

Capstone Project



June 24, 2022

Syed Bilal Afzal Ashraf

Contents

[Code Base Hierarchy 2](#_Toc107006343)

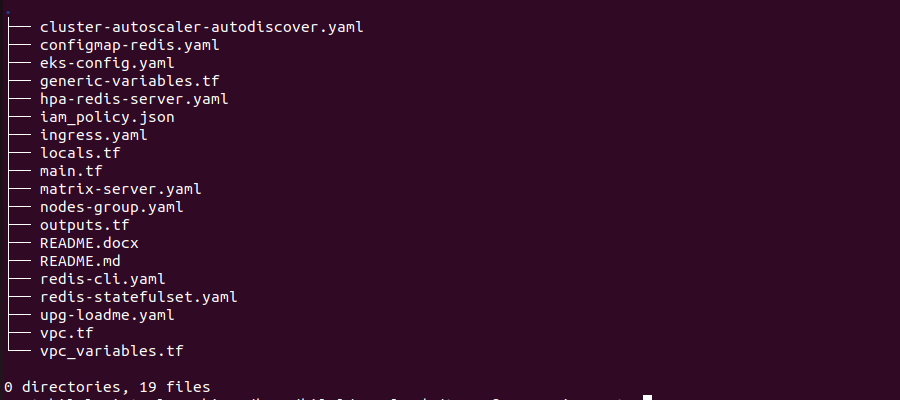
[Steps Required 3](#_Toc107006344)

[Results and Proofs 8](#_Toc107006345)

# Code Base Hierarchy

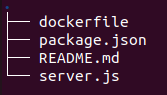
The files needed to deploy terraform, Kubernetes cluster using EKSCTL and Kubernetes resource files are all uploaded to the below repository. The ”outputs.tf” file is having Output variable so that public subnet, private subnet and VPC IDs are display once resources are created. These are helpful to update the EKSCTL config file.

<https://github.com/syedbilalafzal/capstone.git>



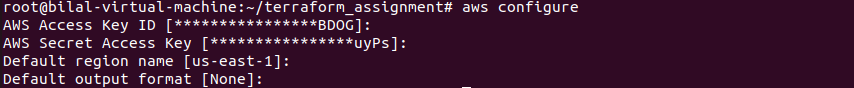
Upg-Loadme app GIT repo link: -

<https://github.com/syedbilalafzal/upg-loadme-app.git>



# Steps Required

* Configure AWS using “aws config” CLI. Enter Access ID, Secret Key and default region name (us-east-1).



* Clone repository

git clone <https://github.com/syedbilalafzal/capstone.git>

cd capstone

* Create S3 bucket using the below CLI.

aws s3api create-bucket --object-lock-enabled-for-bucket --bucket capstone-backend

* Initialize terraform code

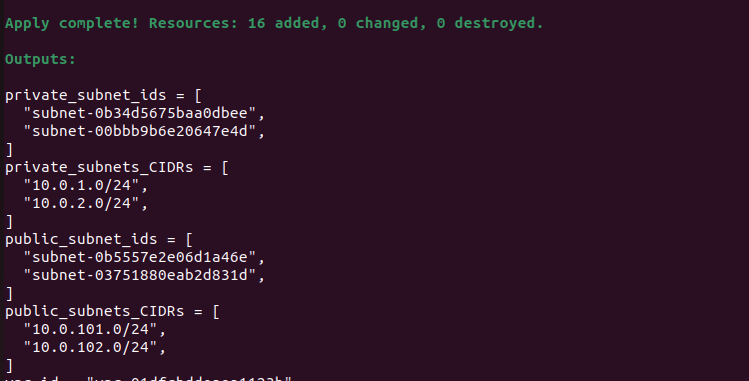
terraform init

* Validate Terraform code

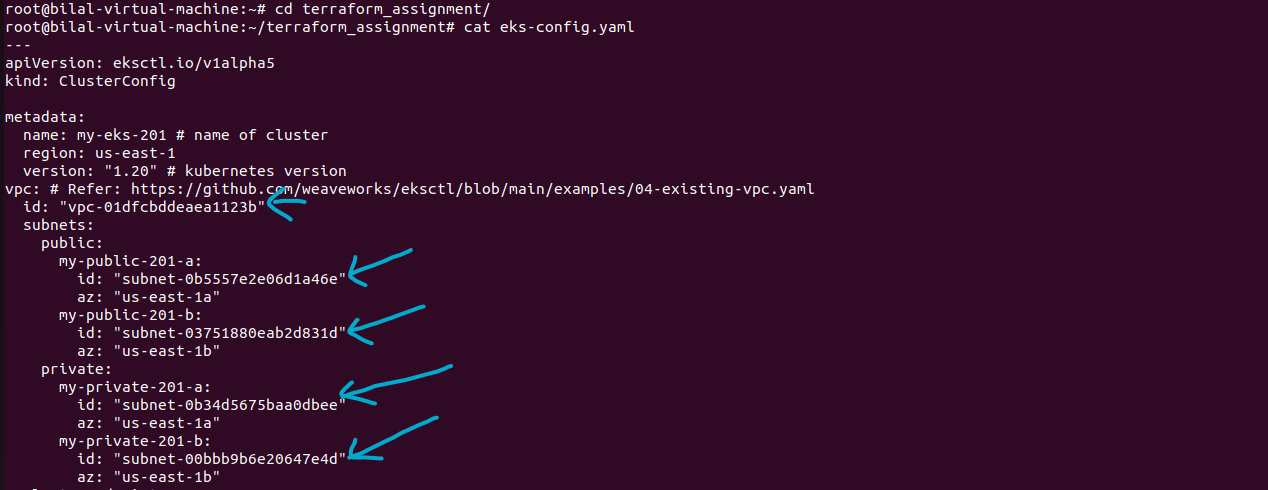
terraform validate

* Apply the terraform config by “terraform apply” and then enter “yes”. Please note down public subnet ID, private subnet ID and VPC ID. Terraform output variables displays this info after the resources are created.

terraform apply



* Modify the “eks-config.yaml” file with correct public subnet ID, private subnet ID and VPC ID collected from the Terraform output.



* Create Kubernetes cluster using EKSCTL and update the kubeconfig file.

eksctl create cluster --config-file eks-config.yaml

aws eks update-kubeconfig --name="my-eks-201"

* Install AWS Loadbalancer controller

helm install aws-load-balancer-controller eks/aws-load-balancer-controller \

-n kube-system \

--set clusterName=my-eks-201 \

--set serviceAccount.create=false \

--set serviceAccount.name=aws-load-balancer-controller \

--set image.repository=602401143452.dkr.ecr.us-east-1.amazonaws.com/amazon/aws-load-balancer-controller

* Deploy matrix server

kubectl apply -f matrix-server.yaml

* Deploy cluster AutoScaler.

kubectl apply -f cluster-autoscaler-autodiscover.yaml

* Login to ECR Repository. Replace *<account ID>* with Amazon Account ID.

aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin <*account ID*>.dkr.ecr.us-east-1.amazonaws.com

* Clone upg-loadme application repository and change directory.

git clone <https://github.com/syedbilalafzal/upg-loadme-app.git>

cd upg-loadme-app

* Build image, tag it and uploaded it to the repository. Replace *<account ID>* with Amazon Account ID.

docker build -t sample-app:latest .

docker tag sample-app:latest <*account ID*>.dkr.ecr.us-east-1.amazonaws.com/sample-app:latest

docker push <*account ID*>.dkr.ecr.us-east-1.amazonaws.com/sample-app:latest

* Goto parent directory

cd ..

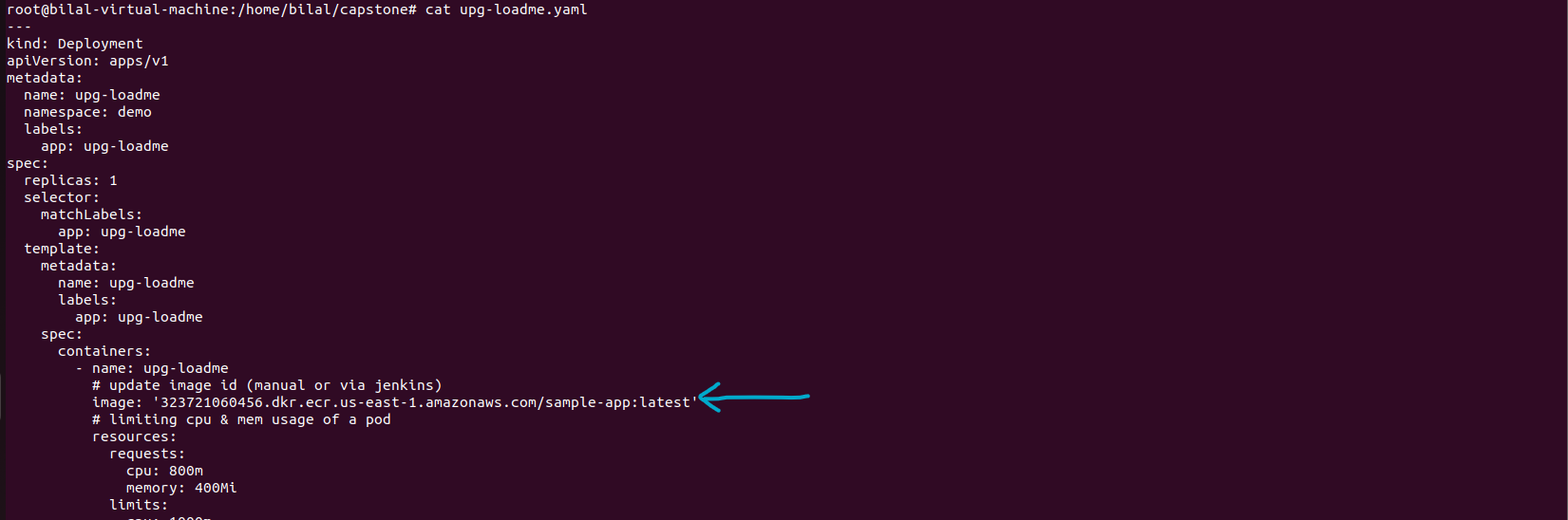
* Create nodegroup with taint.

