Streamlined Deployment of React Web Apps: A Guide to CodePipeline Implementation

Introduction:

My project focuses on automating the software development process using cloud-based Continuous Integration and Continuous Deployment (CI/CD) pipelines. By implementing CI/CD practices, I aim to enhance the speed, reliability, and efficiency of software delivery.

CI/CD Pipeline Overview:

The CI/CD pipeline acts as a series of interconnected stages that automate the steps involved in software delivery. It starts with Continuous Integration, where code changes are regularly merged into a shared repository and tested automatically. Then, Continuous Deployment takes over, automating the deployment of code changes to production or staging environments.

Tools Used:

AWS CodePipeline: Orchestrated the workflow for software delivery.

AWS CodeBuild: Utilized for compiling and packaging the code.

AWS CodeDeploy: Facilitated deployment of applications to different environments.

AWS EC2: Deployment of applications

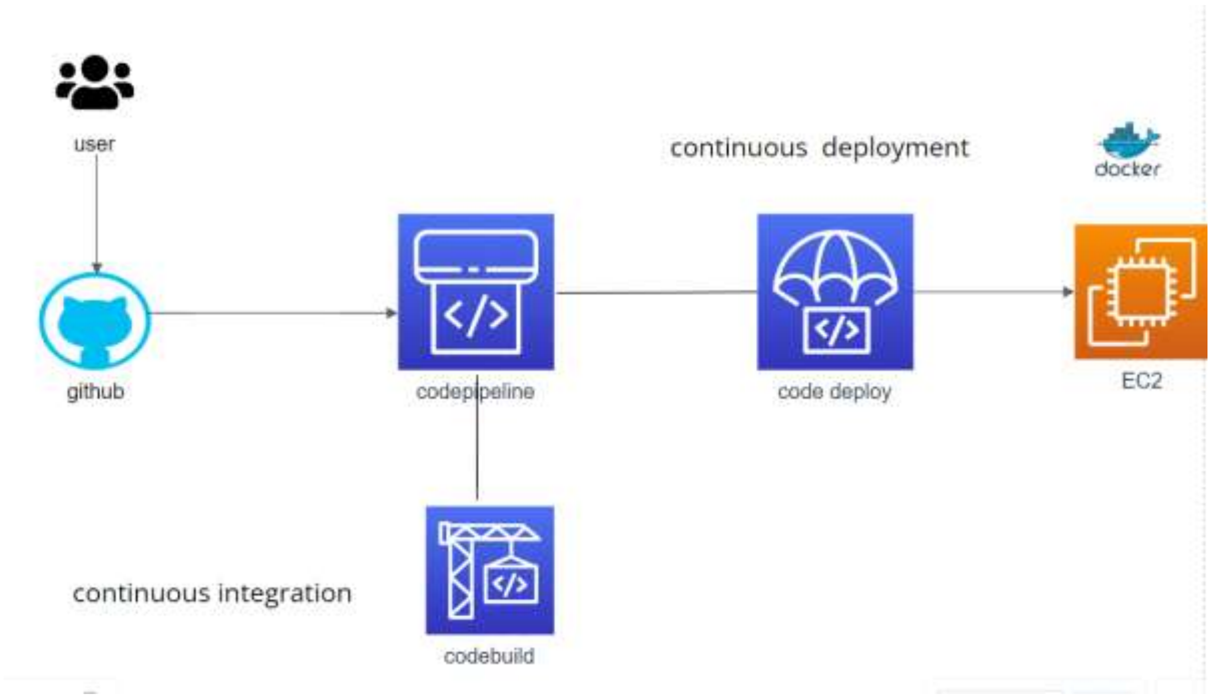
Amazon CloudWatch Logs: Monitored and provided insights into system and application logs.

Amazon S3 (Simple Storage Service): Stored artifacts generated during the CI/CD process.

Docker: Utilized for containerization of the application, ensuring consistency across different environments.

GitHub: Source code repository used for version control and collaboration..

Project Setup:



Setting up the CI/CD pipeline involved configuring CodePipeline to monitor my source code repository for changes. I defined the sequence of actions to be executed at each stage of the pipeline, ensuring a smooth flow from code commit to deployment.

Pipeline Workflow:

The workflow of my CI/CD pipeline encompasses stages such as source code retrieval, automated testing, artifact generation, and deployment. Each stage is designed to validate code changes and ensure they meet quality standards before being deployed.

Source Stage:

During the source stage, CodePipeline retrieves the latest version of the code from my chosen repository, whether it's GitHub, AWS CodeCommit, or another source control system. This ensures that the pipeline operates on the most recent codebase. I had used GitHub for source stage.

Repository Structure and Included Files:

The project repository consists of two main folders: backend and frontend, housing the backend and frontend components, respectively. Each of these folders contains its respective Dockerfile for containerization.

Additionally, the repository includes the following key files:

appspec.yml:

Description: This file is used by AWS CodeDeploy to define the deployment process for the application.

Purpose: It specifies the deployment actions to be performed during the deployment lifecycle hooks, facilitating the smooth deployment of the application onto the target environment.

start_container.sh:

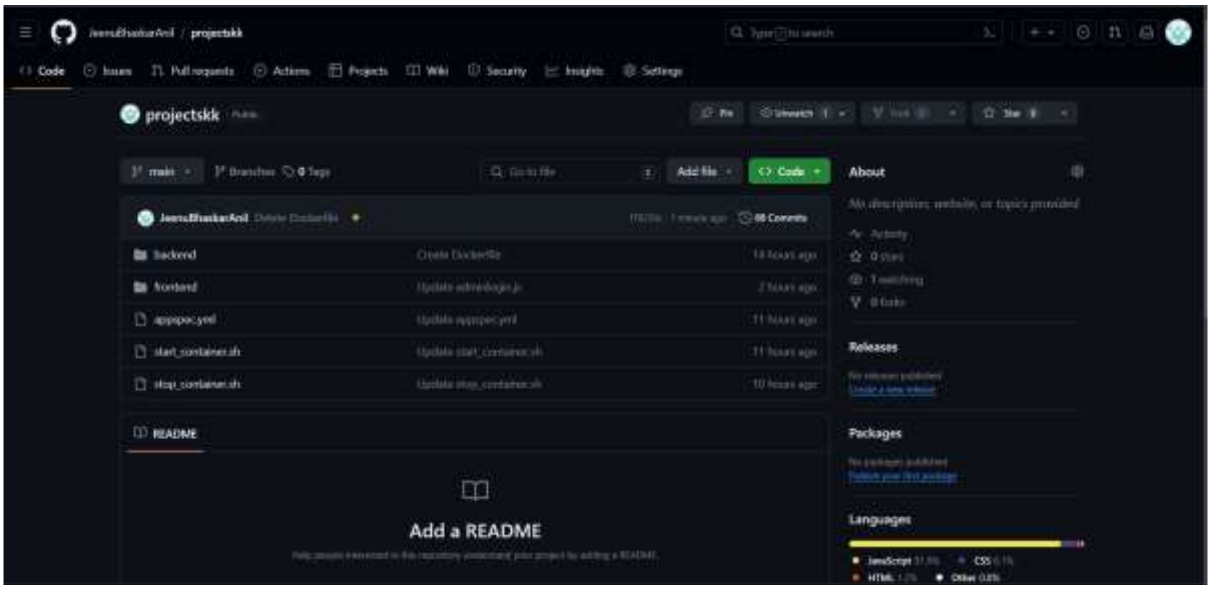
Description: This script is responsible for starting the Docker container hosting the application.

