React App Deployment - Three-Tier Architecture

Introduction

This document outlines the deployment and CI/CD configuration of a React-based web application using a secure, scalable, and maintainable three-tier architecture on AWS. The architecture comprises a **Frontend Tier**, where the React application is served via NGINX on an Ubuntu EC2 instance in a public subnet; an **Application Tier**, with backend services deployed in the first private subnet; and a **Database Tier**, featuring a MongoDB instance in the second private subnet. To streamline the implemented using AWS CodePipeline, CodeBuild, and CodeDeploy, enabling continuous integration and delivery of updates across all tiers within a properly segmented Virtual Private Cloud (VPC) environment.

Aim

To implement and deploy a secure, scalable, and maintainable web application on AWS using a three-tier architecture with proper network segmentation, and to automate the CI/CD pipeline using AWS services for efficient and continuous delivery.

Objective

- Design and configure a secure Virtual Private Cloud (VPC) with one public subnet and two private subnets to enable network-level isolation.
- Deploy the React frontend in the public subnet using NGINX on an Ubuntu EC2 instance for external access.
- Host the backend application in the first private subnet, ensuring it is only accessible internally from the application and frontend tiers.
- Deploy the MongoDB database in the second private subnet with restricted access to backend services only.
- Implement a CI/CD pipeline using AWS CodePipeline, CodeBuild, and CodeDeploy to automate the build, test, and deployment processes across all tiers, ensuring reliable and continuous delivery.

Architecture Overview

A three-tier architecture breaks the application into three logical layers:

1. Frontend (Presentation Tier) - React app served via Nginx, hosted in a Public Subnet.
2. Frontend (Application Tier) - Node.js/Express, hosted in a Private Subnet 1.
3. Backend and Database (Data Tier) - MongoDB, hosted in a Private Subnet 2.

Network Design (VPC)

- VPC: 10.0.0.0/16
- Public Subnet: 10.0.1.0/24
- Private Subnet 1: 10.0.2.0/24
- Private Subnet 2: 10.0.3.0/24
- Internet Gateway: Attached to VPC
- NAT Gateway: In public subnet
- Route Tables: Properly associated

EC2 Instances

- Frontend Ubuntu(public) (Nginx +React)
- Frontend (Private) Ubuntu (Nginx +React)
- Backend and Database (private):
  Ubuntu (mongodb)

Configuration Steps

Frontend (Public):

- Install Nginx: sudo apt install nginx -y
- Deploy React build
- Configure Nginx: reverse proxy /api to backend.

Frontend (Private):

- Install Node
- Use PM2 to run server
- Allow access only from frontend subnet.

Backend and Database (Private):

- Install DB (MongoDB), create DB and user
- Allow access only from backend subnet