eksctl create nodegroup --config-file nodes-group.yaml --include='pvt-201-a-3'

* Create “demo” namespace.

kubectl create ns demo

* Modify the image field with your ECR repository link in the below specified field under the file “upg-loadme.yaml”.



* Apply the file, it will create upg-loadme Deployment and Service resources.

kubectl apply -f upg-loadme.yaml

* Create deploy ingress service. This will create ingress service and the Amazon Loadbalancer Controller will detect the ingress service to configure the ALB and provide URL.

kubectl apply -f ingress.yaml

kubectl get ingress -n demo

* For redis statefulset, we configmap and Service. Both configmap and service are defined under the same file called “configmap-redis.yaml”

kubectl apply -f configmap-redis.yaml

* Deploy redis-server statefulset.

kubectl apply -f redis-statefulset.yaml

kubectl get statefulset -n demo

* Deploy redis-cli and check if redis-cli is able to access redis-server. Replace <pod name> with redis-cli pod name retrieved from “kubectl get pod -n demo”

kubectl apply -f redis-cli.yaml

kubectl get pod -n demo

kubectl exec -n demo -it <pod name> -- sh

* Connect to the redis-server. Set value and get value

redis-cli -h redis-server -p 6379

SET foo 1

GET foo

* Delete redis-server pod.

kubectl delete pod redis-server-0 -n demo

* Wait for pod to come back

kubectl get pod -n demo -w

* SSH to the redis-cli pod again.

kubectl get pod -n demo

kubectl exec -n demo -it <pod name> -- sh

* Get the foo value to confirm if value is retained due to persistent volume.

redis-cli -h redis-server -p 6379

GET foo

* Deploy horizontal pod Autoscaler for upg-loadme app.

kubectl apply -f hpa-upg-loadme.yaml

kubectl get hpa -n demo

* Add Prometheus repo to HELM and update the HELM repo.

helm repo add prometheus-community <https://prometheus-community.github.io/helm-charts>

helm repo update

* Install prometheus.

helm install prometheus prometheus-community/kube-prometheus-stack

* SSH to Prometheus.

kubectl port-forward deployment/prometheus-server 9090

* Open another tab and install apache benchmark “ab”.

apt-get install apache2-utils

* Get Ingress Amazon ALB URL and then to the stress test using “ab” utility.

kubectl get ingress -n demo

ab -n100 -c20 'http://<Amazon ALB URL>/load?scale=300'

* Open another window and watch HPA status.

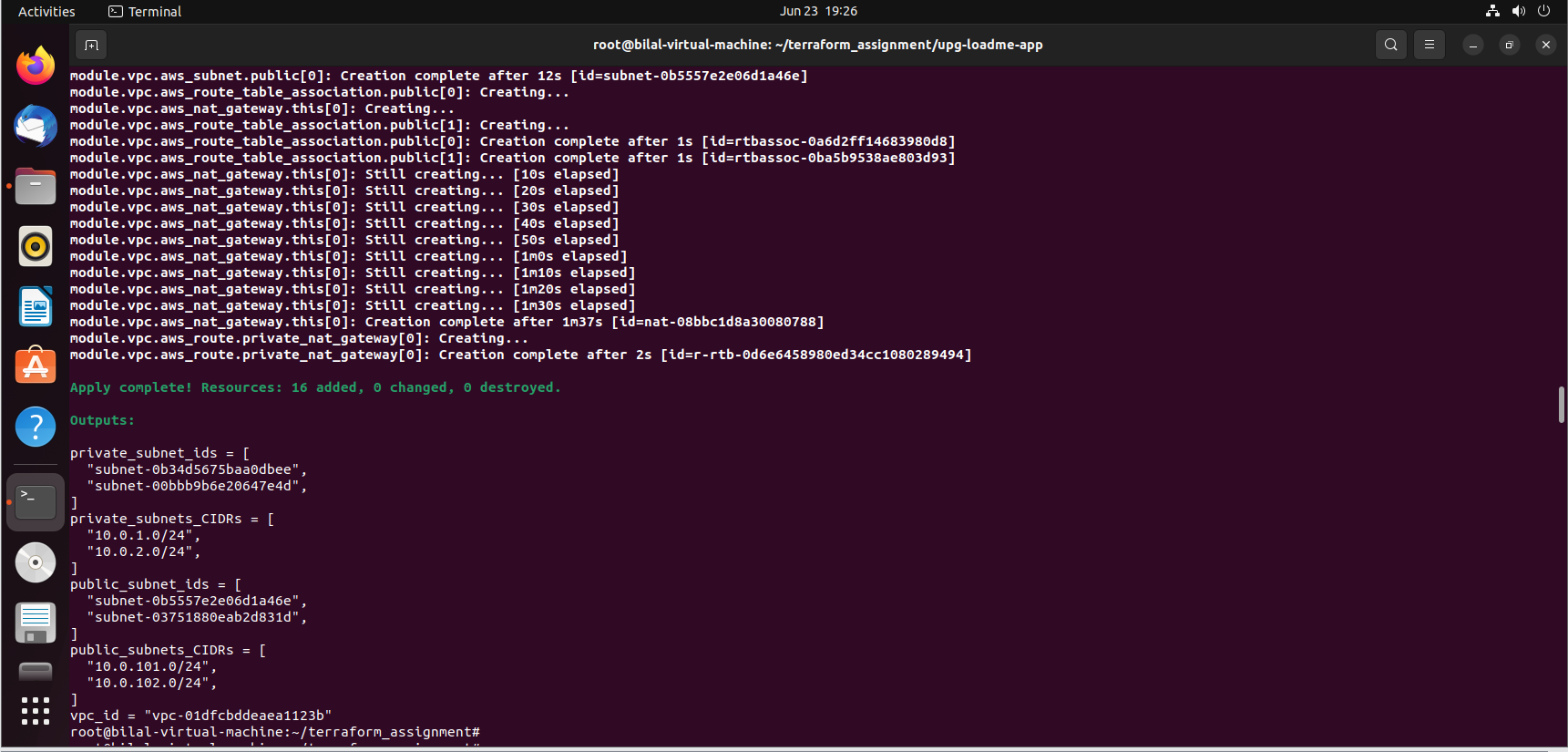
kubectl get hpa -n demo -w

* Go to web browser and enter URL <http://localhost:9090>” to open Prometheus. Use the below Query to display total Pods for upg-loadme app.

