Capstone project

GitHub link: <https://github.com/SuhaasNandeesh/capstone-project>

As part of the Capstone, you will be doing a POC around the setup of a Kubernetes cluster on cloud and managing applications on Kubernetes. The POC will involve the following goals:

Basic VPC setup for a Kubernetes cluster on AWS

EKS cluster setup using eksctl

Stateless & Stateful workload on Kubernetes

Metrics system for Kubernetes

Autoscaling nodes & workload on Kubernetes

Task 0: Environment Setup

Install the following tools in your local linux/mac machine: aws-cli, terraform, eksctl, aws-iam-authenticator, kubectl, helm, docker, ab (apache benchmark). You will need these tools later in the tasks.

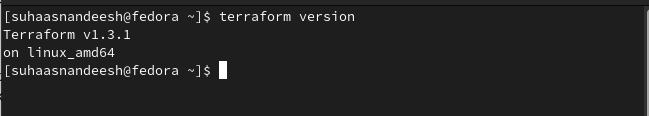
aws-cli

Installation link: <https://docs.aws.amazon.com/cli/latest/userguide/getting-started-install.html>



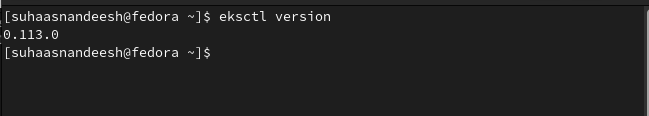
terraform

Installation link: <https://learn.hashicorp.com/tutorials/terraform/install-cli>



eksctl

Installation link: <https://docs.aws.amazon.com/eks/latest/userguide/eksctl.html>

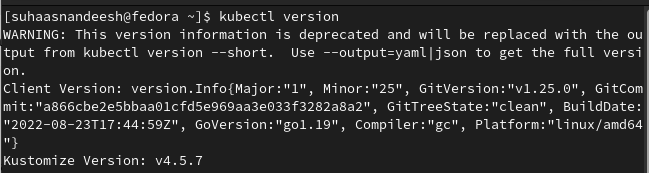


aws-iam-authenticator

If running the AWS CLI version 1.16.156 or later, then we don't need to install the authenticator. Instead, we can use the aws eks get-token command.

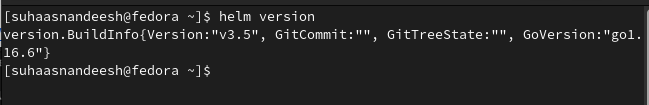
kubectl

Installation link: <https://kubernetes.io/docs/tasks/tools/>



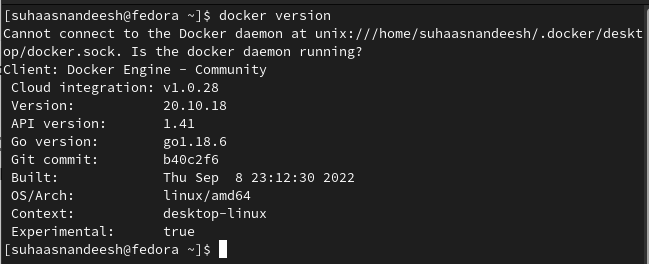
helm

Installation link: <https://helm.sh/docs/intro/install/>



docker

Installation link: <https://docs.docker.com/desktop/install/linux-install/>



apache bench

Installation link: <https://linuxconfig.org/how-to-install-apache-bench-on-redhat-8>



Task 1: Setup EKS Cluster

As part of this task, you are supposed to set up an EKS cluster using a custom VPC. By default, EKS can create all VPC resources but we want to use our terraform created VPC resources instead. This cluster will be used in further tasks. This task can be broadly categorised into three parts:

VPC Setup preferably using Terraform

EKS cluster setup using eksctl: Here are some important points to note about this subtask:

The eksctl configuration file is provided as a stub file with a few placeholders.

You are supposed to thoroughly understand the YAML file, provided in the stub code by using the references provided in the ‘References’ section for this task.

