DEVOPS CAPSTONE PROJECT

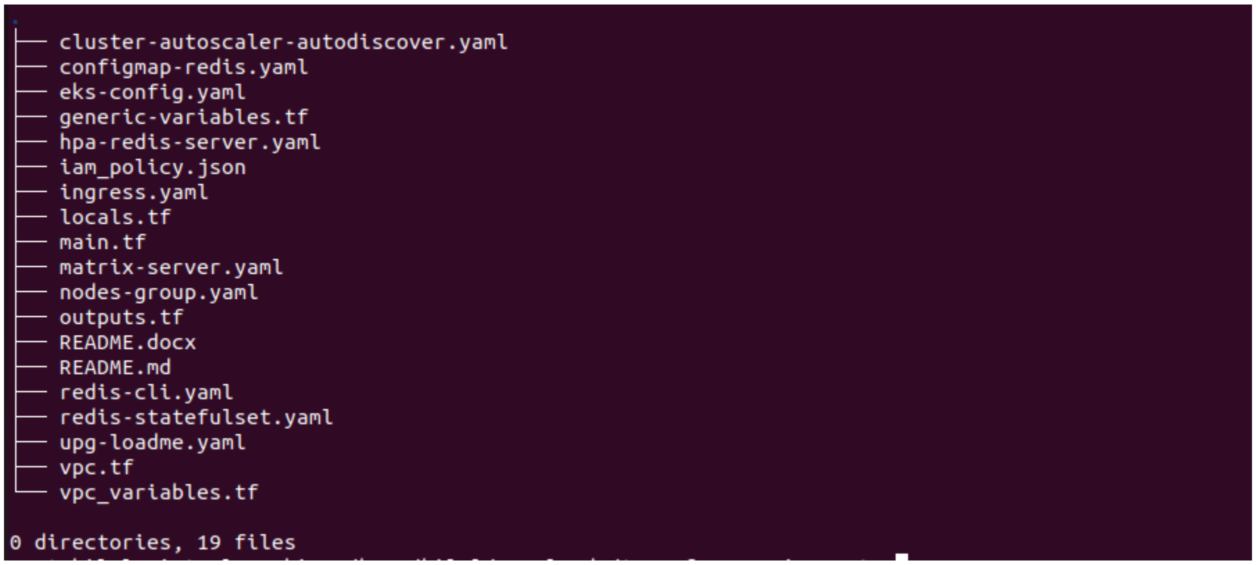
Contents

Code Base Hierarchy 1
Steps Required 1
Results and Proofs 7

Code Base Hierarchy

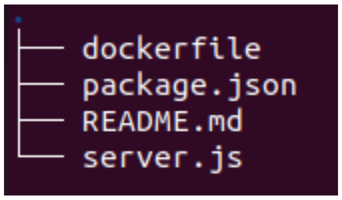
The files needed to deploy terraform, Kubernetes cluster using EKSCTL and Kubernetes resource files are all uploaded to the below repository. The “outputs.tf” file is having Output variable so that public subnet, private subnet and VPC IDs are display once resources are created. These are helpful to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

https://github.com/ms-sourcetech/capstone-devops-upgrad



```
.
├── cluster-autoscaler-autodiscover.yaml
├── configmap-redis.yaml
├── eks-config.yaml
├── generic-variables.tf
├── hpa-redis-server.yaml
├── iam_policy.json
├── ingress.yaml
├── locals.tf
├── main.tf
├── matrix-server.yaml
├── nodes-group.yaml
├── outputs.tf
├── README.docx
├── README.md
├── redis-cli.yaml
├── redis-statefulset.yaml
├── upg-loadme.yaml
├── vpc.tf
└── vpc_variables.tf

0 directories, 19 files
```

Upg-Loadme app GIT repo link: - https://github.com/ms-sourcetech/upg-loadme-app



```
.
├── dockerfile
├── package.json
├── README.md
└── server.js
```

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/ms-sourcetech/capstone-devops-upgrad
cd capstone
```

- Create S3 bucket using the below CLI.

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

- Initialize terraform code

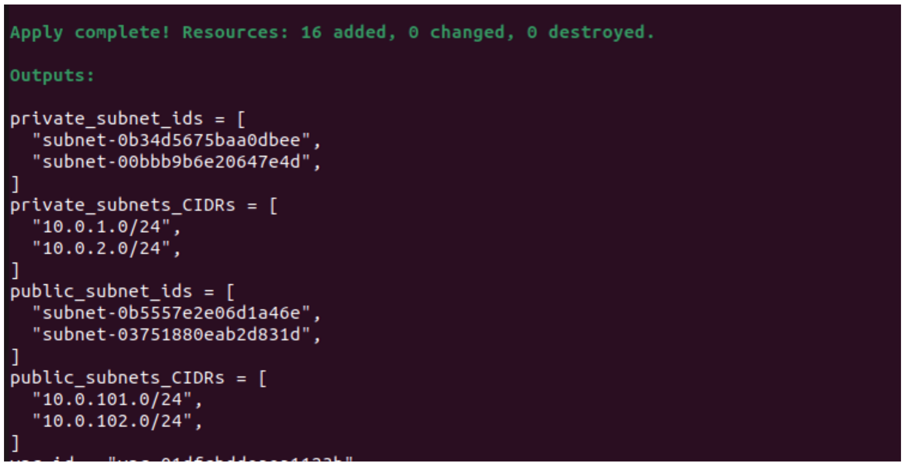
```
terraform init
```

- Validate Terraform code

```
terraform validate
```

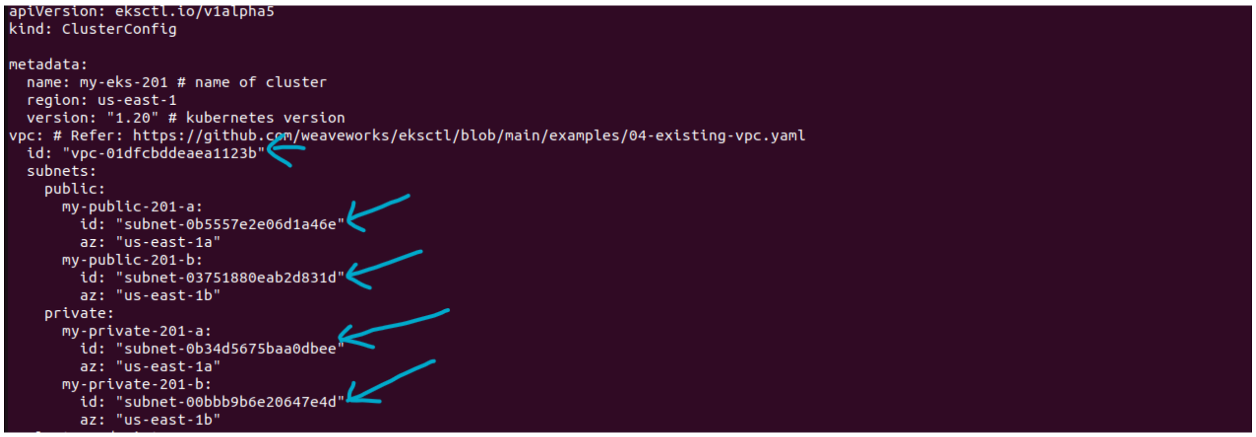
- Apply the terraform config by “terraform apply” and then enter “yes”. Please note down public subnet ID, private subnet ID and VPC ID. Terraform output variables displays this info after the resources are created.

