

# **International Institute of Information Technology, Bangalore**

**Software Production Engineering Project  
AI-on-the-edge**

**Under the Guidance of  
Prof. B. Thangaraju**

**Teaching Assistant- Shubham Agarwal.**



Akshay Nagpal  
MT2020016

Sachin Sharma  
MT2020133

Rahul Modak  
MT2020014

# Table of Contents

## 1. Abstract.

## 2. Introduction.

### 2.1 Overview

### 2.2 Features

### 2.3 Devops

## 3. System Configuration.

## 4. Software Development Lifecycle.

### 4.1 Source Code Management ( SCM )

### 4.2 Build

### 4.3 Docker

### 4.4 Continuous Integration

### 4.5 Deployment using Ansible

### 4.6 Monitoring and Logging

### 4.7 Final Building and deployment

## 5. Experimental Setup.

### 5.1 Use case diagram

### 5.2 Design diagram

## 6. Result and Discussions.

6.1 Home page of the application

6.2 User login

6.3 User Registration page

6.4 File upload page

6.5 Jobs dashboard

6.6 Admin dashboard

## 7. Future scope and Work

# 1.Abstract

With the growth of IoT applications and consumer devices, a constant and significant amount of data is being generated and ingested at the edge. As an example, the edge can consist of a collection of sensors that gather raw data, ranging from a few gigabytes (GB) to a few terabytes (TB) a day, depending on the application.

Moving this volume of raw data from thousands of edge locations over the network at scale is impractical and is prone to performance issues. Shifting analytics processing and inference at the edge is an ideal way to reduce data movement and to provide faster results. In such advanced solution cases, the data that is fed into the data group is being operated upon from an analytics and an AI/DL perspective, and the trained AI/DL models are pushed back to the edge for real-time inferencing.

Artificial Intelligence (AI) has emerged as one of the most powerful forces in the digital transformation. We believe developers, data scientists and enterprises should have easy access to the power of AI so they can build systems that augment human ingenuity in unique and differentiated ways. In this project, we would be aiming at the development of an integrated, mutually reinforcing structure (Platform) that carry value-laden data from the various networked "things"(like sensors, smart devices) to production and traditional IT systems to deliver actionable business insights.

## 2. Introduction

### 2.1 Overview

Today we are in so called “Technology era” in which all kinds of different applications are being developed, one such are IOT applications more particularly applications that uses this large amount of data collected from these various iot devices for analytics and inferring. In this project we aim to develop a platform which will allow users to deploy above mentioned kind of applications called “AI on edge” applications.

### 2.2 Features

- Load balancer
- Allows users to register their machines as server nodes
- Scheduling
- Allows user to deploy applications and run specific jobs(ML models)
- Allow users to download prediction file(.csv)

### 2.3 Devops

We plan to build this project in an incremental manner. The design consists of views and services which function fairly independently. Given the complexity of the project, it's not feasible for one of us to build and test all the code manually every time we make a small change. Also, three of us working from different locations, an automated pipeline will not only make our work easier but also makes it more efficient. The amount of communication that has to happen between us will reduce. DevOps enables us to focus on important aspects of the project, improves efficiency, stability and security. There's less scope for manual errors. Also since we plan to ship this product at some point, continuous delivery makes it easy. Also, monitoring allows us to understand the usage better and helps us improve the application.

### 3. System Configuration

NAME	SPECS
OS - Ubuntu	20.04 LTS
CPU	Intel Core I5 (8 cores)
RAM	8GB
Kernel version	Linux 4.15.0-101 generic
Languages used	Java,Python, HTML, CSS, javascript and XML
Other dependencies	Rabbitmq,tomcat,apscheduler

## 4. Software Development Life Cycle (SDLC)

SDLC is a process of making a project in an organisation. It specifies the steps needed to be taken from Requirement analysis of the project to the monitoring or maintenance of the application. In this section, we will illustrate the steps taken by us analogous to DevOps SDLC, to develop the application.

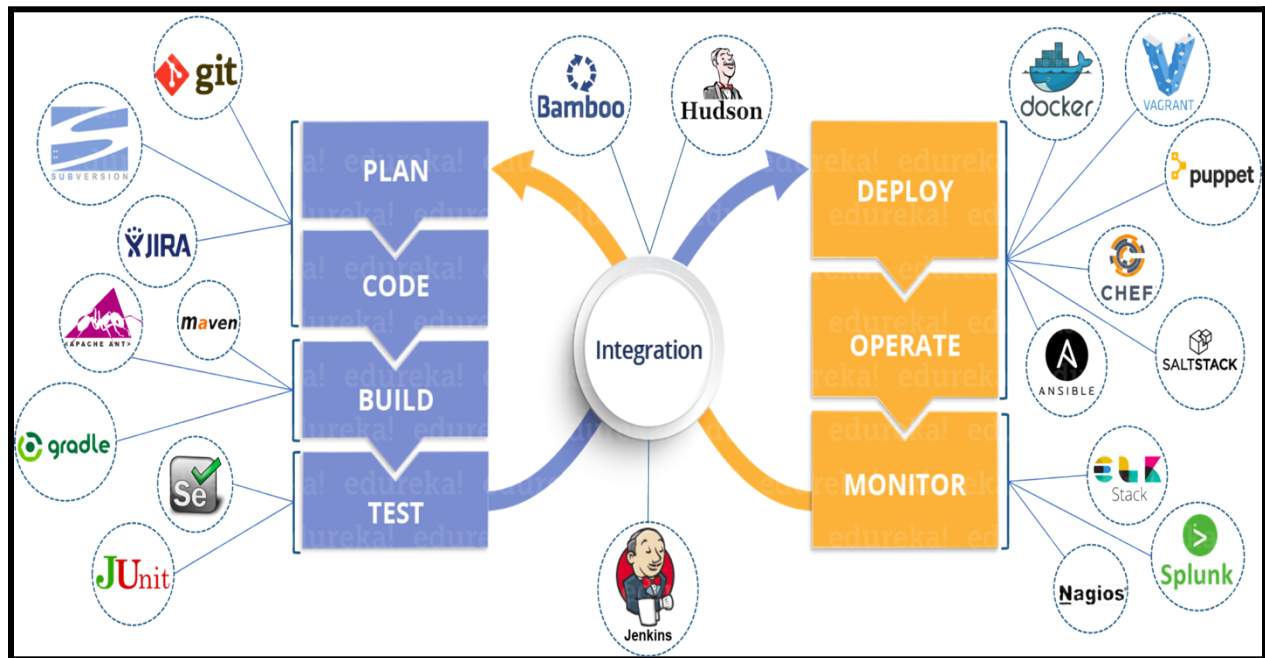


Fig:[1] SDLC Life Cycle

Source: <https://qph.fs.quoracdn.net/+main-qimg-5249df982a9484aa13101391add68e78>

In this section, we will explain the installation procedure of the softwares and libraries used in the project, followed by the Source Code Managements and building the project. It will also include Testing, artifacts, deployment, monitoring and building a CI/CD pipeline using DevOps tools.

### 4.1 Source Code Management ( SCM ):



<https://github.com/akshay-nagpal/AI-on-the-edge/>

SCM is a tool used to manage source code. It can track revisions in the source code with timestamps and the details of who is responsible for the change. We can do different operations like the store, merge and compare revisions.

We used Git as SCM for this project.






















Why Git?

- It is a distributed version control system that is easy to manage.
- Easy to track changes in any file of the project.
- It allows us to revert the code files back to their previous state.

The list of commands used for code management are:

```
$ git init
$ git add -all
$ git commit -m "commit message"
$ git push
```



 <b>akshay-nagpal</b> rectified download feature		1913ed4 1 hour ago	🕒 63 commits
	.idea	rectified download feature	1 hour ago
	lib	ADDED: ansible files added	2 days ago
	src/main	rectified download feature	1 hour ago
	192.168.29.49.txt	ADDED:Download	18 days ago
	192.168.43.220.txt	updated	last month
	192.168.43.49.txt	updated	last month
	AI-on-the-edge.iml	path updated	2 days ago
	Dockerfile	rectified dockerfile	4 hours ago
	README.md	Initial commit	2 months ago
	configure.py	rectified paths	2 days ago
	deploy.yml	ADDED: ansible files added	2 days ago
	initialize.py	path updated	2 days ago
	inventory	ADDED: ansible files added	2 days ago
	load.py	rectified paths	2 days ago
	platform_logstash.conf	rectified download feature	1 hour ago
	pom.xml	ADDED:logging	17 days ago
	readdb.py	calculating score	last month
	recv.py	rectified rabbitmq	2 days ago
	send.py	rectified paths	2 days ago
	test.py	added load_balancer	last month

the description, license, or topics provided.

 Readme





## Releases

No releases published  
[Create a new release](#)

## Packages

No packages published  
[Publish your first package](#)













## Contributors 4

-  **akshay-nagpal** Akshay Nagpal
-  **rahul166** Rahul Modak
-  **Schnshrma** Sachin sharma
-  **shubhamaggarwal890** Shubham A...










## Languages



### Commits on Apr 27, 2021

<b>ADDED: log4j2.xml</b>  <b>akshay-nagpal</b> committed 19 days ago	 <a href="#">ed139e7</a>	
<b>ADDED:logging</b>  <b>rahul166</b> committed 19 days ago	 <a href="#">1f97504</a>	
<b>ADDED: Dockerfile</b>  <b>akshay-nagpal</b> committed 20 days ago	 <a href="#">beb7889</a>	
<b>ADDED:Download</b>  <b>rahul166</b> committed 20 days ago	 <a href="#">ca34683</a>	

### Commits on Apr 15, 2021

<b>ADDED: Application_name persistence</b>  <b>akshay-nagpal</b> committed on 15 Apr	 <a href="#">d238c63</a>	
<b>ADDED: Application_name persistence</b>  <b>akshay-nagpal</b> committed on 15 Apr	 <a href="#">0cb6479</a>	
<b>ADDED:Run job</b>  <b>rahul166</b> committed on 15 Apr	 <a href="#">8be931c</a>	

### Commits on Apr 14, 2021













<b>added user dashboard</b>  <b>akshay-nagpal</b> committed on 14 Apr	 <a href="#">5a9a657</a>	
<b>updated</b>  <b>Schnshrma</b> committed on 14 Apr	 <a href="#">c9f3fc8</a>	
<b>updated</b>  <b>Schnshrma</b> committed on 14 Apr	 <a href="#">5260cb2</a>	
<b>ADDED:Jobcontroller</b>  <b>rahul166</b> committed on 14 Apr	 <a href="#">9c6ef37</a>	

Fig:[2] Git source code and some commit history

## 4.2 Build

Maven is a project management tool. It is usually used in Java-based projects. It is based on the Project Object Model (POM). We used Maven as a build automation tool in our project as It can manage a project's build, reporting and documentation from the pom file.

Maven uses Convention over Configuration, which means developers are not required to create the build process themselves. An XML file(POM.xml) is used to define the Maven project structure referred to as Project Object Model (POM).