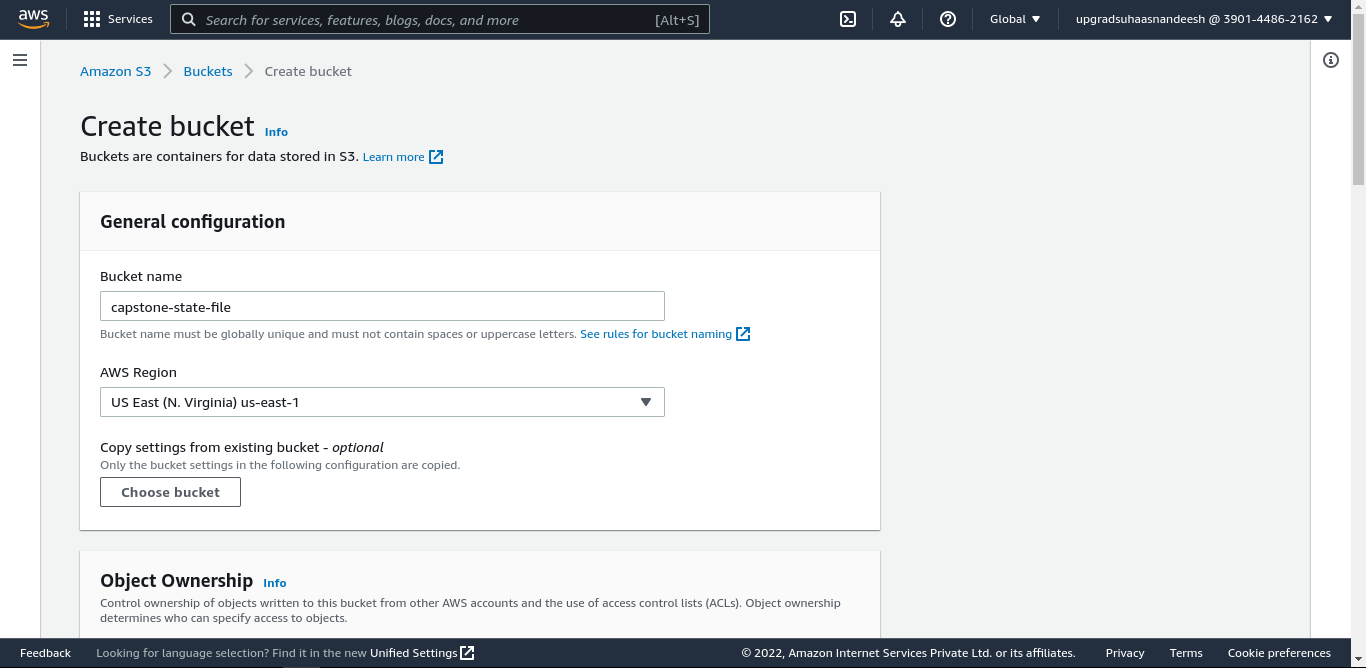
There are a few “TODOs” left for you in the stub files. You need to fill those placeholders and launch the EKS cluster for successful completion of this task.

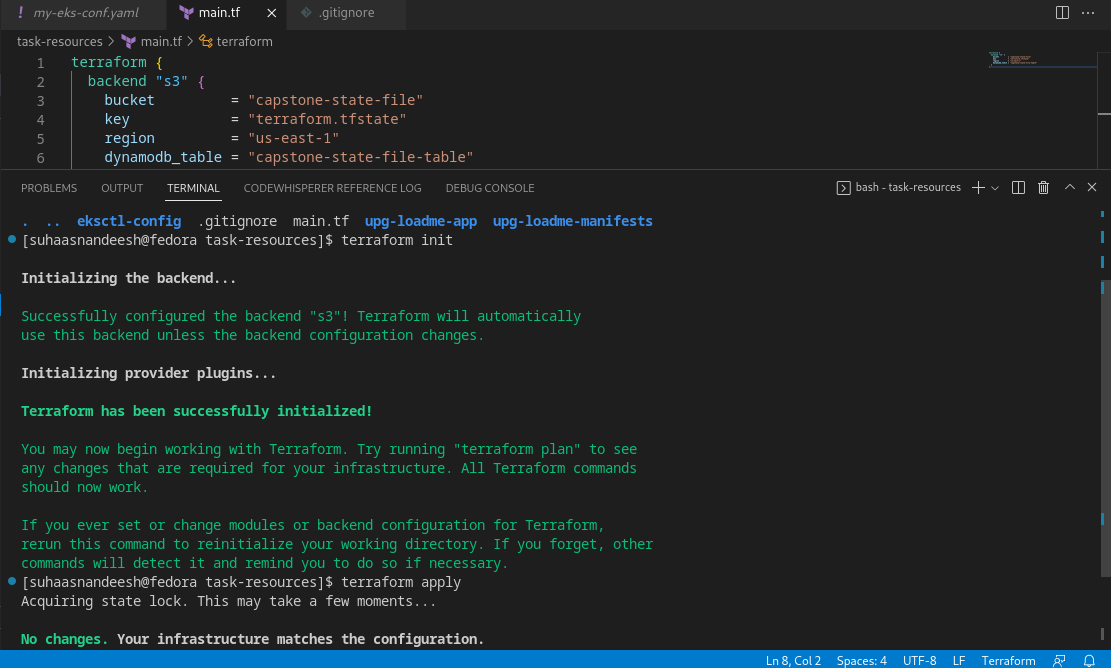
Installation of some necessary Add-ons on the EKS cluster. The resources for this sub-part are provided in the ‘References’ section of this task. You are expected to install them by following the official documentation. Do not worry about the advanced/internal details of these add-ons for the completion of the task.