Purpose: It initiates the execution of the Docker container, ensuring that the application is up and running within the containerized environment.

stop_container.sh:

Description: This script is used to stop the running Docker container.

Purpose: It gracefully terminates the Docker container, allowing for proper cleanup and shutdown procedures.



JeenuBhaskarAnil Delete Dockerfile · 1 minute ago · 88 Commits

backend	Create Dockerfile	14 hours ago
frontend	Update adminlogin.js	2 hours ago
appspec.yml	Update appspec.yml	11 hours ago
start_container.sh	Update start_container.sh	11 hours ago
stop_container.sh	Update stop_container.sh	10 hours ago

No description, website, or topics provided

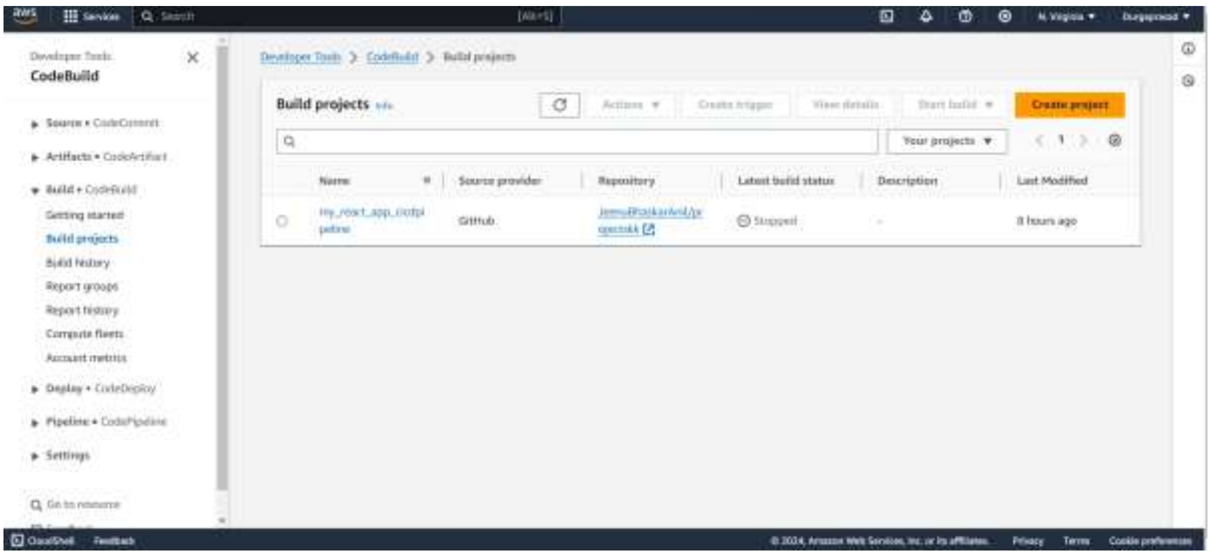
Build Stage:

In the build stage, AWS CodeBuild compiles the source code, runs unit tests, and creates a deployable artifact. This phase is crucial for detecting any build errors or inconsistencies in the codebase before proceeding to deployment.

AWS CodeBuild Project Creation:

Navigate to the AWS Management Console and access the CodeBuild service.

Click on "Create project" to begin creating a new CodeBuild project.



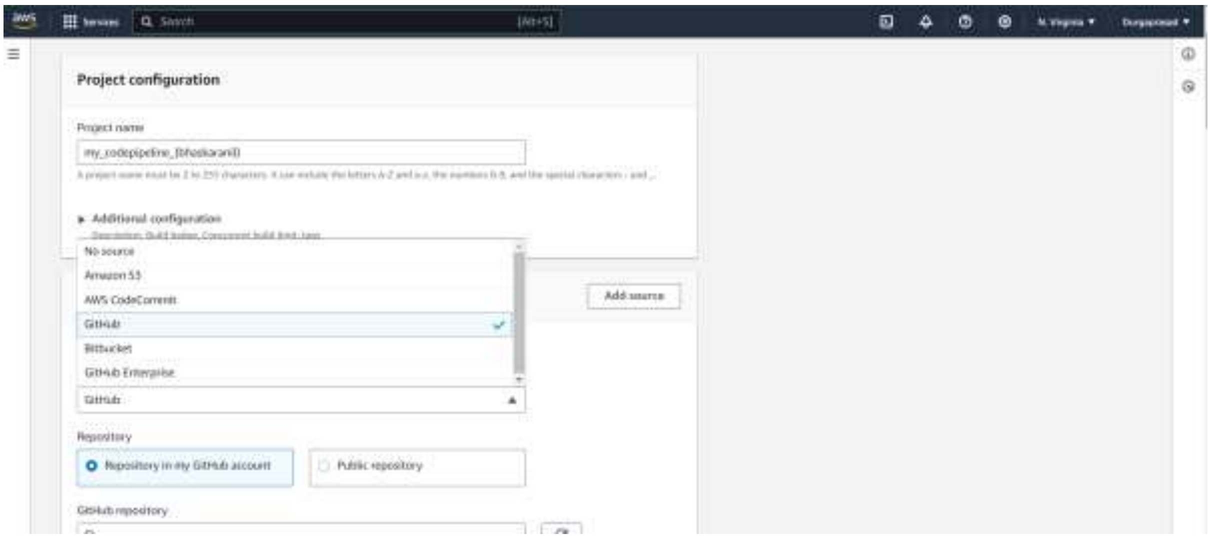
Name	Source provider	Repository	Latest build status	Description	Last Modified
my_react_app_codepipeline	GitHub	[illegible]	Stopped	-	8 hours ago

Specify a unique name for the project and provide a description, if necessary.

Source Configuration:

Select the source provider (in this case, GitHub) and provide the repository URL.

Choose the branch to monitor for changes.



Project name: my_codepipeline_(bhaskaranil)

