

# 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 implementation 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)

The screenshot shows the AWS EC2 Instances page. The left sidebar has sections for EC2, Dashboard, EC2 Global View, Events, Instances (selected), Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images (AMIs, AMI Catalog), and Elastic Block Store. The main content area shows 'Instances (1/3) Info' with a table of three instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability
private2_BE	i-0ab5ec8c076a9b452	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1c
public1_FE_1	i-0ba1190672c88d2f3	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1a
private1_FE_2	i-021085d8c3d2fb8a9	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b

Below the table, the details for instance i-0ba1190672c88d2f3 (public1\_FE\_1) are shown. The 'Details' tab is selected, displaying the following information:

- Instance summary:
  - Instance ID: i-0ba1190672c88d2f3
  - Public IPv4 address: 3.85.103.41 | open address
  - Private IPv4 addresses: 10.0.1.254
  - Public IPv4 DNS: public1\_FE\_1

The screenshot shows the AWS EC2 Instances details page for an instance named 'public1\_FE\_1'. The instance ID is i-0ba1190672c88d2f3. Key details include:

- Public IPv4 address:** 3.85.103.41
- Private IPv4 addresses:** 10.0.1.254
- Instance state:** Running
- VPC ID:** vpc-06b5e911bb41334c4 (final\_vpc)
- Subnet ID:** subnet-092ddb31b4bde689d (public\_subnet)
- Instance ARN:** arn:aws:ec2:us-east-1:703671903114:instance/i-0ba1190672c88d2f3

The left sidebar shows navigation links for EC2, Instances, Images, and Elastic Block Store.

## Frontend (Private)

The screenshot shows the AWS EC2 Instances list page. There are three instances listed:

Name	Instance ID	Instance State	Type	Status Checks	Availability
private2_BE	i-0ab5ec8c076a9b452	Running	t2.micro	2/2 checks passed	us-east-1c
public1_FE_1	i-0ba1190672c88d2f3	Running	t2.micro	2/2 checks passed	us-east-1a
private1_FE_2	i-021085d8c3d2fb8a9	Running	t2.micro	2/2 checks passed	us-east-1b

The instance 'private1\_FE\_2' is selected. Its details page is shown below, including:

- Instance ID:** i-021085d8c3d2fb8a9
- Public IPv4 address:** -
- Private IPv4 addresses:** 10.0.2.214

The screenshot shows the AWS EC2 Instances details page for instance `i-021085d8c3d2fb8a9`. The instance summary table contains the following data:

	Value		Value
Instance ID	<code>i-021085d8c3d2fb8a9</code>	Public IPv4 address	-
IPv6 address	-	Instance state	Running
Hostname type	IP name: <code>ip-10-0-2-214.ec2.internal</code>	Private IP DNS name (IPv4 only)	<code>ip-10-0-2-214.ec2.internal</code>
Answer private resource DNS name	-	Instance type	t2.micro
Auto-assigned IP address	-	VPC ID	<code>vpc-06b5e911bb41334c4 (final_vpc)</code>
IAM Role	-	Subnet ID	<code>subnet-0a4d4b58550e49dec (private1_subnet)</code>

Other visible sections include the EC2 navigation bar, a sidebar with links like Dashboard, Instances, and Images, and a footer with copyright information.

## Backend and Database (Private) (mongodb):

The screenshot shows the AWS EC2 Instances page displaying three running instances:

Name	Instance ID	Instance state	Instance type	Status check	Alarm status	Availability
private2_BE	<code>i-0ab5ec8c076a9b452</code>	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1c
public1_FE_1	<code>i-0ba1190672c88d2f3</code>	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1a
private1_FE_2	<code>i-021085d8c3d2fb8a9</code>	Running	t2.micro	2/2 checks passed	View alarms +	us-east-1b

The instance `i-021085d8c3d2fb8a9` (private1\_FE\_2) is selected, showing its detailed configuration in the bottom panel. The details include:

Attribute	Value
Instance ID	<code>i-021085d8c3d2fb8a9</code>
IPv6 address	-
Public IPv4 address	-
Private IPv4 addresses	<code>10.0.2.214</code>
Instance state	Running
Public IPv4 DNS	-

