APPLICATION DEPLOYMENT - MINDTRACK

TASKS

Objective:

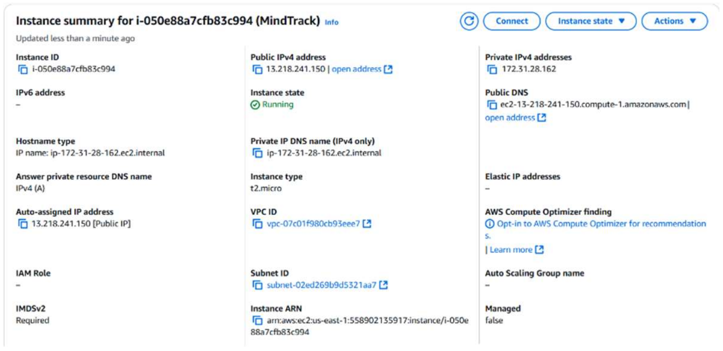
Deploy a Dockerized React application to AWS EKS using a fully automated CI/CD pipeline with GitHub, CodePipeline, CodeBuild, and ECR**.**

Work Flow:

- Create an EC2 instance with the help of AWS Management Console with linux OS of required configuration.
- Now, Connect an EC2 instance with an help of Windows Terminal or Gitbash or Vbox.
- To connect an EC2 instance the command is:
  - ssh -i "**key_file**" ec2-user@"**Public_IP_address**"

Key_file --- Key file of the instance with the extension .pem

Public_IP_address --- Public IP address of the instance.


Instance summary for i-050e88a7cfb83c994 (MindTrack) Info
Updated less than a minute ago

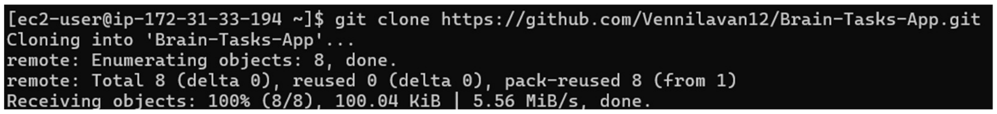
Instance ID i-050e88a7cfb83c994	Public IPv4 address 13.218.241.150 | open address	Private IPv4 addresses 172.31.28.162
IPv6 address –	Instance state Running	Public DNS ec2-13-218-241-150.compute-1.amazonaws.com | open address
Hostname type IP name: ip-172-31-28-162.ec2.internal	Private IP DNS name (IPv4 only) ip-172-31-28-162.ec2.internal	
Answer private resource DNS name IPv4 (A)	Instance type t2.micro	Elastic IP addresses –
Auto-assigned IP address 13.218.241.150 [Public IP]	VPC ID vpc-07c01f980cb93eee7	AWS Compute Optimizer finding Opt-in to AWS Compute Optimizer for recommendations. | Learn more
IAM Role –	Subnet ID subnet-02ed269b9d5321aa7	Auto Scaling Group name –
IMDSv2 Required	Instance ARN arn:aws:ec2:us-east-1:558902135917:instance/i-050e88a7cfb83c994	Managed false

1. Deploy the given React application to a production ready state.

<u>Application:</u>

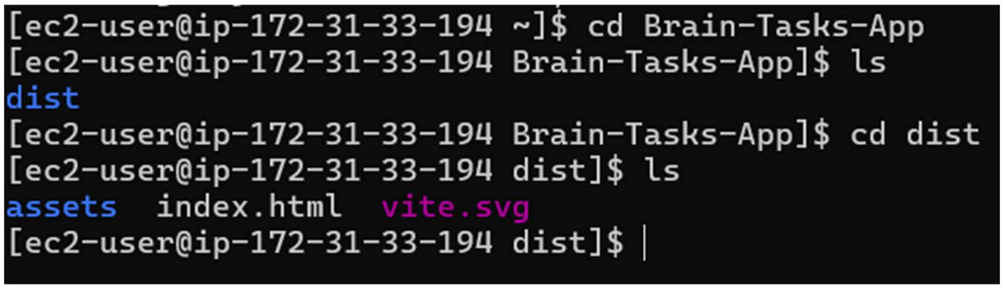
Step 1: Clone the React Application.

- ✓ To begin the deployment process, the first step is to obtain the source code of the React application.
- ✓ The code is publicly available in a GitHub repository. We use the **git clone** command to download the project to our local machine.
- ✓ Command:
  - **git clone https://github.com/Vennilavan12/Brain-Tasks-App.git**
  - **cd Brain-Tasks-App**
- ✓ This command clones the repository named Brain-Tasks-App from GitHub and navigates into the project directory.



```
[ec2-user@ip-172-31-33-194 ~]$ git clone https://github.com/Vennilavan12/Brain-Tasks-App.git
Cloning into 'Brain-Tasks-App'...
remote: Enumerating objects: 8, done.
remote: Total 8 (delta 0), reused 0 (delta 0), pack-reused 8 (from 1)
Receiving objects: 100% (8/8), 100.04 KiB | 5.56 MiB/s, done.
```

- ✓ The application files will be used in the following steps to build a Docker image and deploy it to AWS.

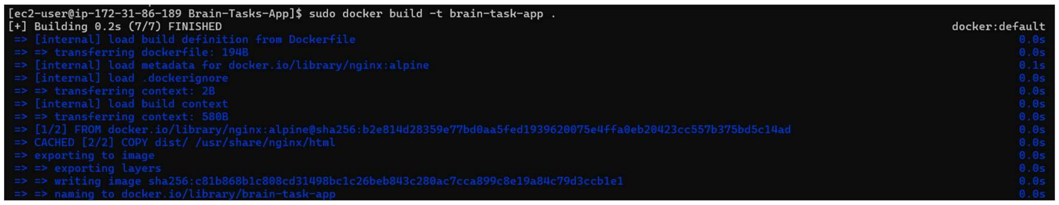


```
[ec2-user@ip-172-31-33-194 ~]$ cd Brain-Tasks-App
[ec2-user@ip-172-31-33-194 Brain-Tasks-App]$ ls
dist
[ec2-user@ip-172-31-33-194 Brain-Tasks-App]$ cd dist
[ec2-user@ip-172-31-33-194 dist]$ ls
assets  index.html  vite.svg
[ec2-user@ip-172-31-33-194 dist]$ |
```

<u>Docker:</u>

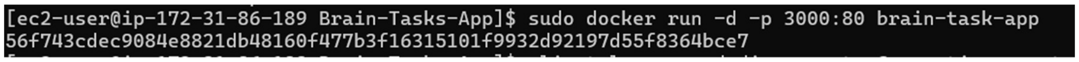
Step 2: Dockerize the Application

- ✓ In this step, we containerize the React application using Docker by creating a Dockerfile in the root directory of the project.
- ✓ Docker Installation:



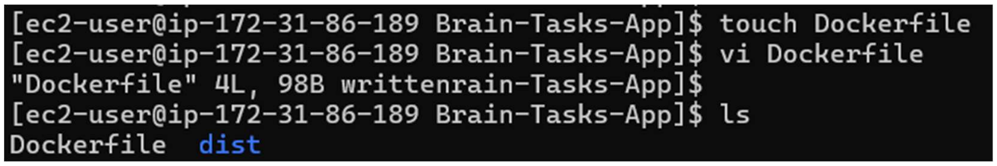
```
[ec2-user@ip-172-31-86-189 Brain-Tasks-App]$ sudo docker build -t brain-task-app .
[+] Building 0.2s (7/7) FINISHED                                                                     docker:default
 => [internal] load build definition from Dockerfile                                                           0.0s
 => => transferring dockerfile: 194B                                                                           0.0s
 => [internal] load metadata for docker.io/library/nginx:alpine                                                0.1s
 => [internal] load .dockerignore                                                                              0.0s
 => => transferring context: 2B                                                                                0.0s
 => [internal] load build context                                                                              0.0s
 => => transferring context: 580B                                                                              0.0s
 => [1/2] FROM docker.io/library/nginx:alpine@sha256:b2e814d28359e77bd0aa5fed1939620075e4ffa0eb20423cc557b375bd5c14ad  0.0s
 => CACHED [2/2] COPY dist/ /usr/share/nginx/html                                                              0.0s
 => exporting to image                                                                                         0.0s
 => => exporting layers                                                                                        0.0s
 => => writing image sha256:c81b868b1c808cd31498bc1c26beb843c280ac7cca899c8e19a84c79d3ccb1e1                   0.0s
 => => naming to docker.io/library/brain-task-app                                                              0.0s
```

- ✓ The Dockerfile uses a multi-stage build where the first stage builds the React application using a Node.js base image, and the second stage uses an Nginx base image to serve the static files generated during the build.
- ✓ This approach ensures that the final image is lightweight and optimized for production.



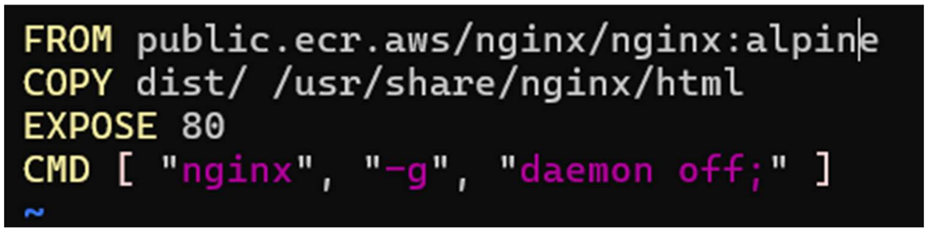
```
[ec2-user@ip-172-31-86-189 Brain-Tasks-App]$ sudo docker run -d -p 3000:80 brain-task-app
56f743cdec9084e8821db48160f477b3f16315101f9932d92197d55f8364bce7
```

- ✓ This confirms that the Dockerized application serves correctly in a production-like containerized environment.



```
[ec2-user@ip-172-31-86-189 Brain-Tasks-App]$ touch Dockerfile
[ec2-user@ip-172-31-86-189 Brain-Tasks-App]$ vi Dockerfile
"Dockerfile" 4L, 98B writtenrain-Tasks-App]$
[ec2-user@ip-172-31-86-189 Brain-Tasks-App]$ ls
Dockerfile  dist
```

✓ The Dockerfile Formate:



```
FROM public.ecr.aws/nginx/nginx:alpine
COPY dist/ /usr/share/nginx/html
EXPOSE 80
CMD [ "nginx", "-g", "daemon off;" ]
~
```

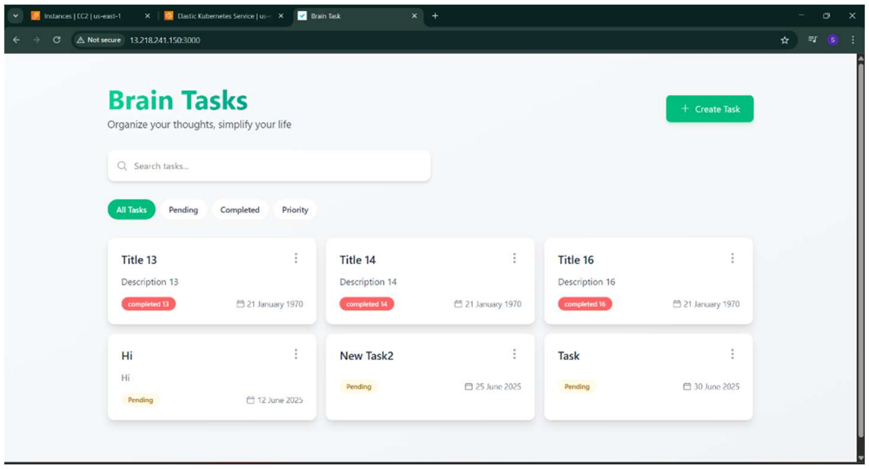
✓ After creating the Dockerfile, we built the Docker image using the docker build command and named the image brain-tasks-app.

✓ To build the dockerfile the command is:

- **docker build -t brain-tasks-app .**
- **docker run -p 3000:80 brain-tasks-app**

✓ Once the image was built successfully, we verified the container by running it locally using the docker run command and mapped it to port 3000.

✓ Now open http://<instance-ip>:3000 in your browser again — the app should load, but this time **inside a Docker container**.

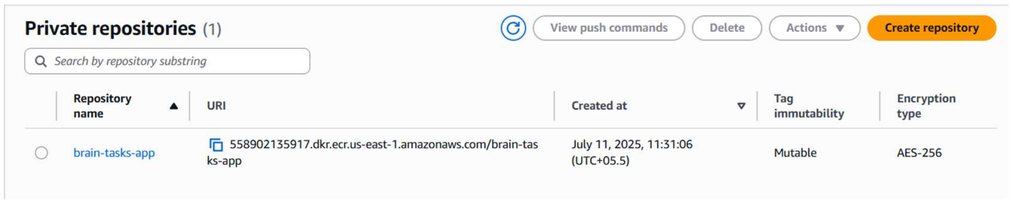


<u>ECR:</u>

Step 3: Push Docker Image to AWS ECR

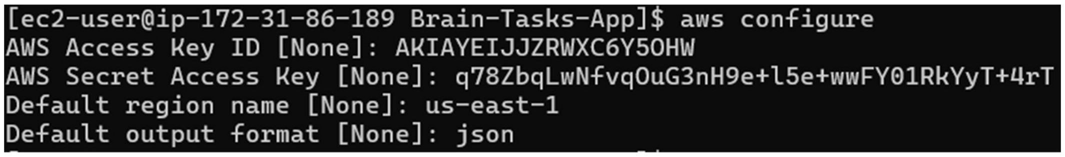
✓ In this step, we store our Docker image in **Amazon Elastic Container Registry (ECR)** so it can be accessed later by our Kubernetes cluster.

- First, we create an ECR repository named `brain-tasks-app` using the AWS CLI. This repository will hold our Docker image in a secure and scalable manner.