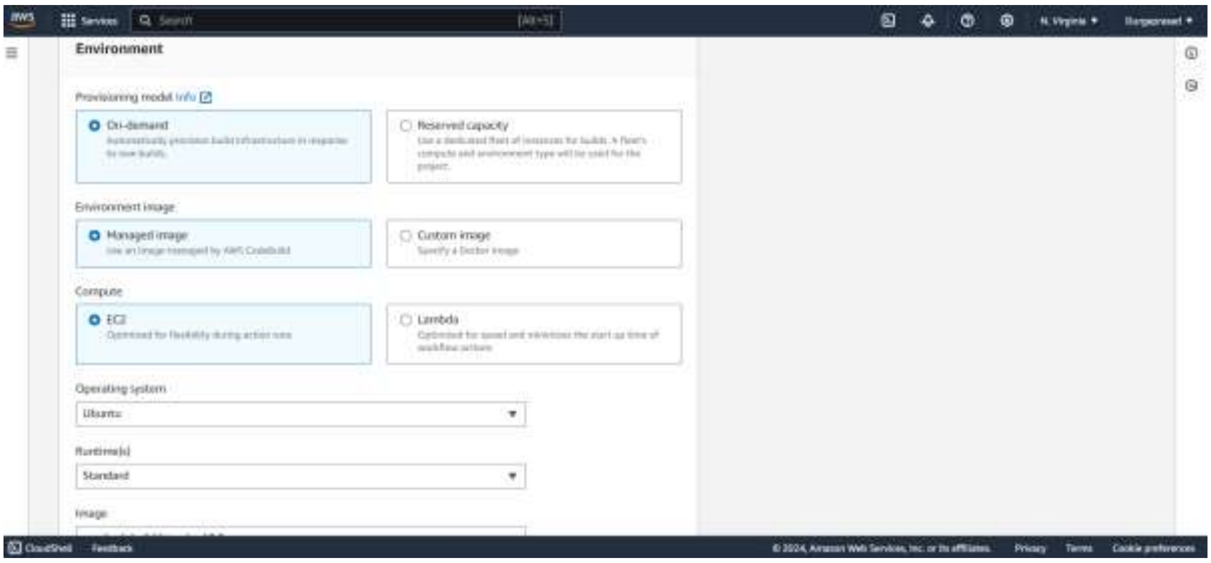
Source provider: GitHub

Repository: [✓ Repository in my GitHub account] Public repository

GitHub repository

Environment Configuration:

Select the operating system and runtime environment suitable for your project (e.g., Ubuntu, Node.js).



Environment

Provisioning model Info

- [✓ On-demand] Automatically provision build infrastructure in response to new builds.
- Reserved capacity Use a dedicated fleet of instances for builds. A fleet's compute and environment type will be used for the project.

Environment image

- [✓ Managed image] Use an image managed by AWS CodeBuild
- Custom image Specify a Docker image

Compute

- [✓ EC2] Optimized for flexibility during action runs
- Lambda Optimized for speed and minimizes the start up time of workflow actions

Operating system: Ubuntu

Runtime(s): Standard

Image

Specify the runtime version and image settings.

Configure additional environment variables, if required.

Build Specifications:

Define the build specification by selecting the buildspec.yml file located in the repository.

Alternatively, define build commands directly within the CodeBuild project configuration.

Here is my buildspec.yml file:

```
version: 0.2
env:
  parameter-store:
    DOCKER_REGISTRY_USERNAME: /myapp/docker-credentials/username
    DOCKER_REGISTRY_PASSWORD: /myapp/docker-credentials/password
    DOCKER_REGISTRY_URL: /myapp/docker-registry/url

phases:
  install:
    runtime-versions:
      nodejs: 20
  pre_build:
    commands:
      - echo "Installing dependencies for the backend..."
      - cd backend && npm install --silent
      - echo "Installing dependencies for the frontend..."
      - cd ../frontend && npm install --silent

  build:
    commands:
      - echo "Building the backend..."
      - echo "$DOCKER_REGISTRY_PASSWORD" | docker login -u "$DOCKER_REGISTRY_USERNAME" --password-
stdin "$DOCKER_REGISTRY_URL"
      - cd ../backend && docker build -t
"$DOCKER_REGISTRY_URL/$DOCKER_REGISTRY_USERNAME/backend:latest" .
      - docker push "$DOCKER_REGISTRY_URL/$DOCKER_REGISTRY_USERNAME/backend:latest"
      - echo "Building the frontend..."
      - cd ../frontend && docker build -t
"$DOCKER_REGISTRY_URL/$DOCKER_REGISTRY_USERNAME/frontend:latest" .
      - echo "Pushing Docker images..."
      - docker push "$DOCKER_REGISTRY_URL/$DOCKER_REGISTRY_USERNAME/frontend:latest"
post_build:
  commands:
    - echo "Pushing Docker images..."
    - echo "Build completed successfully!"
```

After configuring our build process, I ensured secure handling of sensitive information like Docker registry credentials

Artifact Configuration:

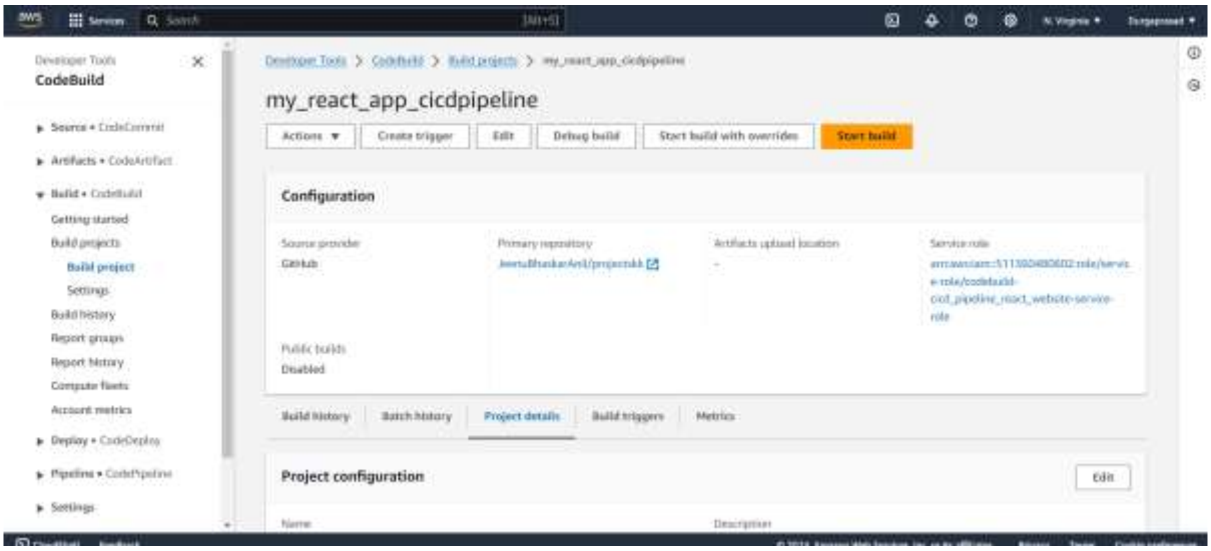
Specify the location where build artifacts should be stored upon successful completion of the build process.

Logs Configuration:

Configure the settings for CloudWatch Logs to capture build logs for troubleshooting and analysis purposes.

Start Build:

Once all configurations are in place, initiate the build process by clicking on "Start build" or wait for automatic triggering based on source code changes.



Developer Tools > CodeBuild > Build projects > my_react_app_cicdpipeline

my_react_app_cicdpipeline

Configuration

Source provider: GitHub

Primary repository: JeenaBhaskarAnil/projectkk

Artifacts upload location: -

Service role: arn:aws:iam::511392480602:role/service-role/codebuild-cicd_pipeline_react_website-service-role

