# Cloud Native Voting Application on EKS

Github:-https://github.com/N4si/K8s-voting-app?tab=readme-ov-file#cloud-native-web-voting-application-with-kubernetes

Youtube Video to refer: Kubernetes Live Project: Deploy Cloud Native Voting Application on EKS - YouTube

This cloud-native web application is built using a mix of technologies. It's designed to be accessible to users via the internet, allowing them to vote for their preferred programming language out of six choices: C#, Python, JavaScript, Go, Java, and NodeJS.

**Technical Stack**

- **Frontend**: The frontend of this application is built using React and JavaScript. It provides a responsive and user-friendly interface for casting votes.
- **Backend and API**: The backend of this application is powered by Go (Golang). It serves as the API handling user voting requests. MongoDB is used as the database backend, configured with a replica set for data redundancy and high availability.

**Kubernetes Resources**

To deploy and manage this application effectively, we leverage Kubernetes and a variety of its resources:

- **Namespace**: Kubernetes namespaces are utilized to create isolated environments for different components of the application, ensuring separation and organization.
- **Secret**: Kubernetes secrets store sensitive information, such as API keys or credentials, required by the application securely.

- **Deployment**: Kubernetes deployments define how many instances of the application should run and provide instructions for updates and scaling.
- **Service**: Kubernetes services ensure that users can access the application by directing incoming traffic to the appropriate instances.
- **StatefulSet**: For components requiring statefulness, such as the MongoDB replica set, Kubernetes StatefulSets are employed to maintain order and unique identities.
- **PersistentVolume and PersistentVolumeClaim**: These Kubernetes resources manage the storage required for the application, ensuring data persistence and scalability.

**Learning Opportunities**

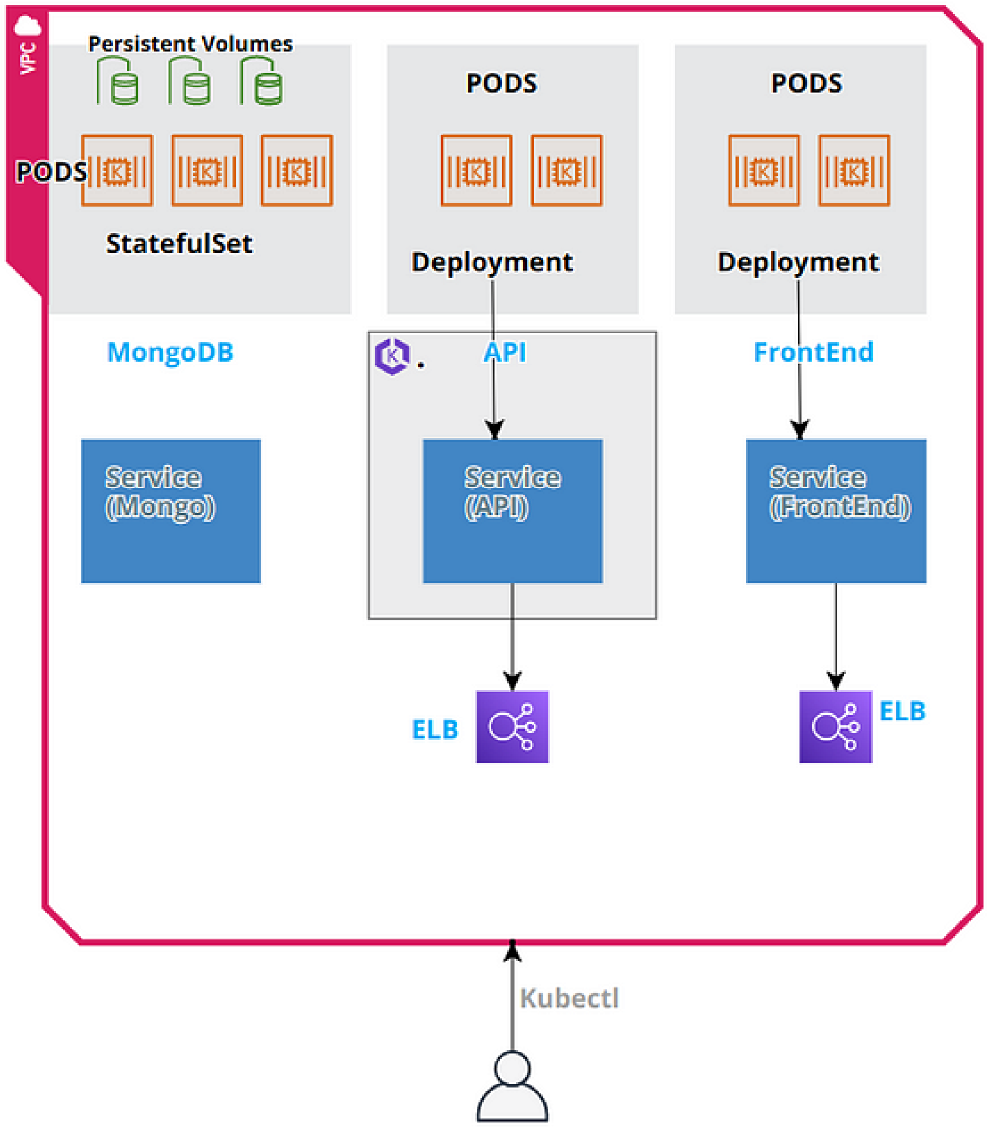
Creating and deploying this cloud-native web voting application with Kubernetes offers a valuable learning experience. Here are some key takeaways:

1. **Containerization**: Gain hands-on experience with containerization technologies like Docker for packaging applications and their dependencies.
2. **Kubernetes Orchestration**: Learn how to leverage Kubernetes to efficiently manage, deploy, and scale containerized applications in a production environment.
3. **Microservices Architecture**: Explore the benefits and challenges of a microservices architecture, where the frontend and backend are decoupled and independently scalable.
4. **Database Replication**: Understand how to set up and manage a MongoDB replica set for data redundancy and high availability.
5. **Security and Secrets Management**: Learn best practices for securing sensitive information using Kubernetes secrets.
6. **Stateful Applications**: Gain insights into the nuances of deploying stateful applications within a container orchestration environment.
7. **Persistent Storage**: Understand how Kubernetes manages and provisions persistent storage for applications with state.

By working through this project, you'll develop a deeper understanding of cloud-native application development, containerization, Kubernetes, and the various technologies involved in building and deploying modern web applications.

**Steps to Deploy**

**Architecture Diagram:**

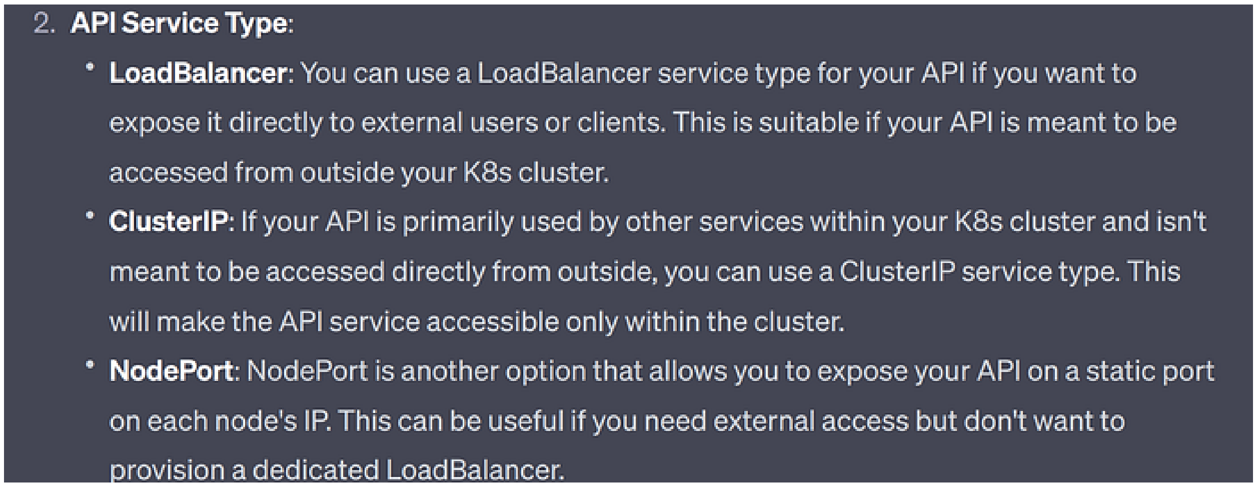


