**EKS-TASK**

**Amazon Elastic Container Service for Kubernetes (EKS) is a cloud-based container management service that natively integrates with Kubernetes to deploy applications.**

Kubernetes on AWS without needing to stand up or maintain your own Kubernetes control plane. Kubernetes is an open-source system for automating the deployment, scaling, and management of containerized applications.

Amazon EKS runs Kubernetes control plane instances across multiple Availability Zones to ensure high availability. Amazon EKS automatically detects and replaces unhealthy control plane instances, and it provides automated version upgrades and patching for them.

Amazon EKS is also integrated with many AWS services to provide scalability and security for your applications, including the following:

**Amazon ECR for container images**

**Elastic Load Balancing for load distribution**

**IAM for authentication**

**Amazon VPC for isolation**

EKS runs up-to-date versions of the open-source Kubernetes software, so you can use all the existing plugins and tooling from the Kubernetes community. Applications running on Amazon EKS are fully compatible with applications running on any standard Kubernetes environment, whether running in on-premises data centers or public clouds. This means that you can easily migrate any standard Kubernetes application to Amazon EKS without any code modification required.

**HOW DOES AMAZON EKS WORKS?**


    How Amazon EKS works
   

Getting started with Amazon EKS is easy:

1. First, create an Amazon EKS cluster in the AWS Management Console or with the AWS CLI or one of the AWS SDKs.
2. Then, launch worker nodes that register with the Amazon EKS cluster. We provide you with an AWS CloudFormation template that automatically configures your nodes.
3. When your cluster is ready, you can configure your favorite Kubernetes tools (such as **kubectl**) to communicate with your cluster.
4. Deploy and manage applications on your Amazon EKS cluster the same way that you would with any other Kubernetes environment.

**Install the AWS CLI**

To install the latest version of the AWS CLI, choose the tab with the name of the operating system that you'd like to install the AWS CLI on.

* **Configure your AWS CLI credentials**

Both eksctl and the AWS CLI require that you have AWS credentials configured in your environment. The **aws configure** command is the fastest way to set up your AWS CLI installation for general use.

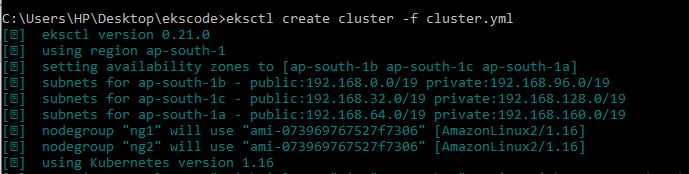
* **Install eksctl**

To install 0.23.0 version or later of the **eksctl** command line utility, choose the tab with the name of the operating system that you'd like to install **eksctl** on.

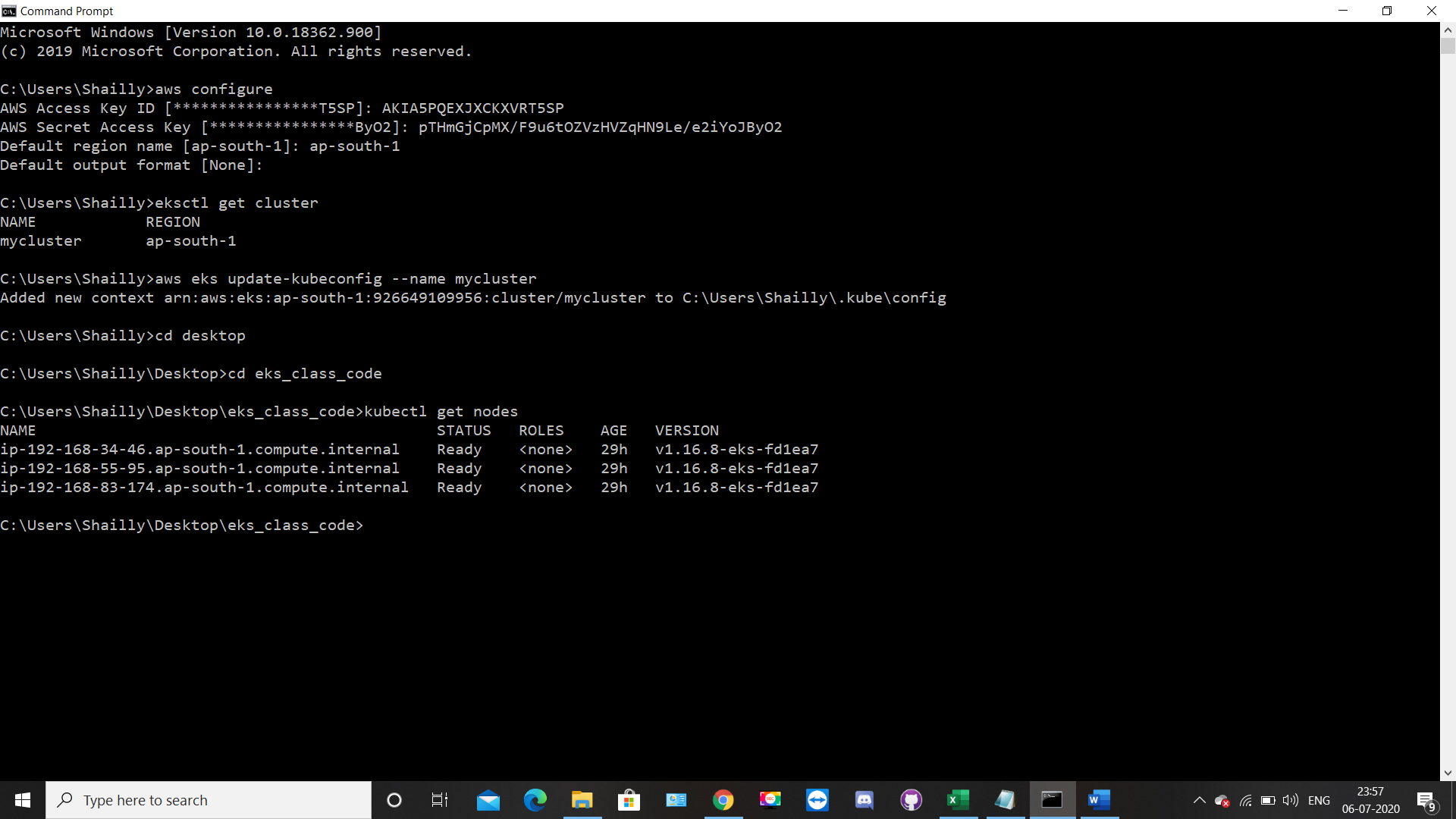
* **Install and configure** **kubectl**

Kubernetes uses the **kubectl** command-line utility for communicating with the cluster API server.

* **Creation of EKS Cluster** :-
* Now, first login to your AWS account by giving the access key and secret key.
* After login to aws , now we have to create the cluster ,for creating the cluster we have to use the cluster.yml script.
* Command for creating the cluster :- eksctl create cluster -f cluster.yml.
* This will take some time to setup all the configuration.



* NOW , check your cluster is created or not by giving the command :-eksctl get cluster,,it shows your cluster.
* Now we will connect the master and for that we need to go to the config fileof .kube and update the file using below commands:-



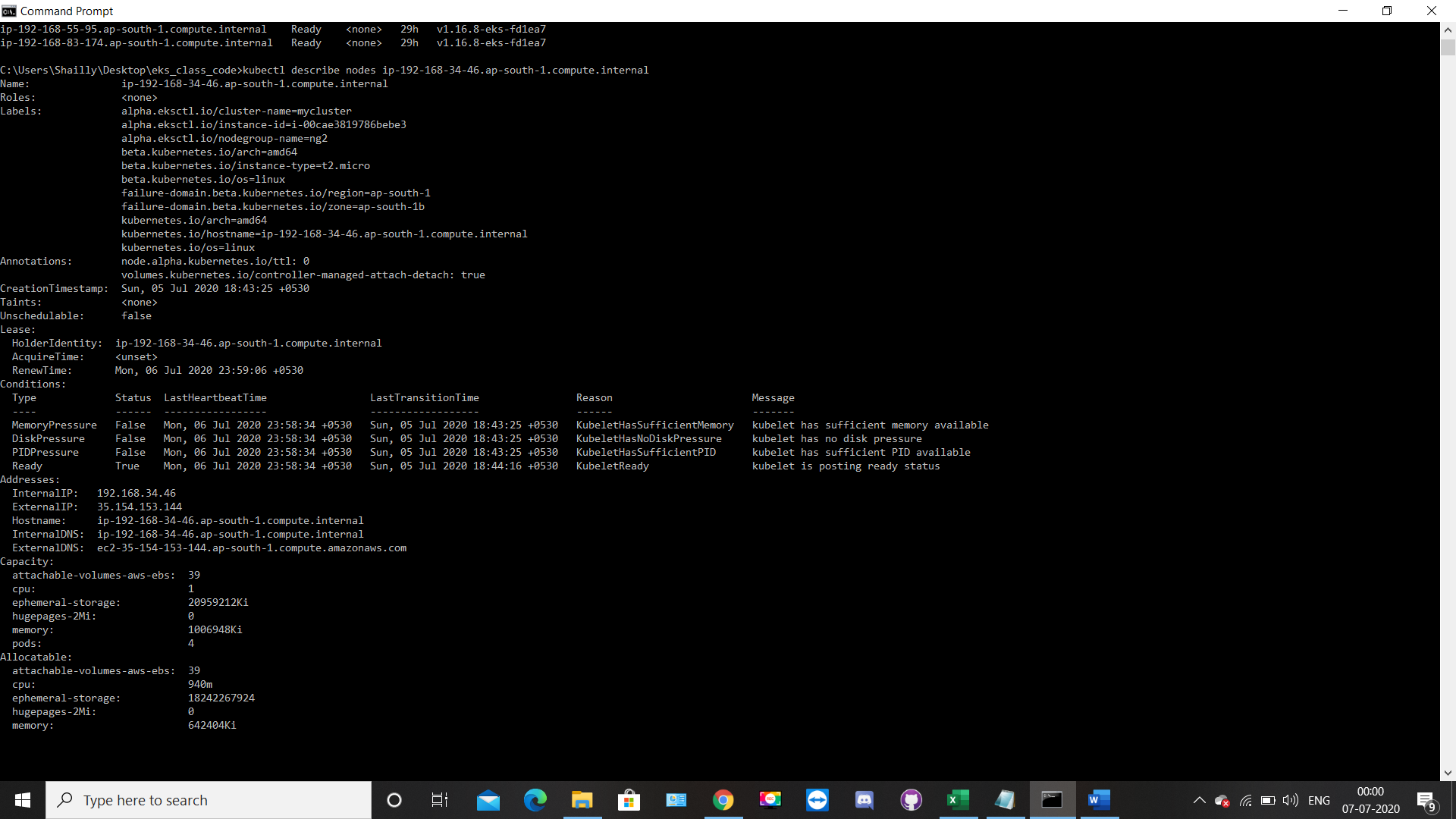
## **eksctl program internally creates cloud formation stack for cluster automatically and with the help of cloud formation we can do auto provisioning.**

NodeGroup – It tells how many similar nodes are there in a nodegroup that you are going to launch. If you want to use kubectl command in EKS cluster, then you need to update the kubeconfig file inside your baseOS.

* Now we check the nodes and describe the nodes by typing the following commands:-

**# kubectl get nodes :- This command is used to see all running pods.**

#kubectl describe node ip-192.168.34.46.ap-south-1.compute.internal



* Now we will create a namespace and set it as a default:-

We also have to update namespace and set the context to this namespace.