```
terraform apply
```



```
Apply complete! Resources: 16 added, 0 changed, 0 destroyed.

Outputs:

private_subnet_ids = [
  "subnet-0b34d5675baa0dbee",
  "subnet-00bbb9b6e20647e4d",
]
private_subnets_CIDRs = [
  "10.0.1.0/24",
  "10.0.2.0/24",
]
public_subnet_ids = [
  "subnet-0b5557e2e06d1a46e",
  "subnet-03751880eab2d831d",
]
public_subnets_CIDRs = [
  "10.0.101.0/24",
  "10.0.102.0/24",
]
```

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



```
apiVersion: eksctl.io/v1alpha5
kind: ClusterConfig

metadata:
  name: my-eks-201 # name of cluster
  region: us-east-1
  version: "1.20" # kubernetes version
vpc: # Refer: https://github.com/weaveworks/eksctl/blob/main/examples/04-existing-vpc.yaml
  id: "vpc-01dfcbddeaea1123b"
  subnets:
    public:
      my-public-201-a:
        id: "subnet-0b5557e2e06d1a46e"
        az: "us-east-1a"
      my-public-201-b:
        id: "subnet-03751880eab2d831d"
        az: "us-east-1b"
    private:
      my-private-201-a:
        id: "subnet-0b34d5675baa0dbee"
        az: "us-east-1a"
      my-private-201-b:
        id: "subnet-00bbb9b6e20647e4d"
        az: "us-east-1b"
```

- 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/ms-sourcetech/upg-loadme-app
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
```

- Go to parent directory

```
cd ..
```

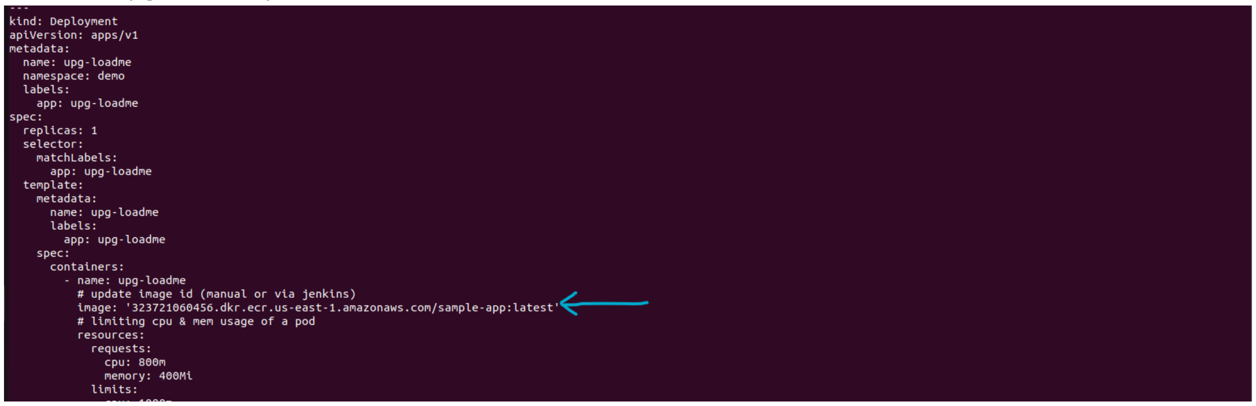
- Create nodegroup with taint.

```
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”.



```
kind: Deployment
apiVersion: apps/v1
metadata:
  name: upg-loadme
  namespace: demo
  labels:
    app: upg-loadme
spec:
  replicas: 1
  selector:
    matchLabels:
      app: upg-loadme
  template:
    metadata:
      name: upg-loadme
      labels:
        app: upg-loadme
    spec:
      containers:
        - name: upg-loadme
          # update image id (manual or via jenkins)
          image: '323721060456.dkr.ecr.us-east-1.amazonaws.com/sample-app:latest'
          # limiting cpu & mem usage of a pod
          resources:
            requests:
              cpu: 800m
              memory: 400Mi
            limits:
```

- 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.

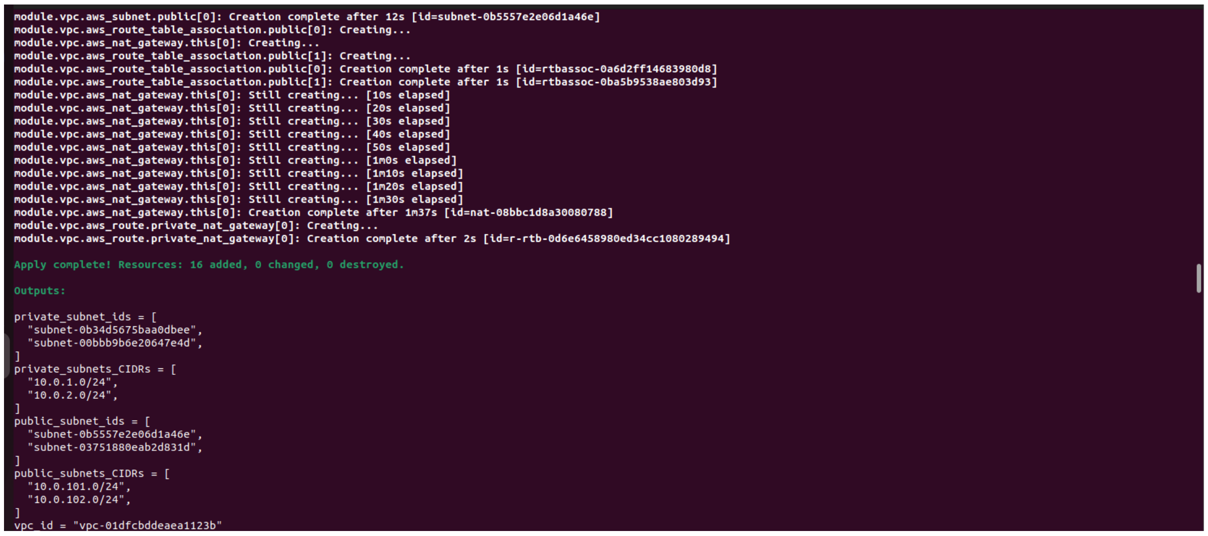
```
kubectl get hpa -n demo -w
```

- 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-.*"})
```

