**Graded Project on Orchestration and Scaling**

Step 1: Set Up the AWS Environment

1. Set Up AWS CLI and Boto3:

* Install AWS CLI and configure it with AWS credentials.
* Install Boto3 for Python and configure it.
  + pip install boto3
  + sudo apt install awscli

Step 2: Prepare the MERN Application

1. Containerize the MERN Application:

* Ensure the MERN application is containerized using Docker. Create a Dockerfile for each component (frontend and backend).

1. Push Docker Images to Amazon ECR:

* Build Docker images for the frontend and backend.
  + Create Dockerfile for all three services (code on git)
  + Build docker image
    - cd backend/helloservice
    - docker build -t samplemernwithmicroservices\_helloservice .
    - cd backend /profileservice
    - docker build -t samplemernwithmicroservices\_profileservice .
    - cd /frontend
    - docker build -t samplemernwithmicroservices\_frontend .

Additionally, we can use docker-compose.yml to run and build the images.

* Create an Amazon ECR repository for each image.
  + aws configure
  + aws ecr create-repository --repository-name helloservice --region ap-south-1
  + aws ecr create-repository --repository-name profileservice --region ap-south-1
  + aws ecr create-repository --repository-name frontend --region ap-south-1
* Push the Docker images to their respective ECR repositories.
  + aws ecr get-login-password --region ap-south-1 | docker login --username AWS --password-stdin 471112564396.dkr.ecr.ap-south-1.amazonaws.com
  + docker tag samplemernwithmicroservices\_helloservice:latest 471112564396.dkr.ecr.ap-south-1.amazonaws.com/helloservice:latest
  + docker tag samplemernwithmicroservices\_profileservice:latest 471112564396.dkr.ecr.ap-south-1.amazonaws.com/profileservice:latest
  + docker tag samplemernwithmicroservices\_frontend:latest 471112564396.dkr.ecr.ap-south-1.amazonaws.com/frontend:latest
  + docker push 471112564396.dkr.ecr.ap-south-1.amazonaws.com/helloservice:latest
  + docker push 471112564396.dkr.ecr.ap-south-1.amazonaws.com/profileservice:latest
  + docker push 471112564396.dkr.ecr.ap-south-1.amazonaws.com/frontend:latest

Step 3: Version Control

1. Use AWS CodeCommit:

* Create a CodeCommit repository.

