International Institute of Information Technology,Bangalore (IIIT Bangalore)

Software Production Engineering

Project Report

EKart : E-Commerce Application

Project TA *: Aman Gupta*

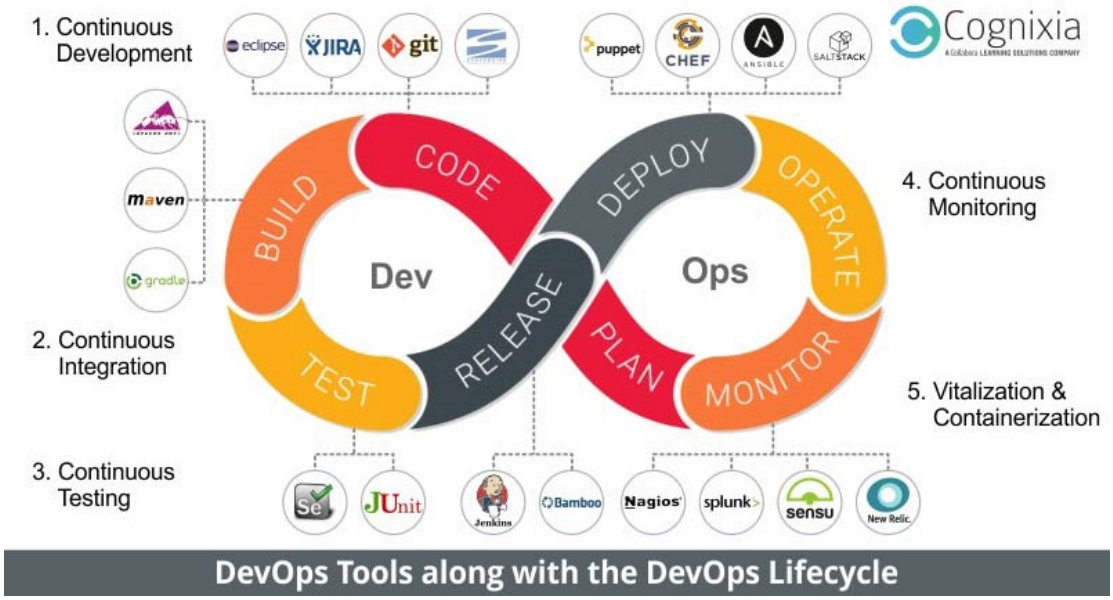
Submitted By,
Sreenidhi KR- MT2022115
Narasimhan N- MT2022062

Project Overview

Ekart is a E-Commerce 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

Source : https://www.cognixia.com/blog/resolving-software-development-woes-with-devops/

- 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.

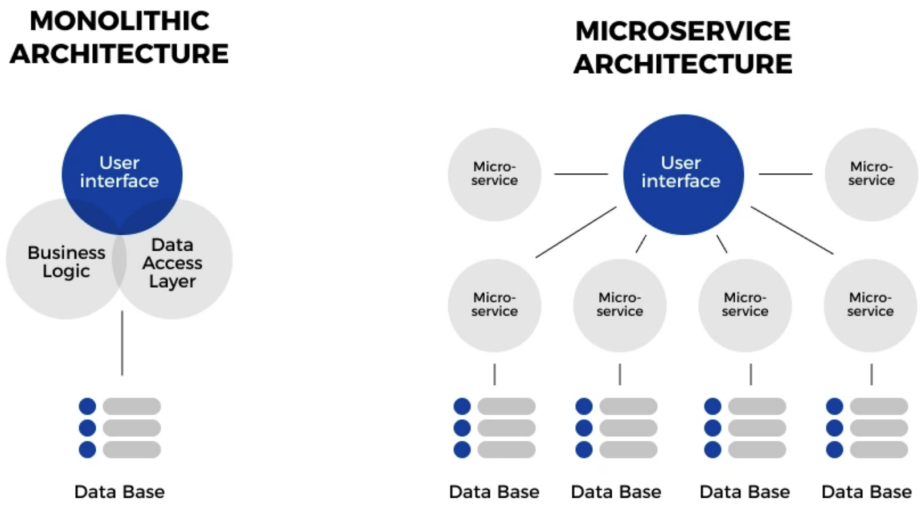
- Why DevOps?
  a. Each company faces its own challenges, but common problems include releases that take too long, software that doesn't meet expectations and IT that limits business growth.Without wait times, manual processes, and lengthy reviews, a DevOps project moves faster from requirements to live 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 occur 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.

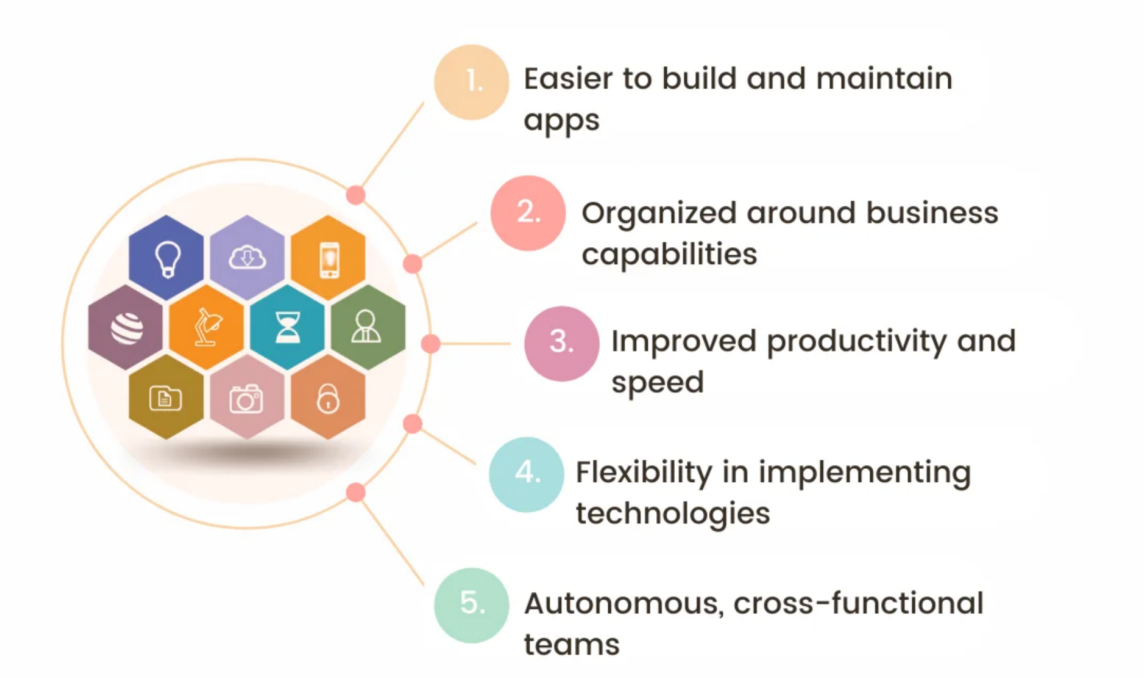
Microservices architectures make applications easier to scale and faster to develop, enabling innovation and accelerating time-to-market for new features.Containers are a well-suited microservices architecture example, since they let you focus on developing the services without worrying about the dependencies.

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.

Synchronous vs Asynchronous communication in Microservices

Synchronous communication and asynchronous communication are two different approaches to exchanging messages and data within a microservices architecture.

Synchronous communication involves a real-time, request-response interaction between services. In this approach, the requesting service sends a request to another service and waits for a response before proceeding. The requesting service remains blocked until it receives the response or a timeout occurs. Synchronous communication follows a direct and immediate exchange of information.

In synchronous communication:

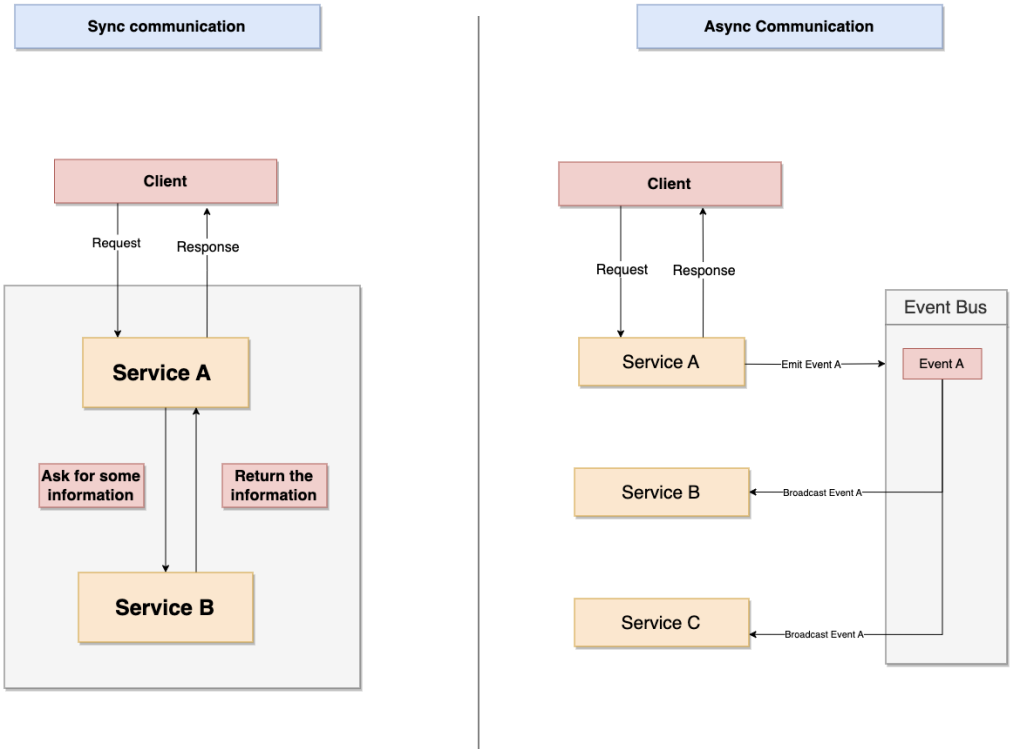
- The requesting service waits for a response.
- The response is expected in real-time or within a specific timeframe.

