International Institute of Information Technology,Bangalore
(IIIT Bangalore)

Software Production Engineering

Project Report

LeetQuiz : Open-source Quiz App

Project TA *: B. Venkatesh*

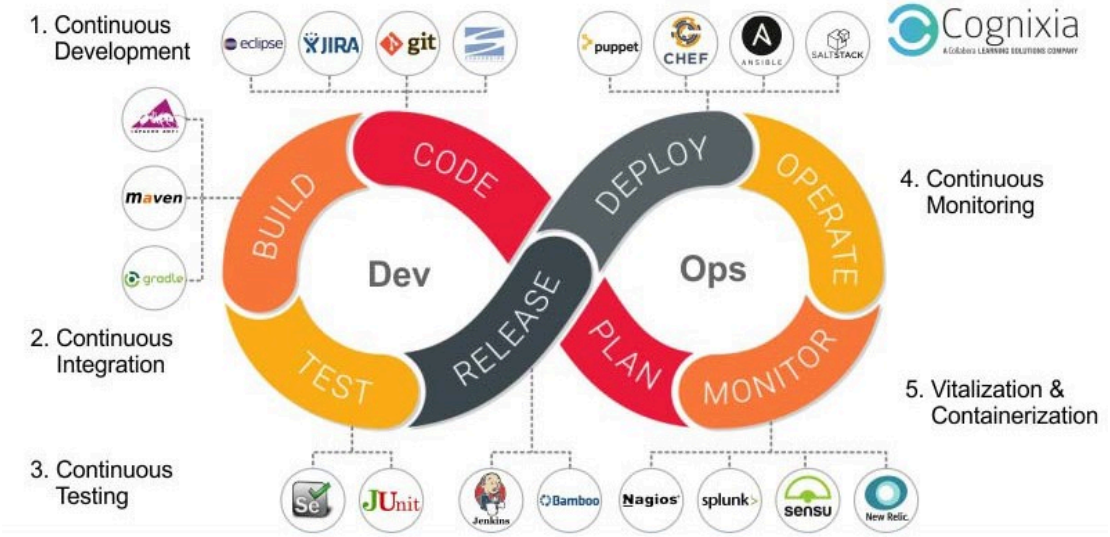
Submitted By,
B. Abhignan MT2023044
G. Sai Hemanth MT2023010

Project Overview

LeetQuiz is an open source quiz application built using micro service architecture which provides high scalability and availability using Docker and Kubernetes.

1. Introduction

- What is DevOps ?



DevOps Tools along with the DevOps Lifecycle

- DevOps is a set of practices , tools, and a cultural philosophy that automate and integrate the processes between software development and IT teams. It emphasizes team empowerment, cross-team communication and collaboration, and technology automation.
- The word DevOps is a combination of the terms development and operations meant to represent a collaborative or shared approach to the tasks performed by a company's application development and IT operations teams.
- Because of the continuous nature of DevOps, practitioners use the infinity loop to show how the phases of the DevOps lifecycle relate to each other. Despite appearing to flow sequentially, the loop symbolizes the need for constant collaboration and iterative improvement throughout the entire lifecycle.

- Why DevOps?

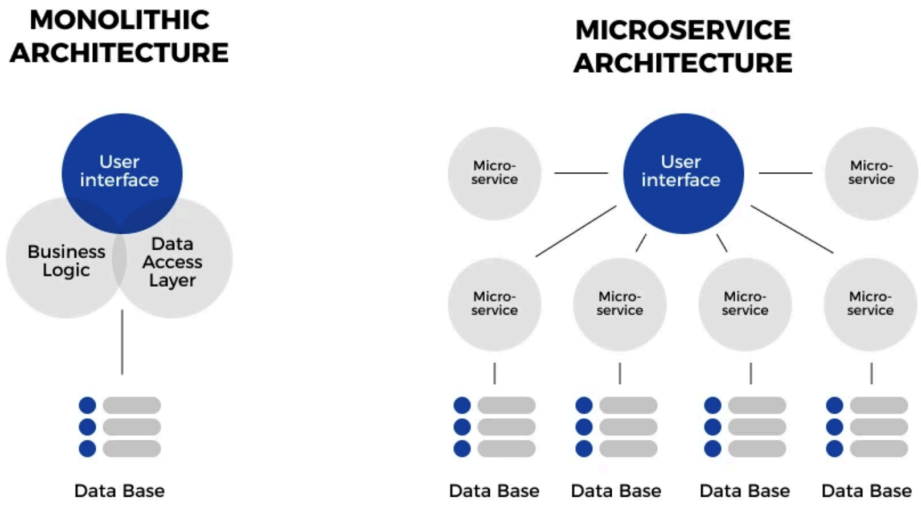
  a.Each company faces its own challenges, but common problems include releases that take too long, software that doesn't meet expectations and IT that limits business growth.Without wait times, manual processes, and lengthy reviews, a DevOps project moves faster from requirements to live software. Shorter cycle times can keep requirements from shifting so that the product delivers what customers want.
  b. DevOps solves communication and priority problems between IT specializations. To build viable software, development teams must understand the production environment and test their code in realistic conditions.
  c. Reduced deployment failures and faster time to recover: Most failures during development occurs due to programming defects. Having a DevOps team will allow for more releases in shorter time spawns. This way, it is easier and more likely to find possible defects in the code. For this same reason, and in case any problem must be solved, recovery will be quicker thanks to the knowledge and participation of all members during the development process.
  d. Improved resource management: Increased efficiency helps speed development and reduce coding defects and problems.

What are Microservices ?

Microservices architecture refers to an architectural style for developing applications. Microservices allow a large application to be separated into smaller independent parts, with each part having its own realm of responsibility. These services are owned by small, self-contained teams.

Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features.Containers are a well-suited microservices architecture example, since they let you focus on developing the services without worrying about the dependencies.

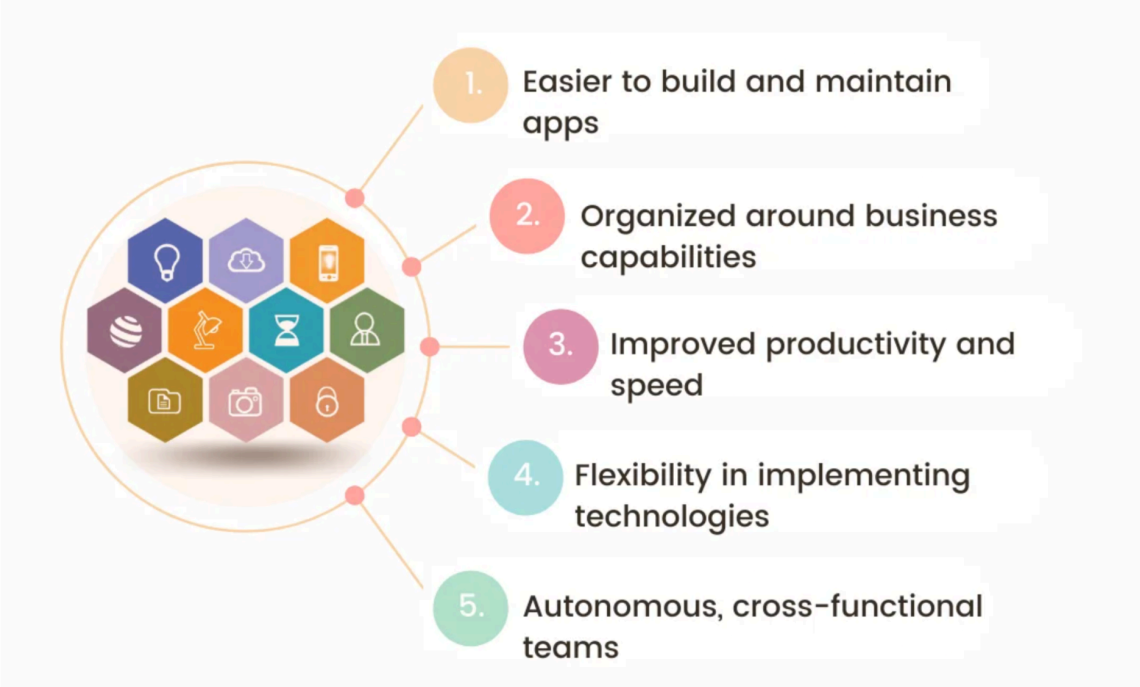
Monolithic vs. Microservices Architecture



With monolithic architectures, all processes are tightly coupled and run as a single service. This means that if one process of the application experiences a spike in demand, the entire architecture must be scaled. Monolithic architectures add risk for application availability because many dependent and tightly coupled processes increase the impact of a single process failure.

With a microservices architecture, an application is built as independent components that run each application process as a service. These services communicate via a well-defined interface using lightweight APIs. Each service performs a single function. Because they are independently run, each service can be updated, deployed, and scaled

Benefits of Microservices



- Scalability: Microservices architecture enables independent scaling of individual services based on their specific needs. This scalability granularity allows organizations to allocate resources efficiently and handle varying workloads effectively. Scaling can be achieved by adding more instances of a particular service without affecting other services, ensuring optimal resource utilization.

- Flexibility and Agility: Microservices offer flexibility in terms of development, deployment, and updates. Since services are decoupled, teams can develop and deploy services independently, without affecting other parts of the system. This modular approach allows for faster development cycles, easier maintenance, and the ability to adopt new technologies and frameworks for specific services as needed.

- Enhanced Team Autonomy: Microservices architecture promotes decentralized decision-making and empowers development teams. Each team can own and manage a specific microservice, making them responsible for its development, deployment, and maintenance. This autonomy fosters innovation, accountability, and ownership, as teams can make independent decisions and rapidly iterate on their services.

- Fault Isolation and Resilience: Microservices are isolated from each other, meaning that a failure in one service does not impact the entire system. Failures are contained within the affected service, allowing the rest of the system to continue functioning. This fault isolation improves overall system resilience and availability, as well as facilitating easier debugging and troubleshooting.

- Technology Heterogeneity: Microservices architecture supports the use of different technologies, frameworks, and programming languages for different services. This flexibility enables teams to choose the most suitable technology stack for their specific service requirements. It also allows organizations to adopt and integrate new technologies into their system without having to overhaul the entire architecture.

- Improved Maintainability and Evolvability: Microservices promote modular development and decoupling of services, making the system more maintainable and evolvable. Each service has a well-defined boundary and can be developed, tested, and updated independently. This modularity simplifies maintenance tasks, reduces the risk of unintended consequences when making changes, and enables the system to evolve more rapidly to meet changing business needs.

- Rapid Deployment and Continuous Delivery: Microservices architecture aligns well with the principles of continuous delivery and DevOps practices. The decoupled nature of microservices allows for independent deployment, testing, and release cycles. This enables organizations to deploy changes and new features more frequently, reducing time-to-market and facilitating iterative development and customer feedback loops.

- Reusability and Modularity: Microservices promote code reusability and modularity. Services can be developed as self-contained modules with well-defined APIs, making it easier to reuse them in different contexts or across multiple projects. This reusability reduces development time, improves code quality, and fosters a more efficient and scalable development process.

Limitations of microservices

Just like monolithic architecture, microservices have disadvantages too. Here are the most noteworthy:

- Increased Complexity: Microservice architecture introduces a higher level of complexity compared to monolithic architectures. Managing a large number of services, each with its

own codebase, database, and communication protocols, requires sophisticated deployment, monitoring, and management tools.

- Distributed System Challenges: As microservices are distributed across different machines and potentially different geographic locations, network latency and communication overhead become significant concerns. Ensuring reliable and efficient communication between services can be challenging, leading to performance issues.

- Service Coordination: In a microservices environment, services often need to coordinate with each other to fulfill business processes. This can be complex and require additional effort to design and implement. Ensuring consistency and data integrity across services can be challenging, as distributed transactions are difficult to achieve.

