**Application Deployment**

(Deploy the given React application to a production ready state)

**Application:**

Clone the below mentioned repository and deploy the application (Run application in port 3000).

Repo URL : <https://github.com/Vennilavan12/Brain-Tasks-App.git>

**Docker:**

* Dockerize the application by creating Dockerfile
* Build an application and check output using docker image.

**ECR:**

* Create an AWS ECR repository for store docker images.

**Kubernetes:**

* Setup Kubernetes in AWS EKS and Confirm EKS cluster is running.
* Write deployment and service YAML files.
* Deploy using kubectl via Codedeploy.

**CodeBuild:**

* Create a CodeBuild project:
* Source: Connect to your repository
* Environment: Use managed image (Amazon Linux, Ubuntu)
* Write and define commands in buildspec.yml.

**CodeDeploy:**

* Create codedeploy application.
* create appspec.yml file to deploy applications in EKS.

**Version Control:**

* Push the codebase to a Git provider (GitHub).
* Use CLI commands to push code.

**CodePipeline:**

* Source: GitHub
* Build: AWS CodeBuild project
* Deploy: AWS CodeDeploy or deploy to EKS via Lambda or custom script.

**Monitoring:**

* Use CloudWatch Logs to track build, deploy, and application logs.

**Submission Guidelines:**

* GitHub Link: Submit full code repository.
* README File: Include setup instructions, pipeline explanation, and screenshots or create a document and attach all screenshots.
* Application deployed kubernetes Loadbalancer ARN.

Brain-tasks-app — Deployment Guide & Files

**Brain-Tasks-App — Full Deployment Guide**

**Goal:** Deploy the Brain-Tasks-App React application to production on port **3000** using Docker → ECR → EKS, automated with AWS CodeBuild / CodeDeploy / CodePipeline. This document contains step-by-step commands, sample files (Dockerfile, buildspec.yml, appspec.yml), Kubernetes manifests, and a README template to submit as your assignment.

**1. Prerequisites**

* AWS account with permissions: ECR, EKS, IAM, CloudFormation/eksctl, CodeBuild, CodeDeploy, CodePipeline, CloudWatch, S3.
* AWS CLI installed and configured (aws configure).
* kubectl installed.
* eksctl installed (or use AWS Console + CloudFormation).
* Docker installed locally.
* Git and GitHub account/repo ready.
* Node.js (for local test builds)

**2. Clone repository (local)**

# clone

git clone https://github.com/Vennilavan12/Brain-Tasks-App.git

cd Brain-Tasks-App

Run locally to verify:

npm install

npm run build # or npm start for dev

# To run on port 3000 in production mode, you may use a simple static server or configure the Dockerfile to run on 3000

## 3. Dockerize (Dockerfile)

Create Dockerfile at repo root (example):

# Build stage

FROM node:18-alpine AS build

WORKDIR /app

COPY package\*.json ./

RUN npm ci

COPY . .

RUN npm run build

# Serve stage

FROM node:18-alpine AS runtime

WORKDIR /app

# install a lightweight static server (serve) or use an express server

RUN npm i -g serve

COPY --from=build /app/build ./build

# serve the build folder on port 3000

EXPOSE 3000

CMD ["serve", "-s", "build", "-l", "3000"]

If your app uses react-scripts start, it runs in dev mode; the recommended approach is a production build served by serve or an nginx image. Above uses serve for simplicity.

Build & test locally:

docker build -t brain-tasks-app:local .

docker run -p 3000:3000 brain-tasks-app:local

# open http://localhost:3000

## 4. Create AWS ECR repository & push image

**Create repo:**

aws ecr create-repository --repository-name brain-tasks-app --region <your-region>

Authenticate Docker to ECR:

aws ecr get-login-password --region <your-region> | docker login --username AWS --password-stdin <aws\_account\_id>.dkr.ecr.<region>.amazonaws.com

**Tag & push:**

export REPO\_URI=<aws\_account\_id>.dkr.ecr.<region>.amazonaws.com/brain-tasks-app

docker build -t brain-tasks-app:latest .

docker tag brain-tasks-app:latest $REPO\_URI:latest

docker push $REPO\_URI:latest

## 5. Setup EKS cluster

Option A — eksctl (recommended for quick setup):

eksctl create cluster \

--name brain-tasks-cluster \

--region <region> \

--nodes 2 \

--node-type t3.medium

# verify

kubectl get nodes

Option B — AWS Console (EKS) + managed nodegroup.

Ensure kubectl context points to the cluster.

