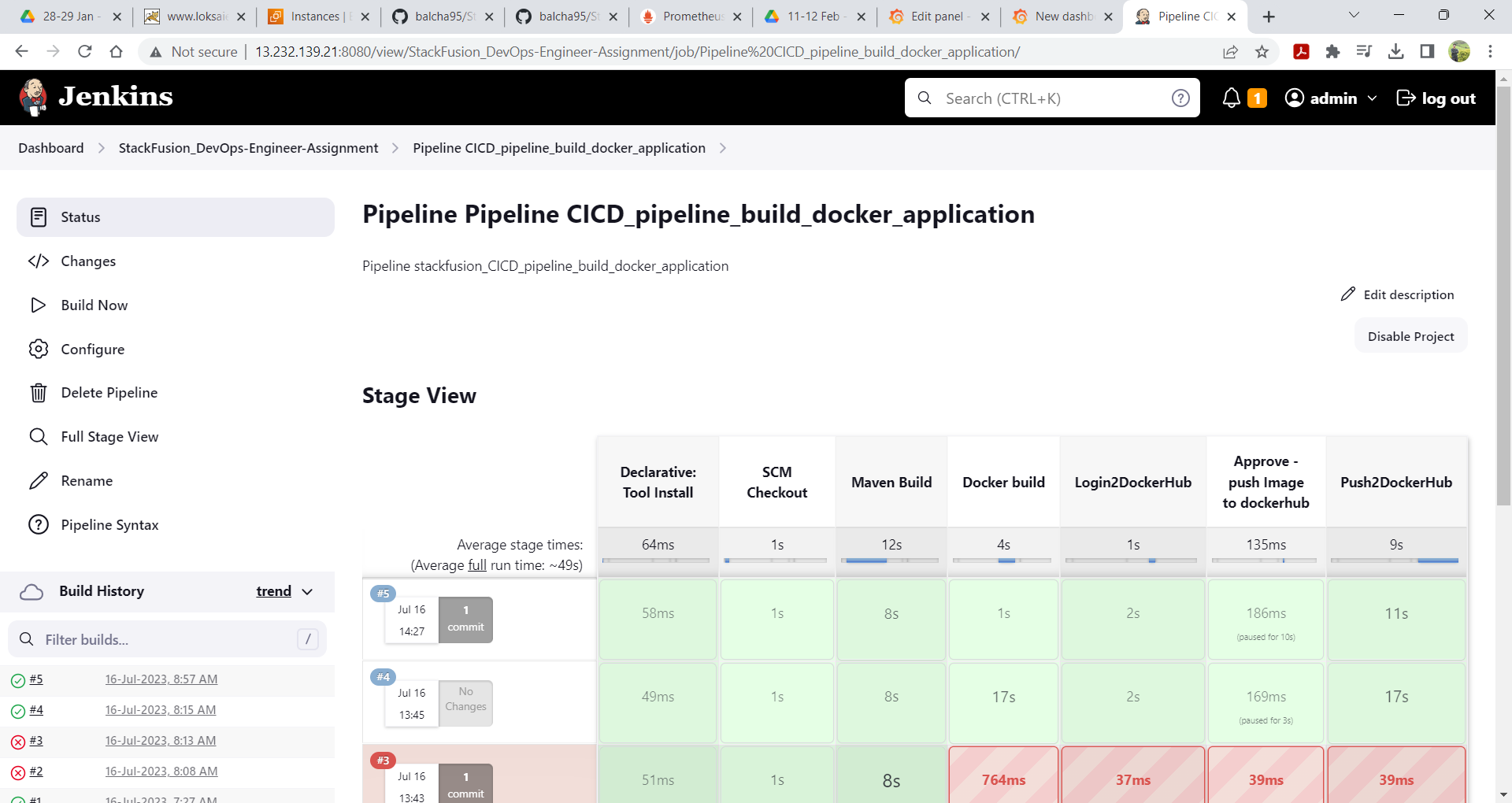


stage2 – **Docker build , Login to DockerHub , Push to DockerHub.**

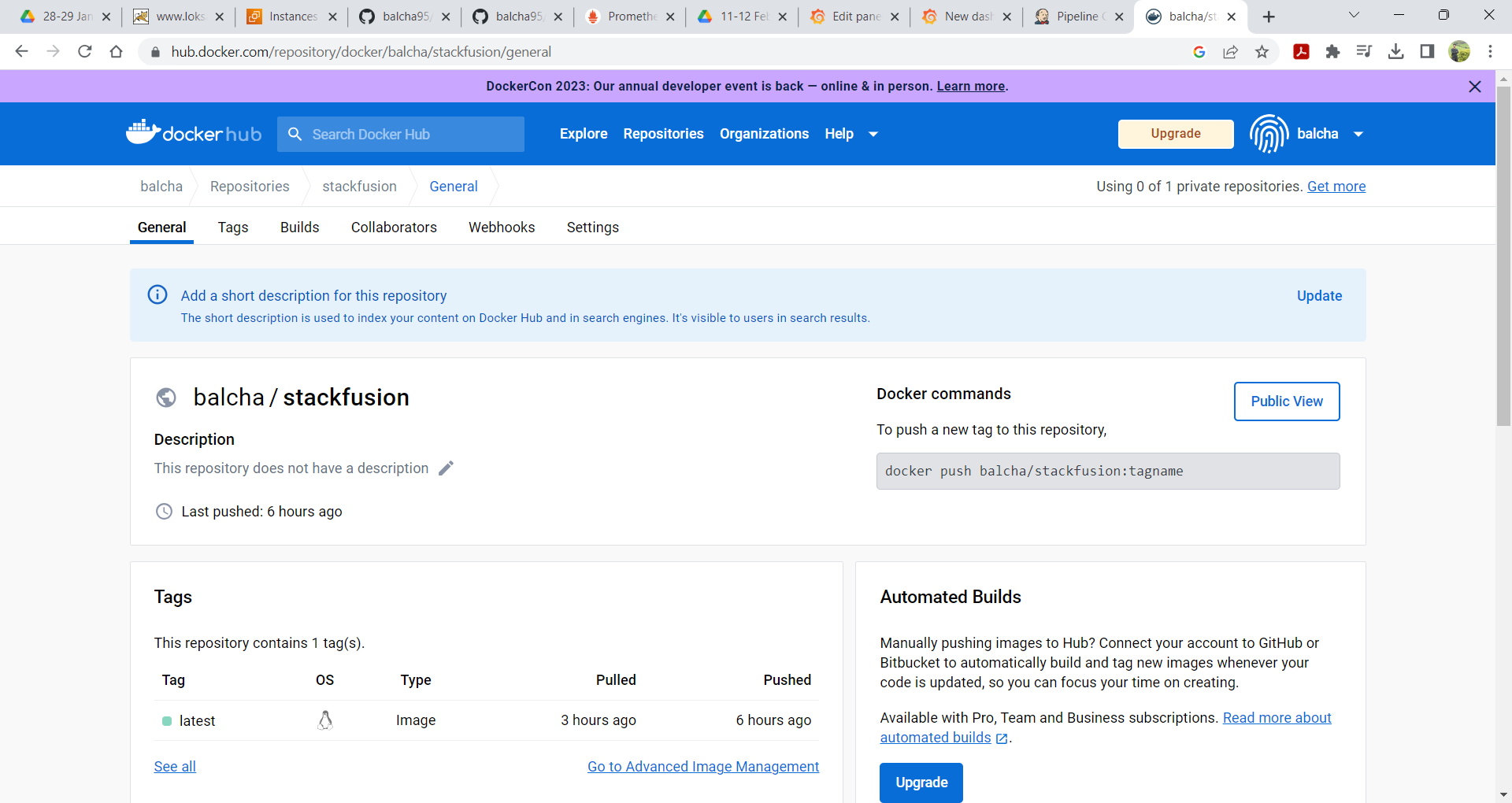
1.