We used the following command to build the maven project:

\$ mvn build - At the end this command will be mentioned in CI pipeline. See image below

```
[INFO] Building AI-on-the-edge 1.0-SNAPSHOT
[INFO] -----[ war ]-----
[INFO]
[INFO] --- maven-resources-plugin:2.6:resources (default-resources) @ AI-on-the-edge ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] Copying 3 resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ AI-on-the-edge ---
[INFO] Changes detected - recompiling the module!
[WARNING] File encoding has not been set, using platform encoding UTF-8, i.e. build is platform dependent!
[INFO] Compiling 20 source files to /mnt/nfs_share/newfolder/AI-on-the-edge/target/classes
[INFO]
[INFO] --- maven-resources-plugin:2.6:testResources (default-testResources) @ AI-on-the-edge ---
[WARNING] Using platform encoding (UTF-8 actually) to copy filtered resources, i.e. build is platform dependent!
[INFO] skip non existing resourceDirectory /mnt/nfs_share/newfolder/AI-on-the-edge/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ AI-on-the-edge ---
[INFO] No sources to compile
[INFO]
[INFO] --- maven-surefire-plugin:2.12.4:test (default-test) @ AI-on-the-edge ---
[INFO] No tests to run.
[INFO]
[INFO] --- maven-war-plugin:3.3.0:war (default-war) @ AI-on-the-edge ---
[INFO] Packaging webapp
[INFO] Assembling webapp [AI-on-the-edge] in [/mnt/nfs_share/newfolder/AI-on-the-edge/target/AI-on-the-edge-1.0-SNAPSHOT]
[INFO] Processing war project
[INFO] Copying webapp resources [/mnt/nfs_share/newfolder/AI-on-the-edge/src/main/webapp]
[INFO] Building war: /mnt/nfs_share/newfolder/AI-on-the-edge/target/AI-on-the-edge-1.0-SNAPSHOT.war
[INFO]
[INFO] --- maven-install-plugin:2.4:install (default-install) @ AI-on-the-edge ---
[INFO] Installing /mnt/nfs_share/newfolder/AI-on-the-edge/target/AI-on-the-edge-1.0-SNAPSHOT.war to /home/akshay/.m2/repository/com/example/AI-on-the-edge/1.0-SNAPSHOT/AI-on-the-edge-1.0-SNAPSHOT.war
[INFO] Installing /mnt/nfs_share/newfolder/AI-on-the-edge/pom.xml to /home/akshay/.m2/repository/com/example/AI-on-the-edge/1.0-SNAPSHOT/AI-on-the-edge-1.0-SNAPSHOT.pom
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 7.752 s
[INFO] Finished at: 2021-05-16T15:44:13+05:30
```

Fig:[3] Build inside jenkins pipeline

```

13     <properties>
14         <maven.compiler.target>1.8</maven.compiler.target>
15         <maven.compiler.source>1.8</maven.compiler.source>
16         <junit.version>5.7.0</junit.version>
17     </properties>
18
19     <dependencies>
20         <dependency>
21             <groupId>javax.ws.rs</groupId>
22             <artifactId>javax.ws.rs-api</artifactId>
23             <version>2.1.1</version>
24             <scope>provided</scope>
25         </dependency>
26         <dependency>
27             <groupId>org.glassfish.jersey.containers</groupId>
28             <artifactId>jersey-container-servlet</artifactId>
29             <version>2.31</version>
30         </dependency>
31         <dependency>
32             <groupId>org.glassfish.jersey.media</groupId>
33             <artifactId>jersey-media-json-jackson</artifactId>
34             <version>2.31</version>
35         </dependency>
36         <dependency>
37             <groupId>org.glassfish.jersey.media</groupId>
38             <artifactId>jersey-media-multipart</artifactId>
39             <version>2.31</version>
40         </dependency>
41         <dependency>
42             <groupId>org.glassfish.jersey.inject</groupId>
43             <artifactId>jersey-hk2</artifactId>
44             <version>2.31</version>
45         </dependency>
46         <dependency>
47             <groupId>org.glassfish.jersey.core</groupId>
48             <artifactId>jersey-client</artifactId>
49             <version>2.31</version>
50         </dependency>
51         <dependency>
52             <groupId>org.junit.jupiter</groupId>
53             <artifactId>junit-jupiter-api</artifactId>
54             <version>5.7.0</version>

```

Fig:[4] Pom.xml

## Build Result:

A war file is created upon the successful build.

### 4.3 Docker

Docker is a software platform for building applications based on containers — small and lightweight execution environments that make shared use of the operating system kernel but otherwise run-in isolation from one another. For our project since we will be automating the entire SDLC life cycle so to automatically deploy the project in a container we need to write the steps in a Docker file in our project. The Docker file will be as follows.

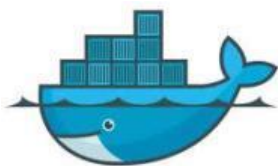
```
1 FROM tomcat
2
3 ADD ./target/AI-on-the-edge-1.0-SNAPSHOT.war /usr/local/tomcat/webapps/AI.war
4
5 EXPOSE 8080
6 EXPOSE 5672
7
8 RUN apt-get update && cd -- && apt-get install nfs-client -y && mkdir -p /mnt/nfs_share && apt-get install python3 -y && apt install python3-pip -y && pip3 insta
9
```

Fig:[5]

- tomcat is the name of the image.
- Add command is used to add the war file to tomcat webapps
- Expose is used to expose specified ports so that they can be used outside of the container here 8080 is exposed so to access our application and 5672 is the port for rabbitmq server.

These steps will be performed inside the container after the image will be downloaded from Docker Hub.

### Docker Hub



<https://hub.docker.com/repository/docker/akshaynagpal22/AI-on-the-edge>

Above is the link to the Docker hub repository.

You need to create an account in Docker Hub and create a repository in it. This docker image will be uploaded using CI pipeline which actually runs the above mentioned Dockerfile. This uploaded image will help us in our deployment part which is described in later part of this report.

## 4.4 Continuous Integration

### Jenkins

- We can now automate the whole process of pulling the latest code from the GitHub repository,
- Build the code using maven
- The program is compiled in a jar file. This is present in the target directory of Jenkins.
- This jar file is used to build the docker image.
- This docker image is pushed onto the docker hub repository using Jenkins.
- Docker image is deployed using ansible onto the client system.

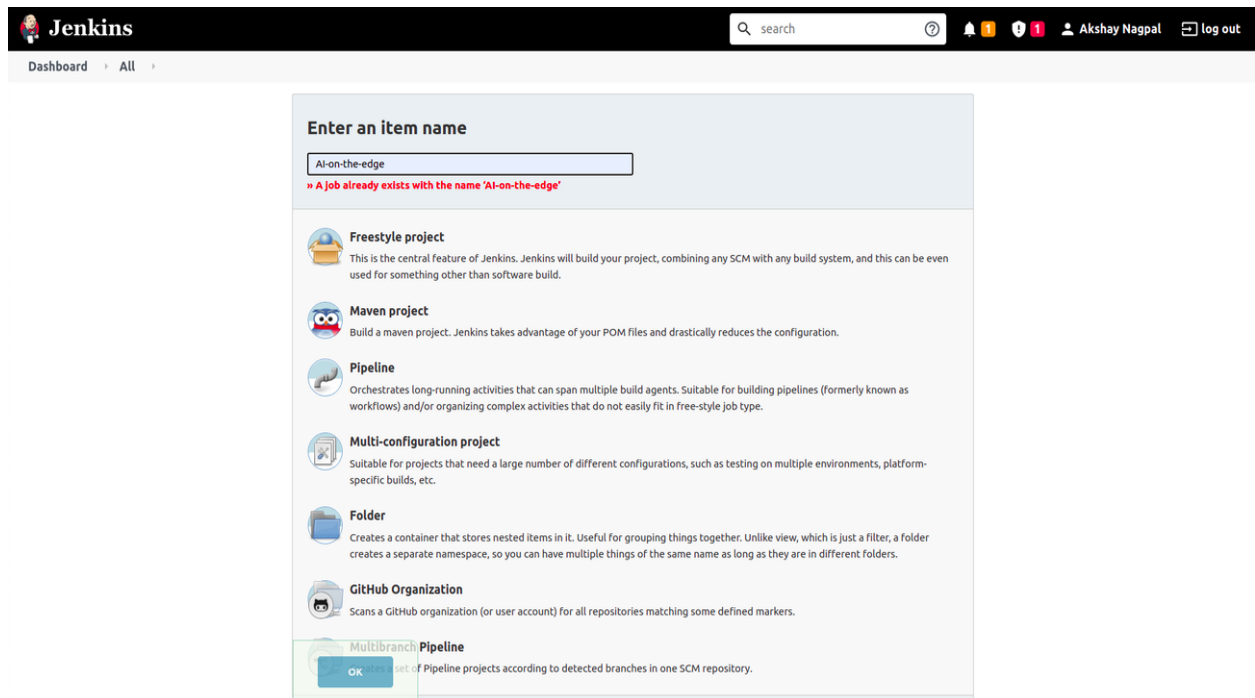


Fig:[6] Creating pipeline in jenkins

Create a new item as a pipeline project. Press the —configure link on the left side of the project.

Download all the plugins like Docker Plugin, Ansible Plugin, Git, etc.

Write these steps in the pipeline script.

1. Cloning git
2. Maven Build
3. Building docker image
4. Deploying docker image
5. Deploying docker image with ansible

## Configure Docker with Jenkins

For configuring docker, we need to enter the DockerHub credentials into Jenkins.

1. Dashboard > Manage Jenkins > Manage Credentials
2. Add Docker Hub ID and password

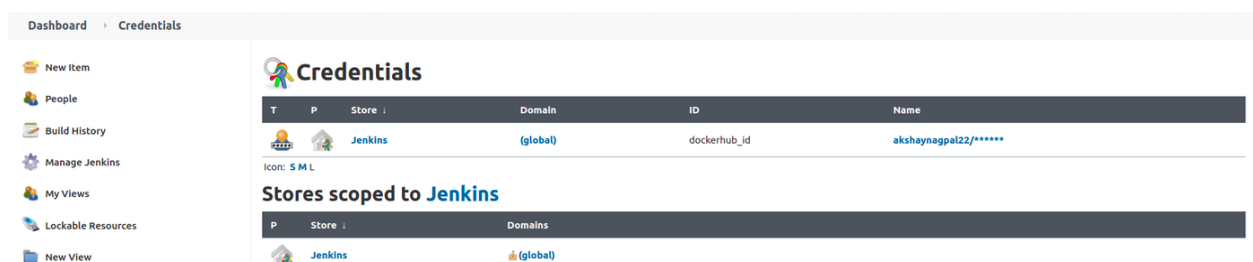


Fig:[7] Config Docker Credentials inside jenkins

## Pipeline script

1. We declare the pipeline script with initially declaring the environment variable. The below script is to declare the docker hub image. Enter

the same variable name as configured in the above step , example-dockerhub\_id.

```
1 pipeline { |
2
3   environment {
4
5
6     registryCredential = 'dockerhub_id'
7
8     dockerImage = ''
9
10  }
11 }
```