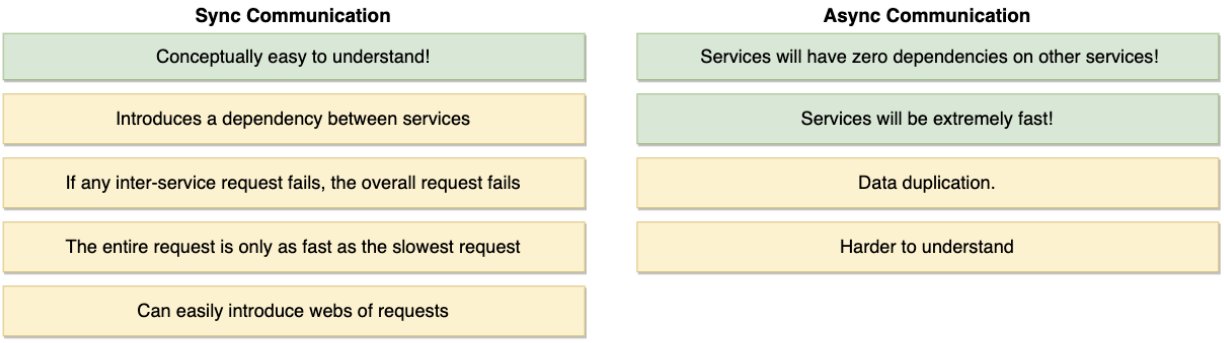
- The requesting service is tightly coupled to the availability and responsiveness of the responding service.
- It typically follows a simple request-response pattern, often through direct method invocations or API calls.
- Synchronous communication is straightforward to implement and understand.

Asynchronous communication, on the other hand, involves message-based or event-driven interactions between services. In this approach, the requesting service sends a message or publishes an event to a message queue or event bus without waiting for an immediate response. The requesting service continues with its execution independently of the response.

In asynchronous communication:

The requesting service does not wait for an immediate response.
The response may be processed by the responding service at its own pace.
There is loose coupling between services, as the requesting service is not dependent on the availability or responsiveness of the responding service.
It promotes decoupling and independence among services.
Asynchronous communication often utilizes message queues or event-driven architectures to manage the messages or events.
It enables scalability, flexibility, and resilience by allowing services to process messages or events asynchronously.





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. **Programming Language:** Javascript
b. **IDE :** Visual studio
c. **Testing:**
d. **Build tool:**
e. **Source Code Management:** GitHub for Git.
f. **Containerization:** Docker & DockerHub
g. **Continuous Integration:** Jenkins
h. **Continuous Deployment:** Ansible
i. **Container Orchestration**: Kubernates
j. **Cluster** : Minikube

k. **Generate Logs:** winston
l. **Monitoring:** ELK Stack

GitHub: Helps in automation through Jenkin Integration Calculator with DevOps.
Jenkins: It is used for Continuous Integration and Continuous Deployment portion.
Docker: It is used to make images through containerization.
Ansible: It automates and simplifies repetitive, complex, and tedious operations. It saves a lot of time when we install packages or configure large numbers of servers.
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.
Skaffold: Automatically updates and redeploys the deployment after any changes

Skaffold

Skaffold is a command line tool that facilitates continuous development for container based & Kubernetes applications. Skaffold handles the workflow for building, pushing, and deploying your application, and provides building blocks for creating CI/CD pipelines. This enables you to focus on iterating on your application locally while Skaffold continuously deploys to your local or remote Kubernetes cluster, local Docker environment or Cloud Run project.
Features

- Fast local Kubernetes Development
    - optimized “Source to Kubernetes” - Skaffold detects changes in your source code and handles the pipeline to build, push, test and deploy your application automatically with policy-based image tagging and highly optimized, fast local workflows
    - continuous feedback - Skaffold automatically manages deployment logging and resource port-forwarding
- Skaffold projects work everywhere
    - share with other developers - Skaffold is the easiest way to share your project with the world: git clone and skaffold run
    - context aware - use Skaffold profiles, local user config, environment variables, and flags to easily incorporate differences across environments

    - platform aware - use cross-platform and multi-platform build support, with automatic platform detection, to easily handle operating system and architecture differences between the development machine and Kubernetes cluster nodes.
    - CI/CD building blocks - use skaffold build, skaffold test and skaffold deploy as part of your CI/CD pipeline, or simply skaffold run end-to-end
    - GitOps integration - use skaffold render to build your images and render templated Kubernetes manifests for use in GitOps workflows
- skaffold.yaml - a single pluggable, declarative configuration for your project
    - skaffold init - Skaffold can discover your build and deployment configuration and generate a Skaffold config
    - multi-component apps - Skaffold supports applications with many components, making it great for microservice-based applications
    - bring your own tools - Skaffold has a pluggable architecture, allowing for different implementations of the build and deploy stages
- Lightweight
    - client-side only - Skaffold has no cluster-side component, so there's no overhead or maintenance burden to your cluster
    - minimal pipeline - Skaffold provides an opinionated, minimal pipeline to keep things simple

3. Setup/Installations

a. Skaffold

i. To install Skaffold type the following commands on terminal

**$ curl -Lo skaffold https://storage.googleapis.com/skaffold/releases/latest/skaffold-linux-amd64 && **
sudo install skaffold /usr/local/bin/

b. Nodejs

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

$ sudo apt update
$ sudo apt install nodejs

ii. Check the Node version

$ node -v or node –version

c. 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

```
simha@simha-Predator-PH315-54:~$ git --version
git version 2.34.1
simha@simha-Predator-PH315-54:~$
```

- GitHub :

  i. Create your own GitHub account.

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

d. 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>

e. 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.

f. Docker & DockerHub

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

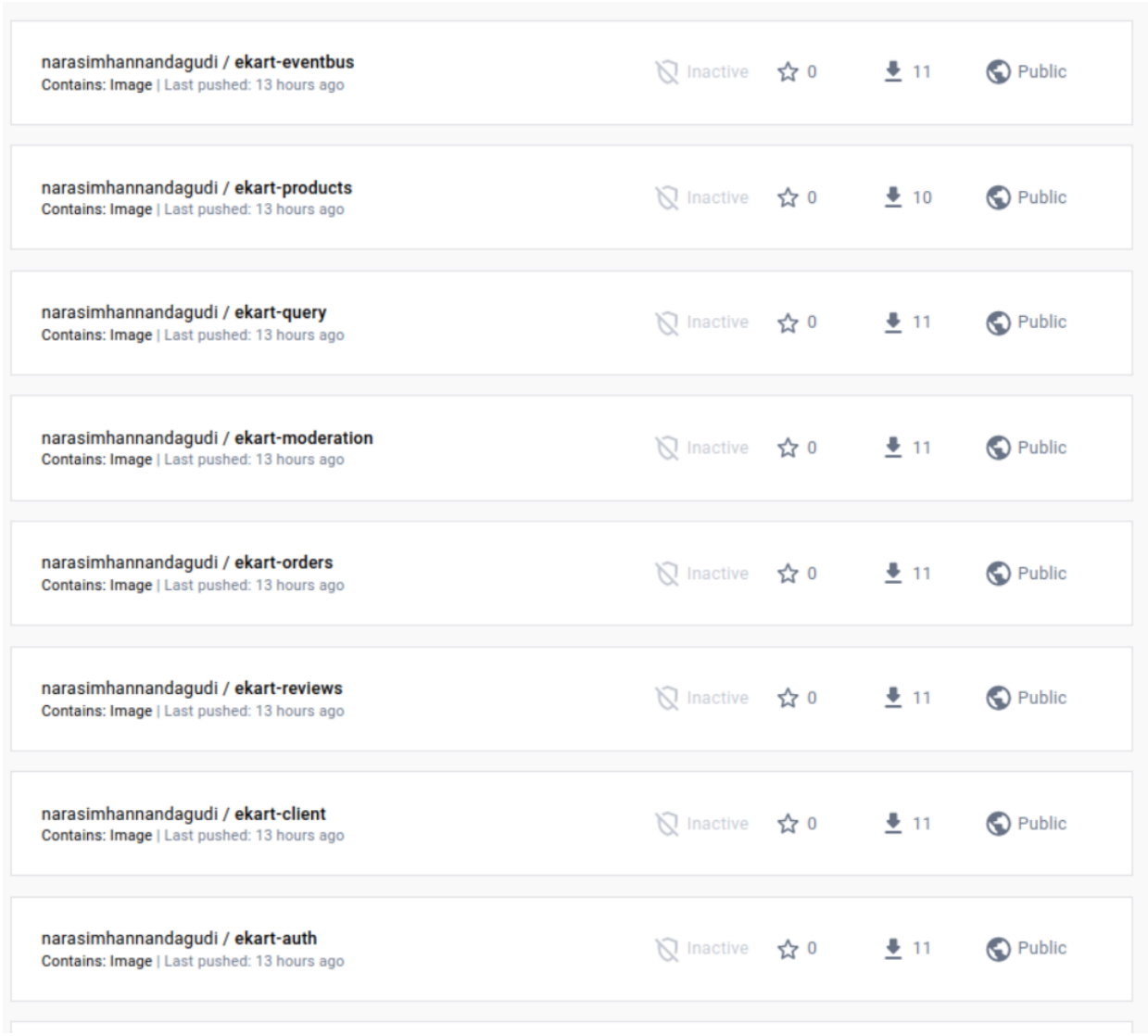
$ apt-get install docker.io

$ service docker start

$ service docker status

ii. DockerHub

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



g. 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

```
simha@simha-Predator-PH315-54:~$ kubectl version --client
WARNING: This version information is deprecated and will be replaced with the ou
tput from kubectl version --short.  Use --output=yaml|json to get the full versi
on.
Client Version: version.Info{Major:"1", Minor:"26", GitVersion:"v1.26.3", GitCom
mit:"9e644106593f3f4aa98f8a84b23db5fa378900bd", GitTreeState:"clean", BuildDate:
"2023-03-15T13:40:17Z", GoVersion:"go1.19.7", Compiler:"gc", Platform:"linux/amd
64"}
```