2.



3. **DockerHub.**



**#Github webhook to automatically trigger build pipeline!**

#Goto Github Repository setting,

Select webhook,

Click Add Webhook

Enter Jenkins Master URL. eg.:

http://<public-IP>:8080/github-webhook/

http://13.232.139.21:8080/github-webhook/

Choose the push event and save the webhook configuration

**Task 4: Kubernetes and MicroServices Architecture**

To create any microservice centralized application so Kubernetes services for local development.

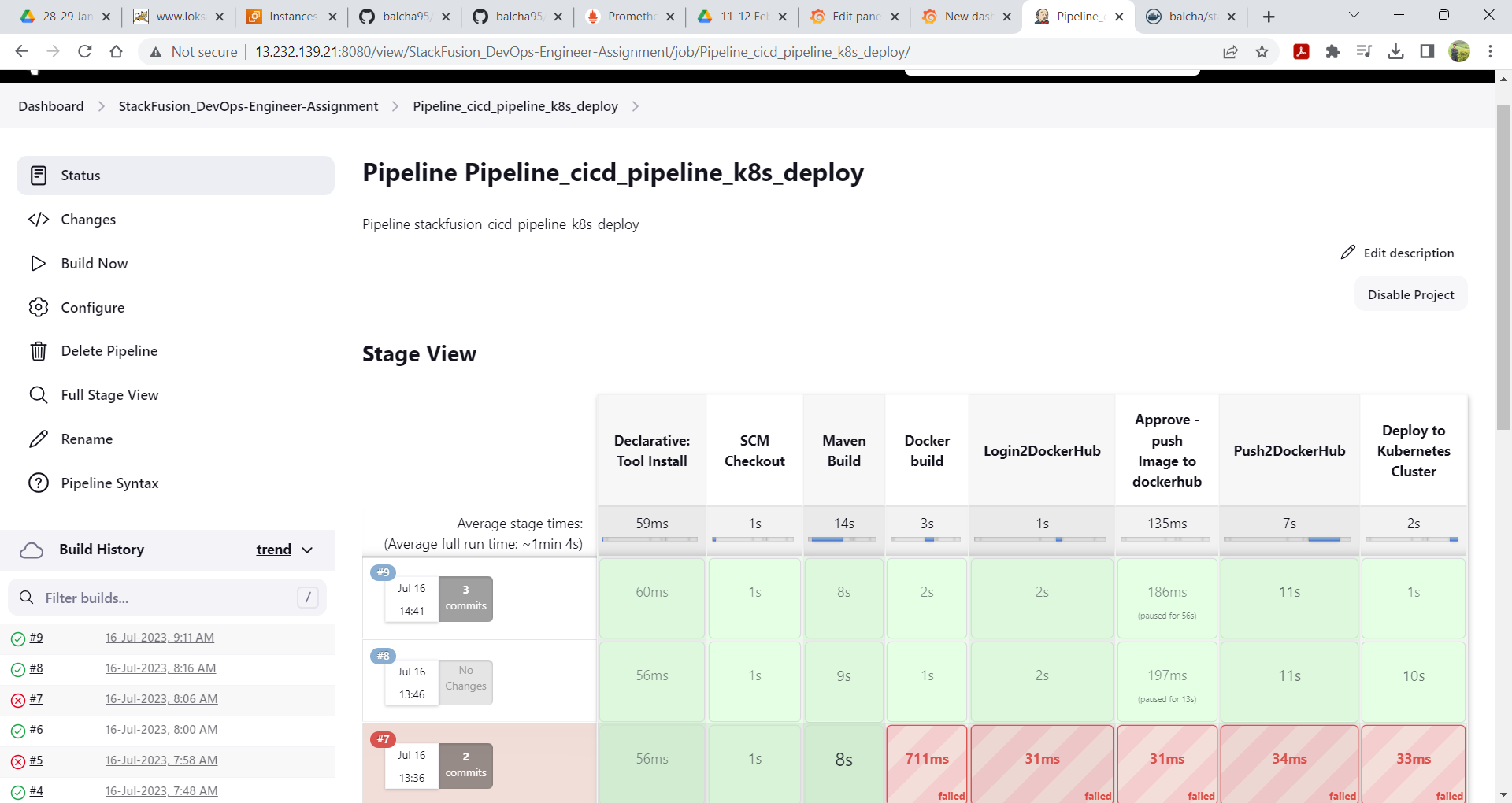
● Deploy an Application that should act as a mediator between a client and your local system so that your local

environment will communicate with the mediator and the mediator will send the appropriate response to the