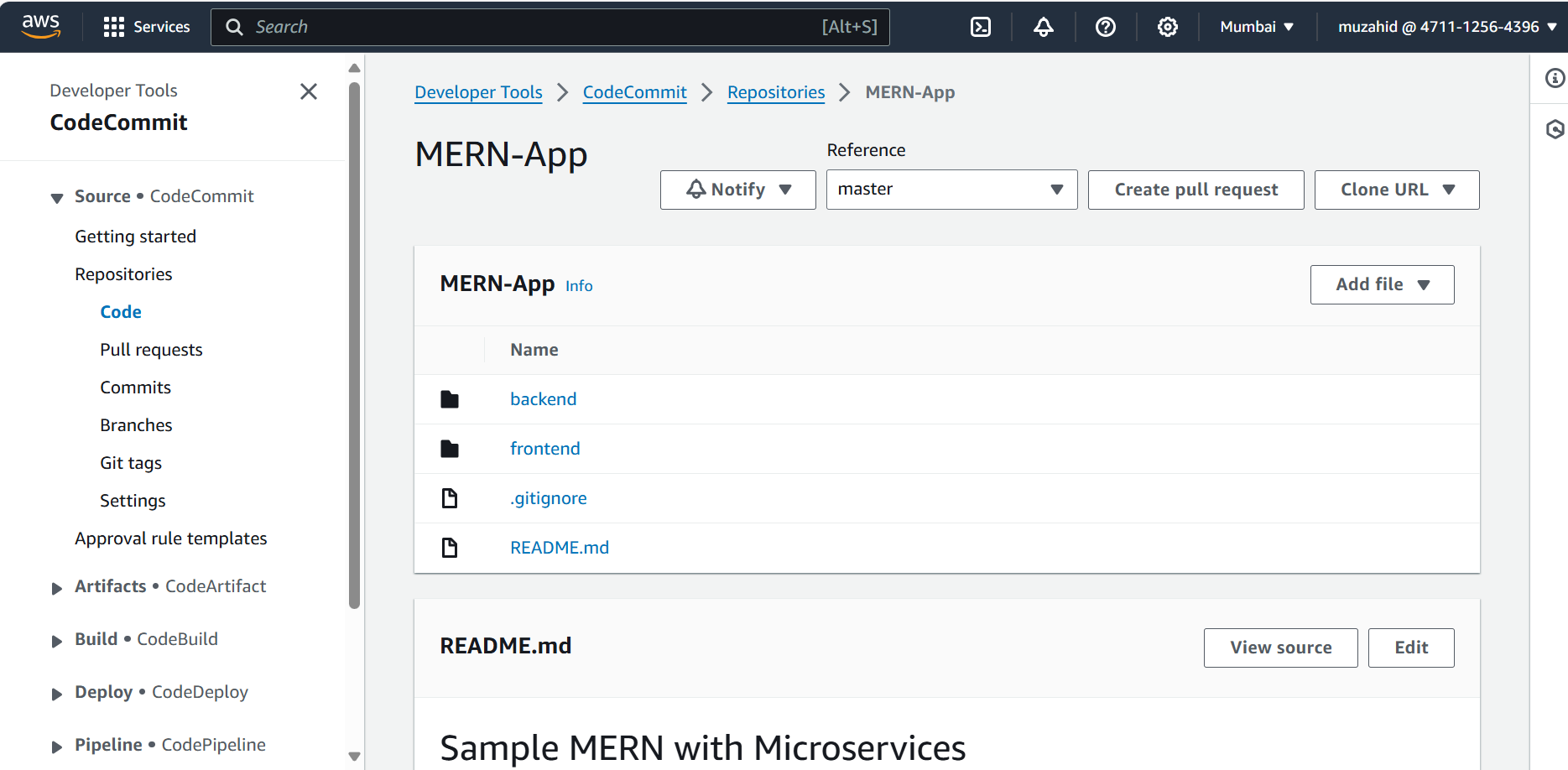
A screenshot of a computer

Description automatically generated

* Push the MERN application source code to the CodeCommit repository.
  + git init
  + git remote add origin <https://git-codecommit.ap-south-1.amazonaws.com/v1/repos/MERN-App>
  + git add .
  + git commit -m "First Commit"
  + git push -u origin main