Public builds: Disabled

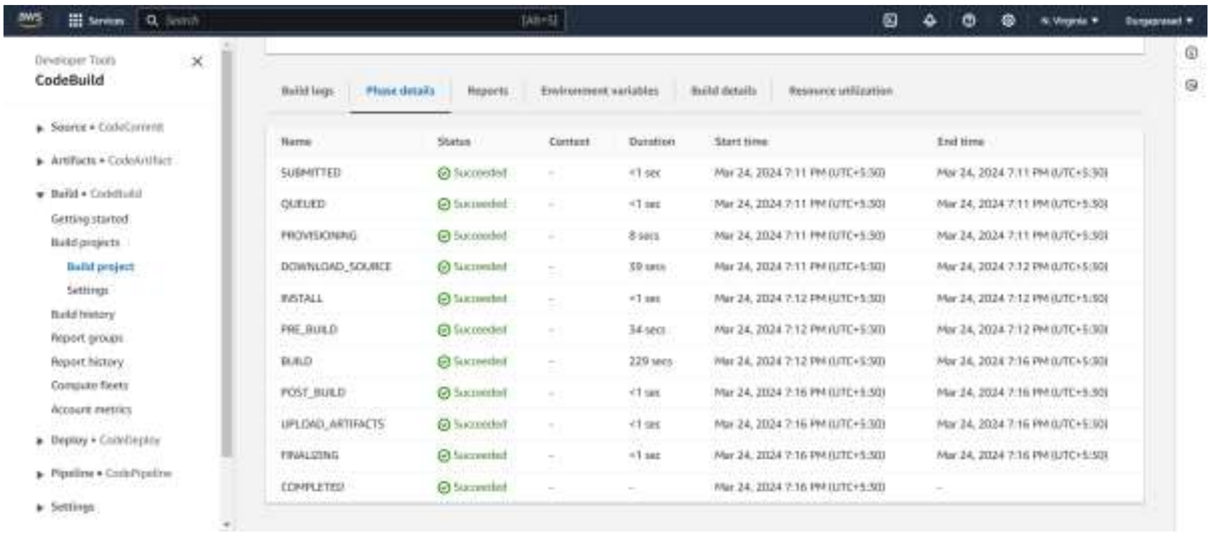
Project configuration

Name

Description

Monitor Build Progress:

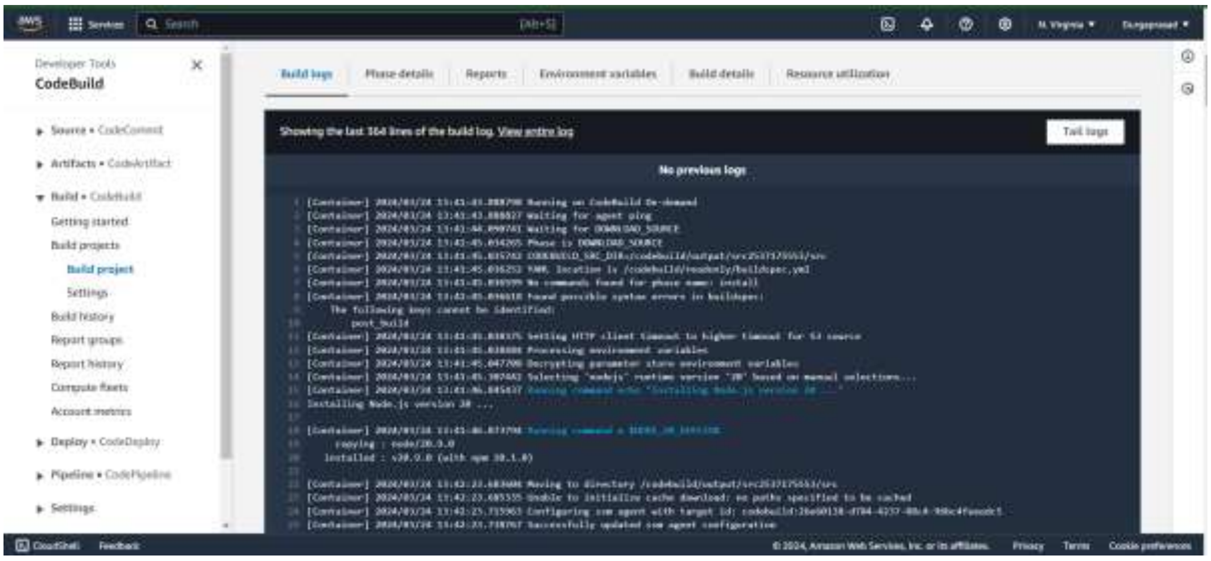
Monitor the build progress through the CodeBuild console, observing each phase of the build process including initialization, pre-build, build, post-build, and finalization.



Name	Status	Context	Duration	Start time	End time
SUBMITTED	✓ Succeeded	-	<1 sec	Mar 24, 2024 7:11 PM (UTC+5:30)	Mar 24, 2024 7:11 PM (UTC+5:30)
QUEUED	✓ Succeeded	-	<1 sec	Mar 24, 2024 7:11 PM (UTC+5:30)	Mar 24, 2024 7:11 PM (UTC+5:30)
PROVISIONING	✓ Succeeded	-	8 secs	Mar 24, 2024 7:11 PM (UTC+5:30)	Mar 24, 2024 7:11 PM (UTC+5:30)
DOWNLOAD_SOURCE	✓ Succeeded	-	59 secs	Mar 24, 2024 7:11 PM (UTC+5:30)	Mar 24, 2024 7:12 PM (UTC+5:30)
INSTALL	✓ Succeeded	-	<1 sec	Mar 24, 2024 7:12 PM (UTC+5:30)	Mar 24, 2024 7:12 PM (UTC+5:30)
PRE_BUILD	✓ Succeeded	-	34 secs	Mar 24, 2024 7:12 PM (UTC+5:30)	Mar 24, 2024 7:12 PM (UTC+5:30)
BUILD	✓ Succeeded	-	229 secs	Mar 24, 2024 7:12 PM (UTC+5:30)	Mar 24, 2024 7:16 PM (UTC+5:30)
POST_BUILD	✓ Succeeded	-	<1 sec	Mar 24, 2024 7:16 PM (UTC+5:30)	Mar 24, 2024 7:16 PM (UTC+5:30)
UPLOAD_ARTIFACTS	✓ Succeeded	-	<1 sec	Mar 24, 2024 7:16 PM (UTC+5:30)	Mar 24, 2024 7:16 PM (UTC+5:30)
FINALIZING	✓ Succeeded	-	<1 sec	Mar 24, 2024 7:16 PM (UTC+5:30)	Mar 24, 2024 7:16 PM (UTC+5:30)
COMPLETED	✓ Succeeded	-	-	Mar 24, 2024 7:16 PM (UTC+5:30)	-

View Build Logs:

Access build logs and artifacts generated during the build process through CloudWatch Logs and the specified artifact location.


Showing the last 364 lines of the build log. View entire log

No previous logs

```
[Container] 2024/03/24 13:41:41.888798 Running on CodeBuild On-demand
[Container] 2024/03/24 13:41:43.888827 Waiting for agent ping
[Container] 2024/03/24 13:41:44.090741 Waiting for DOWNLOAD_SOURCE
[Container] 2024/03/24 13:41:45.034265 Phase is DOWNLOAD_SOURCE
[Container] 2024/03/24 13:41:45.035762 CODEBUILD_SRC_DIR=/codebuild/output/src2537175553/src
[Container] 2024/03/24 13:41:45.036253 YAML location is /codebuild/readonly/buildspec.yml
[Container] 2024/03/24 13:41:45.036599 No commands found for phase name: install
[Container] 2024/03/24 13:41:45.036618 Found possible syntax errors in buildspec:
    The following keys cannot be identified:
        post_build
[Container] 2024/03/24 13:41:45.038375 Setting HTTP client timeout to higher timeout for S3 source
[Container] 2024/03/24 13:41:45.038808 Processing environment variables
[Container] 2024/03/24 13:41:45.047708 Decrypting parameter store environment variables
[Container] 2024/03/24 13:41:45.307442 Selecting 'nodejs' runtime version '20' based on manual selections...
[Container] 2024/03/24 13:41:46.045437 Running command echo "Installing Node.js version 20 ..."
Installing Node.js version 20 ...

[Container] 2024/03/24 13:41:46.073798 Running command n $NODE_20_VERSION
     copying : node/20.9.0
   installed : v20.9.0 (with npm 10.1.0)

[Container] 2024/03/24 13:42:23.683606 Moving to directory /codebuild/output/src2537175553/src
[Container] 2024/03/24 13:42:23.685335 Unable to initialize cache download: no paths specified to be cached
[Container] 2024/03/24 13:42:23.715963 Configuring ssm agent with target id: codebuild:26e60138-d784-4237-88c4-98bc4faeedc5
[Container] 2024/03/24 13:42:23.718767 Successfully updated ssm agent configuration
```

By successful completion of the CodeBuild process, Docker images are automatically created and pushed to Docker Hub.