Security Group Rules

Frontend(Public) : 0.0.0.0/0

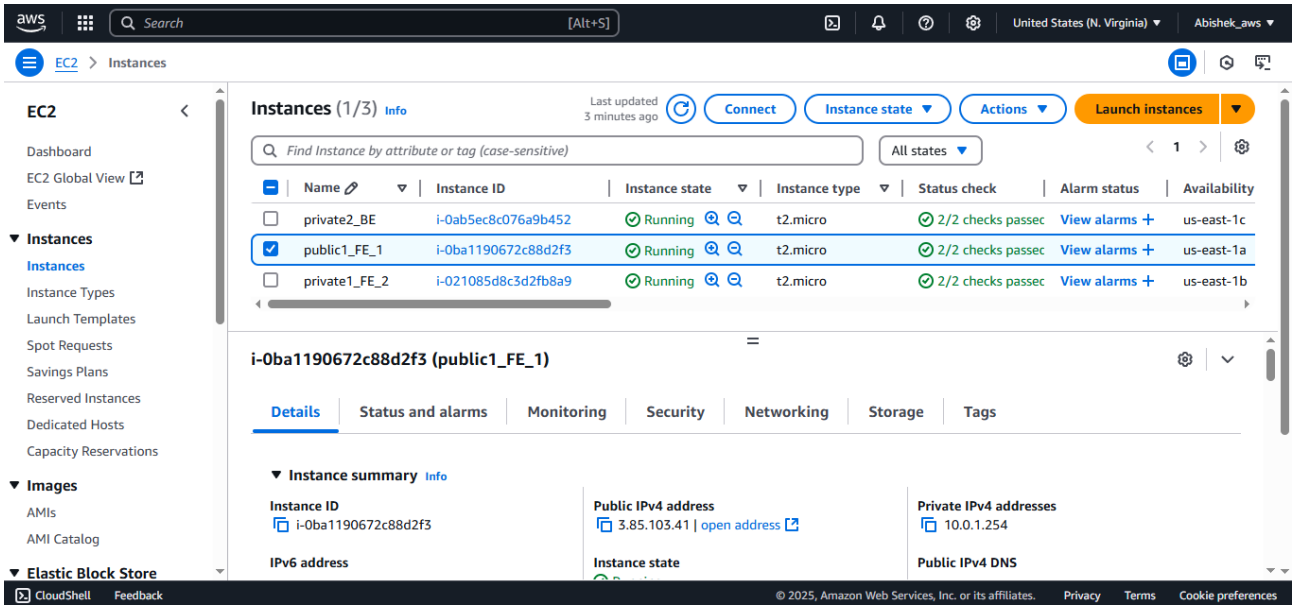
Frontend(Private) : 10.0.2.214/32

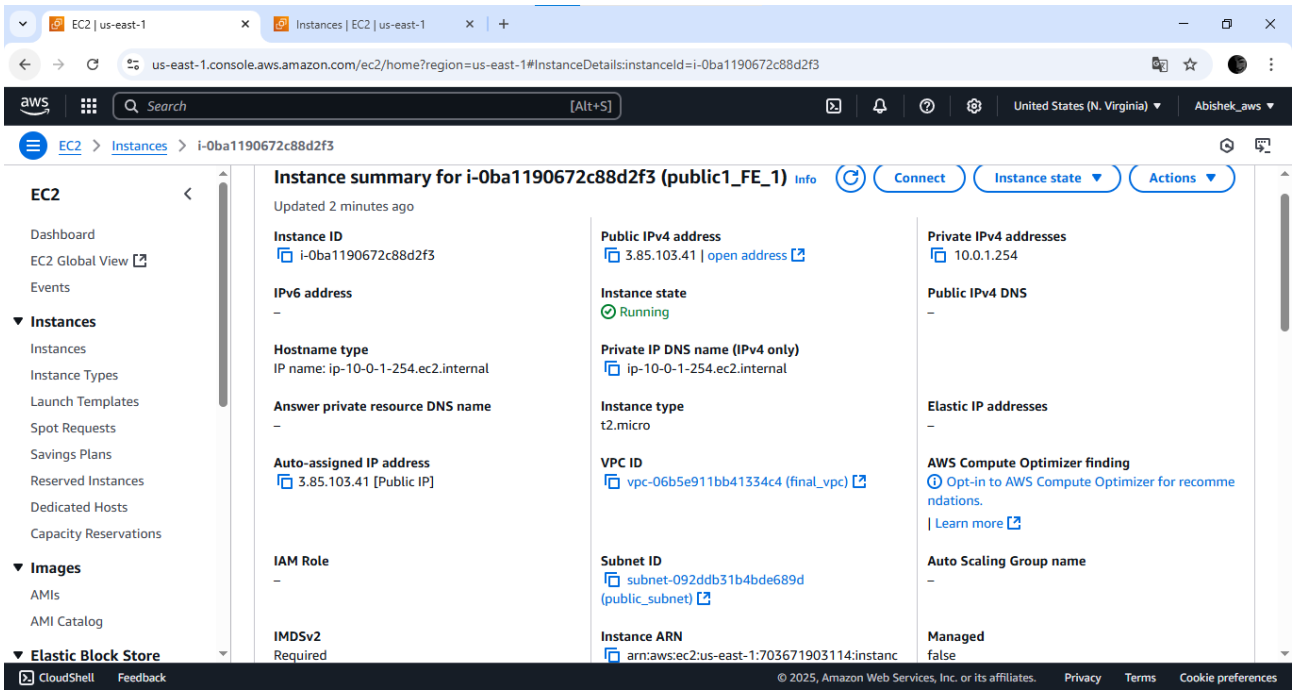
Backend & Database: 10.0.3.122/32

Deployment Verification

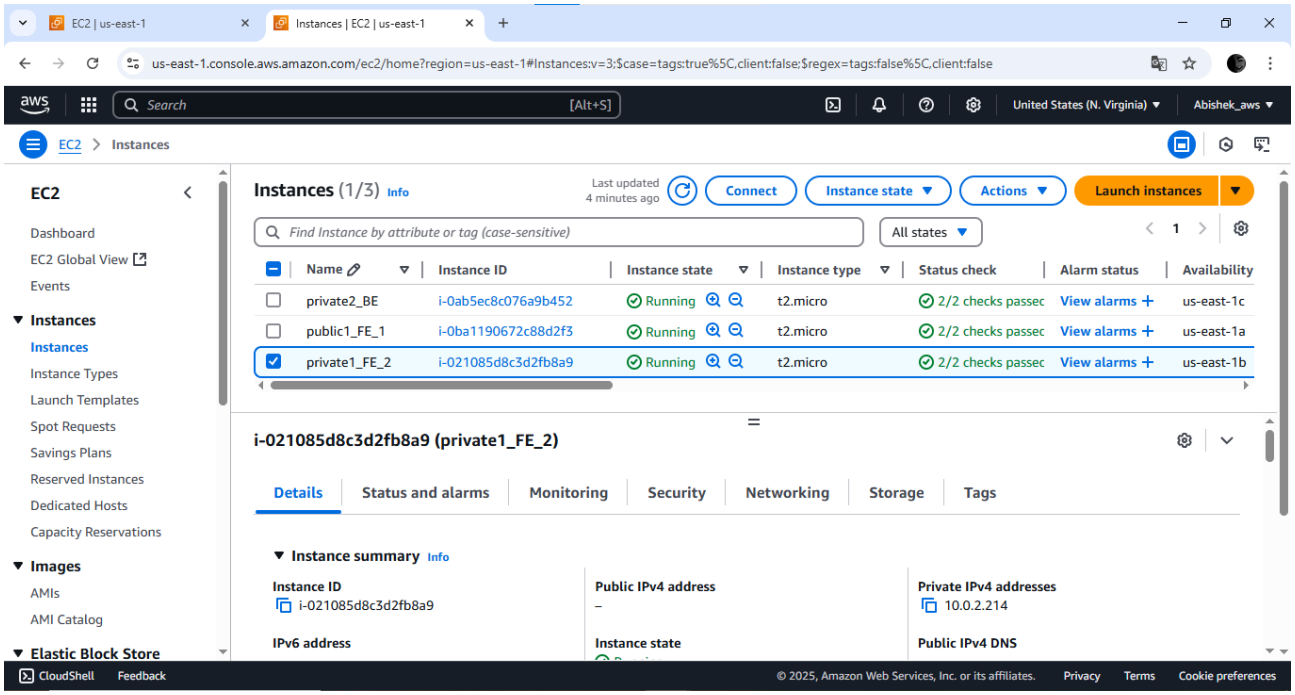
1. Access React app via public IP/domain.
2. React calls backend via /api.
3. Backend connects to DB securely.