- Data Management: Microservices often have their own dedicated databases, which can lead to data duplication and redundancy. Maintaining data consistency and synchronization across multiple databases becomes more complex, requiring careful design and implementation of data management strategies.

- Operational Overhead: Managing and monitoring a large number of microservices requires advanced infrastructure and operational expertise. DevOps practices and tools are necessary to handle deployment, scaling, monitoring, and debugging of individual services, adding operational complexity and overhead.

- Deployment Challenges: Deploying microservices involves managing the deployment of multiple independent services, which can be time-consuming and error-prone. Ensuring the compatibility and versioning of different services becomes crucial to avoid compatibility issues and service disruptions.

- Testing and Debugging: Testing and debugging in a microservice architecture can be more complex than in a monolithic system. Inter-service communication, network latency, and the need for distributed testing environments make it challenging to reproduce and isolate issues.

- Learning Curve and Skills Gap: Adopting microservice architecture often requires a shift in mindset and new skill sets for development teams. It can take time and effort to understand and adapt to the distributed nature of microservices, leading to a learning curve and potential skills gap within the organization.

- Cost and Infrastructure: Deploying and managing a microservices-based system may require additional infrastructure and operational resources. The cost of maintaining

multiple services, databases, and monitoring tools can be higher compared to a monolithic architecture.

In Conclusion While microservice architecture offers numerous advantages such as scalability, flexibility, and autonomy for development teams, it is important to consider the disadvantages and limitations associated with it. The increased complexity, distributed system challenges, service coordination, and data management complexities require careful planning, implementation, and ongoing maintenance. Organizations adopting microservices should be prepared to invest in the necessary infrastructure, operational expertise, and skill development to overcome these challenges and harness the benefits of this architecture successfully.

2. Tools Used

a. **IDE :** IntelliJ & Visual Studio
b. **Testing:** JUnit
c. **Build tool:** Maven
d. **Source Code Management:** GitHub for Git.
e. **Containerization:** Docker & DockerHub
f. **Continuous Integration:** Jenkins
g. **Container Orchestration**: Kubernates
h. **Cluster** : Minikube
i. **Logging:** ELK Stack

GitHub: Helps in automation through Jenkin Integration Calculator with DevOps.
Jenkins: It is used for the Continuous Integration and Continuous Deployment portion.
Docker: It is used to make images through containerization.
WebHooks: To automate the build process whenever the developer commits the code to GitHub.
Ngrok: To convert the private IP address of the local machine to a public IP address to perform webhook.
ELK : The ELK stack is popular because it fulfills a need in the log management and analytics space.
Kubernetes : Kubernetes is an open-source container orchestration system for automating software deployment, scaling, and management.

3. Setup/Installations

a. Git & GitHub

- Git

  i. To install git open terminal and run the following commands

  $ sudo apt update

  $ sudo apt install git

  ii. Check the Git version

  $ git --version

```
billa@billa-VivoBook-ASUSLaptop-X571GT-F571GT:~$ git --version
git version 2.40.1
billa@billa-VivoBook-ASUSLaptop-X571GT-F571GT:~$
```

- GitHub :

  i. Create your own GitHub account.

  ii. Create a new public repository for source code management.

b. Ngrok

- It is used to convert the private IP address of the local machine to a public IP address to perform webhook.
- To install Ngrok perform the following steps -

1. Sign up in https://ngrok.com/
2. Download ngrok from: https://bin.equinox.io/c/4VmDzA7iaHb/ngrok-stable-linux-amd64.tgz
3. Then extract ngrok from the terminal: $sudo tar xvzf ~/Downloads/ngrok-stable-linux-amd64.tgz -C /usr/local/bin
4. Copy Authtoken from: https://dashboard.ngrok.com/get-started/your-authtoken
5. Add Authtoken: $ngrok authtoken <token>

c. Jenkins

i. To install Jenkins type the below commands in terminal -

Install Jenkins
wget -q -O - https://pkg.jenkins.io/debian-stable/jenkins.io.key | sudo apt-key add -
sudo sh -c 'echo deb http://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'

Install ca-certificates (man 8 update-ca-certificates)
sudo apt install ca-certificates

Install Jenkins
sudo apt-get update
sudo apt-get install jenkins

To check Jenkins version: vim /var/lib/jenkins/config.xml
To copy admin password: sudo cat /var/lib/jenkins/secrets/initialAdminPassword

ii. Execute https://<ipaddress>:8080
iii. copy and paste the one-time secret password

iv. Install the default plugins

v. Continue as Admin and set up the Jenkins URL and Admin credentials.

d. Docker & DockerHub

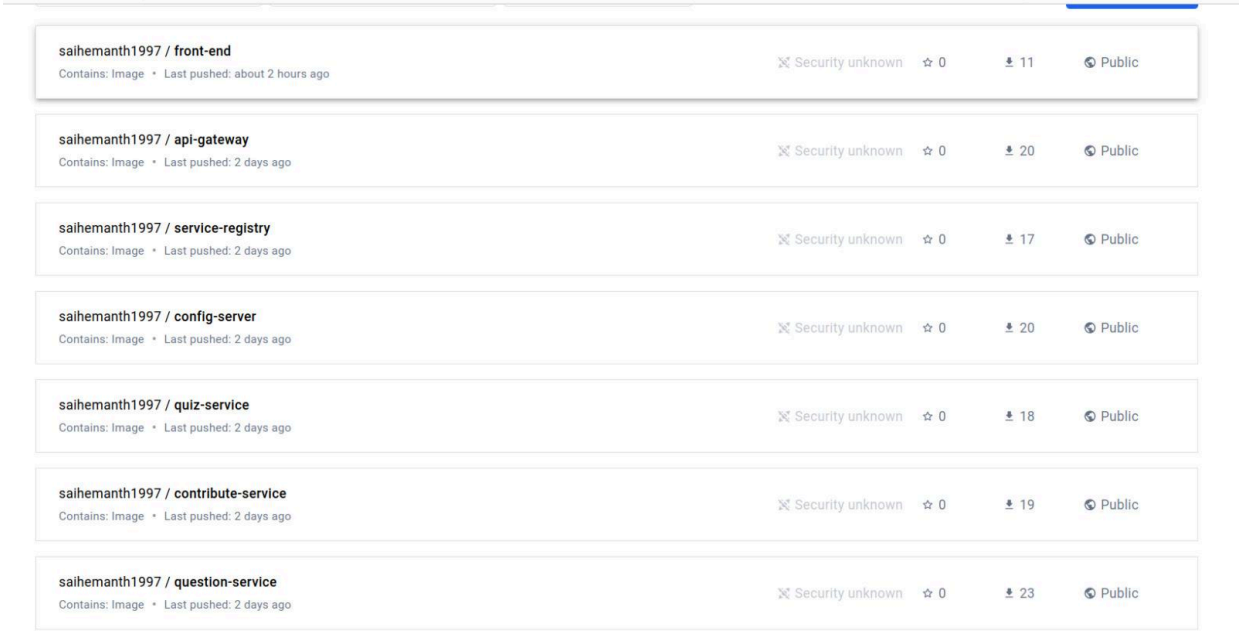
i. Docker : Install and run the docker service using below commands

$ apt-get install docker.io
$ service docker start

$ service docker status

ii. DockerHub

- Visit https://hub.docker.com/
- Signup and Login
- Create new public repository



e. Kubernetes

i. Download the latest release with the command:

$ curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

ii. Install kubectl

$ sudo install -o root -g root -m 0755 kubectl /usr/local/bin/kubectl

iii. Test to ensure the version you installed is up-to-date:

$ kubectl version --client

f. Minikube :

.To install Minikube, enter the below command

$curl -LO
https://storage.googleapis.com/minikube/releases/latest/minikube-linux-amd64

$ sudo install minikube-linux-amd64 /usr/local/bin/minikube

ii.To start minikube
$ minikube start

```
billa@billa-VivoBook-ASUSLaptop-X571GT-F571GT:~$ minikube start
😄  minikube v1.33.0 on Ubuntu 23.10
✨  Using the docker driver based on existing profile
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.43 ...
🔄  Restarting existing docker container for "minikube" ...
🎉  minikube 1.33.1 is available! Download it: https://github.com/kubernetes/minikube/
releases/tag/v1.33.1
💡  To disable this notice, run: 'minikube config set WantUpdateNotification false'

🐳  Preparing Kubernetes v1.30.0 on Docker 26.0.1 ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, default-storageclass

❗  /usr/local/bin/kubectl is version 1.19.6-eks-49a6c0, which may have incompatibilit
ies with Kubernetes 1.30.0.
    ▪ Want kubectl v1.30.0? Try 'minikube kubectl -- get pods -A'
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace
by default
```