—

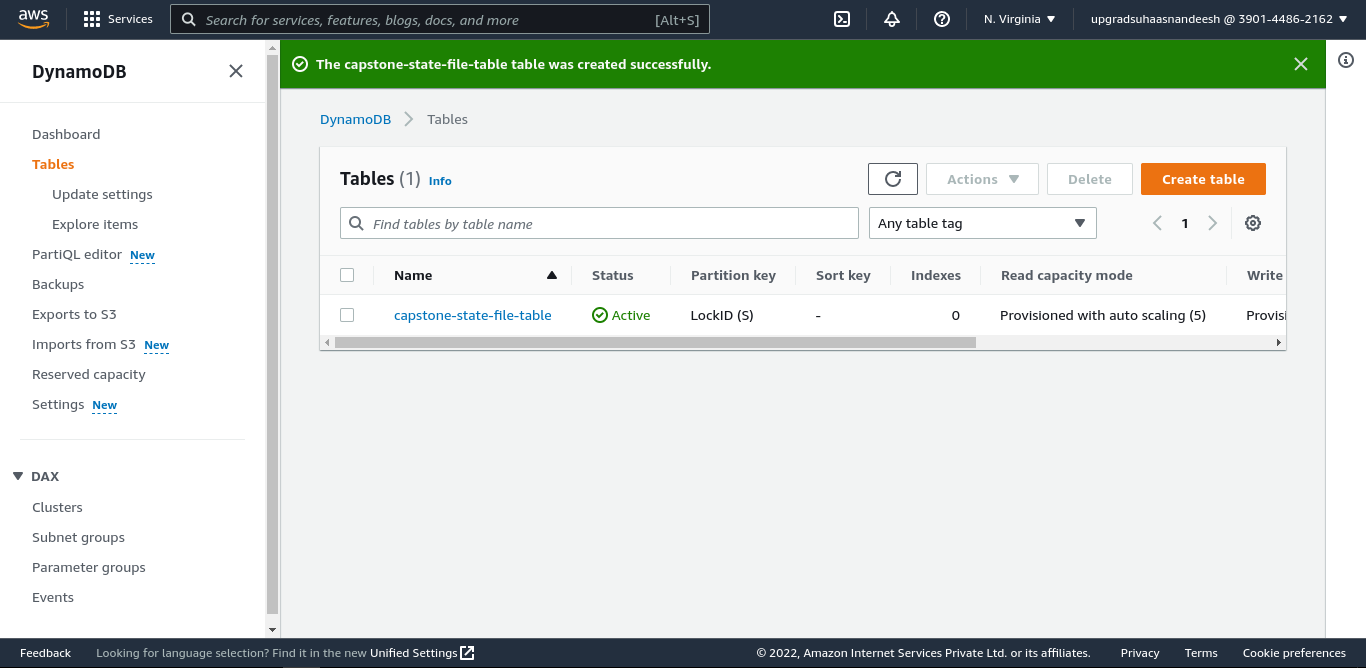
aws –version to check aws-cli version installed in the local system (which was installed as part of Task 0).

S3 bucket is created from the AWS console in order to store the state files for terraform scripts.





And also to avoid multiple writes on the state of terraform files, a dynamoDB table is created to lock the state files.