h. 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

```
simha@simha-Predator-PH315-54:~$ minikube start
😄  minikube v1.29.0 on Ubuntu 22.04
✨  Using the docker driver based on existing profile
👍  Starting control plane node minikube in cluster minikube
🚜  Pulling base image ...
🔄  Restarting existing docker container for "minikube" ...
🐳  Preparing Kubernetes v1.26.1 on Docker 20.10.23 ...
🔗  Configuring bridge CNI (Container Networking Interface) ...
🔎  Verifying Kubernetes components...
    ▪ Using image registry.k8s.io/ingress-nginx/controller:v1.5.1
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20220916-gd32f8c343
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20220916-gd32f8c343
🔎  Verifying ingress addon...
🌟  Enabled addons: storage-provisioner, default-storageclass, ingress
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

iii. To Check status

$ minikube status

```
simha@simha-Predator-PH315-54:~$ minikube status
minikube
type: Control Plane
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured
```

iv.Enable Ingress

$ minikube addons enable ingress

```
simha@simha-Predator-PH315-54:~$ minikube addons enable ingress
💡  ingress is an addon maintained by Kubernetes. For any concerns contact minik
ube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/
minikube/blob/master/OWNERS
    ▪ Using image registry.k8s.io/ingress-nginx/controller:v1.5.1
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20220916-g
d32f8c343
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v20220916-g
d32f8c343
🔎  Verifying ingress addon...
🌟  The 'ingress' addon is enabled
```

i. Ansible

i. Ansible installation on server :

$ sudo apt update

$ sudo apt install ansible

ii. To check the version :

$ ansible --version

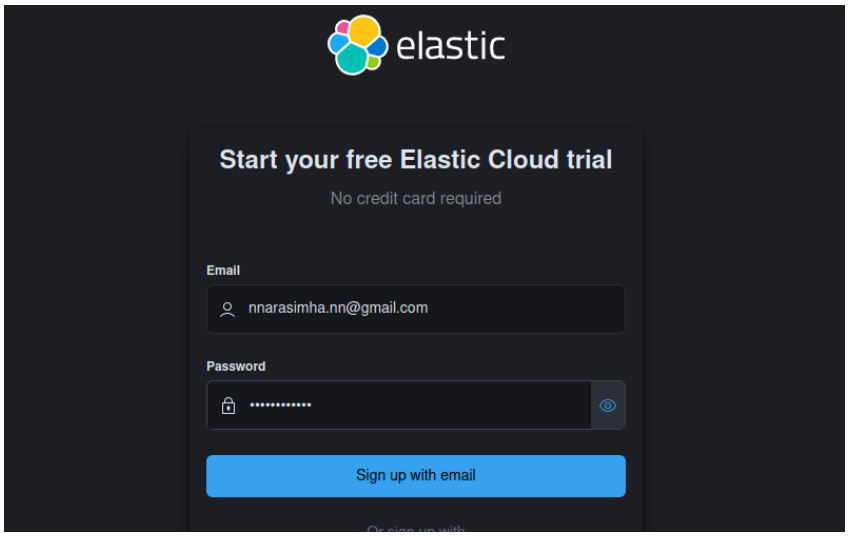
```
simha@simha-Predator-PH315-54:~$ ansible --version
ansible [core 2.14.3]
  config file = /etc/ansible/ansible.cfg
  configured module search path = ['/home/simha/.ansible/plugins/modules', '/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python3/dist-packages/ansible
  ansible collection location = /home/simha/.ansible/collections:/usr/share/ansible/collections
  executable location = /usr/bin/ansible
  python version = 3.10.6 (main, Nov 14 2022, 16:10:14) [GCC 11.3.0] (/usr/bin/python3)
  jinja version = 3.0.3
  libyaml = True
```

- You can find the ansible.cfg configuration file inside /etc/ansible folder. Gotothat file using command cd /etc/ansible/
- Here you find ansible.cfg file and host file/ inventory file. If host file is not there just create one host file using command touch hosts. Here the host file is an inventory file.
- Default location of inventory file is /etc/ansible/hosts .You can also create your own inventory file. Here in path /etc/ansible/ we can keep all our playbooks.

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 Javascript

d. Create a repository in DockerHub for your project

e. Create Pipeline Script in Jenkins

- Git Pull
- Build Docker Image s
- Pushing Images to Docker Hub

- Clear the docker images
- Ansible Deploy

f. Build the project.

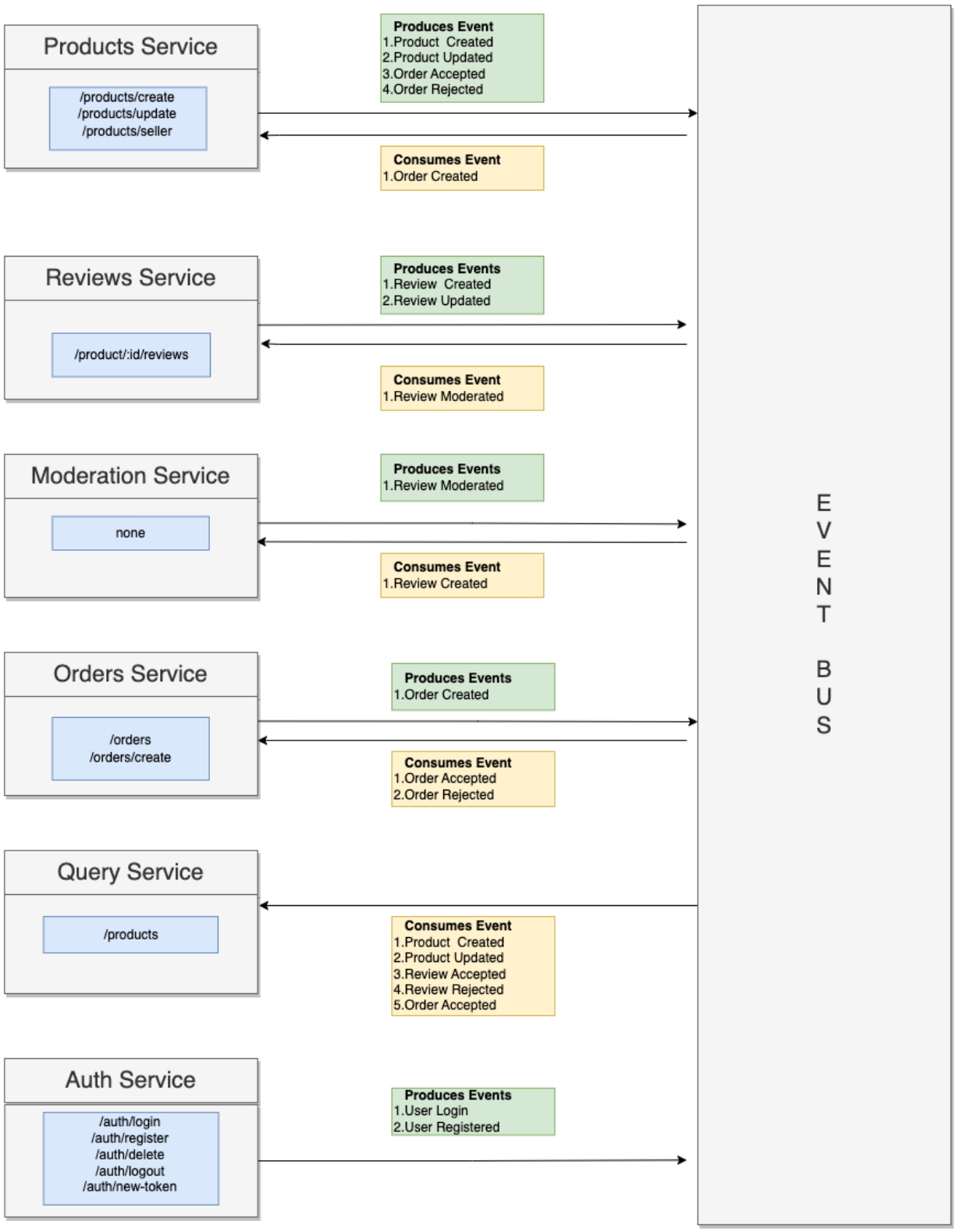
g. Run the image

h. Monitor, Analyze and Create visualization using ELK

6. Software - Develop, Build and Test

i. Development:

- Tech Stack:
  - Frontend : React
  - Backend: Nodejs with Express
  - Database: MongoDB
- The application 8 microservices:
  - Auth : For Authorisation
  - Products: For Products management
  - Orders: For Order management
  - Query: To display all products
  - Review: To handle reviews
  - Moderation: To flag comments with illegal words
  - Event bus: To broadcast all services
  - Client: Web application



List of events produced and consumed by Services

ii. Testing : Jest and Supertest are commonly used together for testing node js applications and APIs. Jest is a popular testing framework, and Supertest is an extension library specifically designed for testing HTTP servers and APIs. Supertest provides an easy way to send HTTP requests and make assertions on the responses.

To get started with Jest and Supertest, follow these steps:

npm install jest supertest --save-dev

Set up your test files and directory structure:

Create a tests directory in your project.

Place your test files inside the tests directory. Test files typically have a .test.js or .spec.js extension.

Write your tests using Jest and Supertest in your test files:

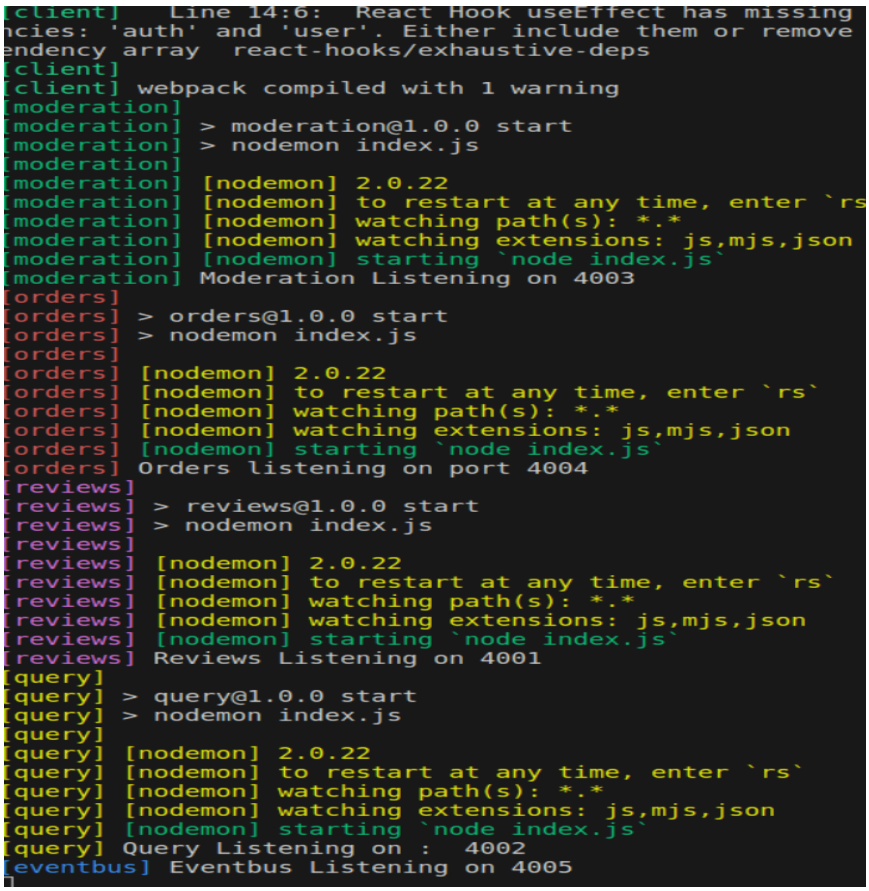
javascript Copy code

