

CI/CD Pipeline Project - ASI Insurance

28.02.2024

—

Akshay Imanuel N

Bangalore,
India

Objective

To create a microservice application architecture for an Insurance company through DevOps pipeline and deployment on Docker.

Tools used:

1. Jenkins
2. Github
3. Docker and Docker Hub
4. AWS

Description

ASI Insurance is facing challenges in improving the SLA to its customers due to its organizational growth and existing monolithic application architecture. It requires transformation of the existing architecture to a microservice application architecture, while also implementing DevOps pipeline and automations.

Tasks

1. Create the Dockerfile, Jenkinsfile, Ansible playbook, and the source file of the static website
2. Upload all the created files to GitHub
3. Go to the terminal and install NodeJS 16
4. Open the browser and access the Jenkins application
5. Create Jenkins pipeline to perform CI/CD for a Docker container
6. Create Docker Hub Credentials and other necessary pre-requisites before running build
7. Set up Docker remote host on AWS and configure deploy stage in pipeline
8. Execute Jenkins Build
9. Access deployed application on Docker container

1. Creating Github repository and Required Files:

I have created a new github repository, where I have stored the source files of the application.

<https://github.com/akshay-immanuel/Dockerised-CICD-pipeline.git>

I have created the Dockerfile, Jenkins file and Ansible playbook file also in the git hub

2. Install nodejs

To install nodejs use the below command

```
sudo apt install nodejs -y
```

```
labsuser@master:~$ sudo apt install nodejs -y
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  javascript-common libjs-highlight.js libnode72 nodejs-doc
Suggested packages:
  apache2 | lighttpd | httpd npm
The following NEW packages will be installed:
  javascript-common libjs-highlight.js libnode72 nodejs nodejs-doc
0 upgraded, 5 newly installed, 0 to remove and 86 not upgraded.
Need to get 13.7 MB of archives.
```

Verify the installation using

```
node -v
```

```
labsuser@master:~$ node -v
v12.22.9
labsuser@master:~$ █
```

3. Install and Access Jenkins Application

Java is a prerequisite for Jenkins to be installed, hence we will install java first followed by jenkins itself.

```
sudo apt install openjdk-17-jre-headless
```

```
labsuser@master:~$ sudo apt install openjdk-17-jre-headless
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following additional packages will be installed:
  ca-certificates-java java-common
Suggested packages:
  default-jre fonts-dejavu-extra fonts-ipafont-gothic fonts-ipafont-mincho fonts-wqy-microhei
The following NEW packages will be installed:
```

Verify the installation

java -version

```
labsuser@master:~$ java --version
openjdk 17.0.10 2024-01-16
OpenJDK Runtime Environment (build 17.0.10+7-Ubuntu-122.04.1)
OpenJDK 64-Bit Server VM (build 17.0.10+7-Ubuntu-122.04.1, mixed mode, sharing)
labsuser@master:~$
```

Installing jenkins: Use the code below to install jenkins

```
sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
```

```
labsuser@master:~$ sudo wget -O /usr/share/keyrings/jenkins-keyring.asc \
  https://pkg.jenkins.io/debian-stable/jenkins.io-2023.key
echo deb [signed-by=/usr/share/keyrings/jenkins-keyring.asc] \
  https://pkg.jenkins.io/debian-stable binary/ | sudo tee \
  /etc/apt/sources.list.d/jenkins.list > /dev/null
sudo apt-get update
sudo apt-get install jenkins
```

Verify the installation:

jenkins - -version

Sudo systemctl status jenkins

```
labuser@master:~$ jenkins --version
2.448.1
labuser@master:~$ sudo systemctl status jenkins
● jenkins.service - Jenkins Continuous Integration Server
   Loaded: loaded (/lib/systemd/system/jenkins.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2024-02-28 11:18:29 UTC; 4min 13s ago
       Main PID: 16896 (java)
          Tasks: 46 (limit: 4598)
         Memory: 1.826s
        CPU: 1m11.826s
      CGroup: /system.slice/jenkins.service
              └─16896 /usr/bin/java -Djava.awt.headless=true -jar /usr/share/java/jenkins.war --webroot=/var/cache/jenkins/war --httpPort=8080

Feb 28 11:17:59 master jenkins[16896]: b5f7d4961f741cea2bdd11c5833de6d
Feb 28 11:17:59 master jenkins[16896]: This may also be found at: /var/lib/jenkins/secrets/initialAdminPassword
Feb 28 11:17:59 master jenkins[16896]: ****
Feb 28 11:18:29 master jenkins[16896]: 2024-02-28 11:18:29.246+0000 [id:32] INFO jenkins.InitReactorRunner$1#onAttained: Completed initialization
Feb 28 11:18:29 master jenkins[16896]: 2024-02-28 11:18:29.274+0000 [id:24] INFO hudson.lifecycle.Lifecycle$onReady: Jenkins is fully up and running
Feb 28 11:18:29 master systemd[1]: Started Jenkins Continuous Integration Server.
Feb 28 11:18:29 master jenkins[16896]: 2024-02-28 11:18:29.556+0000 [id:49] INFO h.m.DownloadService$Downloadable#load: Obtained the updated data file for hudson.tasks.M
Feb 28 11:18:29 master jenkins[16896]: 2024-02-28 11:18:29.557+0000 [id:49] INFO hudson.util.Retriger#start: Performed the action check updates server successfully at the
Lines 1-20/20 (END)
```