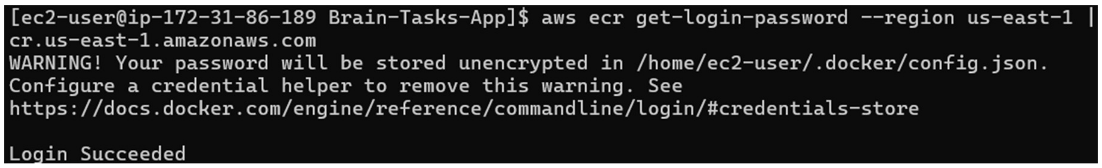


Repository name	URI	Created at	Tag immutability	Encryption type
brain-tasks-app	558902135917.dkr.ecr.us-east-1.amazonaws.com/brain-tasks-app	July 11, 2025, 11:31:06 (UTC+05.5)	Mutable	AES-256

- Next, we authenticate Docker with our AWS ECR registry using the `get-login-password` command. This ensures that we have permission to push images.



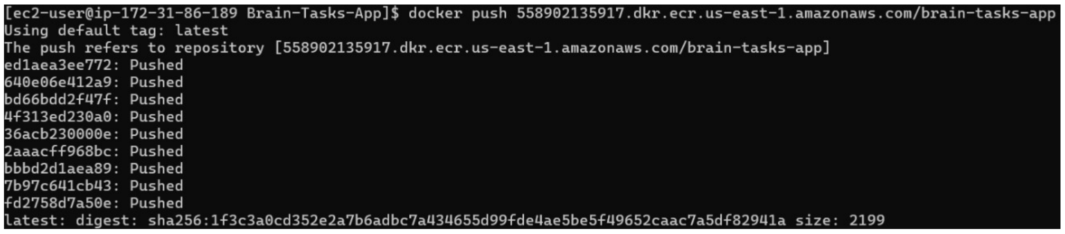
```
[ec2-user@ip-172-31-86-189 Brain-Tasks-App]$ aws configure
AWS Access Key ID [None]: AKIAYEIJJZRWXC6Y5OHW
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-1
Default output format [None]: json
```



```
[ec2-user@ip-172-31-86-189 Brain-Tasks-App]$ aws ecr get-login-password --region us-east-1 |
cr.us-east-1.amazonaws.com
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
```

- After building the Docker image locally, we tag it with the ECR repository URI so that Docker knows where to push the image.



```
[ec2-user@ip-172-31-86-189 Brain-Tasks-App]$ docker push 558902135917.dkr.ecr.us-east-1.amazonaws.com/brain-tasks-app
Using default tag: latest
The push refers to repository [558902135917.dkr.ecr.us-east-1.amazonaws.com/brain-tasks-app]
ed1aea3ee772: Pushed
640e06e412a9: Pushed
bd66bdd2f47f: Pushed
4f313ed230a0: Pushed
36acb230000e: Pushed
2aaacff968bc: Pushed
bbbd2d1aea89: Pushed
7b97c641cb43: Pushed
fd2758d7a50e: Pushed
latest: digest: sha256:1f3c3a0cd352e2a7b6adbc7a434655d99fde4ae5be5f49652caac7a5df82941a size: 2199
```

- Finally, we push the tagged image to ECR. This uploads our image to the cloud and makes it available for deployment in the EKS cluster.

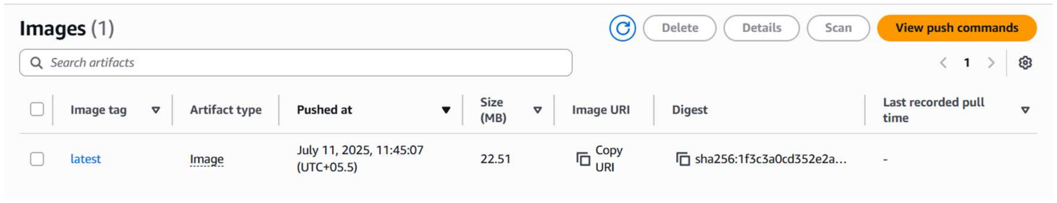


Image tag	Artifact type	Pushed at	Size (MB)	Image URI	Digest	Last recorded pull time
latest	Image	July 11, 2025, 11:45:07 (UTC+05.5)	22.51	Copy URI	sha256:1f3c3a0cd352e2a...	-

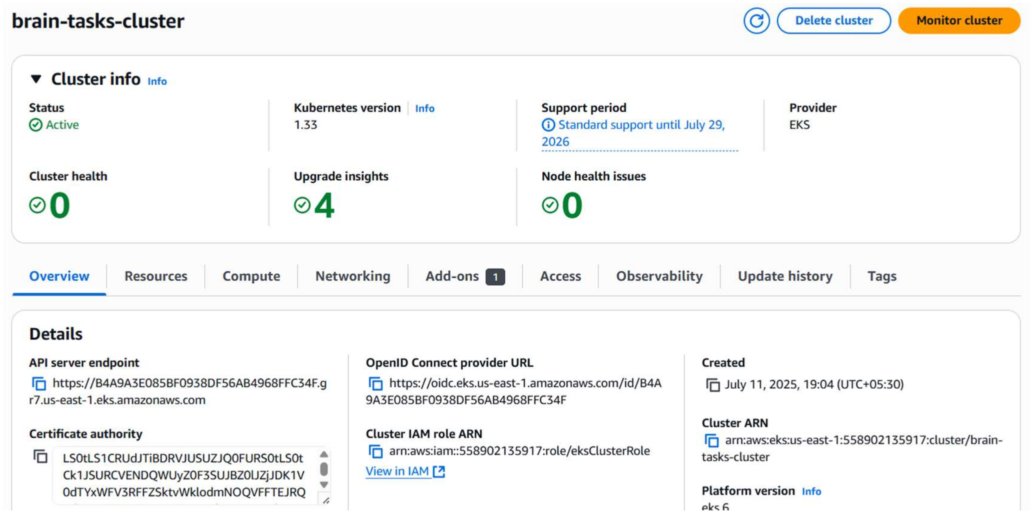
✓ By completing this step, we have successfully stored our application image in a cloud-native registry, making it ready for deployment via Kubernetes.

<u>Kubernetes:</u>

Step 4: Setup Kubernetes in AWS EKS

✓ To deploy our Dockerized React application in a scalable and managed environment, we set up a Kubernetes cluster using AWS EKS (Elastic Kubernetes Service).