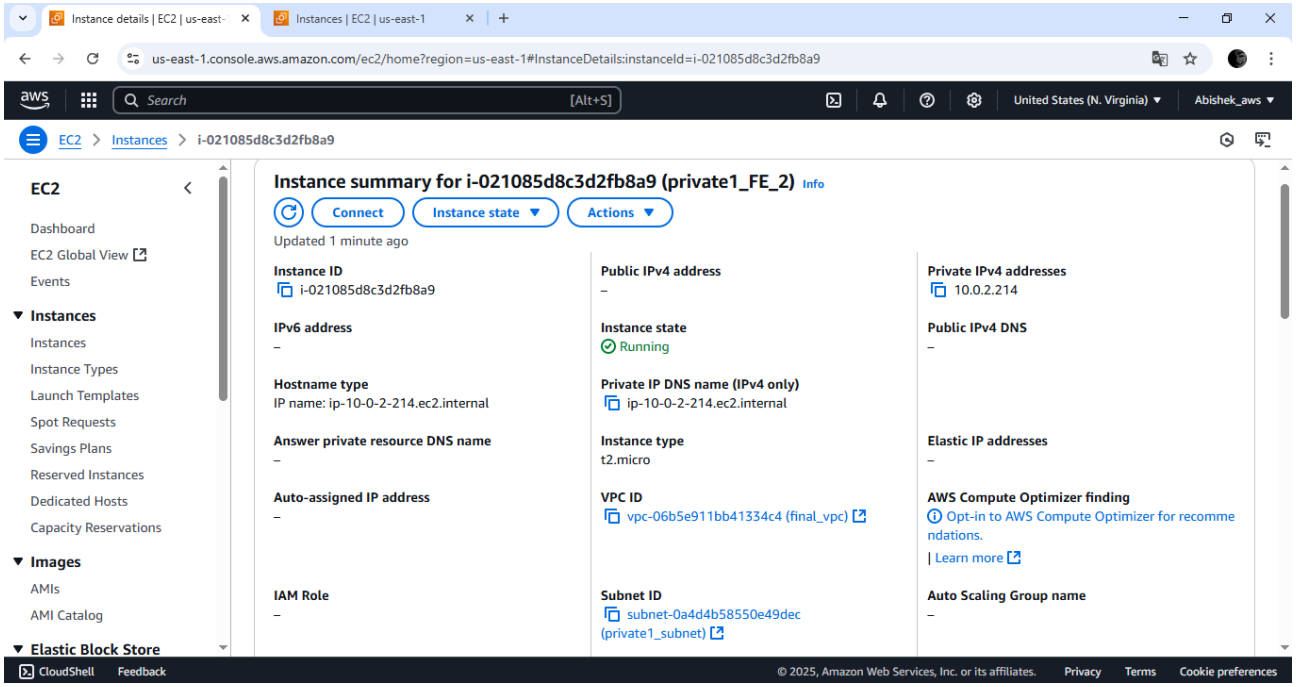
Frontend Ubuntu(public) (Nginx +React)