The screenshot shows the AWS EC2 Instances page for an instance named 'i-0ab5ec8c076a9b452'. The instance is currently running. Key details include:

- Instance ID:** i-0ab5ec8c076a9b452
- Public IPv4 address:** -
- Private IPv4 addresses:** 10.0.3.122
- Public IPv4 DNS:** -
- Private IP DNS name (IPv4 only):** ip-10-0-3-122.ec2.internal
- Instance state:** Running
- Instance type:** t2.micro
- VPC ID:** vpc-06b5e911bb41334c4 (final\_vpc)
- IAM Role:** -
- Subnet ID:** subnet-0515f0e94a33fd459 (private2\_subnet)
- Instance ARN:** arn:aws:ec2:us-east-1:703671903114:instance/i-0ab5ec8c076a9b452
- AWS Compute Optimizer finding:** Opt-in to AWS Compute Optimizer for recommendations.
- Elastic IP addresses:** -
- Auto Scaling Group name:** -
- Managed:** false

## Public configuration:

```
ubuntu@ip-10-0-1-254: ~
GNU nano 7.2
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
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
server {
    listen 80 default_server;
    listen [::]:80 default_server;

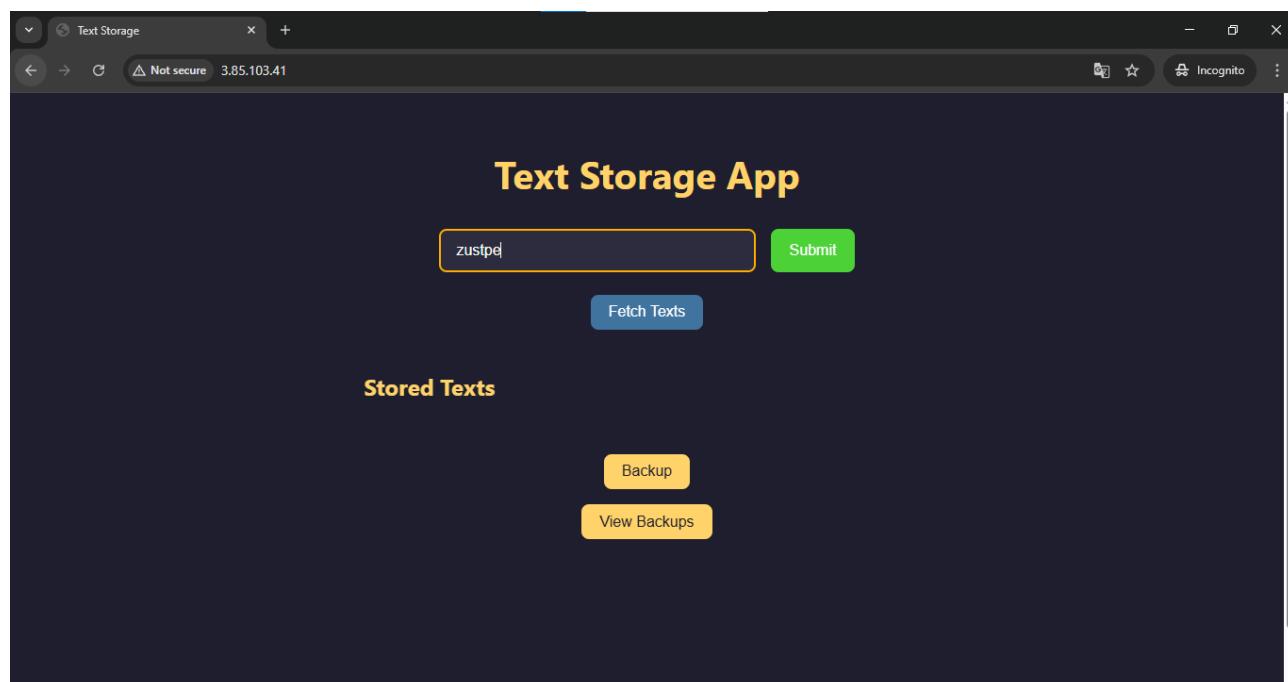
    root /var/www/html;

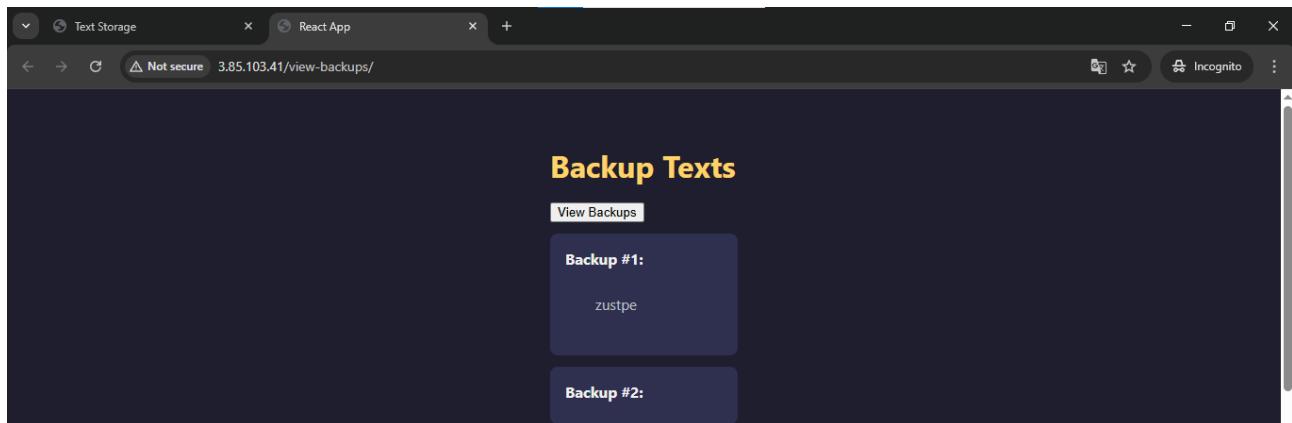
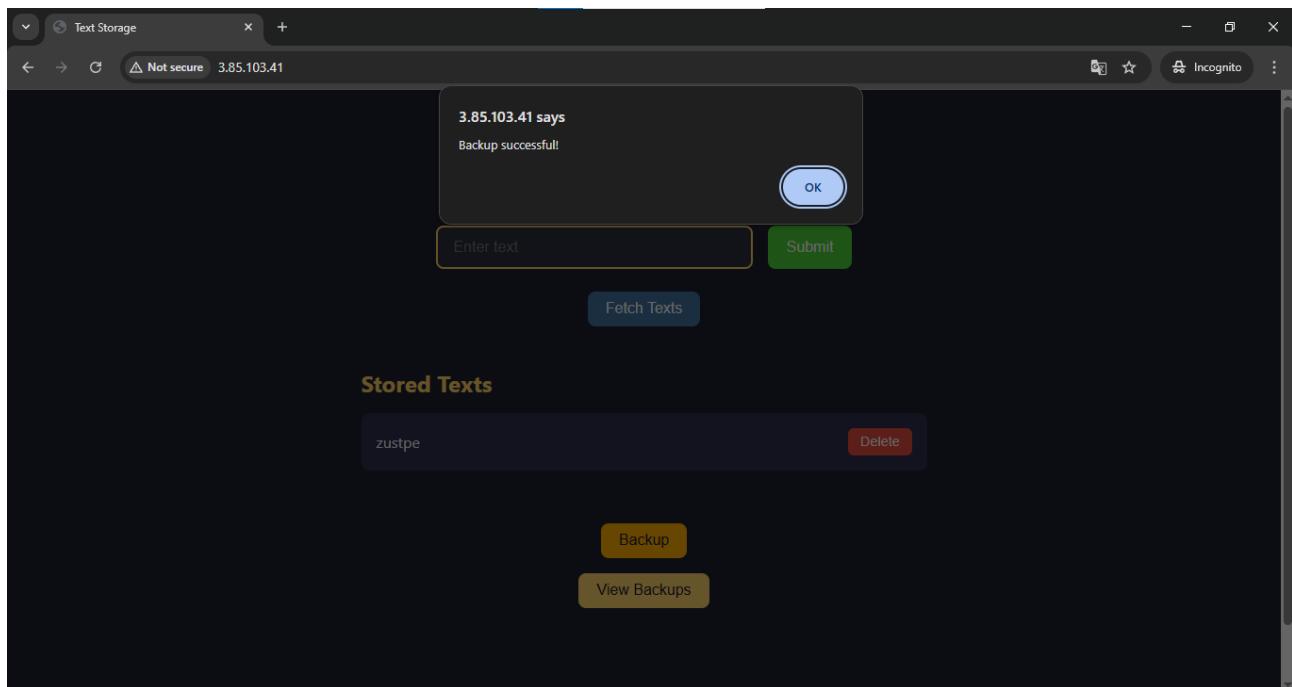
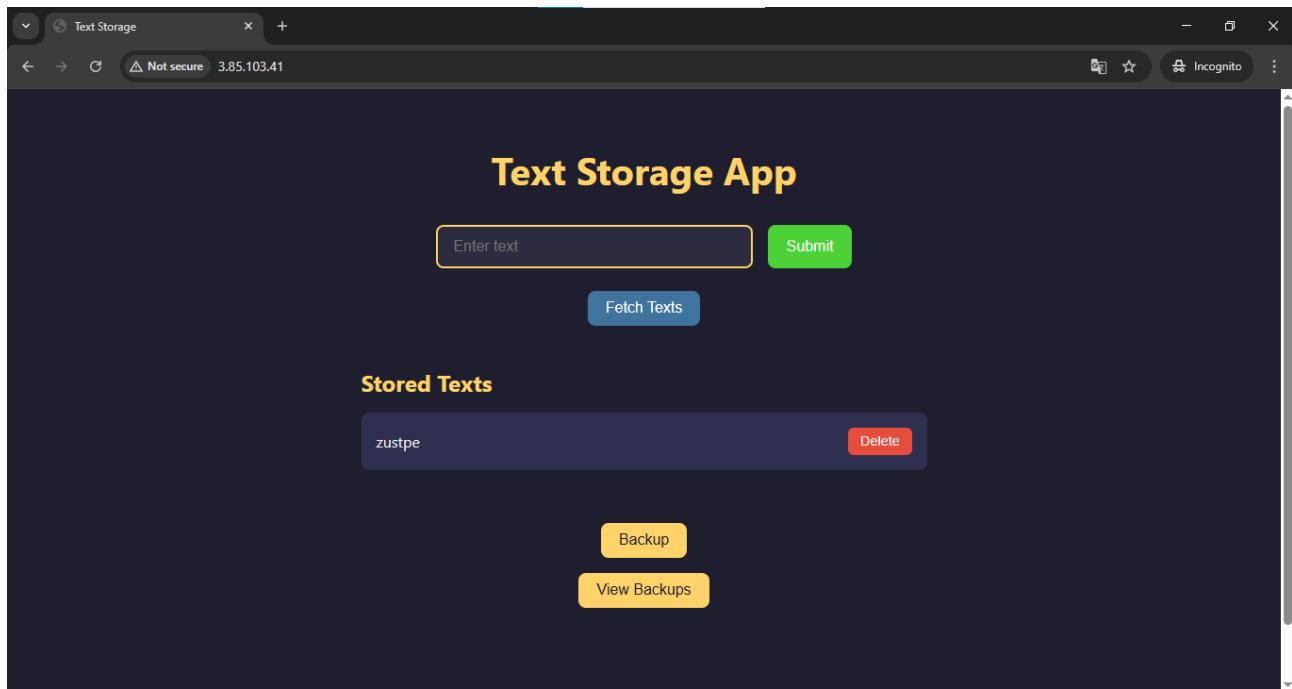
    index index.html index.htm index.nginx-debian.html;

    server_name _;