- ✓ First, we used the AWS Management Console or the eksctl CLI tool to create a new EKS cluster. The cluster was created in a specific region with default settings, and we waited until the cluster status showed as "Active".



brain-tasks-cluster

Cluster info Info

Status: Active	Kubernetes version: 1.33	Support period: Standard support until July 29, 2026	Provider: EKS
Cluster health: 0	Upgrade insights: 4	Node health issues: 0	

Details

API server endpoint
https://B4A9A3E085BF0938DF56AB4968FFC34F.gr7.us-east-1.eks.amazonaws.com

Certificate authority
LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSURCVENDQWUyZ0F3SUJBZ0lJZjJDK1V0dTYxWFV3RFFZSktvWklodmNOQVFFTEJRQ

OpenID Connect provider URL
https://oidc.eks.us-east-1.amazonaws.com/id/B4A9A3E085BF0938DF56AB4968FFC34F

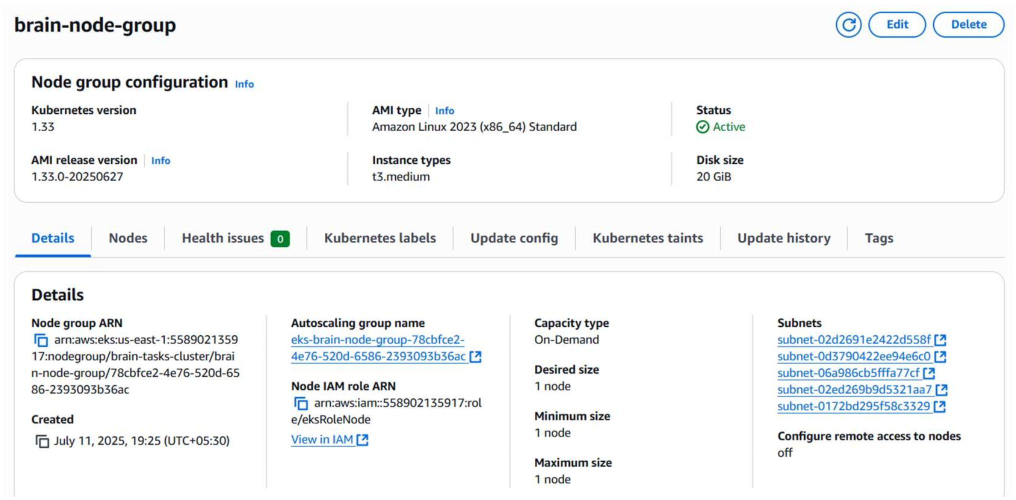
Cluster IAM role ARN
arn:aws:iam::558902135917:role/eksClusterRole
View in IAM

Created
July 11, 2025, 19:04 (UTC+05:30)

Cluster ARN
arn:aws:eks:us-east-1:558902135917:cluster/brain-tasks-cluster

Platform version Info
eks.6

- ✓ After the cluster was created, we configured our local machine to interact with it using kubectl by updating the kubeconfig file through the AWS CLI. This allowed us to run commands against the EKS cluster from our terminal.



brain-node-group

Node group configuration Info

Kubernetes version: 1.33	AMI type: Amazon Linux 2023 (x86_64) Standard	Status: Active
AMI release version: 1.33.0-20250627	Instance types: t3.medium	Disk size: 20 GiB

Details

Node group ARN
arn:aws:eks:us-east-1:558902135917:nodegroup/brain-tasks-cluster/brain-node-group/78cbfce2-4e76-520d-6586-2393093b36ac

Created
July 11, 2025, 19:25 (UTC+05:30)

Autoscaling group name
eks-brain-node-group-78cbfce2-4e76-520d-6586-2393093b36ac

Node IAM role ARN
arn:aws:iam::558902135917:role/eksRoleNode
View in IAM

Capacity type
On-Demand

Desired size
1 node

Minimum size
1 node

Maximum size
1 node

Subnets
subnet-02d2691e2422d558f
subnet-0d3790422ee94e6c0
subnet-06a986cb5fffa77cf
subnet-02ed269b9d5321aa7
subnet-0172bd295f58c3329

Configure remote access to nodes
off

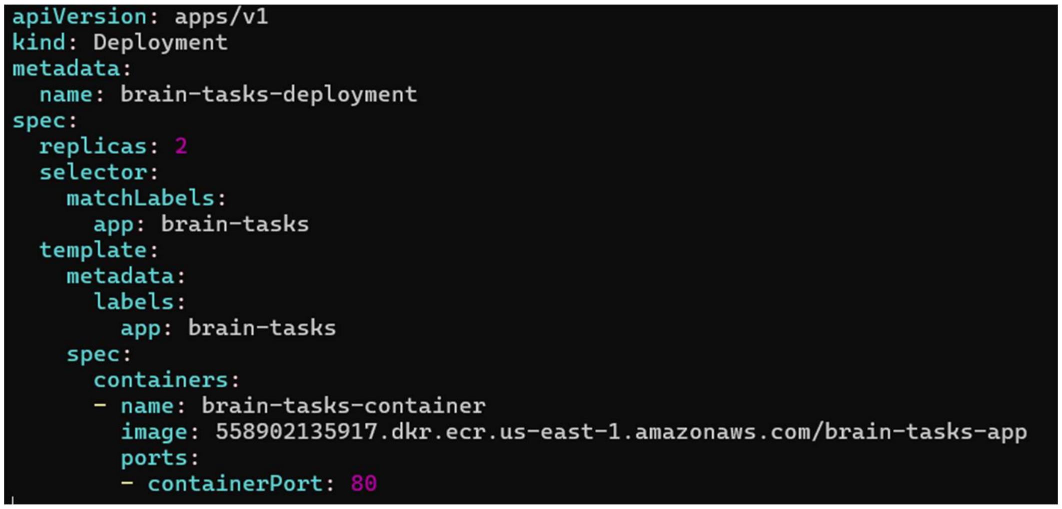
- ✓ We verified the setup by running the command kubectl get svc which confirmed that our EKS cluster was running and accessible.
- ✓ By completing this step, we now have a managed Kubernetes environment where we can deploy our React application using Kubernetes manifests in the next step.

Step 5: Create Kubernetes Deployment and Service YAML Files

- ✓ In this step, we define the Kubernetes configuration files required to deploy our Dockerized React application to the EKS cluster.
- ✓ First, we create a deployment.yaml file which describes the desired state of the application, including the number of replicas, the Docker image to use from ECR, and the

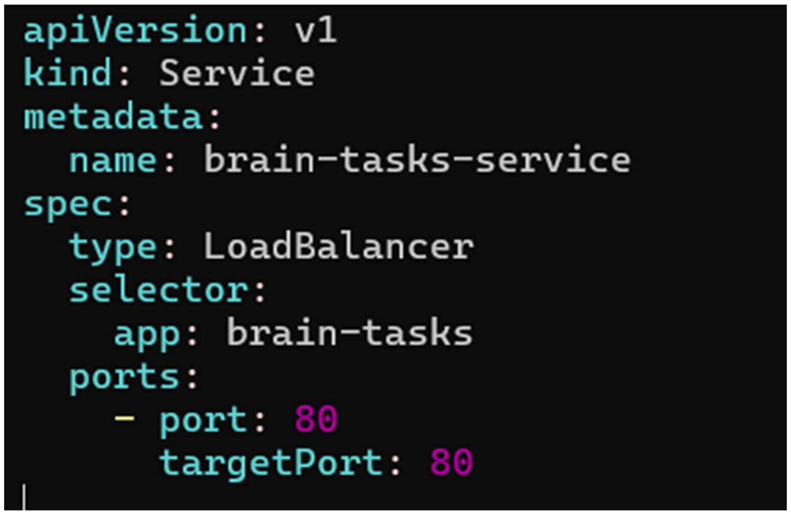
container port. This configuration ensures that the application runs in multiple pods for high availability.