Artifact Storage:

AWS S3 serves as the storage solution for artifacts generated during the build process. Storing artifacts in S3 provides durability, scalability, and easy accessibility, allowing for seamless integration with the deployment process.

Deployment Stage:

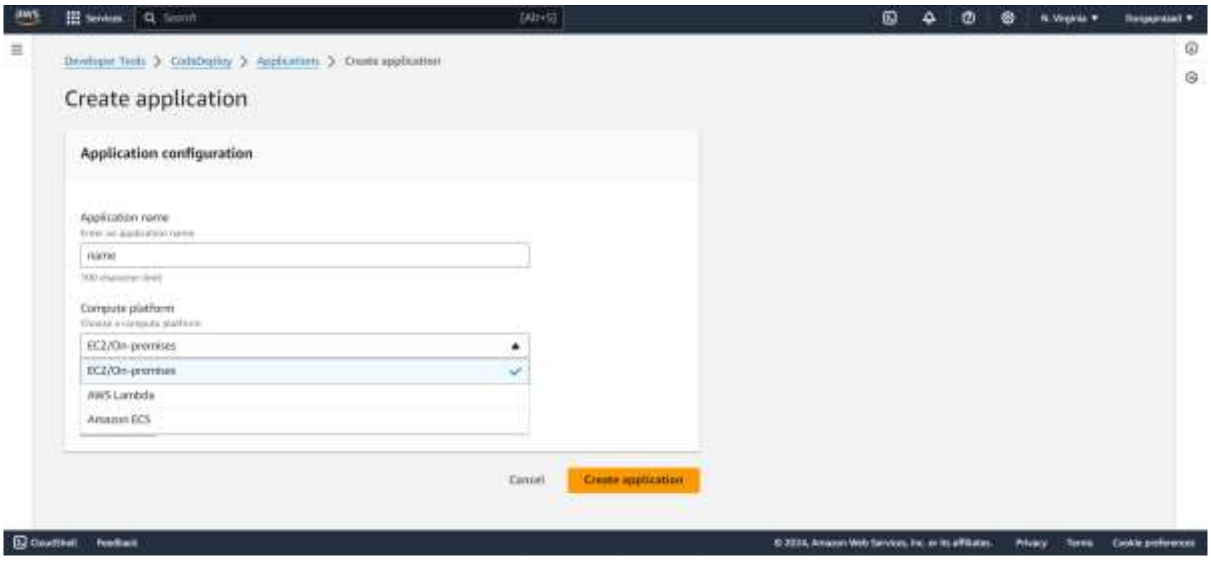
Create Deployment Configuration:

Log in to the AWS Management Console and navigate to the CodeDeploy service.

Click on "Create Deployment Configuration".

Specify a name for the configuration and define the deployment settings, such as the deployment type (in-place or blue/green), deployment strategy, and any optional settings.

Click "Create deployment configuration" to save the settings.


Developer Tools > CodeDeploy > Applications > Create application

Create application

Application configuration

Application name
Enter an application name
name
100 character limit

Compute platform
Choose a compute platform
EC2/On-premises
EC2/On-premises ✓
AWS Lambda
Amazon ECS

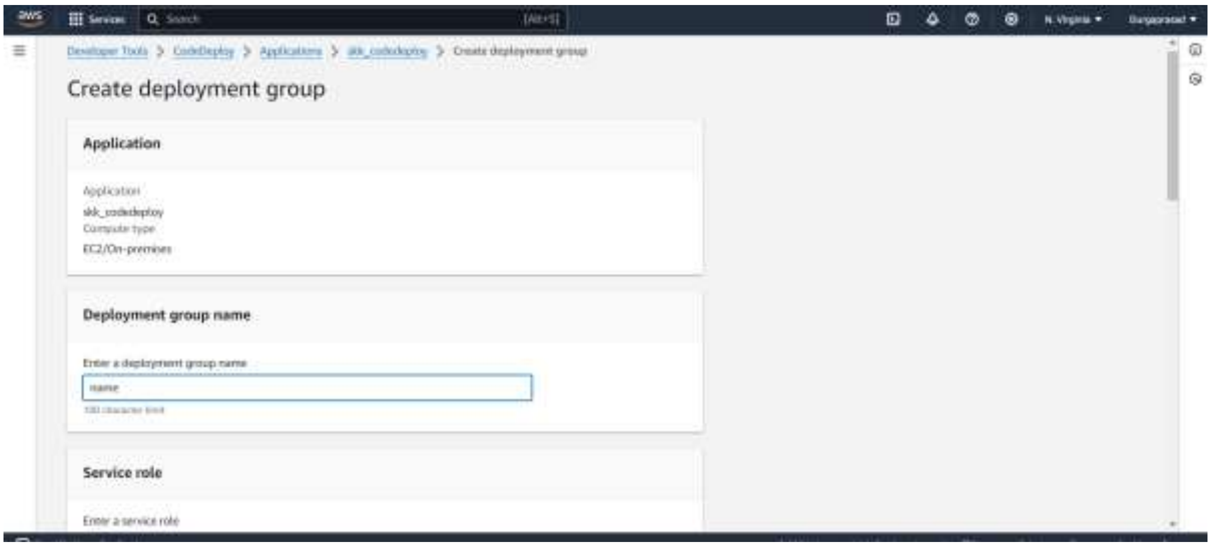
Cancel | Create application

Configure Deployment Group:

In the CodeDeploy dashboard, select the application for which you want to create a deployment group.

Click on "Create deployment group".

Provide a name for the deployment group and select the deployment configuration created in the previous step.