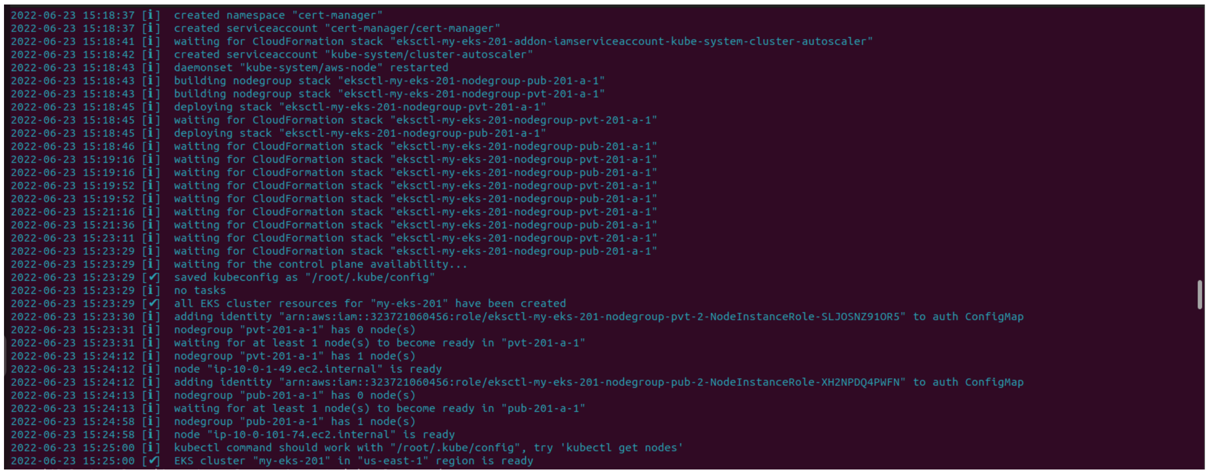
Results and Proofs

- Terraform infrastructure.



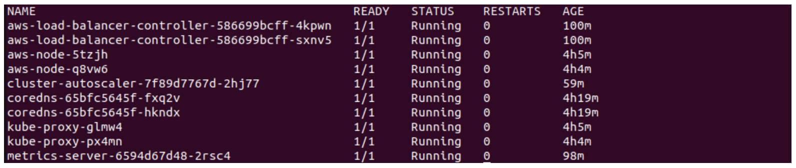
```
module.vpc.aws_subnet.public[0]: Creation complete after 12s [id=subnet-0b5557e2e06d1a46e]
module.vpc.aws_route_table_association.public[0]: Creating...
module.vpc.aws_nat_gateway.this[0]: Creating...
module.vpc.aws_route_table_association.public[1]: Creating...
module.vpc.aws_route_table_association.public[0]: Creation complete after 1s [id=rtbassoc-0a6d2ff14683980d8]
module.vpc.aws_route_table_association.public[1]: Creation complete after 1s [id=rtbassoc-0ba5b9538ae803d93]
module.vpc.aws_nat_gateway.this[0]: Still creating... [10s elapsed]
module.vpc.aws_nat_gateway.this[0]: Still creating... [20s elapsed]
module.vpc.aws_nat_gateway.this[0]: Still creating... [30s elapsed]
module.vpc.aws_nat_gateway.this[0]: Still creating... [40s elapsed]
module.vpc.aws_nat_gateway.this[0]: Still creating... [50s elapsed]
module.vpc.aws_nat_gateway.this[0]: Still creating... [1m0s elapsed]
module.vpc.aws_nat_gateway.this[0]: Still creating... [1m10s elapsed]
module.vpc.aws_nat_gateway.this[0]: Still creating... [1m20s elapsed]
module.vpc.aws_nat_gateway.this[0]: Still creating... [1m30s elapsed]
module.vpc.aws_nat_gateway.this[0]: Creation complete after 1m37s [id=nat-08bbc1d8a30080788]
module.vpc.aws_route.private_nat_gateway[0]: Creating...
module.vpc.aws_route.private_nat_gateway[0]: Creation complete after 2s [id=r-rtb-0d6e6458980ed34cc1080289494]

Apply complete! Resources: 16 added, 0 changed, 0 destroyed.

Outputs:

private_subnet_ids = [
  "subnet-0b34d5675baa0dbee",
  "subnet-00bbb9b6e20647e4d",
]
private_subnets_CIDRs = [
  "10.0.1.0/24",
  "10.0.2.0/24",
]
public_subnet_ids = [
  "subnet-0b5557e2e06d1a46e",
  "subnet-03751880eab2d831d",
]
public_subnets_CIDRs = [
  "10.0.101.0/24",
  "10.0.102.0/24",
]
vpc_id = "vpc-01dfcbddeaea1123b"
```