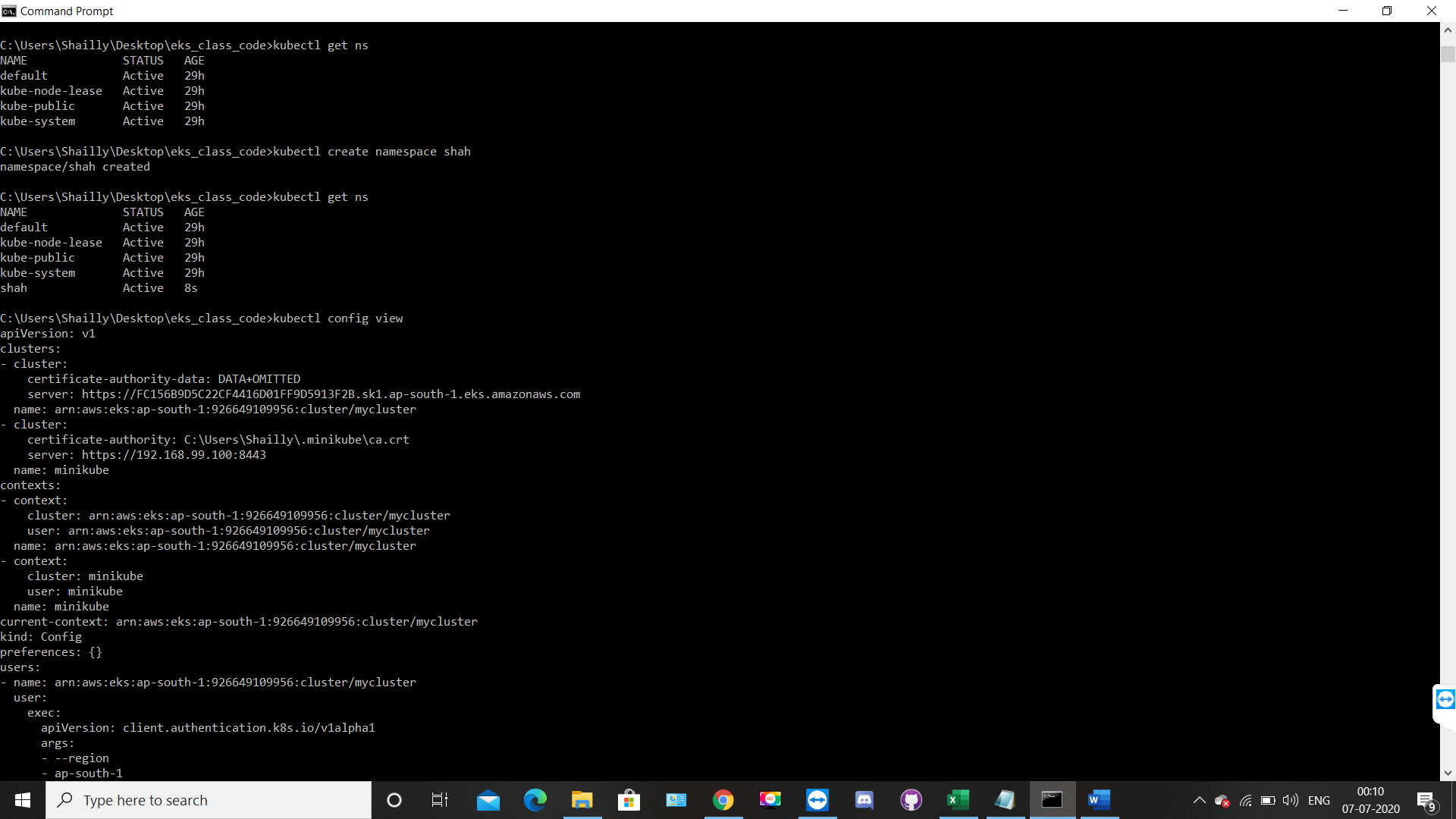
Because bydefault kubectl create everything in default namespace.

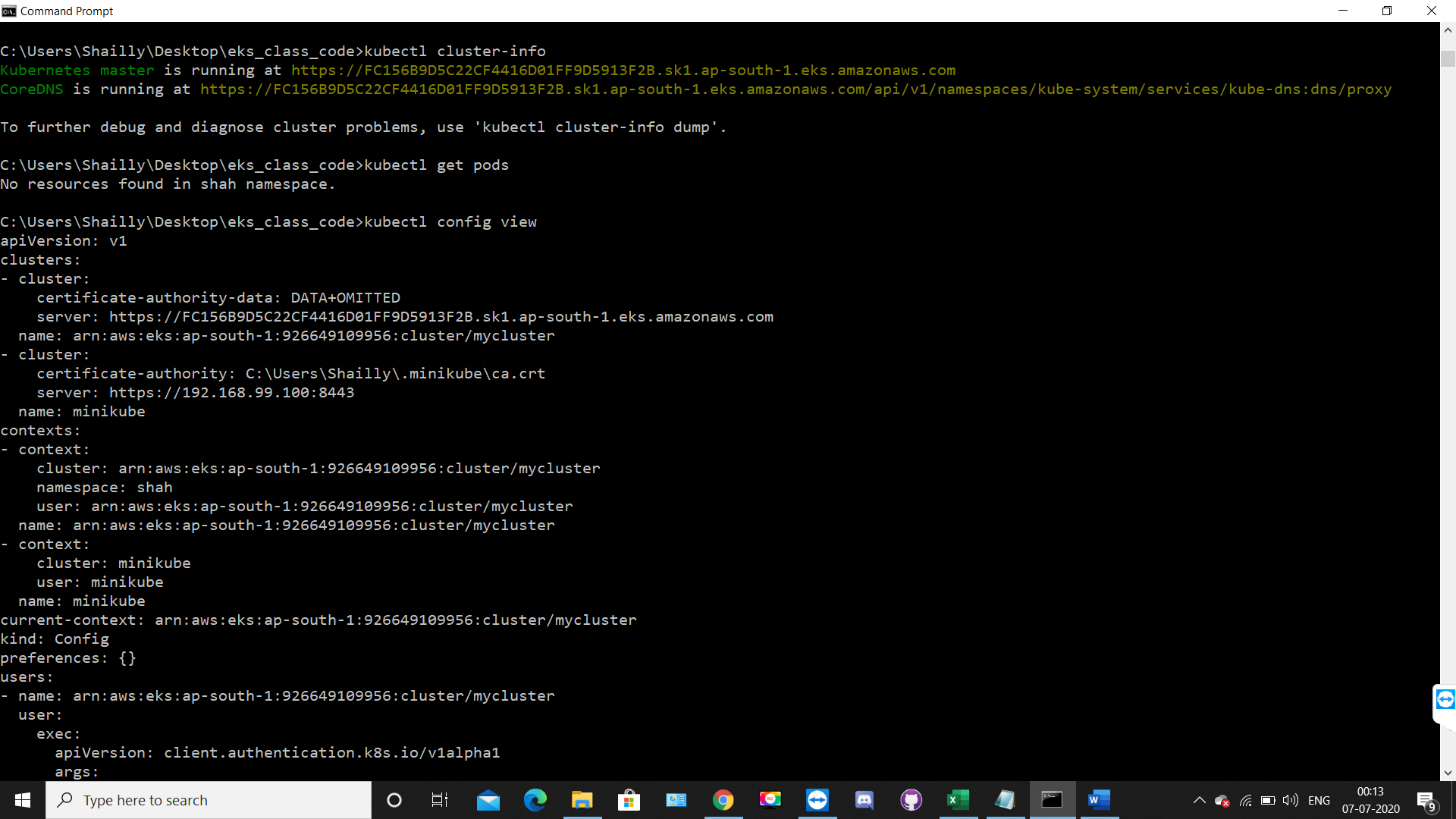
**# kubectl create namespace shah**

**# kubectl config set-context --current --namespace=shah**

**#kubectl get pods**

**#kubectl clueter-info**





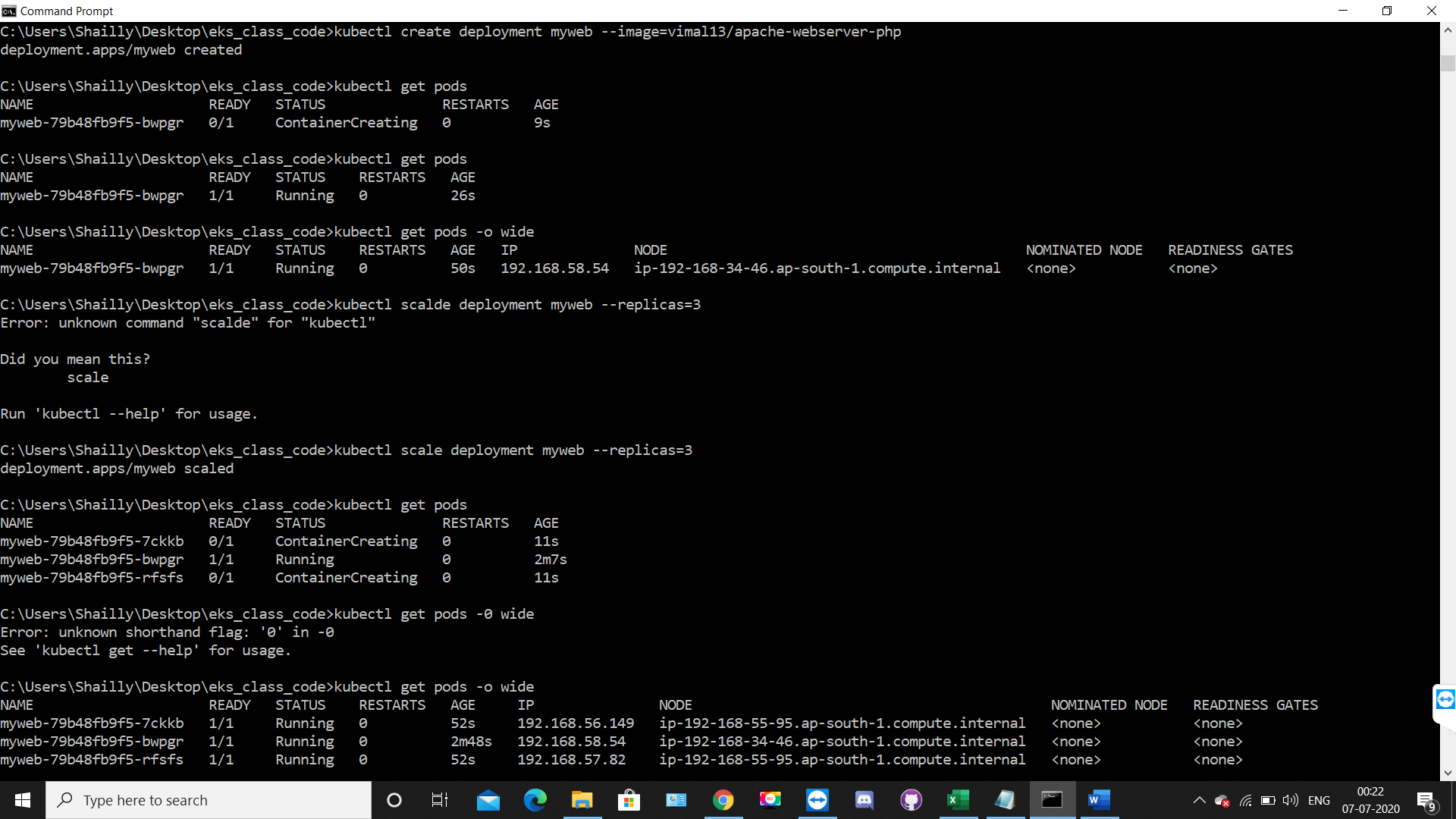
* Now I am going to create one deployment,scale it and also expose my pod to the outside world with service type load balancer at port no 80

Creation of deployment inside the node :-

# kubectl create deployment myweb --image=vimal13/apache-webserver-php

# kubectl get pods

#kubectl get pods -o wide



* Creation of replica in order to cope up with fault-tolerance.

# kubectl scale deployment myweb --replicas=3

#kubectl get pods

#kubectl get pods -o wide

* You can’t connect to the POD internally the can connect to each other inside the cluster expose the pods. We can expose pods by 3 different ways.

- Cluster IP

- NodePort

- LoadBalancer

Here we use LoadBalancer , to expose, so that outside world can access it.

Elastic Load Balancing automatically distributes incoming application traffic across multiple targets, such as Amazon o wide o the POD/containers from the outside world but the can connect to each other inside the cluster expose the pods.

We can expose pods by 3 different ways. LoadBalancer to expose, so that outside world can access automatically distributes incoming application traffic across multiple targets,

such as Amazon EC2 instances, the outside world but they can connect to each other inside the cluster.

So we need to expose the pods. We can expose pods by 3 different ways.