## 6. Kubernetes manifests

### deployment.yaml

apiVersion: apps/v1

kind: Deployment

metadata:

name: brain-tasks-deployment

labels:

app: brain-tasks

spec:

replicas: 2

selector:

matchLabels:

app: brain-tasks

template:

metadata:

labels:

app: brain-tasks

spec:

containers:

- name: brain-tasks

image: <aws\_account\_id>.dkr.ecr.<region>.amazonaws.com/brain-tasks-app:latest

ports:

- containerPort: 3000

resources:

requests:

cpu: "100m"

memory: "128Mi"

limits:

cpu: "250m"

memory: "256Mi"

### service.yaml (LoadBalancer)

apiVersion: v1

kind: Service

metadata:

name: brain-tasks-svc

spec:

selector:

app: brain-tasks

type: LoadBalancer

ports:

- protocol: TCP

port: 80

targetPort: 3000

Apply:

kubectl apply -f deployment.yaml

kubectl apply -f service.yaml

kubectl rollout status deployment/brain-tasks-deployment

kubectl get svc brain-tasks-svc -w

When kubectl get svc shows an external IP / ELB hostname, note the LoadBalancer ARN from AWS Console (EC2 > Load Balancers) — that's your submission artifact.

## 7. CodeBuild — buildspec.yml

Create buildspec.yml at repo root. This build will build the Docker image and push to ECR.

version: 0.2

env:

variables:

IMAGE\_REPO\_NAME: brain-tasks-app

AWS\_DEFAULT\_REGION: <region>

phases:

pre\_build:

commands:

- echo Logging in to Amazon ECR...

- aws --version

- $(aws ecr get-login-password --region $AWS\_DEFAULT\_REGION | docker login --username AWS --password-stdin <aws\_account\_id>.dkr.ecr.$AWS\_DEFAULT\_REGION.amazonaws.com)

- REPOSITORY\_URI=<aws\_account\_id>.dkr.ecr.$AWS\_DEFAULT\_REGION.amazonaws.com/$IMAGE\_REPO\_NAME

- echo Repository URI is $REPOSITORY\_URI

build:

commands:

- echo Build started on `date`

- docker build -t $IMAGE\_REPO\_NAME:latest .

- docker tag $IMAGE\_REPO\_NAME:latest $REPOSITORY\_URI:latest

post\_build:

commands:

- echo Pushing the Docker image...

- docker push $REPOSITORY\_URI:latest

- printf '[{"name":"brain-tasks","imageUri":"%s"}]' $REPOSITORY\_URI:latest > imagedefinitions.json

artifacts:

files: imagedefinitions.json

**CodeBuild settings:**

* Environment image: Managed image (e.g., Ubuntu standard) with Docker enabled (Privileged mode = true) because we build Docker image.
* Service role: give permissions for ECR push and CloudWatch logs.

**8. AppSpec for CodeDeploy (deploy to EKS)**

For using CodeDeploy to deploy to EKS (Blue/Green or in-place), appspec.yml can reference Kubernetes manifests or use a deployment script. A simple appspec.yml (for CodeDeploy with ECS is different). For EKS, you might use CodePipeline + CodeBuild + a deploy step with a custom action or a Lambda that runs kubectl apply with the new image.

**Simple approach (recommended for this project):**

* Use CodePipeline with: Source (GitHub) → Build (CodeBuild builds/pushes image and outputs imagedefinitions.json) → Deploy (a CodeBuild deploy job that runs kubectl set image or kubectl apply).

**appspec.yml (if you still use CodeDeploy custom):**

version: 0.0

Resources: []

# Note: CodeDeploy's AppSpec for EKS is non-trivial. For this assignment a CodeBuild deployment step that runs kubectl is simpler and more common.

If you must create appspec.yml, create a lightweight script that applies deployment.yaml and service.yaml. Put appspec.yml that points to that script.

## 9. Using CodeBuild to deploy (deploy phase commands)

Create a second CodeBuild project (or reuse the same via multiple actions) for deployment. This project's buildspec can grab the image URI from the previous build artifact and update the deployment.

Example buildspec-deploy.yml:

version: 0.2

env:

variables:

KUBE\_NAMESPACE: default

phases:

install:

commands:

- curl -LO "https://dl.k8s.io/release/$(curl -L -s https://dl.k8s.io/release/stable.txt)/bin/linux/amd64/kubectl"

- chmod +x kubectl

- mv kubectl /usr/local/bin/

- aws eks update-kubeconfig --name brain-tasks-cluster --region <region>

build:

commands:

- echo "Updating deployment image"

- kubectl set image deployment/brain-tasks-deployment brain-tasks=<aws\_account\_id>.dkr.ecr.<region>.amazonaws.com/brain-tasks-app:latest -n $KUBE\_NAMESPACE

- kubectl rollout status deployment/brain-tasks-deployment -n $KUBE\_NAMESPACE

Make sure the CodeBuild role has permissions to call eks:DescribeCluster and to read the cluster's kubeconfig (or attach IRSA / aws-auth mapping for the cluster worker role). Alternatively, use a deploy Lambda.