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

## Output :





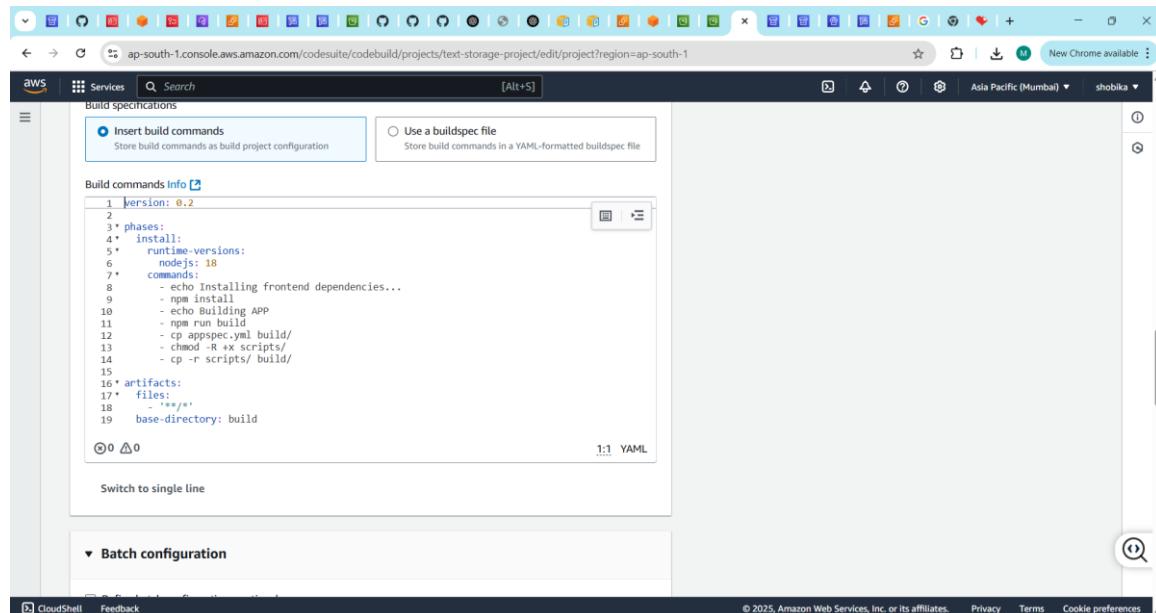
# Pipeline Stages:

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.

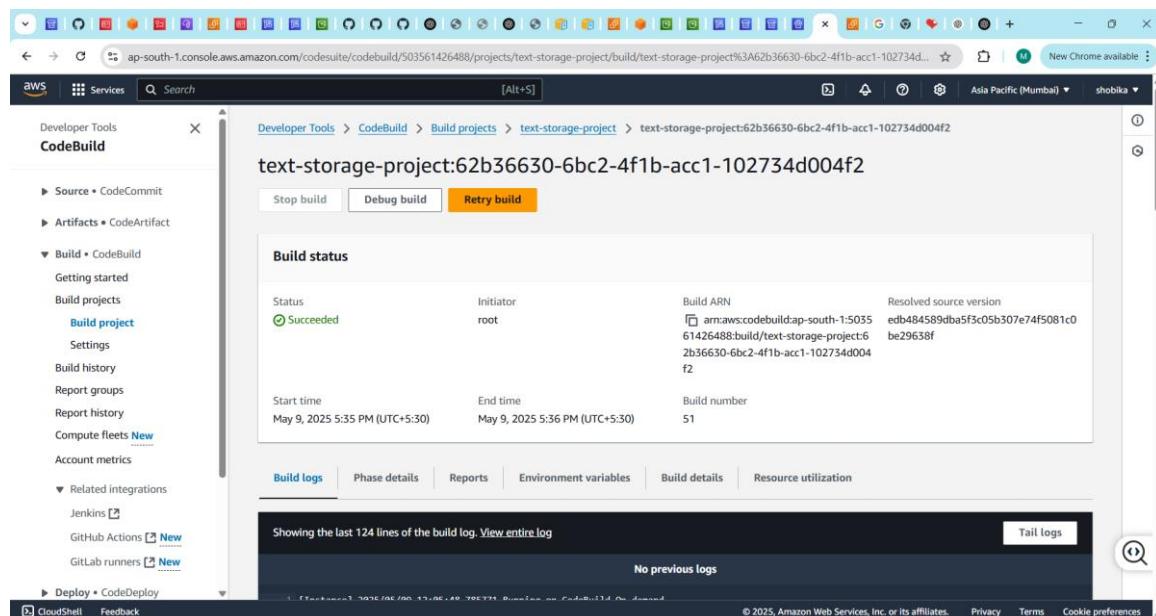
## CodeBuild



The screenshot shows the AWS CodeBuild configuration interface. It displays a buildspec file in YAML format. The file defines a build process with phases: install, runtime-versions, and commands. The commands phase includes npm install, echo, run build, cp appspec.yml build/, chmod -R +x scripts/, and cp -r scripts/ build/. The buildspec file is stored in a YAML-formatted buildspec