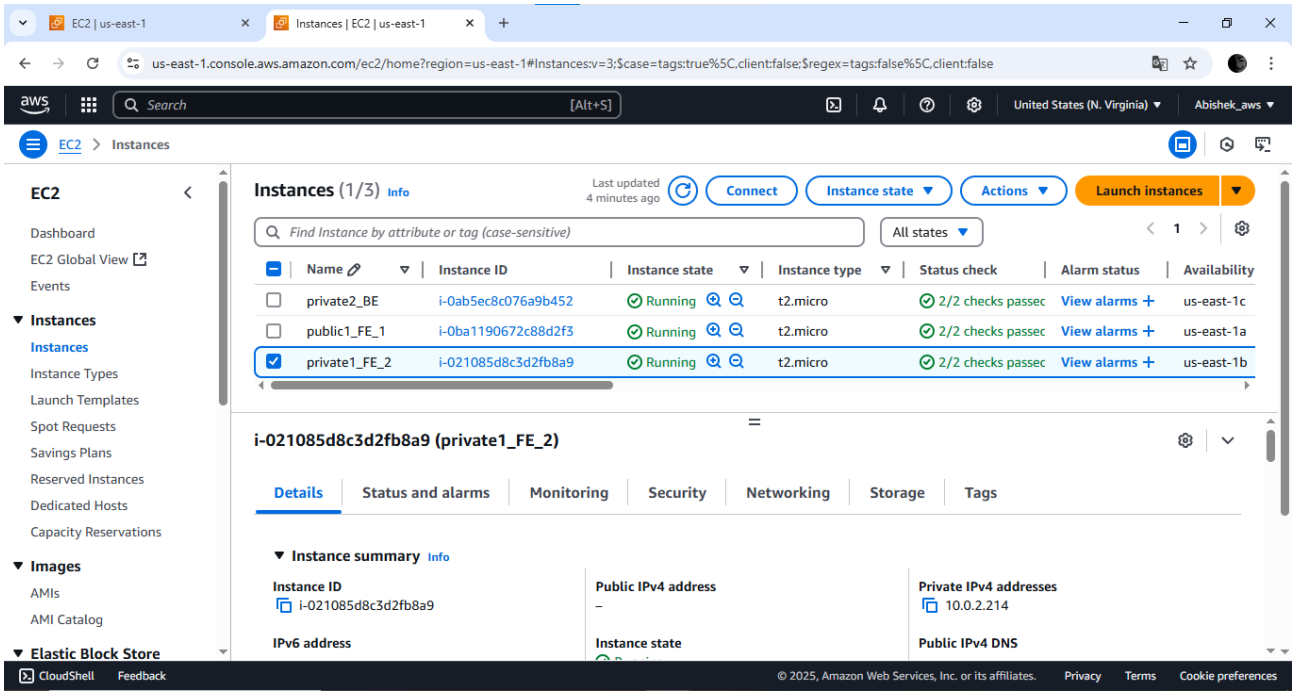


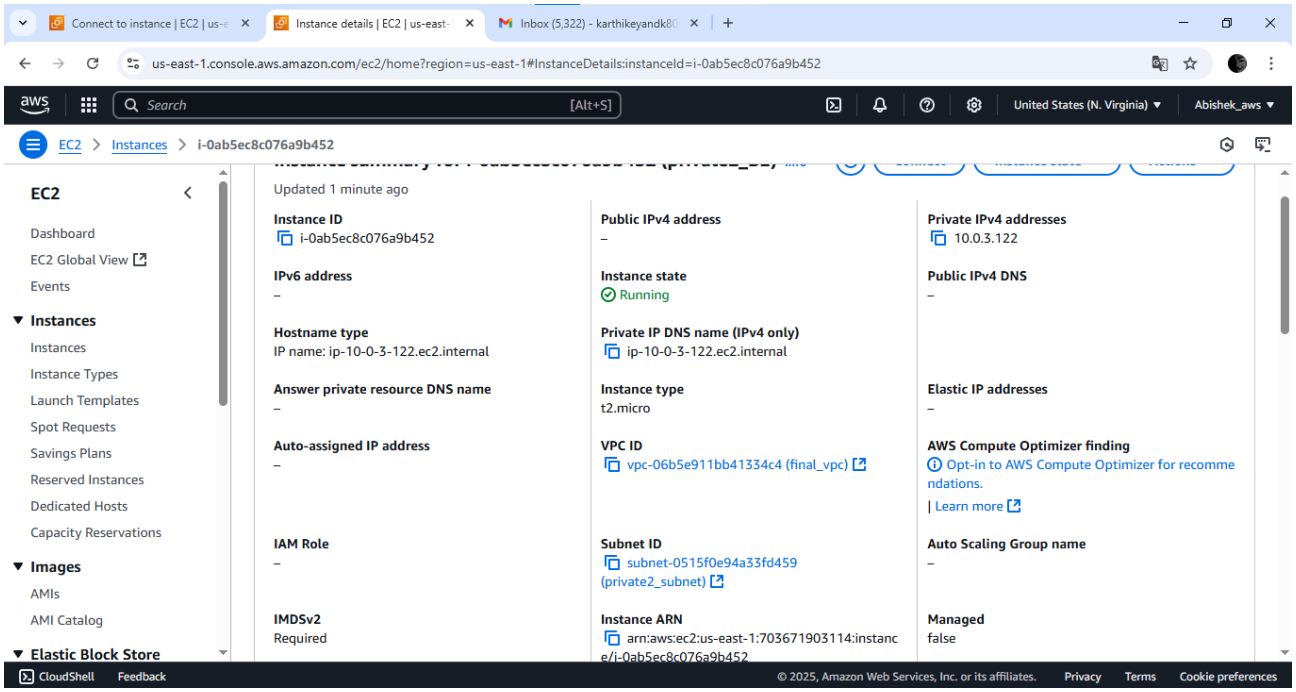
Frontend (Private)





Backend and Database (Private) (mongodb):