- ✓ The deployment.yaml file formate:



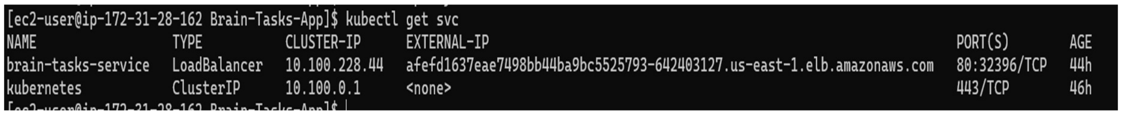
```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: brain-tasks-deployment
spec:
  replicas: 2
  selector:
    matchLabels:
      app: brain-tasks
  template:
    metadata:
      labels:
        app: brain-tasks
    spec:
      containers:
      - name: brain-tasks-container
        image: 558902135917.dkr.ecr.us-east-1.amazonaws.com/brain-tasks-app
        ports:
        - containerPort: 80
```

- ✓ Next, we create a service.yaml file to expose the application to the internet using a LoadBalancer. This allows users to access the application from outside the Kubernetes cluster.
- ✓ The service.yaml file formate:



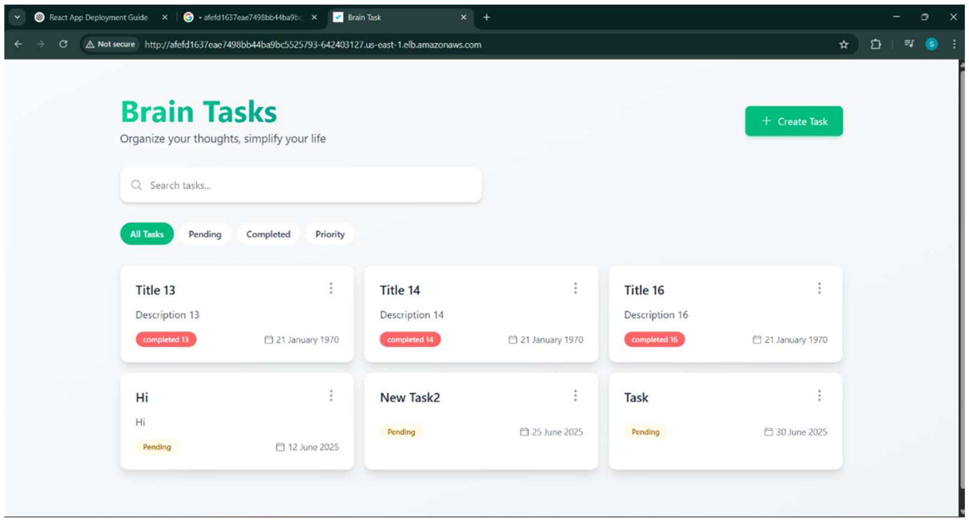
```yaml
apiVersion: v1
kind: Service
metadata:
  name: brain-tasks-service
spec:
  type: LoadBalancer
  selector:
    app: brain-tasks
  ports:
    - port: 80
      targetPort: 80
```

- ✓ After writing both files, we deploy them to the EKS cluster using the kubectl apply command. This command instructs Kubernetes to create the necessary resources based on our configuration.
- ✓ Commands:
  - **kubectl apply -f deployment.yaml**
  - **kubectl apply -f service.yaml**
- ✓ By completing this step, our application is now deployed and accessible through the public IP address provided by the LoadBalancer service.
- ✓ Command:
  - **kubectl get svc**



```
[ec2-user@ip-172-31-28-162 Brain-Tasks-App]$ kubectl get svc
NAME                 TYPE           CLUSTER-IP      EXTERNAL-IP                                                              PORT(S)        AGE
brain-tasks-service  LoadBalancer   10.100.228.44   afefd1637eae7498bb44ba9bc5525793-642403127.us-east-1.elb.amazonaws.com   80:32396/TCP   44h
kubernetes           ClusterIP      10.100.0.1      <none>                                                                   443/TCP        46h
```

- ✓ It should show an **EXTERNAL-IP** for the LoadBalancer — that's the URL to access your app!
    - **a966e5b1b41ea4417817f66e3b8411c7-591184991.us-east-1.elb.amazonaws.com**



http://afefd1637eae7498bb44ba9bc5525793-642403127.us-east-1.elb.amazonaws.com

Brain Tasks

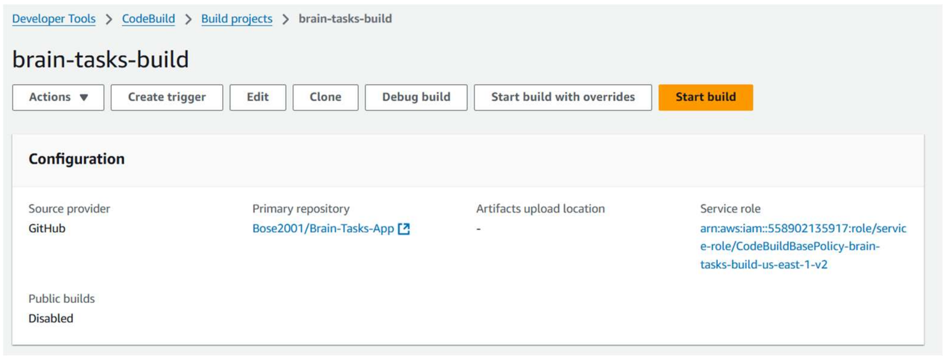
Organize your thoughts, simplify your life

Title 13 Description 13 completed 13 · 21 January 1970	Title 14 Description 14 completed 14 · 21 January 1970	Title 16 Description 16 completed 16 · 21 January 1970
Hi Hi Pending · 12 June 2025	New Task2 Pending · 25 June 2025	Task Pending · 30 June 2025

CodeBuild:

Step 6: Set Up AWS CodeBuild

- ✓ In this step, we automate the process of building and pushing the Docker image to Amazon ECR using AWS CodeBuild.