```javascript
const request = require('supertest');
const app = require('../app'); // Import your Express app or server file

describe('API endpoints', () => {
  test('GET /api/users should return a list of users', async () => {
    const response = await request(app).get('/api/users');
    expect(response.statusCode).toBe(200);
    expect(response.body).toEqual(expect.arrayContaining([
      { id: 1, name: 'John' },
      { id: 2, name: 'Jane' }
    ]));
  });

  test('POST /api/users should create a new user', async () => {
    const response = await request(app).post('/api/users').send({ name: '
    expect(response.statusCode).toBe(201);
    expect(response.body).toHaveProperty('id');
    expect(response.body.name).toBe('Alice');
  });

  // Add more test cases as needed
});
```

Run **npx jest** to run the tests

iii. Build : Skaffold can be used to build during development

$ skaffold dev



This is to be done in the directory of Skaffold file. Snapshot of the skaffold.yaml

```yaml
apiVersion: skaffold/v2alpha3
kind: Config
deploy:
  kubectl:
    manifests:
      - ./infra/k8s/*
build:
  local:
    push: false
  artifacts:
    - image: narasimhannandagudi/ekart-auth
      context: auth
      docker:
        dockerfile: Dockerfile
      sync:
        manual:
          - src: "*.js"
            dest: .
    - image: narasimhannandagudi/ekart-client
      context: client
      docker:
        dockerfile: Dockerfile
      sync:
        manual:
          - src: "src/**/*.js"
            dest: .
          - src: "src/**/*.css"
            dest: .
    - image: narasimhannandagudi/ekart-products
      context: products
      docker:
        dockerfile: Dockerfile
      sync:
        manual:
          - src: "*.js"
            dest: .
    - image: narasimhannandagudi/ekart-eventbus
      context: eventbus
      docker:
        dockerfile: Dockerfile
```

iv. Project Dependencies:

Every microservices has its dependencies defined in the package.json file

```
{
  "name": "orders",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "start": "nodemon index.js"
  },
  "keywords": [],
  "author": "",
  "license": "ISC",
  "dependencies": {
    "axios": "^1.3.4",
    "cors": "^2.8.5",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.0",
    "nodemon": "^2.0.22",
    "mongoose": "^7.1.0"
  }
}
```

v.Output:

We can check if the deployments are running by the following

$kubectl apply -f .

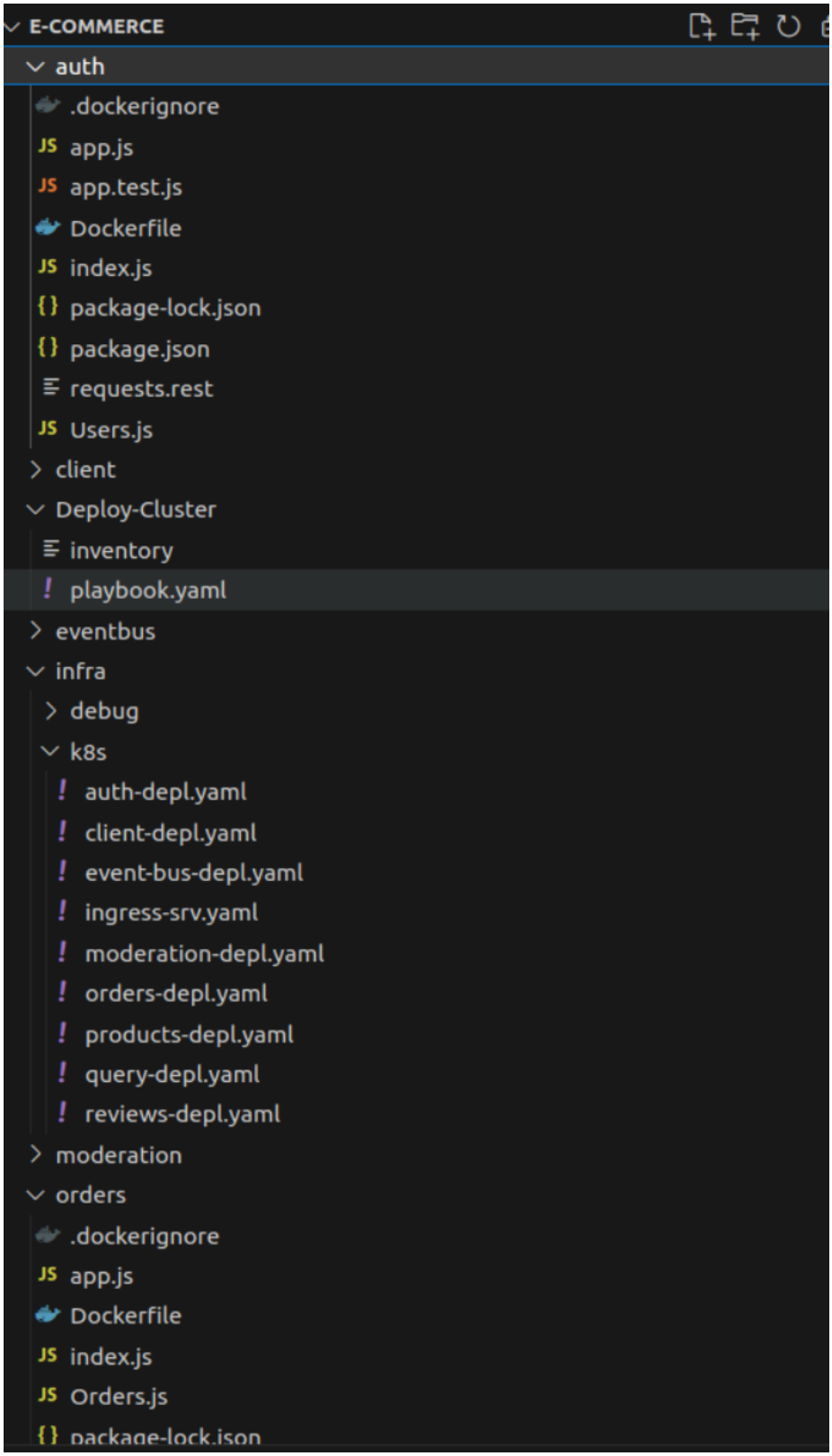
Applies to all the deployment files

To view all deployments :

$kubectl get all