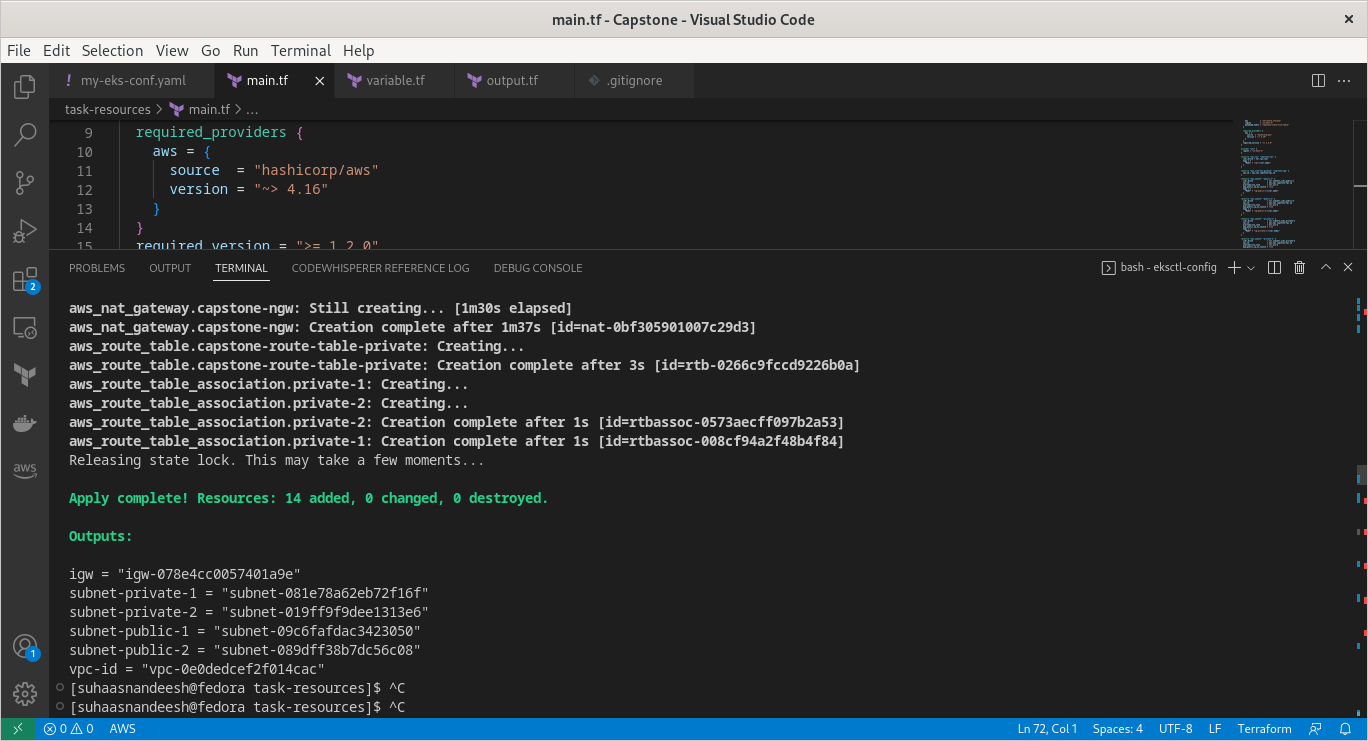
Refer to the main.tf file to create vpc, public and private subnets, NAT gateway and route tables.

variable.tf file is used to create and store values for frequently used parameters like CIDR block.

output.tf file is used to display the required values after the infrastructure is created in AWS.

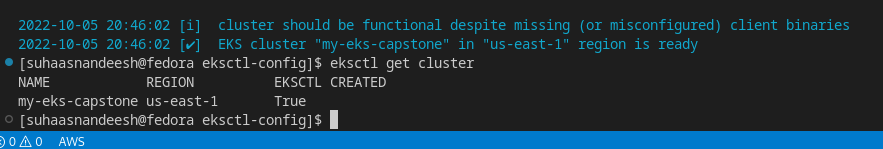
Use command -> terraform apply

terraform apply will check the local and remote state of configurations and create resources in AWS.



my-eks-conf.yaml file given as part of stub code is updated with terraform generated vpc and subnet ids.