iii. To Check status

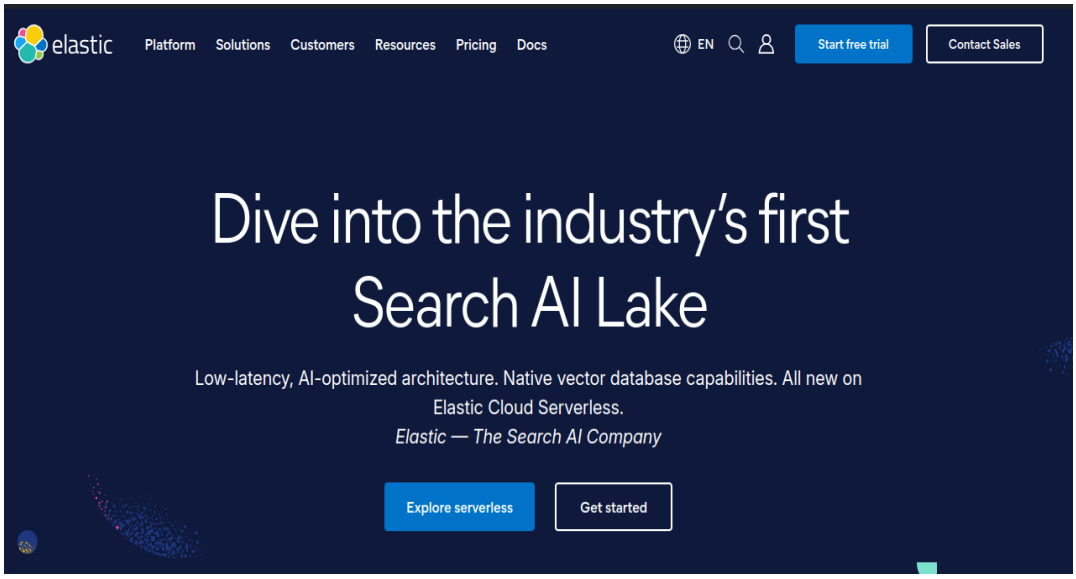
$ minikube status

```
billa@billa-VivoBook-ASUSLaptop-X571GT-F571GT:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

j. ELK Stack

i. Cloud version of ELK provides 14 days trial for the user

ii. Visit : https://www.elastic.co/ Click on Start free trial. Then you can sign in using your Google account.



5. Solution Steps

a. Write your code in React(FrontEnd) & Spring Boot (BackEnd)

b. Create a repository in DockerHub for your project

c. Create Pipeline Script in Jenkins

1. Clone Repository
2. Build Back-end Services
3. Test Back-end Services
4. Build and Test Front-end
5. Dockerize Files
6. Docker Push Images
7. Deploy to kubernetes
8. Verify the deployment

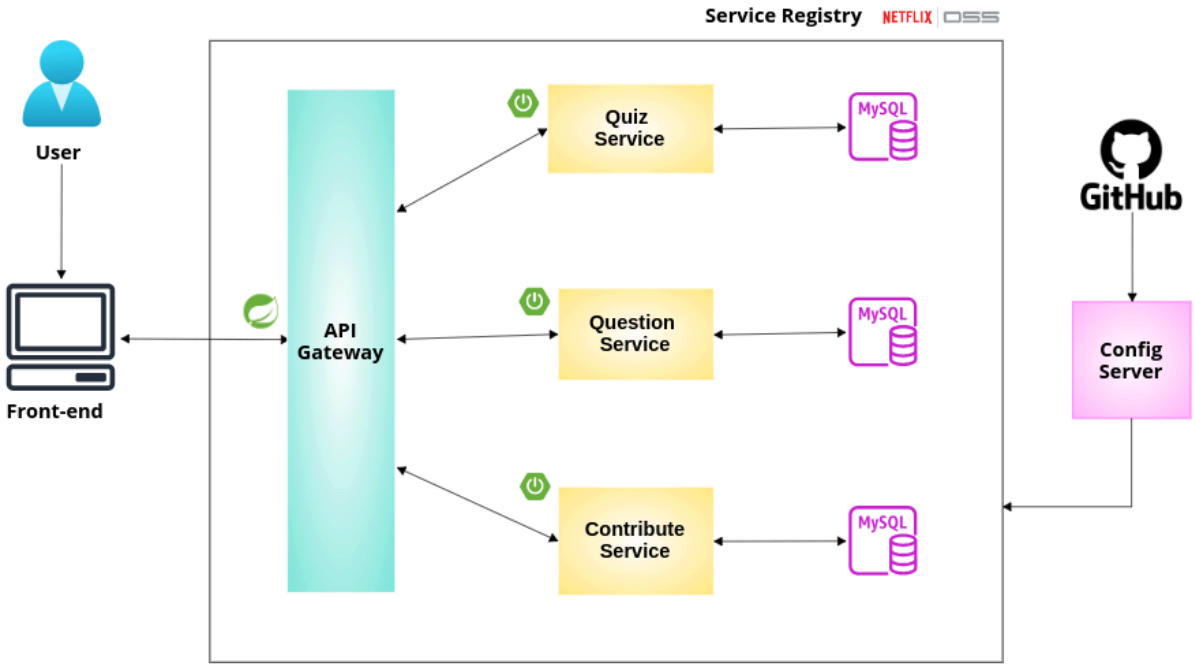
d. Monitor, Analyze and Create visualization using ELK

6. Software - Develop, Build and Test

i. Development:

- Tech Stack:
  - Frontend : React
  - Backend: SpringBoot
  - Database: MySQL

- The application 7 microservices:
    - Question : For handling questions
    - Quiz : For handling quizzes
    - Contribute : For handling open-source contributions
    - Service-Registry : For registering all services
    - Api-Gateway : To handle all the relative routings to individual services
    - Cloud-config Server : For handling centralized cloud configurations



ii. Testing : To employ JUnit for testing Spring Boot applications, first, integrate the necessary dependencies into your project configuration using Maven or Gradle. Organize your test files within a dedicated directory, typically named src/test/java, adhering to the *Test.java naming convention. Utilize annotations like @SpringBootTest or @WebMvcTest provided by Spring Boot to facilitate testing of different components. Within your test methods, implement assertions and logic to verify the behavior of your Spring components. Execute your tests using your preferred build tool, such as Maven or Gradle. JUnit will then generate detailed reports summarizing the test results, aiding in identifying any issues or failures. By following these steps, you can ensure the reliability and correctness of your Spring Boot application through comprehensive testing practices. Run **mvn test** to run the tests.

```java
class QuestionServiceTest {

    6 usages
    @Mock
    private QuestionDao questionDao;

    5 usages
    @InjectMocks
    private QuestionService questionService;

    10 usages
    private List<Question> testQuestions;

    saihemath2000
    @BeforeEach
    void setUp() {
        testQuestions = Arrays.asList(
                new Question( i: 1,  s: "Question 1",  s1: "Option 1",  s2: "Option 2",  s3: "Option 3",  s4: "Option 4",  s5: "Option 1"),
                new Question( i: 2,  s: "Question 2",  s1: "Option A",  s2: "Option B",  s3: "Option C",  s4: "Option D",  s5: "Option C")
        );
    }

    saihemath2000
    @Test
    void testGetAllQuestions() {
        when(questionDao.findAll()).thenReturn(testQuestions);

        ResponseEntity<List<Question>> response = questionService.getAllQuestions();

        assertEquals(HttpStatus.OK, response.getStatusCode());
        assertEquals(testQuestions, response.getBody());
```

iii. Build : mvn clean package for building the project

```
❯ mvn clean package
[INFO] Scanning for projects...
[INFO]
[INFO] --------------------< com.example:question-service >--------------------
[INFO] Building question-service 0.0.1
[INFO] --------------------------------[ jar ]---------------------------------
[INFO]
[INFO] --- maven-clean-plugin:3.2.0:clean (default-clean) @ question-service ---
[INFO] Deleting /home/hemanth/Desktop/iiitb/SPE/SPEMAJORPROJECT/question-service/target
[INFO]
[INFO] --- maven-resources-plugin:3.3.1:resources (default-resources) @ question-service ---
[INFO] Copying 2 resources from src/main/resources to target/classes
[INFO] Copying 2 resources from src/main/resources to target/classes
[INFO]
[INFO] --- maven-compiler-plugin:3.11.0:compile (default-compile) @ question-service ---
[INFO] Changes detected - recompiling the module! :source
[INFO] Compiling 7 source files with javac [debug release 17] to target/classes
[INFO]
[INFO] --- maven-resources-plugin:3.3.1:testResources (default-testResources) @ question-service ---
[INFO] skip non existing resourceDirectory /home/hemanth/Desktop/iiitb/SPE/SPEMAJORPROJECT/question-service/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.11.0:testCompile (default-testCompile) @ question-service ---
[INFO] Changes detected - recompiling the module! :dependency
[INFO] Compiling 2 source files with javac [debug release 17] to target/test-classes
[INFO]
[INFO] --- maven-surefire-plugin:3.0.0:test (default-test) @ question-service ---
[INFO] Using auto detected provider org.apache.maven.surefire.junitplatform.JUnitPlatformProvider
[INFO]
```

```
[INFO]
[INFO]  ---> 653f74eb632f
[INFO] Step 4/6 : WORKDIR /app
[INFO]
[INFO]  ---> Running in a76403f6d067
[INFO]  ---> Removed intermediate container a76403f6d067
[INFO]  ---> 26d6e5ee1059
[INFO] Step 5/6 : ENTRYPOINT ["java","-jar","question-service-0.0.1.jar"]
[INFO]
[INFO]  ---> Running in 0e7e36ca8215
[INFO]  ---> Removed intermediate container 0e7e36ca8215
[INFO]  ---> ce98babcf757
[INFO] Step 6/6 : EXPOSE 8080
[INFO]
[INFO]  ---> Running in 40501317ccb2
[INFO]  ---> Removed intermediate container 40501317ccb2
[INFO]  ---> 4ba2a7575e16
[INFO] Successfully built 4ba2a7575e16
[INFO] Successfully tagged saihemanth1997/question-service:0.0.1
[INFO]
[INFO] Detected build of image with id 4ba2a7575e16
[INFO] Building jar: /home/hemanth/Desktop/iiitb/SPE/SPEMAJORPROJECT/question-service/target/question-service-0.0.1-docker-info.jar
[INFO] Successfully built saihemanth1997/question-service:0.0.1
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  23.687 s
[INFO] Finished at: 2024-05-24T20:14:55+05:30
```

iv. Output:

We can check if the deployments are running by the following

$kubectl apply -f .

Applies to all the deployment files

To view all deployments :

$kubectl get all

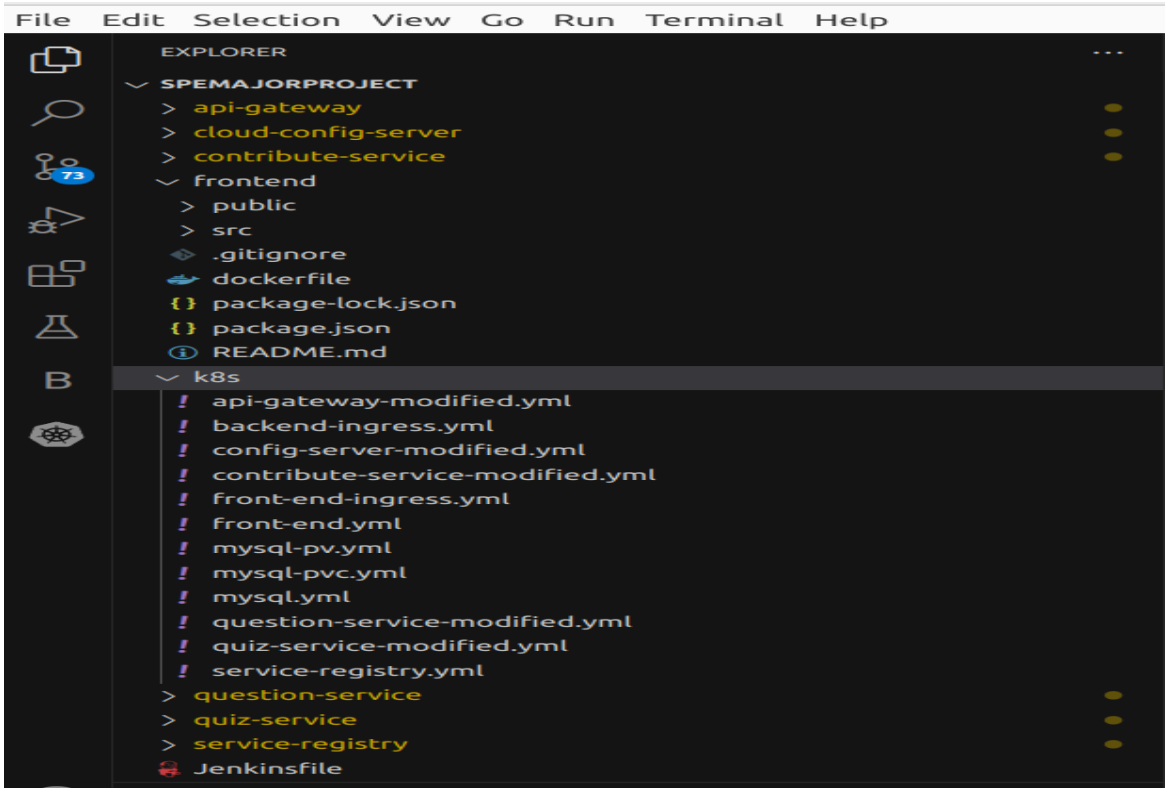
```
❯ kubectl apply -f .
deployment.apps/api-gateway-app created
service/api-gateway-app created
configmap/api-gateway-config unchanged
deployment.apps/config-server-app created
service/config-server-app created
configmap/config-server-config unchanged
deployment.apps/contribute-service-app created
service/contribute-service-app created
configmap/contribute-service-config unchanged
deployment.apps/frontend-deployment created
service/frontend-svc created
persistentvolume/mysql-pv-volume unchanged
persistentvolumeclaim/mysql-pv-claim unchanged
service/mysql-svc created
deployment.apps/mysql created
deployment.apps/question-service-app created
service/question-service-app created
configmap/question-service-config unchanged
deployment.apps/quiz-service-app created
service/quiz-service-app created
configmap/quiz-service-config unchanged
configmap/eureka-cm unchanged
service/eureka created
statefulset.apps/eureka unchanged
service/eureka-lb created
```