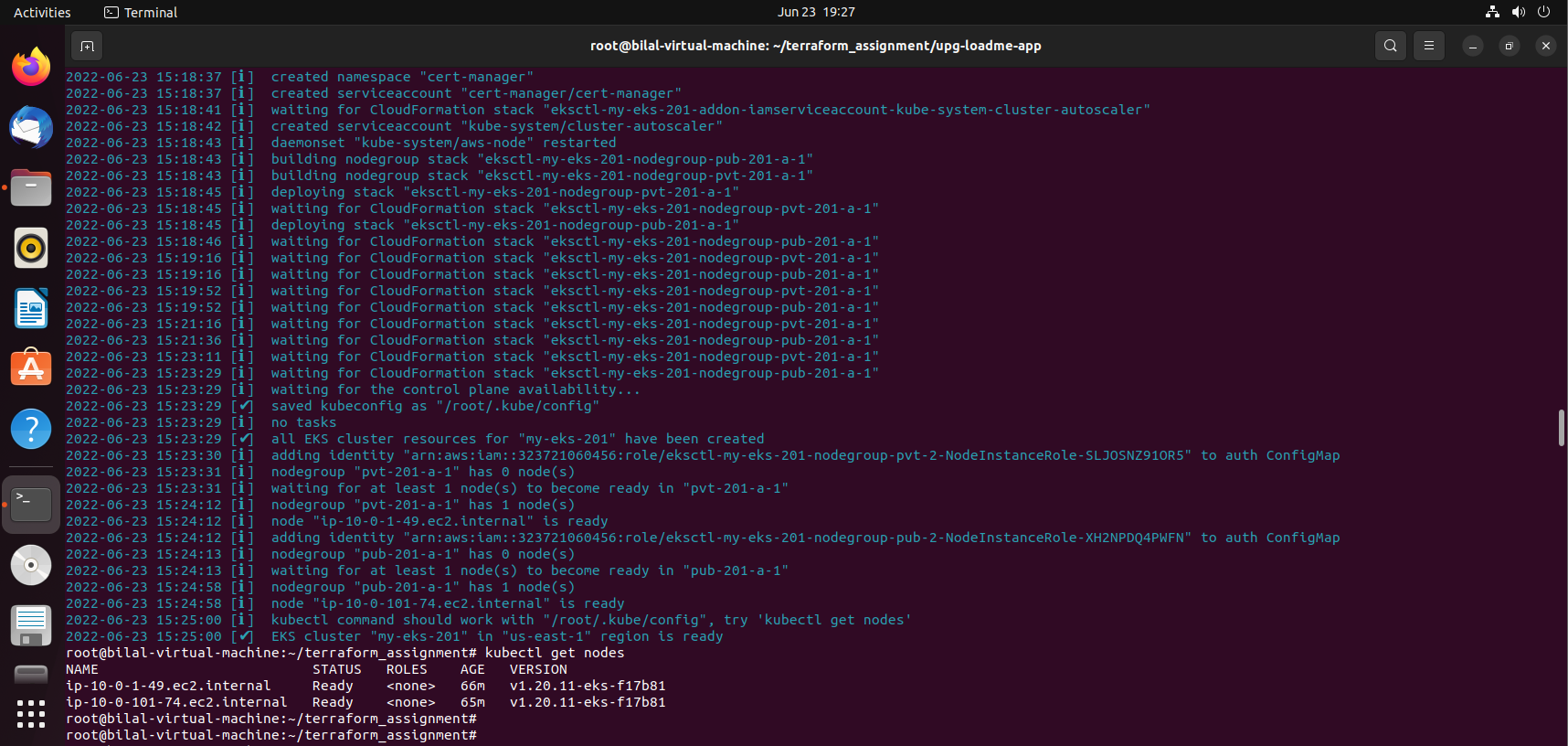
sum(kube\_pod\_container\_status\_ready{namespace="demo",pod=~"upg-loadme-.\*"})

# Results and Proofs

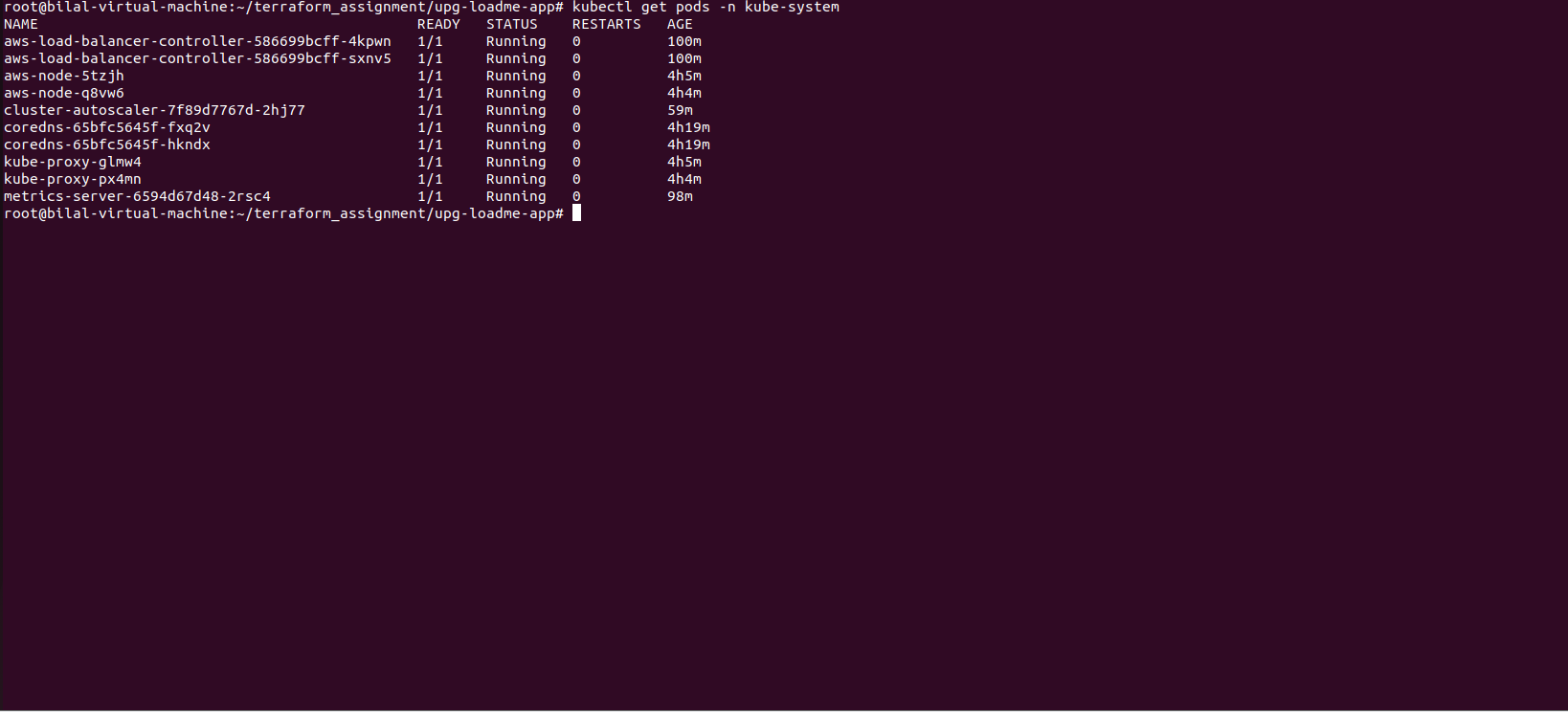
* Terraform infrastructure.



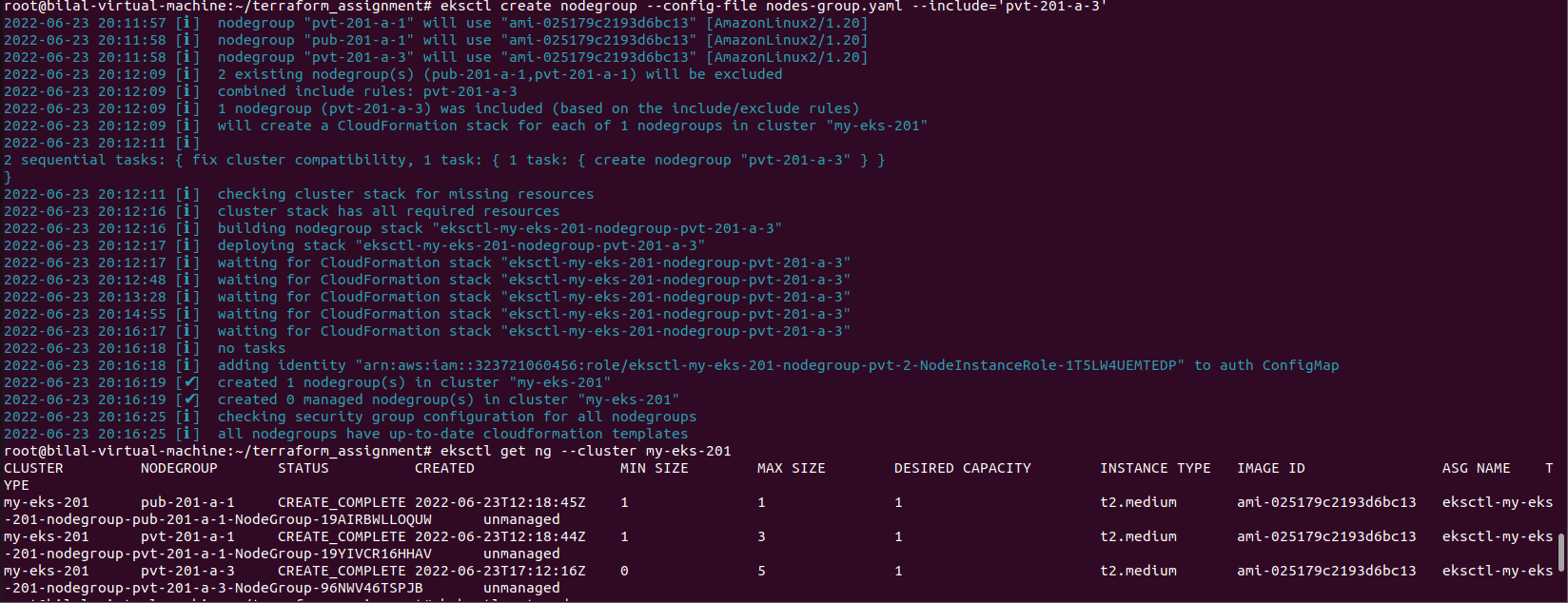
* EKS cluster created successfully using EKSCTL

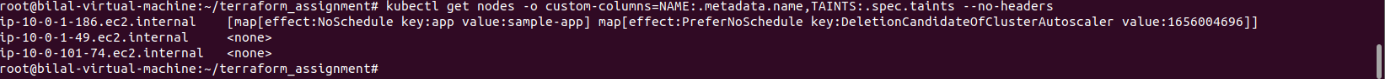


* Add-ons installed.