Public configuration:

```
ubuntu@ip-10-0-1-254: ~
  GNU nano 7.2                                  /etc/nginx/sites-available/default *
server {
    listen 80 default_server;
    listen [::]:80 default_server;

    root /var/www/html/First-App-FE/build;

    index index.html index.htm index.nginx-debian.html;
    server_name _;
    location / {
        try_files $uri /html;
    }

    location /view-backups/ {
        proxy_pass http://10.0.2.214;
    }
    location /api/ {
        proxy_pass http://10.0.3.122;
    }

}
```

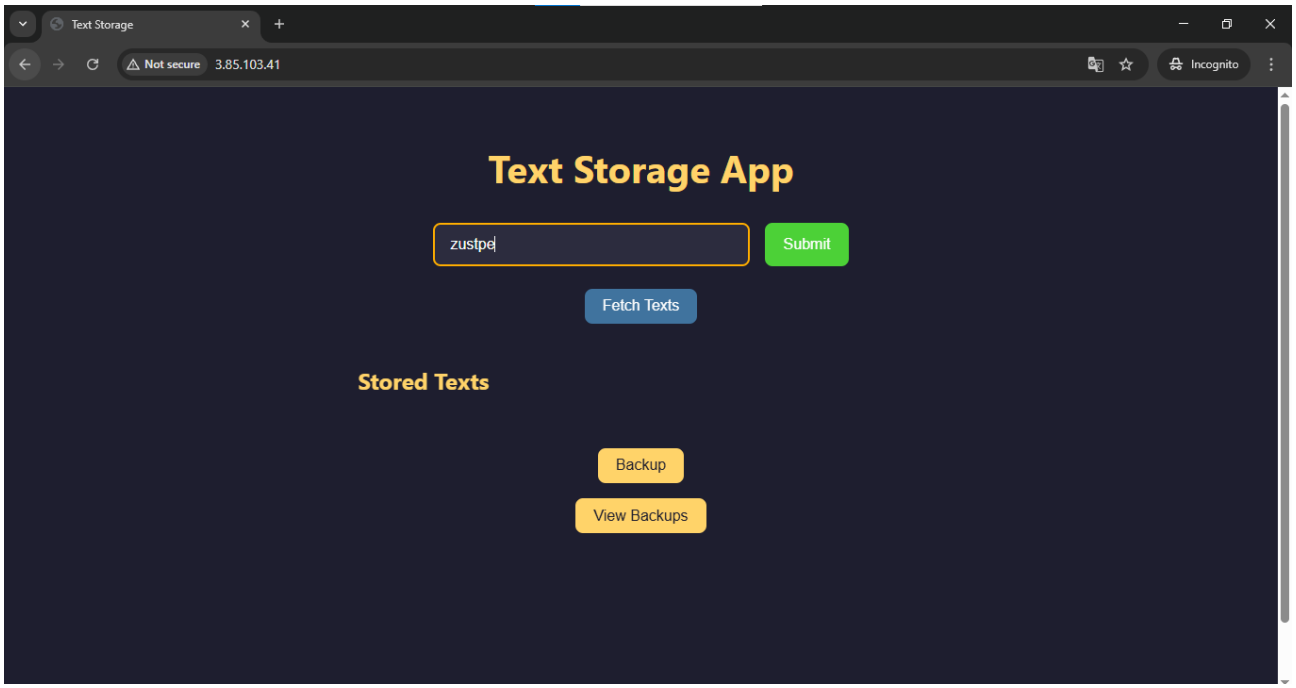
Private Configuration(Frontend):