**MongoDb:**

We will be using three pods for our database mongodb service one pod will be primary which will be in read and write state while two pods will be secondary which will have complete replica of primary pod but only in read mode, the replica pods can become primary pod also in case primary pods fails so fault tolerance also achieve through it.

**API:**

We will create API through deployment and use two pods for this and will expose service through service type load balancer.



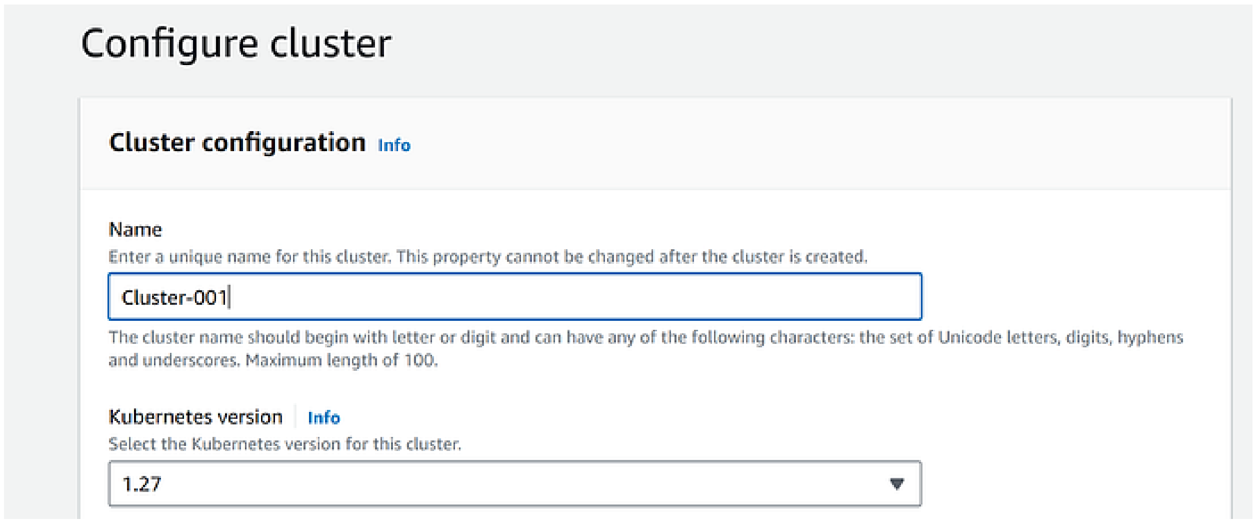
2. **API Service Type**:
    - **LoadBalancer**: You can use a LoadBalancer service type for your API if you want to expose it directly to external users or clients. This is suitable if your API is meant to be accessed from outside your K8s cluster.
    - **ClusterIP**: If your API is primarily used by other services within your K8s cluster and isn't meant to be accessed directly from outside, you can use a ClusterIP service type. This will make the API service accessible only within the cluster.
    - **NodePort**: NodePort is another option that allows you to expose your API on a static port on each node's IP. This can be useful if you need external access but don't want to provision a dedicated LoadBalancer.

**Front End:**

The APIs will get the data from front end, front end will also have two pods. We will also expose it through service type load balancer.

Lets start the project:

We will be using Amazon EKS to deploy the application on kubernetes:


Configure cluster

Cluster configuration

Name

Enter a unique name for this cluster. This property cannot be changed after the cluster is created.

