### **1. Cloning the Repository and Initial Setup**

git clone https://github.com/sriram-R-krishnan/devops-build

### **2. Dockerizing the Application**

#### **Dockerfile:**

Create a Dockerfile in the root of your project:

# Stage 1 - Build React Application

FROM node:14-alpine as build

WORKDIR /app

COPY package.json package-lock.json ./

RUN npm install

COPY . .

RUN npm run build

# Stage 2 - Serve React Application using Nginx

FROM nginx:alpine

COPY --from=build /app/build /usr/share/nginx/html

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]

#### **docker-compose.yml:**

Create a docker-compose.yml file in the root of your project:

version: '3'

services:

app:

build: .

ports:

- "80:80"

### **3. Bash Scripting**

Also  
  
version: '3'

services:

app:

build: .

ports:

- "80:80"

### **4. Version Control**

Make sure to add a .gitignore file to ignore node\_modules and other generated files. Also, add a .dockerignore file to exclude unnecessary files from the Docker build context.

### **5. Docker Hub**

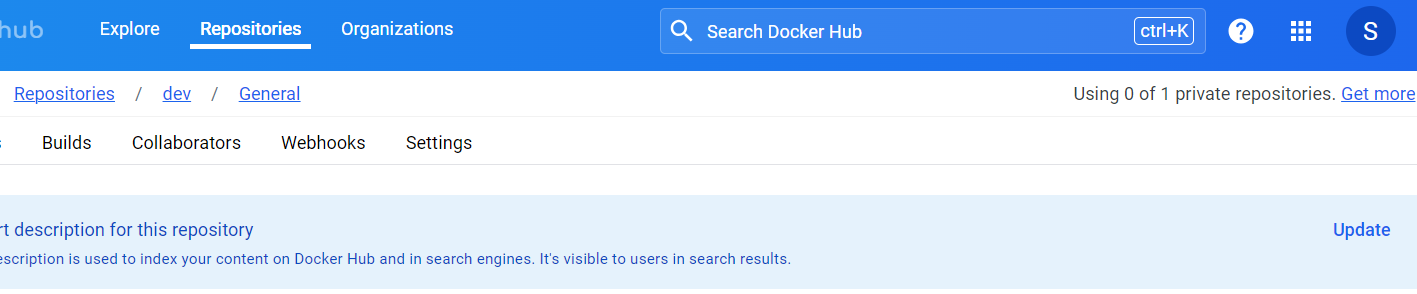
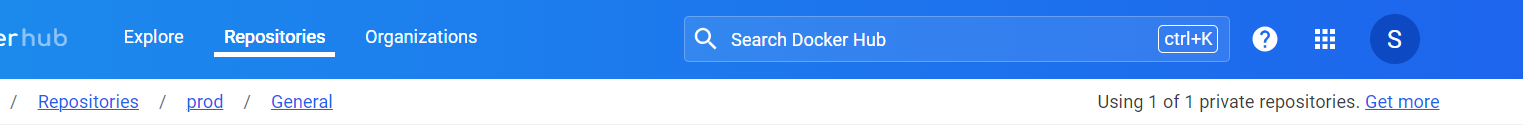
docker login

docker tag saipuja1996/dev:latest

docker push saipuja1996/dev:latest

# For prod, make sure to tag and push as well

docker tag saipuja1996/prod:latest

docker push saipuja1996/prod:latest  
  
  
  
  


### **6. Jenkins**

Install Jenkins and configure build steps to build Docker images, push them to Docker Hub, and deploy them to AWS. Configure Jenkins to trigger builds on both dev and master branches.  
  