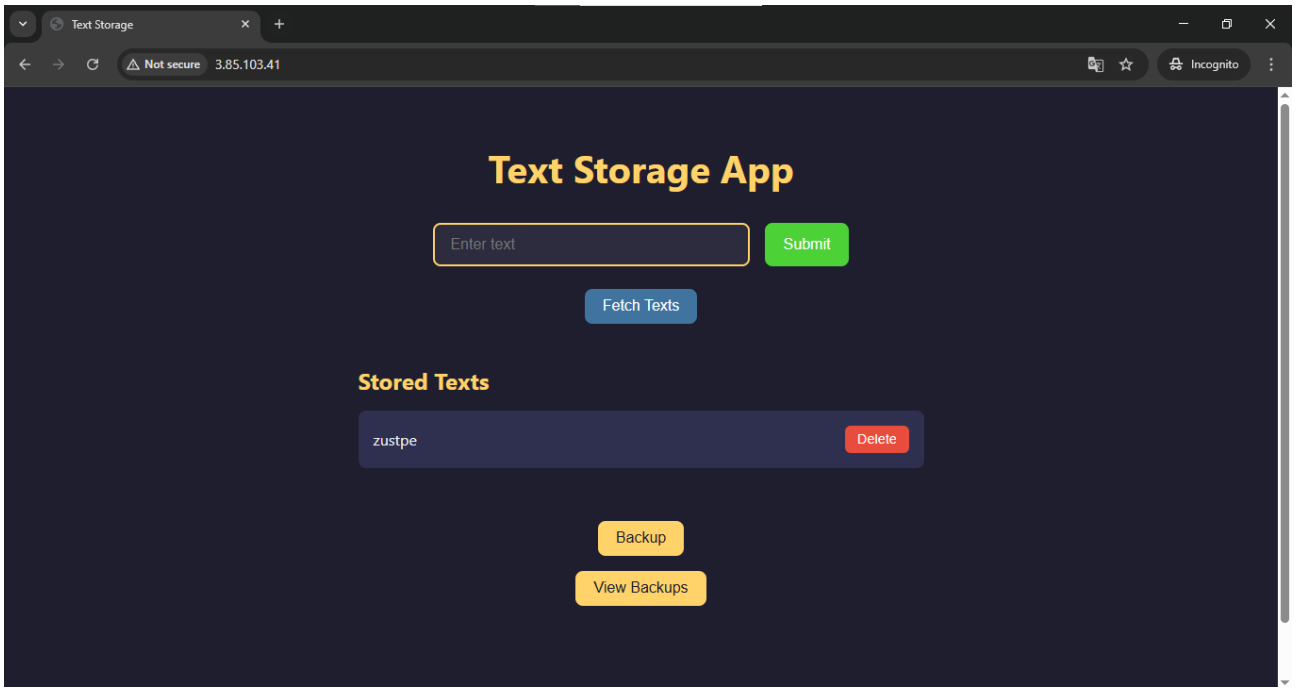
```
ubuntu@ip-10-0-2-214: ~
  GNU nano 7.2                                  /etc/nginx/sites-available/default *
server {
        listen 80 default_server;
        listen [::]:80 default_server;


        root /var/www/html/Second-App-FE/build;

        index index.html index.htm index.nginx-debian.html;

        server_name _;

        location /view-backups {
                try_files $uri $uri/ =404;
                alias /var/www/html/Second-App-FE/build;
        }
}
```

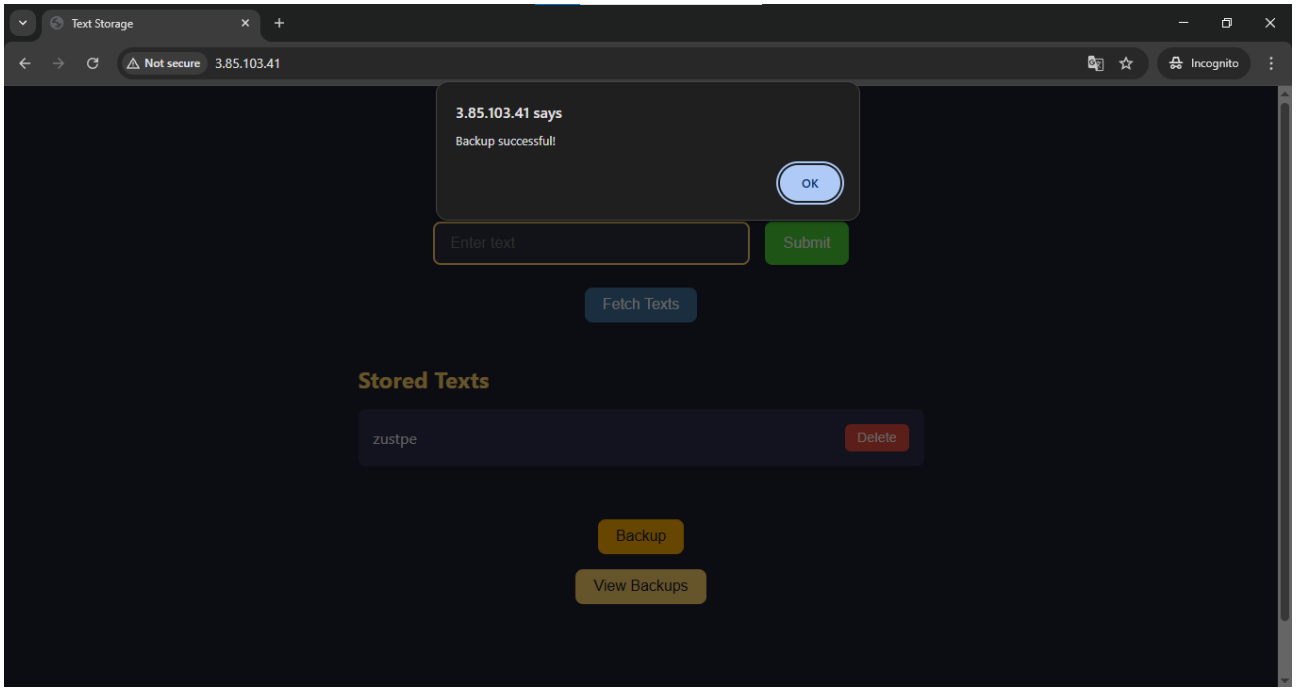
Private configuration(Backend and DB)