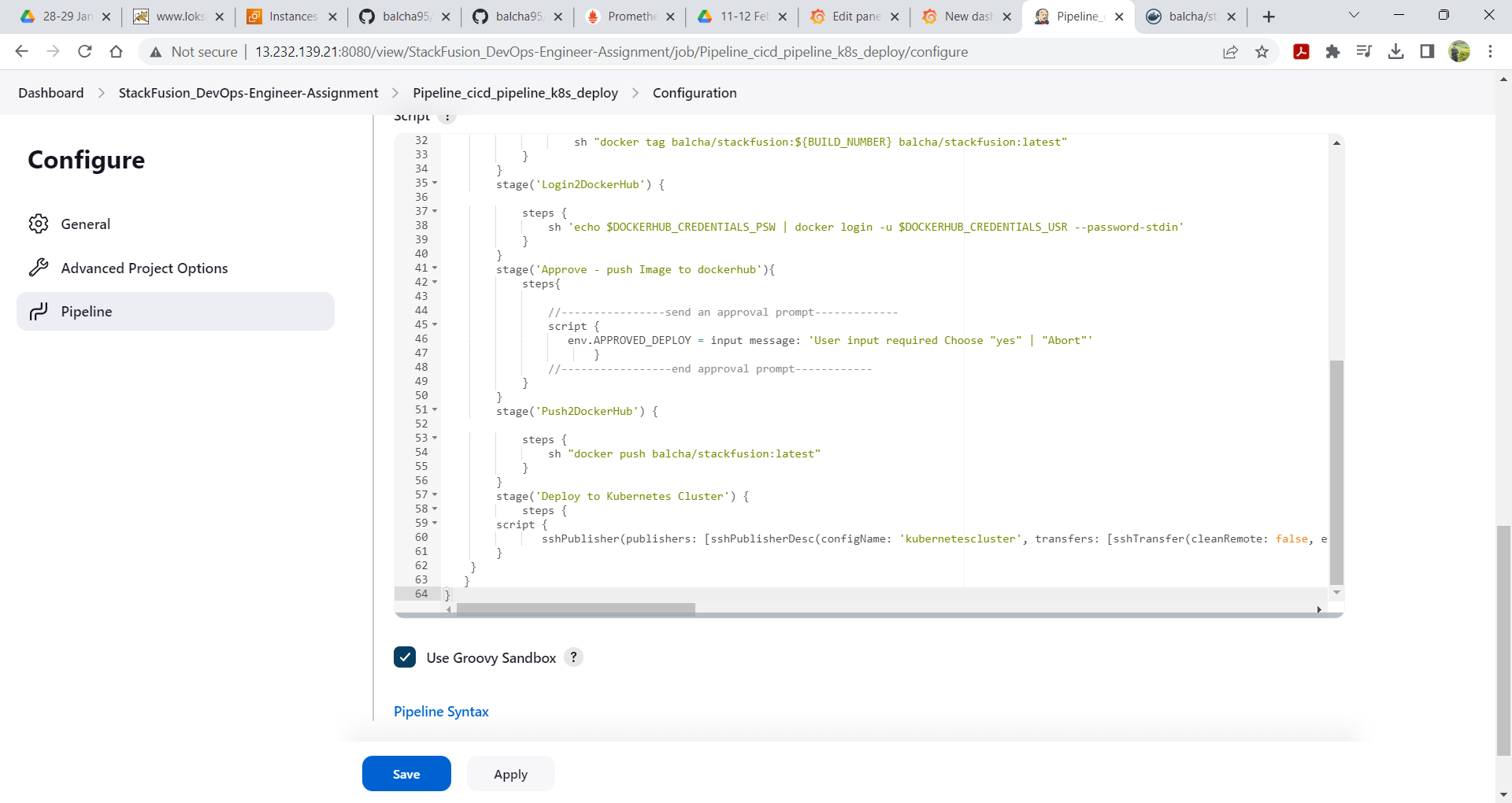
client application

**stage3 – Deployment to Kubernetes Cluster.**

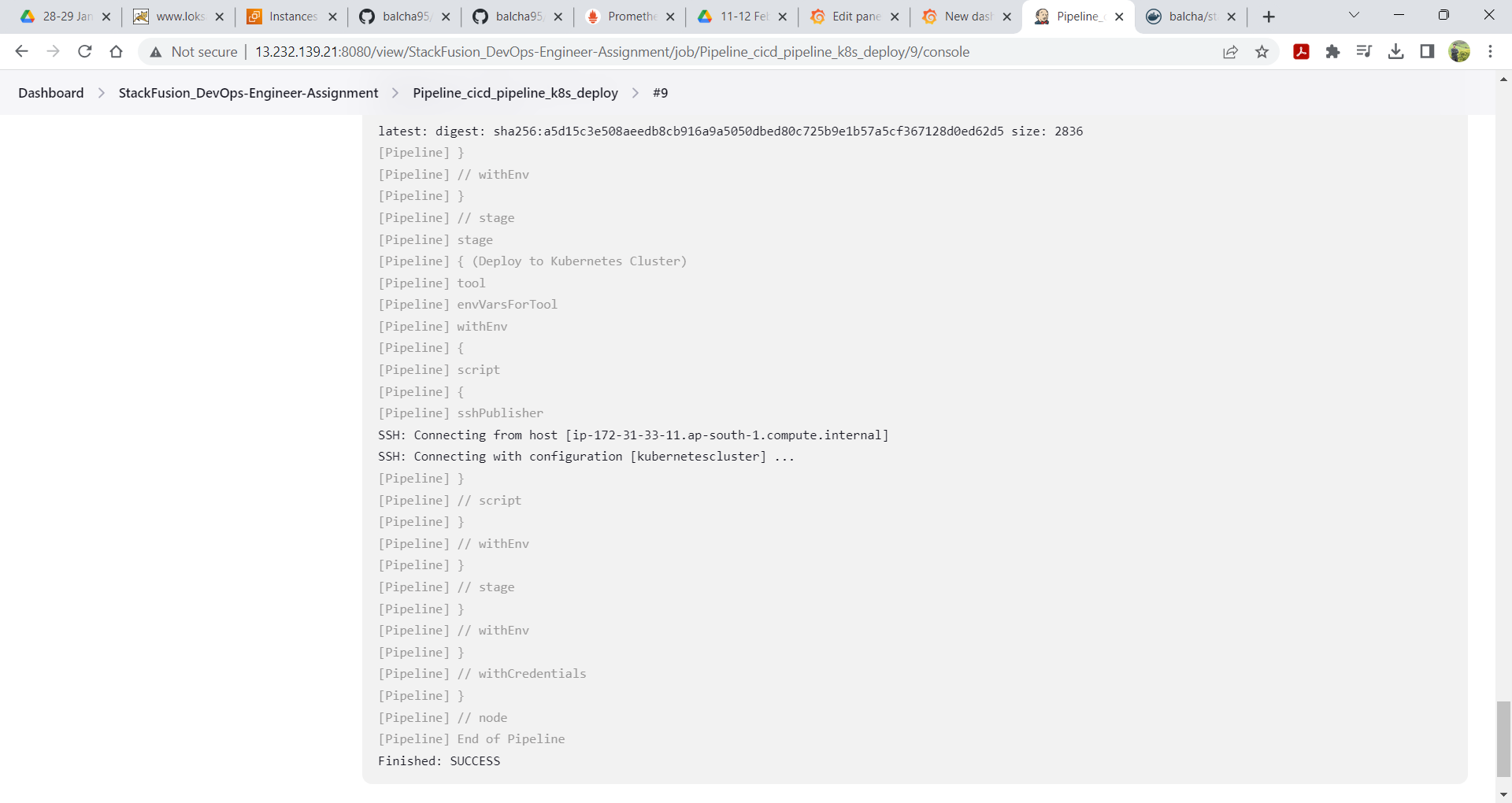
1.