```
~/Desktop/iiitb/k8s                                                    ✔ ( 08:23:
❯ kubectl get all
NAME                                          READY   STATUS    RESTARTS        AGE
pod/api-gateway-app-687b845747-h76hk          1/1     Running   0               34s
pod/config-server-app-7fb788dbdb-8b2qs        1/1     Running   0               34s
pod/contribute-service-app-8d64498bd-8glkg    1/1     Running   0               34s
pod/eureka-0                                  1/1     Running   4 (2m18s ago)   21h
pod/frontend-deployment-7874bfc989-wvjfz      1/1     Running   0               34s
pod/mysql-6554db7784-k8qqb                    1/1     Running   0               34s
pod/question-service-app-746b7447b9-4mf4p     1/1     Running   0               34s
pod/quiz-service-app-fc7dd5cdd-w277f          1/1     Running   0               34s

NAME                              TYPE           CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
service/api-gateway-app           ClusterIP      10.101.34.99     <none>        80/TCP           34s
service/config-server-app         ClusterIP      10.105.2.140     <none>        80/TCP           34s
service/contribute-service-app    ClusterIP      10.98.169.245    <none>        80/TCP           34s
service/eureka                    ClusterIP      None             <none>        8761/TCP         34s
service/eureka-lb                 LoadBalancer   10.111.44.181    <pending>     80:31144/TCP     34s
service/frontend-svc              NodePort       10.104.111.242   <none>        3000:31804/TCP   34s
service/kubernetes                ClusterIP      10.96.0.1        <none>        443/TCP          88s
service/mysql-svc                 ClusterIP      10.105.113.126   <none>        3306/TCP         34s
service/question-service-app      ClusterIP      10.107.188.29    <none>        80/TCP           34s
service/quiz-service-app          ClusterIP      10.110.206.11    <none>        80/TCP           34s

NAME                                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/api-gateway-app            1/1     1            1           34s
deployment.apps/config-server-app          1/1     1            1           34s
deployment.apps/contribute-service-app     1/1     1            1           34s
deployment.apps/frontend-deployment        1/1     1            1           34s
deployment.apps/mysql                      1/1     1            1           34s
deployment.apps/question-service-app       1/1     1            1           34s
```

v. Folder Structure:



7. Source Code Management

- Git & GitHub is used for Source Code Management.
- Source code management (SCM) is used to track modifications to a source code repository. SCM tracks a running history of changes to a code base and helps resolve conflicts when merging updates from multiple contributors. SCM is also synonymous with Version control.

- Install git as mentioned earlier.
- In this step, we create a repository on GitHub and then configure the maven project on our local machine and link it to the Git repository.

$ git init

Open the command line on the root of the Maven project and type the above command to initialize the Maven project as a Git repository.

$ git remote add origin https://github.com/saihemath2000/SPEMAJOR.git

This command helps to link the Git repository with the local repository.

$ git add .

This command is to Stage the Maven project in the Git environment. ("." command specifies to stage all the files in the current folder that are modified or created.)

$git commit -m "Commit Message "

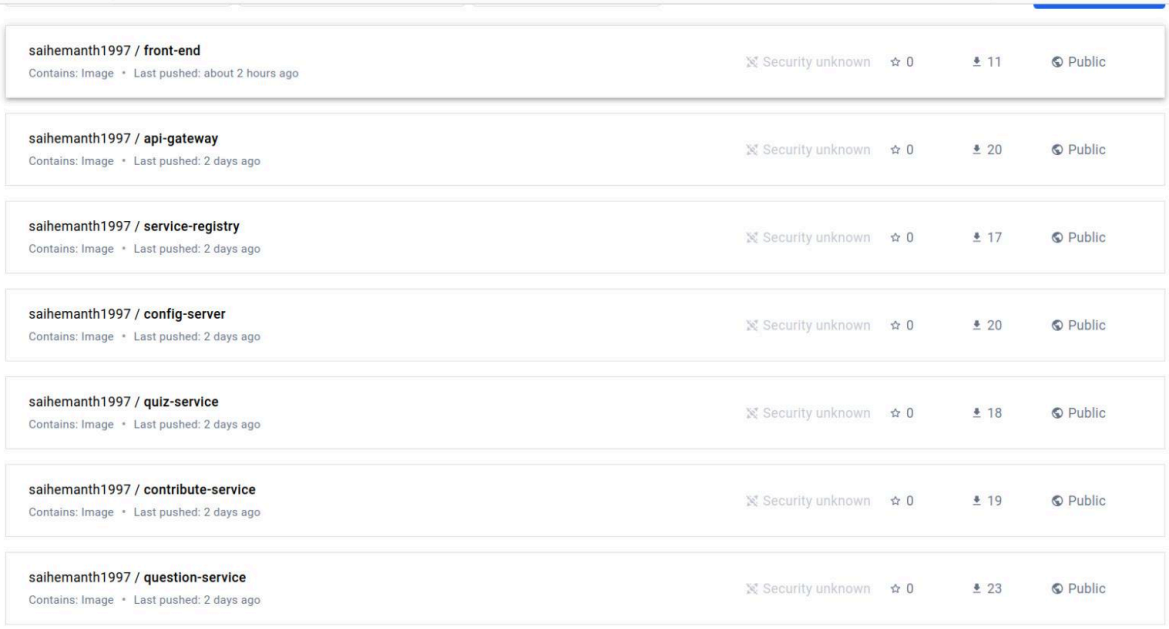
This command is to commit the staged files to GitHub.

$ git push -u origin master

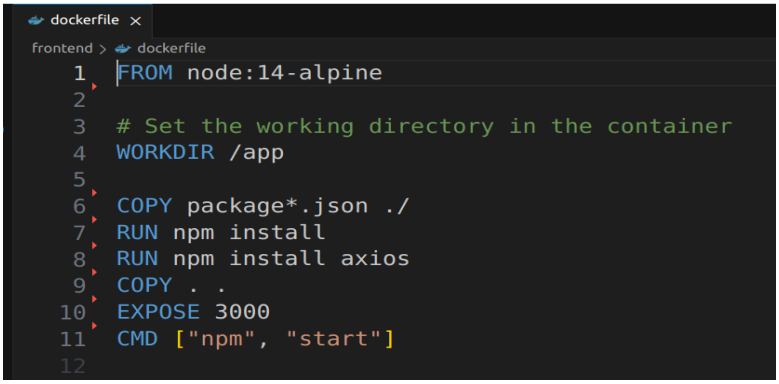
This command is to push the changes to the GitHub repository.

8. Containerization

- Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure the same ways you manage your applications.
- Create a repository in DockerHub & Install docker as mentioned earlier



- **Create a Dockerfile**: After creating a Dockerfile, push the changes to GitHub.



- **Add Jenkins user and localhost to docker group :**

  This is done so that jenkins can run the docker without changing permission each time or switching as root user.

  $ sudo usermod -aG docker jenkins

9. Continuous Integration

● Jenkins is a powerful application that allows continuous integration and continuous delivery of projects, regardless of the platform you are working on. It is a free source that can handle any kind of build or continuous integration. Youcanintegrate Jenkins with a number of testing and deployment technologies.

● **Add Jenkins to the Docker group**

$ sudo apt install openssh-server

$ sudo su - jenkins

● By executing the above commands, we get logged into jenkins. Here we configureJenkins to use Docker images via ssh.

$ mkdir .ssh

$ cd .ssh

$ ssh-keygen -t rsa

$ ssh-copy-id sai@localhost

$ ssh sai@localhost

● After executing the last command, we get automatically directed outside the Jenkins user. We can now start Jenkins with the command:

$ sudo systemctl start jenkins

● Jenkins starts at port number 8080 so we login on to http://localhost:8080 on to the browser

● **Prerequisites for Jenkins:**

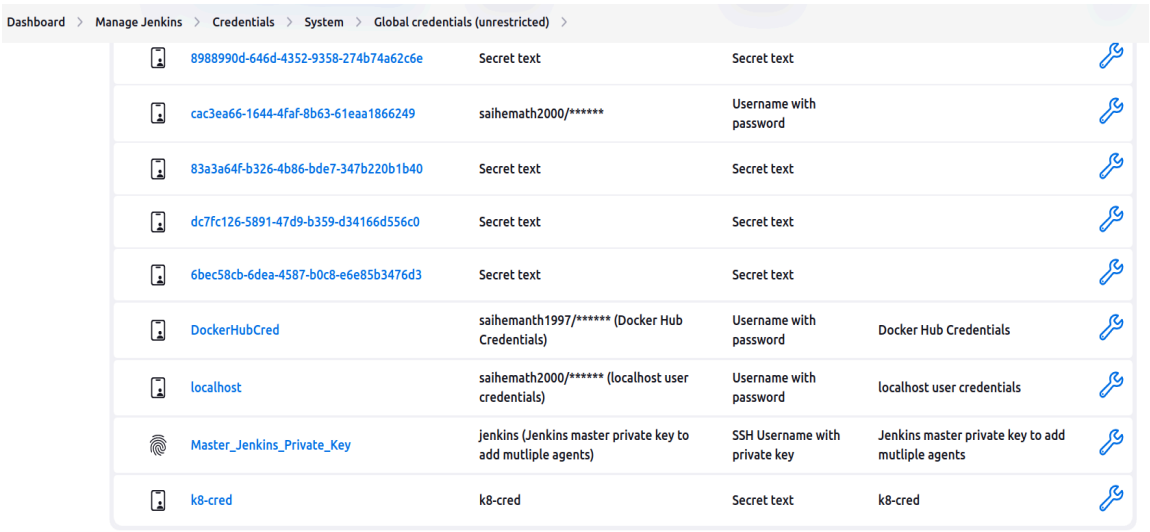
We will need a variety of plugins to utilize Jenkins Pipeline. Install following things from Plugin Manager

- Git
- Docker and Kubernetes

● Set up global configurations : Manage Jenkins -> Global Tool Configuration

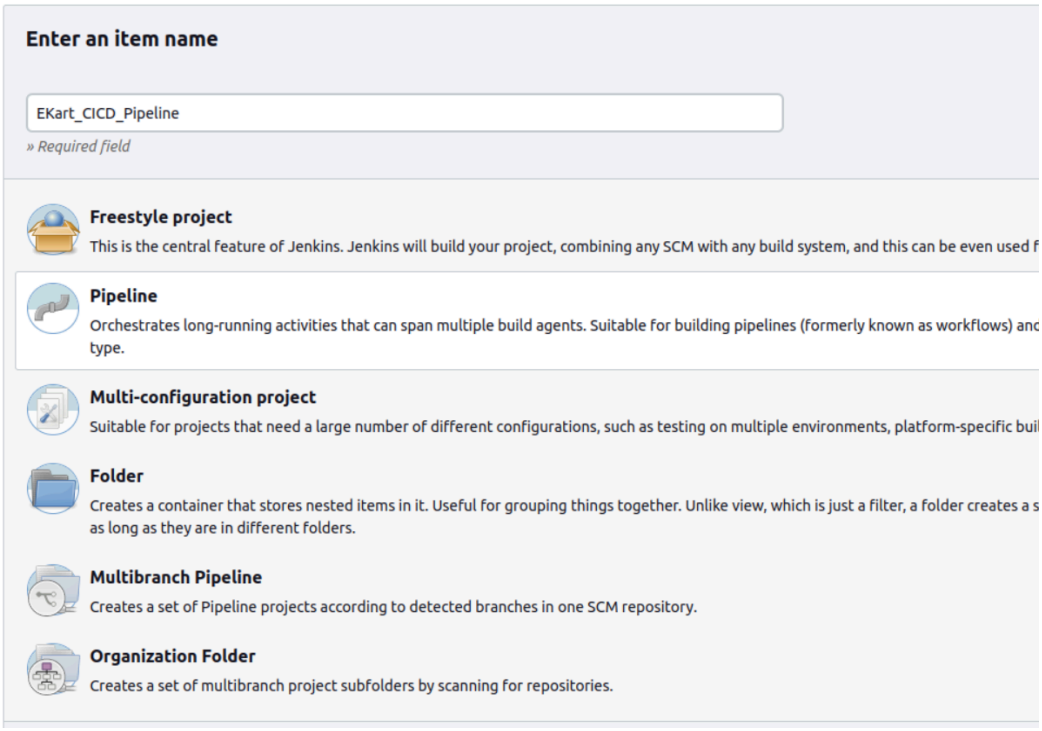
● Add Docker credentials:In the Jenkins dashboard we add credentials to theDockerHub repository and we set an unique id which is equal to docker with Registry Credentials id in pipeline script.

● Goto Manage Jenkins->Manage Credentials->Global credential->Add credentia



- **Create a New Pipeline in Jenkins**

  - Create new item, and select pipeline



Setup a Jenkins Pipeline Write a Pipeline Script which includes steps likeGitClone, Building a Docker image, Pushing the docker image to docker hub and Ansible Deployment. After we are done setting up every aspect of our DevOps tool chain, we may now build the Jenkins job.

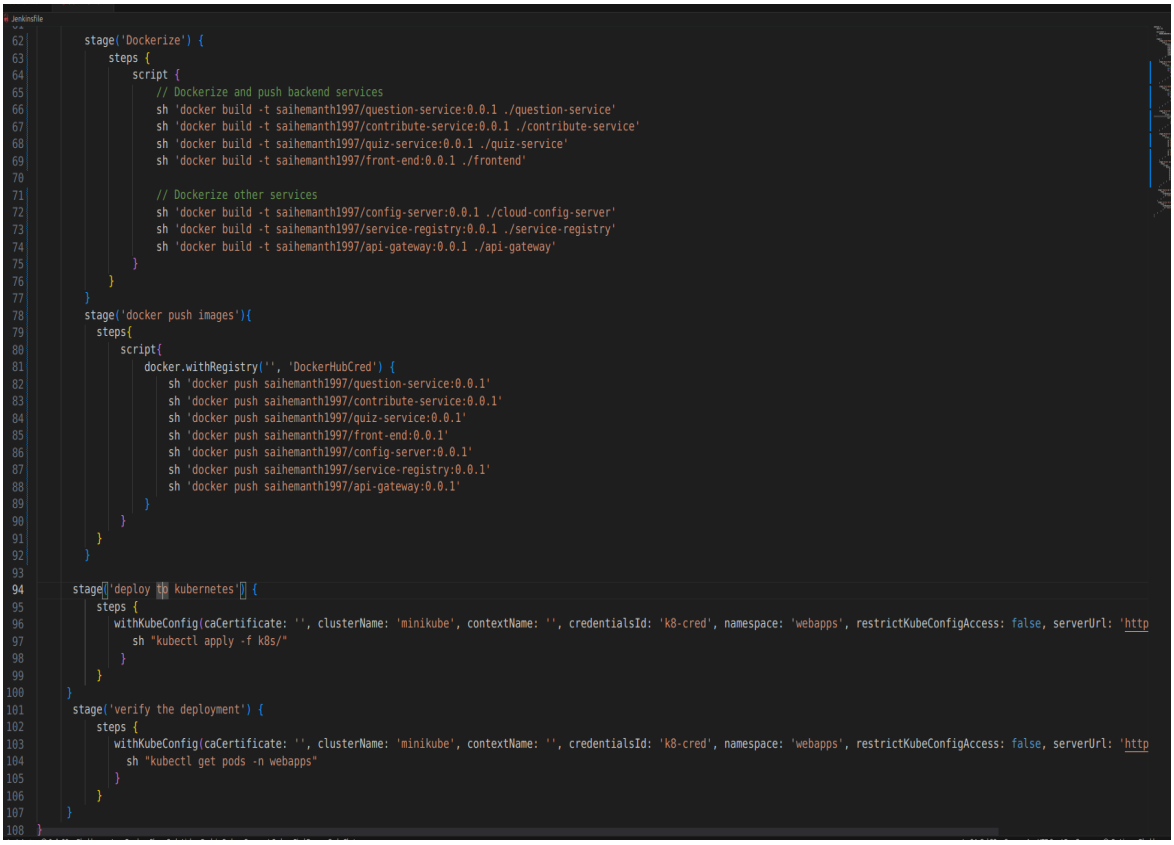
Jenkinsfile