Developer Tools > CodeBuild > Build projects > brain-tasks-build

brain-tasks-build

Configuration

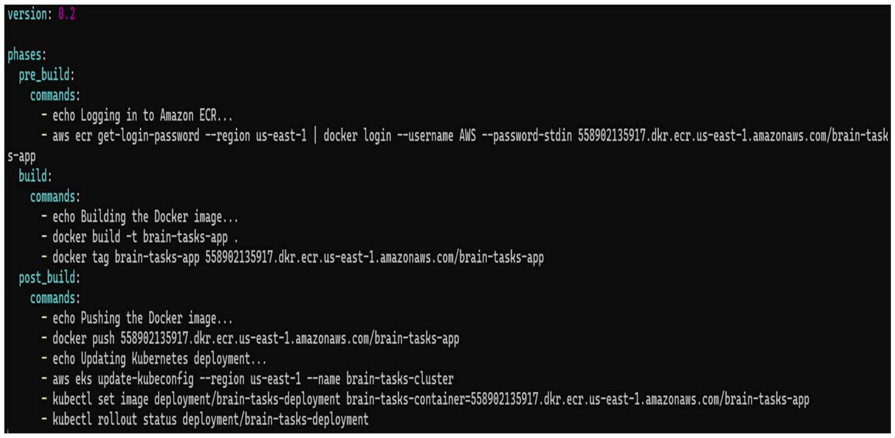
Source provider	Primary repository	Artifacts upload location	Service role
GitHub	Bose2001/Brain-Tasks-App	-	arn:aws:iam::558902135917:role/service-role/CodeBuildBasePolicy-brain-tasks-build-us-east-1-v2

Public builds
Disabled

- ✓ First, a buildspec.yml file is created in the root directory of the project. This file defines the build phases and the commands required to build the Docker image and push it to the ECR repository.
- ✓ The buildspec.yml file includes the following phases:
    - **Pre-Build Phase**: Authenticates Docker with Amazon ECR using the AWS CLI.

- **Build Phase**: Builds the Docker image from the Dockerfile and tags it with the ECR repository URL.
- **Post-Build Phase**: Pushes the tagged Docker image to the ECR repository.

✓ The buildspec.yml file format:



```
version: 0.2

phases:
  pre_build:
    commands:
      - echo Logging in to Amazon ECR...
      - aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 558902135917.dkr.ecr.us-east-1.amazonaws.com/brain-task
s-app
  build:
    commands:
      - echo Building the Docker image...
      - docker build -t brain-tasks-app .
      - docker tag brain-tasks-app 558902135917.dkr.ecr.us-east-1.amazonaws.com/brain-tasks-app
  post_build:
    commands:
      - echo Pushing the Docker image...
      - docker push 558902135917.dkr.ecr.us-east-1.amazonaws.com/brain-tasks-app
      - echo Updating Kubernetes deployment...
      - aws eks update-kubeconfig --region us-east-1 --name brain-tasks-cluster
      - kubectl set image deployment/brain-tasks-deployment brain-tasks-container=558902135917.dkr.ecr.us-east-1.amazonaws.com/brain-tasks-app
      - kubectl rollout status deployment/brain-tasks-deployment
```

✓ After creating the buildspec.yml file, an AWS CodeBuild project is set up in the AWS Management Console**.**

✓ The source is connected to the GitHub repository where the project code is hosted. The environment is configured to use a managed image (e.g., Amazon Linux 2 with Docker support).

✓ Finally, CodeBuild is triggered to execute the instructions in buildspec.yml, which results in a new Docker image being pushed to ECR.

✓ This step ensures that every time changes are pushed to the repository, the latest version of the application is automatically built and stored in ECR for deployment.

<u>**CodeDeploy:**</u>

Step 7: Configure CodeDeploy for EKS

✓ In this step, we configure AWS CodeDeploy to deploy our application to the EKS cluster using Kubernetes manifests.

✓ We begin by creating an appspec.yml file, which defines how CodeDeploy will handle the deployment process. This file specifies the target EKS cluster, the namespace (default in our case), and the Kubernetes resource files that need to be applied (i.e., deployment.yaml and service.yaml).

✓ The appspec.yml acts as the deployment blueprint for CodeDeploy and ensures that the application is rolled out to the EKS cluster in a structured and repeatable manner.

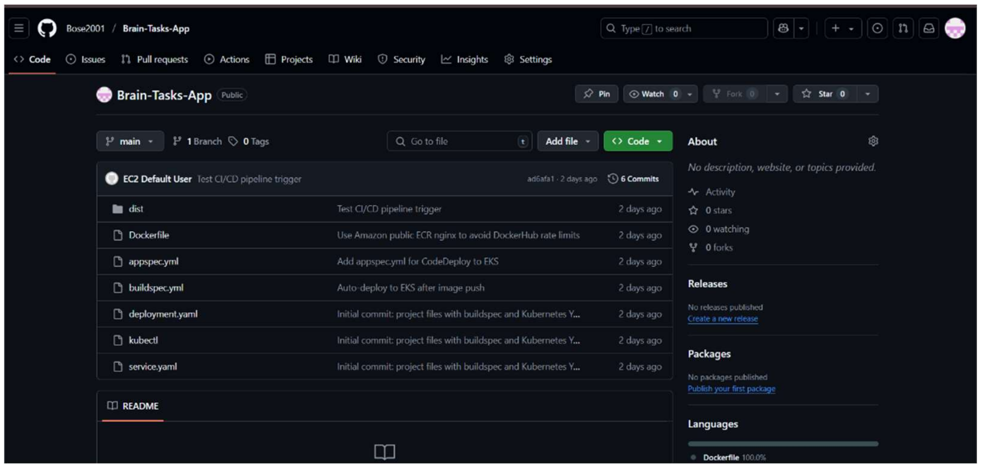
✓ This configuration is necessary for CodeDeploy to locate and apply the appropriate manifests to the correct cluster environment.

- ✓ By linking CodeDeploy with EKS using this setup, we enable automated and consistent deployment of containerized applications directly into our Kubernetes cluster.

<u>Version Control:</u>

Step 8: Push the Project Code to GitHub

- ✓ In this step, the complete project codebase, including the React application, Dockerfile, Kubernetes deployment files, and CI/CD configuration files (like buildspec.yml and appspec.yml), was pushed to a GitHub repository.
- ✓ The process began by initializing a Git repository in the project folder using the git init command.
- ✓ The remote repository was then added using the git remote add origin command, pointing to the desired GitHub repository URL. After staging all the files using git add . and committing the changes with git commit -m "Initial commit", the project was successfully pushed to the main branch using git push -u origin main.
- ✓ The commands:
  - git init
  - git remote add origin https://github.com/<your-username>/<your-repo-name>.git
  - git add .
  - git commit -m "Initial commit: Docker, buildspec, and Kubernetes files"
  - git push -u origin main
- ✓ This step ensures the codebase is version controlled and integrated with AWS CodePipeline for automated build and deployment.



EC2 Default User Test CI/CD pipeline trigger ad6afa1 · 2 days ago 6 Commits

dist	Test CI/CD pipeline trigger	2 days ago
Dockerfile	Use Amazon public ECR nginx to avoid DockerHub rate limits	2 days ago
appspec.yml	Add appspec.yml for CodeDeploy to EKS	2 days ago
buildspec.yml	Auto-deploy to EKS after image push	2 days ago
deployment.yaml	Initial commit: project files with buildspec and Kubernetes Y...	2 days ago
kubectl	Initial commit: project files with buildspec and Kubernetes Y...	2 days ago
service.yaml	Initial commit: project files with buildspec and Kubernetes Y...	2 days ago

About

No description, website, or topics provided.