Command to create cluster: eksctl create cluster -f my-eks-conf.yaml

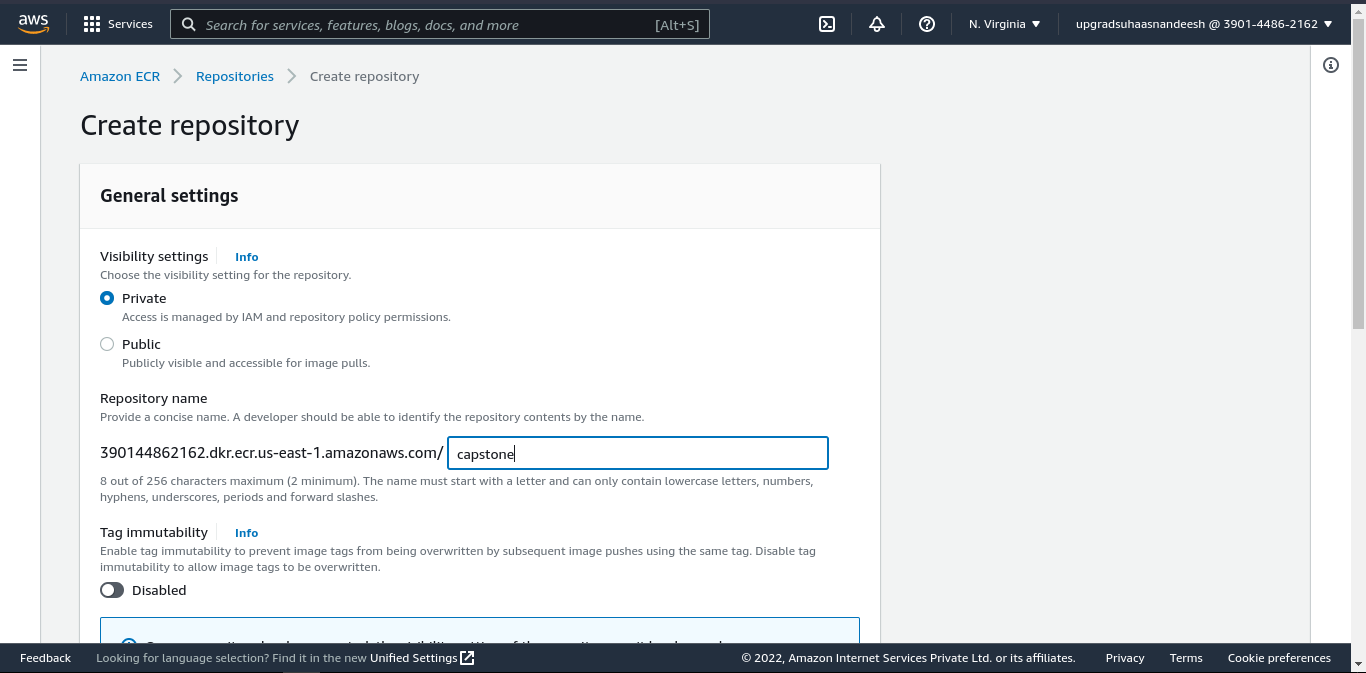


Task 2: Deployment of sample application

After successfully setting up the EKS cluster, the next step involves deployment of a sample node application on the cluster.