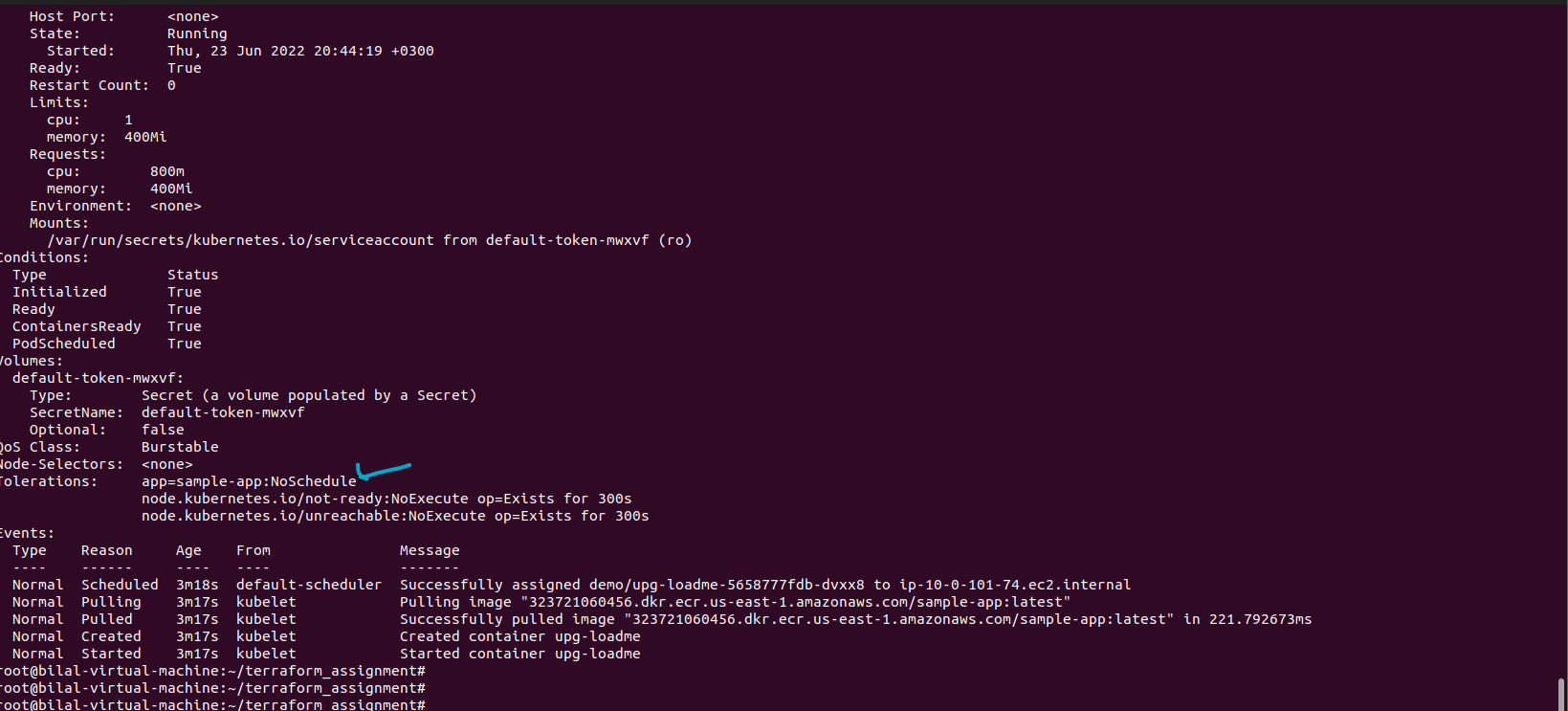


* 3rd Node Group with Taints added to EKS.