Activity
0 stars
0 watching
0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

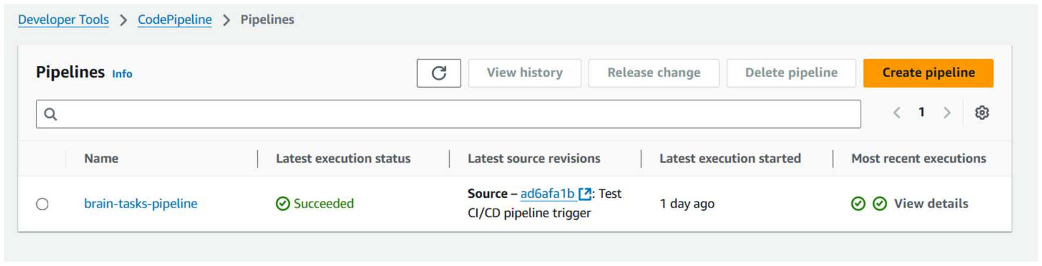
Languages

Dockerfile 100.0%

<u>CodePipeline:</u>

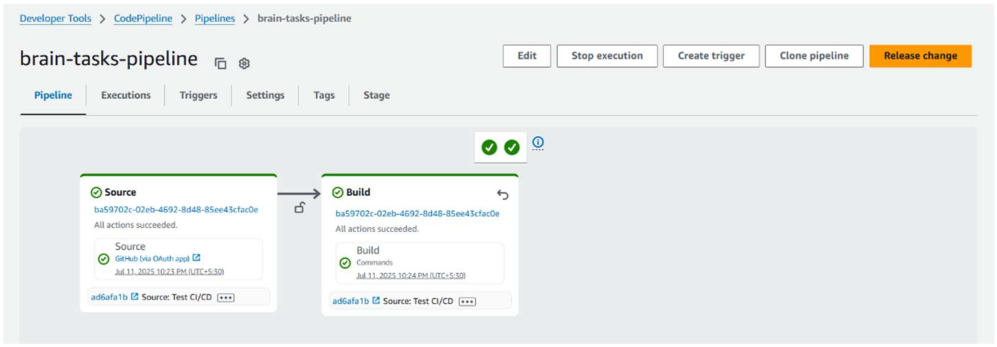
Step 9: Set Up CodePipeline

- ✓ In this step, we configured AWS CodePipeline to automate the end-to-end CI/CD process, which connects our GitHub repository to AWS CodeBuild and AWS CodeDeploy.
- ✓ First, we selected **GitHub** as the source provider and authorized access to the repository containing our application code, Dockerfile, and deployment manifests.
- ✓ Next, we added a **Build stage** using the previously created **CodeBuild project**, which is responsible for building the Docker image and pushing it to AWS ECR based on the instructions defined in the buildspec.yml file.



Name	Latest execution status	Latest source revisions	Latest execution started	Most recent executions
brain-tasks-pipeline	Succeeded	Source – ad6afa1b: Test CI/CD pipeline trigger	1 day ago	✓ ✓ View details

- ✓ Finally, we added a **Deploy stage** by integrating **AWS CodeDeploy**, which uses the appspec.yml to deploy the application to the EKS cluster using the Kubernetes manifests (deployment.yaml and service.yaml).
- ✓ Once the pipeline was created and executed, it automatically triggered a new build and deployment whenever code changes were pushed to the GitHub repository.
- ✓ This ensures that our application remains up-to-date and follows a continuous delivery workflow.



Source
ba59702c-02eb-4692-8d48-85ee43cfac0e
All actions succeeded.
Source
GitHub (via OAuth app)
Jul 11, 2025 10:23 PM (UTC+5:30)
ad6afa1b Source: Test CI/CD

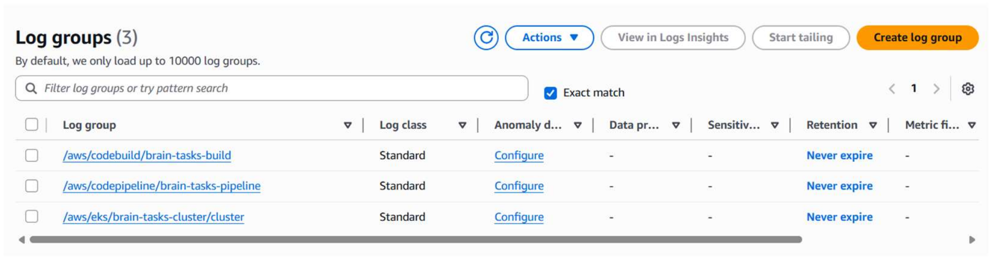
Build
ba59702c-02eb-4692-8d48-85ee43cfac0e
All actions succeeded.
Build
Commands
Jul 11, 2025 10:24 PM (UTC+5:30)
ad6afa1b Source: Test CI/CD

<u>Monitoring:</u>

Step 10: Monitoring with CloudWatch

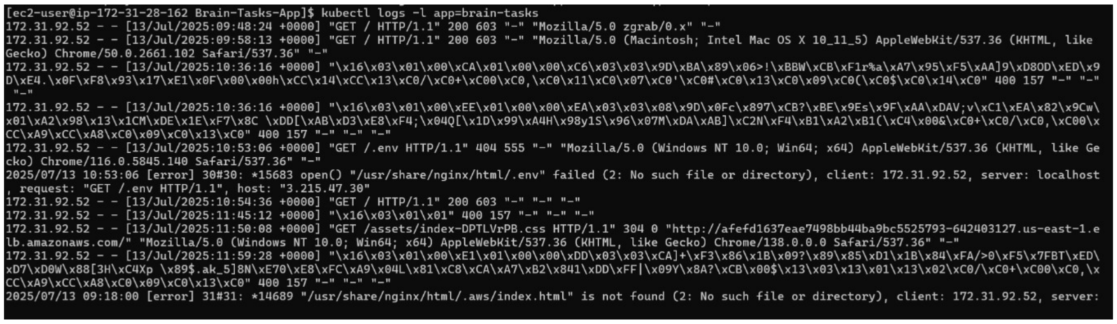
- ✓ To monitor the application and deployment pipeline, we used **Amazon CloudWatch** for centralized logging and performance tracking.
- ✓ CloudWatch helps in identifying issues during the build, deployment, and runtime of the application.

- ✓ For CodeBuild and CodeDeploy, logs are automatically pushed to CloudWatch, where we can view detailed build and deployment history, error messages, and timestamps to troubleshoot issues.