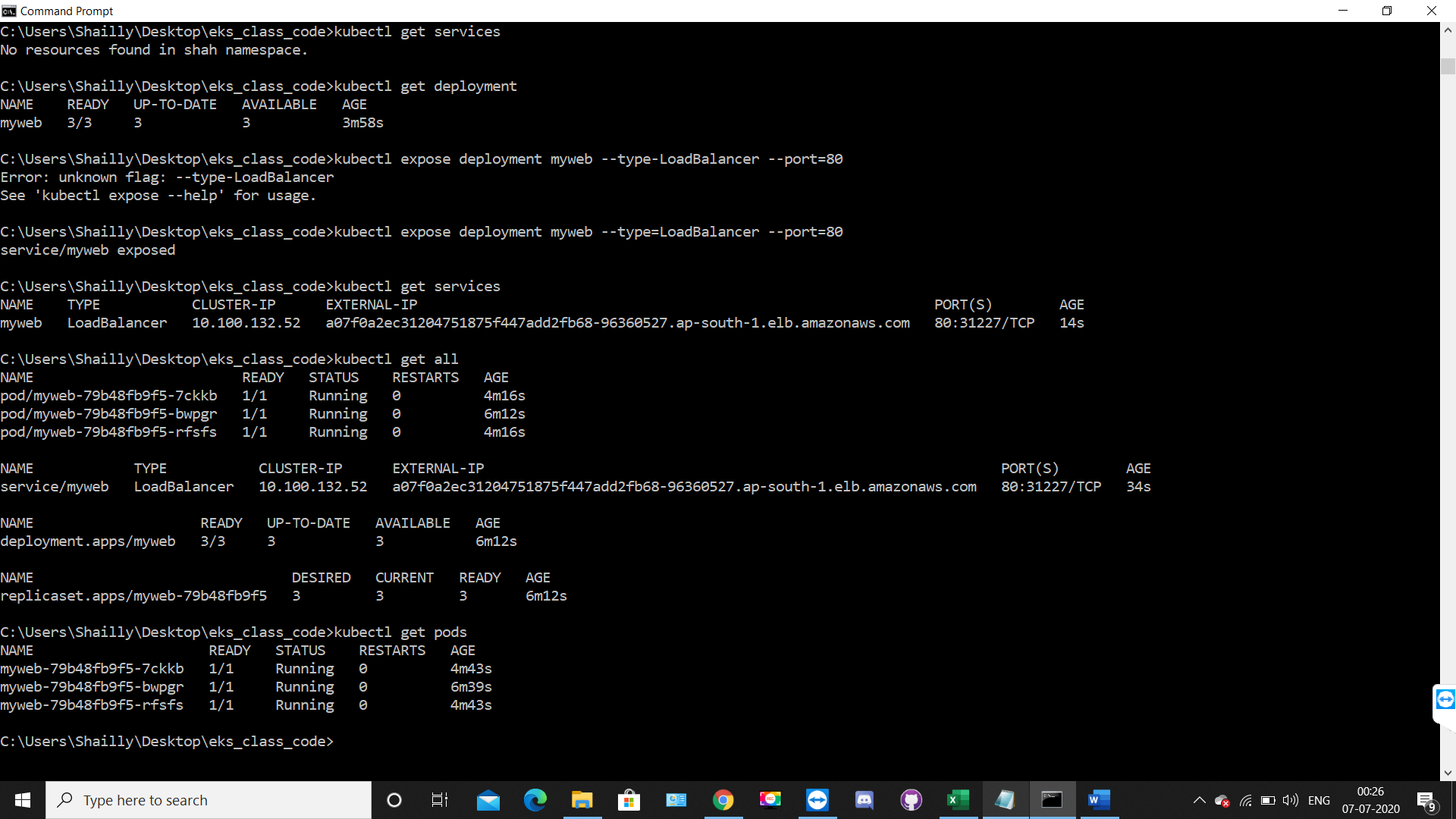
to expose, so that outside world can access automatically distributes incoming application instances, containers, IP addresses, and Lambda functions.

It can handle the varying load of your application traffic in a single Availability Zone or across multiple Availability Zones.

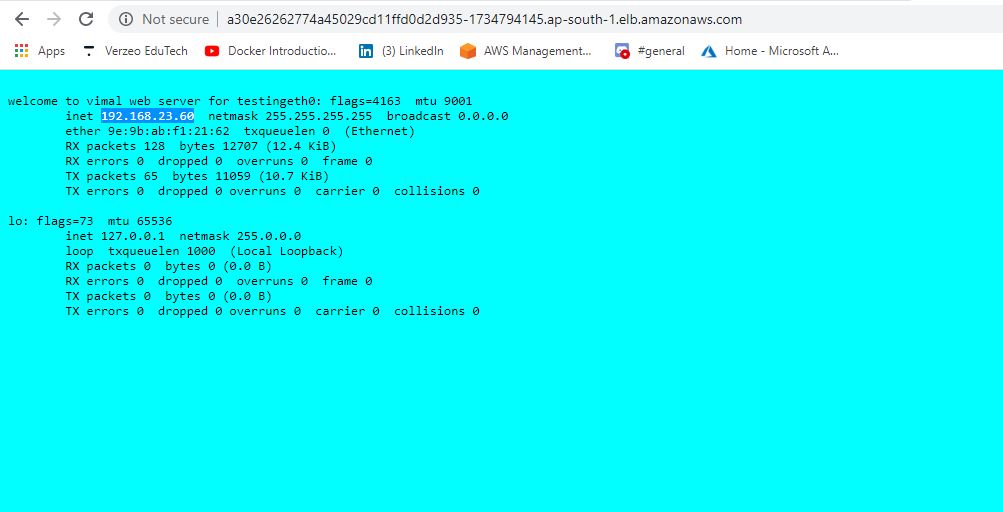
# kubectl expose deployment myweb --type=LoadBalancer --port=80

Refresh the load balancer page on aws page, there it automatically create one load balancer.

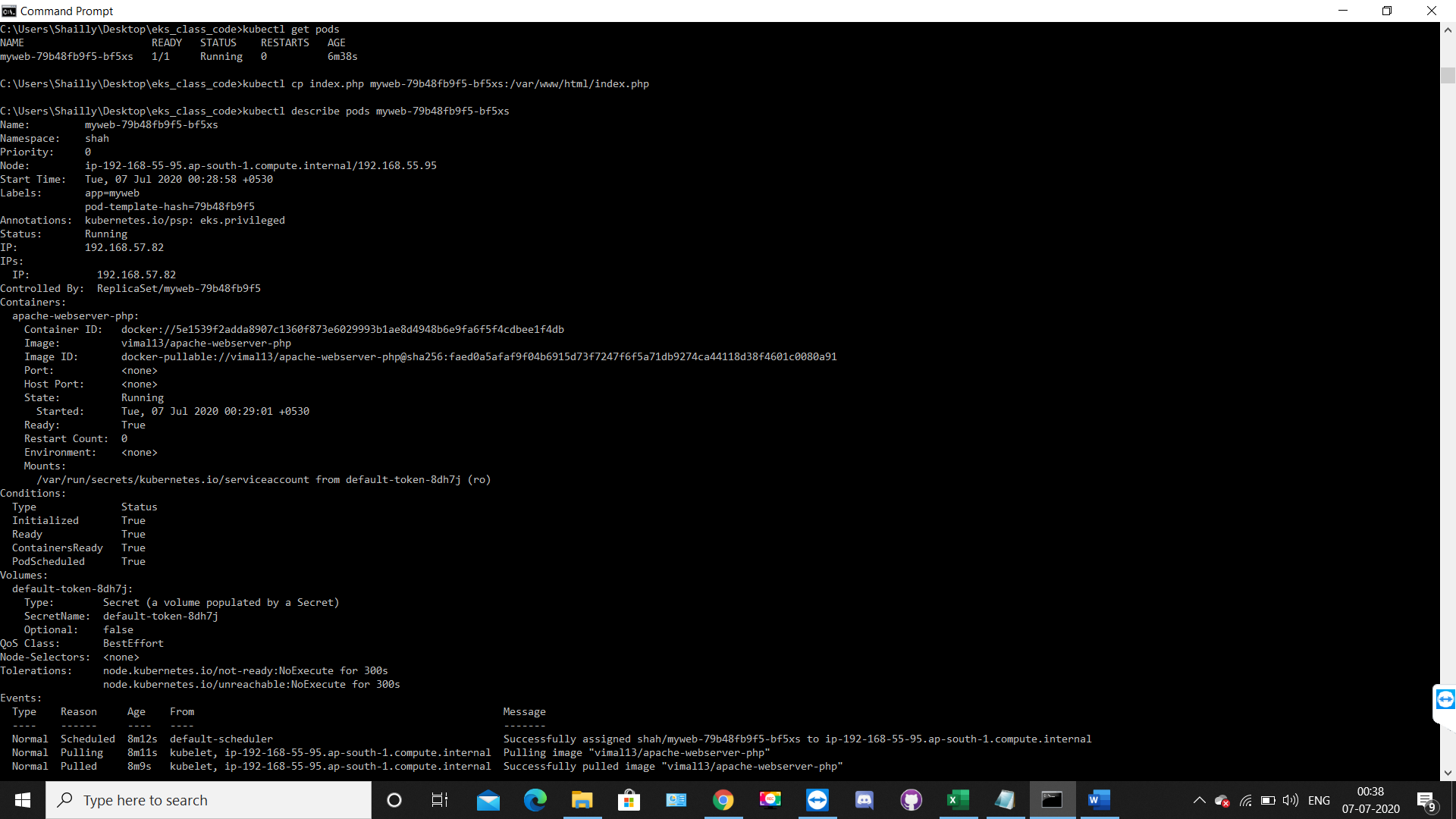
# kubectl describe svc/myweb



Now we will see the working of load balancer here it is showing IP 192.168.23.60 but whenever we will reload the page the IP will get change







* **Creation of PVC :-**

By default storage folder inside a pod is temporary or ephemeral in nature.

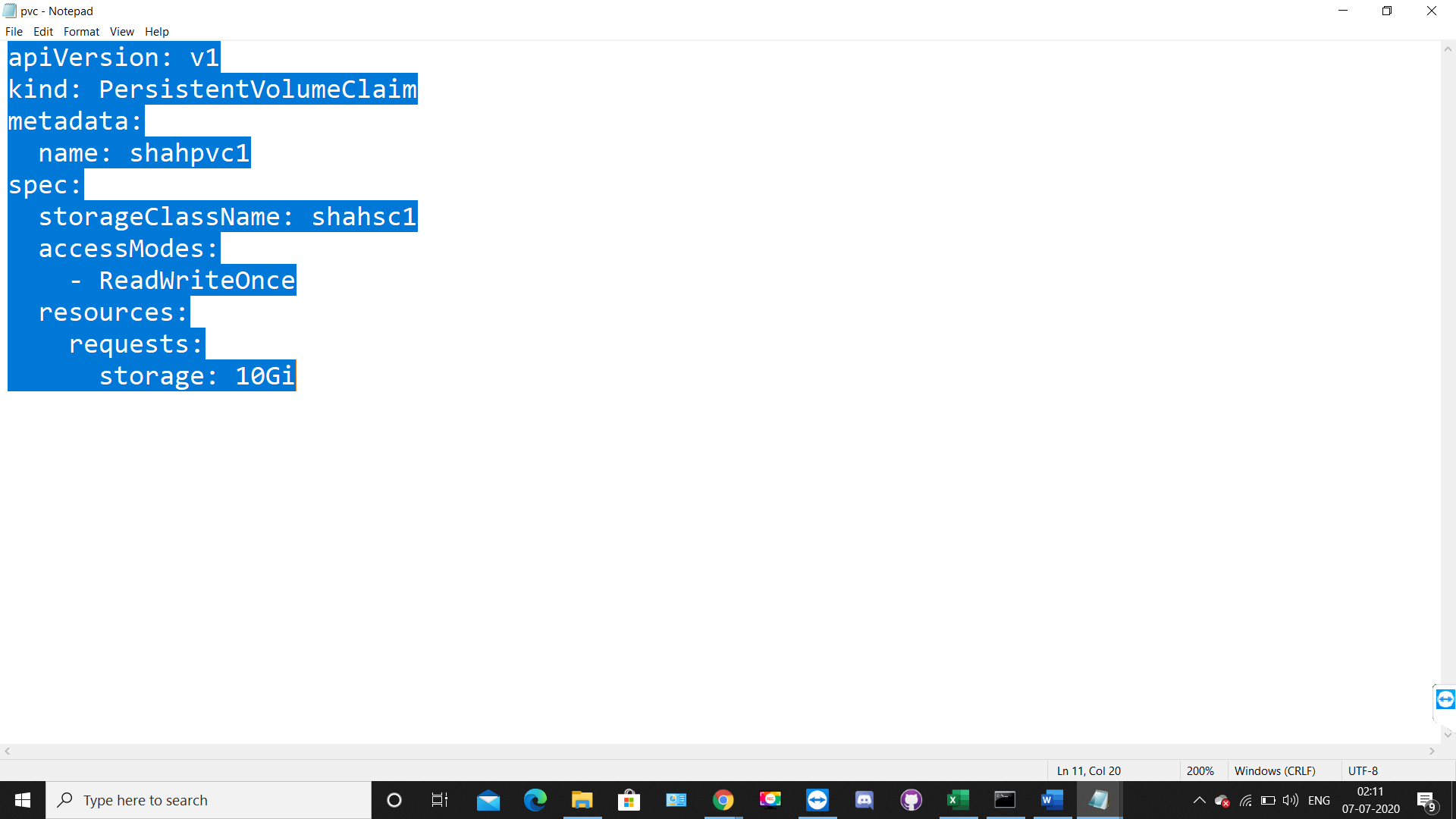
Which means if anyhow your pod get deleted or fails due to any reason, then all data stored in that pod will not persistent.