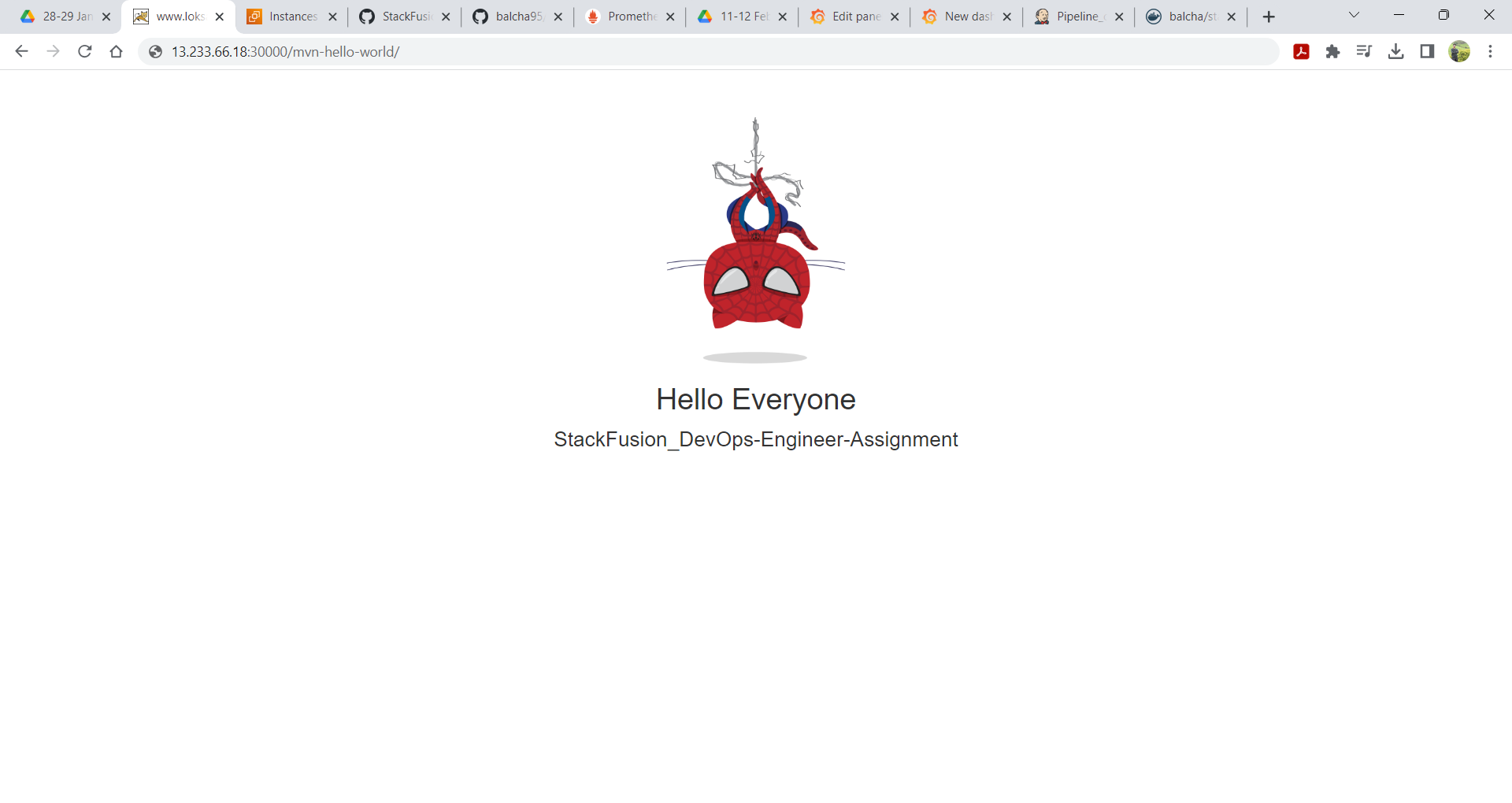
2.