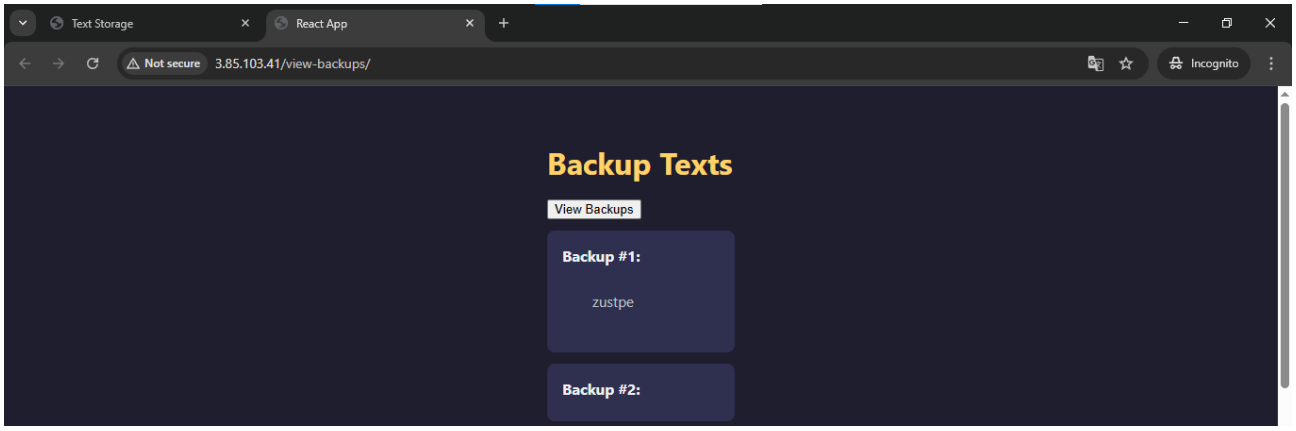
```
ubuntu@ip-10-0-3-122: ~
  GNU nano 7.2                                  /etc/nginx/sites-available/default
server {
        listen 80 default_server;
        listen [::]:80 default_server;

        root /var/www/html;

        index index.html index.htm index.nginx-debian.html;

        server_name _;

        location / {
                try_files $uri $uri/ =404;
        }
        location /api/ {
                proxy_pass http://localhost:5000;
        }
}
```

Output :




Text Storage
Not secure
3.85.103.41
Incognito
Text Storage App
Enter text
Submit
Fetch Texts
Stored Texts
zustpe
Delete
Backup
View Backups


Text Storage
Not secure
3.85.103.41
Incognito
3.85.103.41 says
Backup successful!
OK
Enter text
Submit
Fetch Texts
Stored Texts
zustpe
Delete
Backup
View Backups


Text Storage
React App
Not secure
3.85.103.41/view-backups/
Incognito
Backup Texts
View Backups
Backup #1:
zustpe
Backup #2:

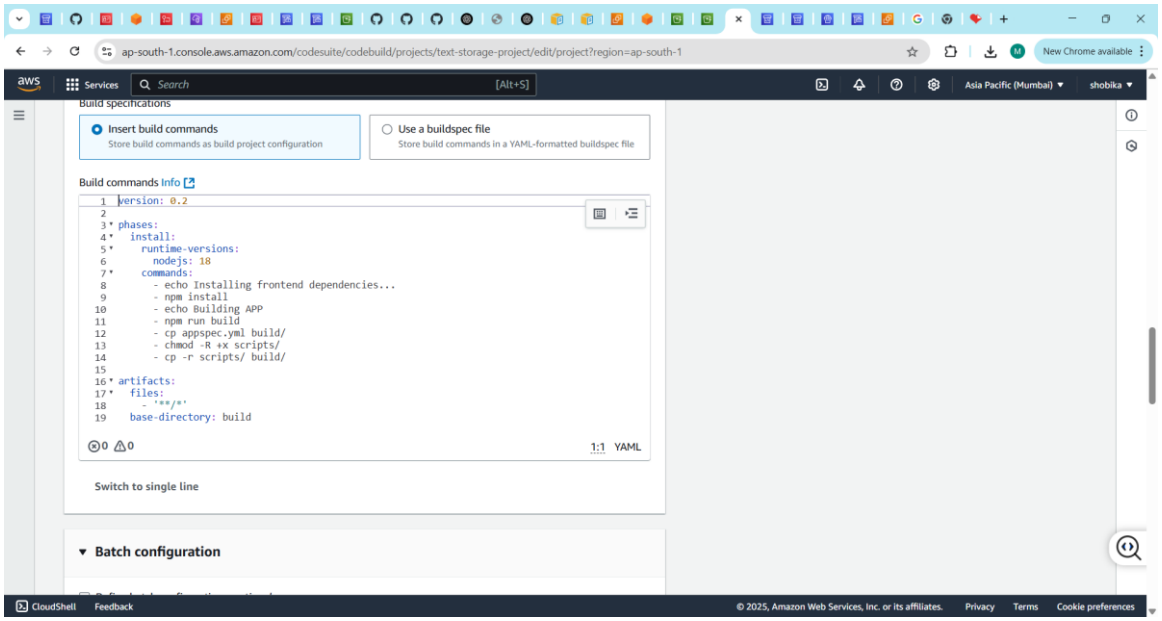
Pipeline Stages:

1. Source: Fetch from GitHub
2. Build: Execute tests and build artifacts
3. Deploy: Use CodeDeploy to deploy frontend to public subnet and backend to private subnet