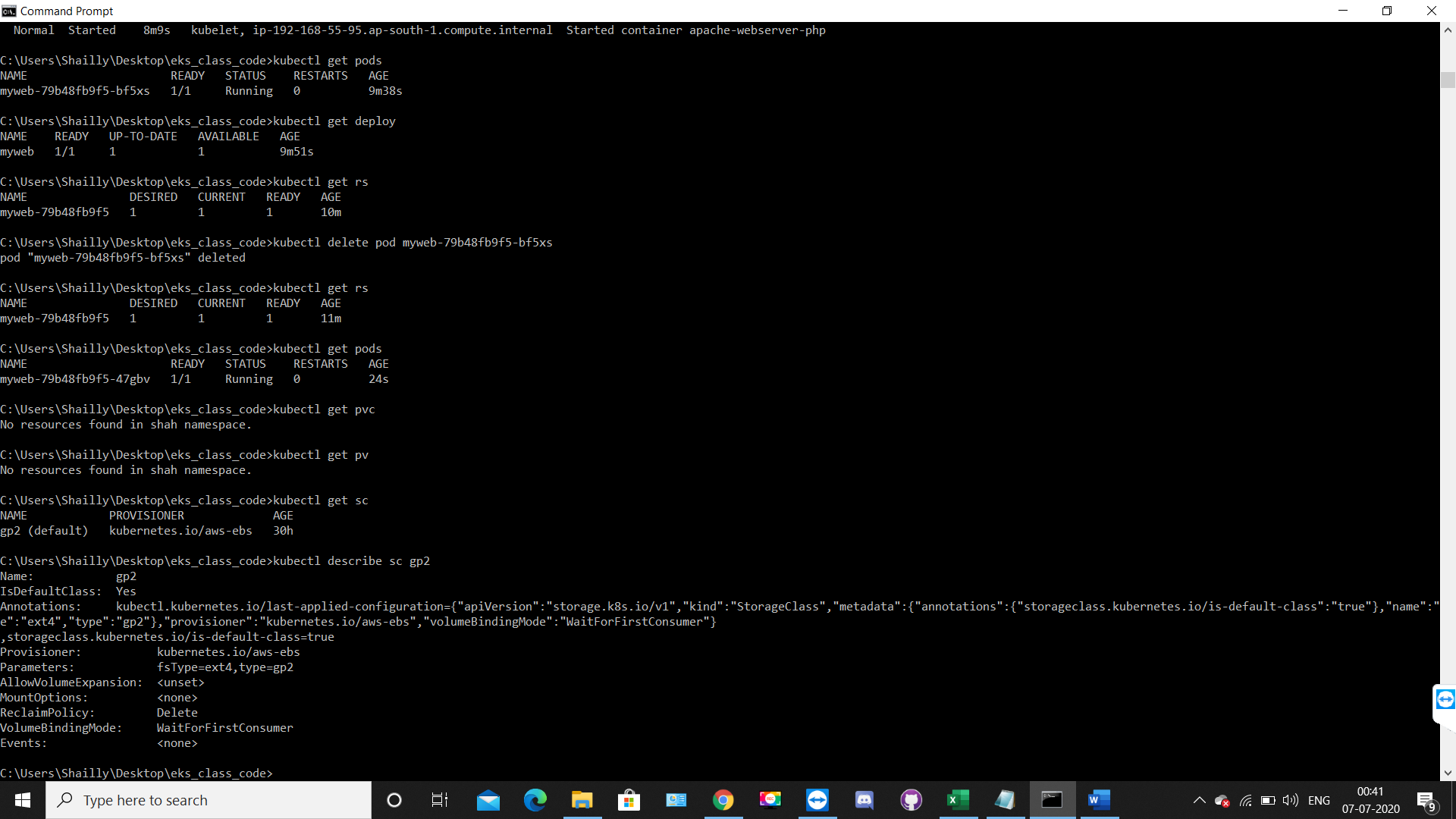
To make you data persistent you need to attach some hard disk & mount that folder inside it.

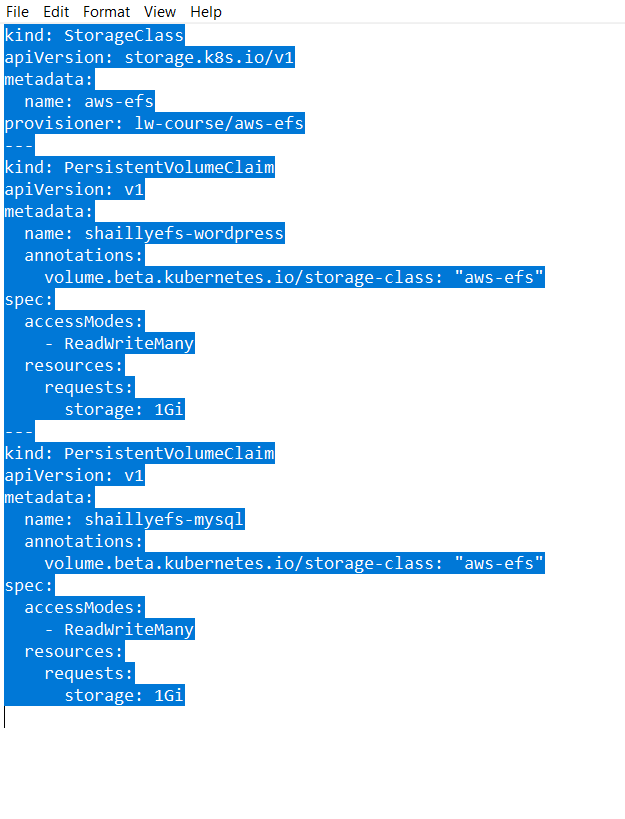
So, if your pod get deleted or fails somehow in this case also your data will be persistent.

Here the role of pvc (PersistentVolumeClaim) comes in play.