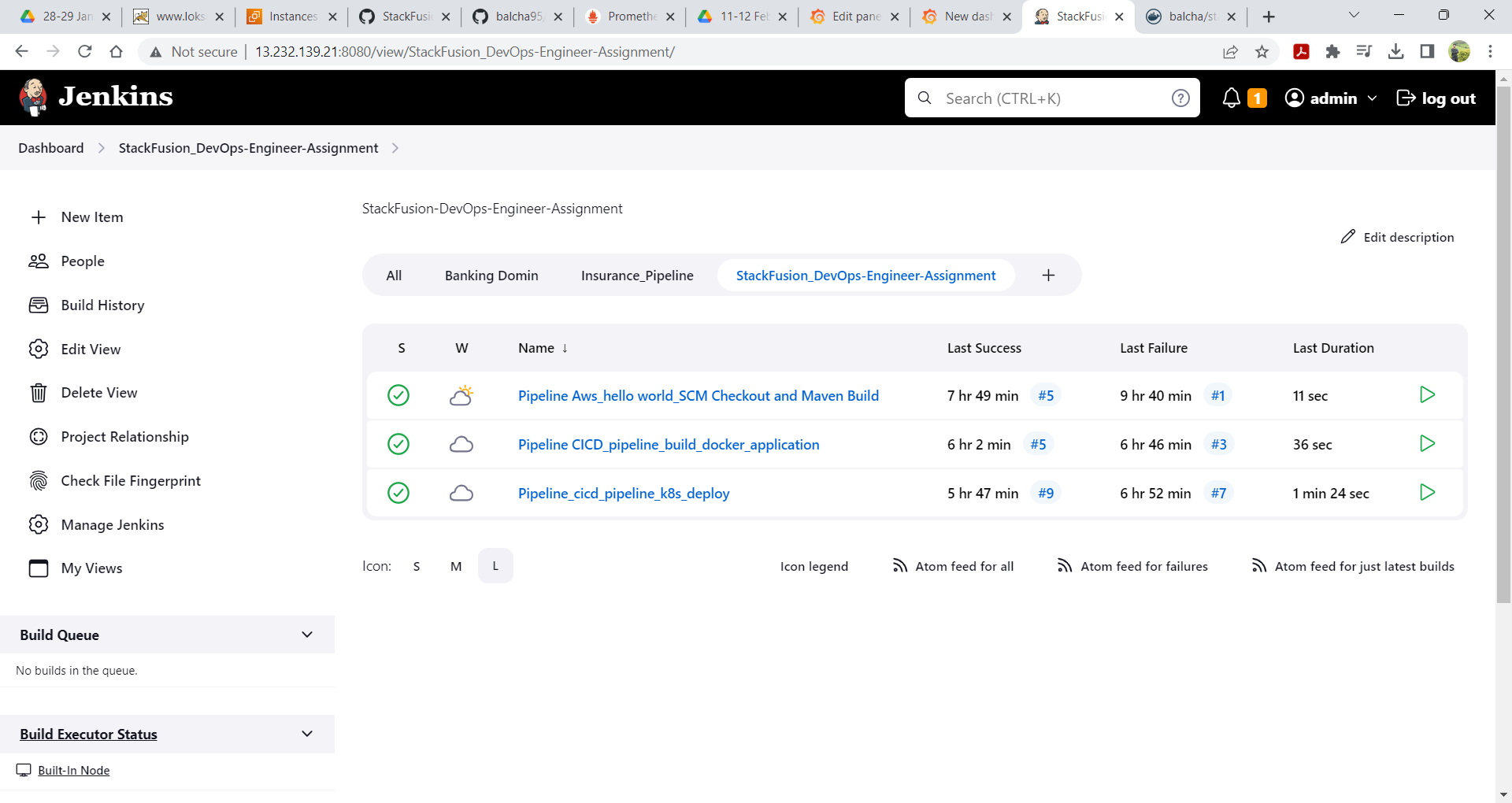
3.



4.



4.1



**Task 3: Monitoring and Logging**

Implement a basic monitoring and logging solution using a tool of your choice (e.g., Prometheus, Grafana, ELK Stack).

The solution should include the following components:

1. Monitor the web server created in Task 1 and collect relevant metrics (e.g., CPU usage, memory usage,

response time).

2. Visualize the collected metrics in a dashboard.

3. Set up logging to capture application logs and store them centrally for easy access and analysis.

**NOTE:** Provide all the necessary configuration files and scripts required to set up the monitoring and logging solution.

stage5 – **Monitor the servers using Monitoring Tools.**

configured to display a dashboard with following metrics.

1. CPU utilization

2. Disk Space Utilization

3. Total Available Memory

4. Network Incoming Traffic

**Test in Prometheus-server query**

**https://prometheus.io/docs/prometheus/latest/querying/examples/**

**CPU Usage**:

Average CPU usage across all cores:

avg(rate(node\_cpu\_seconds\_total{mode="idle"}[5m])) \* 100

Sum by(mode)(irate(node\_cpu\_seconds\_total{mode!="idle"}[5m])) > 0

**Memory Usage:**

Total memory usage:

node\_memory\_MemTotal\_bytes - node\_memory\_MemAvailable\_bytes

**Network Traffic:**

Incoming network traffic:

sum(rate(node\_network\_receive\_bytes\_total[5m])) by (instance)

Outgoing network traffic:

sum(rate(node\_network\_transmit\_bytes\_total[5m])) by (instance)

**Request Count:**

Total number of HTTP requests:

sum(rate(http\_requests\_total[5m])) by (job)