```
pipeline {
    agent any

    environment {
        DOCKER_REGISTRY_CREDENTIALS = 'DockerHubCred'
    }

    stages {
        stage('Clone Repository') {
            steps {
                checkout([
                    $class: 'GitSCM',
                    branches: [[name: '*/master']],
                    doGenerateSubmoduleConfigurations: false,
                    extensions: [[$class: 'CloneOption', timeout: 15]],
                    submoduleCfg: [],
                    userRemoteConfigs: [[url: 'https://github.com/saihemath2000/SPEMAJOR.git']]
                ])
            }
        }
        stage('Build Backend Services') {
            steps {
                script {
                    // Build each Spring Boot service
                    def services = ['question-service', 'contribute-service', 'quiz-service','service-registry','api-gateway','cloud-config-server']
                    for (service in services) {
                        dir(service) {
                            sh "mvn clean package"
                        }
                    }
                }
            }
        }

        stage('Test Backend Services') {
            steps {
                script {
                    // Test each Spring Boot service
                    def services = ['question-service', 'contribute-service', 'quiz-service','service-registry','api-gateway','cloud-config-server']
                    for (service in services) {
                        dir(service) {
                            sh "mvn test"
                        }
                    }
                }
            }
        }
```



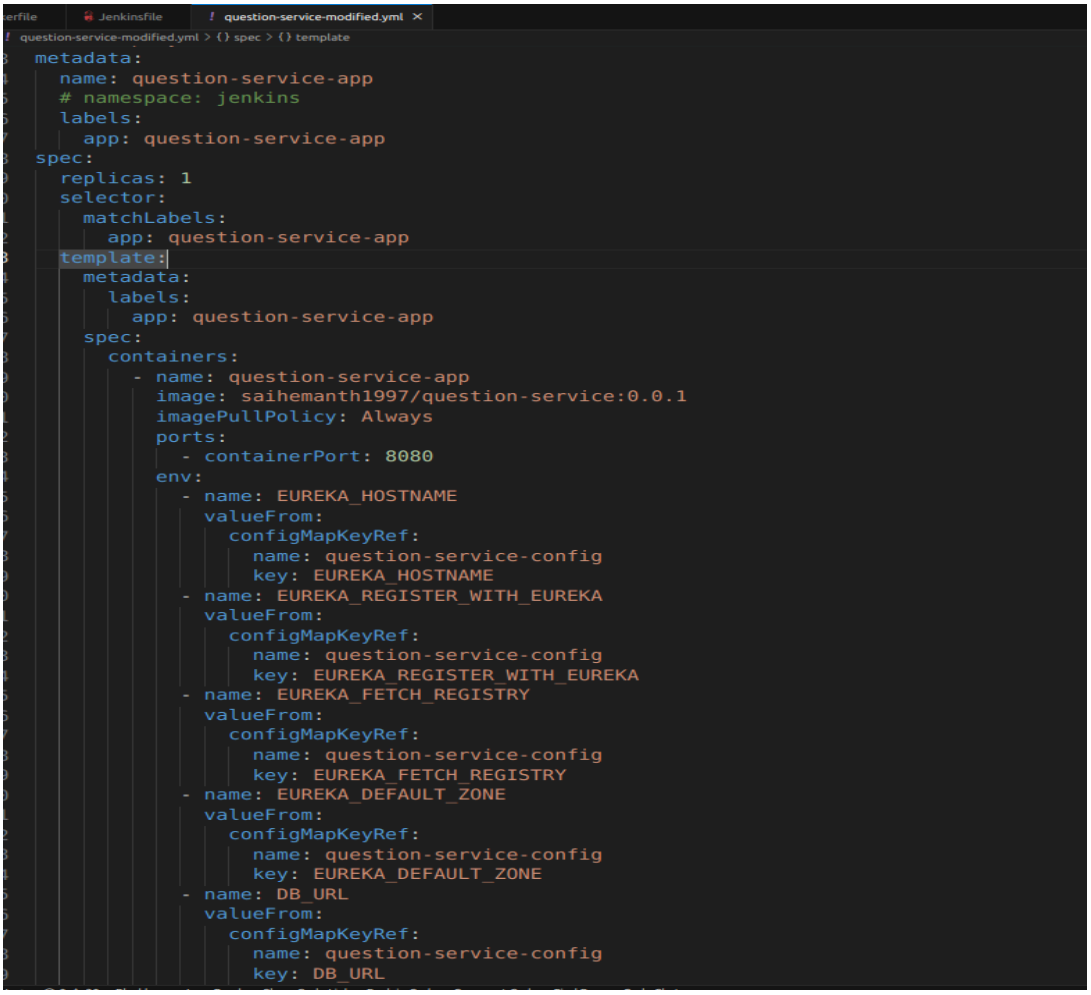
```
Jenkinsfile
62          stage('Dockerize') {
63              steps {
64                  script {
65                      // Dockerize and push backend services
66                      sh 'docker build -t saihemanth1997/question-service:0.0.1 ./question-service'