## 10. CodeDeploy pipeline (alternate)

If required to use CodeDeploy explicitly, follow AWS docs for integrating CodeDeploy with EKS (Blue/Green) — it needs an agent and proper AppSpec. For this assignment, **simpler and robust** is: use CodePipeline with two CodeBuild actions (build and deploy). Put a small shell script deploy.sh that runs kubectl apply / set image.

**11. Git & Version Control (push to GitHub)**

# create repository on GitHub manually or via CLI

git init

git add .

git commit -m "Initial: dockerize and add CI/CD files"

git remote add origin https://github.com/<your-username>/Brain-Tasks-App.git

git push -u origin main

Include these files in your repo: Dockerfile, buildspec.yml, buildspec-deploy.yml (or scripts), deployment.yaml, service.yaml, deploy.sh, README.md, and screenshots folder.

**12. CodePipeline configuration (high-level)**

1. **Source**: GitHub connection (OAuth or GitHub App)
2. **Build**: CodeBuild project that builds Docker image and pushes to ECR (uses buildspec.yml) — outputs imagedefinitions.json
3. **Deploy**: CodeBuild deploy project (uses buildspec-deploy.yml) that runs kubectl to update the EKS deployment.

Alternative: Use AWS CodeDeploy with a Lambda or custom action to run kubectl if you prefer.

**13. Monitoring (CloudWatch)**

* CodeBuild and CodePipeline automatically push logs to CloudWatch Logs. In the build project, ensure CloudWatch logging is enabled.
* For application logs: if you need pod logs, use kubectl logs or configure Fluent Bit / CloudWatch Container Insights to collect pod logs and metrics.
* To view logs:

# CloudWatch console -> Logs -> /aws/codebuild/<project-name>

# or use aws cli

aws logs get-log-events --log-group-name "/aws/codebuild/<project-name>" --log-stream-name <stream>

## 14. Screenshots & Submission checklist

* Screenshot: kubectl get nodes (cluster running)
* Screenshot: kubectl get deployments / kubectl rollout status
* Screenshot: kubectl get svc brain-tasks-svc showing external IP / hostname
* Screenshot: ECR repository showing pushed images
* Screenshot: CodePipeline pipeline run success
* Screenshot: CloudWatch logs for CodeBuild
* Note the **Load Balancer ARN** found in AWS Console (EC2 > Load Balancers) for the service created by the Kubernetes service. Include ARN in the README.

**15. README template (to include in repo)**

# Brain-Tasks-App — Deployment

## Overview

Short summary of what you did.

## How to run locally

1. npm install

2. npm run build

3. docker build -t brain-tasks-app .

4. docker run -p 3000:3000 brain-tasks-app

## CI/CD Pipeline

- Source: GitHub

- Build: CodeBuild (buildspec.yml) — builds and pushes Docker image to ECR

- Deploy: CodeBuild deploy step (buildspec-deploy.yml) — updates EKS deployment

## Kubernetes

- Cluster: brain-tasks-cluster (EKS)

- Deployment file: deployment.yaml

- Service file: service.yaml (type: Loa

I've created a complete, ready-to-use deployment guide (with sample files and commands) in a canvas document titled **"Brain-Tasks-App — Deployment Guide & Files"** — you can open it now in the panel next to our chat.

The document includes:

* exact Dockerfile, buildspec.yml, buildspec-deploy.yml, deployment.yaml, and service.yaml examples
* step-by-step CLI commands for ECR, EKS (eksctl), kubectl, and Docker
* recommended CodeBuild/CodePipeline setup and a simple approach to deploy to EKS (via CodeBuild)
* troubleshooting tips, README template, and the list of screenshots & deliverables (including where to find the Load Balancer ARN)

Want me to generate the actual files (Dockerfile, buildspecs, k8s manifests, deploy script) and commit them into your repo now? If so, tell me:

* your GitHub repo URL (or confirm you want me to create branch and files locally for you to push), and
* your AWS region and AWS account ID (so I can customize ECR image URIs in the files).

If you'd rather I produce the files without AWS account-specific values, I can do that too. Which option do you prefer?