Developer Tools > CodeDeploy > Applications > skk_codedeploy > Create deployment group

Create deployment group

Application

Application
skk_codedeploy
Compute type
EC2/On-premises

Deployment group name

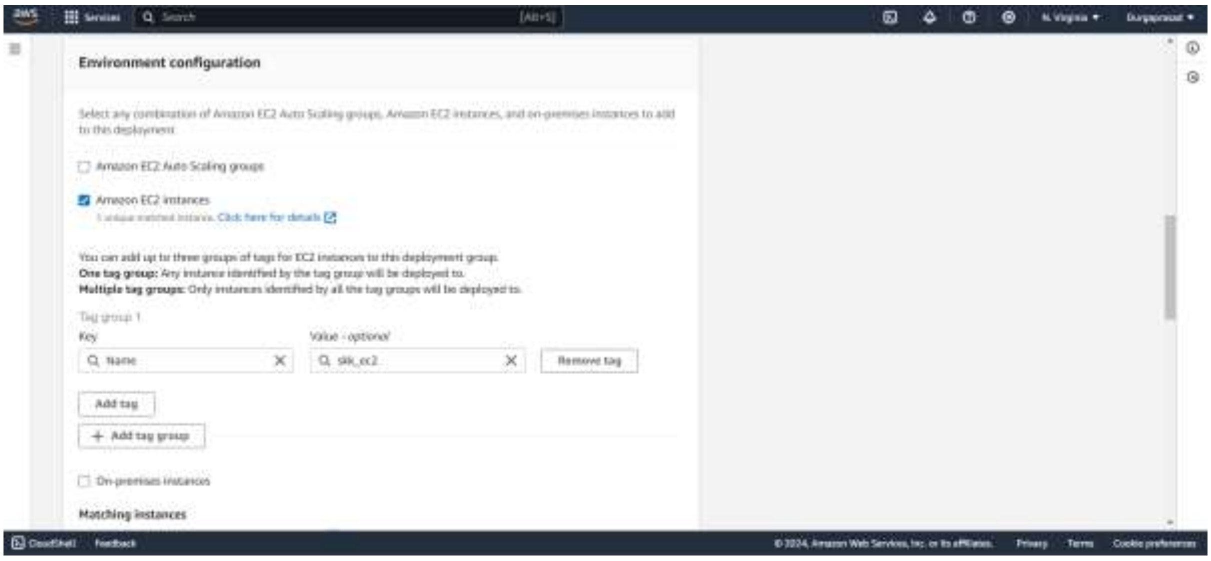
Enter a deployment group name
name
100 character limit

Service role

Enter a service role

Specify the deployment type (EC2/On-Premises instances, AWS Lambda, or ECS), compute platform, and any required tags to identify target instances.

Download the codedeploy-agent in ec2 instance by using aws documentation in EC2 instance.


Environment configuration

Select any combination of Amazon EC2 Auto Scaling groups, Amazon EC2 instances, and on-premises instances to add to this deployment

- [ ] Amazon EC2 Auto Scaling groups
- [x] Amazon EC2 instances
  1 unique matched instance. Click here for details

You can add up to three groups of tags for EC2 instances to this deployment group.
One tag group: Any instance identified by the tag group will be deployed to.
Multiple tag groups: Only instances identified by all the tag groups will be deployed to.

Tag group 1

Key	Value - optional
Name	sik_ec2

- [ ] On-premises instances

Matching instances

Click "Create deployment group" to complete the configuration.

Create AppSpec File:

Craft an AppSpec file named appspec.yml that outlines the deployment process. This file specifies how CodeDeploy should deploy the application onto the target instances.

Include instructions for tasks such as stopping, deploying, and starting services, executing scripts, and managing application files.

Ensure that the AppSpec file is structured according to the requirements of your application and deployment strategy.

Here is my appspec.yml file : version: 0.0

```
os: linux

hooks:
  ApplicationStop:
    - location: stop_container.sh
      timeout: 300
      runas: root
  AfterInstall:
    - location: start_container.sh
      timeout: 300
      runas: root
```

Place this appspec.yml file at root directory of Github repositery

Create Deployment:

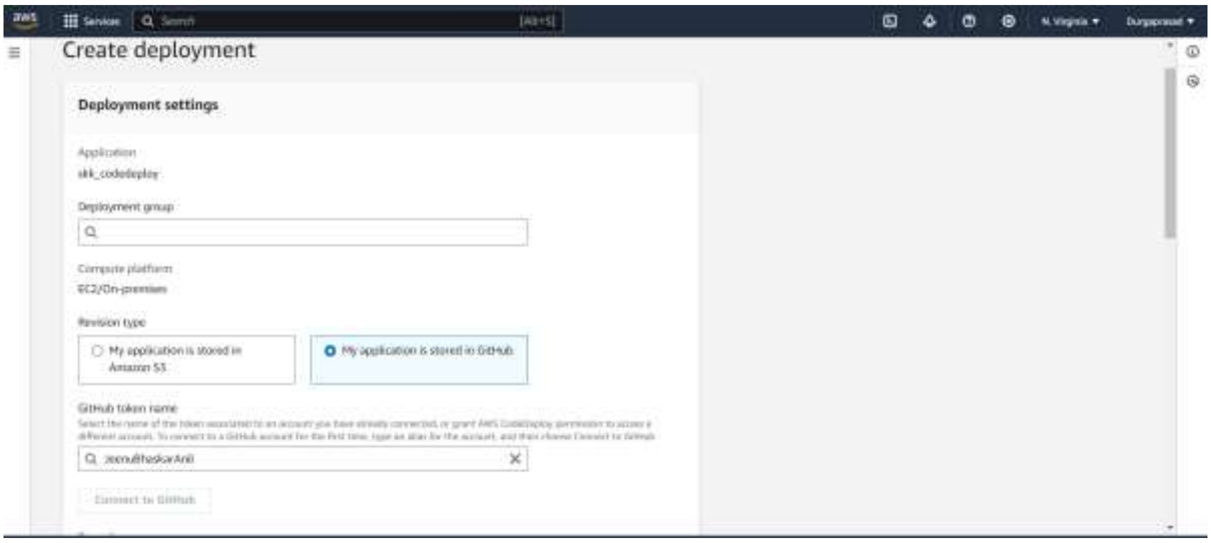
After setting up the deployment group, navigate to your CodeDeploy dashboard.

Select the application for which you configured the deployment group.

Click on the "Deploy New Revision" button.

Specify the deployment group you want to deploy to.

Choose the revision source, which could be Amazon S3, GitHub, or another source.



Create deployment

Deployment settings

Application
skk_codedeploy

Deployment group

Compute platform
EC2/On-premises

Revision type
My application is stored in Amazon S3 [✓ My application is stored in GitHub]

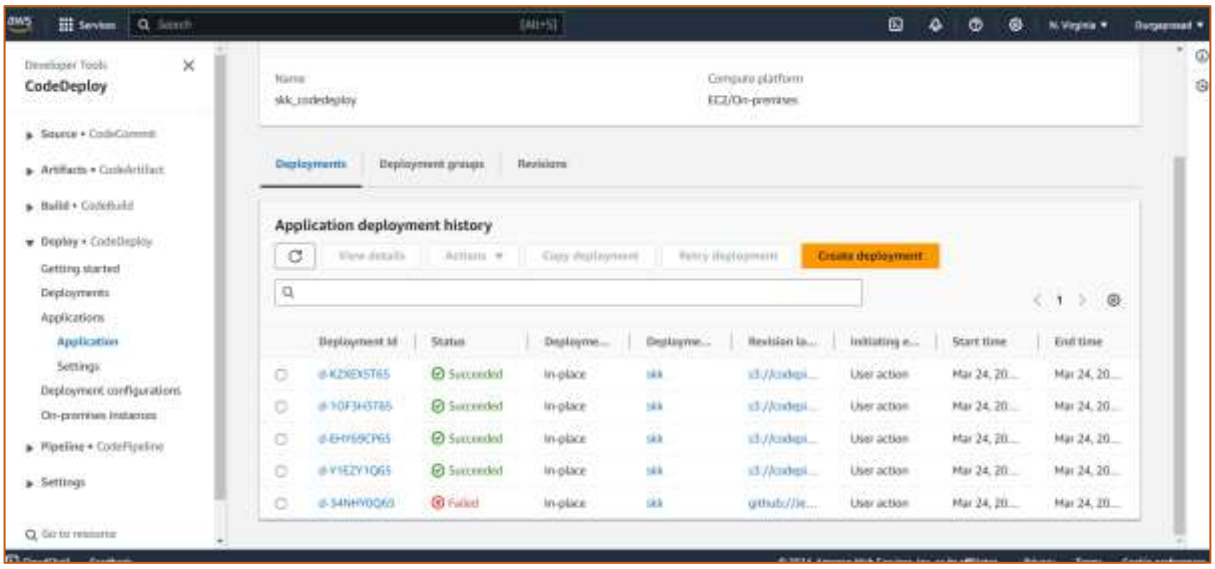
GitHub token name
[illegible]
zeenuBhaskarAnil

Connect to GitHub

Optionally, specify any deployment settings such as deployment configuration, deployment description, and additional options.