67                      sh 'docker build -t saihemanth1997/contribute-service:0.0.1 ./contribute-service'
68                      sh 'docker build -t saihemanth1997/quiz-service:0.0.1 ./quiz-service'
69                      sh 'docker build -t saihemanth1997/front-end:0.0.1 ./frontend'
70
71                      // Dockerize other services
72                      sh 'docker build -t saihemanth1997/config-server:0.0.1 ./cloud-config-server'
73                      sh 'docker build -t saihemanth1997/service-registry:0.0.1 ./service-registry'
74                      sh 'docker build -t saihemanth1997/api-gateway:0.0.1 ./api-gateway'
75                  }
76              }
77          }
78          stage('docker push images'){
79            steps{
80                script{
81                    docker.withRegistry('', 'DockerHubCred') {
82                        sh 'docker push saihemanth1997/question-service:0.0.1'
83                        sh 'docker push saihemanth1997/contribute-service:0.0.1'
84                        sh 'docker push saihemanth1997/quiz-service:0.0.1'
85                        sh 'docker push saihemanth1997/front-end:0.0.1'
86                        sh 'docker push saihemanth1997/config-server:0.0.1'
87                        sh 'docker push saihemanth1997/service-registry:0.0.1'
88                        sh 'docker push saihemanth1997/api-gateway:0.0.1'
89                    }
90                }
91            }
92          }
93
94        stage('deploy to kubernetes') {
95            steps {
96              withKubeConfig(caCertificate: '', clusterName: 'minikube', contextName: '', credentialsId: 'k8-cred', namespace: 'webapps', restrictKubeConfigAccess: false, serverUrl: 'http
97                 sh "kubectl apply -f k8s/"
98              }
99            }
100       }
101       stage('verify the deployment') {
102           steps {