file. The interface also includes tabs for 'Insert build commands' and 'Batch configuration'.

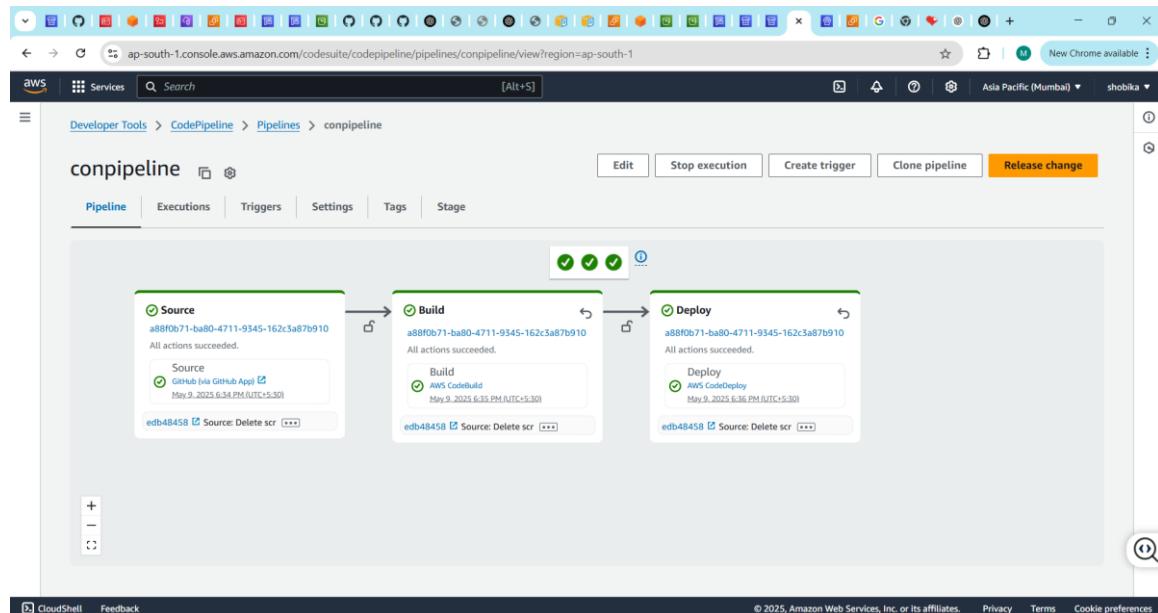
```
version: 0.2
phases:
  install:
    runtime-versions:
      nodejs: 18
    commands:
      - echo Installing frontend dependencies...
      - npm install
      - echo Building app
      - ./node_modules/.bin/rollup build
      - cp appspec.yml build/
      - chmod -R +x scripts/
      - cp -r scripts/ build/
artifacts:
  files:
    - **/*
base-directory: build
```

## Code Deploy



The screenshot shows the AWS CodeDeploy configuration interface. It displays a build status for a specific build. The build has succeeded with a status of 'Succeeded'. The initiator was 'root'. The build ARN is arn:aws:codebuild:ap-south-1:503561426488:build/text-storage-project:62b36630-6bc2-4f1b-acc1-102734d004f2. The resolved source version is edb48459dba5fc05b307e74f5f081c02b56630-6bc2-4f1b-acc1-102734d004f2. The start time was May 9, 2025 5:35 PM (UTC+5:30) and the end time was May 9, 2025 5:36 PM (UTC+5:30). The build number is 51. The interface includes tabs for 'Build logs', 'Phase details', 'Reports', 'Environment variables', 'Build details', and 'Resource utilization'.

# 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.