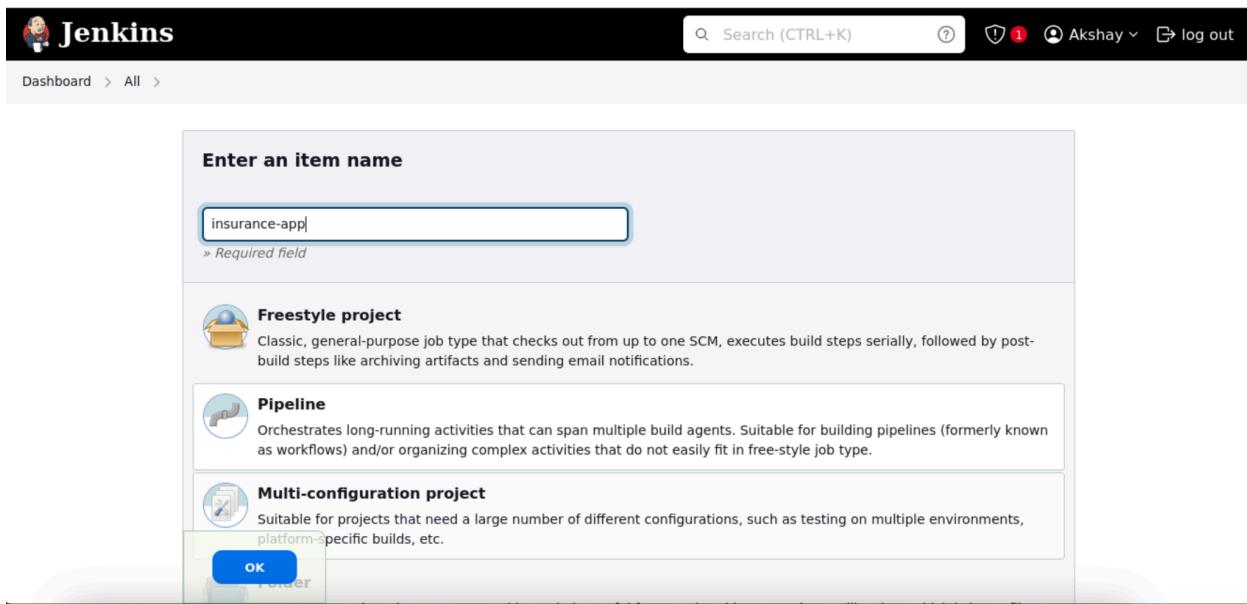
Access Jenkins using the public IP and the port 8080, as seen below. And enter the token

The screenshot shows the Jenkins dashboard. On the left, there's a sidebar with links: 'New Item', 'People', 'Build History', 'Manage Jenkins', and 'My Views'. Below these are two collapsed sections: 'Build Queue' (empty) and 'Build Executor Status' (showing 1 Idle and 2 Idle). The main content area has a title 'Welcome to Jenkins!' with a sub-instruction: 'This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.' It features a 'Start building your software project' button and several cards: 'Create a job' (with a '+' icon), 'Set up a distributed build' (with a 'Set up an agent' and 'Configure a cloud' option), and 'Learn more about distributed builds' (with a help icon).

4. Create Jenkins Pipeline

First we will create a test pipeline and we will build each step one by one and in this process we will update all our required files in Github and in the end we will make the necessary changes to run the complete pipeline for the actual app.

Lets create a pipeline:



The screenshot shows the Jenkins interface with a search bar at the top containing the text 'Search (CTRL+K)'. Below it, there are user icons for 'Akshay' and a 'log out' button. The main area is titled 'Enter an item name' with a required field input box containing 'insurance-app'. Below this, there are three project type options: 'Freestyle project' (selected), 'Pipeline', and 'Multi-configuration project'. Each option has a brief description and an 'OK' button at the bottom right. The 'Freestyle project' description states: 'Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.' The 'Pipeline' description states: 'Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.' The 'Multi-configuration project' description states: 'Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.'