103             withKubeConfig(caCertificate: '', clusterName: 'minikube', contextName: '', credentialsId: 'k8-cred', namespace: 'webapps', restrictKubeConfigAccess: false, serverUrl: 'http
104               sh "kubectl get pods -n webapps"
105             }
106           }
107       }
108   }
```

10. Continuous Deployment

Kubernetes (k8s) provides powerful container orchestration capabilities, allowing users to automate the deployment, scaling, and management of containerized applications. With Kubernetes, deployment becomes more streamlined and efficient as it automates tasks like container scheduling, load balancing, and health monitoring.

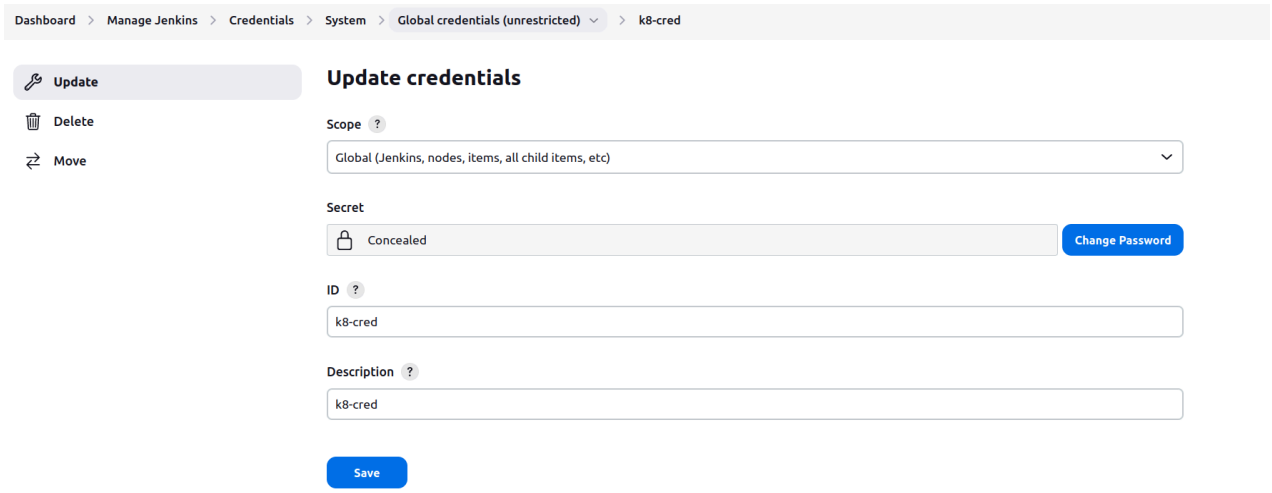
I. **Create a Kubernetes Deployment YAML**:

Within our project directory, we create a YAML file defining the Kubernetes deployment.



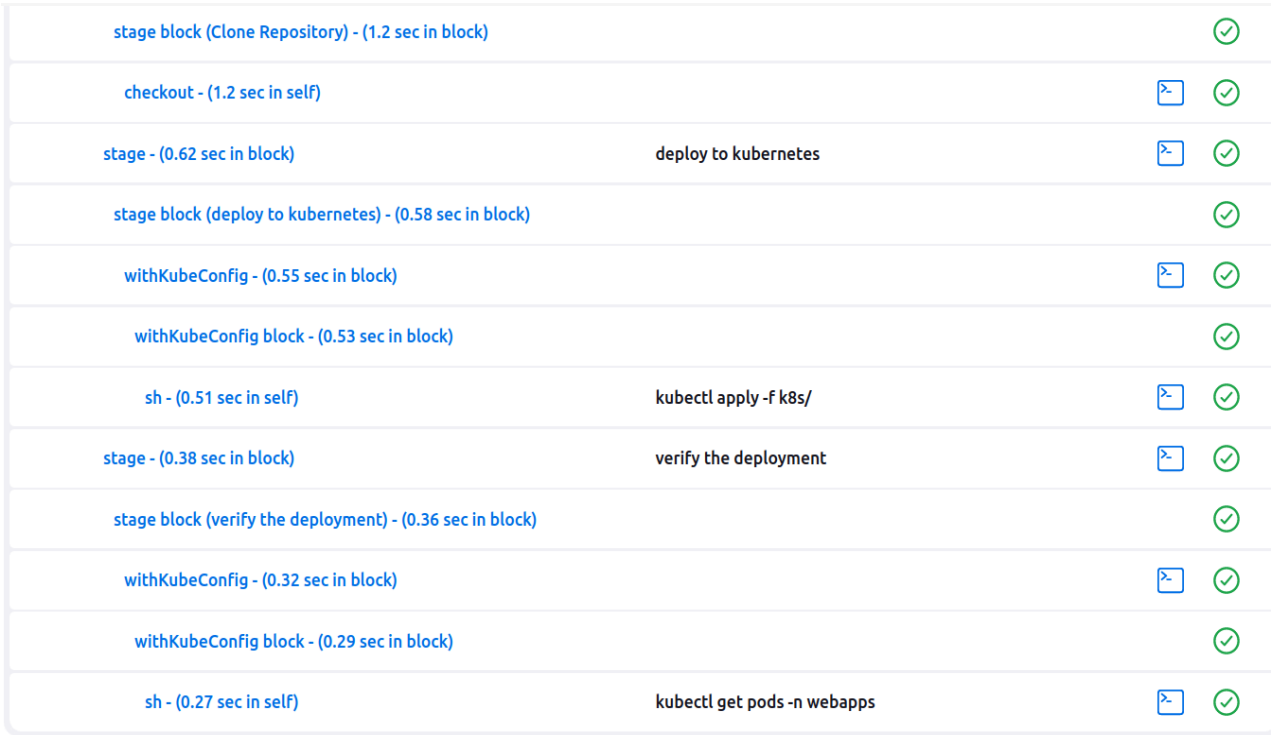
II. **Configure Kubernetes in Jenkins**:

Navigate to Jenkins -> Dashboard -> Manage Jenkins -> Global Tool Configuration. Add Kubernetes as a tool or configure Kubernetes Cloud if using Jenkins Kubernetes plugin. Specify Kubernetes credentials and connection details such as server URL and cluster certificate.



III. **Kubernetes Stage in the Jenkins Pipeline Script**:

Define a stage in the Jenkins pipeline script dedicated to Kubernetes deployment. Use Kubernetes plugin or Kubernetes CLI (kubectl) commands within the pipeline stage to interact with the Kubernetes cluster. Tasks in this stage may include deploying the application, updating configurations, scaling replicas, etc.



11. Build Pipeline & Run Docker Image

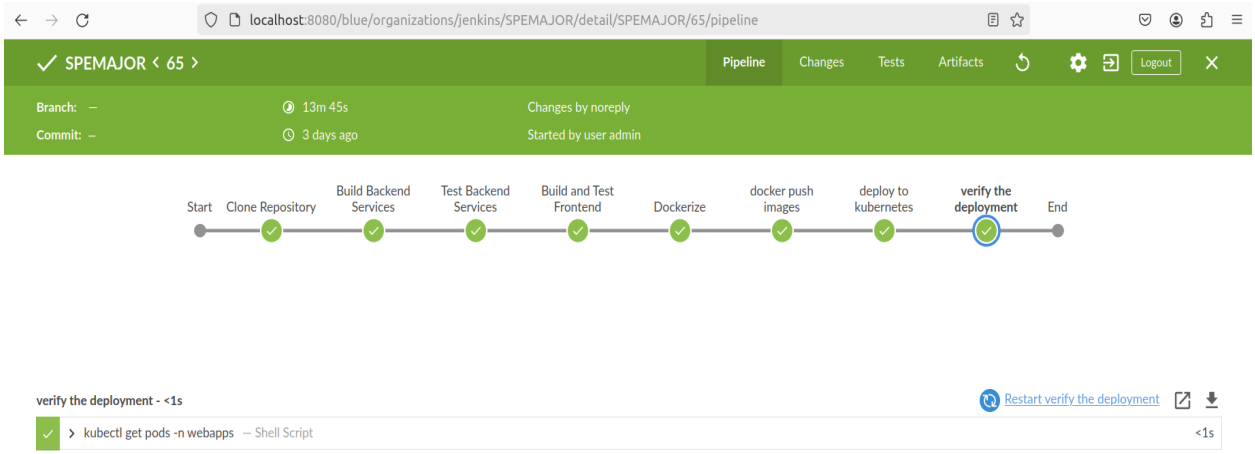
- It can be triggered based on events such as new push or new pull but for GitHub might have set up a webhook.
- This can be done via port forwarding; we have to make jenkins port 8080 open wide internet so it can be triggered based on events using Ngrok and webhooks. But Initially we perform this job manually.

- **Ngrok**:

To convert the private IP address of the local machine to a public IP address to perform webhook.