  
Jenkins  
  
pipeline {

agent any

stages {

stage('Build Docker Image') {

steps {

sh 'bash build.sh'

}

}

stage('Push Docker Image to Dev') {

when {

branch 'dev'

}

steps {

script {

docker.withRegistry('https://registry.hub.docker.com', 'saipuja1996') {

docker.image(saipuja1996/dev:latest').push()

}

}

}

}

stage('Push Docker Image to Prod') {

when {

branch 'master'

}

steps {

script {

docker.withRegistry('https://registry.hub.docker.com', 'saipuja1996') {

docker.image(saipuja1996/prod:latest').push()

}

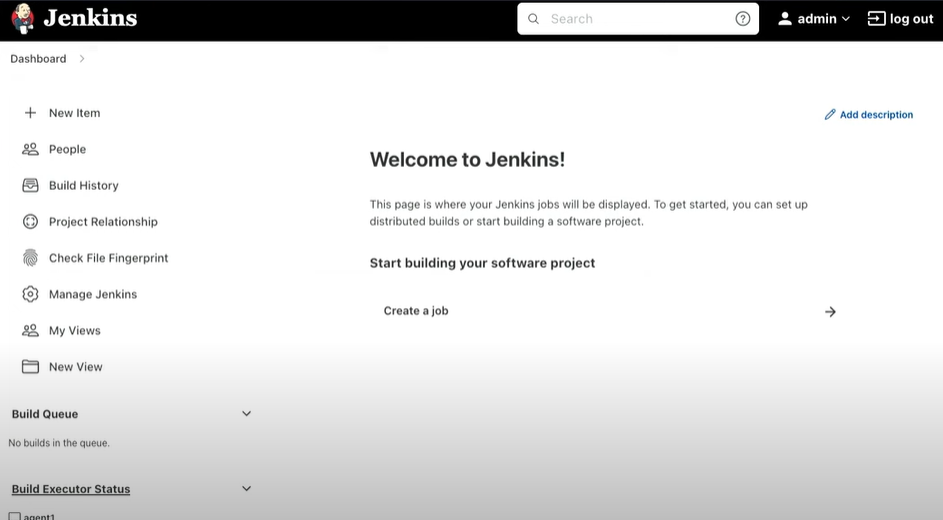
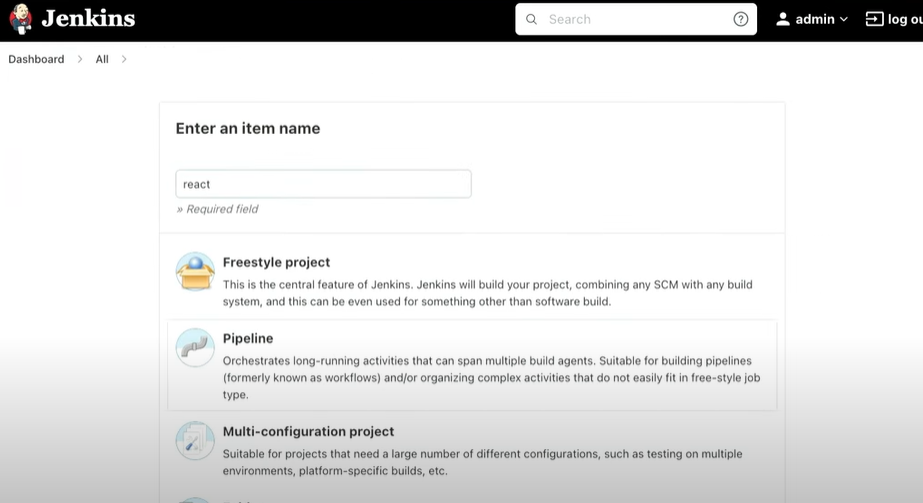
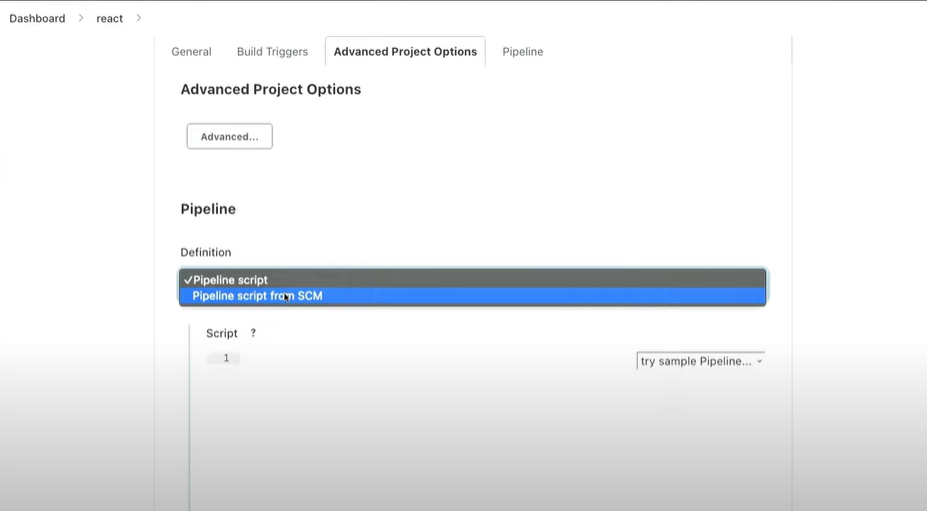
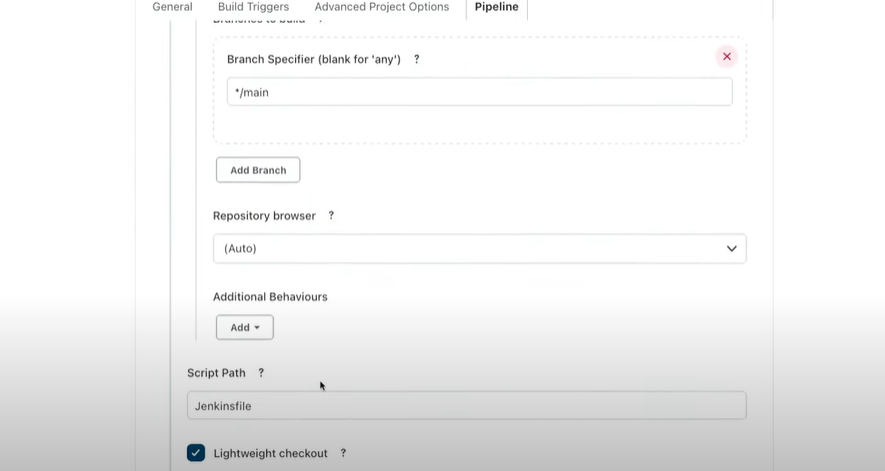
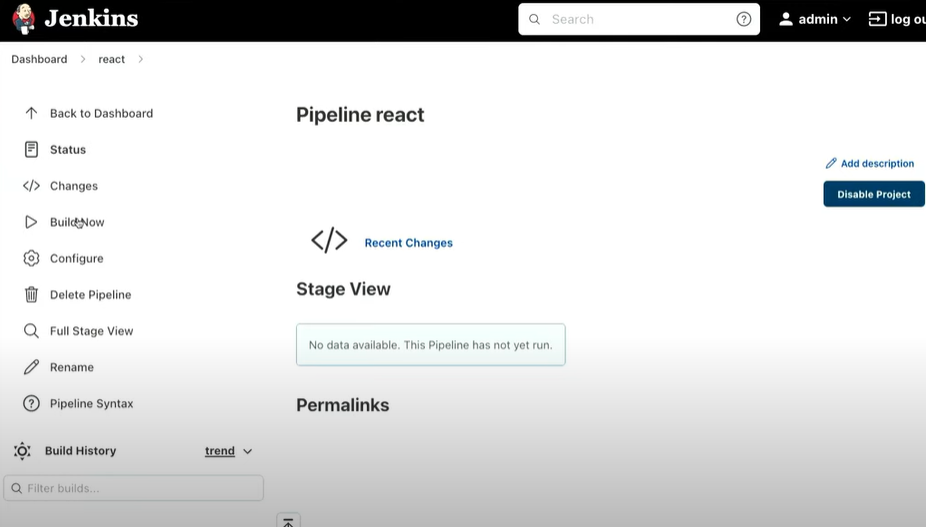
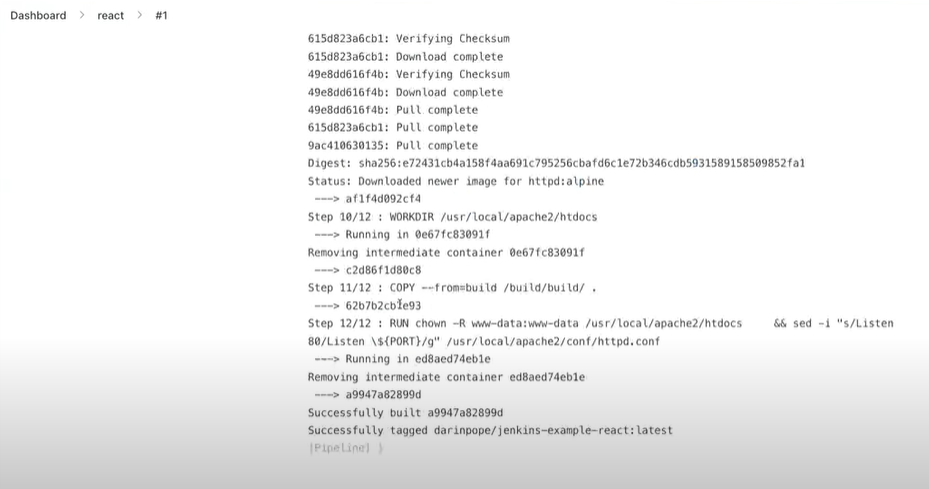
}