**Response Time:**

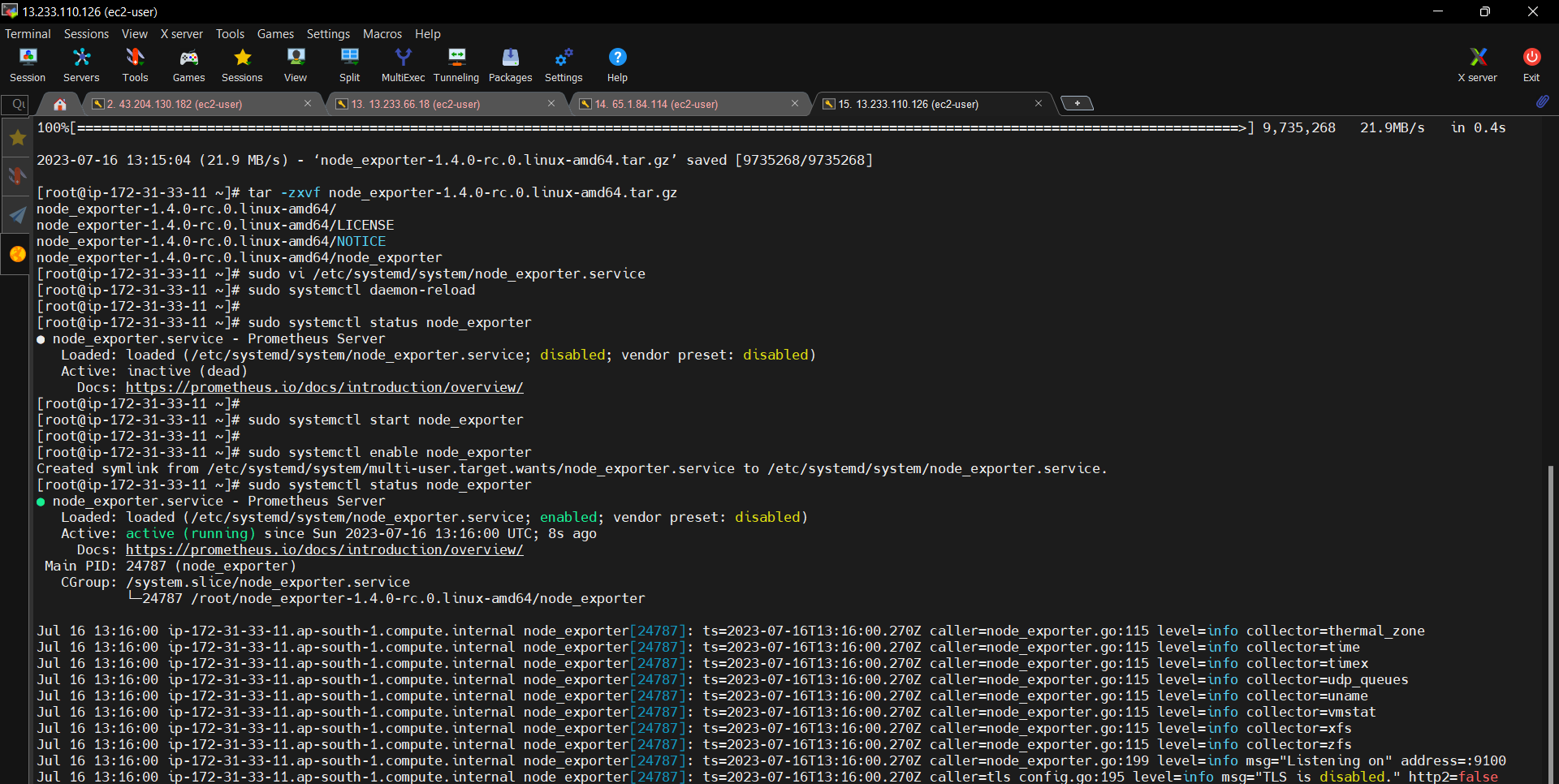
Average response time:

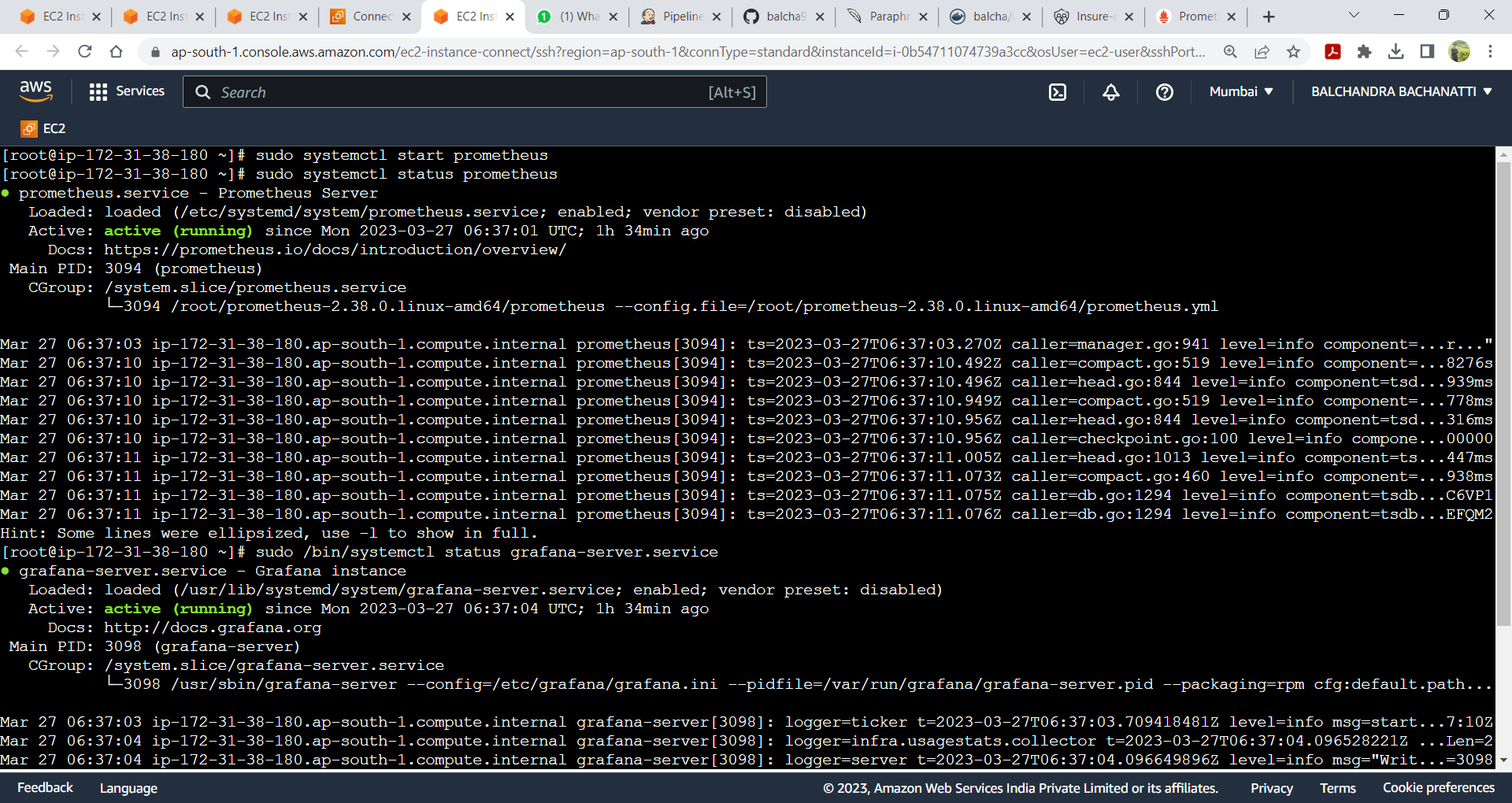
avg(rate(http\_request\_duration\_seconds\_sum[5m])) / avg(rate(http\_request\_duration\_seconds\_count[5m]))