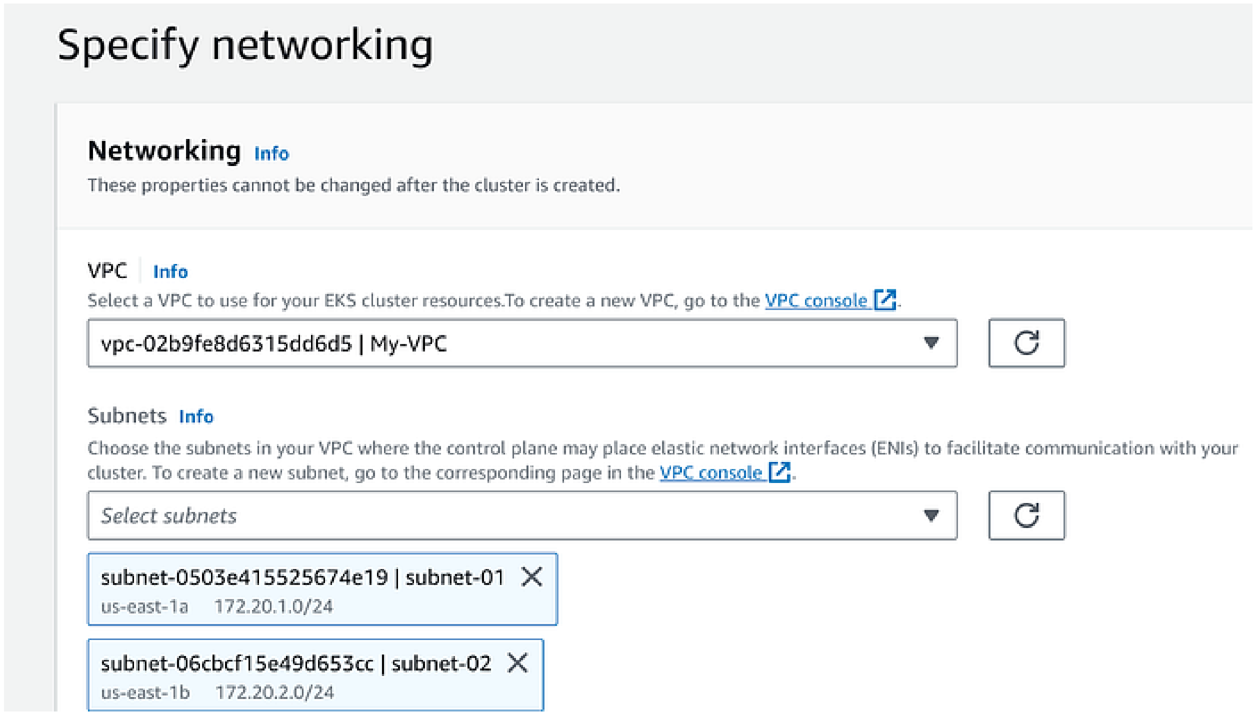
Cluster-001

The cluster name should begin with letter or digit and can have any of the following characters: the set of Unicode letters, digits, hyphens and underscores. Maximum length of 100.

Kubernetes version

Select the Kubernetes version for this cluster.

1.27


Specify networking

Networking

These properties cannot be changed after the cluster is created.

VPC

Select a VPC to use for your EKS cluster resources.To create a new VPC, go to the VPC console.