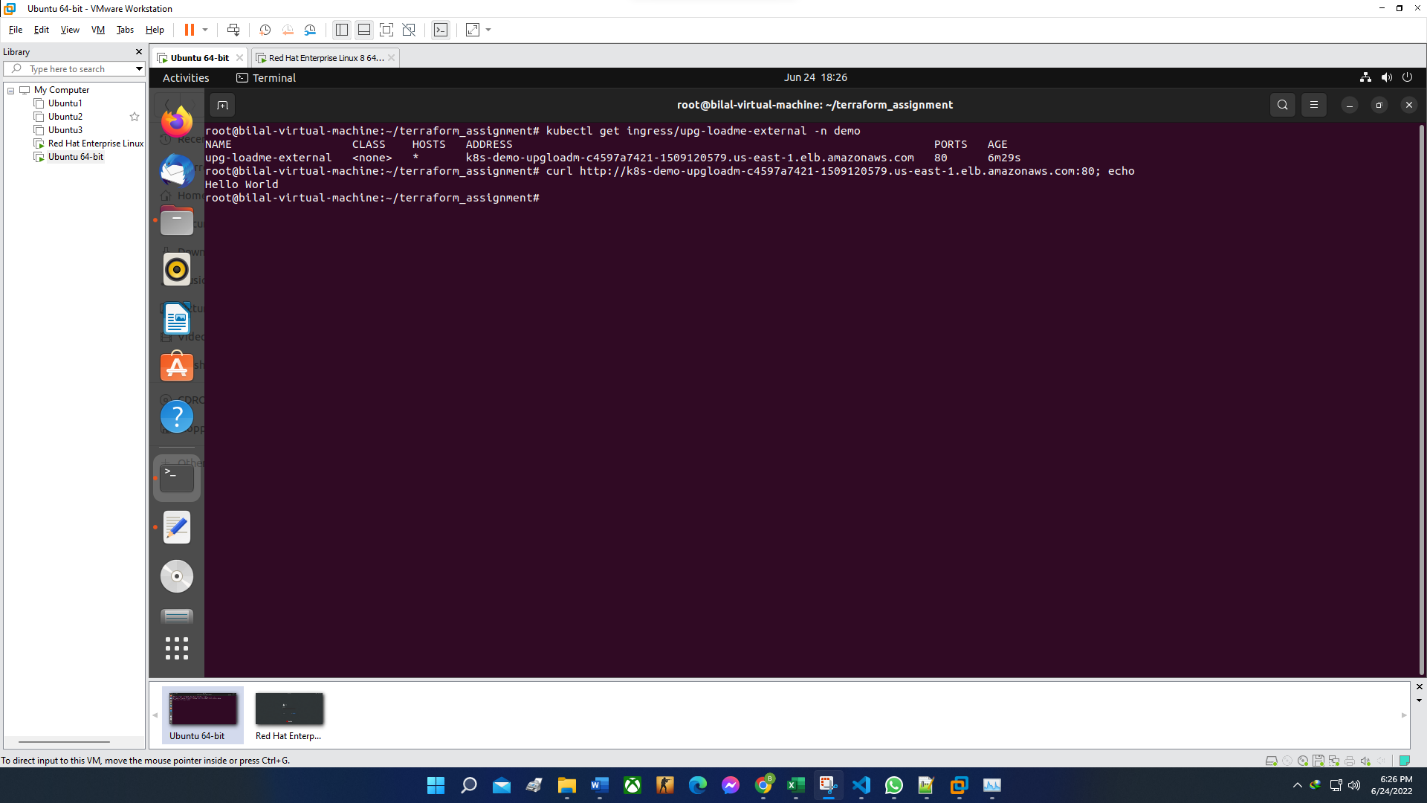




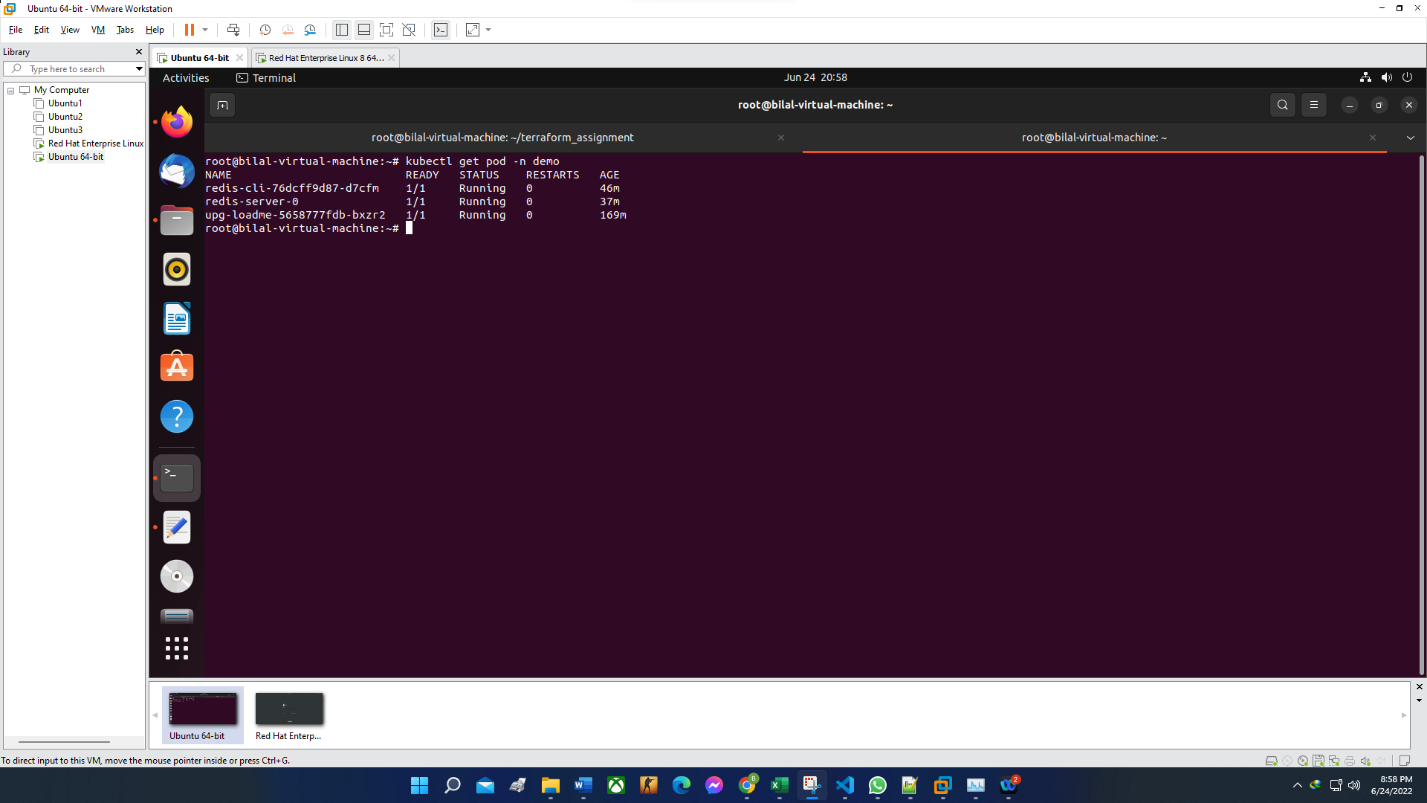
* Toleration applied to the deployment pods.



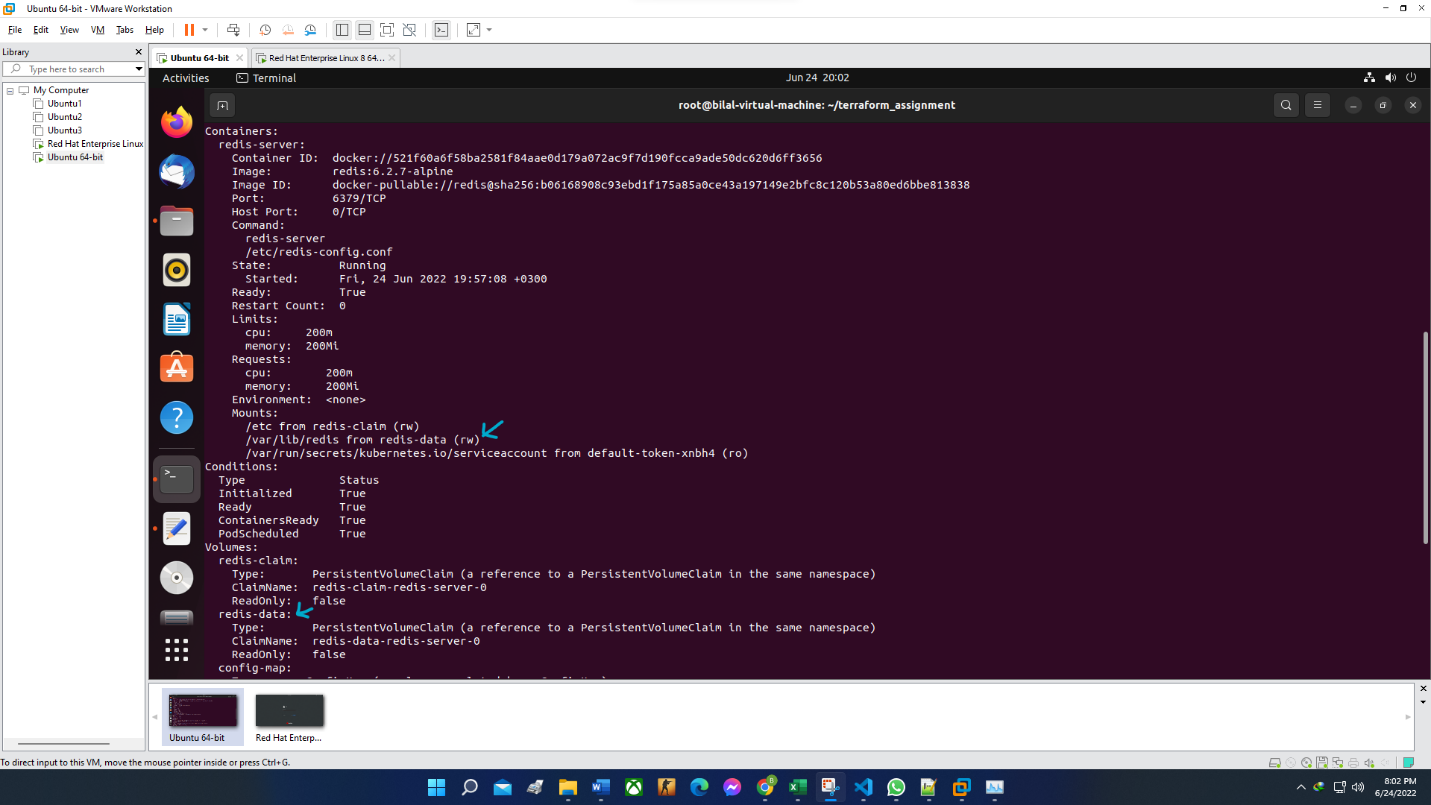
* App “upg-loadme” CURL successful.