Fig:[8] Setting env variable of docker credentials

2. The second step is to add the GitHub project URL. Make sure that the project repository is publicly visible.

```
14 stages {
15
16   stage('Cloning our Git') {
17
18     steps {
19
20       git branch: 'main', url: 'https://github.com/akshay-nagpal/AI-on-the-edge.git'
21
22     }
23
24   }
25 }
```

Fig:[9] Git clone

3. The next step is Maven Build.

```
25
26 stage('Maven Build'){
27   steps{
28     sh 'mvn install'
29   }
30 }
31
```

Fig:[10] Maven build step

4. Building Docker image.

```
32
33 stage('Building our image') {
34
35   steps {
36
37     script {
38
39       dockerImage = docker.build 'akshaynagpal22/ai_on_the_edge:latest'
40
41     }
42
43   }
44
45 }
```

Fig:[11] Build docker image

## 5. Deploying Docker image.

```
47 | stage('Deploy our image') {  
48 |  
49 | | steps {  
50 | | | script {  
51 | | | | docker.withRegistry( '', registryCredential ) {  
52 | | | | | dockerImage.push()  
53 | | | | }  
54 | | | }  
55 | | }  
56 | }  
57 | }  
58 | }  
59 | }  
60 | }  
61 | }
```

Fig:[12] Pushing docker image to docker hub

## 6. Deploying with ansible.

```
63 |  
64 | stage('Deploy with ansible') {  
65 | | steps {  
66 | | | ansiblePlaybook becomeUser: null, colored: true, disableHostKeyChecking: true, installation: 'Ansible', inventory: 'invent  
67 | | | }  
68 | | }  
69 | }  
70 | }  
71 | }  
72 | }  
73 | }  
74 | }
```

Fig:[13] Deploy using ansible

## Configure Ansible with Jenkins-

To configure Ansible with Jenkins,

1. Dashboard > Manage Jenkins > Global Tool Configuration > Ansible.
2. Enter an ansible path.



## 4.5 Deployment using Ansible

Ansible is a software tool that provides simple but powerful automation for cross-platform computer support. It is primarily intended for application deployment, updates on workstations and servers, cloud provisioning, configuration management, intra-service orchestration, and nearly anything a systems administrator does on a weekly or daily basis. Ansible doesn't depend on agent software and has no additional security infrastructure, so it's easy to deploy.

We can deploy our docker image to multiple linux machines using ansible. We just need to configure the Ansible inventory file with appropriate configurations.

```
1 lines (1 sloc) | 27 Bytes
1 localhost ansible_user=mars
```

The above inventory file deploys the image on our own host machine for different users. But we can configure other machines also by adding.

<IP\_Address> ansible\_user=<user\_name>

Example: 192.168.43.64 ansible\_user=akshay

For executing various tasks like pushing docker images onto various linux machines. We need to write an Ansible playbook in which we mention all the tasks and commands we want Ansible to run.

Configuring ansible with different Linux machines-

In host machine-

1. \$ sudo apt install openssh-server
2. \$ sudo su jenkins (Switch to jenkins user)
3. \$ ssh-keygen -t rsa
4. Click enter
5. \$ ssh-copy-id @localhost

In client machine

1. \$ sudo apt install openssh-server
2. \$ sudo useradd username (This command is for creating a new user in local machine example mars)
3. \$ sudo usermod -aG docker ansible
4. \$ ssh-keygen -t rsa
5. Click enter
6. \$ hostname -I (To get the IP address)

As we mentioned in our deploy.yml file in the jenkins pipeline it will run all the commands that are mentioned in the playbook. These commands are necessary for successfully deploying the project on a machine mentioned in the inventory file.

```
---
- name: Pull and Run docker image of akshaynagpal22/ai_on_the_edge
  hosts: all
  tasks:
    - name: Pull akshaynagpal22/ai_on_the_edge
      docker_image:
        name: akshaynagpal22/ai_on_the_edge
        source: pull
    - name: Pull and Run docker SQL image
      hosts: all
      tasks:
        - name: Pull Mysql
          docker_image:
            name: mysql
            source: pull
        - name: Create default containers
          shell:
            cmd: docker container run -d -p 8085:3306 -e MYSQL_ROOT_PASSWORD=admin -e MYSQL_DATABASE=platformdb -e MYSQL_USER=test -e MYSQL_PASSWORD=test --name platform-mysql mysql
    - name: Create default containers
      shell:
        cmd: docker run -itd --privileged -p 8088:8080 -p 8090:5672 -e HIBERNATE_MYSQL_URL=192.168.29.132:8085 --name platform akshaynagpal22/ai_on_the_edge
    - name: Running commands
      shell:
        cmd: docker exec platform mount 192.168.29.132:/mnt/nfs_share/ /mnt/nfs_share/
    - name: rabbitmq service
      shell:
        cmd: docker exec platform service rabbitmq-server start
    - name: rabbitmq user add
      shell:
        cmd: docker exec platform rabbitmqctl add_user test test
    - name: rabbitmq user add
      shell:
        cmd: docker exec platform rabbitmqctl set_user_tags test administrator
    - name: rabbitmq user add
      shell:
        cmd: docker exec platform rabbitmqctl set_permissions -p / test ".*" ".*" ".*"
    - name: HB_manager
      shell:
        cmd: docker exec platform python3 /mnt/nfs_share/HB_cron.py
```

Fig:[14] deploy.yml

## Description of commands mentioned in deploy.yml

1. We will pull the docker images of the project akshaynagpal122/ai\_on\_the\_edge and also mysql image.  
<docker hub image>

2. **docker container run -d -p 8085:3306 -e MYSQL\_ROOT\_PASSWORD=admin -e MYSQL\_DATABASE=platformdb -e MYSQL\_USER=test -e MYSQL\_PASSWORD=test --name platform-mysql mysql**

This command will run the platform-mysql container which is using the image mysql.specifications that are passed-

1. Run the docker container on 8085.
2. Setting the root password, database name,mysql user and it's password
3. Finally it specifies the docker container name platform-mysql which is using docker image mysql.

3. **docker run -itd --privileged -p 8088:8080 -p 8090:5672 -e HIBERNATE\_MYSQL\_URL=192.168.29.132:8085 --name platform akshaynagpal22/ai\_on\_the\_edge**

This command will run docker container akshaynagpal22/ai\_on\_the\_edge with following specifications-

1. Run the docker container on port 8088.
2. rabbit mq port 5672 is mapped to 8090.
3. In this we are also passing mysql container url which was created above as a environment variable so that it can be used inside this container.

4. After this we are running some commands that are to be executed inside akshaynagpal22/ai\_on\_the\_edge container described below in the same order as seen in image.

1. `docker exec platform mount 192.168.29.132:/mnt/nfs_share/ /mnt/nfs_share/` --- this is used to mount nfs
2. `docker exec platform service rabbitmq-server start` -- this is used to start rabbitmq server
3. `docker exec platform rabbitmqctl add_user test test` -- this is used to create a user for rabbitmq
4. `docker exec platform rabbitmqctl set_user_tags test administrator` -- this is used to give administrator privileges to the user(test created in above step).
5. `docker exec platform rabbitmqctl set_permissions -p / test ".*" ".*" ".*"` -- this is used to set permissions for the test user

## 4.6 Monitoring and Logging

Monitoring in DevOps is proactive, meaning it finds ways to enhance the quality of applications before bugs appear. For this project, we have used the ELK stack for monitoring purposes.

- E stands for Elastic search which is real time search engine and used for storing logs
- L stands for Logstash which is a data processing pipeline and feeds data to elastic search
- K stands for Kibana which is used for Visualization.

All Logstash installation commands are mentioned in Dockerfile so no need to install it manually. We will be using cloud deployment for elastic search and kibana known as elastic cloud.

- Firstly, create log4j2.xml for specifying the path where logs are to be stored and also the pattern for the logs.

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <Configuration status="INFO">
3   <Properties>
4     <Property name="logdir"/>./home</Property>
5     <Property name="layout">%d [%t] %-5p %c- %m%n</Property>
6   </Properties>
7   <Appenders>
8     <Console name="ConsoleAppender" target="SYSTEM_OUT">
9       <PatternLayout pattern="%d{HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n" />
10    </Console>
11    <File name="FileAppender" fileName="${logdir}/platform.log" immediateFlush="true" append="true">
12      <PatternLayout pattern="%d{yyy-MM-dd HH:mm:ss.SSS} [%t] %-5level %logger{36} - %msg%n"/>
13    </File>
14  </Appenders>
15  <Loggers>
16    <Logger name="org.hibernate" level="error" additivity="false">
17      <AppenderRef ref="Console-Appender"/>
18    </Logger>
19    <Root level="info">
20      <AppenderRef ref="ConsoleAppender" />
21      <AppenderRef ref="FileAppender"/>
22    </Root>
23  </Loggers>
24 </Configuration>
```

Fig:[15] log4j2.xml

- Prepare logs for each function.i.e using logger class we can write statements inside functions which will appear in logs whenever that function is reached an example is shown in below image.

```
45 @POST
46 @Path("/registeruser")
47 @Produces(MediaType.TEXT_PLAIN)
48 @Consumes(MediaType.APPLICATION_JSON)
49 public Response registeruser(Login loginobj) throws URISyntaxException {
50     LoginService ls_obj = new LoginService();
51     int ret = ls_obj.registeruser(loginobj);
52     System.out.println(loginobj.getEmail());
53     System.out.println(loginobj.getPassword());
54     System.out.println(loginobj.getUsertype());
55     if (ret == 0) {
56         logger.info("{} User registration is unsuccessful ", loginobj.getEmail());
57
58         return Response.status(406).build();
59     }
60     logger.info("{} User registration is successful ", loginobj.getEmail());
61
62     return Response.ok().status(200).build();
63 }
```

Fig:[16] Login controller logger example

- Logfile will be generated (platform.log) for test cases. Also, logs will be generated while executing the .jar file.
- We have to create a logstash conf file to configure logstash and upload logs on elastic search. For configuring cloud elastic search we need to create deployment on elastic search website which will give us the cloud id, username, password.