Click "Deploy" to initiate the deployment process.

Monitor the deployment progress on the CodeDeploy dashboard and review any deployment logs or errors encountered during the deployment process.



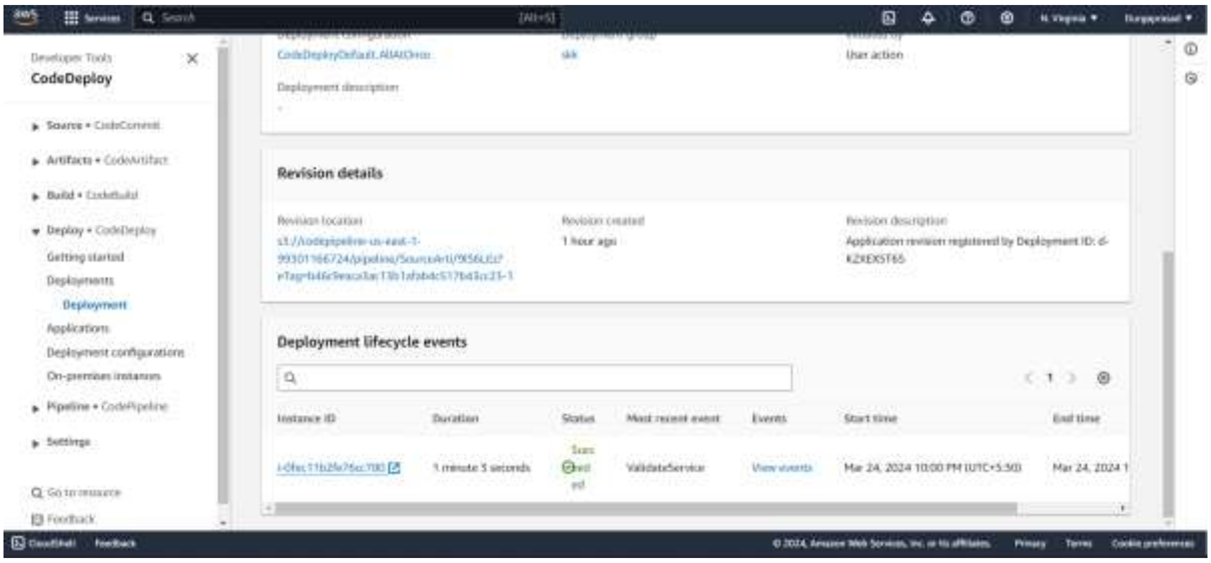
Name
skk_codedeploy

Compute platform
EC2/On-premises

Application deployment history

Deployment Id	Status	Deployme...	Deployme...	Revision lo...	Initiating e...	Start time	End time
d-K2XEX5T65	Succeeded	In-place	skk	s3://codepi...	User action	Mar 24, 20...	Mar 24, 20...
d-10F3H5T85	Succeeded	In-place	skk	s3://codepi...	User action	Mar 24, 20...	Mar 24, 20...
d-EHY69CP65	Succeeded	In-place	skk	s3://codepi...	User action	Mar 24, 20...	Mar 24, 20...
d-Y1EZY1Q65	Succeeded	In-place	skk	s3://codepi...	User action	Mar 24, 20...	Mar 24, 20...
d-34NHY0Q65	Failed	In-place	skk	github://je...	User action	Mar 24, 20...	Mar 24, 20...

Click on the deployment id and click on view events :



CodeDeployDefault.AllAtOnce | skk | User action

Deployment description

-

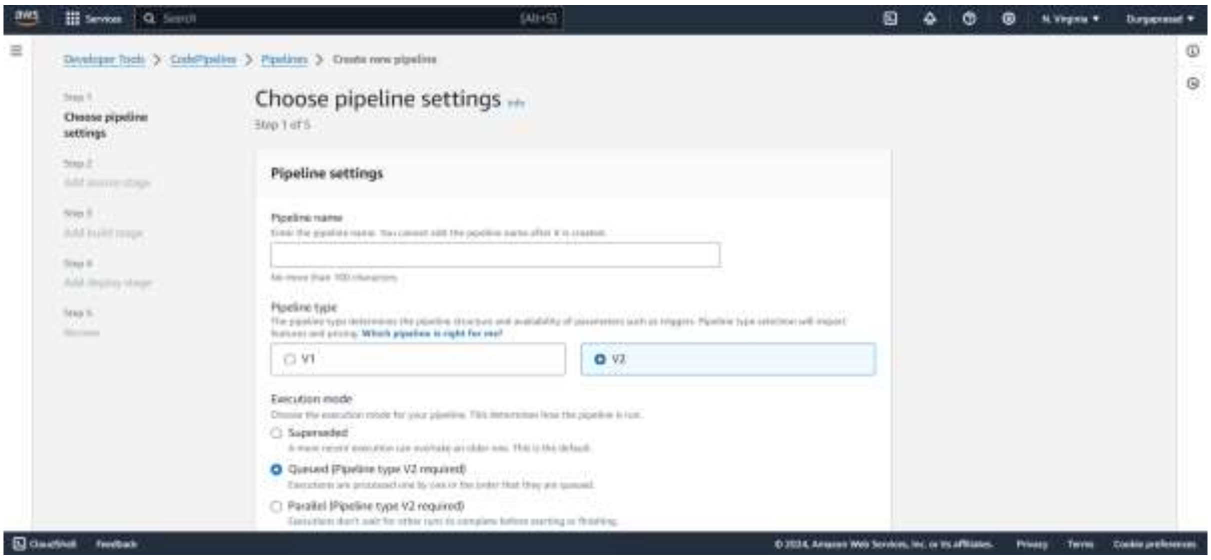
Revision details

Revision location	Revision created	Revision description
s3://codepipeline-us-east-1-993011166724/pipeline/SourceArti/9556LEc?eTag=b46c9eaca3ac13b1afabdc517bd3cc23-1	1 hour ago	Application revision registered by Deployment ID: d-KZXEXST65

Deployment lifecycle events

Instance ID	Duration	Status	Most recent event	Events	Start time	End time
i-0fec11b2fe76cc780	1 minute 3 seconds	Succeeded	ValidateService	View events	Mar 24, 2024 10:00 PM (UTC+5:30)	Mar 24, 2024 1

Integrate with CodePipeline:



Choose pipeline settings Info

Step 1 of 5

Pipeline settings

Pipeline name

Enter the pipeline name. You cannot edit the pipeline name after it is created.