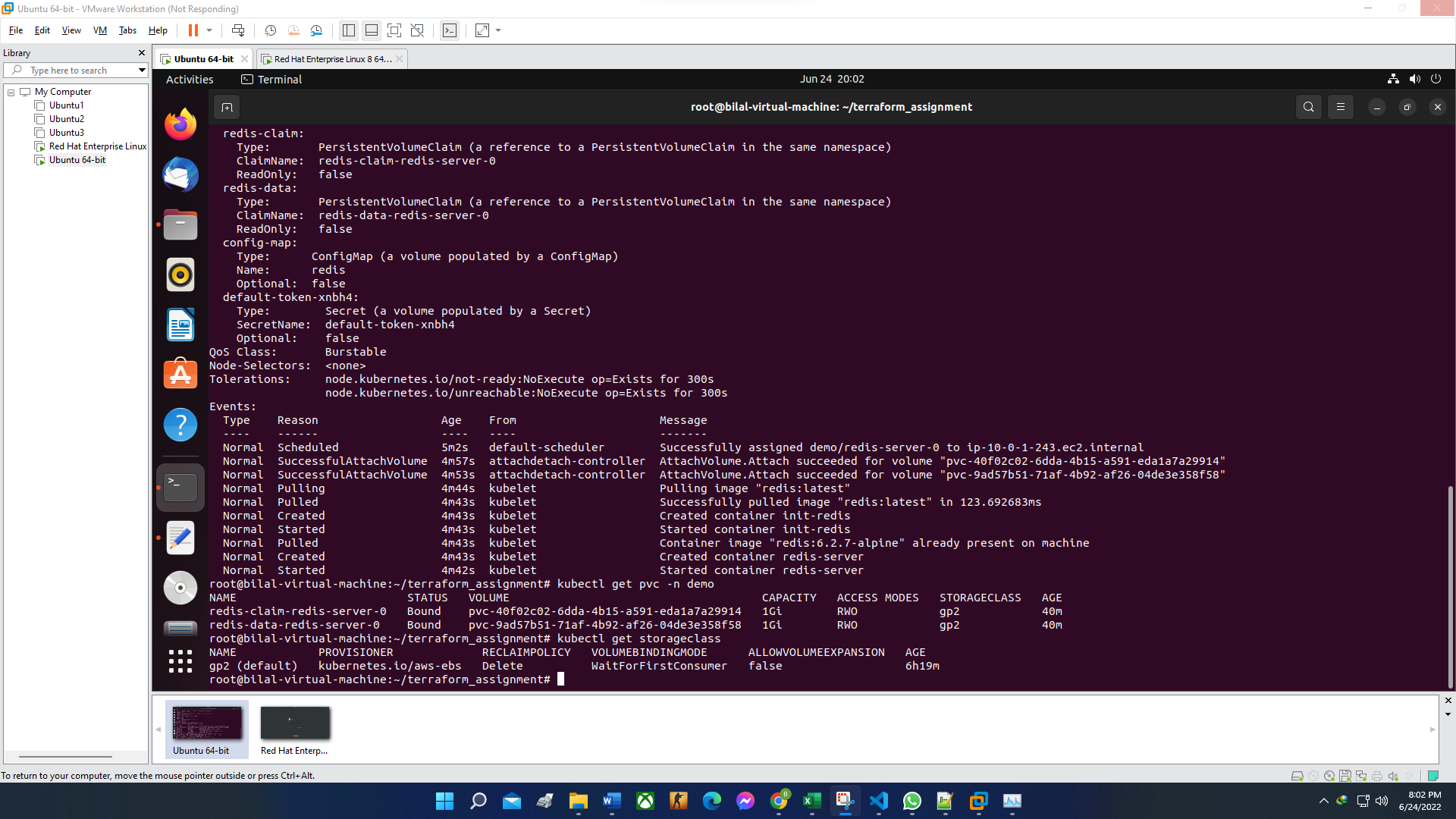


* Redis-Server and Redis-cli installed successfully.

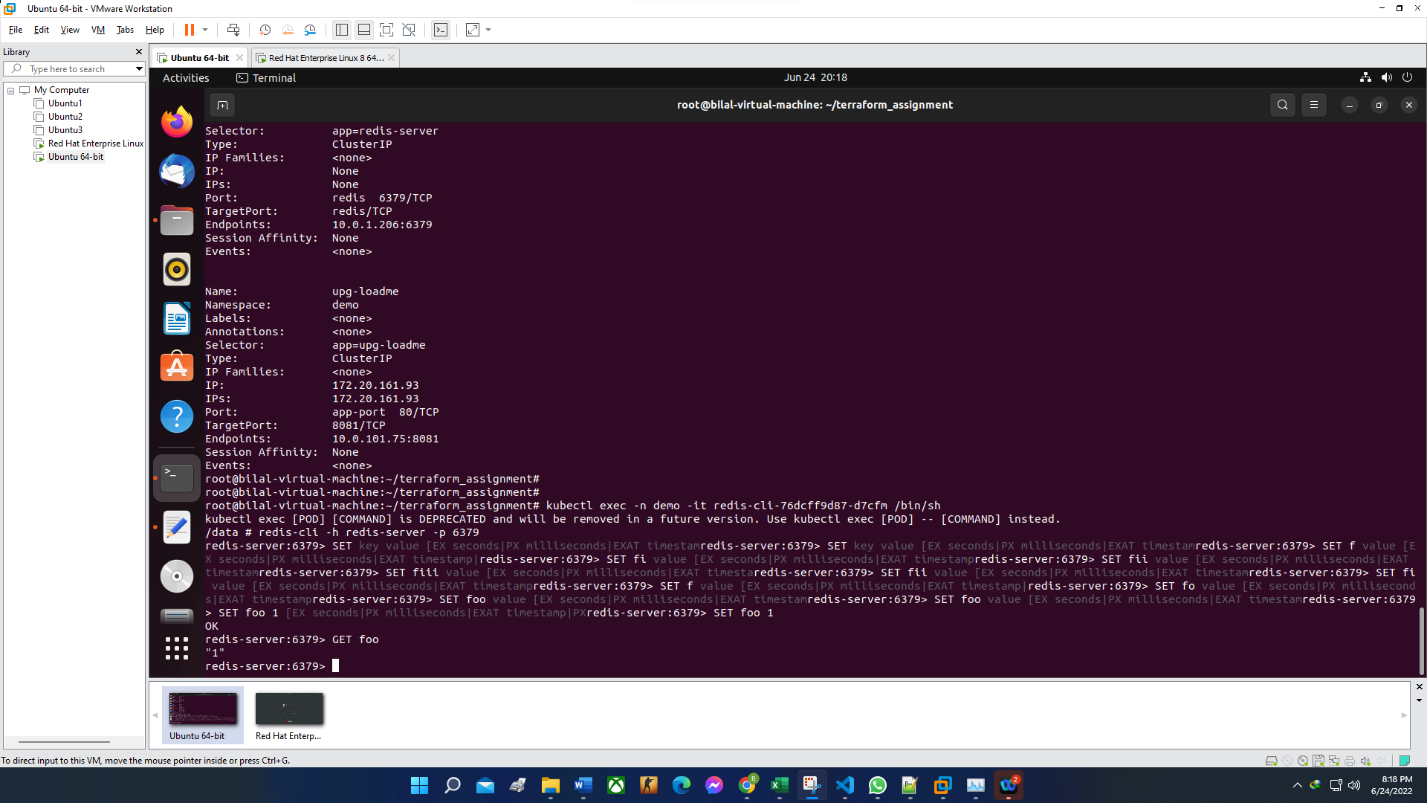


* PVC by EBS mounted to “redis-server”