Once Clicked on Ok you will be navigated to Jenkins job configuration page where we can provide Jenkins job details such as parameters, Jenkins pipeline configuration etc.

Enter the github repo link and your default branch, and give the Jenkinsfile name in the repo

The screenshot shows the Jenkins Pipeline configuration page for the 'Insurance-app' project. The pipeline definition is set to 'Pipeline script from SCM'. Under the 'SCM' section, 'Git' is selected. The 'Repository URL' is set to `https://github.com/akshay-immanuel/Dockerised-CICD-pipeline.git`. The 'Credentials' dropdown is currently empty ('- none -'). There is also an 'Advanced' button. At the bottom are 'Save' and 'Apply' buttons.

The screenshot shows the Jenkins Pipeline configuration page for the 'Insurance-app' project. The pipeline definition is set to 'Pipeline script from SCM'. Under the 'Branch Specifier' section, the value is `*/master`. The 'Repository browser' is set to '(Auto)'. In the 'Additional Behaviours' section, there is an 'Add' button. The 'Script Path' is set to `Jenkinsfile`. The 'Lightweight checkout' checkbox is checked. At the bottom are 'Save' and 'Apply' buttons. The footer indicates REST API and Jenkins 2.440.1.

1. Add dependency maven tool

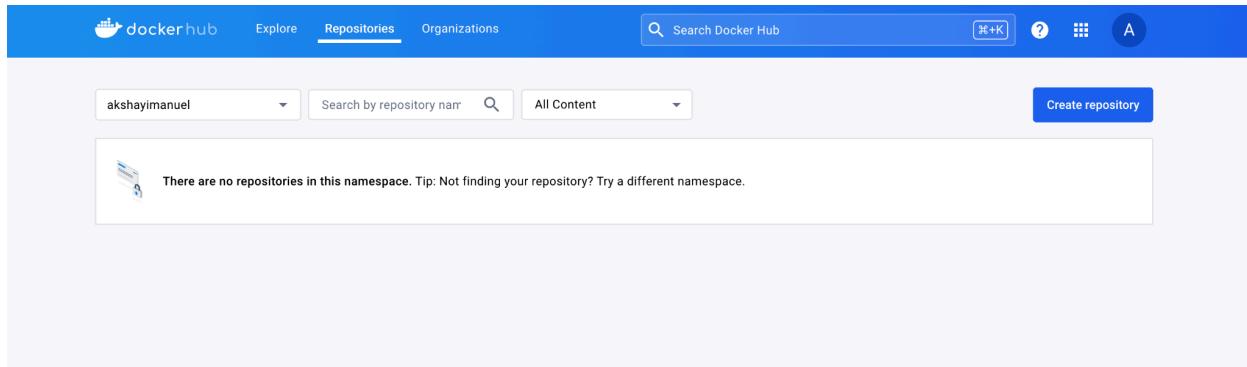
The screenshot shows the Jenkins 'Manage Jenkins' interface under the 'Tools' section. The 'Maven installations' page is displayed. A new Maven installation is being configured with the name 'maven'. The 'Install automatically' checkbox is checked. Under the 'Install from Apache' section, the version '3.9.6' is selected. There is an 'Add Installer' button available. At the bottom are 'Save' and 'Apply' buttons.

2. Install the reports plugin

The screenshot shows the Jenkins 'Manage Jenkins' interface under the 'Plugins' section. The 'Installed plugins' tab is selected. A search bar at the top contains the text 'reports'. A single plugin, 'HTML Publisher plugin 1.32', is listed. It is described as publishing HTML reports and has a 'Report an issue with this plugin' link. The plugin status is 'Enabled' with a green toggle switch. There are other tabs like 'Updates', 'Available plugins', and 'Advanced settings' visible on the left.

5. Create Docker Hub Credentials and other necessary prerequisites before running build.

Once Jenkins job configuration are done, we have to create a Docker credential which will be used to push Docker image to Docker hub



Once a Docker hub account is created, navigate to Manage Jenkins à Manage Credentials and then select the global domain.

A screenshot of the Jenkins Manage Credentials page. The top navigation bar includes the Jenkins logo, a search bar, and a user dropdown for 'Akshay'. Below the navigation is a breadcrumb trail: 'Dashboard > Manage Jenkins > Credentials'. The main content area has a title 'Credentials' with a subtitle 'Stores scoped to Jenkins'. There is a table with columns: 'T', 'P', 'Store ↓', 'Domain', 'ID', and 'Name'. Under 'Domain', it says '(global)'. Below the table, there's a section titled 'Stores scoped to Jenkins' with a table showing 'System' under 'Domains'. At the bottom left, there are icons for 'S' (small), 'M' (medium), and 'L' (large). At the very bottom left, it says 'Icon:'.