```
simha@simha-Predator-PH315-54:~$ kubectl get all
NAME                                    READY   STATUS    RESTARTS   AGE
pod/auth-depl-5bddcd6d49-vqr22          1/1     Running   0          5m32s
pod/client-depl-74cd6f4c94-v857m        1/1     Running   0          5m32s
pod/eventbus-depl-5b5d965bfb-rzm54      1/1     Running   0          5m32s
pod/moderation-depl-5c754df96f-fs5f6    1/1     Running   0          5m32s
pod/orders-depl-58bdc777bf-hgq7f        1/1     Running   0          5m32s
pod/products-depl-bc8bd67fc-f44xg       1/1     Running   0          5m32s
pod/query-depl-66b654d845-8lczd         1/1     Running   0          5m32s
pod/reviews-depl-7b59c4b744-vzf8t       1/1     Running   0          5m32s

NAME                     TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)    AGE
service/auth-srv         ClusterIP   10.106.229.168   <none>        3001/TCP   12m
service/client-srv       ClusterIP   10.107.95.106    <none>        3000/TCP   12m
service/eventbus-srv     ClusterIP   10.111.135.224   <none>        4005/TCP   12m
service/kubernetes       ClusterIP   10.96.0.1        <none>        443/TCP    17m
service/moderation-srv   ClusterIP   10.96.21.166     <none>        4003/TCP   12m
service/orders-srv       ClusterIP   10.109.167.76    <none>        4004/TCP   12m
service/products-srv     ClusterIP   10.103.54.70     <none>        4000/TCP   12m
service/query-srv        ClusterIP   10.103.218.5     <none>        4002/TCP   12m
service/reviews-srv      ClusterIP   10.105.100.166   <none>        4001/TCP   12m

NAME                                READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/auth-depl           1/1     1            1           12m
deployment.apps/client-depl         1/1     1            1           12m
deployment.apps/eventbus-depl       1/1     1            1           12m
deployment.apps/moderation-depl     1/1     1            1           12m
deployment.apps/orders-depl         1/1     1            1           12m
deployment.apps/products-depl       1/1     1            1           12m
deployment.apps/query-depl          1/1     1            1           12m
deployment.apps/reviews-depl        1/1     1            1           12m
```

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 out 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 <github repo URL>

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

$ git add .

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

$git commit -m "Commit Message "

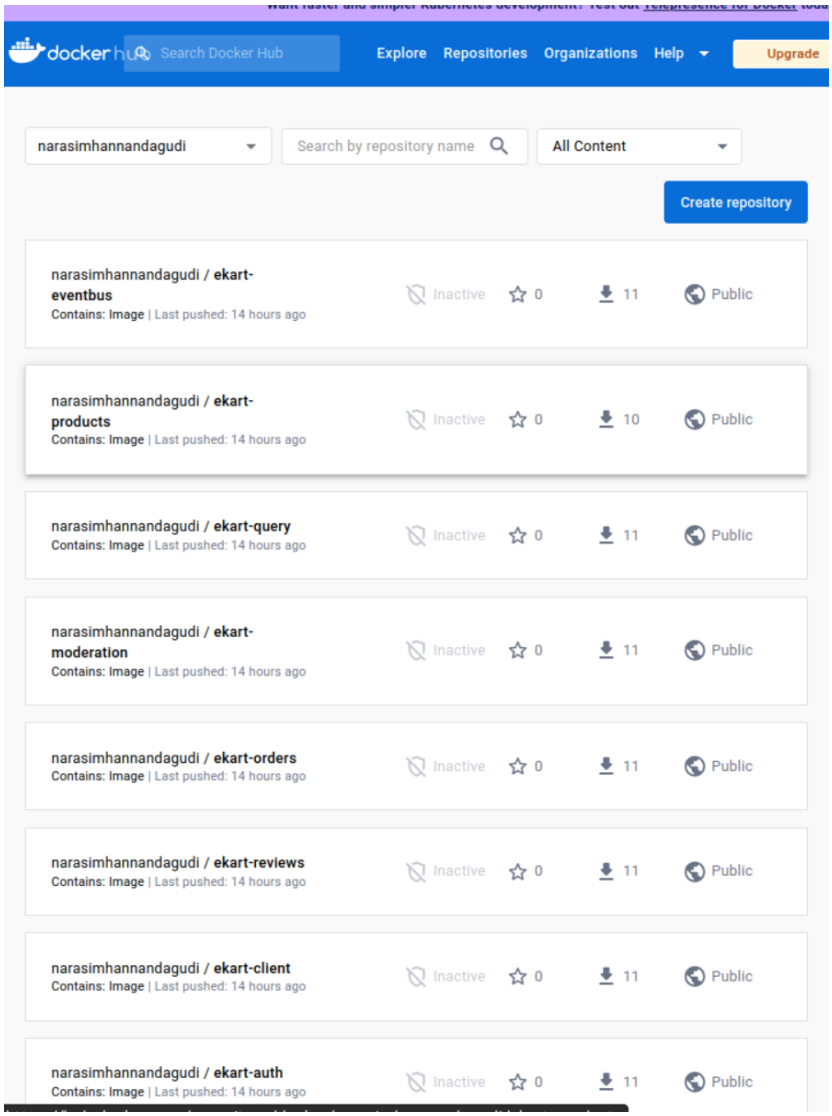
This command is to commit the staged files to GitHub.

$ git push -u origin master

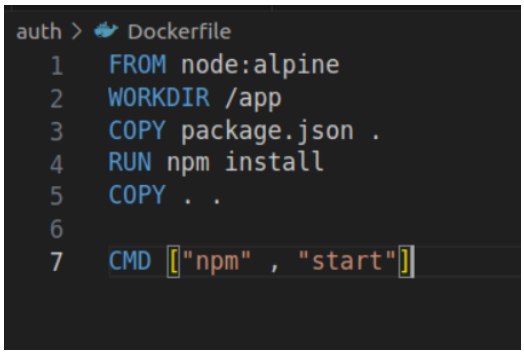
This command is to push the changes to the GitHub repository.

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 image via ssh.

$ mkdir .ssh

$ cd .ssh

$ ssh-keygen -t rsa

$ ssh-copy-id simha@localhost

$ ssh simha@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:**

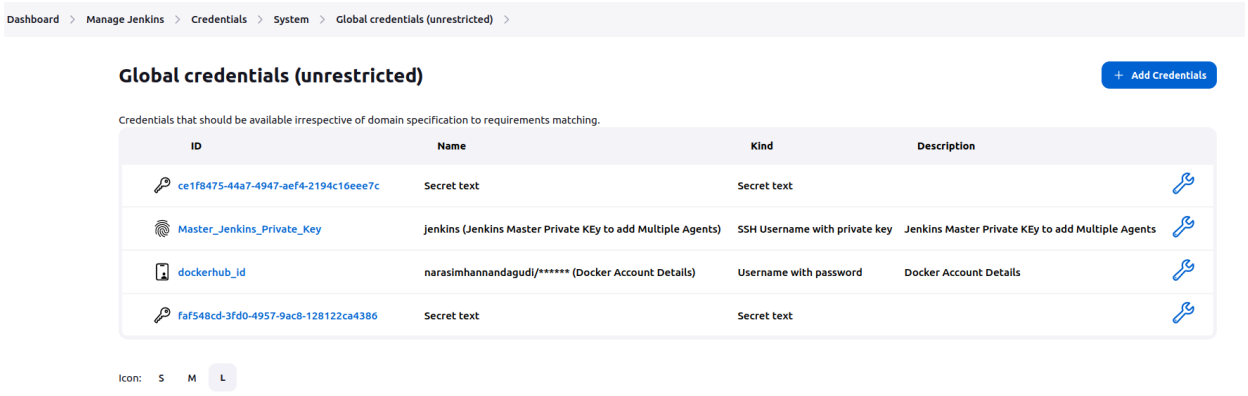
We will need variety of plugins to utilizing Jenkins Pipeline. Install following thingsfrom Plugin Manager

- Git
- Ansible
- Docker

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

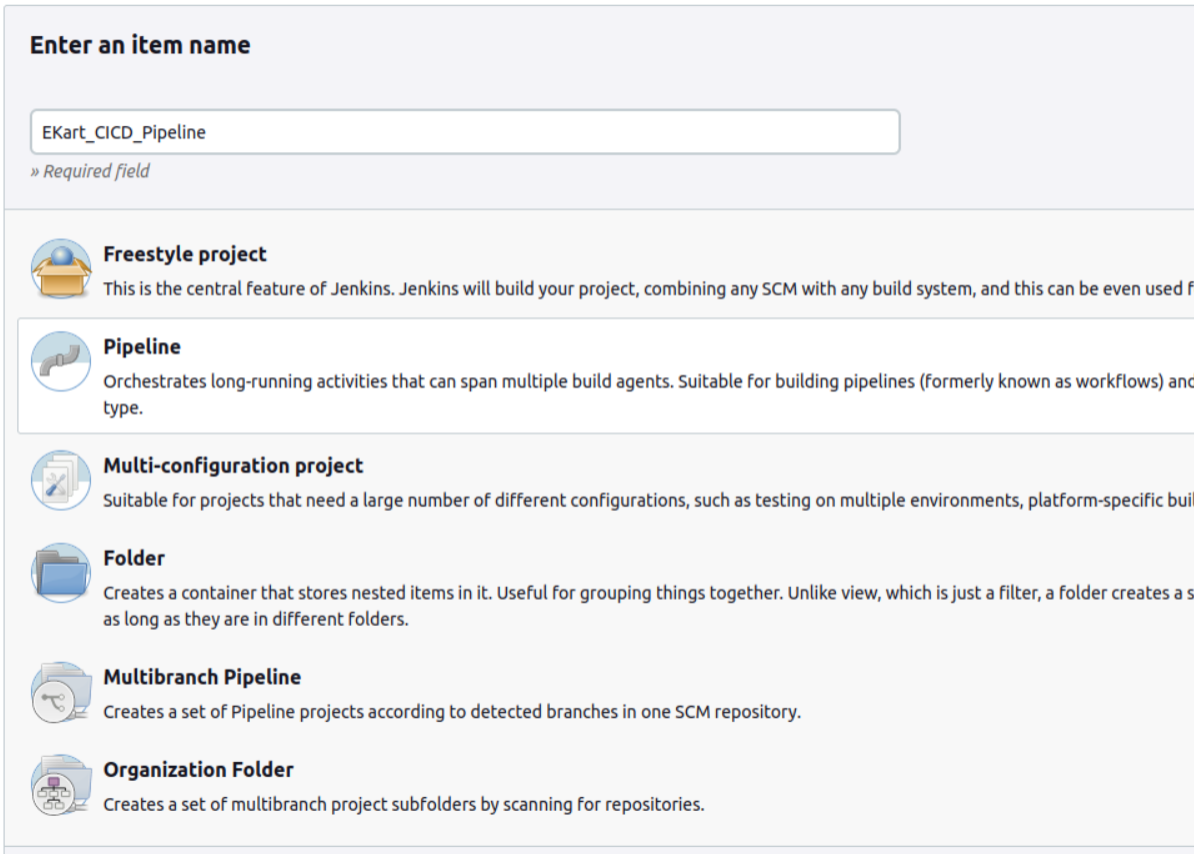
● Add Docker credentials:In the Jenkins dashboard we add credentials to theDockerHub repository and we set an unique id which is equal to docker with Registry Credentials id in pipeline script.

● 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.