Here Is our pvc.yml file:-

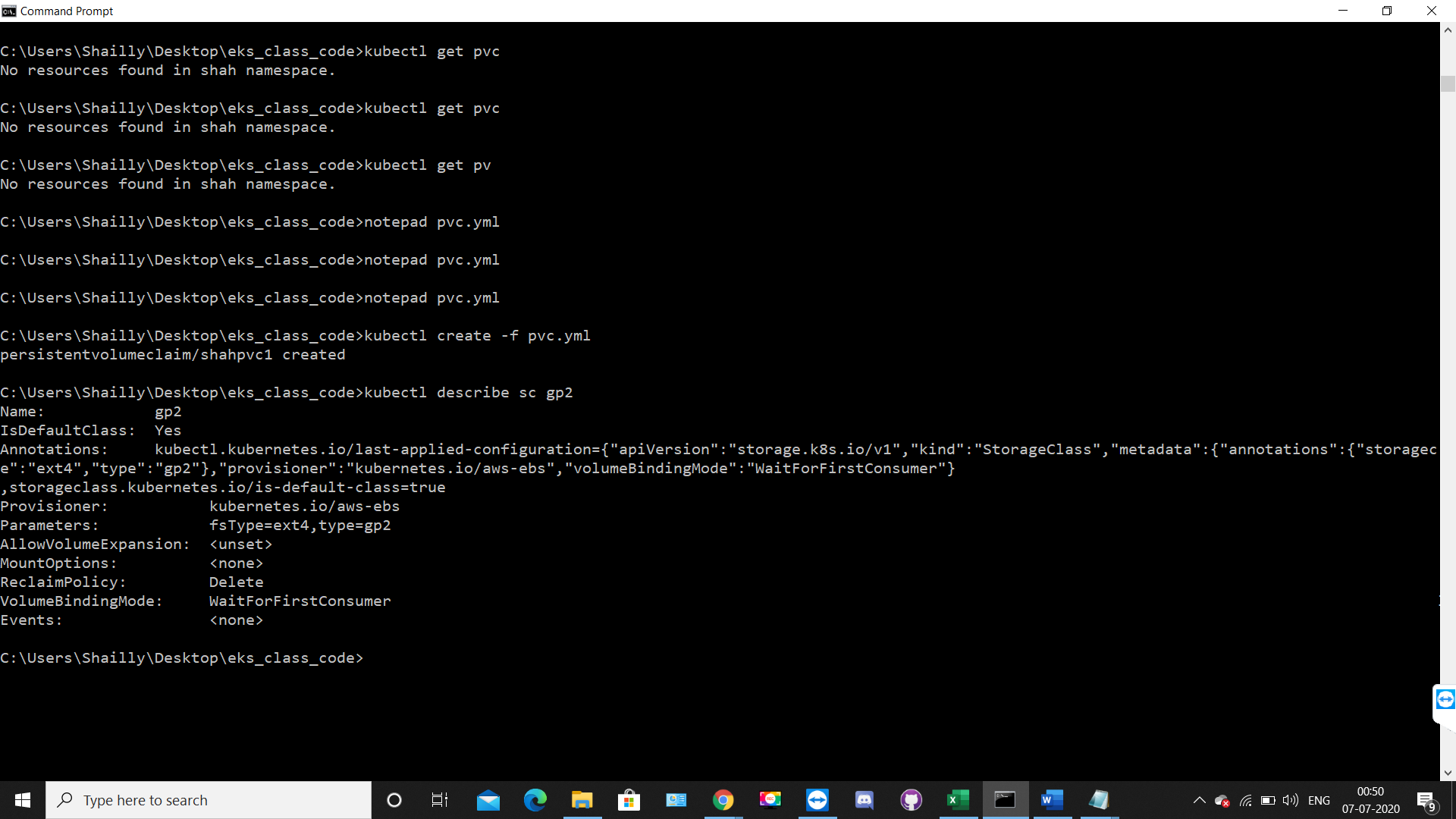


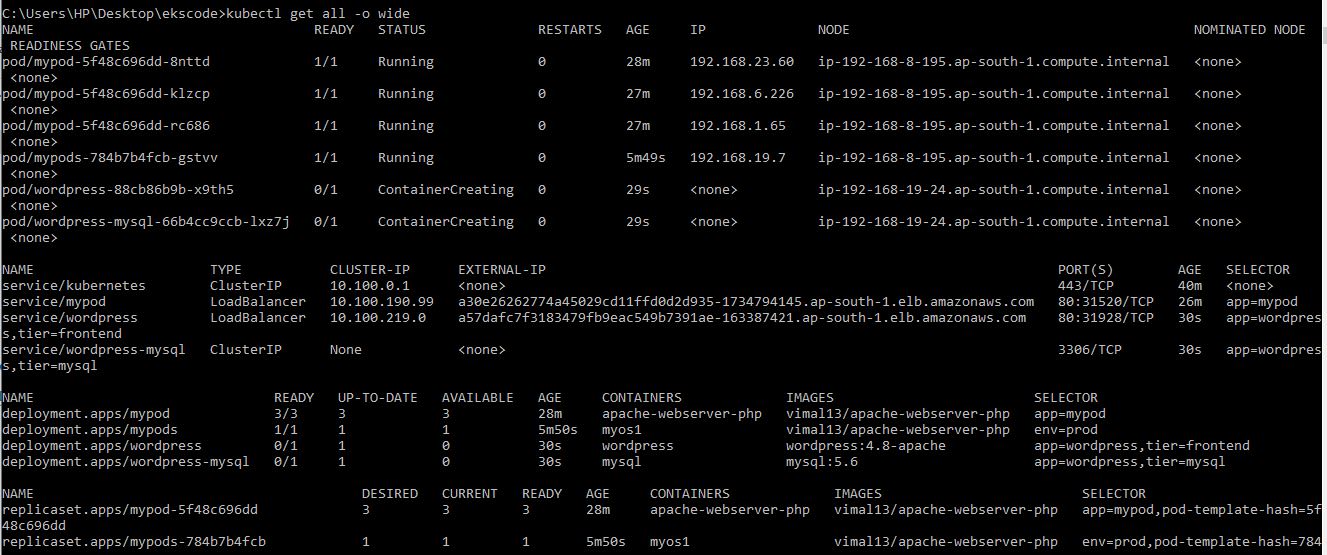




#kubectl create -f pvc.yml

#kubectl describe sc gp2

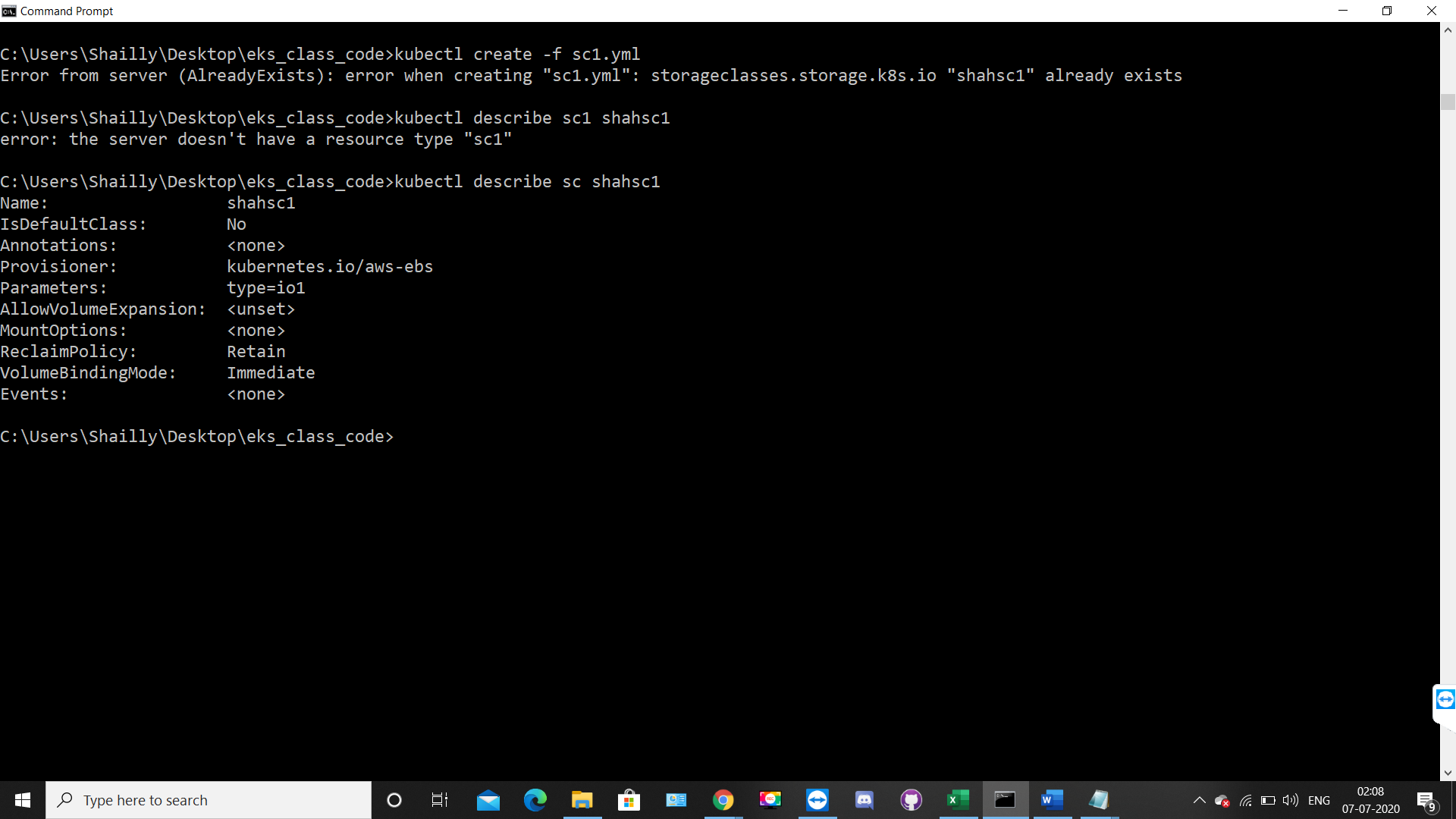




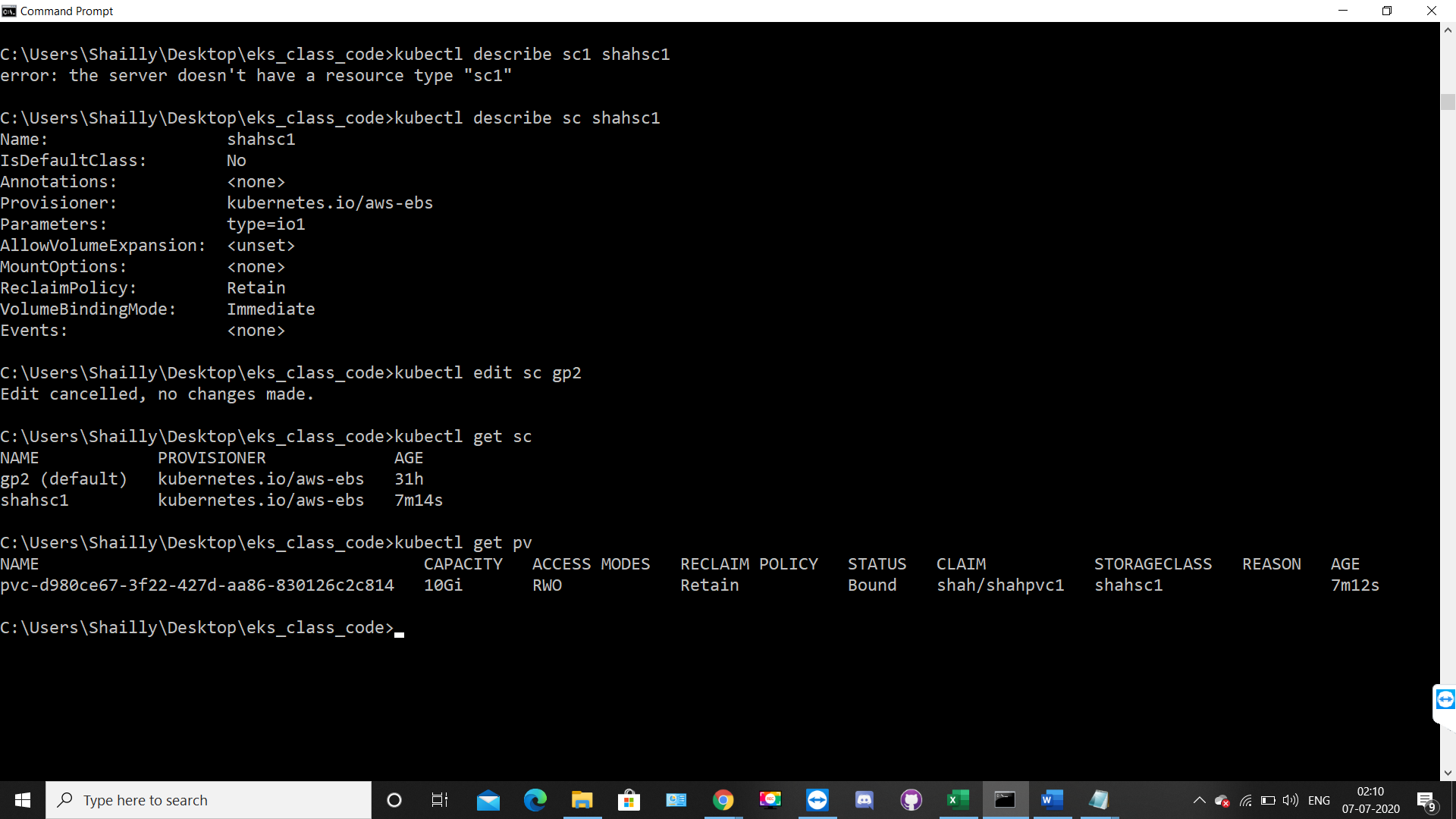
* **Creation of our own storage class**

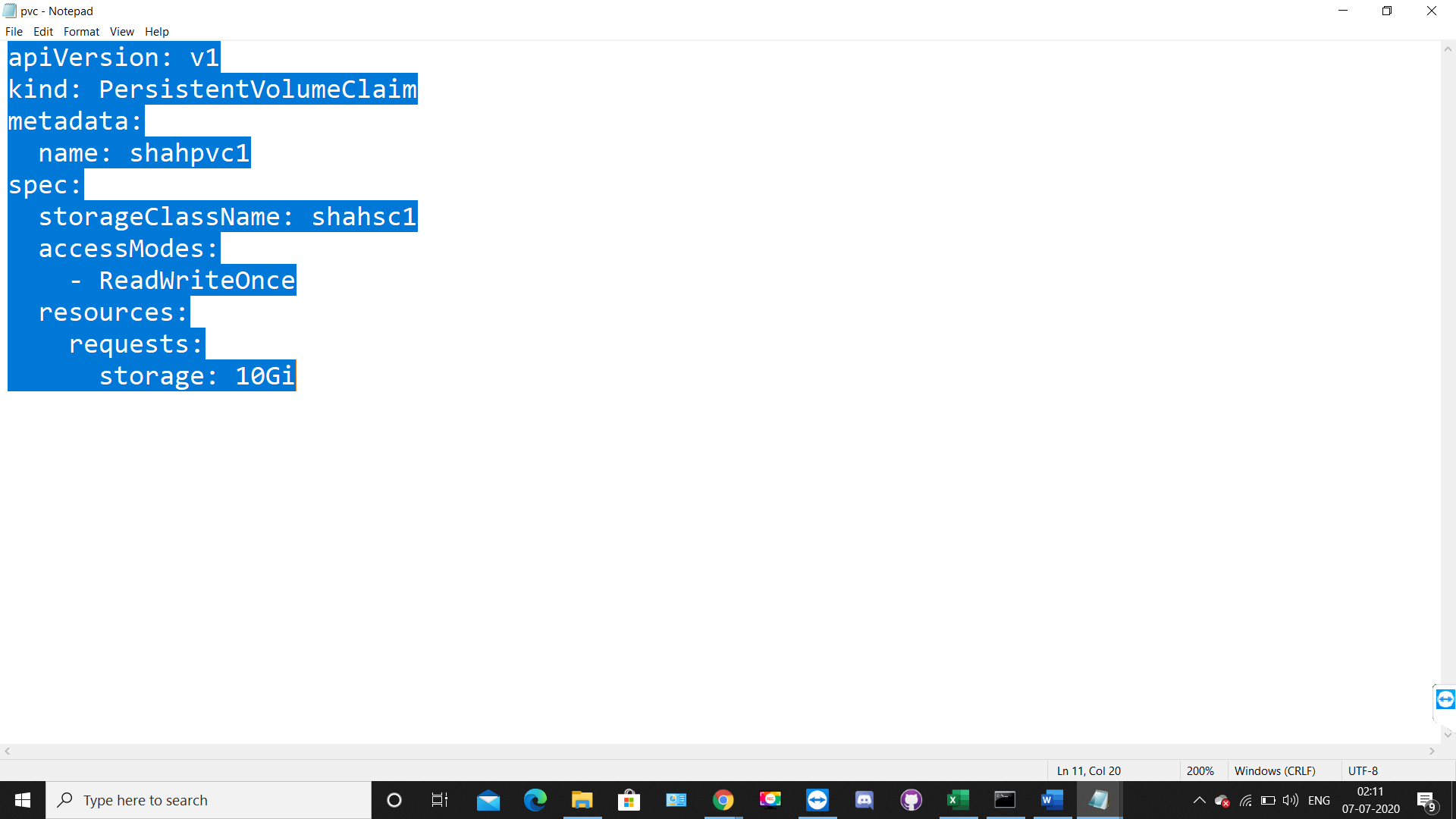
**#notepad sc1.yml**

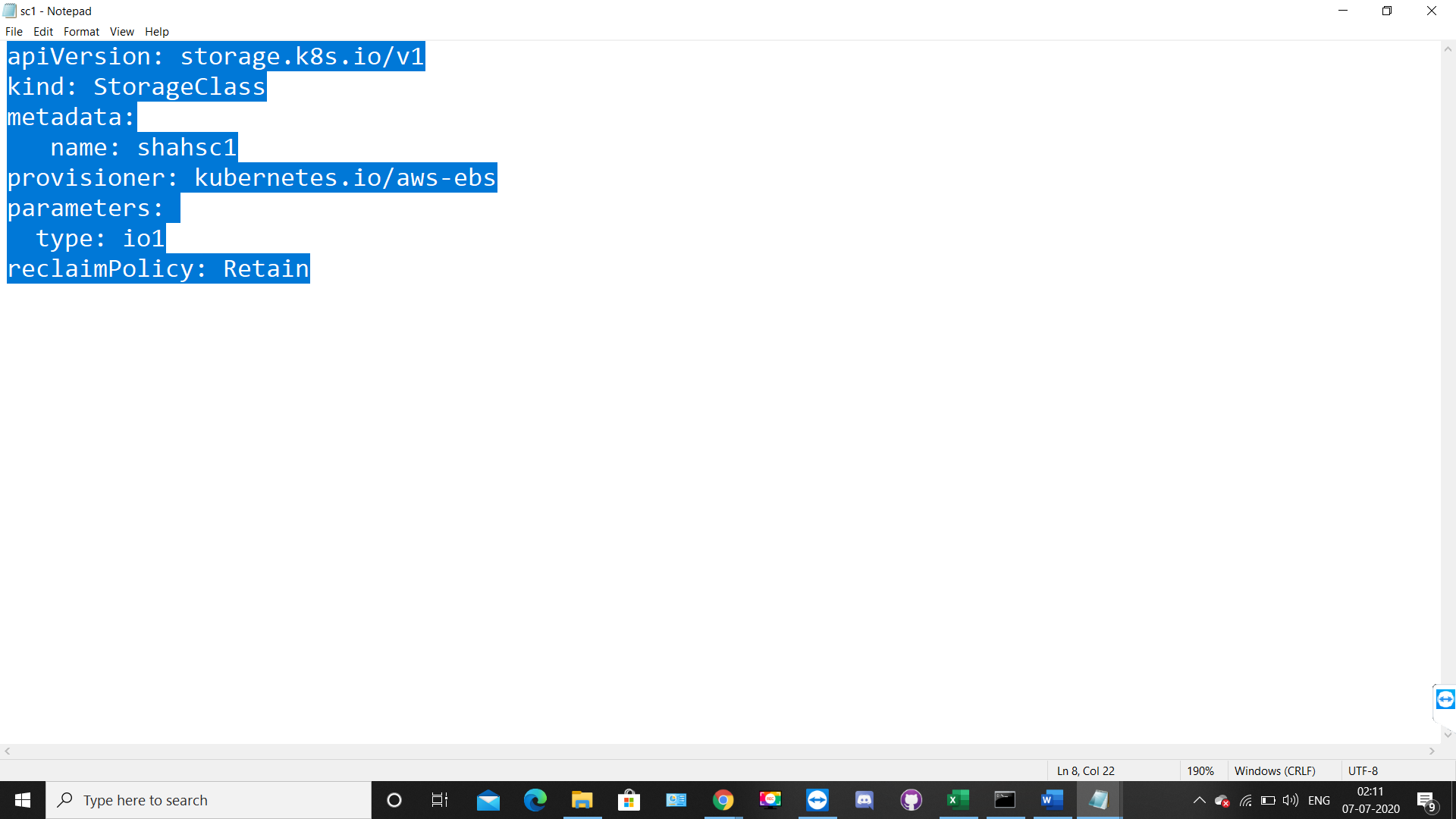
**# kubectl describe sc shahsc1**

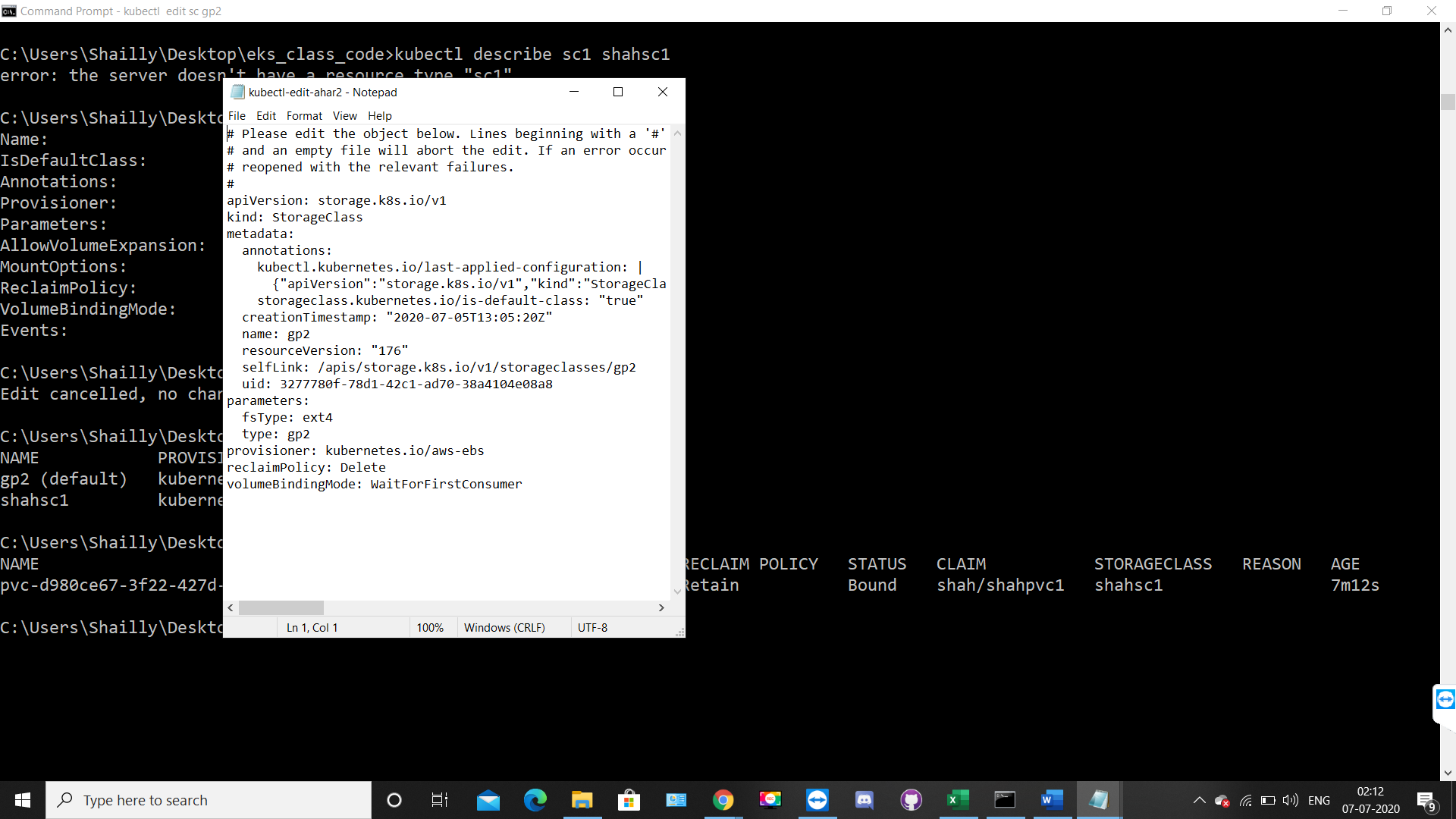


**#kubectl get pv**

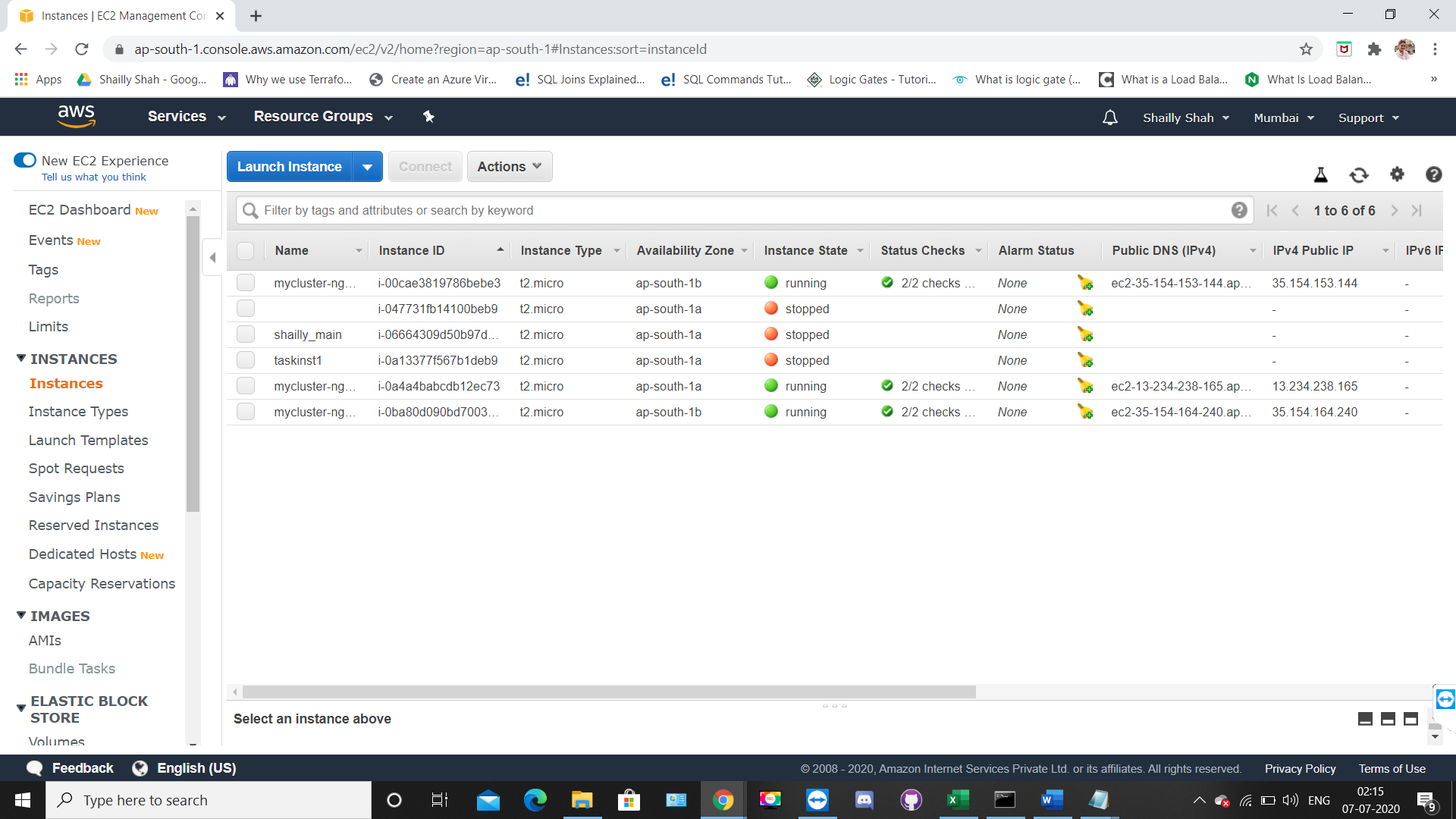