}

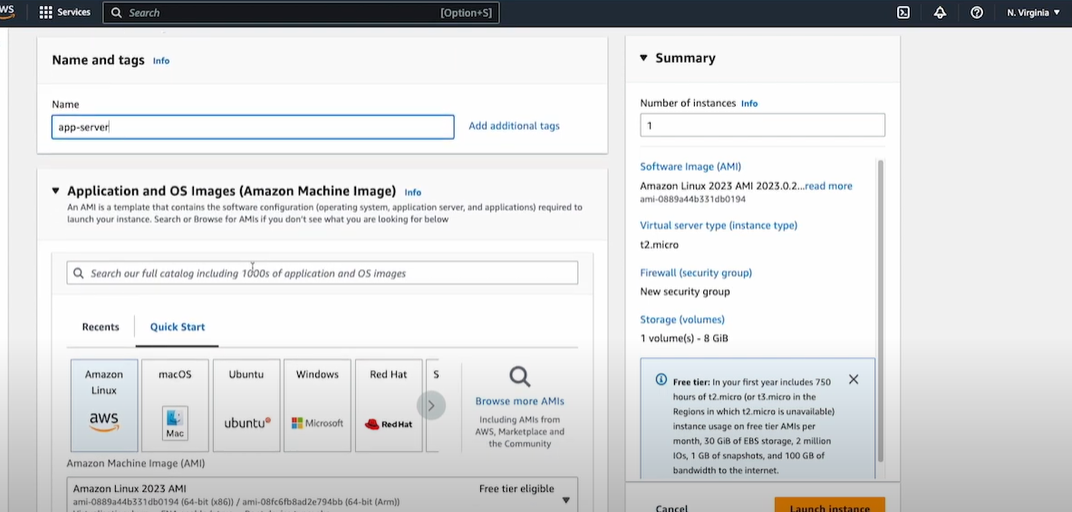
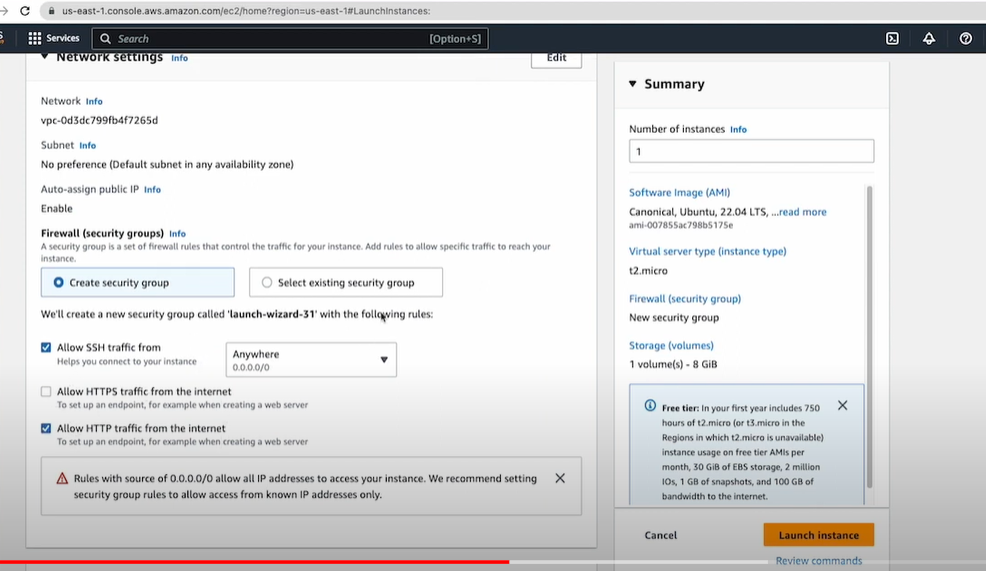
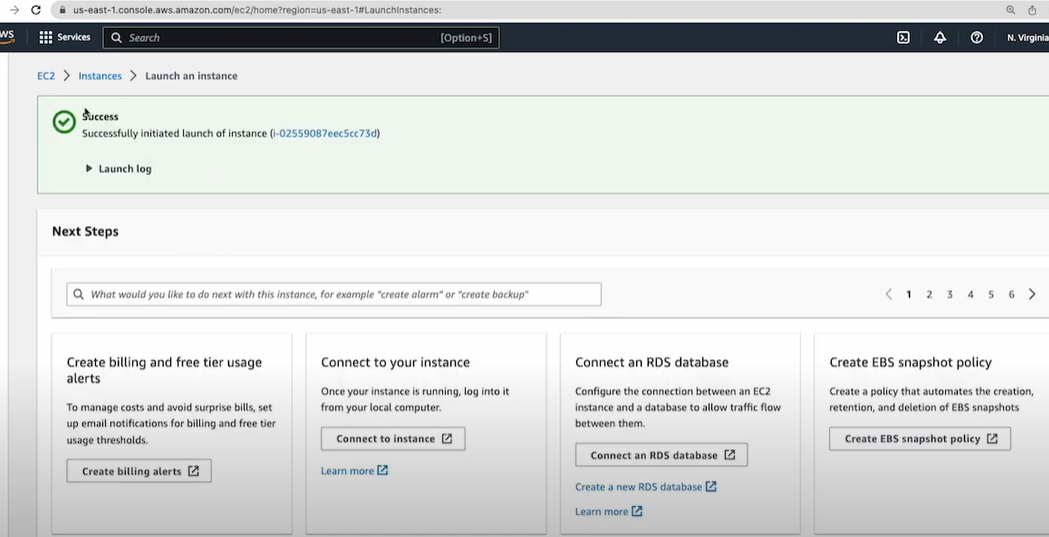
}

}

}

### **7. AWS**

Launch an EC2 instance (t2.micro) and configure Security Groups to allow access to port 80 from all IPs, but restrict SSH access to only your IP.  
  
  
  
  
  


### **8. Monitoring**

Set up monitoring using an open-source tool like Prometheus. Configure alerts to notify when the application goes down.  
  
  
# prometheus.yml

alerting:

alertmanagers:

- static\_configs:

- targets:

- alertmanager:9093

rule\_files:

- alerts.yml

# prometheus.yml

alerting:

alertmanagers:

- static\_configs:

- targets:

- alertmanager:9093

rule\_files:

- alerts.yml

global:

resolve\_timeout: 1m

route:

receiver: 'email-notifications'

receivers:

- name: 'email-notifications'

email\_configs:

- to: saipuja\_1996@gmail.com

from: saipuja#1996@gmail.com

smarthost: smtp.gmail.com:587

auth\_username: saipuja#1996@gmail.com

auth\_identity: saipuja#1996@gmail.com

auth\_password: saipuja#1996

send\_resolved: true