```

1 input {
2   file {
3     path => ["/home/platform.log"]
4     start_position => "beginning"
5   }
6 }
7
8 filter {
9   grok {
10    match => [
11      "message", "%{TIMESTAMP_ISO8601:timestamp_string} \\[%{GREEDYDATA:thread}\\] %{LOGLEVEL:level} %{GREEDYDATA:logger} \\- %{GREEDYDATA:line}"
12    ]
13  }
14
15  date {
16    match => ["timestamp_string", "YYYY-MM-dd HH:mm:ss.SSS"]
17  }
18
19  mutate {
20    remove_field => [timestamp_string]
21  }
22 }
23
24 output {
25   elasticsearch {
26     cloud_id => "i-o-optimized-deployment:dXMtd2VzdDEuZ2NwLmNsb3VhLnVzLmVjZDkyNWNiZmM2MzZiYjQ4ZjZiODIzMmM5NWZlYzIhMzEwJDVjOWIwMjI5YTg2MzQyYmY4OGE4NGI3ZjRkZmVhMTk5"
27     cloud_auth => "elastic:9BEPs40KHKyxCs41VAEwCNU1"
28     index => "platform_ai"
29   }
30
31   stdout {
32     codec => rubydebug
33   }
34 }

```

Fig:[17] logstash conf file

-Now after running `/bin/logstash <path to platform.log(log file)>` it will send data to elastic search.

-Now we can visualize our data on elastic search using Kibana (using an elastic cloud for this project).

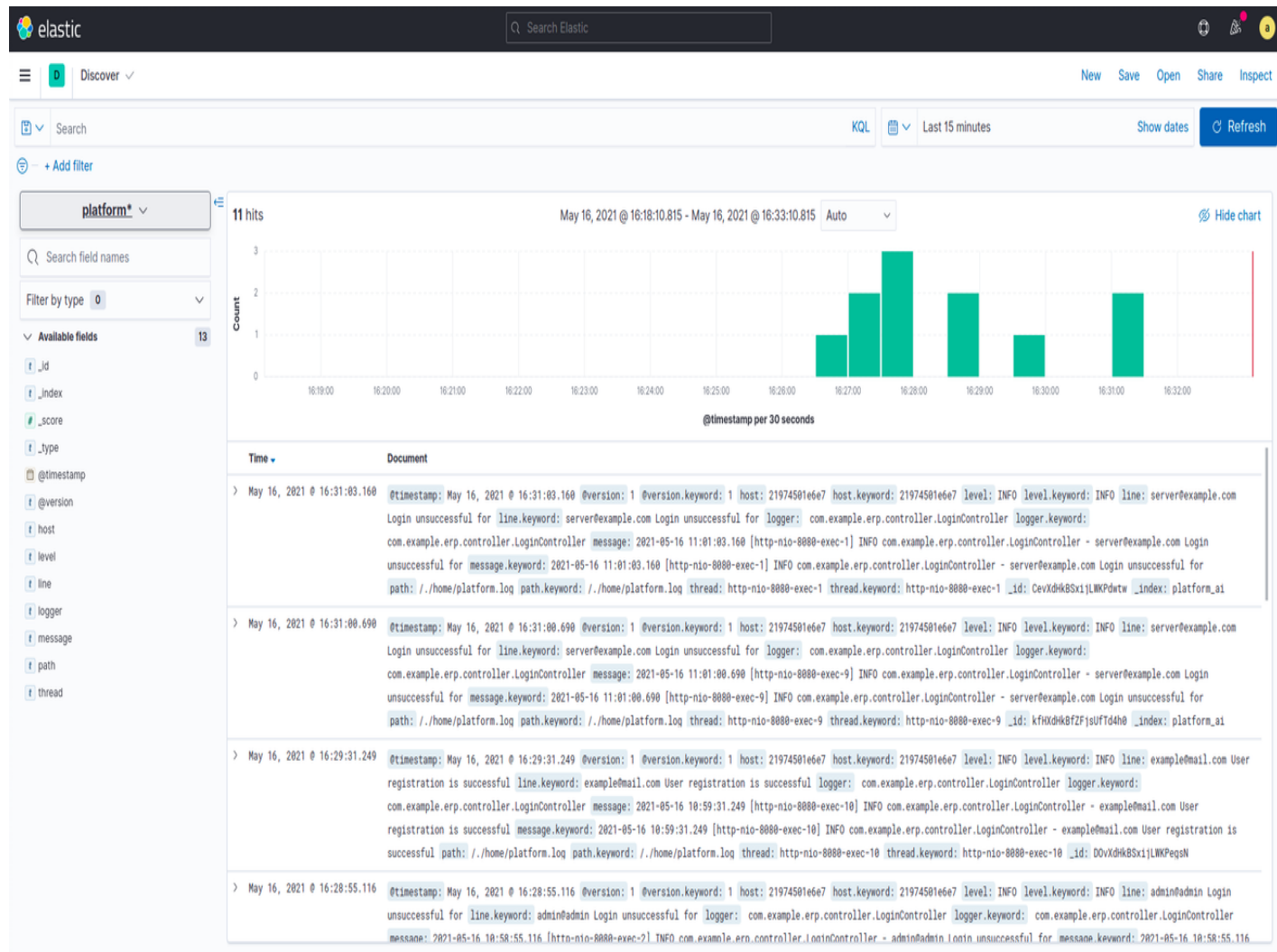


Fig:[18] bar plot showing the time intervals in which users interacted with app

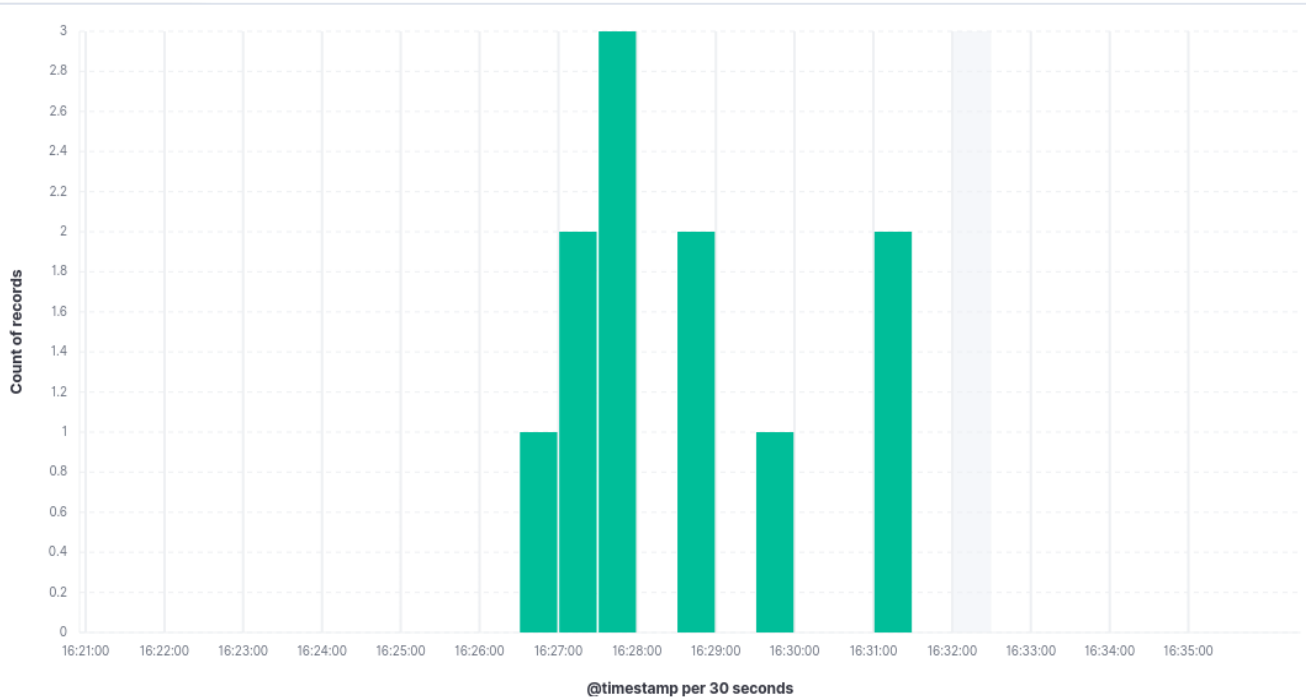


Fig:[19] Above bar plot shown clearly

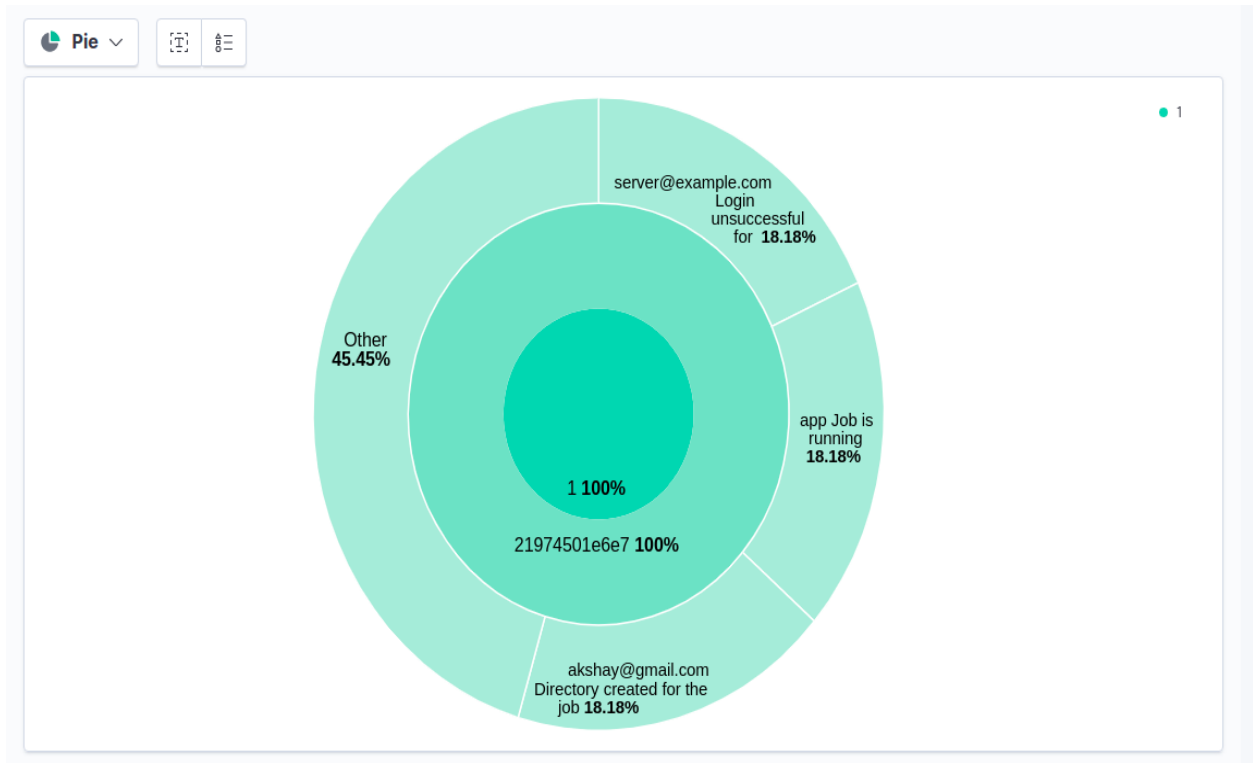


Fig:[20] Pie chart shows percentage of different types of functionalities used by users in app



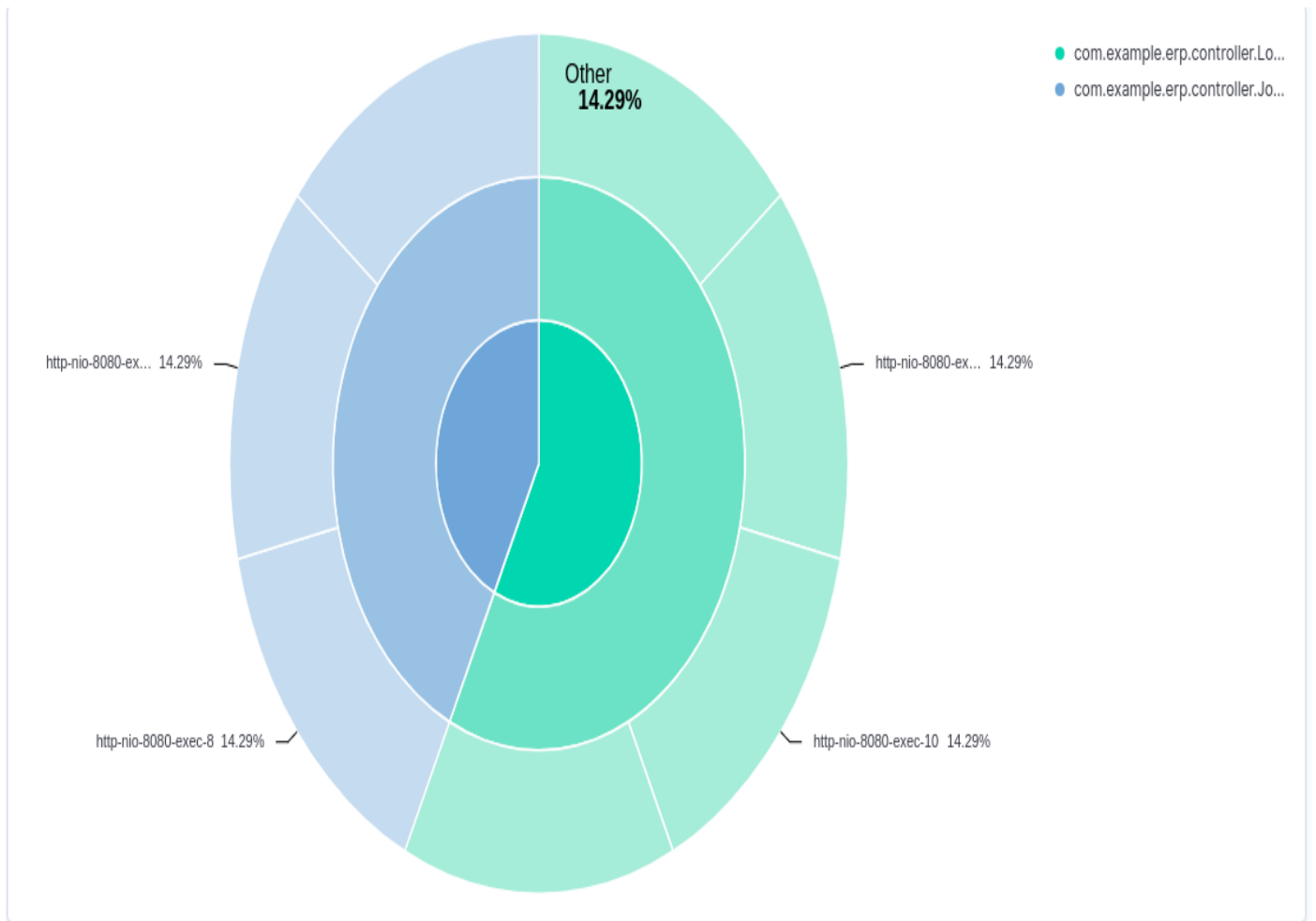


Fig:[21] Pie chart shows percentage of usage of different controllers in our app

## 4.7 Final Building and deployment

After all this setup Our pipeline script will look like this-

