

## **Section 1.0**

git token: ghp\_55kVuROMhEkuz2I1A83tgtHE2yjcxc0HFC6d

sudo apt install openjdk-jdk-11

sudo apt install maven

for docker installation: <https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-20-04>

run this command for skipping unit test in maven

mvn clean install -DskipTests

for jenkins installation:

<https://www.jenkins.io/doc/book/installing/docker/>

password123 ---jenkins password

### **1.1 CI Pipeline**

To install Jenkins on Ubuntu, you can follow these steps:

Step 1: Update System Packages

sudo apt update

Step 2: Install Java Development Kit (JDK)

Jenkins requires Java to run, so you'll need to install the JDK.

sudo apt install openjdk-11-jdk

Step 3: Add Jenkins Repository

Import the GPG key used to sign Jenkins packages:

```
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -
```

Add the Jenkins repository to your system:

```
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'
```

Step 4: Install Jenkins

Update the package list again:

```
sudo apt update
```

Install Jenkins:

Copy code

```
sudo apt install jenkins
```

Step 5: Start Jenkins

Start the Jenkins service:

Copy code

```
sudo systemctl start jenkins
```

Enable the Jenkins service to start on boot:

Copy code

```
sudo systemctl enable jenkins
```

Step 6: Access Jenkins Web Interface

Jenkins runs on port 8080 by default. Open your web browser and visit:

-----

create jenkins freestyle project with required plugins

add github webhook

-----

install docker-compse

## **Section 2.0 DevOps process:**

When defining a Software Development Life Cycle (SDLC) for a Software-as-a-Service (SaaS) application, the following steps can be considered:

### 1. Requirements Gathering and Analysis:

- Define and document the functional and non-functional requirements of the SaaS application.
- Use tools like Jira, Trello, or Asana to track and manage requirements.

### 2. System Design:

- Design the architecture and components of the SaaS application.
- Use tools like Lucidchart, draw.io, or UML tools to create system diagrams and design documentation.

### 3. Development:

- Implement the SaaS application based on the defined requirements and design.
- Use programming languages like Java, Python, or JavaScript.
- Utilize integrated development environments (IDEs) like IntelliJ IDEA, PyCharm, or Visual Studio Code.

### 4. Testing:

- Perform various types of testing, including unit testing, integration testing, system testing, and acceptance testing.
- Use testing frameworks like JUnit, pytest, or Jasmine for unit testing.
- Employ tools like Selenium, Cypress, or Puppeteer for automated browser testing.
- Use load testing tools like Apache JMeter or Gatling for performance testing.

#### 5. Deployment:

- Prepare the SaaS application for deployment to production environments.
- Utilize tools like Docker or Kubernetes for containerization and orchestration.
- Employ CI/CD tools like Jenkins, GitLab CI/CD, or CircleCI for automated builds, testing, and deployment.

#### 6. Monitoring and Maintenance:

- Set up monitoring and logging systems to track the performance and health of the SaaS application.
- Utilize tools like Prometheus, Grafana, or ELK Stack (Elasticsearch, Logstash, Kibana) for monitoring and logging.

#### 7. Customer Support and Feedback:

- Establish channels for customer support and feedback.
- Utilize tools like Zendesk, Intercom, or Freshdesk for managing customer support tickets and interactions.

#### 8. Continuous Improvement:

- Continuously gather user feedback and analyze usage data to identify areas of improvement.
- Employ tools like Google Analytics, Mixpanel, or Hotjar for user analytics and behavior tracking.

Note that the specific tools mentioned are just examples, and there are numerous alternatives available for each step depending on individual preferences and requirements.

### **Section 3.0 - AWS & Solution Drafting**

To address the team's pain point of searching and reading logs in an aggregated fashion, here's a first draft of the architecture and a high-level checklist for migrating the current setup to use the AWS ElasticSearch (ES) domain.

#### Architecture:

1. Set up an AWS ElasticSearch domain in the main AWS account to serve as the centralized log storage and search engine.

2. Configure the following components to forward logs to the ElasticSearch domain:

- RabbitMQ on ECS Fargate: Configure RabbitMQ to publish logs to a designated CloudWatch log group.
- Java REST API on ECS Fargate: Configure the REST API to log directly to the designated CloudWatch log group.
- Java microservices on ECS Fargate: Configure each microservice to log to their respective designated CloudWatch log groups.
- EC2 ASG running microservice: Update the CloudWatch log agent configuration to forward logs to the designated CloudWatch log group.

3. Set up Log Groups and Log Streams in CloudWatch to organize the logs from different components.

4. Configure CloudWatch Log Subscriptions for each component to stream logs to the ElasticSearch domain in the main account.

5. Create a new Kibana instance in the main account to provide a user-friendly interface for searching and visualizing logs from the ElasticSearch domain.

Checklist:

1. Set up an AWS ElasticSearch domain in the main AWS account.

2. Configure log forwarding for RabbitMQ on ECS Fargate to publish logs to a designated CloudWatch log group.

3. Configure log forwarding for the Java REST API on ECS Fargate to log directly to the designated CloudWatch log group.

4. Configure log forwarding for each Java microservice on ECS Fargate to log to their respective designated CloudWatch log groups.

5. Update the CloudWatch log agent configuration on the EC2 ASG running microservice to forward logs to the designated CloudWatch log group.

6. Set up Log Groups and Log Streams in CloudWatch to organize the logs.

7. Configure CloudWatch Log Subscriptions for each component to stream logs to the ElasticSearch domain in the main account.

8. Create a Kibana instance in the main account and configure it to connect to the ElasticSearch domain.

9. Test the log forwarding and ensure logs from all components are successfully reaching the ElasticSearch domain.

10. Conduct thorough testing of log search and visualization using Kibana to ensure it meets the team's requirements.

11. Document the architecture, configurations, and steps taken for future reference and knowledge sharing within the team.

12. Conduct a thorough review with the team, considering their feedback and suggestions for further improvements.