```
pipeline {
    environment {
    registryCredential = 'dockerhub_id'

    authImage = ''
    clientImage=''
    productsImage=''
    ordersImage=''
    queryImage=''
    reviewsImage=''
    moderationImage=''
    eventbusImage=''
    }
    agent any

    stages {
        stage('Git access') {
            steps {
               git branch: 'main', url: 'https://github.com/NarasimhanN/E-Commerce.git'

            }
        }
          stage('Build Docker Image'){
            steps{
                  script{
                   sh 'cd auth'

                   authImage = docker.build("narasimhannandagudi/ekart-auth","./auth/")
                   clientImage = docker.build("narasimhannandagudi/ekart-client","./client")
                   eventbusImage = docker.build("narasimhannandagudi/ekart-eventbus","./eventbus/")
                   reviewsImage = docker.build("narasimhannandagudi/ekart-reviews","./reviews/")
                   ordersImage = docker.build("narasimhannandagudi/ekart-orders","./orders/")
                   moderationImage = docker.build("narasimhannandagudi/ekart-moderation","./moderation/")
                   queryImage = docker.build("narasimhannandagudi/ekart-query","./query/")
                   productsImage = docker.build("narasimhannandagudi/ekart-products","./products/")



                  }

            }
        }
        stage("Upload Image to DockerHub"){
            steps{
                script{
                    docker.withRegistry('',registryCredential){
                        authImage.push()
                        clientImage.push()
                        reviewsImage.push()
                        ordersImage.push()
                        moderationImage.push()
                        queryImage.push()
                        productsImage.push()
                        eventbusImage.push()
                    }
                }
            }
        }
        stage('Test') {
            steps {
                dir(auth){
                    sh 'jest'
                }
```

10. Continuous Deployment

- Ansible is an open-source automation tool, or platform, used for IT tasks such as configuration management, application deployment, intraservice orchestration, and provisioning.
- Ansible is mainly used to perform a lot of tasks that otherwise are time consuming, complex, repetitive, and can make a lot of errors or issues.

i. Create an Ansible playbook We make a new directory in our working directory create the Ansible playbook.

```yaml
- name: Deploy Kubernates cluster
  hosts: localhost
  remote_user: simha

  tasks:
    - name: Run Kubernetes Deployment files files
      shell: kubectl apply -f "{{ item }}"
      with_fileglob:
        - ./../infra/k8s/*.yaml
```

Deploy-Cluster > inventory

```
1   [localhost]
2   127.0.0.1 ansible_user=simha
```

ii. Configure Ansible in Jenkins We need to go to *Jenkins -> Dashboard ->ManageJenkins -> Global Tool Configuration* and add Ansible there. Here, we also need to give the correct path to Ansible.

iii. Ansible Stage in the Jenkins pipeline script

```
stage("Ansible Deploy cluster"){
    steps{
        ansiblePlaybook colorized: true, disableHostKeyChecking: true,
        inventory: 'Deploy-Cluster/inventory', playbook: 'Deploy-Cluster/playbook.yaml',
         sudoUser: null