```
1 pipeline { |
2
3   environment {
4
5     registryCredential = 'dockerhub_id'
6
7     dockerImage = ''
8
9   }
10
11   agent any
12
13   stages {
14
15     stage('Cloning our Git') {
16
17       steps {
18
19         git branch: 'main', url: 'https://github.com/akshay-nagpal/AI-on-the-edge.git'
20
21       }
22
23     }
24
25     stage('Maven Build'){
26
27       steps{
28
29         sh 'mvn install'
30
31       }
32
33     }
34
35     stage('Building our image') {
36
37       steps {
38
39         script {
40
41           dockerImage = docker.build 'akshaynagpal22/ai_on_the_edge:latest'
42
43         }
44
45       }
46
47     }
48
49     stage('Deploy our image') {
50
51       steps {
52
53         script {
54
55           docker.withRegistry( '', registryCredential ) {
56
57             dockerImage.push()
58
59           }
60
61         }
62
63       }
64
65     }
66
67     stage('Deploy with ansible') {
68
69       steps {
70
71         ansiblePlaybook becomeUser: null, colored: true, disableHostKeyChecking: true, installation: 'Ansible', inventory: 'inven
72
73       }
74
75     }
76
77   }
78 }
```

Fig:[22] pipeline script

Save this pipeline script and hit build now. After successful execution of pipeline we will see something like this. In ansible will do all the tasks as mentioned in “Deploy with ansible” step.

# Pipeline AI-on-the-edge



## Stage View

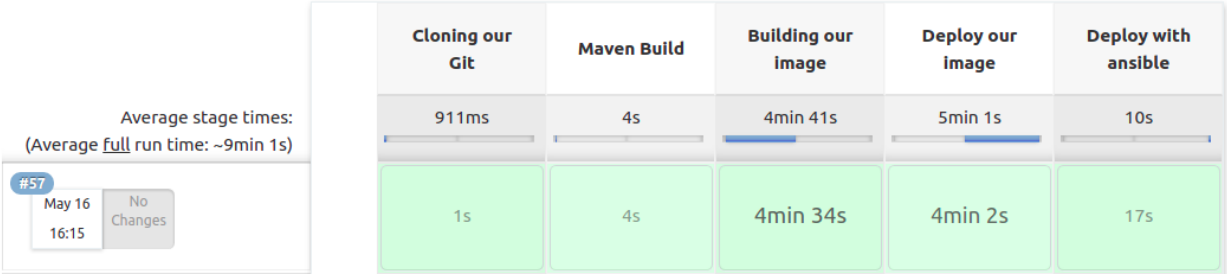


Fig:[23] Jenkins pipeline

Now we can access the application on **localhost:8088/AI.** On all the nodes that are mentioned in the inventory file.

## 5. Experimental Setup

### 5.1 Use case diagram

The use case dig shows all the functional requirements the application provides and also different type of actors that will be involved. As we can see in the below dig we have a different types of actors which are-

- User- Which will use the platform for running their applications
- Resource - These are the one which we will register their resources and these resources(server nodes) will be used to run application by the platform
- Admin - They will have admin level access to the platform

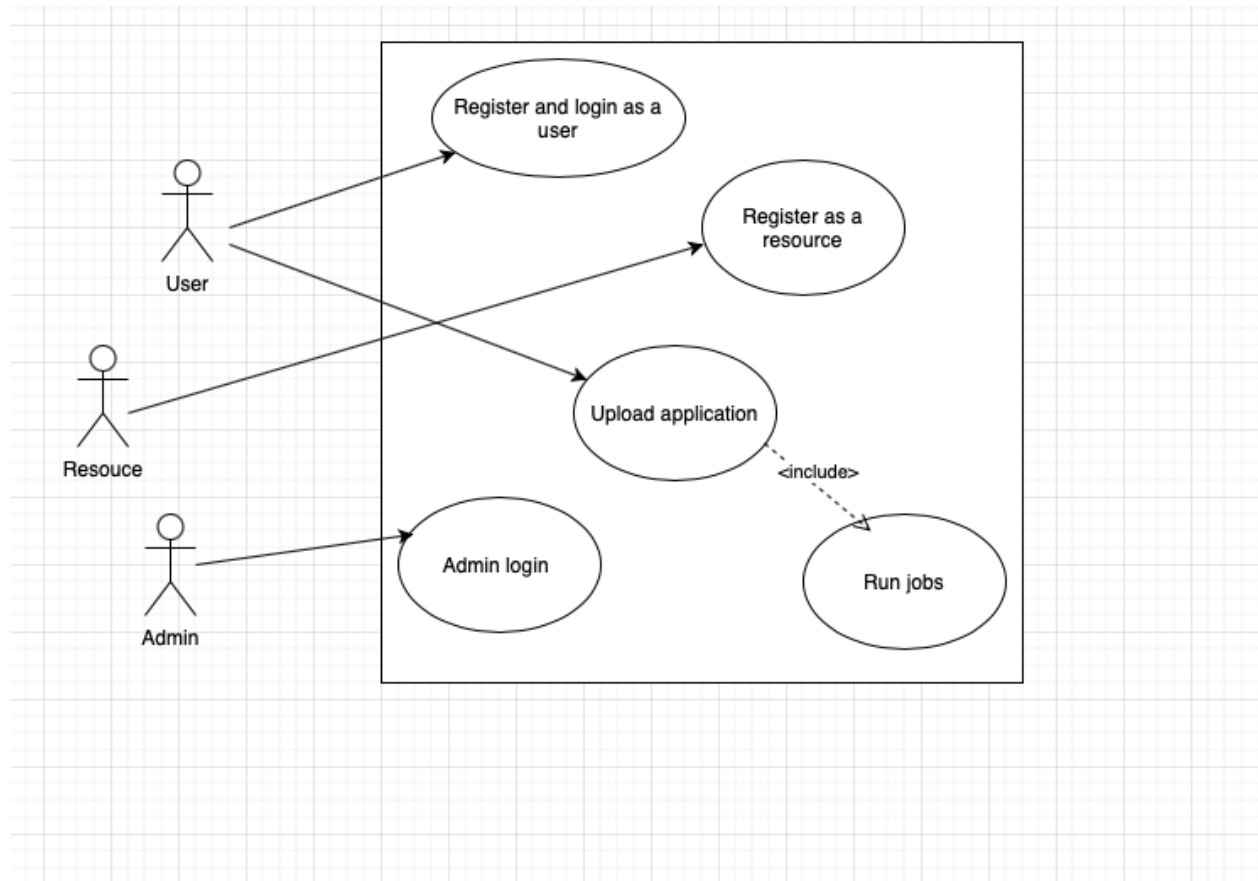


Fig:[24] Use case dig

## 5.2 Design diagram

There are two Design diagrams, first one shows the abstract view in which all the core functionalities and interaction are compressed inside Platform manager and how it interacts with other external entities such as node machine, nfs(Network file system). Second one shows the details inside the platform manager.

First diagram shows the interaction of platform manager with NFS server in which all the code files, scripts and necessary intermediary data is stored(will be elaborated in second diagram). Platform manager also interacts with node machines(which have nfs client) by scheduling jobs on them.

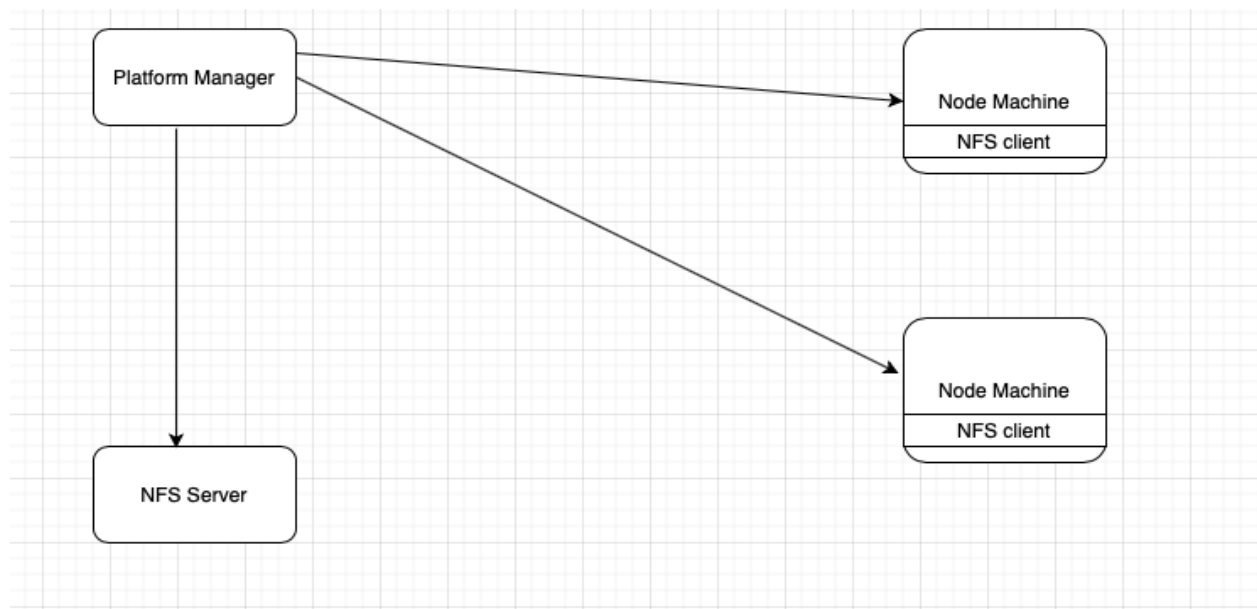


Fig:[25] Interaction of platform manager

Second diagram shows the inside working of platform manager and how two types of users interact with it i.e user and resource. First user which wants to run application on the platform will first sign in or sign up. After that user will be redirected to a page which will allow the user to upload application (All necessary files with a standard naming convention) and also at the same time heart beat manager cron job will be called which will continuously create active list of resources (among the ones that are registered), after this user will be redirected to a dashboard which will allow user to run their applications by entering the name of the application, now as soon as user press run job load balancer module gets called. This load balancer module is responsible for selecting the best resource from the active list that was created before. This selection is done by calculating the score of each resource using the metrics of that resource (such as CPU temperature, events per second etc). Now when load balancer select the best resource a send script is called which is responsible for scheduling job on this resource. For this scheduling purposes we used Rabbitmq which is a

queue based scheduling package in python In which platform work as rabbitmq server and all the resources(Nodes) act as listeners.The listener script sends ack whenever job is successfully completed, which will be displayed on the run job page where now user can download the results.Now there may be the case when job fails or abruptly resource goes down in that case load balancer will be called again so that it can be scheduled again.

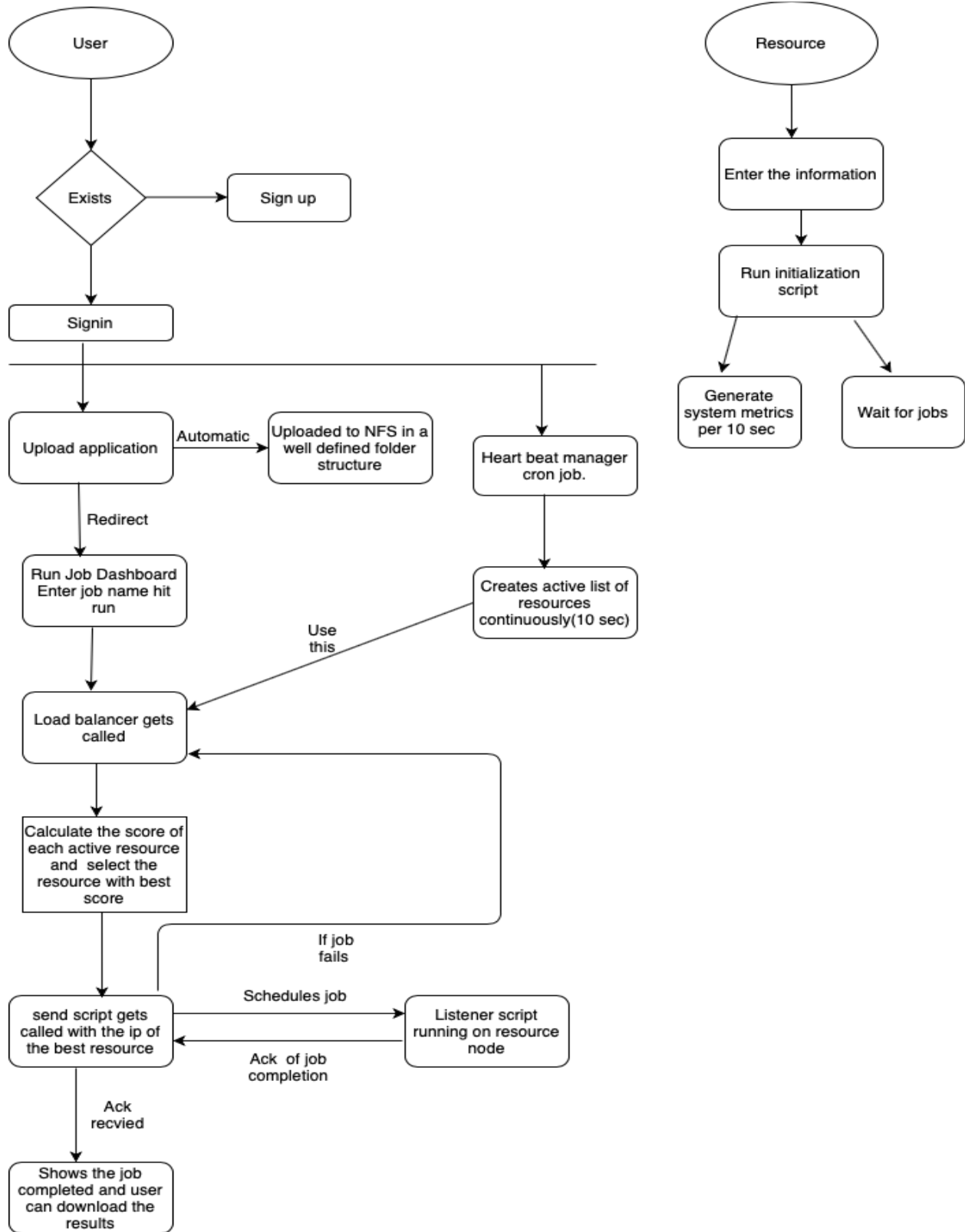


Fig:[26] Inside platform manager



## 1. Initialization script