Click on new Credential by clicking on Add Credentials to create a new Docker hub credentials as per below details:

The screenshot shows the Jenkins interface for creating new credentials. The top navigation bar includes links for Dashboard, Manage Jenkins, Credentials, System, and Global credentials (unrestricted). The main title is "New credentials". The "Kind" dropdown is set to "Username with password". The "Scope" dropdown is set to "Global (Jenkins, nodes, items, all child items, etc)". The "Username" field contains "akshayimmanuel". There is an unchecked checkbox for "Treat username as secret". The "Password" field contains a series of asterisks. The "ID" field is filled with "dockerHunAccount". The "Description" field is empty. A blue "Create" button is at the bottom left.

Next provide full access to Docker Sock file using below command:

```
chmod 777 /var/run/docker.sock
```

6. Execute Jenkins Build

1. Verify Jenkins Build success

Stage View

	Prepare Environment	Code Checkout	Maven Build	Publish Test Reports	Docker Image Build	Docker Image Scan	Publishing Image to DockerHub	Docker Container Deployment
Average stage times: (Average full run time: ~23s)	73ms	237ms	12s	56ms	2s	567ms	5s	1s
#2 Feb 28 19:27 No Changes	73ms	237ms	12s	56ms	2s	567ms	5s	1s

Permalinks

- Last build (#2), 4 min 32 sec ago
- Last stable build (#2), 4 min 32 sec ago
- Last successful build (#2), 4 min 32 sec ago
- Last completed build (#2), 4 min 32 sec ago

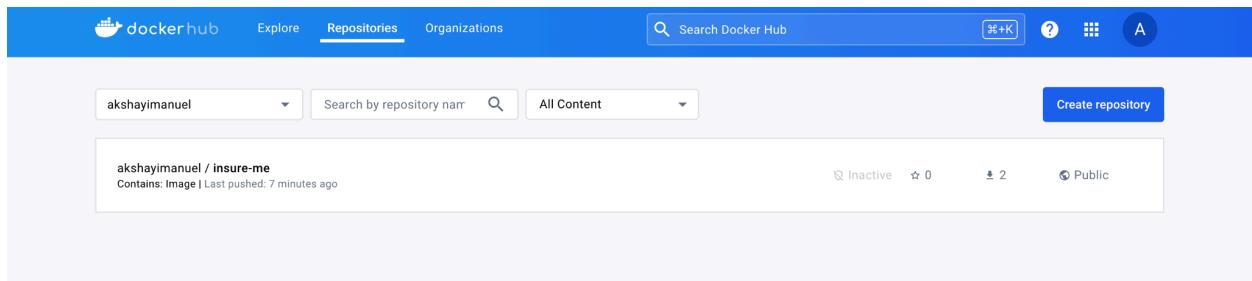
REST API Jenkins 2.440.1

```

9c742cd6c7a5: Mounted from library/openjdk
80a84a0b1ea3: Pushed
3.0: digest: sha256:55ae0c92baa0b15823a21841b2aaa4a7bd4b2d30844f475bd2b493c5609f2f15 size: 2007
[Pipeline] echo
Image push complete
[Pipeline]
[Pipeline] // withCredentials
[Pipeline]
[Pipeline] // stage
[Pipeline] stage
[Pipeline] {
  (Docker Container Deployment)
[Pipeline] sh
+ docker rm insure-me -f
Error response from daemon: No such container: insure-me
[Pipeline] sh
+ docker pull akshaymanuel/insure-me:3.0
3.0: Pulling from akshaymanuel/insure-me
Digest: sha256:55ae0c92baa0b15823a21841b2aaa4a7bd4b2d30844f475bd2b493c5609f2f15
Status: Image is up to date for akshaymanuel/insure-me:3.0
docker.io/akshaymanuel/insure-me:3.0
[Pipeline] sh
+ docker run -d --rm -p 8081:8081 --name insure-me akshaymanuel/insure-me:3.0
2cd781ecdc9e46975fdfbad152d54e8c8392a27c330d8350784fb723023c372
[Pipeline] echo
Application started on port: 8081 (http)
[Pipeline]
[Pipeline] // stage
[Pipeline]
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS

```

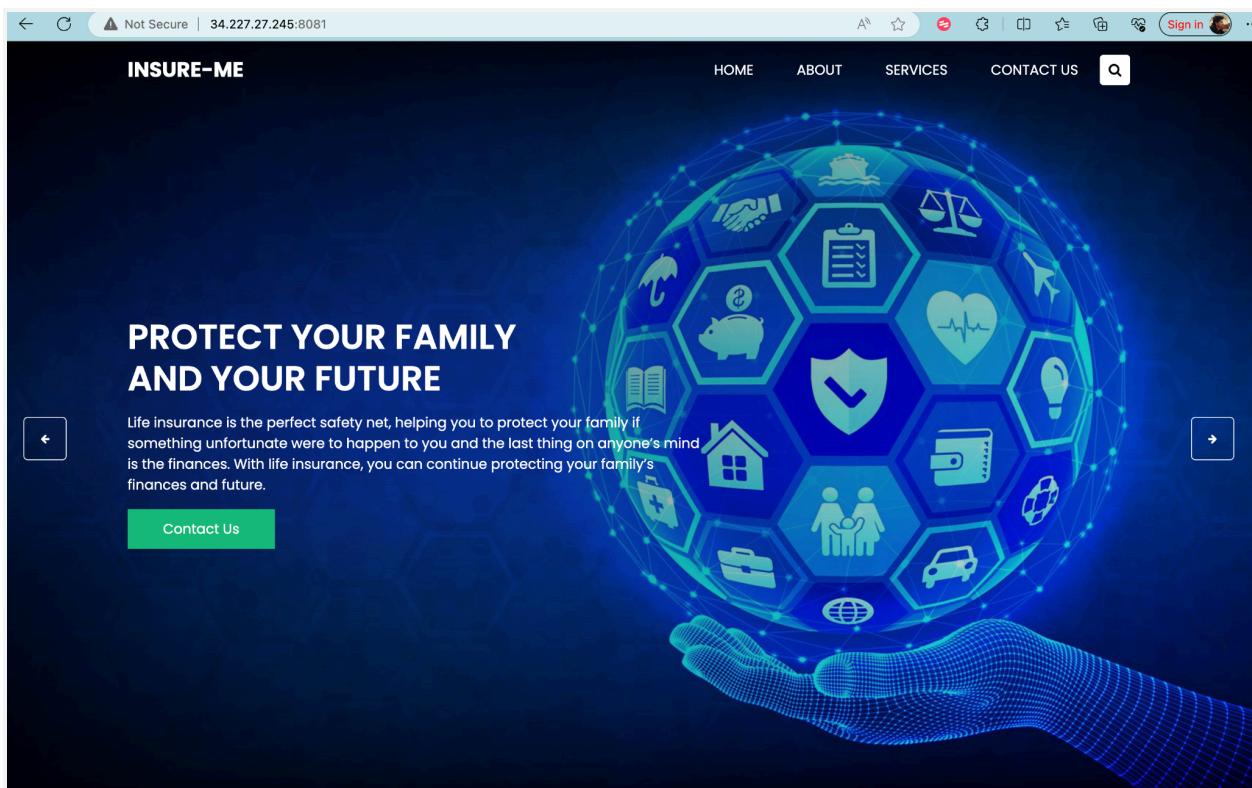
2. Verify the Image push to Docker Hub



3. Verify the application deployment on the docker container

Once Docker container is deployed using server IP address and port 8081 to access application.

[http://34.227.27.245 :8081](http://34.227.27.245:8081)



The screenshot shows a web browser window with the URL 34.227.27.245:8081/service.html. The page has a dark blue header with the logo 'INSURE-ME'. Below the header is a navigation bar with links for HOME, ABOUT, SERVICES, NEWS, and CONTACT US, along with a search icon and a 'Sign in' button.

OUR SERVICES

There are many variations of passages of Lorem Ipsum available, but the majority have suffered alteration



HOME INSURANCE

fact that a reader will be distracted by the readable looking at its layout.

[Read More](#)



HEALTH INSURANCE

fact that a reader will be distracted by the readable looking at its layout.

[Read More](#)



CAR INSURANCE

fact that a reader will be distracted by the readable looking at its layout.

[Read More](#)



LIFE INSURANCE

fact that a reader will be distracted by the readable looking at its layout.

[Read More](#)

[View All](#)

INSUDOR

Soluta odit exercitationem rerum aperiam eos consectetur impedit delectus qui

CONTACT US

Ipsum dolor sit amet consectetur adipisicing elit

SIGN UP TO OUR NEWSLETTER

Enter Your Email

Outcome

The successful completion of the project will enable ASI Insurance

- To improve its overall application deployment process.
- Enhance system scalability.
- Deliver better products and services to its customers.