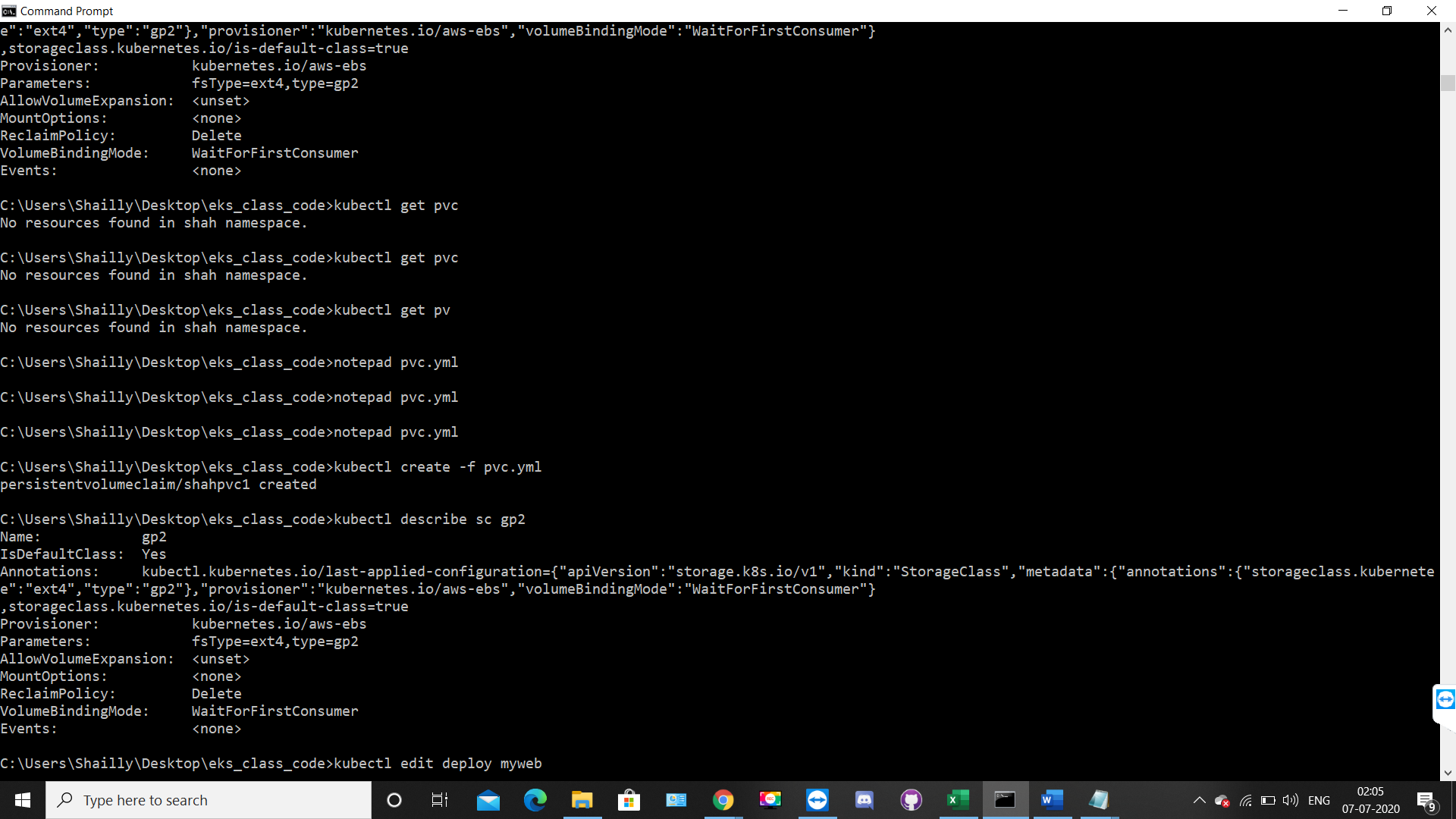




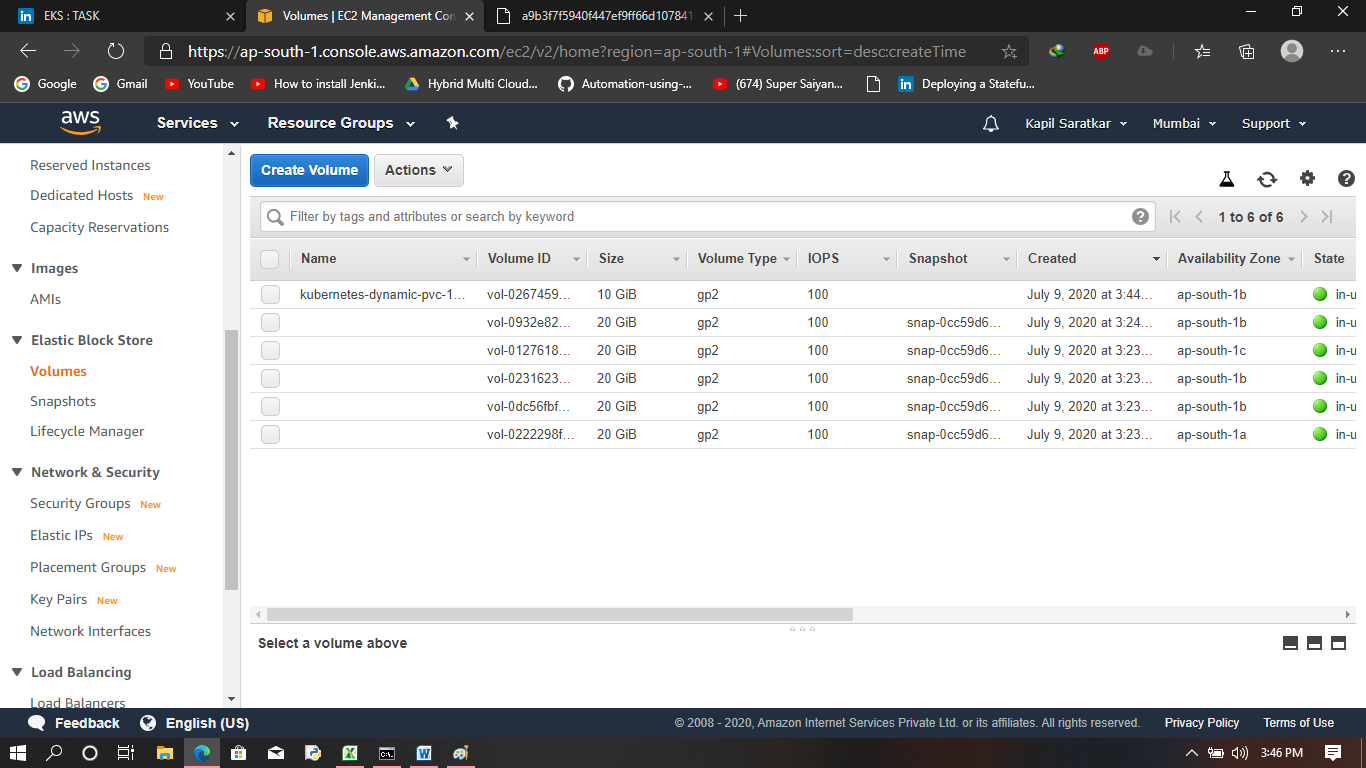
* Here our running instances , which made through our cluster creation:-



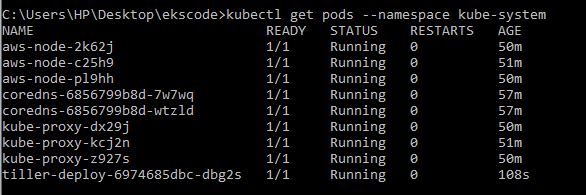
* **We can change the PVC reclaim Policy to retain, in case if we delete pvc, PV won’t delete but it released.**



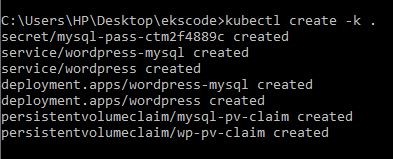
* **You can also check it in the EBS service of AWS**

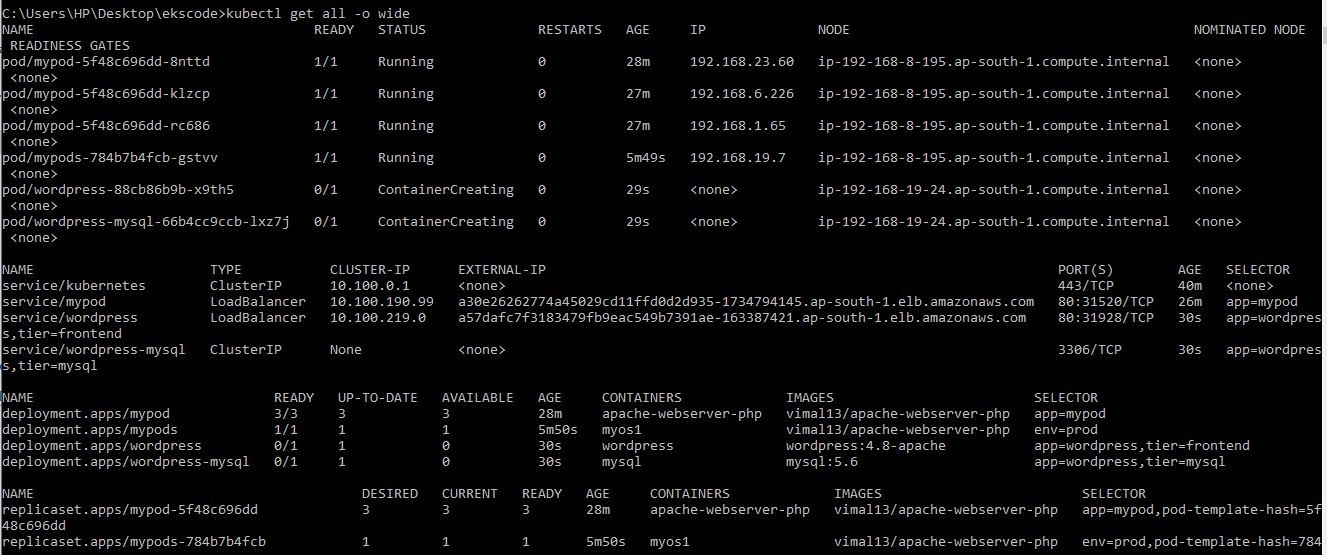


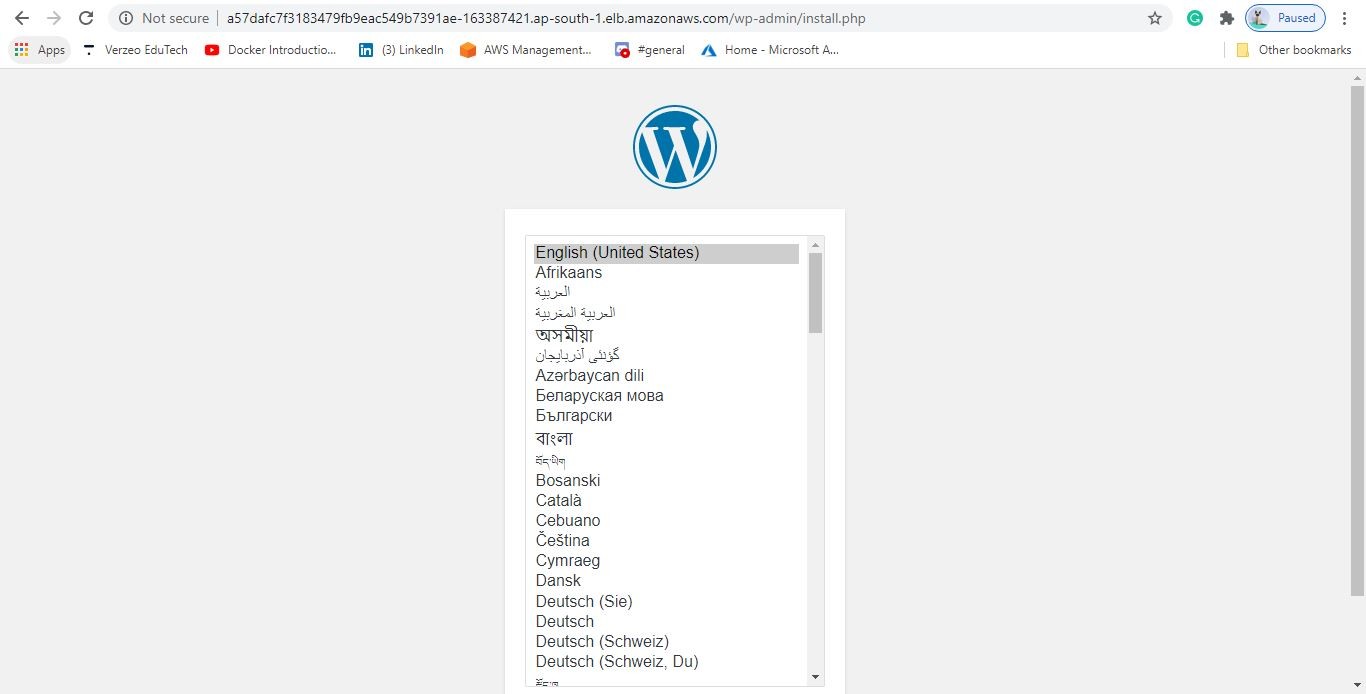