No more than 100 characters

Pipeline type

The pipeline type determines the pipeline structure and availability of parameters such as triggers. Pipeline type selection will impact features and pricing. Which pipeline is right for me?

V1 [✓ V2]

Execution mode

Choose the execution mode for your pipeline. This determines how the pipeline is run.

Superseded

A more recent execution can overtake an older one. This is the default.

[✓ Queued (Pipeline type V2 required)]

Executions are processed one by one in the order that they are queued.

Parallel (Pipeline type V2 required)

Executions don't wait for other runs to complete before starting or finishing.

Source Stage:

In the source stage of your CodePipeline, you specify the source from which your application code will be retrieved. This could be a source code repository like GitHub or AWS CodeCommit.

Configure the source stage to pull the latest changes from your repository whenever a new commit is made to the specified branch.

Once the source code is retrieved, it is passed on as an artifact to the subsequent stages of the pipeline for further processing.

Build Stage:

In the build stage, you define the actions required to build your application code. This typically involves compiling, testing, and packaging your code into deployable artifacts.

Configure a CodeBuild project as a build action within your pipeline. This project will execute the build commands defined in your buildspec.yml file.

The build process generates artifacts, such as Docker images or compiled binaries, which are then uploaded to an artifact store like Amazon S3.

Deploy Stage:

In the deploy stage, you define the actions required to deploy your application to your target environment.

Configure a CodeDeploy deployment action within your pipeline. Specify the deployment group created in CodeDeploy, which identifies the target instances or services for deployment.

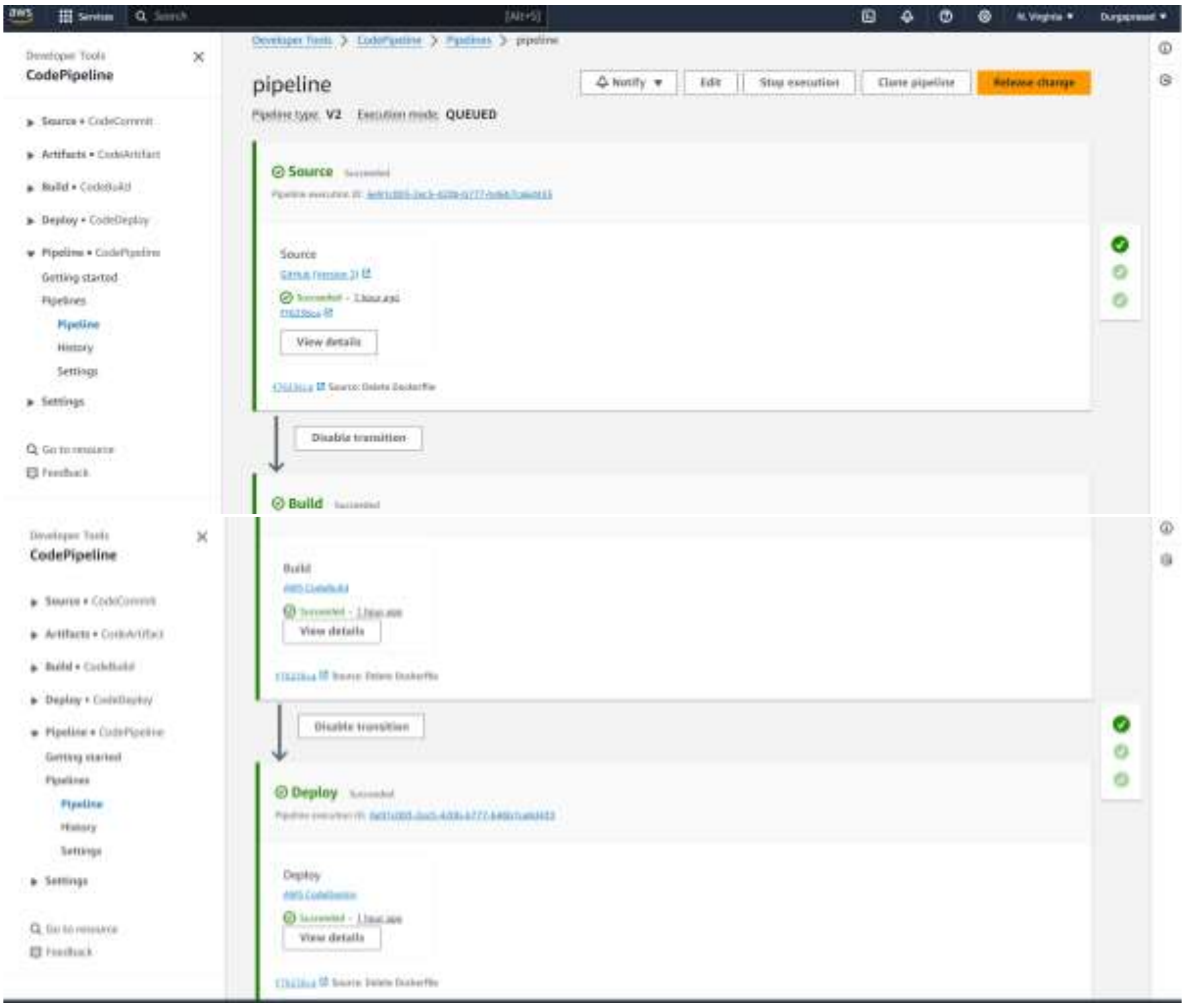
Choose the input artifacts generated in the build stage as the deployment source for CodeDeploy. These artifacts contain the packaged application code ready for deployment.

Optionally, configure additional settings such as deployment configuration, deployment description, and rollback behavior as needed for your deployment.

Once configured, the deploy action triggers CodeDeploy to initiate the deployment process, which deploys the application to the specified deployment group.Trigger Deployment:

Once testing is complete and you're confident in the deployment process, trigger the deployment either manually or automatically through the CI/CD pipeline.

Monitor the deployment progress and verify that the application is successfully deployed to the target instances.



Monitor Deployment Health:

Monitor the health and status of the deployed application using AWS CodeDeploy.

Utilize built-in monitoring features to track deployment progress, detect any errors or failures, and ensure the application's availability and functionality post-deployment.

The deployment stage, facilitated by AWS CodeDeploy, automates the rollout of new code changes to the target environment. Whether it's updating a web application, a serverless function, or a containerized service, CodeDeploy ensures a smooth and reliable deployment process.

Monitoring and Alerts:

To monitor the health and performance of my CI/CD pipeline, I implemented AWS CloudWatch alarms and metrics. These monitoring tools help me proactively detect issues, such as failed builds or deployment errors, and take corrective actions promptly.

Troubleshooting:

Throughout the project, I encountered various challenges, such as configuration errors and environment setup complexities. To overcome these challenges, I relied on troubleshooting techniques like analyzing logs, consulting documentation, and seeking assistance from online communities.

Conclusion:

In conclusion, this project has significantly enriched my understanding of cloud-native development practices, particularly in the realm of CI/CD pipelines. Through hands-on experience with automation and ensuring system resilience, I've gained valuable insights into cloud-based technologies.

This project has not only enhanced my technical skills but has also provided a strong basis for pursuing roles in cloud computing. I'm enthusiastic about applying the knowledge and expertise gained through this project to future opportunities in the cloud domain.

………… Thank you ……….