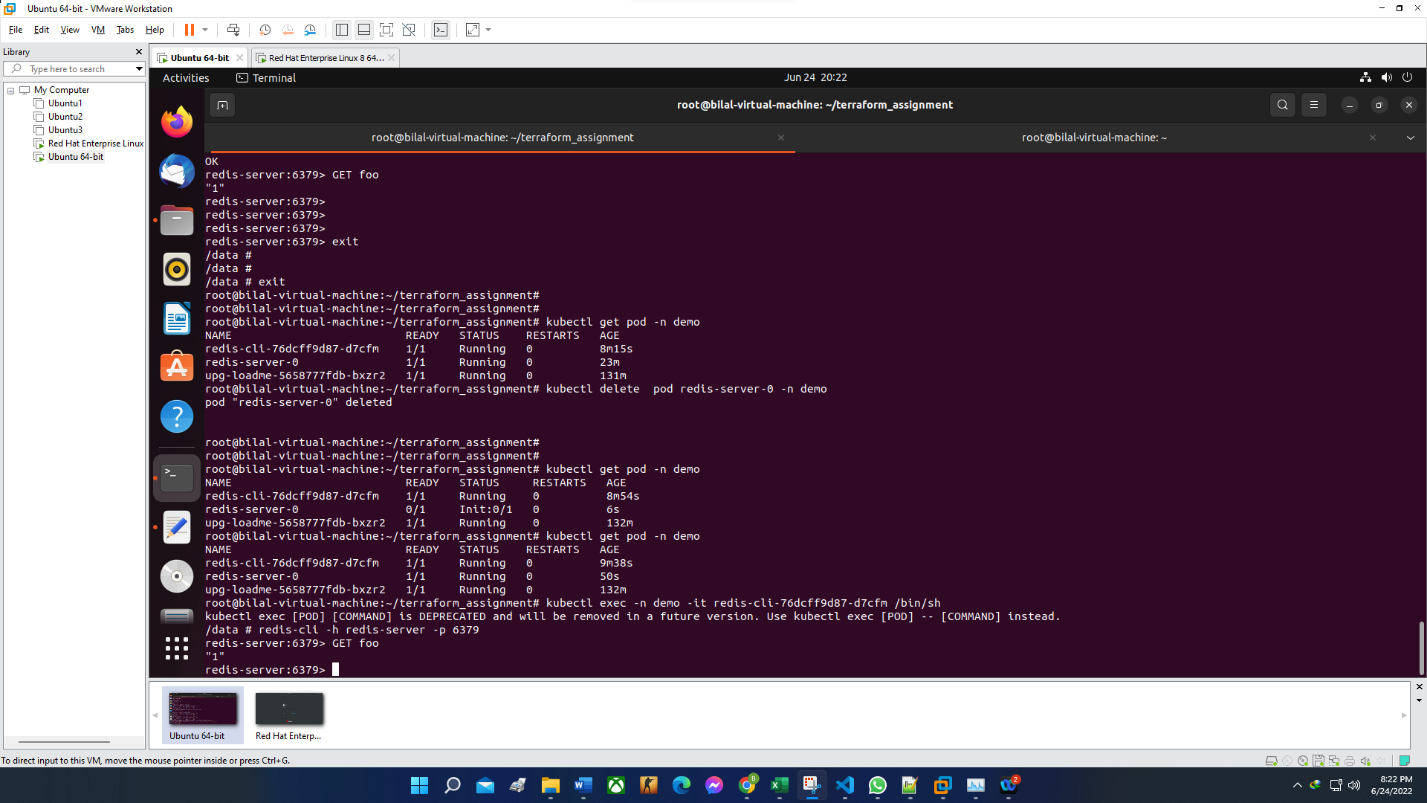




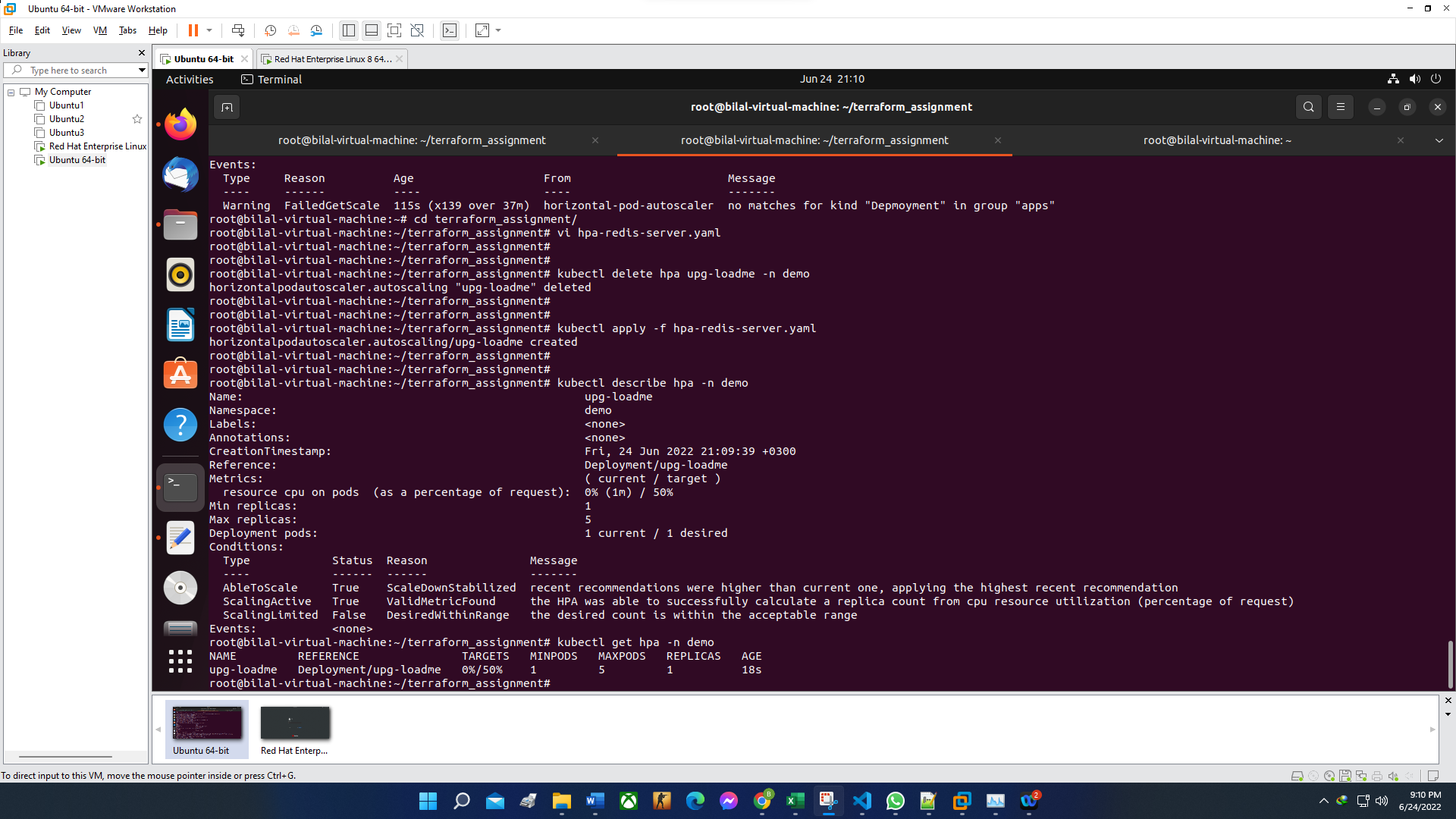
* Set and Get value using “redis-cli” to “redis-server before deleting the pod.



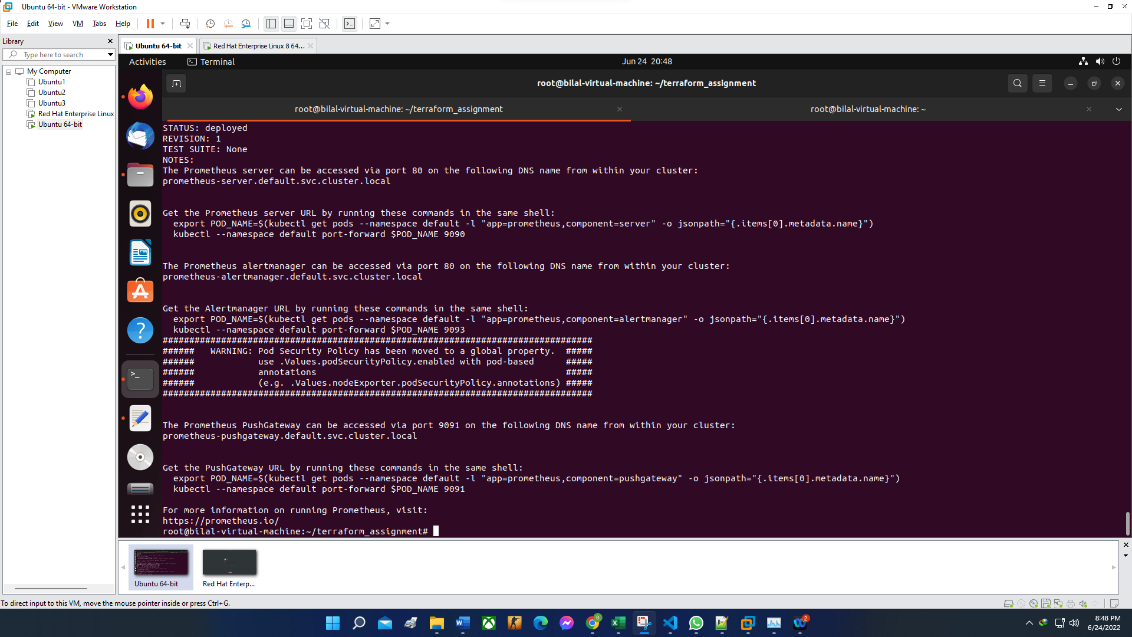
* Get “foo” value after “redis-server” is restarted.



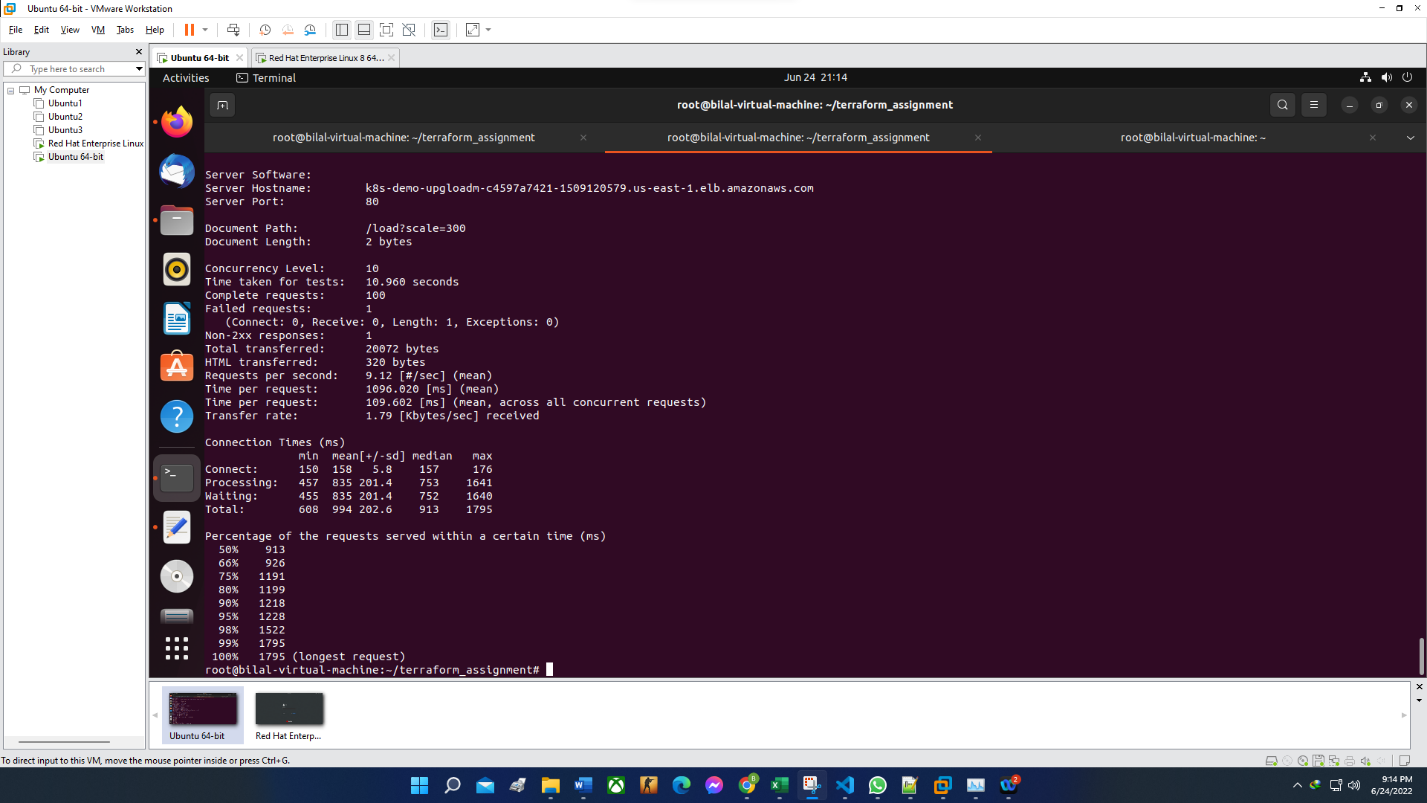
* HPA deployed on upg-loadme.