* NOW,we will see how many pods are running inside the kube-system

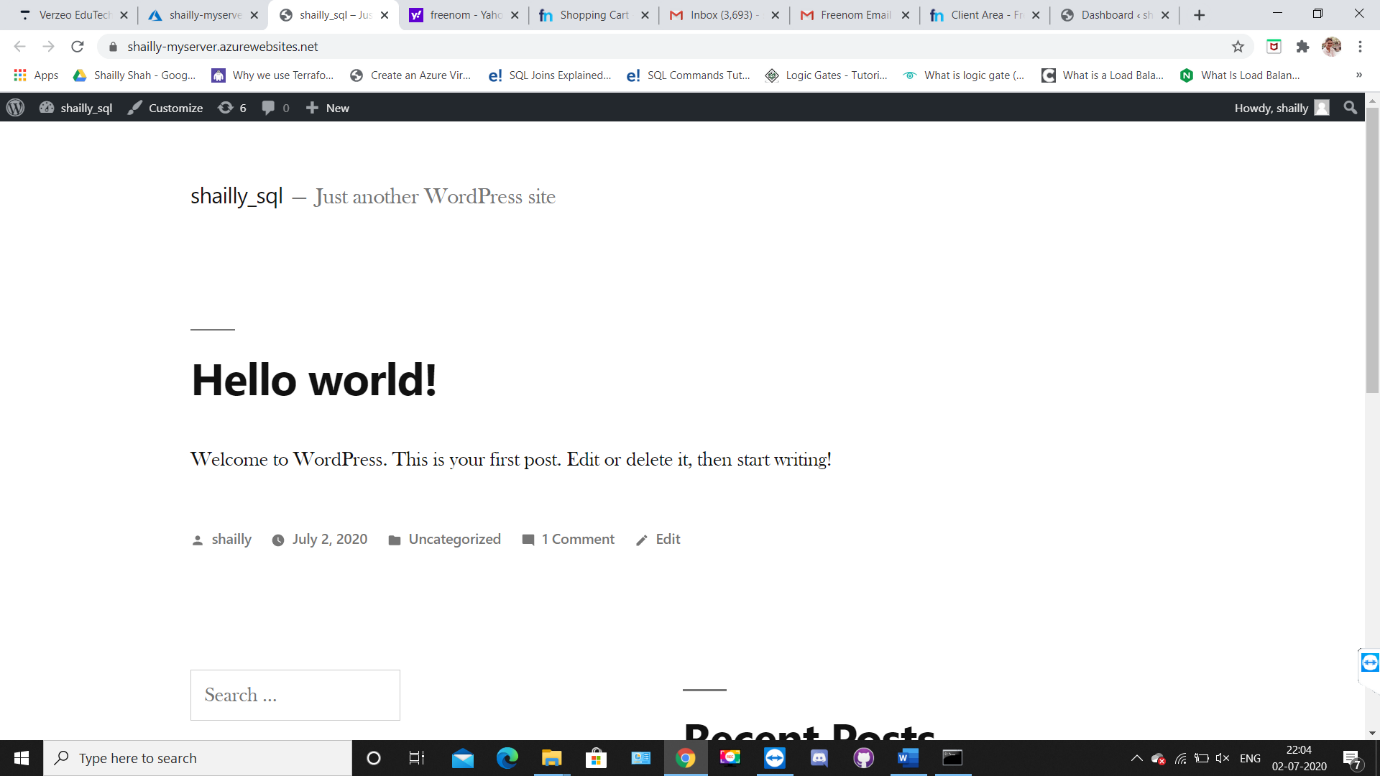


* Now , we are going to create a multitier architecture of Wordpress and MySQL using a single yml file known as kustomization.yml







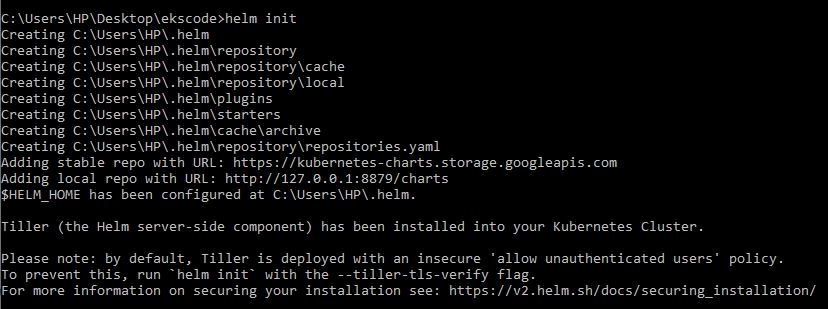


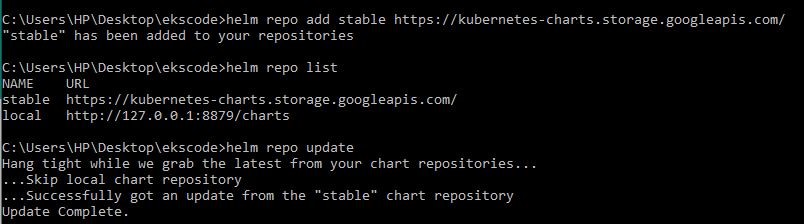
**Here wordpress process is completed,**

* **Now we initialized the helm.**

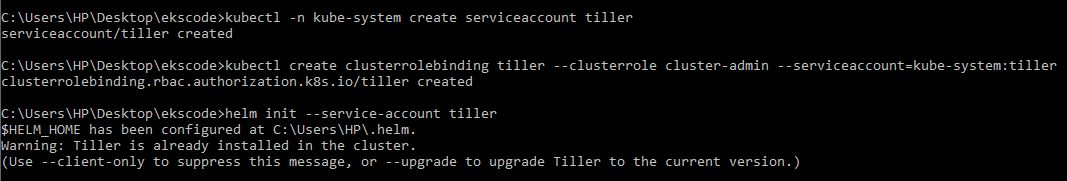
**The Helm package manager for Kubernetes helps you install and manage applications on your Kubernetes cluster**.