- EKS cluster created successfully using EKSCTL



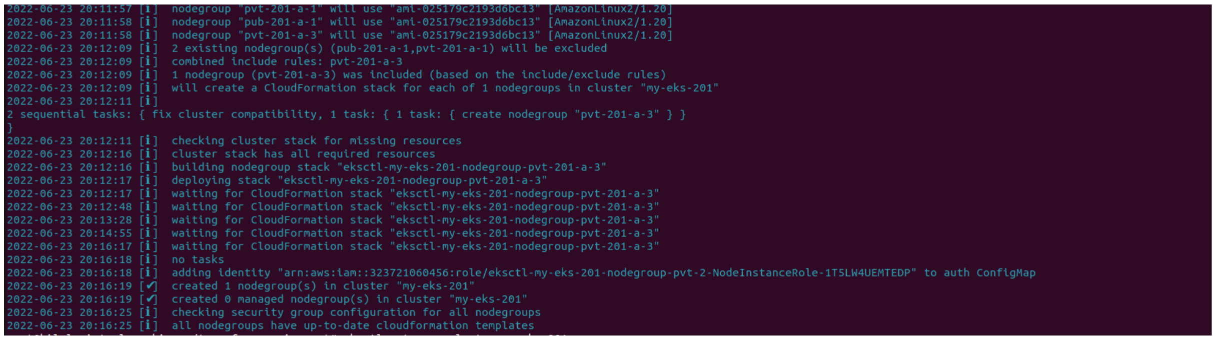
```
2022-06-23 15:18:37 [i]  created namespace "cert-manager"
2022-06-23 15:18:37 [i]  created serviceaccount "cert-manager/cert-manager"
2022-06-23 15:18:41 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-addon-iamserviceaccount-kube-system-cluster-autoscaler"
2022-06-23 15:18:42 [i]  created serviceaccount "kube-system/cluster-autoscaler"
2022-06-23 15:18:43 [i]  daemonset "kube-system/aws-node" restarted
2022-06-23 15:18:43 [i]  building nodegroup stack "eksctl-my-eks-201-nodegroup-pub-201-a-1"
2022-06-23 15:18:43 [i]  building nodegroup stack "eksctl-my-eks-201-nodegroup-pvt-201-a-1"
2022-06-23 15:18:45 [i]  deploying stack "eksctl-my-eks-201-nodegroup-pvt-201-a-1"
2022-06-23 15:18:45 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pvt-201-a-1"
2022-06-23 15:18:45 [i]  deploying stack "eksctl-my-eks-201-nodegroup-pub-201-a-1"
2022-06-23 15:18:46 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pub-201-a-1"
2022-06-23 15:19:16 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pvt-201-a-1"
2022-06-23 15:19:16 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pub-201-a-1"
2022-06-23 15:19:52 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pvt-201-a-1"
2022-06-23 15:19:52 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pub-201-a-1"
2022-06-23 15:21:16 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pvt-201-a-1"
2022-06-23 15:21:36 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pub-201-a-1"
2022-06-23 15:23:11 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pvt-201-a-1"
2022-06-23 15:23:29 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pub-201-a-1"
2022-06-23 15:23:29 [i]  waiting for the control plane availability...
2022-06-23 15:23:29 [✔]  saved kubeconfig as "/root/.kube/config"
2022-06-23 15:23:29 [i]  no tasks
2022-06-23 15:23:29 [✔]  all EKS cluster resources for "my-eks-201" have been created
2022-06-23 15:23:30 [i]  adding identity "arn:aws:iam::323721060456:role/eksctl-my-eks-201-nodegroup-pvt-2-NodeInstanceRole-SLJOSNZ91OR5" to auth ConfigMap
2022-06-23 15:23:31 [i]  nodegroup "pvt-201-a-1" has 0 node(s)
2022-06-23 15:23:31 [i]  waiting for at least 1 node(s) to become ready in "pvt-201-a-1"
2022-06-23 15:24:12 [i]  nodegroup "pvt-201-a-1" has 1 node(s)
2022-06-23 15:24:12 [i]  node "ip-10-0-1-49.ec2.internal" is ready
2022-06-23 15:24:12 [i]  adding identity "arn:aws:iam::323721060456:role/eksctl-my-eks-201-nodegroup-pub-2-NodeInstanceRole-XH2NPDQ4PWFN" to auth ConfigMap
2022-06-23 15:24:13 [i]  nodegroup "pub-201-a-1" has 0 node(s)
2022-06-23 15:24:13 [i]  waiting for at least 1 node(s) to become ready in "pub-201-a-1"
2022-06-23 15:24:58 [i]  nodegroup "pub-201-a-1" has 1 node(s)
2022-06-23 15:24:58 [i]  node "ip-10-0-101-74.ec2.internal" is ready
2022-06-23 15:25:00 [i]  kubectl command should work with "/root/.kube/config", try 'kubectl get nodes'
2022-06-23 15:25:00 [✔]  EKS cluster "my-eks-201" in "us-east-1" region is ready
```

- Add-ons installed.

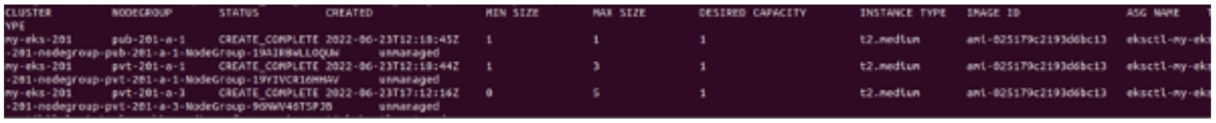


```
NAME                                           READY   STATUS    RESTARTS   AGE
aws-load-balancer-controller-586699bcff-4kpwn   1/1     Running   0          100m
aws-load-balancer-controller-586699bcff-sxnv5   1/1     Running   0          100m
aws-node-5tzjh                                  1/1     Running   0          4h5m
aws-node-q8vw6                                  1/1     Running   0          4h4m
cluster-autoscaler-7f89d7767d-2hj77             1/1     Running   0          59m
coredns-65bfc5645f-fxq2v                        1/1     Running   0          4h19m
coredns-65bfc5645f-hkndx                        1/1     Running   0          4h19m
kube-proxy-glmw4                                1/1     Running   0          4h5m
kube-proxy-px4mn                                1/1     Running   0          4h4m
metrics-server-6594d67d48-2rsc4                 1/1     Running   0          98m
```

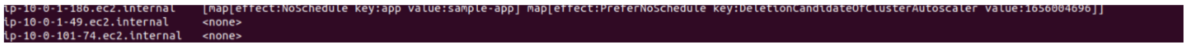
- 3rd Node Group with Taints added to EKS.



```
2022-06-23 20:11:57 [i]  nodegroup "pvt-201-a-1" will use "ami-025179c2193d6bc13" [AmazonLinux2/1.20]
2022-06-23 20:11:58 [i]  nodegroup "pub-201-a-1" will use "ami-025179c2193d6bc13" [AmazonLinux2/1.20]
2022-06-23 20:11:58 [i]  nodegroup "pvt-201-a-3" will use "ami-025179c2193d6bc13" [AmazonLinux2/1.20]
2022-06-23 20:12:09 [i]  2 existing nodegroup(s) (pub-201-a-1,pvt-201-a-1) will be excluded
2022-06-23 20:12:09 [i]  combined include rules: pvt-201-a-3
2022-06-23 20:12:09 [i]  1 nodegroup (pvt-201-a-3) was included (based on the include/exclude rules)
2022-06-23 20:12:09 [i]  will create a CloudFormation stack for each of 1 nodegroups in cluster "my-eks-201"
2022-06-23 20:12:11 [i]
2 sequential tasks: { fix cluster compatibility, 1 task: { 1 task: { create nodegroup "pvt-201-a-3" } }
}
2022-06-23 20:12:11 [i]  checking cluster stack for missing resources
2022-06-23 20:12:16 [i]  cluster stack has all required resources
2022-06-23 20:12:16 [i]  building nodegroup stack "eksctl-my-eks-201-nodegroup-pvt-201-a-3"
2022-06-23 20:12:17 [i]  deploying stack "eksctl-my-eks-201-nodegroup-pvt-201-a-3"
2022-06-23 20:12:17 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pvt-201-a-3"
2022-06-23 20:12:48 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pvt-201-a-3"
2022-06-23 20:13:28 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pvt-201-a-3"
2022-06-23 20:14:55 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pvt-201-a-3"
2022-06-23 20:16:17 [i]  waiting for CloudFormation stack "eksctl-my-eks-201-nodegroup-pvt-201-a-3"
2022-06-23 20:16:18 [i]  no tasks
2022-06-23 20:16:18 [i]  adding identity "arn:aws:iam::323721060456:role/eksctl-my-eks-201-nodegroup-pvt-2-NodeInstanceRole-1T5LW4UEMTEDP" to auth ConfigMap
2022-06-23 20:16:19 [✓]  created 1 nodegroup(s) in cluster "my-eks-201"
2022-06-23 20:16:19 [✓]  created 0 managed nodegroup(s) in cluster "my-eks-201"
2022-06-23 20:16:25 [i]  checking security group configuration for all nodegroups
2022-06-23 20:16:25 [i]  all nodegroups have up-to-date cloudformation templates
```