Type the command **$ngrok http 8080** in the terminal. This gives a public IP address which can be then used along with webhooks.

- **Webhooks :**
  This is used to automate the build process whenever the developer commits the code to GitHub.
  - Copy the public IP address generated by ngrok and paste it as payloadURLof webhook of the github repository that is associated with the project. - Also goto Jenkins -> Manage jenkins ->Configure Systems-> Jenkin Location->paste the public IP in Jenkins URL.
  - Then check the GitHub hook trigger for the GITSCM polling option in Build trigger of the job.

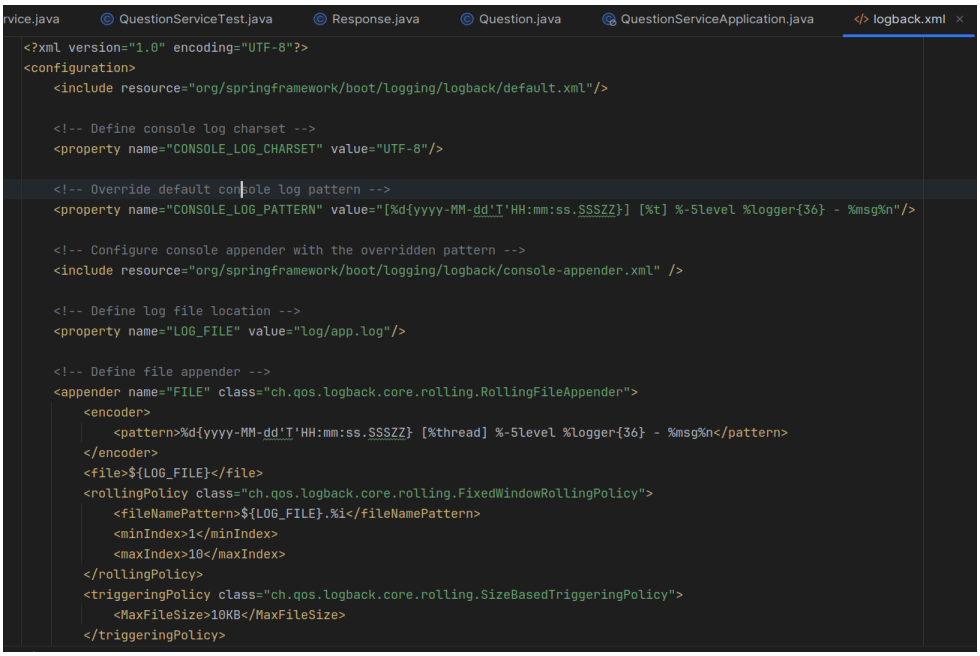
- Whenever build now is clicked or any changes committed to the repo the pipeline script is triggered.



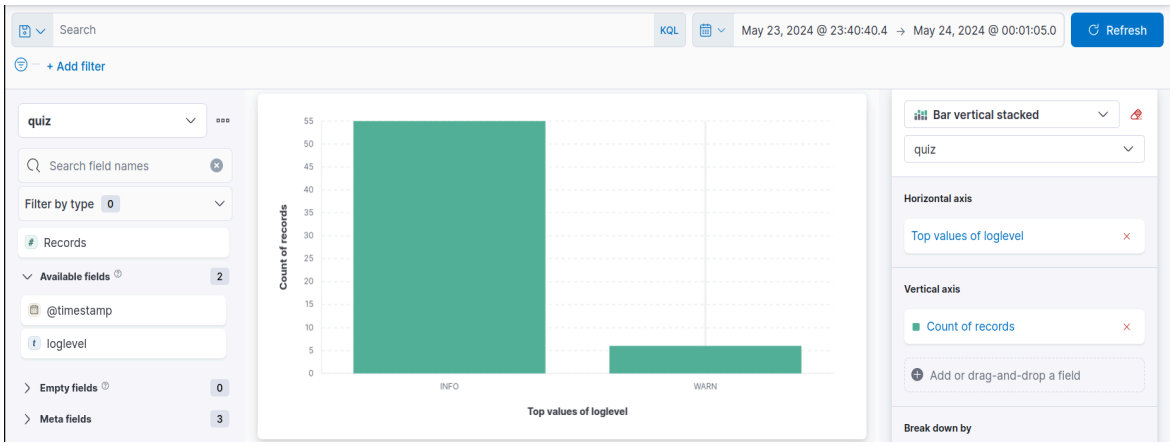
- There are 6 stages in my Jenkins pipeline:
    1. Git Clone
    2. Build project using mvn
    2. Build Docker images
    3. Push Docker Image to Hub
    4. Deploy to kubernetes
    5. Verify the deployment
- After doing all this, the docker image is pulled on to the local machine. We can run this image on the local machine to generate logs which act as input to the ELKmonitoring.

12. ELK LOGS:

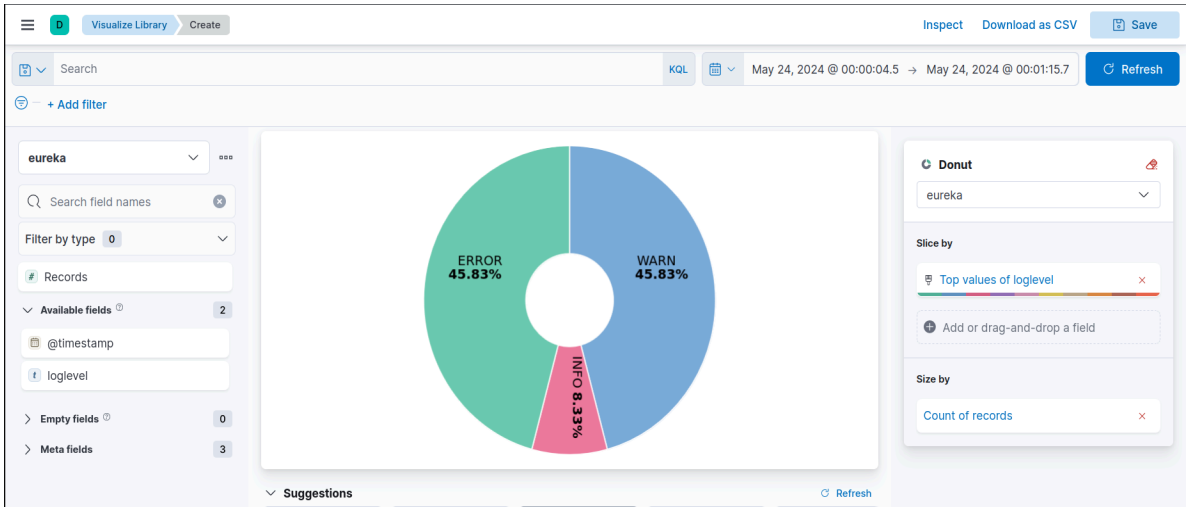
1. Logfile



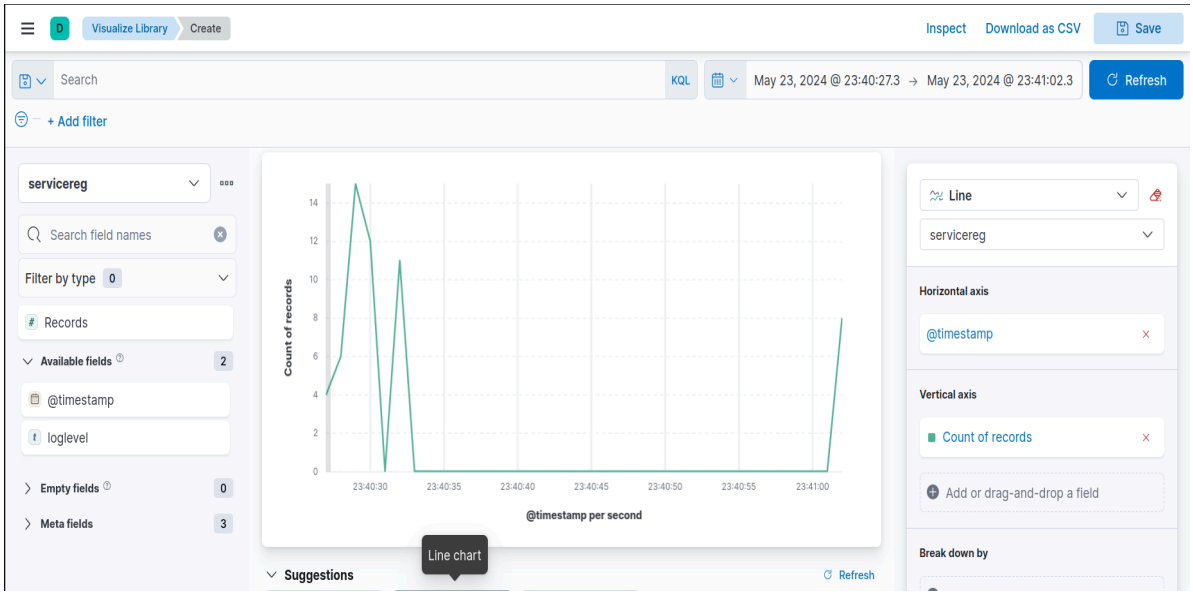
2. Quiz log levels



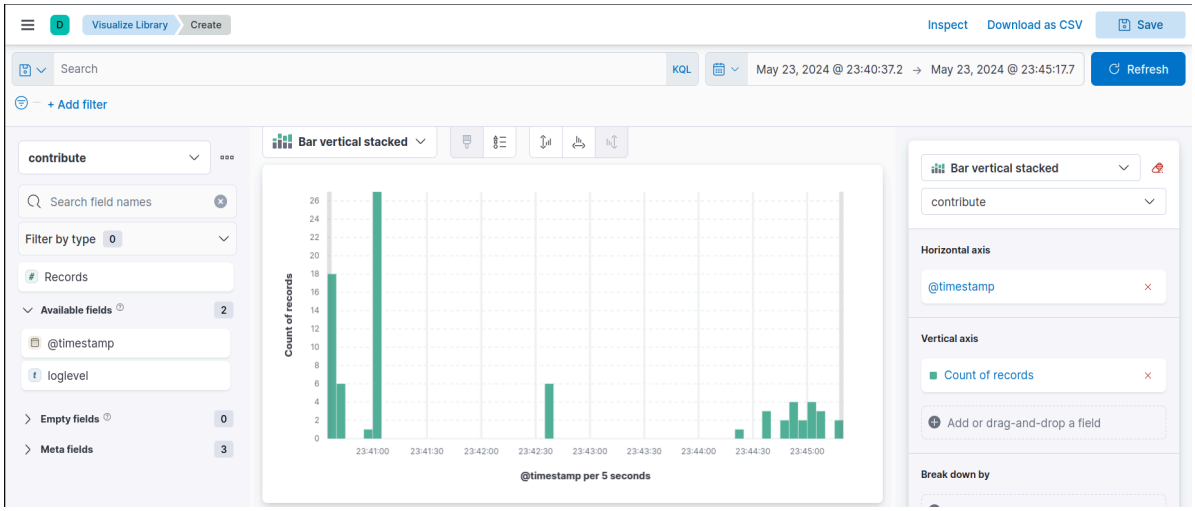
3. Eureka Log Levels



4. Api Gateway Timestamps



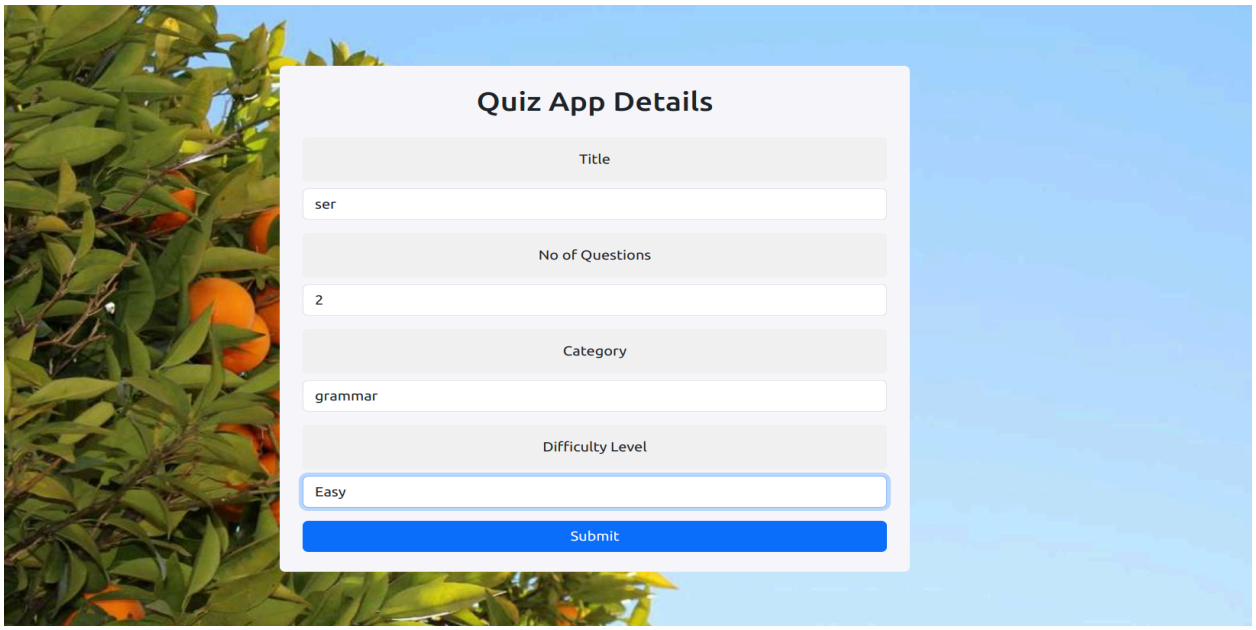
5. Contribute service timestamp



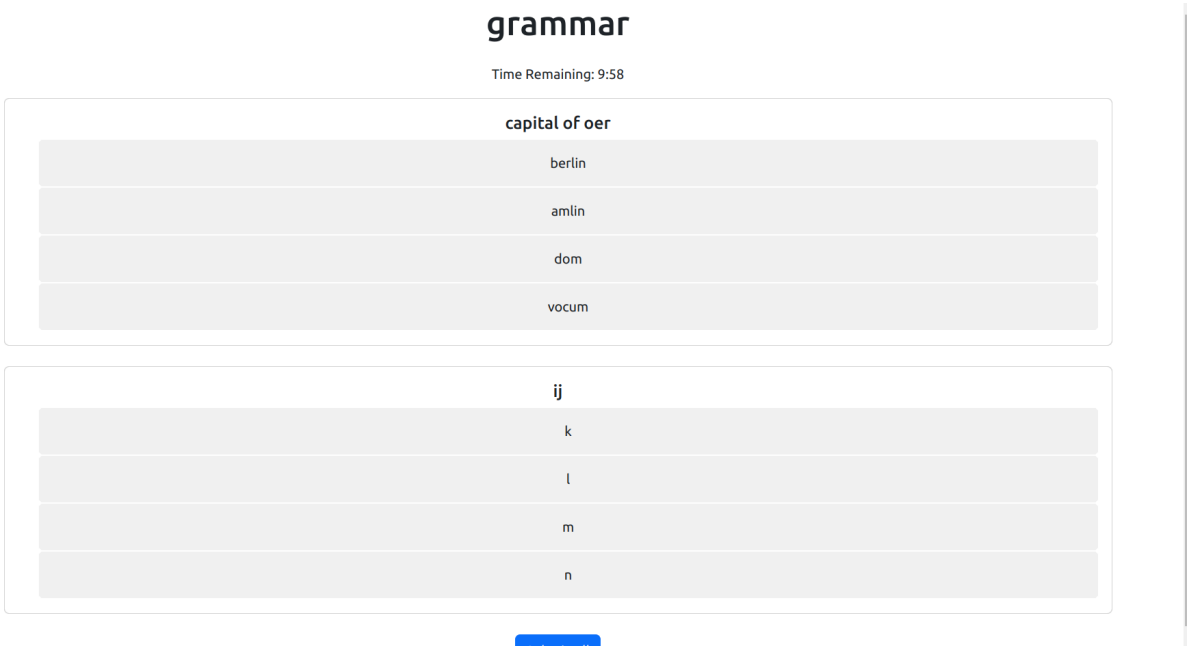
12. Application Snapshots

I. Home page

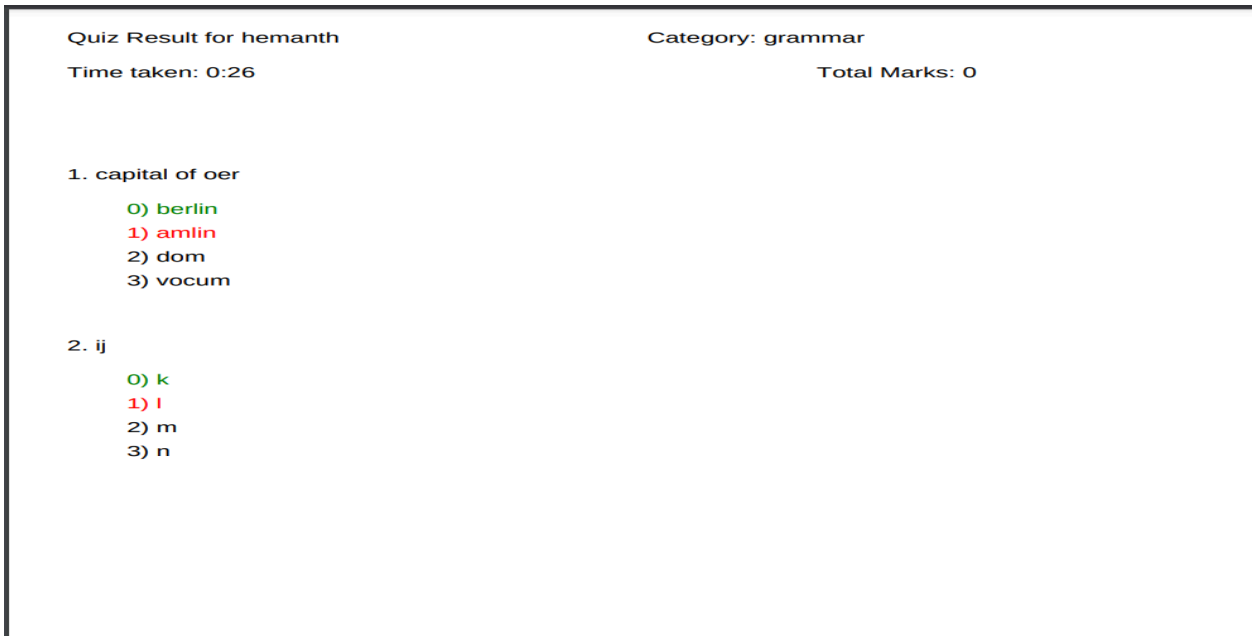
II. Create Quiz



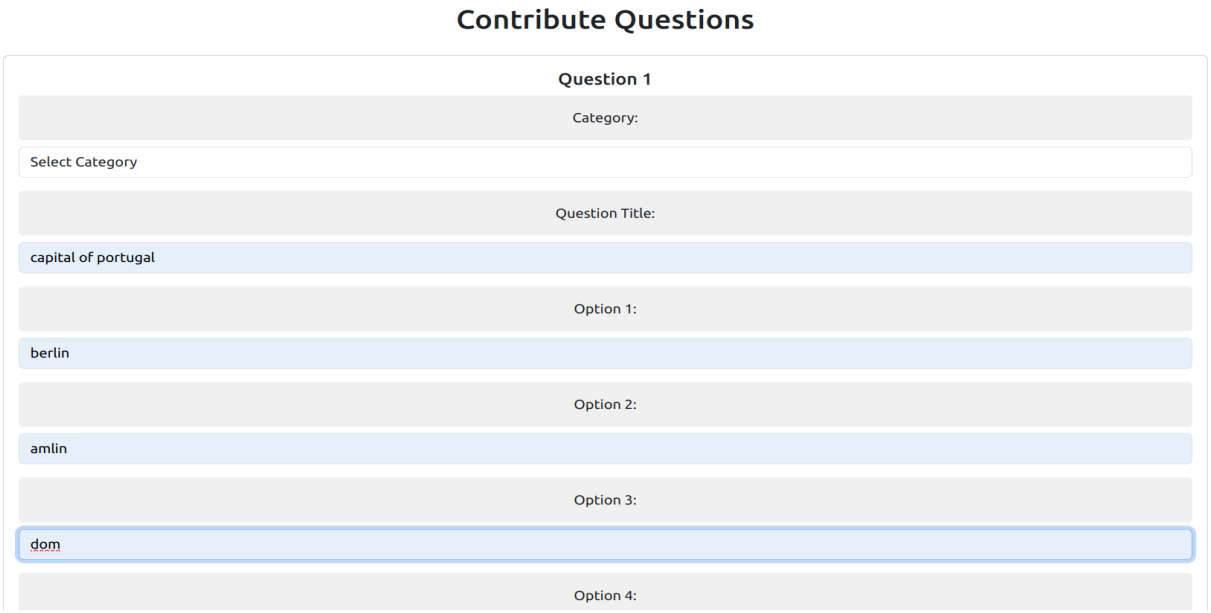
III. Attempt Quiz

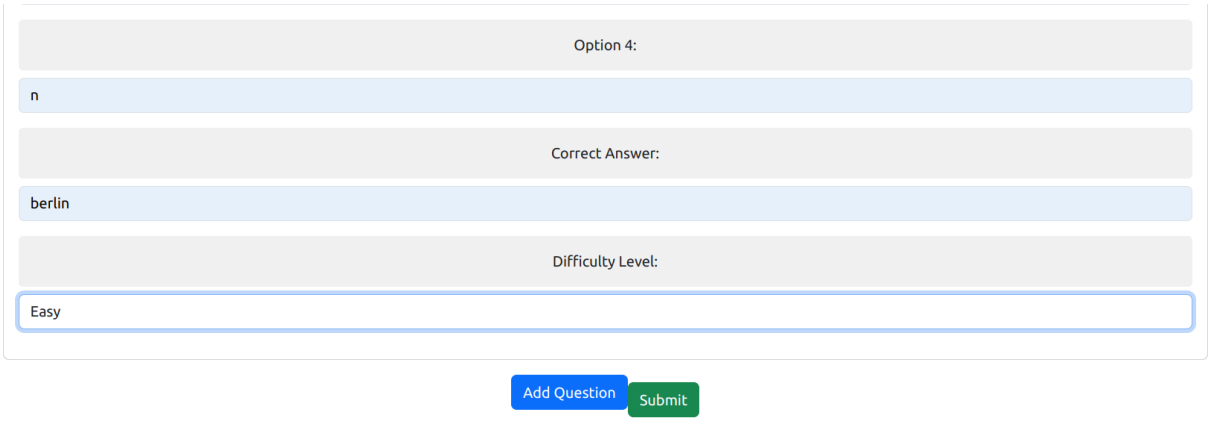


IV. Submission report of user

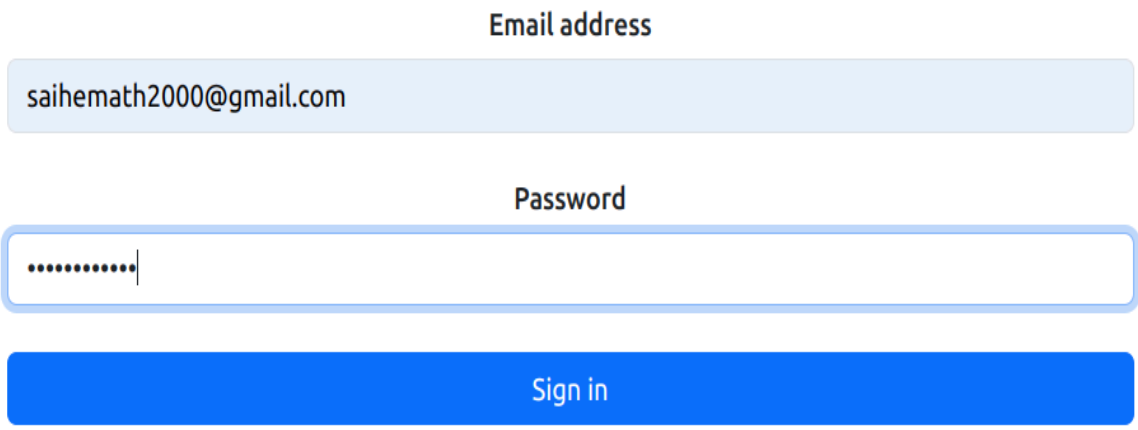


V. Contribute Questions

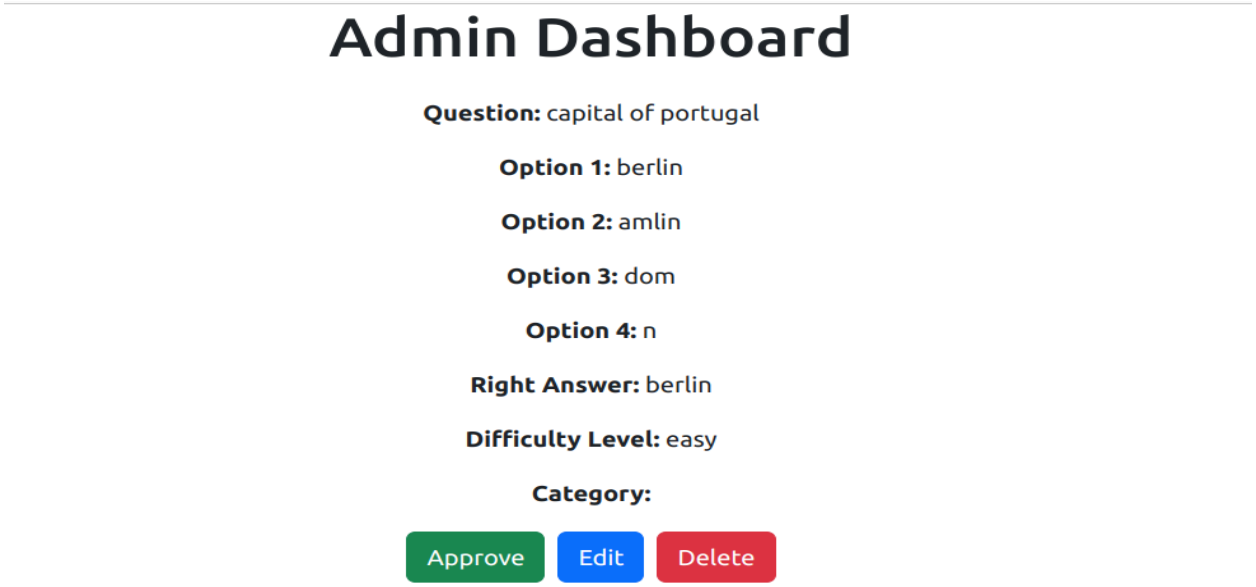




VI. Admin Login



VII. Admin dashboard



VIII. Approved Question by Admin