vpc-02b9fe8d6315dd6d5 | My-VPC

Subnets

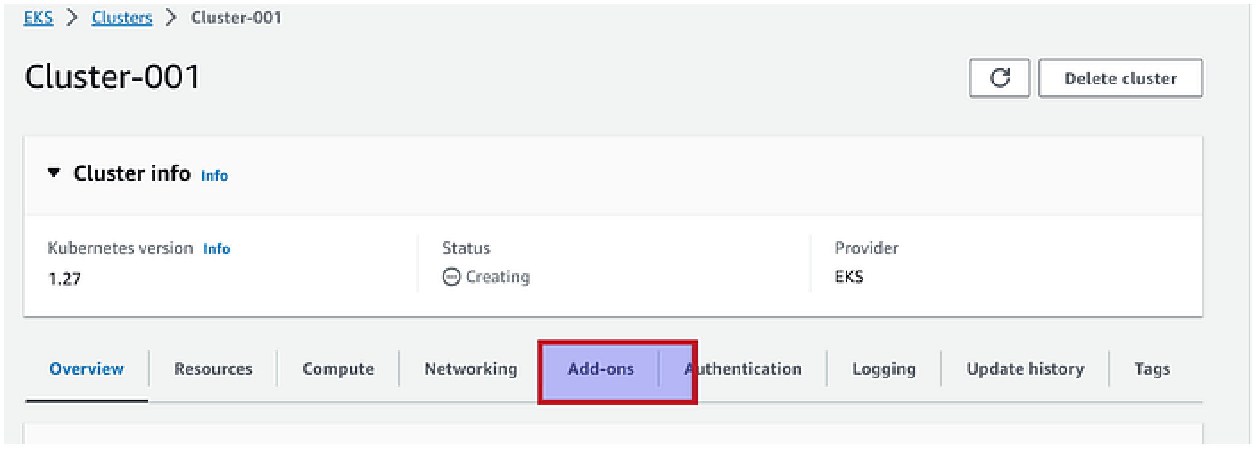
Choose the subnets in your VPC where the control plane may place elastic network interfaces (ENIs) to facilitate communication with your cluster. To create a new subnet, go to the corresponding page in the VPC console.

Select subnets

subnet-0503e415525674e19 | subnet-01
us-east-1a 172.20.1.0/24

subnet-06cbcf15e49d653cc | subnet-02
us-east-1b 172.20.2.0/24

Once you click on create cluster then select option add on:


EKS > Clusters > Cluster-001

Cluster-001

Delete cluster

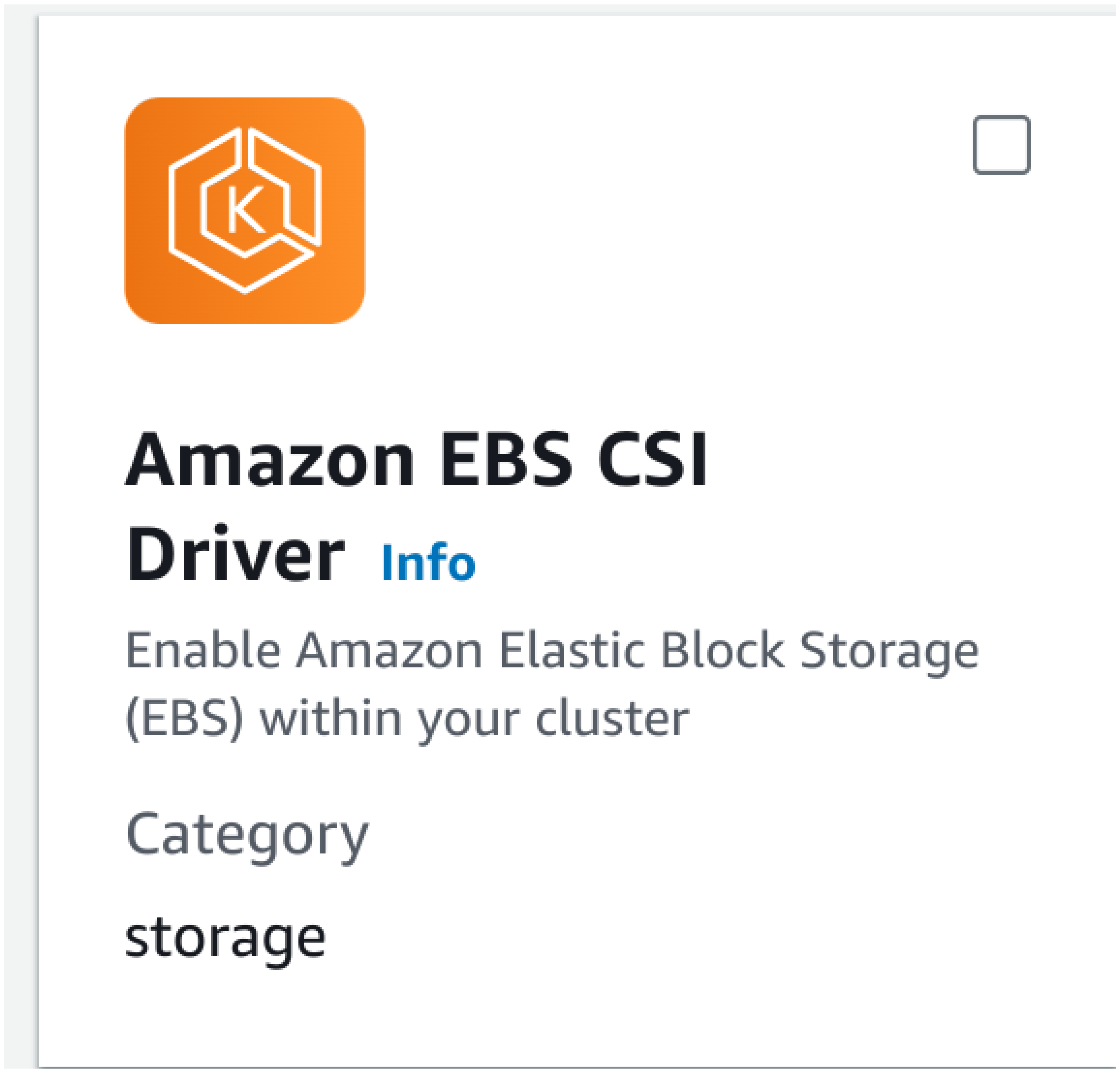
Cluster info Info

Kubernetes version Info: 1.27

Status: Creating

Provider: EKS

And search for amazon EBS add on:


Amazon EBS CSI Driver Info

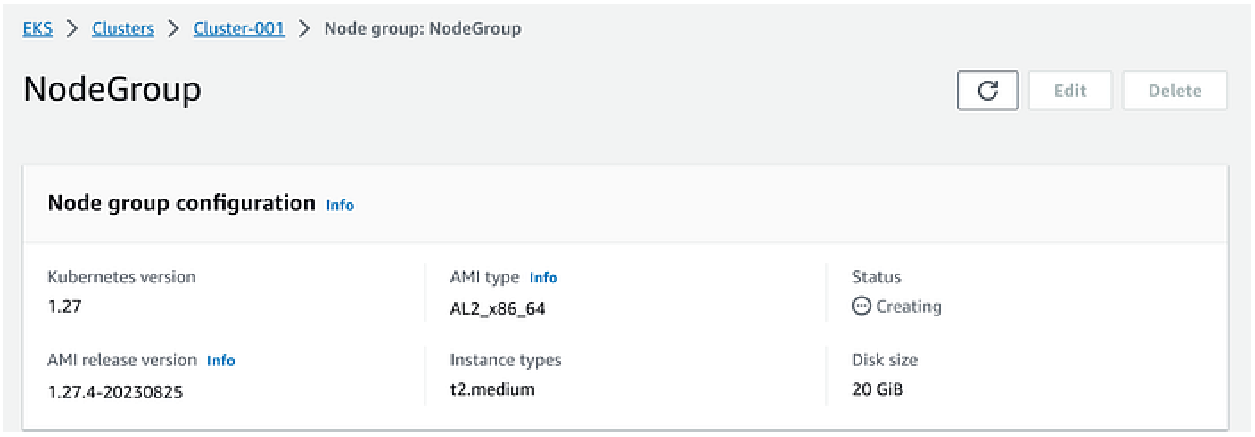
Enable Amazon Elastic Block Storage (EBS) within your cluster

Category

storage

This is to create EBS volume automatically for PODs in cluster.