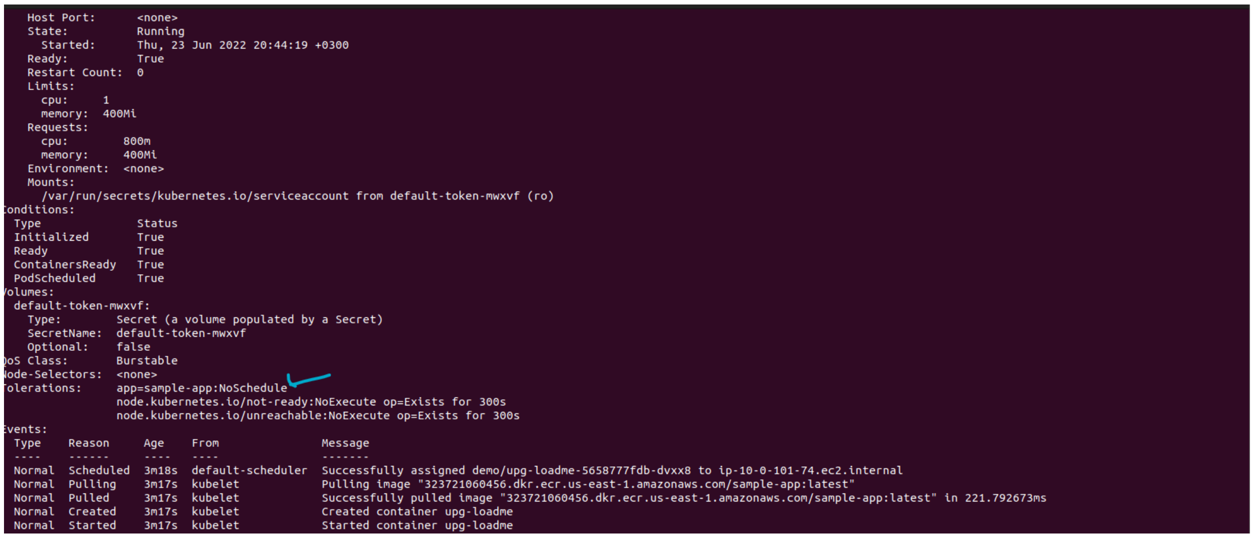


```
CLUSTER         NODEGROUP       STATUS          CREATED                 MIN SIZE        MAX SIZE        DESIRED CAPACITY        INSTANCE TYPE   IMAGE ID                ASG NAME        T
YPE
my-eks-201      pub-201-a-1     CREATE_COMPLETE 2022-06-23T12:18:45Z    1               1               1                       t2.medium       ami-025179c2193d6bc13   eksctl-my-eks
-201-nodegroup-pub-201-a-1-NodeGroup-19A3RBWLLOQUW      unmanaged
my-eks-201      pvt-201-a-1     CREATE_COMPLETE 2022-06-23T12:18:44Z    1               3               1                       t2.medium       ami-025179c2193d6bc13   eksctl-my-eks
-201-nodegroup-pvt-201-a-1-NodeGroup-19YIVCR16MMAV      unmanaged
my-eks-201      pvt-201-a-3     CREATE_COMPLETE 2022-06-23T17:12:16Z    0               5               1                       t2.medium       ami-025179c2193d6bc13   eksctl-my-eks
-201-nodegroup-pvt-201-a-3-NodeGroup-90NWV48T5PJB       unmanaged
```



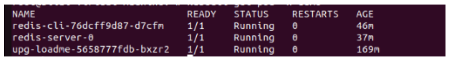
```
ip-10-0-1-186.ec2.internal    [map[effect:NoSchedule key:app value:sample-app] map[effect:PreferNoSchedule key:DeletionCandidateOfClusterAutoscaler value:1656004696]]
ip-10-0-1-49.ec2.internal     <none>
ip-10-0-101-74.ec2.internal   <none>
```

- Toleration applied to the deployment pods.



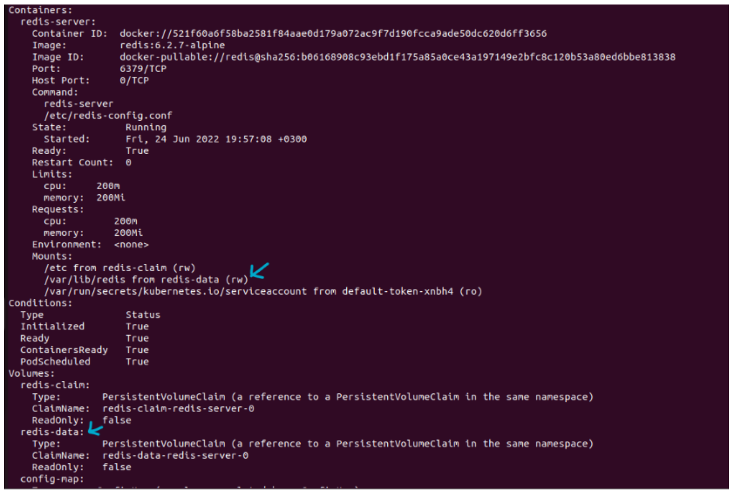
```
    Host Port:      <none>
    State:          Running
      Started:      Thu, 23 Jun 2022 20:44:19 +0300
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     1
      memory:  400Mi
    Requests:
      cpu:        800m
      memory:     400Mi
    Environment:  <none>
    Mounts:
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-mwxvf (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  default-token-mwxvf:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-mwxvf
    Optional:    false
QoS Class:       Burstable
Node-Selectors:  <none>
Tolerations:     app=sample-app:NoSchedule
                 node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason     Age    From               Message
  ----    ------     ----   ----               -------
  Normal  Scheduled  3m18s  default-scheduler  Successfully assigned demo/upg-loadme-5658777fdb-dvxx8 to ip-10-0-101-74.ec2.internal
  Normal  Pulling    3m17s  kubelet            Pulling image "323721060456.dkr.ecr.us-east-1.amazonaws.com/sample-app:latest"
  Normal  Pulled     3m17s  kubelet            Successfully pulled image "323721060456.dkr.ecr.us-east-1.amazonaws.com/sample-app:latest" in 221.792673ms
  Normal  Created    3m17s  kubelet            Created container upg-loadme
  Normal  Started    3m17s  kubelet            Started container upg-loadme
```

- Redis-Server and Redis-cli installed successfully.