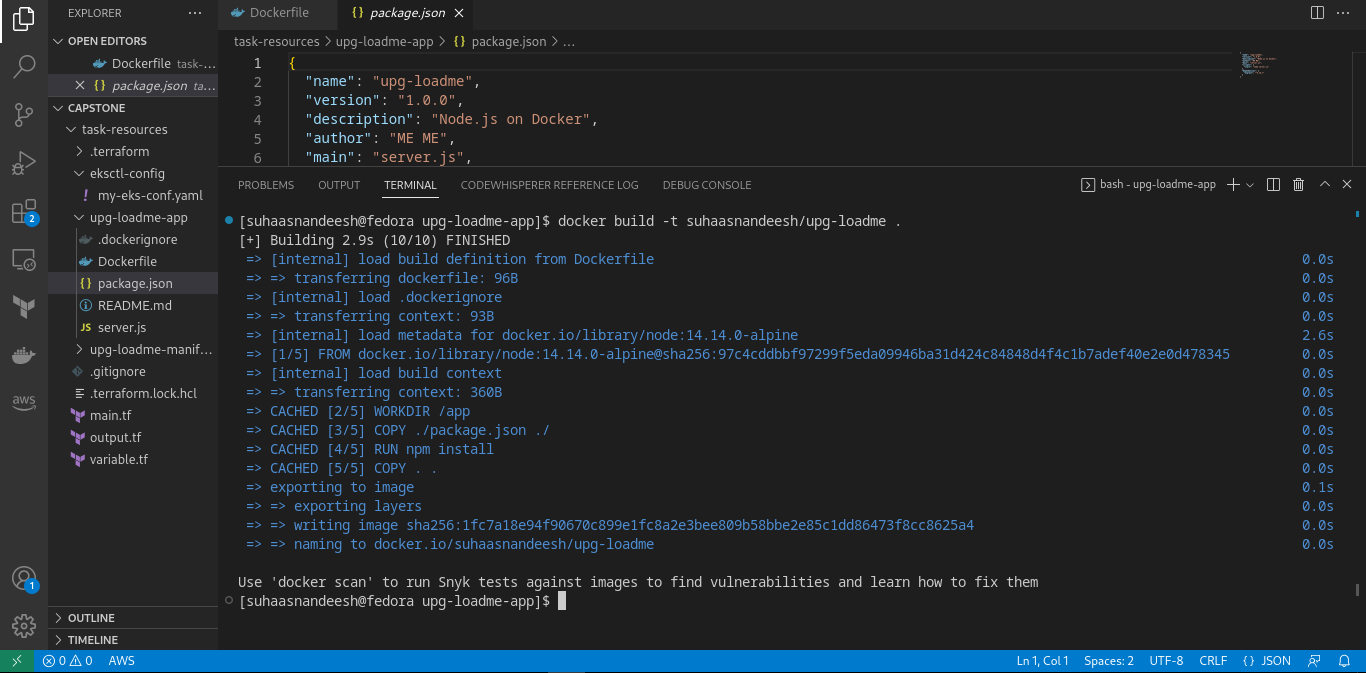
—

ECR repository is create to save docker images for the node application



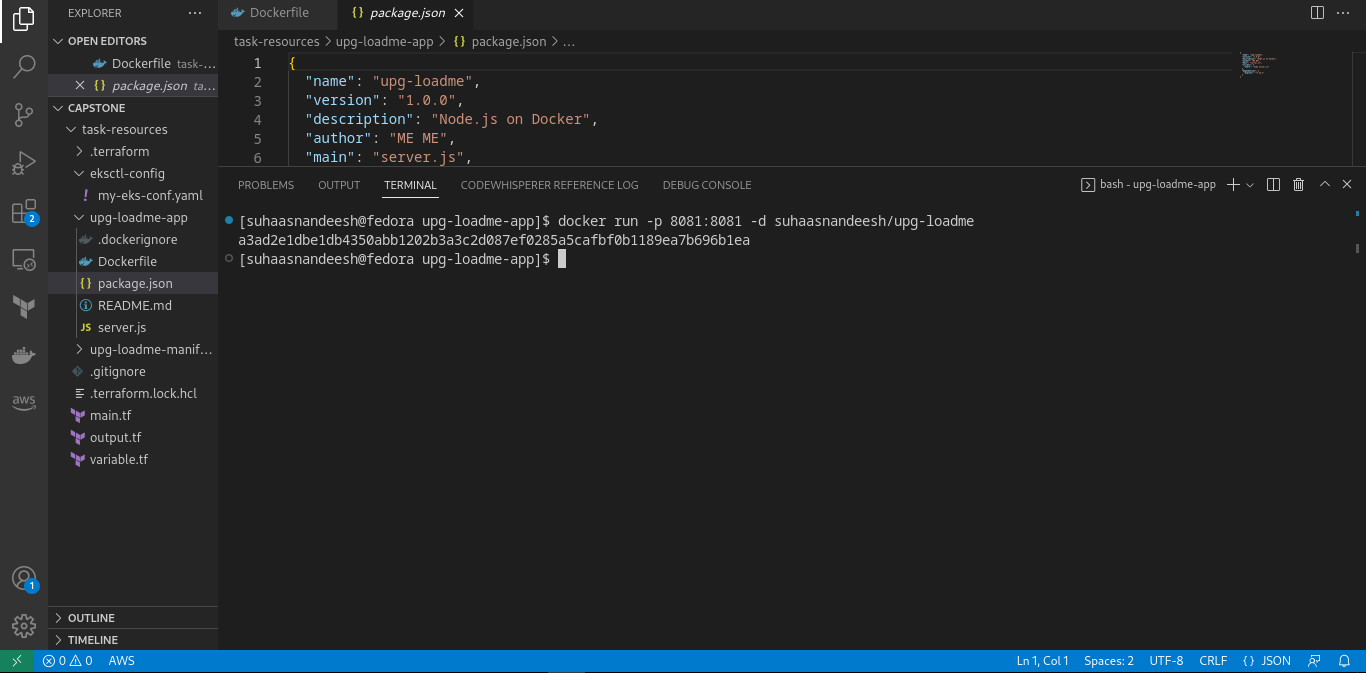
Refer Dockerfile for node application

Command to build docker image -> docker build -t suhaasnandeesh/upg-loadme .



Command to list docker images -> docker images

Command to run the image -> docker run -p 8081:8081 -d <image id> (exposing localhost 8081 port to container 8081 port)





Task 3: Deploy Redis server on Kubernetes

So far, you've got your hands dirty with setting up the EKS cluster and deploying a sample node application. The app is an example of a stateless app as there is no state associated with any specific instance of the app. If any of the application pods gets killed the new pod spawned can substitute the previous pod without caring about the state of the dead pod. However, some applications may have a state associated with them. For example, databases such as MongoDb, Cassandra, MySQL, etc. For such workloads, Kubernetes has the concept of Statefulsets.

This task will essentially give you an experience of deploying a stateful application/service using statefulset and attaching a disk for storing state (i.e, db file).

—

Refer files under ‘redis’ folder for redis related deployment, configmap, statefulset files.