```
1 import subprocess
2 import sys
3 import os
4 import time
5 os.system("pip3 install schedule")
6 import schedule
7 os.system("pip3 install apscheduler")
8 from apscheduler.schedulers.background import BackgroundScheduler
9 from apscheduler.schedulers.background import BlockingScheduler
10 print ("start")
11 ip=sys.argv[1]
12 # user=sys.argv[2]
13 os.system("sudo apt update")
14 os.system("sudo apt install nfs-common -y")
15 os.system("sudo mkdir -p /mnt/nfs_share")
16 os.system("sudo mount 192.168.29.132:/mnt/nfs_share /mnt/nfs_share")
17 os.system("sudo echo '192.168.29.132:/nfs_share /mnt nfs defaults 0 0' >>sudo /etc/fstab")
18 os.system("sudo apt install sysbench")
19 os.system("sudo apt-get install lm-sensors ")
20 # os.system("sudo apt install openssh-server")
21 # os.system("ssh-keygen")
22 os.system("pip3 install pika")
23 os.system("sudo sensors-detect ")
24 os.system("sudo service kmod start")
25 def run():
26     os.system("python3 /mnt/nfs_share/newfolder/AI-on-the-edge/load.py "+ip+"")
27 scheduler = BlockingScheduler()
28 scheduler2=BackgroundScheduler()
29 scheduler2.add_job(run,'cron',second='*/10')
30 # scheduler.add_job(run, 'cron', second='*/5')
31 scheduler2.start()
32 os.system("python3 /mnt/nfs_share/newfolder/AI-on-the-edge/recv.py "+ip+"")
33 # scheduler.every(5).seconds.do(run)0
34 # while(True):
35 #     schedule.run_pending()
36 print ("end")
```

Fig:[27] Intialize.py

This is the script that we will perform several tasks on each Resource for making it suitable to be used as a server node. First, it will download some necessary packages and mount NFS. Second, it will continuously generate metrics by running load.py inside a file name ip\_address.txt here IP address is the IP of Resource(node) this file will be present in NFS so that it can be used.

Following metrics are taken into consideration-

- Free CPU percentage
- Free Memory Percentage
- CPU's events per second
- Actual free RAM
- Current temperature

With this, it also runs **recv.py** which acts as a listener for the jobs at that resource. In this we defined two queues, one queue for consuming information about jobs and other is ack queue for sending acknowledgements.

```
1 import pika
2 import sys
3 import os
4 #change this
5 credentials = pika.PlainCredentials(username='test', password='test')
6 connection = pika.BlockingConnection(pika.ConnectionParameters(host='192.168.29.132',port='8090',credentials=
7 channel = connection.channel()
8
9 channel.exchange_declare(exchange='jobs', exchange_type='direct')
10
11 result = channel.queue_declare(queue='', exclusive=True)
12 queue_name = result.method.queue
13
14 ip = sys.argv[1]
15
16 #Apply some error handling
17 ack_connection = pika.BlockingConnection(pika.ConnectionParameters(host='192.168.29.132',port='8090',credenti
18 ack_channel = ack_connection.channel()
19 ack_channel.queue_declare(queue='ackqueue')
20
21
22 channel.queue_bind(
23     exchange='jobs', queue=queue_name, routing_key=ip)
24
25 print(' [*] Waiting for jobs. To exit press CTRL+C')
26
27
28 #this the function the will execute the task
29 def callback(ch, method, properties, body):
30     print(" [x] %r:%r" % (method.routing_key, body))
31     os.system(body)
32     print("DONE!!!")
33     send_ack()
34
35 def send_ack():
36     ack_channel.basic_publish(exchange='', routing_key='ackqueue',body='1')
37     return
38
39 channel.basic_consume(
40     queue=queue_name, on_message_callback=callback, auto_ack=True)
41
42 channel.start_consuming()
```

Fig:[28] recv.py

Now whenever a Resource node is already configured using initialize.py but it shuts down and after some time it is live again at that time it should run configure.py that only does the second and third task.

```
1 import os
2 import sys
3 from apscheduler.schedulers.background import BackgroundScheduler
4 from apscheduler.schedulers.background import BlockingScheduler
5 ip=sys.argv[1]
6 os.system("sudo mount 192.168.29.132:/mnt/nfs_share /mnt/nfs_share")
7 def run():
8     os.system("python3 ../mnt/nfs_share/newfolder/AI-on-the-edge/load.py "+ip+"")
9     #scheduler = BlockingScheduler()
10    scheduler2=BackgroundScheduler()
11    scheduler2.add_job(run,'cron',second='*/10')
12    scheduler2.start()
13    os.system("python3 ../mnt/nfs_share/newfolder/AI-on-the-edge/recv.py "+ip+"")
```

Fig:[29] configure.py

## 2. Heart\_beat Manager cron job

This script gets executed as soon as some user sign in. It creates cron job for executing **HB\_manager.py**. This **HB\_manager.py** is responsible for creating something called **active.pickle** which is the list of ip addresses of all the active resources with their score (explained below) so the final structure inside **active.pickle** is

[(ip1,score1),(ip2,score2).....].