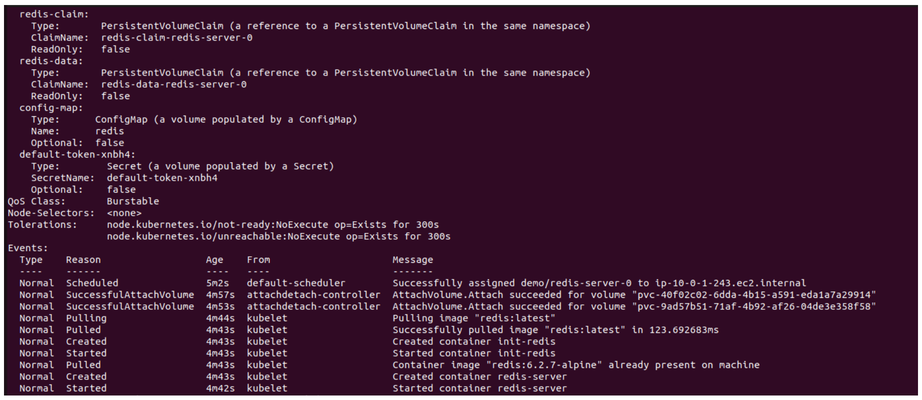


```
NAME                          READY   STATUS    RESTARTS   AGE
redis-cli-76dcff9d87-d7cfm    1/1     Running   0          46m
redis-server-0                1/1     Running   0          37m
upg-loadme-5658777fdb-bxzr2   1/1     Running   0          169m
```

- PVC by EBS mounted to “redis-server”

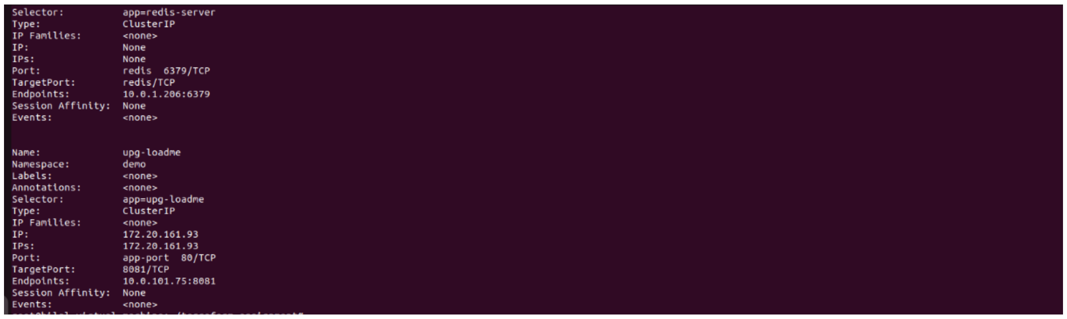


```
Containers:
  redis-server:
    Container ID:  docker://521f60a6f58ba2581f84aae0d179a072ac9f7d190fcca9ade50dc620d6ff3656
    Image:         redis:6.2.7-alpine
    Image ID:      docker-pullable://redis@sha256:b06168908c93ebd1f175a85a0ce43a197149e2bfc8c120b53a80ed6bbe813838
    Port:          6379/TCP
    Host Port:     0/TCP
    Command:
      redis-server
      /etc/redis-config.conf
    State:          Running
      Started:      Fri, 24 Jun 2022 19:57:08 +0300
    Ready:          True
    Restart Count:  0
    Limits:
      cpu:     200m
      memory:  200Mi
    Requests:
      cpu:        200m
      memory:     200Mi
    Environment:  <none>
    Mounts:
      /etc from redis-claim (rw)
      /var/lib/redis from redis-data (rw)
      /var/run/secrets/kubernetes.io/serviceaccount from default-token-xnbh4 (ro)
Conditions:
  Type              Status
  Initialized       True
  Ready             True
  ContainersReady   True
  PodScheduled      True
Volumes:
  redis-claim:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  redis-claim-redis-server-0
    ReadOnly:   false
  redis-data:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  redis-data-redis-server-0
    ReadOnly:   false
  config-map:
```



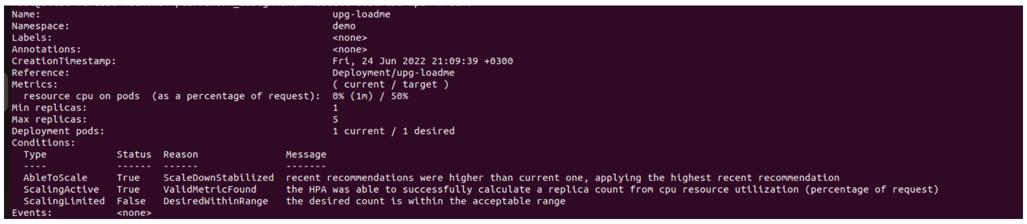
```
  redis-claim:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  redis-claim-redis-server-0
    ReadOnly:   false
  redis-data:
    Type:       PersistentVolumeClaim (a reference to a PersistentVolumeClaim in the same namespace)
    ClaimName:  redis-data-redis-server-0
    ReadOnly:   false
  config-map:
    Type:      ConfigMap (a volume populated by a ConfigMap)
    Name:      redis
    Optional:  false
  default-token-xnbh4:
    Type:        Secret (a volume populated by a Secret)
    SecretName:  default-token-xnbh4
    Optional:    false
QoS Class:       Burstable
Node-Selectors:  <none>
Tolerations:     node.kubernetes.io/not-ready:NoExecute op=Exists for 300s
                 node.kubernetes.io/unreachable:NoExecute op=Exists for 300s
Events:
  Type    Reason                  Age    From                     Message
  ----    ------                  ----   ----                     -------
  Normal  Scheduled               5m2s   default-scheduler        Successfully assigned demo/redis-server-0 to ip-10-0-1-243.ec2.internal
  Normal  SuccessfulAttachVolume  4m57s  attachdetach-controller  AttachVolume.Attach succeeded for volume "pvc-40f02c02-6dda-4b15-a591-eda1a7a29914"
  Normal  SuccessfulAttachVolume  4m53s  attachdetach-controller  AttachVolume.Attach succeeded for volume "pvc-9ad57b51-71af-4b92-af26-04de3e358f58"
  Normal  Pulling                 4m44s  kubelet                  Pulling image "redis:latest"
  Normal  Pulled                  4m43s  kubelet                  Successfully pulled image "redis:latest" in 123.692683ms
  Normal  Created                 4m43s  kubelet                  Created container init-redis
  Normal  Started                 4m43s  kubelet                  Started container init-redis
  Normal  Pulled                  4m43s  kubelet                  Container image "redis:6.2.7-alpine" already present on machine
  Normal  Created                 4m43s  kubelet                  Created container redis-server
  Normal  Started                 4m42s  kubelet                  Started container redis-server
```