    }
```

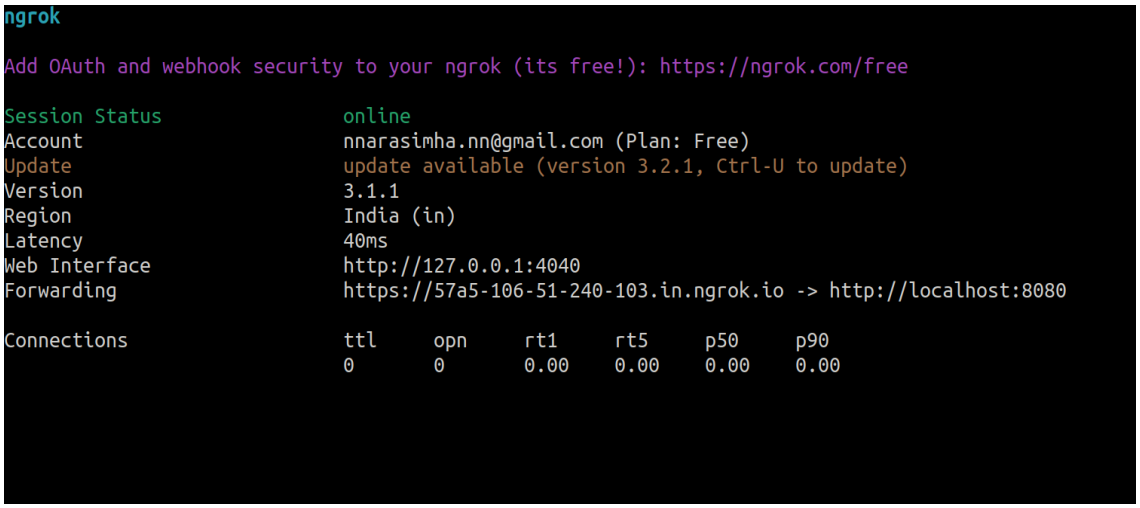
11. Build Pipeline & Run Docker Image

- 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 event using Ngrok and webhooks. But Initially we perform this job manually.

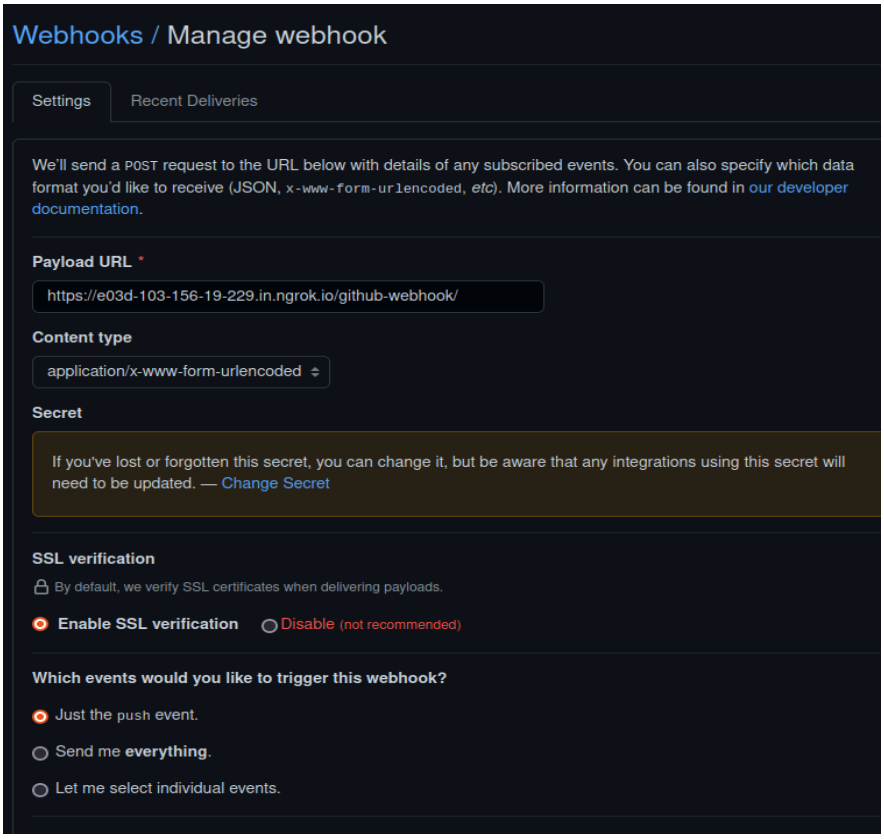
- **Ngrok**:

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.

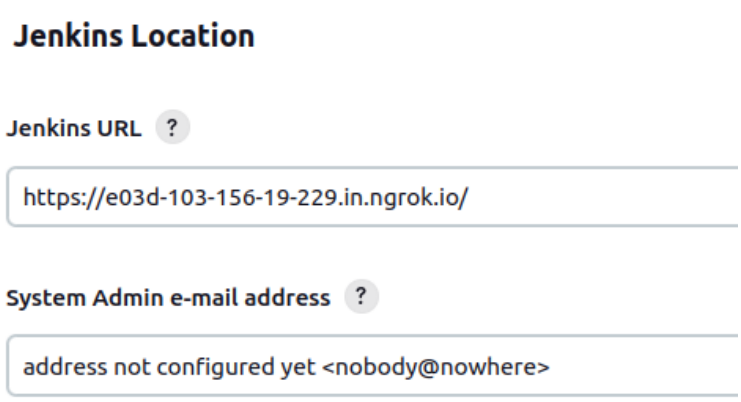


- **Webhooks :**
  This is used to automate the build process whenever the developer commits the code to GitHub.
  - Copy the public IP address generated by ngrok and paste it as payloadURLof webhook of the github repository that is associated with the project. - Also goto Jenkins -> Manage jenkins ->Configure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

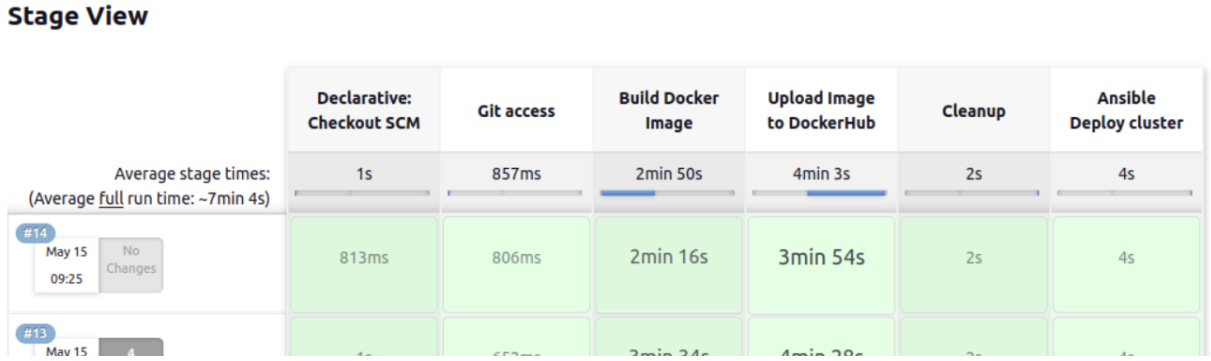


Build Triggers

- [ ] Build after other projects are built ?
- [ ] Build periodically ?
- [x] GitHub hook trigger for GITScm polling ?