Error Rate:

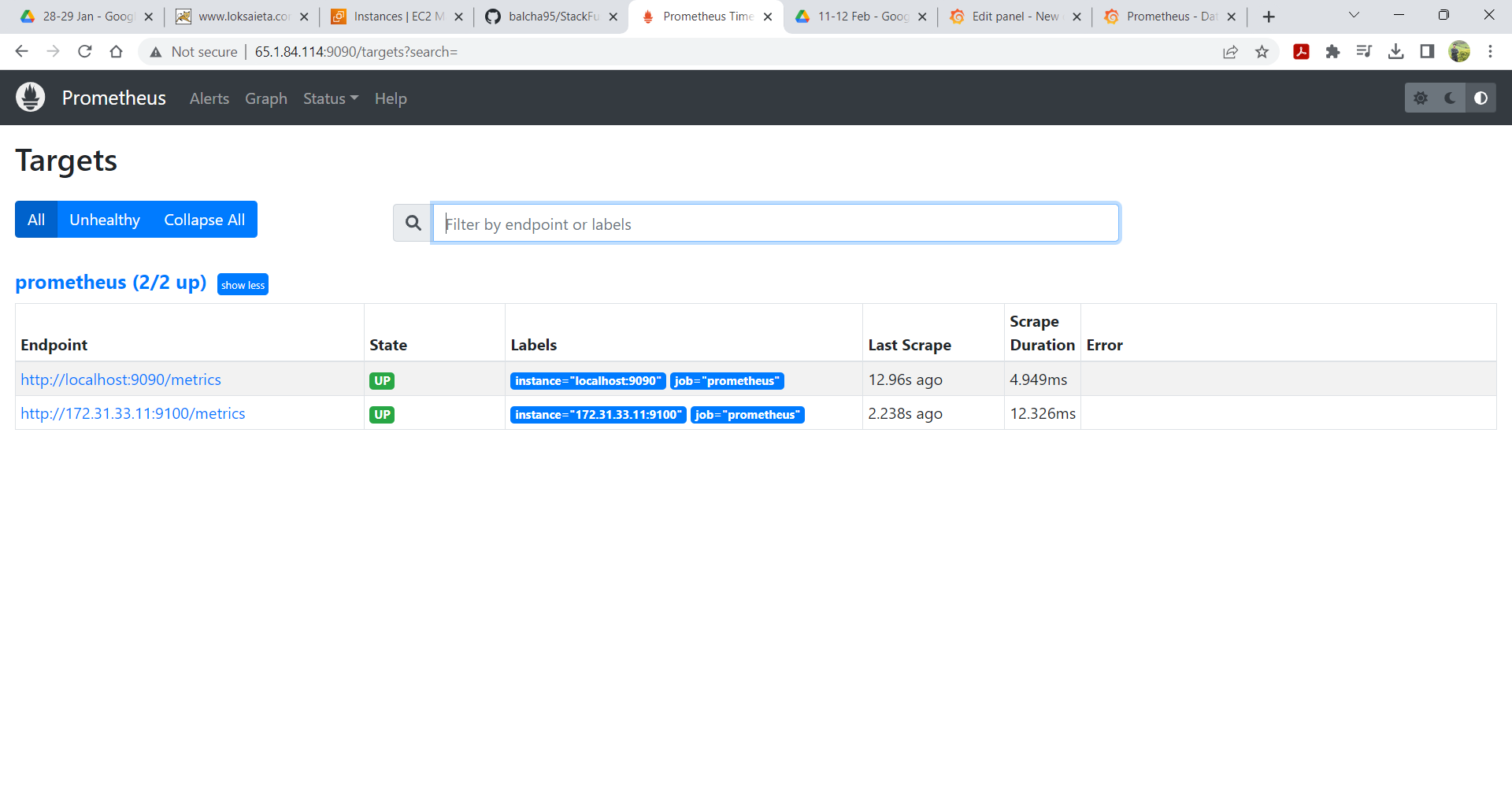
Total number of HTTP errors:

sum(rate(http\_requests\_total{status=~"4..|5.."}[5m])) b

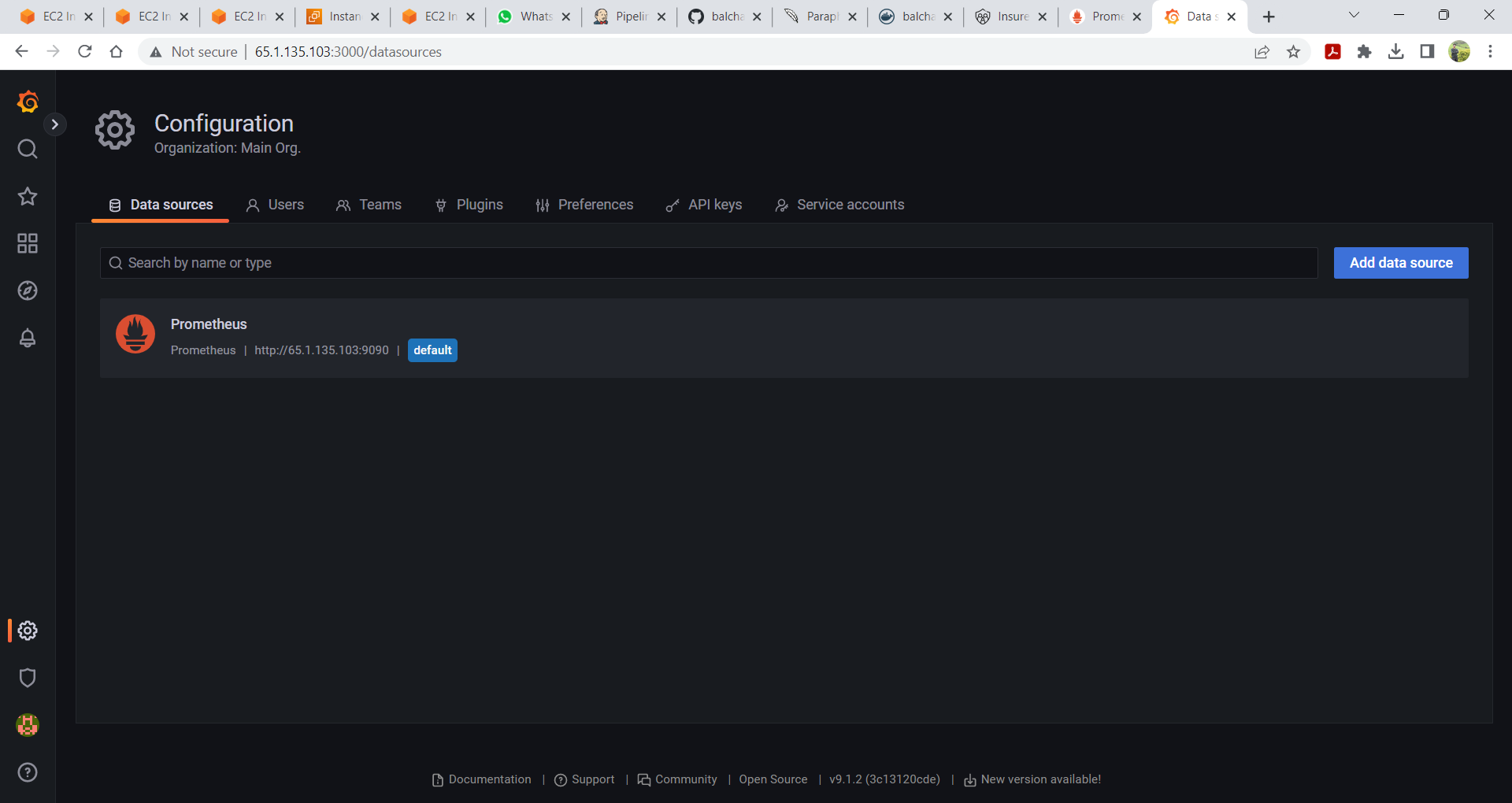
1.

2.

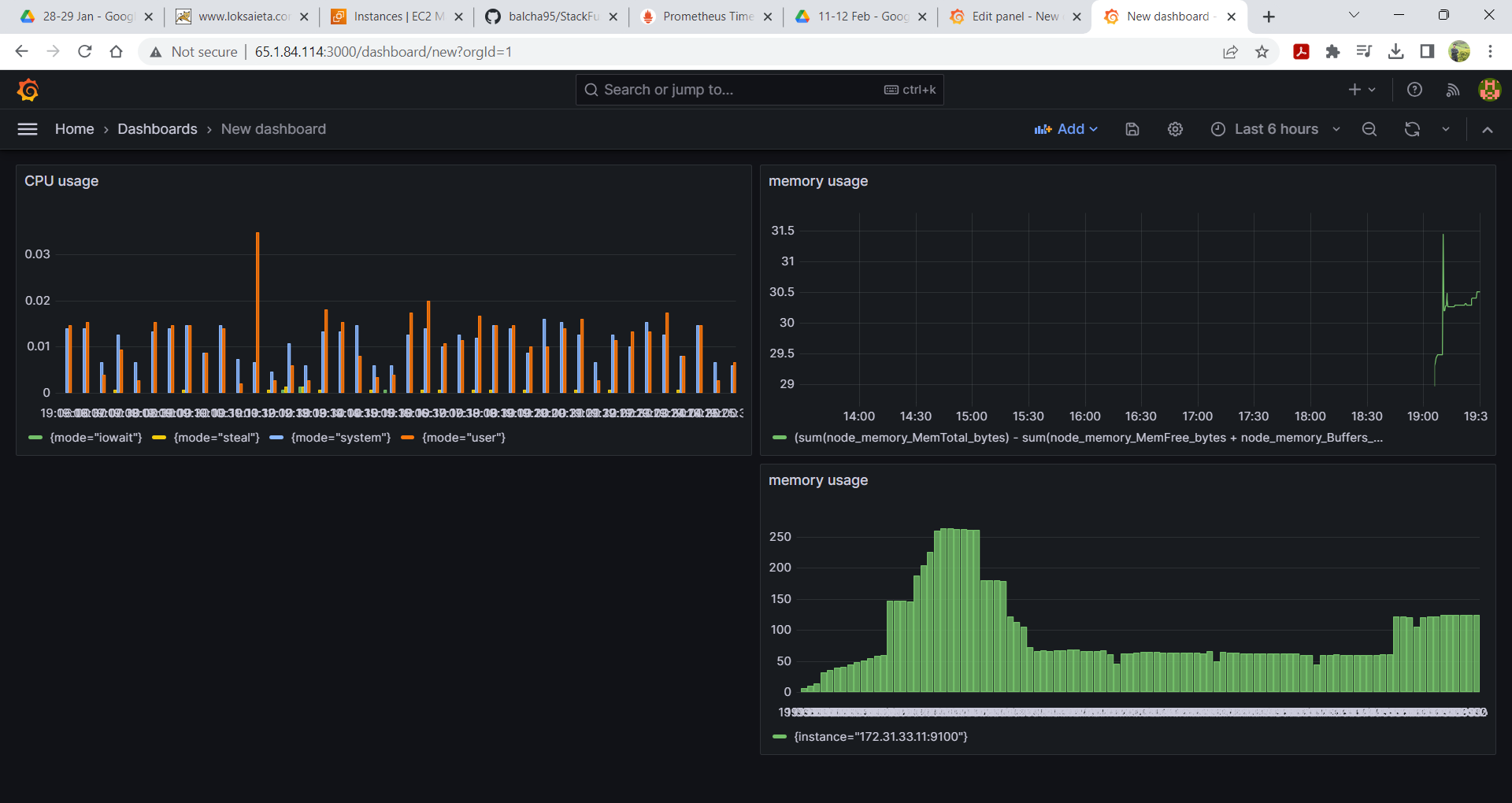
3.



4.



5.



THANKS&REGARD.

BALCHANDRA BACHANATTI.