```
61 mydb = mysql.connector.connect( host="192.168.29.132",port="8085",user="test",password="test",database="platform")
62 mycursor = mydb.cursor()
63 mycursor.execute("SELECT * FROM Resource")
64 myresult = mycursor.fetchall()
65 mapping=defaultdict(lambda x:[])
66 #Constructing mapping
67 for x in myresult:
68     mapping[x[1]]=x[3],x[2]]
69
70 #Constructing dead and active lists
71 print(x)
72 dead=[]
73 active=[]
74
75
76 mapping=dict(mapping)
77 print(mapping)
78
79 for k,v in mapping.items():
80     # print(k,v)
81     # if k=='192.168.43.49':
82     #     continue
83     response=os.system("ping -w 1 " + k)
84     # sock=socket(socket.AF_INET, socket.SOCK_STREAM)
85     # response=sock.connect_ex((k,))
86     print(response)
87     if response==0:
88         active.append(k)
89     else:
90         dead.append(k)
91
92
93 print("Active ::" , active)
94 print("dead ::" , dead)
95 fa=open("./mnt/nfs_share/active.pickle","wb")
96 # fa=open("/home/rahul/Documents/active.pickle","wb")
97 pickle.dump(active,fa)
98 fd=open("./mnt/nfs_share/dead.pickle","wb")
99 # fd=open("/home/rahul/Documents/dead.pickle","wb")
100 pickle.dump(dead,fd)
101
```

In this we get the ip of each resource from our database Resource Table which contains the ip addr of each resource registered with our platform.

*Using this ip addr values we read ip\_addr.txt files to get the values of different metrics and calculate the load score, we describe the load score as follows:*

1. **System's current load:** This is a metric computed using the free CPU percentage and free memory percentage. This is a score that ranges from 0- 40. Higher score means less load. This is calculated as follows:

$$\text{load} = 1 / ( 3 / \text{free\_cpu\%} + 1 / \text{free\_mem\%} )$$

The CPU weightage in this formula is three times as that of memory.

2. **System's performance:** This is a performance measure of the system. This is computed using the formula:

$$\text{performance} = \text{number\_of\_events\_per\_sec} / 10000 + \min(2, \text{free\_RAM(in GB)} / 10)$$

The number of events per seconds are calculated using the library 'sysbench'. A good CPU will have this benchmark greater than 10000. We assume that a service needs at max 2 GB of memory so that memory free larger than 2 GB does not increase the score. The weightage of memory is less than CPU, so we divide the memory term by 10.

3. **System Temperature:** If the temperature of the system is critically high, the score will be made zero, so that no more services will be scheduled on that system.

Finally score is calculated as follows-

$$\text{Score} = \text{current\_load} \times \text{performance} \times f(\text{current\_temperature})$$

```

31 def score(name):
32     command="cd ../mnt/nfs_share/"
33     # os.system(command)
34     full_name="../mnt/nfs_share/{name}.txt".format(name=name)
35     f=open(full_name)
36
37     lst=f.readlines()[::-1]
38     lst=[float(i.replace('\n','')) for i in lst]
39     print(lst)
40
41     free_cpu=lst[1]
42
43     free_mem=lst[2]
44
45     load=(1/((3/free_cpu)+(1/free_mem)))
46
47     free_RAM=lst[3]
48
49     current_temperature=lst[4]
50
51     number_of_events_per_sec=lst[0]
52
53     performance = ((number_of_events_per_sec) + min(2, free_RAM) / 10)
54
55     Score = load * performance * (current_temperature)
56     if(current_temperature >= 100):
57         Score=0
58
59     return Score

```

Fig:[30] score function inside HB\_manager.py

### 3. Load Balancer

Now this will be called whenever user submits a job to run. Now to decide where to run it will read active.pickle(see above) and sort that list according to the score in increasing order. It will select the ip address at the first location of the sorted list. After that it will call **send.py** and pass this ip as a argument

```
13 active=pickle.load(fa)
14
15 # fd=open("/mnt/nfs_share/dead.pickle","rb")
16 # fd=open("./mnt/nfs_share/dead.pickle","rb")
17 # dead=pickle.load(fd)
18 print("Active :::", active)
19 # print("dead :::", dead)
20 dict1 = []
21 for i in active:
22     # load(i)
23     dict1.append(i)
24 ip=dict1[0]
25
26
27 # ip='192.168.29.49'
28 dict1.sort(key = lambda x: x[1])
29 # command="python3 "+job_path
30 # os.mkdir("/mnt/nfs_share/"+email+"/"+appname+".py")
31 # os.system("python3 ./mnt/nfs_share/newfolder/AI-on-the-edge/send.py "+ip+" "+email+" "+python3 "+appname+".py")
32 path="/mnt/nfs_share/"+email+"/"
33 os.system("chmod -R 777 "+path)
34 test=open("./mnt/nfs_share/test.txt","w")
35 test.write("working")
36 #making a directory for a app to store completed.txt
37 # os.makedirs(path+appname,exist_ok=True)
38 # temp=open(path+appname+"/"+completed.txt","w")
39 # temp.write("")
40 # temp.close()
41 os.system("python3 ./mnt/nfs_share/newfolder/AI-on-the-edge/send.py "+ip+" "+email+" "+appname+" "+python3 "+path+appname+".py")
```

Fig:[31] load\_balancer.py

### 4. Sending jobs to resources

Script responsible for this is Send.py. This will be called by Load balancer. It will schedule job on a resource which is also done by rabbitmq i.e it publish information of the job to a queue(name of the queue is the ip address of resource and remember above that we are

running recv.py which is listening with a queue name of it's ip).This structure of rabbitmq is called routing.

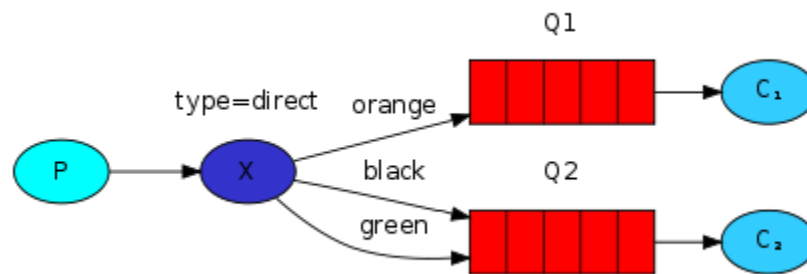


Fig:[32] Routing Rabbitmq

ref(<https://www.rabbitmq.com/tutorials/tutorial-four-python.html>)

In above diagram these orange black and green are routing\_keys in our case these are the ip addresses of our resources.To specify this queue name we use something called routing\_key so when we publish a message to a queue we also have to pass routing\_key say R1 which is matched and the data gets publish to a particular queue(with routing\_key R1).

```

55 scheduler = BlockingScheduler()
56 scheduler2=BackgroundScheduler()
57 # scheduler.add_job(run, 'cron', second='*/5')
58 #change this
59 scheduler2.start()
60 credentials = pika.PlainCredentials(username='test', password='test')
61 connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost',credentials=credentials))
62 channel = connection.channel()
63
64 # channel.confirm_delivery()
65 channel.exchange_declare(exchange='jobs', exchange_type='direct')
66
67 ack_connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
68 ack_channel = ack_connection.channel()
69 q=ack_channel.queue_declare(queue='ackqueue')
70 message = ' '.join(sys.argv[4:]) or 'DEFAULT MESSAGE'
71 # pika.queue_purge('ackqueue')
72 os.system('rabbitmqctl purge_queue ackqueue')
73 channel.basic_publish(
74     exchange='jobs', routing_key=ip, body=message, properties=pika.BasicProperties(content_type='text/plain',
75     delivery_mode=1))
76
77 # scheduler2.add_job(run,'cron',second='*/30')
78 scheduler2.add_job(run,'interval',ack_flag,seconds=10)
79 # print(" [x] Sent %r:%r" % (severity, message))
80
81 ack_channel.basic_consume(
82     queue='ackqueue', on_message_callback=callback)
83 ack_channel.start_consuming()
84 print("Reached2")
85
86 connection.close()
87

```

Fig:[33] send.py sending job info to a resource