Log groups (3)
Actions
View in Logs Insights
Start tailing
Create log group
By default, we only load up to 10000 log groups.
Filter log groups or try pattern search
Exact match
1

Log group	Log class	Anomaly d...	Data pr...	Sensitiv...	Retention	Metric fi...
/aws/codebuild/brain-tasks-build	Standard	Configure	-	-	Never expire	-
/aws/codepipeline/brain-tasks-pipeline	Standard	Configure	-	-	Never expire	-
/aws/eks/brain-tasks-cluster/cluster	Standard	Configure	-	-	Never expire	-

- ✓ Additionally, for monitoring the **application running in the EKS cluster**, we used the following command to fetch logs directly from the application pods:
  - **kubectl logs -l app=brain-tasks**

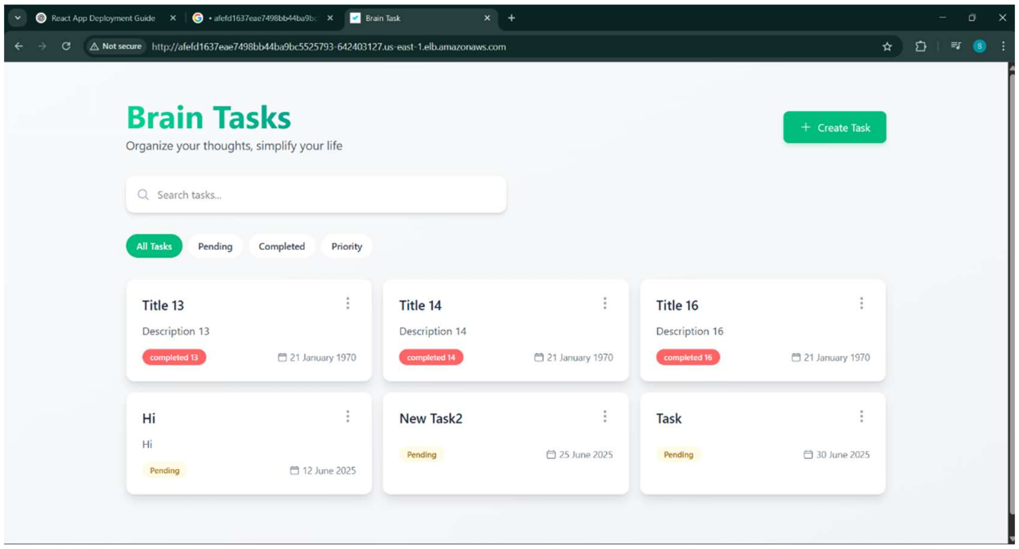


```
[ec2-user@ip-172-31-28-162 Brain-Tasks-App]$ kubectl logs -l app=brain-tasks
172.31.92.52 - - [13/Jul/2025:09:48:24 +0000] "GET / HTTP/1.1" 200 603 "-" "Mozilla/5.0 zgrab/0.x" "-"
172.31.92.52 - - [13/Jul/2025:09:58:13 +0000] "GET / HTTP/1.1" 200 603 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_11_5) AppleWebKit/537.36 (KHTML, like
Gecko) Chrome/50.0.2661.102 Safari/537.36" "-"
172.31.92.52 - - [13/Jul/2025:10:36:16 +0000] "\x16\x03\x01\x00\xCA\x01\x00\x00\xC6\x03\x03\x9D\xBA\x89\x06>!\xBBW\xCB\xF1r%a\xA7\x95\xF5\xAA]9\xD8OD\xED\x9
D\xE4.\x0F\xF8\x93\x17\xE1\x0F\x00\x00h\xCC\x14\xCC\x13\xC0/\xC0+\xC00\xC0,\xC0\x11\xC0\x07\xC0'\xC0#\xC0\x13\xC0\x09\xC0(\xC0$\xC0\x14\xC0" 400 157 "-" "-"
 "-"
172.31.92.52 - - [13/Jul/2025:10:36:16 +0000] "\x16\x03\x01\x00\xEE\x01\x00\x00\xEA\x03\x03\x08\x9D\x0Fc\x897\xCB?\xBE\x9Es\x9F\xAA\xDAV;v\xC1\xEA\x82\x9Cw\
x01\xA2\x98\x13\x1CM\xDE\x1E\xF7\x8C \xDD[\xAB\xD3\xE8\xF4;\x04Q[\x1D\x99\xA4H\x98y1S\x96\x07M\xDA\xAB]\xC2N\xF4\xB1\xA2\xB1(\xC4\x00&\xC0+\xC0/\xC0,\xC00\x
CC\xA9\xCC\xA8\xC0\x09\xC0\x13\xC0" 400 157 "-" "-" "-"
172.31.92.52 - - [13/Jul/2025:10:53:06 +0000] "GET /.env HTTP/1.1" 404 555 "-" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Ge
cko) Chrome/116.0.5845.140 Safari/537.36" "-"
2025/07/13 10:53:06 [error] 30#30: *15683 open() "/usr/share/nginx/html/.env" failed (2: No such file or directory), client: 172.31.92.52, server: localhost
, request: "GET /.env HTTP/1.1", host: "3.215.47.30"
172.31.92.52 - - [13/Jul/2025:10:54:36 +0000] "GET / HTTP/1.1" 200 603 "-" "-" "-"
172.31.92.52 - - [13/Jul/2025:11:45:12 +0000] "\x16\x03\x01\x01" 400 157 "-" "-" "-"
172.31.92.52 - - [13/Jul/2025:11:50:08 +0000] "GET /assets/index-DPTLVrPB.css HTTP/1.1" 304 0 "http://afefd1637eae7498bb44ba9bc5525793-642403127.us-east-1.e
lb.amazonaws.com/" "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/138.0.0.0 Safari/537.36" "-"
172.31.92.52 - - [13/Jul/2025:11:59:28 +0000] "\x16\x03\x01\x00\xE1\x01\x00\x00\xDD\x03\x03\xCA]+\xF3\x86\x1B\x09?\x89\x85\xD1\x1B\x84\xFA/>0\xF5\x7FBT\xED\
xD7\xD0W\x88[3H\xC4Xp \x89$.ak_5]8N\xE70\xE8\xFC\xA9\x04L\x81\xC8\xCA\xA7\xB2\x841\xDD\xFF|\x09Y\x8A?\xCB\x00$\x13\x03\x13\x01\x13\x02\xC0/\xC0+\xC00\xC0,\x
CC\xA9\xCC\xA8\xC0\x09\xC0\x13\xC0" 400 157 "-" "-" "-"
2025/07/13 09:18:00 [error] 31#31: *14689 "/usr/share/nginx/html/.aws/index.html" is not found (2: No such file or directory), client: 172.31.92.52, server:
```

- ✓ This command retrieves the logs from the containers matching the label app=brain-tasks-app, helping to verify that the application is running correctly and to check for any runtime errors.
- ✓ In summary, CloudWatch provides an essential observability layer to track the health of our CI/CD pipeline and application, ensuring reliable and continuous deployment.

Conclusion:

- ✓ Successfully deployed the "Brain Tasks App" React application to a production-ready environment on AWS using Docker, ECR, EKS, CodeBuild, CodeDeploy, and CodePipeline. The process is fully automated with monitoring enabled via CloudWatch.
- ✓ **Application deployed kubernetes Loadbalancer ARN.**
  - http://a966e5b1b41ea4417817f66e3b8411c7-591184991.us-east-1.elb.amazonaws.com



************** **TASK COMPLETED** **************