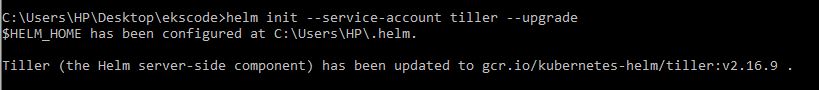
**Helm helps you manage Kubernetes applications — Helm Charts help you define, install, and upgrade even the most complex Kubernetes application.**





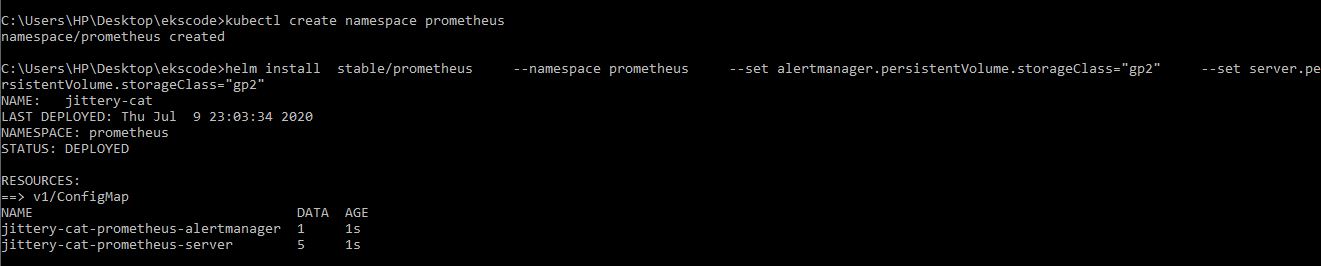
After doing this we will create a tiller it is a server side component of helm and for this first we will see that whether the tiller-deploy is available in kube-system or not

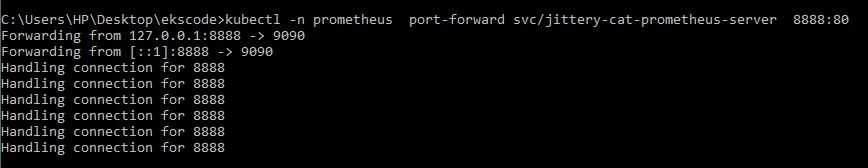


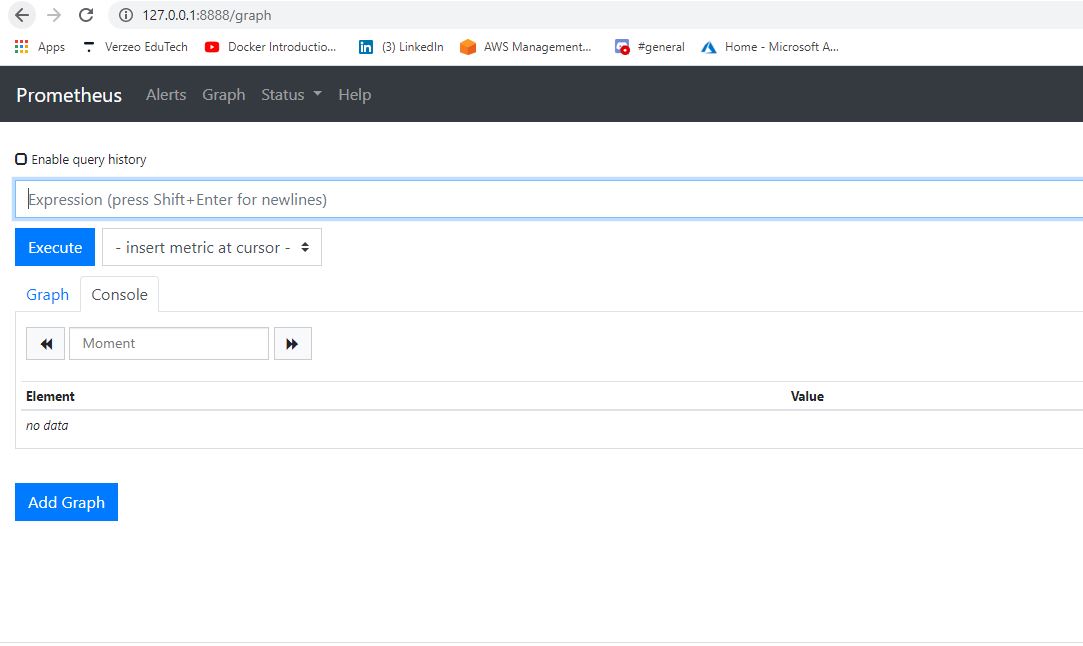


* Now with help of helm and tiller we can install Prometheus.

--for this we first create the namespace





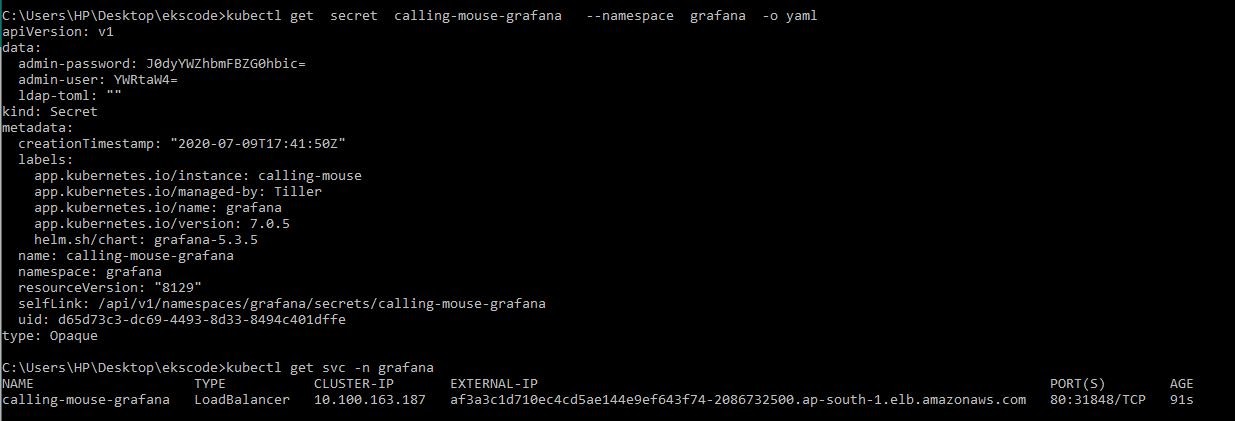


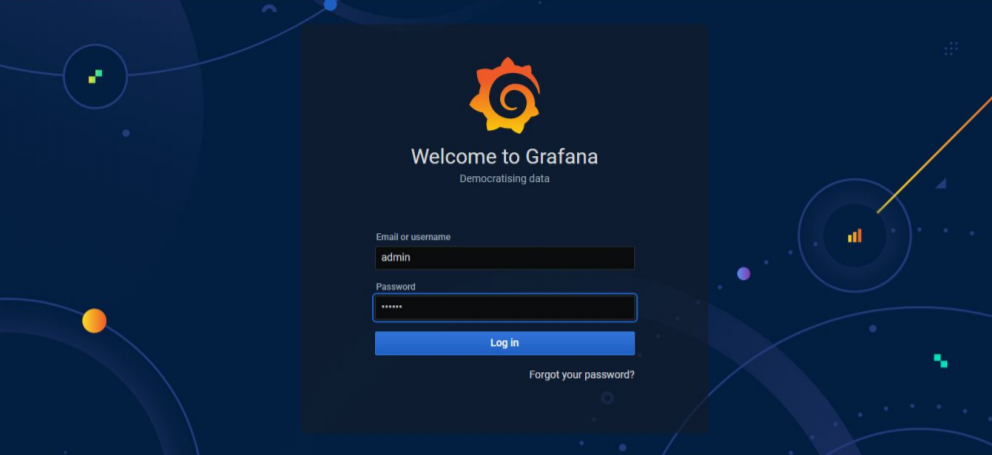


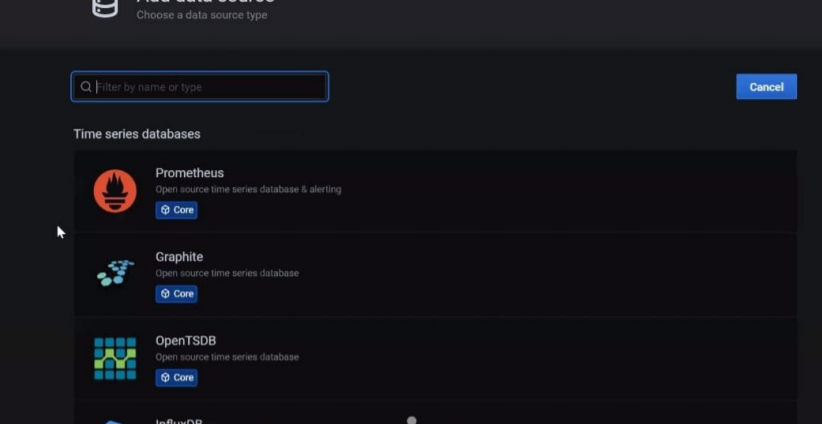
Here we done with our Prometheus,

* Lets go for the installation of Grafana.









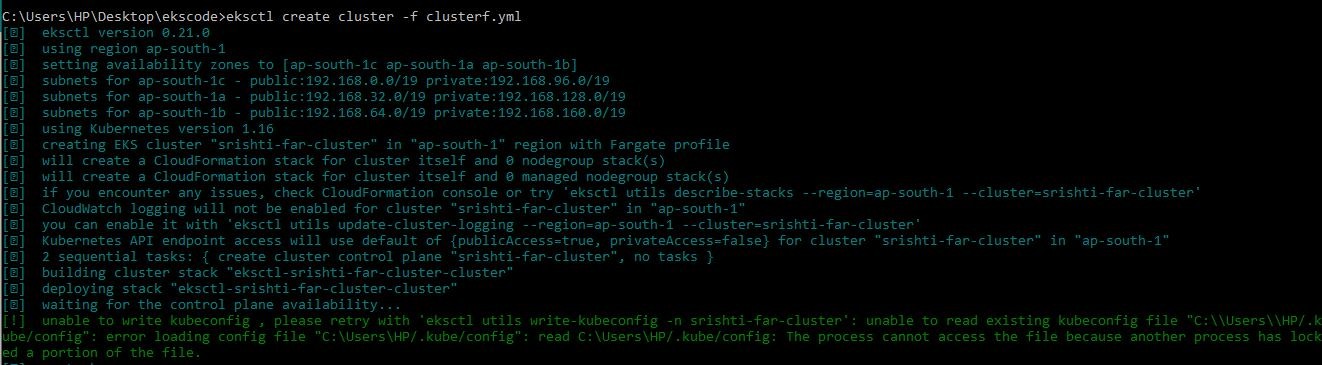
* **Fargate Cluster Creation :-**

AWS Fargate is a technology that you can use with Amazon EC2 to run containers without having to manage servers or clusters of Amazon EC2 instances.

With AWS Fargate, you no longer have to provision, configure, or scale clusters of virtual machines to run containers.

It launch the node automatically on run time. i.e on demand. Thia also give RAM/CPU/Hard-disks etc automatically as per demand.

This concept is known as Serverless Architecture. Fargate is one service in aws-eks world which provide Serverless architecture, i.e black box architecture



* Yml files