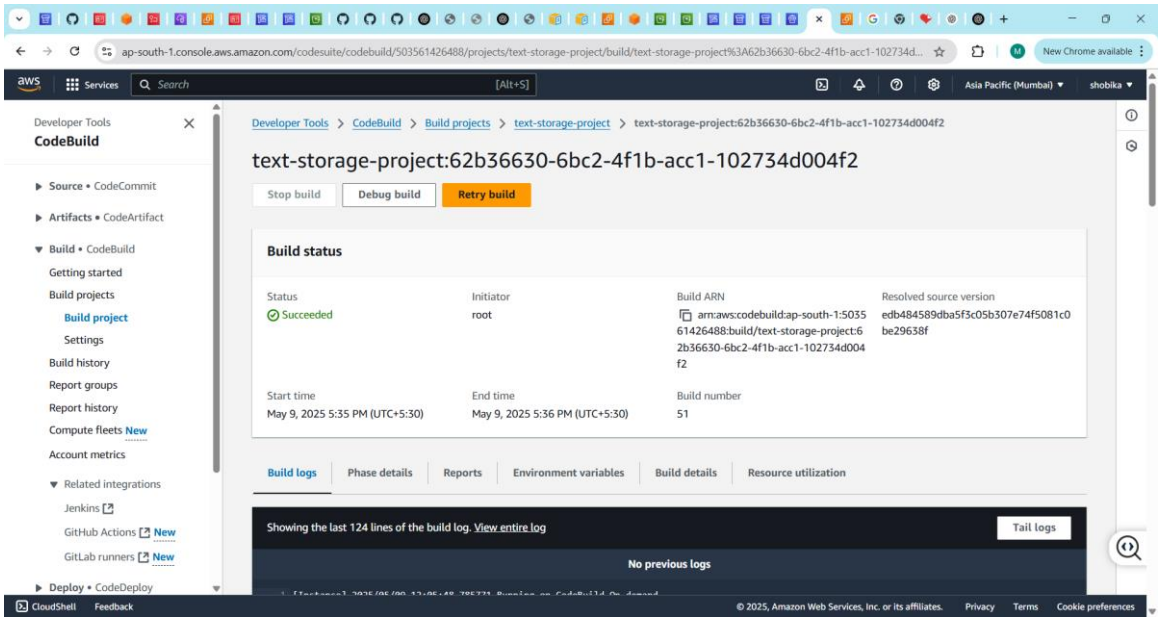
Source Code Integration:

- Connect GitHub to CodePipeline.

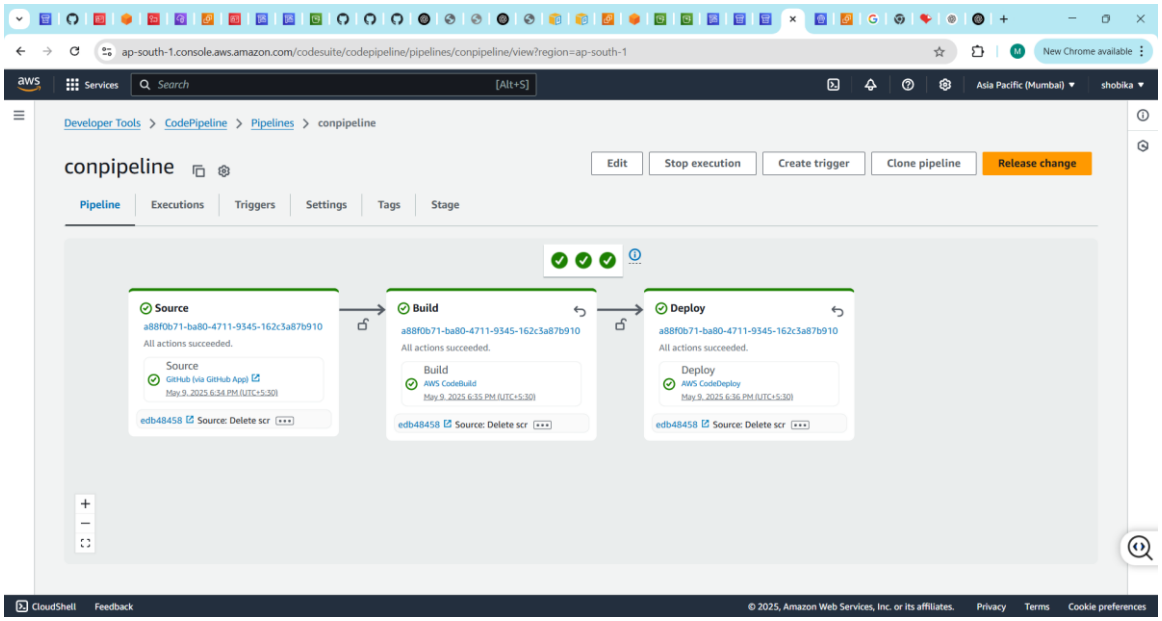
CodeBuild



Code Deploy



Code pipeline



Conclusion:

This configuration provides a secure and automated deployment pipeline for a three-tier application. The use of subnet segmentation ensures high security, and AWS CI/CD tools enhance deployment efficiency and reliability.