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



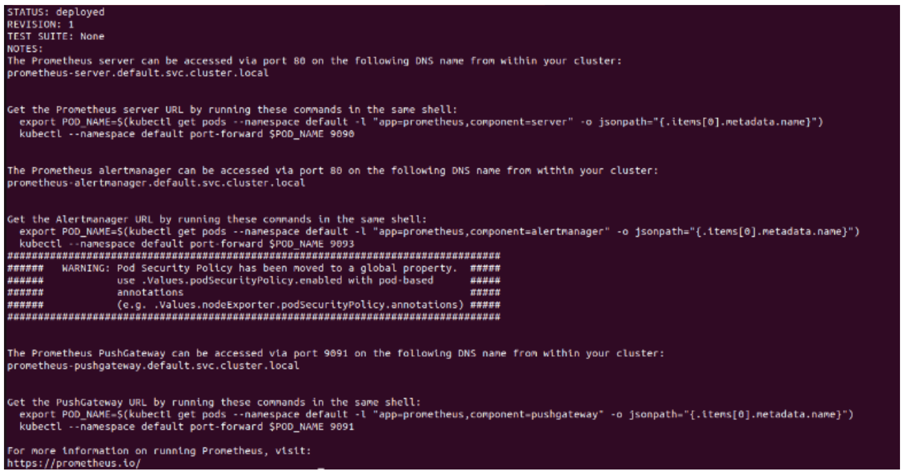
```
Selector:          app=redis-server
Type:              ClusterIP
IP Families:       <none>
IP:                None
IPs:               None
Port:              redis  6379/TCP
TargetPort:        redis/TCP
Endpoints:         10.0.1.206:6379
Session Affinity:  None
Events:            <none>


Name:              upg-loadme
Namespace:         demo
Labels:            <none>
Annotations:       <none>
Selector:          app=upg-loadme
Type:              ClusterIP
IP Families:       <none>
IP:                172.20.161.93
IPs:               172.20.161.93
Port:              app-port  80/TCP
TargetPort:        8081/TCP
Endpoints:         10.0.101.75:8081
Session Affinity:  None
Events:            <none>
```

- HPA deployed on upg-loadme.



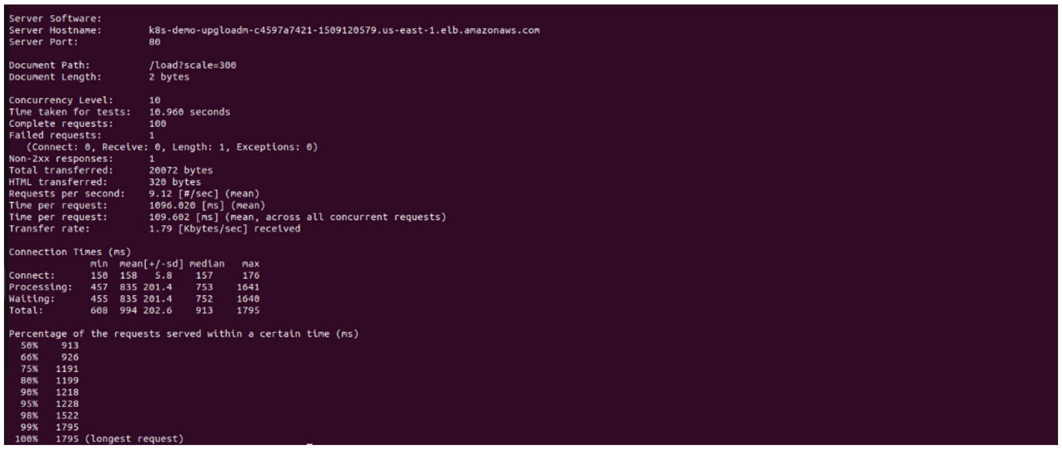
```
Name:                                                  upg-loadme
Namespace:                                             demo
Labels:                                                <none>
Annotations:                                           <none>
CreationTimestamp:                                     Fri, 24 Jun 2022 21:09:39 +0300
Reference:                                             Deployment/upg-loadme
Metrics:                                               ( current / target )
  resource cpu on pods  (as a percentage of request):  0% (1m) / 50%
Min replicas:                                          1
Max replicas:                                          5
Deployment pods:                                       1 current / 1 desired
Conditions:
  Type            Status  Reason               Message
  ----            ------  ------               -------
  AbleToScale     True    ScaleDownStabilized  recent recommendations were higher than current one, applying the highest recent recommendation
  ScalingActive   True    ValidMetricFound     the HPA was able to successfully calculate a replica count from cpu resource utilization (percentage of request)
  ScalingLimited  False   DesiredWithinRange   the desired count is within the acceptable range
Events:           <none>
```

- Prometheus installed.



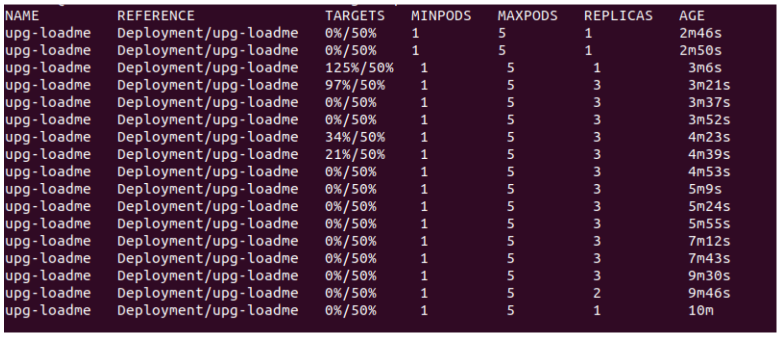
```
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
The Prometheus server can be accessed via port 80 on the following DNS name from within your cluster:
prometheus-server.default.svc.cluster.local


Get the Prometheus server URL by running these commands in the same shell:
  export POD_NAME=$(kubectl get pods --namespace default -l "app=prometheus,component=server" -o jsonpath="{.items[0].metadata.name}")
  kubectl --namespace default port-forward $POD_NAME 9090


The Prometheus alertmanager can be accessed via port 80 on the following DNS name from within your cluster:
prometheus-alertmanager.default.svc.cluster.local


Get the Alertmanager URL by running these commands in the same shell:
  export POD_NAME=$(kubectl get pods --namespace default -l "app=prometheus,component=alertmanager" -o jsonpath="{.items[0].metadata.name}")
  kubectl --namespace default port-forward $POD_NAME 9093
#################################################################################
######   WARNING: Pod Security Policy has been moved to a global property.  #####
######            use .Values.podSecurityPolicy.enabled with pod-based      #####
######            annotations                                               #####
######            (e.g. .Values.nodeExporter.podSecurityPolicy.annotations) #####
#################################################################################


The Prometheus PushGateway can be accessed via port 9091 on the following DNS name from within your cluster:
prometheus-pushgateway.default.svc.cluster.local


Get the PushGateway URL by running these commands in the same shell:
  export POD_NAME=$(kubectl get pods --namespace default -l "app=prometheus,component=pushgateway" -o jsonpath="{.items[0].metadata.name}")
  kubectl --namespace default port-forward $POD_NAME 9091

For more information on running Prometheus, visit:
https://prometheus.io/
```