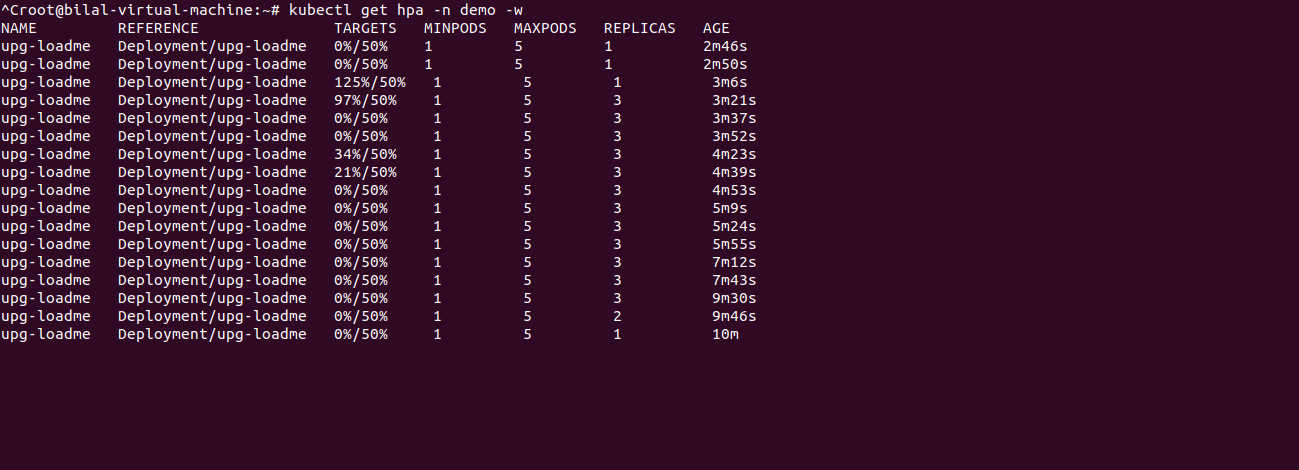
* Prometheus installed.



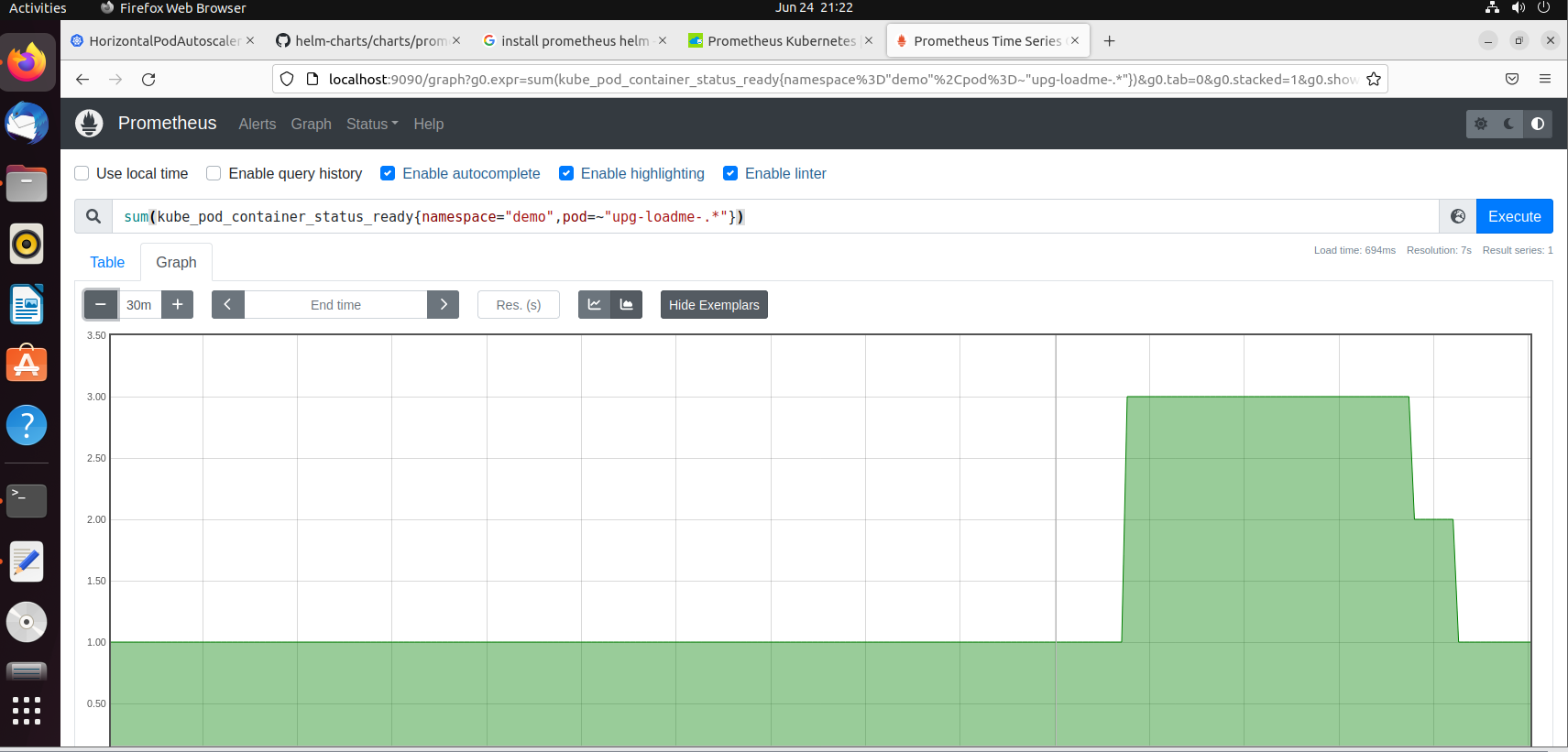
* Generate load from Apache Benchmark utility.



* HPA status during load test.



* Prometheus Graph covering the load test period.



# Bonus Task

Below were the security groups created by EKSCTL along with their explaination.

|  |  |
| --- | --- |
| GroupName | Description |
| eksctl-my-eks-201-nodegroup-pvt-201-a-1-SG-1ECO24GNLDQS9 | Communication between the control plane and worker nodes in group pvt-201-a-1 |
| eksctl-my-eks-201-cluster-ControlPlaneSecurityGroup-ZYEUREN2DTO0 | Communication between the control plane and worker nodegroups |
| eks-cluster-sg-my-eks-201-167234222 | EKS created security group applied to ENI that is attached to EKS Control Plane master nodes, as well as any managed workloads. |
| k8s-traffic-myeks201-d0d8972b01 | [k8s] Shared Backend SecurityGroup for LoadBalancer |
| eksctl-my-eks-201-nodegroup-pub-201-a-1-SG-17ADPVLIC96YO | Communication between the control plane and worker nodes in group pub-201-a-1 |
| eksctl-my-eks-201-cluster-ClusterSharedNodeSecurityGroup-LZPAVNCBUB7G | Communication between all nodes in the cluster |
| eksctl-my-eks-201-nodegroup-pvt-201-a-3-SG-5YU774M3V4RE | Communication between the control plane and worker nodes in group pvt-201-a-3 |
| k8s-demo-upgloadm-39e9f15560 | [k8s] Managed SecurityGroup for LoadBalancer |