```

55 scheduler = BlockingScheduler()
56 scheduler2=BackgroundScheduler()
57 # scheduler.add_job(run, 'cron', second='*/5')
58 #change this
59 scheduler2.start()
60 credentials = pika.PlainCredentials(username='test', password='test')
61 connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost',credentials=credentials))
62 channel = connection.channel()
63
64 # channel.confirm_delivery()
65 channel.exchange_declare(exchange='jobs', exchange_type='direct')
66
67 ack_connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
68 ack_channel = ack_connection.channel()
69 q=ack_channel.queue_declare(queue='ackqueue')
70 message = ' '.join(sys.argv[4:]) or 'DEFAULT MESSAGE'
71 # pika.queue_purge('ackqueue')
72 os.system('rabbitmqctl purge_queue ackqueue')
73 channel.basic_publish(
74     exchange='jobs', routing_key=ip, body=message, properties=pika.BasicProperties(content_type='text/plain',
75                                                                                     delivery_mode=1))
76
77 # scheduler2.add_job(run, 'cron', second='*/30')
78 scheduler2.add_job(run, 'interval', ack_flag, seconds=10)
79 # print(" [x] Sent %r:%r" % (severity, message))
80
81 ack_channel.basic_consume(
82     queue='ackqueue', on_message_callback=callback)
83 ack_channel.start_consuming()
84 print("Reached2")
85
86 connection.close()
87

```

---

Fig:[34] Handling ack and failures in send.py

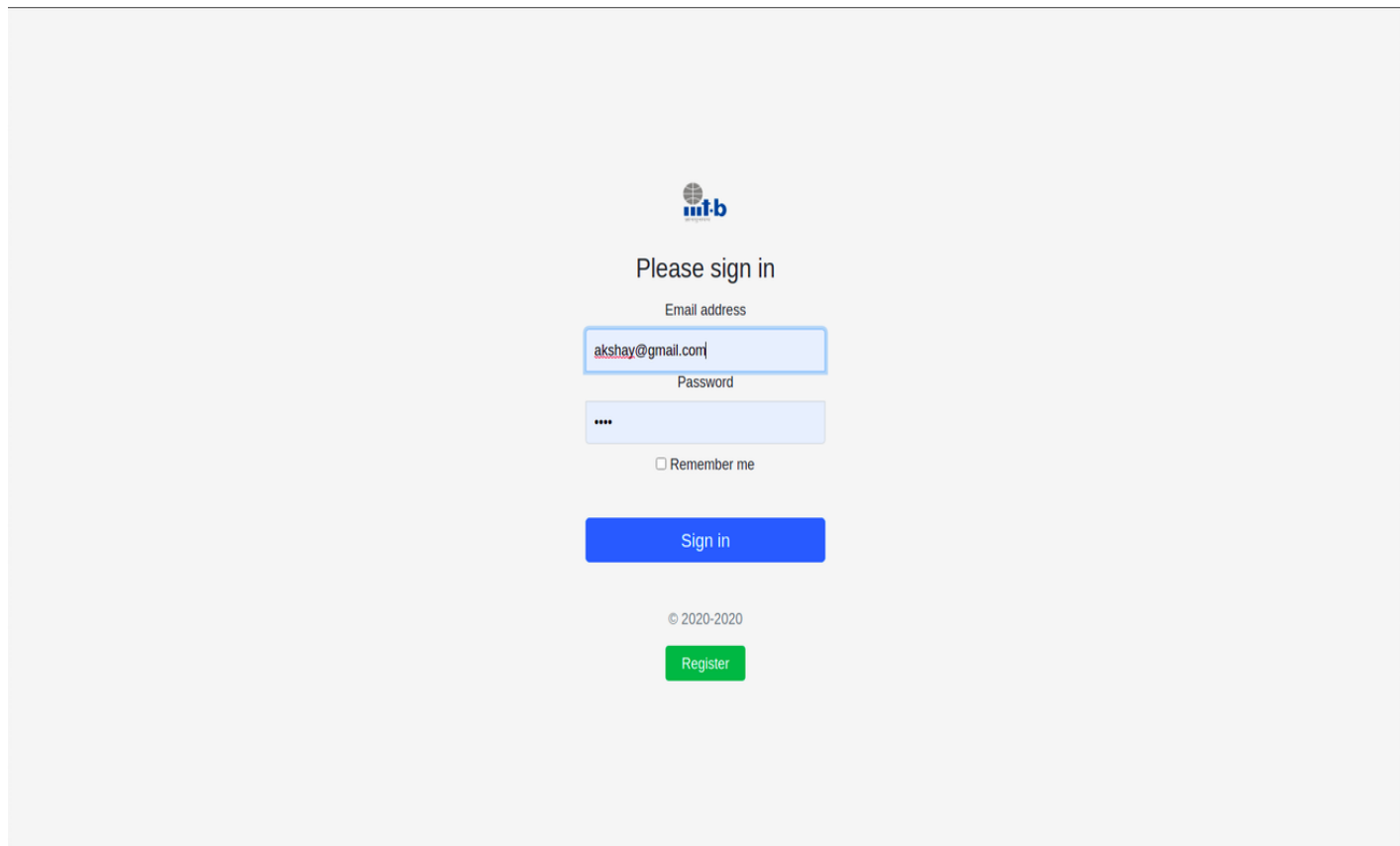
## 6. Results and Discussions

### 6.1 Home page of the application



Fig:[35] Login page

## 6.2 User login



The image shows a user login page with a light gray background. At the top center is a logo consisting of a globe icon and the text 'mt-b'. Below the logo, the text 'Please sign in' is displayed. Underneath, there are two input fields: the first is labeled 'Email address' and contains the text 'akshay@gmail.com'; the second is labeled 'Password' and contains four dots. Below the password field is a checkbox labeled 'Remember me'. A blue 'Sign in' button is positioned below the checkbox. At the bottom center, there is a copyright notice '© 2020-2020' and a green 'Register' button.

mt-b

Please sign in

Email address

akshay@gmail.com

Password

....

☐ Remember me

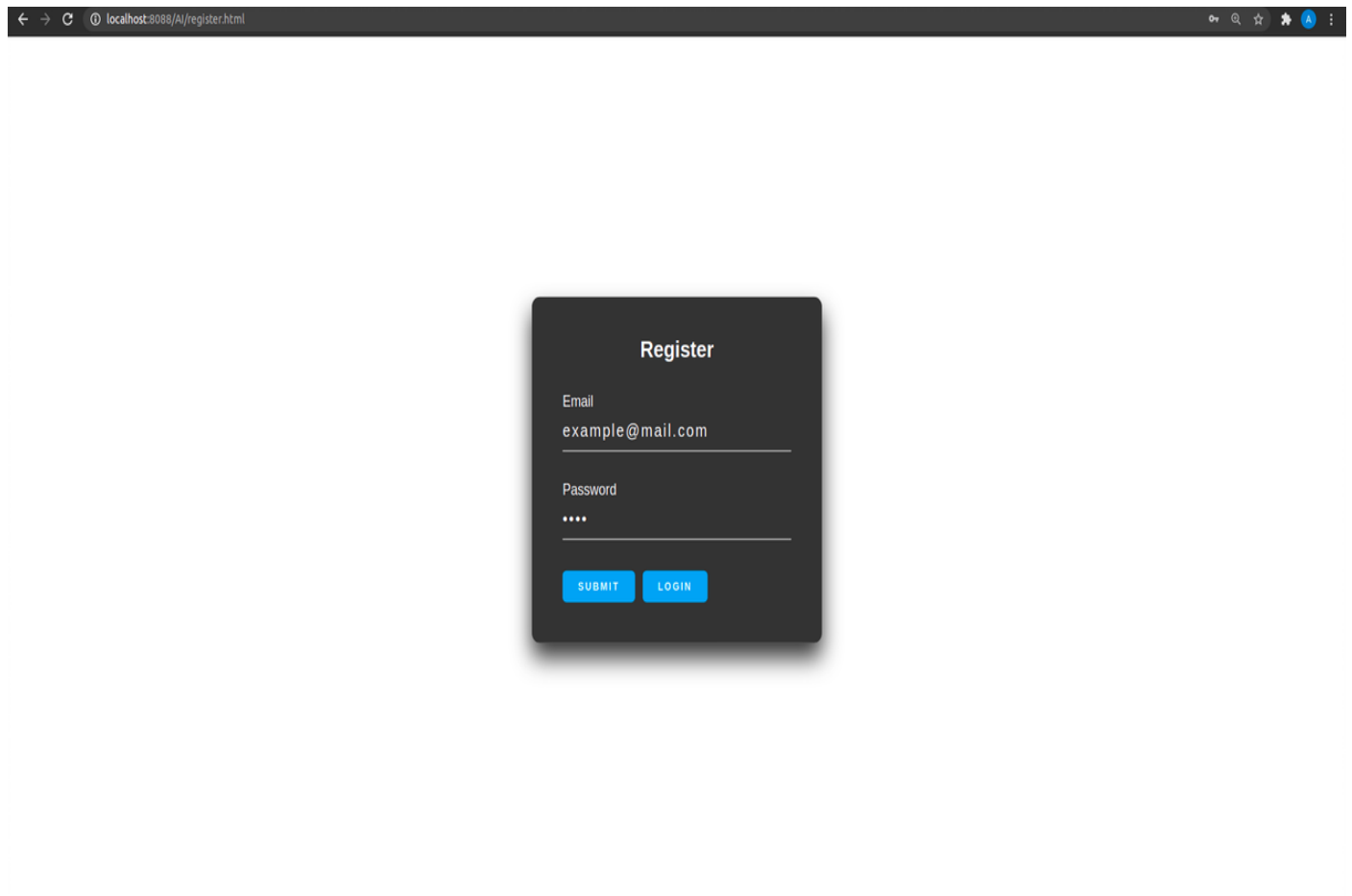
Sign in

© 2020-2020

Register

Fig:[36]User page

## 6.3 User Registration page



The screenshot shows a web browser window with the address bar displaying "localhost:8088/AI/register.html". The main content area features a dark-themed registration form titled "Register". The form includes two input fields: "Email" with the value "example@mail.com" and "Password" with four dots indicating a masked password. Below the input fields are two blue buttons labeled "SUBMIT" and "LOGIN".

Fig:[37] User Registration page

## 6.4 File upload page

Select jobs here and run them

Choose files

No file chosen

Submit

## 6.5 Jobs dashboard

Select jobs here and run them

Run job

app

Appname	Status	Download
app	1	<a href="#">Download</a>
app	1	<a href="#">Download</a>

Fig:[38] Run jobs dashboard

## 6.6 Admin dashboard

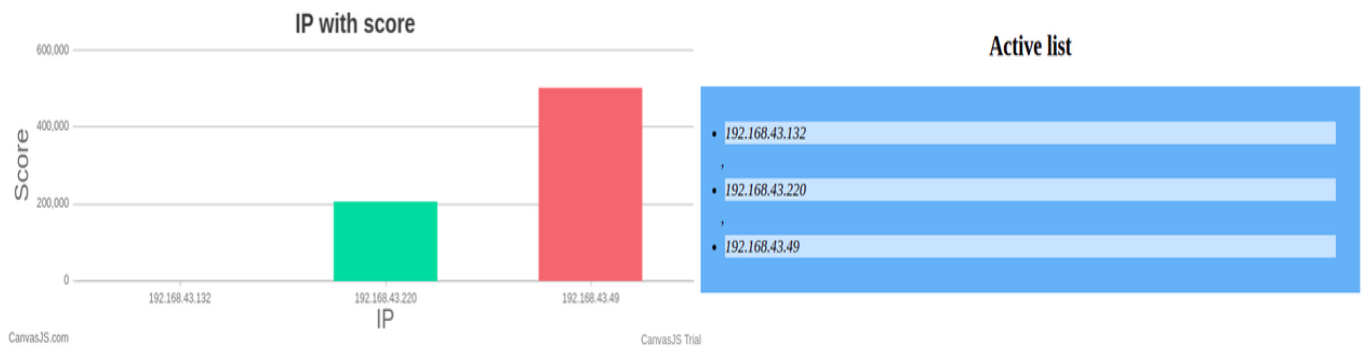


Fig:[39] Admin dashboard

## 7 Future scope and Work

### 1. **Stream data -**

Due to time constraints we were not able to add features for stream data i.e when a user will upload a ML model prediction will be done in real time or continuously on the data that is coming from iot devices.

### 2. **User defined scheduling -**

We would also like to give this feature in future which will allow users to enter custom scheduling information such as day and time etc.

### 3. **Add support for more kinds of IOT applications-**

We would like to add support for more different kinds iot applications and iot devices

## 8 Conclusion

We successfully build a web application which allows user to deploy their “Ai on edge” applications which uses iot devices data to make predictions using the model which is uploaded by user. Now there are different type of applications that belong to this category such as **Surveillance & Monitoring, Autonomous Vehicles, Traffic management etc.** Our web app will help in easily configuring, deploying and managing these kinds of applications.

As can be seen above we have used devops methodology which proves to be really efficient than it's counterparts such as agile etc. Various devops components i.e continuous integration, continuous deployment makes the task of collaborating, testing changes and deploying really easy and fast.

## 9 References

1. <https://www.rabbitmq.com/getstarted.html>
2. <https://apscheduler.readthedocs.io/en/stable/>
3. <https://docs.docker.com/engine/reference/commandline/run/>
4. [https://docs.ansible.com/ansible/latest/user\\_guide/playbooks\\_intro.html](https://docs.ansible.com/ansible/latest/user_guide/playbooks_intro.html)
5. <https://www.journaldev.com/1964/servlet-upload-file-download-example>