In case of redis deployment failing with ‘Unable to connect to the server: getting credentials: decoding stdout: no kind "ExecCredential" is registered for version "client.authentication.k8s.io/v1alpha1’ error, update aws cli version (it should be >2.7)

In order to update aws cli use below commands,

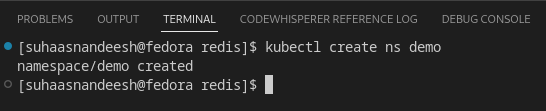
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86\_64.zip" -o "awscliv2.zip"

unzip awscliv2.zip

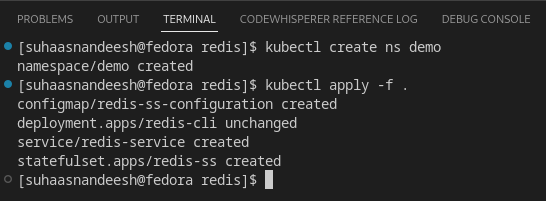
sudo ./aws/install

Use command ‘kubectl exec -it redis-ss-0 -c redis -n demo -- /bin/bash’ to exec into redis-cli deployment.

demo namespace created.



kubectl apply command for all the files required for kubernetes cluster

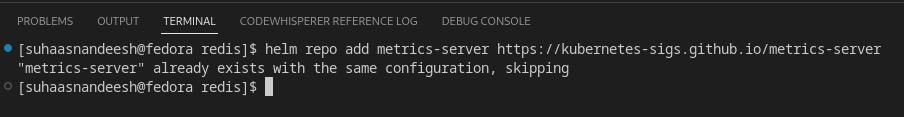


Task 4: Test auto scaling of the application.

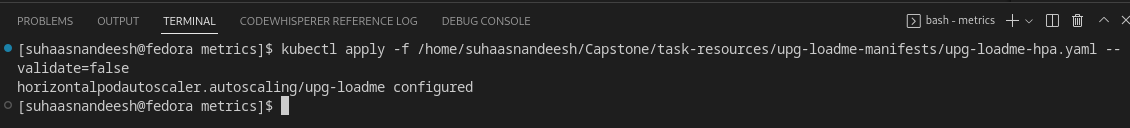
After having deployed the upg-loadme application, it’s important to ensure that your application is able to scale well automatically in case of a surge in usage. Kubernetes has the concept of Horizontal Pod Autoscaler for handling scale up/down of deployment & statefulsets. As part of this task, you will first ensure that an HPA is configured for scaling the deployed application and later test the scaling actions by simulating requests to the application using Apache Benchmark (ab) tool.

—

In order to work, HPA needs a metrics server available in the cluster to scrape required metrics, such as CPU and memory utilisation.



HPA configuration for upg-loadme app



Installation of prometheus using helm