- [ ] Poll SCM ?
- [ ] Quiet period ?



- 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:

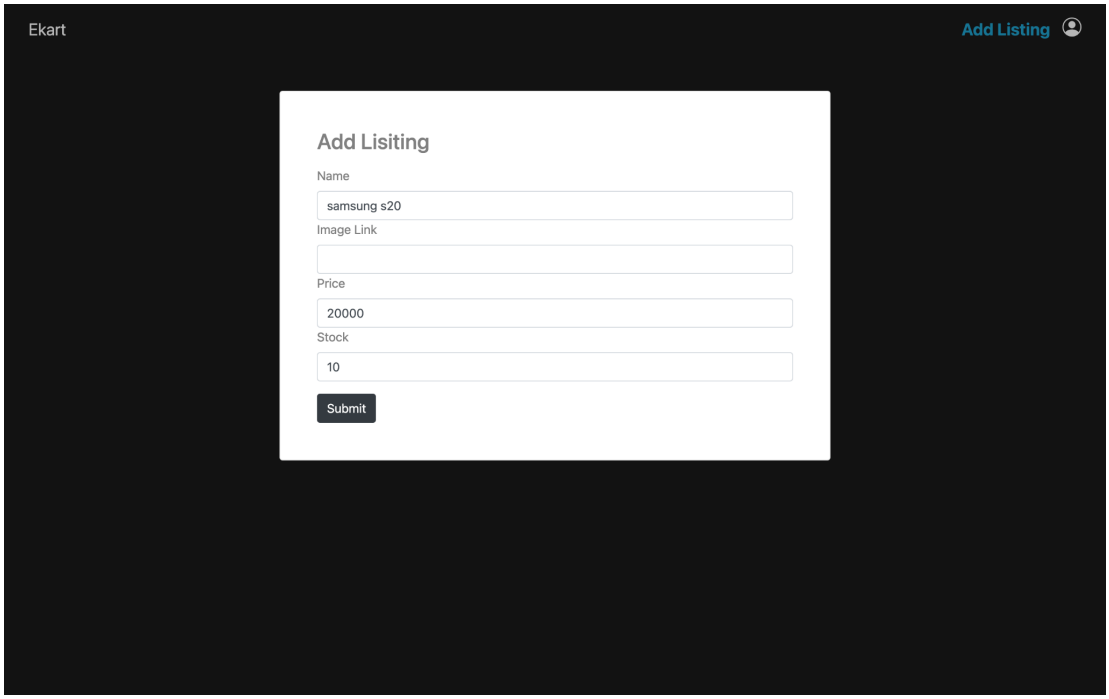
    1. Git Clone
    2. Build Docker images
    3. Push Docker Image to Hub
    4. Clean the docker images and containers
    5. Ansible Deployment

- 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.

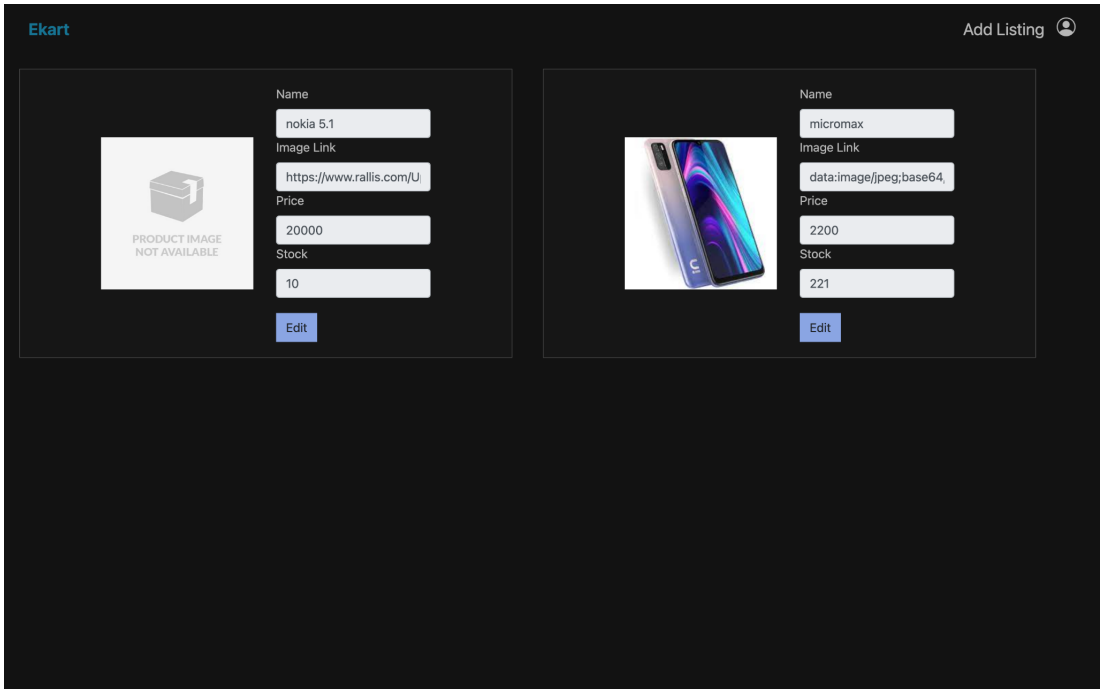
12. Application Snapshots

i.Seller

Add Listing Page:

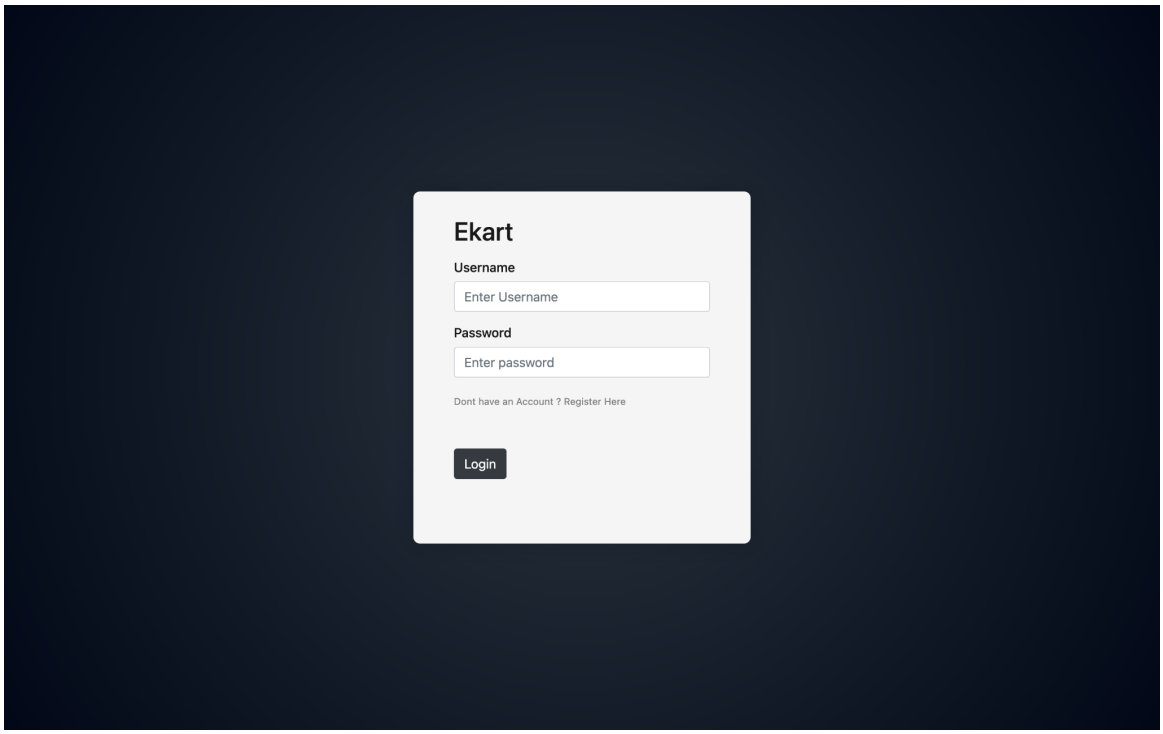


Seller Listings :Products listed by seller , can view and update the listing here

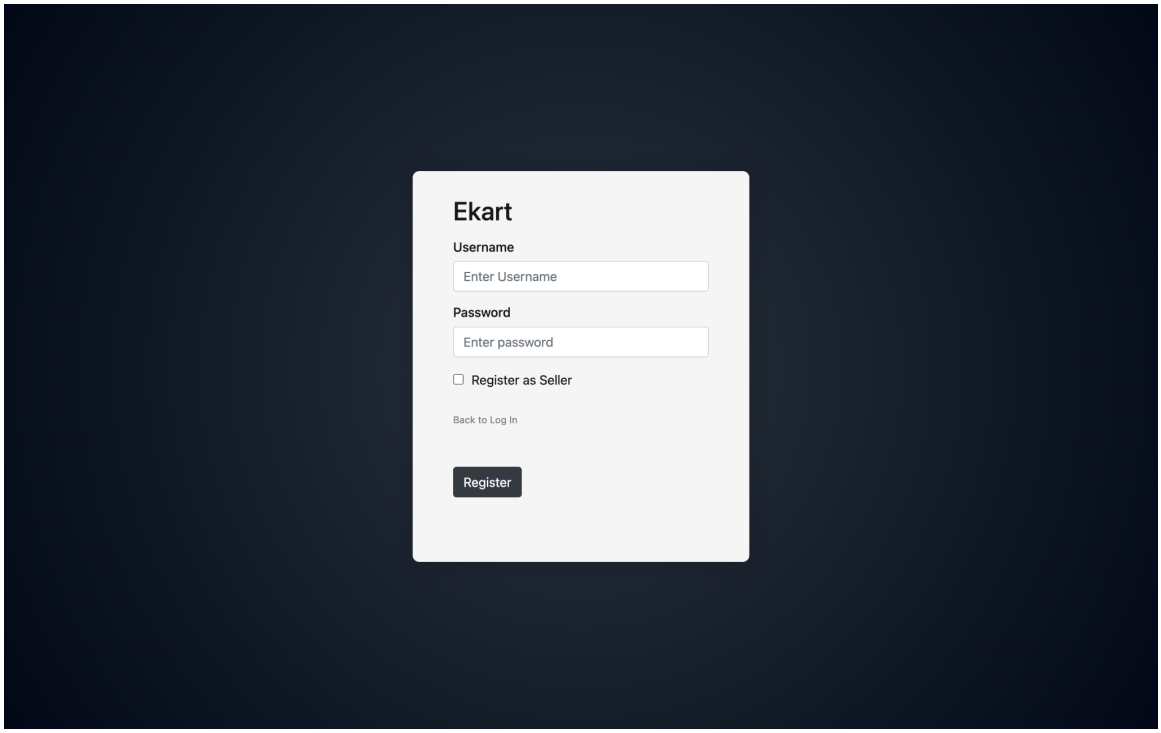


ii.Customer

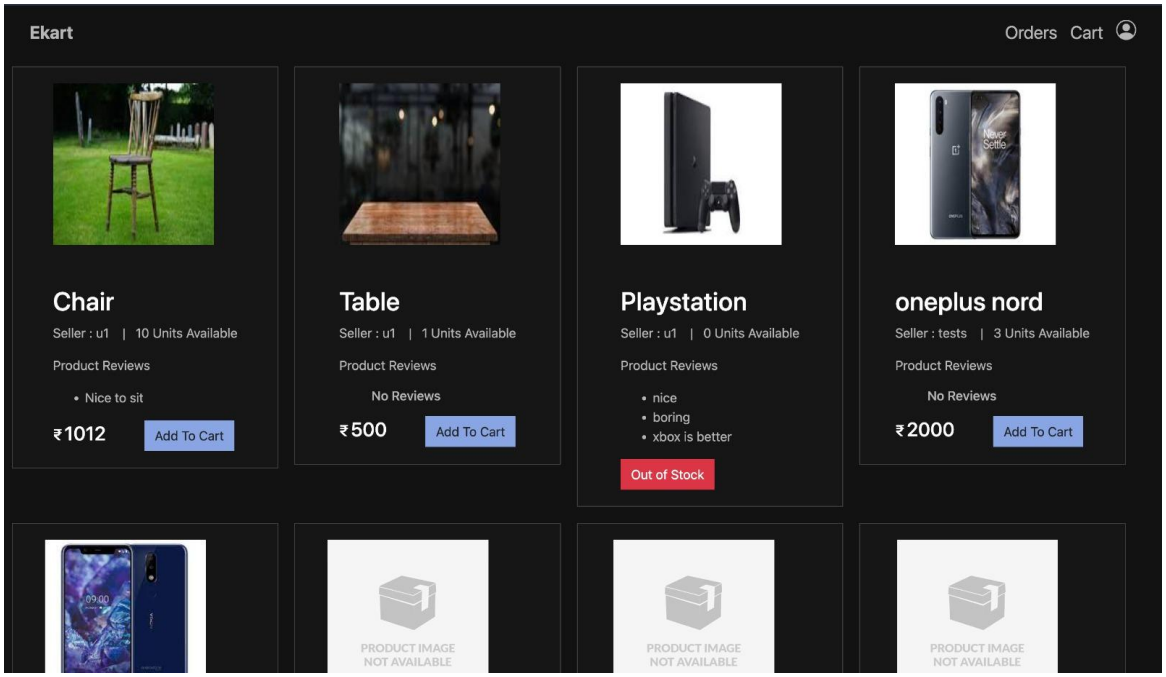
Login Page



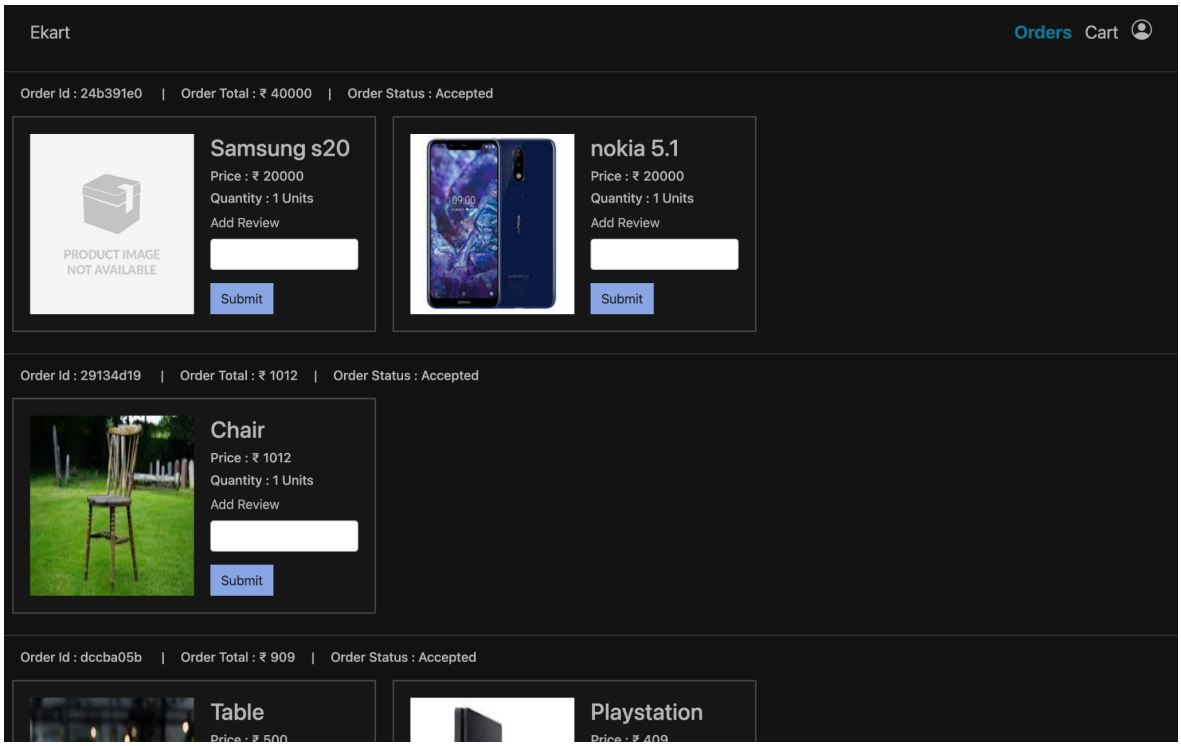
Register Page : Can register as seller or user



Home Page : Listing of all products



Orders page : User’s orders are displayed , can add reviews here once ordered



Cart Page : Products added to cart