- Generate load from Apache Benchmark utility.



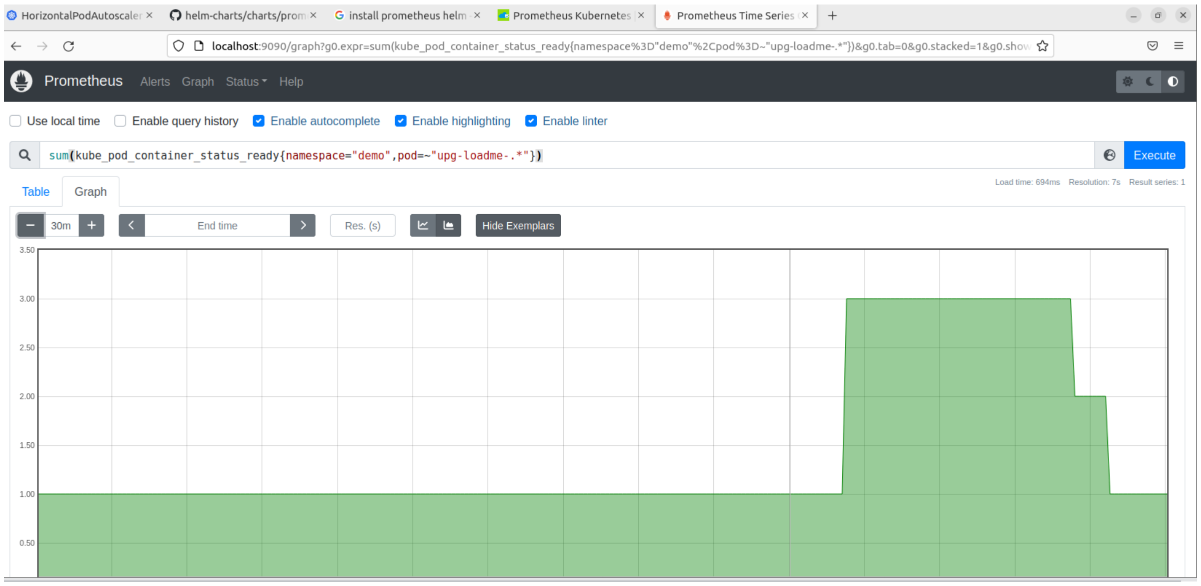
```
Server Software:
Server Hostname:        k8s-demo-upgloadm-c4597a7421-1509120579.us-east-1.elb.amazonaws.com
Server Port:            80

Document Path:          /load?scale=300
Document Length:        2 bytes

Concurrency Level:      10
Time taken for tests:   10.960 seconds
Complete requests:      100
Failed requests:        1
   (Connect: 0, Receive: 0, Length: 1, Exceptions: 0)
Non-2xx responses:      1
Total transferred:      20072 bytes
HTML transferred:       320 bytes
Requests per second:    9.12 [#/sec] (mean)
Time per request:       1096.020 [ms] (mean)
Time per request:       109.602 [ms] (mean, across all concurrent requests)
Transfer rate:          1.79 [Kbytes/sec] received

Connection Times (ms)
              min  mean[+/-sd] median   max
Connect:      150  158   5.8    157     176
Processing:   457  835 201.4    753    1641
Waiting:      455  835 201.4    752    1640
Total:        608  994 202.6    913    1795

Percentage of the requests served within a certain time (ms)
  50%    913
  66%    926
  75%   1191
  80%   1199
  90%   1218
  95%   1228
  98%   1522
  99%   1795
 100%   1795 (longest request)
```

- HPA status during load test.



```
NAME         REFERENCE               TARGETS   MINPODS   MAXPODS   REPLICAS   AGE
upg-loadme   Deployment/upg-loadme   0%/50%    1         5         1          2m46s
upg-loadme   Deployment/upg-loadme   0%/50%    1         5         1          2m50s
upg-loadme   Deployment/upg-loadme   125%/50%   1         5         1          3m6s
upg-loadme   Deployment/upg-loadme   97%/50%    1         5         3          3m21s
upg-loadme   Deployment/upg-loadme   0%/50%     1         5         3          3m37s
upg-loadme   Deployment/upg-loadme   0%/50%     1         5         3          3m52s
upg-loadme   Deployment/upg-loadme   34%/50%    1         5         3          4m23s
upg-loadme   Deployment/upg-loadme   21%/50%    1         5         3          4m39s
upg-loadme   Deployment/upg-loadme   0%/50%     1         5         3          4m53s
upg-loadme   Deployment/upg-loadme   0%/50%     1         5         3          5m9s
upg-loadme   Deployment/upg-loadme   0%/50%     1         5         3          5m24s
upg-loadme   Deployment/upg-loadme   0%/50%     1         5         3          5m55s
upg-loadme   Deployment/upg-loadme   0%/50%     1         5         3          7m12s
upg-loadme   Deployment/upg-loadme   0%/50%     1         5         3          7m43s
upg-loadme   Deployment/upg-loadme   0%/50%     1         5         3          9m30s
upg-loadme   Deployment/upg-loadme   0%/50%     1         5         2          9m46s
upg-loadme   Deployment/upg-loadme   0%/50%     1         5         1          10m
```

- Prometheus Graph covering the load test period.


```
localhost:9090/graph?g0.expr=sum(kube_pod_container_status_ready{namespace%3D"demo"%2Cpod%3D~"upg-loadme-.*"})&g0.tab=0&g0.stacked=1&g0.show
```

```
sum(kube_pod_container_status_ready{namespace="demo",pod=~"upg-loadme-.*"})
```

Load time: 694ms Resolution: 7s Result series: 1

Bonus Task

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

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