A screenshot of a computer

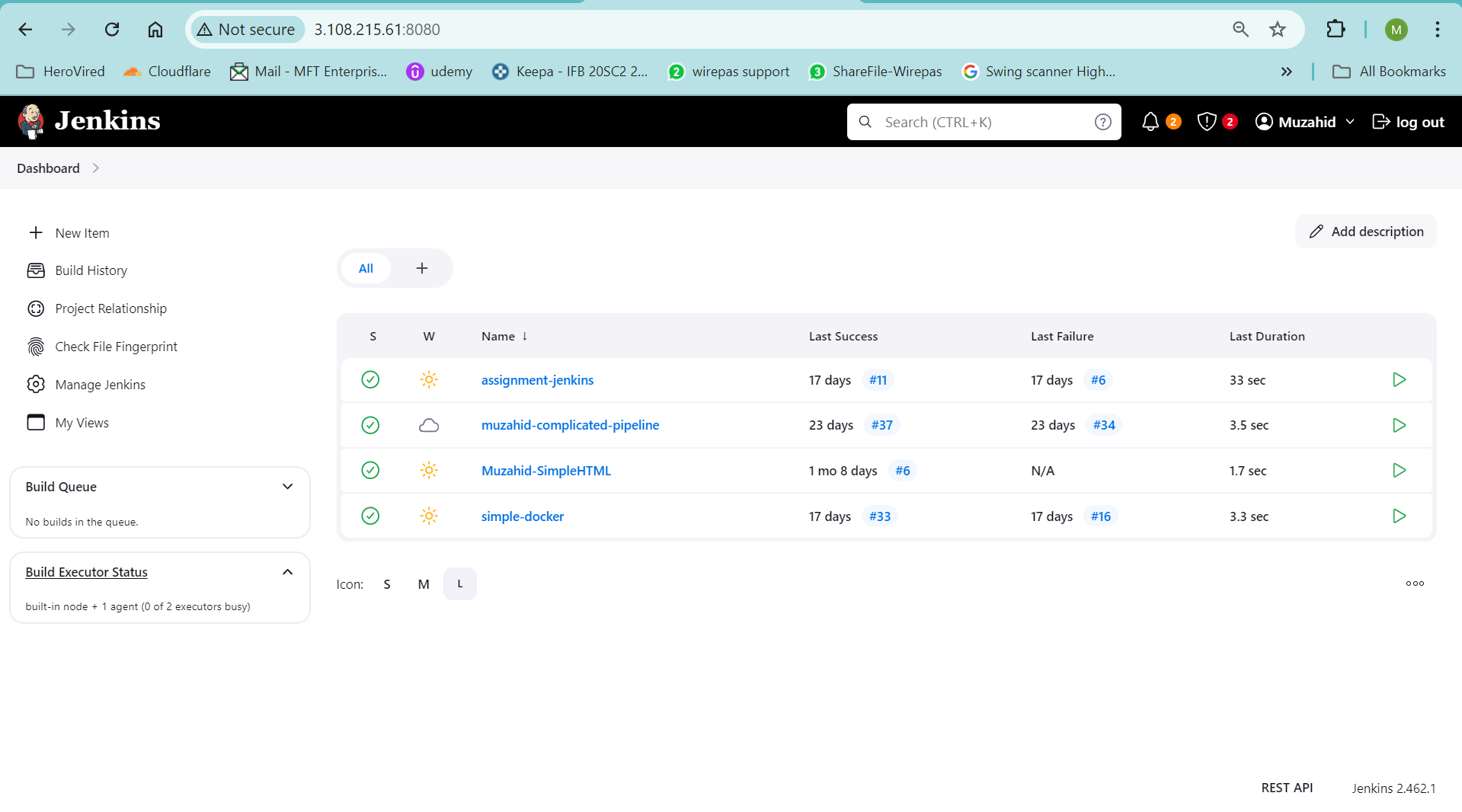
Description automatically generated



Step 4: Continuous Integration with Jenkins

1. Set Up Jenkins:

* Install Jenkins on an EC2 instance.
* Configure Jenkins with necessary plugins.



1. Create Jenkins Jobs:

* Create Jenkins jobs for building and pushing Docker images to ECR.
* Trigger the Jenkins jobs whenever there's a new commit in the Code Commit repository.

1. Add AWS Accesskey and codecommit HTTPS credential in Jenkins credential manager.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Create Jenkins Pipeline

A screenshot of a computer

Description automatically generated

1. The Jenkins code is in github repo.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Step 5: Infrastructure as Code (IaC) with Boto3

1. Define Infrastructure with Boto3 (Python Script):

* Use Boto3 to define the infrastructure (VPC, subnets, security groups).
* Define an Auto Scaling Group (ASG) for the backend.
* Create AWS Lambda functions if needed.

Code added in Github repo.

Step 6: Deploying Backend Services

1. Deploy Backend on EC2 with ASG:

* Use Boto3 to deploy EC2 instances with the Dockerized backend application in the ASG.

Code added in Github repo.

Step 7: Set Up Networking

1. Create Load Balancer:

* Set up an Elastic Load Balancer (ELB) for the backend ASG.

1. Configure DNS:

* Set up DNS using Route 53 or any other DNS service.

1. Create Target group for both helloservice and profileservice for port 3001 and 3002 respectively.

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

1. Create Network load balancer and attach both target group

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Step 8: Deploying Frontend Services

1. Deploy Frontend on EC2:

* Use Boto3 to deploy EC2 instances with the Dockerized frontend application.

Code added in Github repo.

Step 9: AWS Lambda Deployment

1. Create Lambda Functions:

* Use Boto3 to create AWS Lambda functions for specific tasks within the application.
* Backup of Db using Lambda Functions and store in S3 bucket - put time stamping on the backup

Step 10: Kubernetes (EKS) Deployment

1. Create EKS Cluster:

* Use eksctl or other tools to create an Amazon EKS cluster.

1. Create EKS cluster

*eksctl create cluster --name my-cluster --region us-west-2 --nodegroup-name my-nodes --node-type t3.medium --nodes 2 --nodes-min 1 --nodes-max 4 –managed*

1. Update kubeconfig

*aws eks --region us-west-2 update-kubeconfig --name my-cluster*

1. Verify the cluster

*Kubectl get nodes*

1. Deploy Application with Helm:

* Use Helm to package and deploy the MERN application on EKS.

Helm file is added in github repository

Step 11: Monitoring and Logging

1. Set Up Monitoring:

* Use CloudWatch for monitoring and setting up alarms.

1. Configure Logging:

* Use CloudWatch Logs or another logging solution for collecting logs.