REACT App DevOps Assignment Documentation

Kubernetes and Application architecture overview.

For deploying a k8s cluster we are using the Kops method

Bootstrap server

- The bootstrap server will create a Kubernetes cluster (Control plane and worker node).
- First, we create a Bootstrap server with AWS cli, kubectl and kops installed on it.
- We also create s3 bucket to store all the configuration and the state of cluster.
- We create IAM roles with policies and attach it with Bootstrap server.

Worker Node

- This worker node is where the docker, docker-compose will be installed and we deploy our multi-container application.
- We have separate folders for frontend and backend with respective Docker Files in it.
- We create **.env** file and **docker-compose.yml** file for multi-container deployment.

Frontend:

- A react application that will serve the user interface.
- This will run in a container and is served using lightweight web server, here we are using Nginx.
- The frontend User Interface will be accessible on http://localhost:3000.
- This depends on backend and will communicate ad fetch data through Restful API's on http://backend:5000.

Backend:

- A Node.js application that handles API requests, business logic and connection with database.
- Here we have Mongoose to interact with mongodb.
- This will be running in the 2nd container and listens on port 5000.

Database:

- A MongoDB instance that will store application data.
- This will be running in the 3rd container and is only accessible by backend.

- Listens on port 27017 and is accessible to backend via **DB=mongodb://mongo:27017/database** environment variable.

Steps to Deploy the Application on Worker-node in Kops using Bootserver

- First, we start the bootstrap server and setup a kubernetes cluster using the Kops.
- Here I have set the execution permissions using **chmod +x kubectl** and **chmod +x kops** commands on both kops and kubectl and moved them to the **/usr/local/bin** folder.
- Creating a s3 bucket with name **kopsbuckwt** with enabled versioning.
- Then I have created a local domain name in route53 to manage DNS records.
- Then we create the cluster configuration in the required availability zone which includes first setting the environment variables (Kops cluster name and Kops's state store)

  export KOPS_CLUSTER_NAME= dptcluster.k8s.local
  export KOPS_STATE_STORE=s3://kopsbuckwt

  Kops create cluster --zone = us-west-1b --name=dptcluster.k8s.local

  Deploying and validating the cluster
  Kops update cluster --name=dptcluster.k8s.local --yes
  Kops validate cluster --name=dptcluster.k8s.local --yes
  Kops rolling-update cluster --name=dptcluster.k8s.local --yes

  Kops get clusters (Verifying if the cluster is created and ready to use)

  Once the cluster is ready there will be a control plane and a worker node created in a VPC along with subnets, security groups, Autoscaling groups and loadbalancer.

- We login into the worker node through ssh client and install docker, docker compose and enable the docker engine using command
  Sudo service docker start

  Verifying by running a hello-world image
  Sudo docker run hello-world

- Then creating two different directories for frontend and backend to store their respective Docker Files
  mkdir frontend
  vi Dockerfile
  mkdir backend
  vi Dockerfile

- Then we create a **.env** file in the root directory that will contain database environment variable
  DB=mongodb://mongo:27017/database this will be used as reference in docker-compose file that will create the connection between backend and database as mentioned in **conn.js** file
- We create a **docker-compose.yml** file to define the multi-container creation and their dependencies.
- Then we use docker compose commands to deploy the containers and run the application.
  docker compose build
  docker compose up –d

  Then we can verfiying once the containers are deployed and running using **docker ps** command.
- Before accessing the application, we need to allow through custom TCP in the inbound rules of worker node's security group.
  port 22(for ssh)
  port 3000(for frontend)
  port 5000(for backend)
  port 27017(for database)
- FInally we can access the application using the http://<worker-node_public-ip>:3000 address.

Troubleshooting

- Checking Logs:
  - Frontend: docker logs react-frontend
  - Backend: docker logs react-backend
  - Mongodb: docker logs mongo-database
- Connection Issues:
  - Ensuring mongo is the name mentioned in docker compose network.
  - Verifying environment variables in docker-compose.yml
- Network issues:
  - Verifying work-node security group allows access to ports 3000, 5000, 27017.