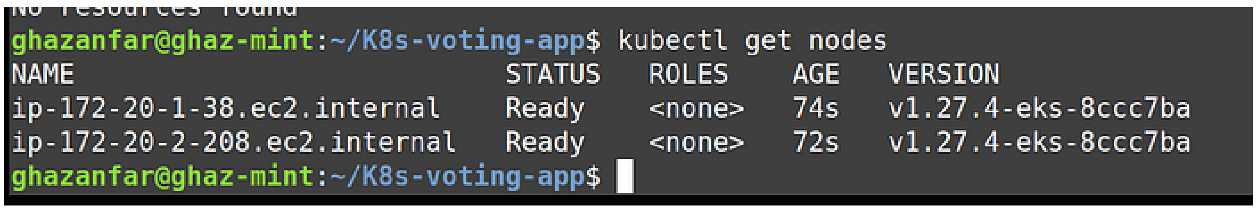
Once EBS added go to compute option to add node groups and select t2.medium instances there:



NodeGroup

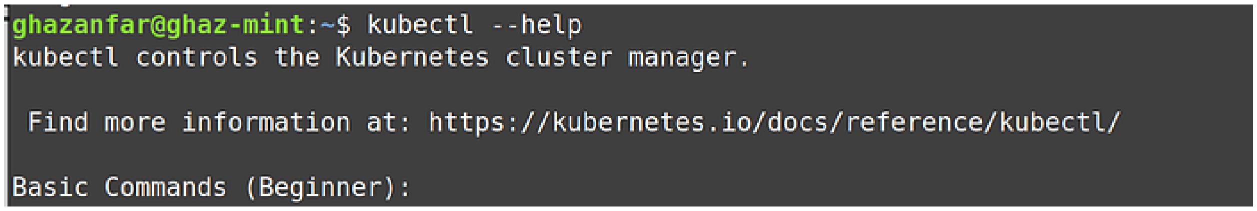
**Node group configuration** Info

| | | |
|---|---|---|
| Kubernetes version | AMI type Info | Status |
| 1.27 | AL2_x86_64 | Creating |
| AMI release version Info | Instance types | Disk size |
| 1.27.4-20230825 | t2.medium | 20 GiB |



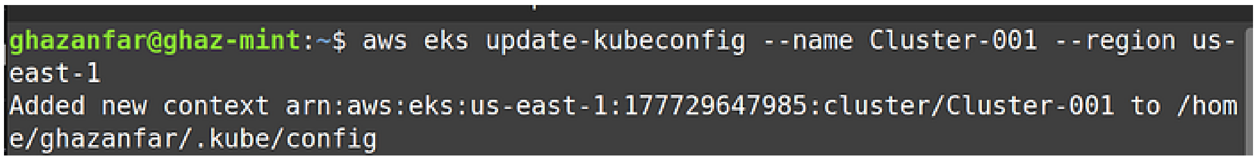
```
ghazanfar@ghaz-mint:~/K8s-voting-app$ kubectl get nodes
NAME                           STATUS   ROLES    AGE   VERSION
ip-172-20-1-38.ec2.internal    Ready    <none>   74s   v1.27.4-eks-8ccc7ba
ip-172-20-2-208.ec2.internal   Ready    <none>   72s   v1.27.4-eks-8ccc7ba
ghazanfar@ghaz-mint:~/K8s-voting-app$
```

Install kubectl on your local system to interact with kubernetes cluster:



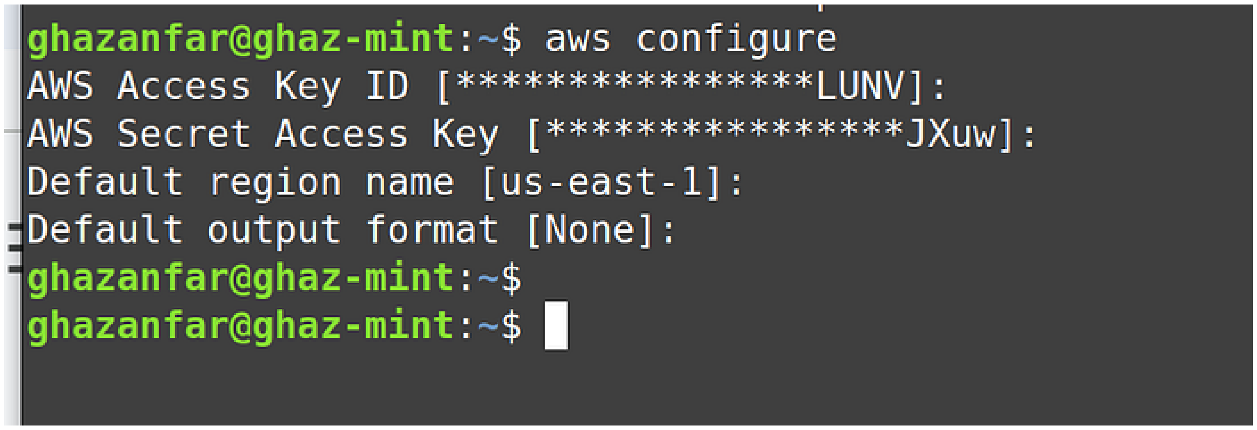
```
ghazanfar@ghaz-mint:~$ kubectl --help
kubectl controls the Kubernetes cluster manager.

 Find more information at: https://kubernetes.io/docs/reference/kubectl/

Basic Commands (Beginner):
```

Now add your cluster info in kubectl config file so that kubectl able to interact with kubernetes cluster:



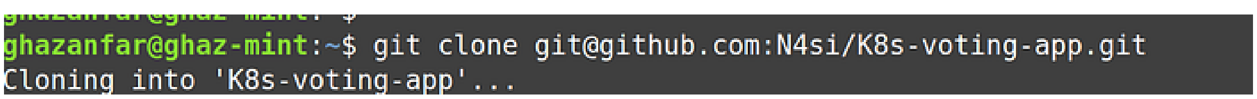
```
ghazanfar@ghaz-mint:~$ aws eks update-kubeconfig --name Cluster-001 --region us-
east-1
Added new context arn:aws:eks:us-east-1:177729647985:cluster/Cluster-001 to /hom
e/ghazanfar/.kube/config
```

Along with kubectl you should have aws cli in your local system so that it interact with AWS for that create IAM user in AWS and then create its secret and access keys and configure those keys in your system via aws configure:



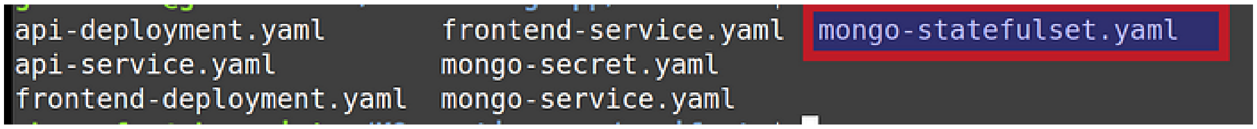
```
ghazanfar@ghaz-mint:~$ aws configure
AWS Access Key ID [****************LUNV]:
AWS Secret Access Key [****************JXuw]:
Default region name [us-east-1]:
Default output format [None]:
ghazanfar@ghaz-mint:~$
ghazanfar@ghaz-mint:~$ 
```

Now clone the manifest file from below github link:



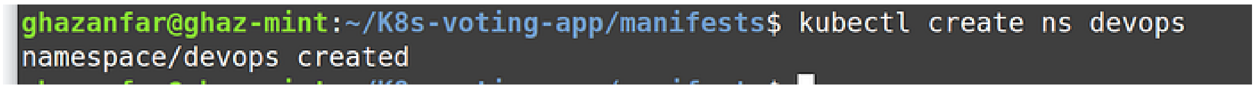
```
ghazanfar@ghaz-mint:~$ git clone git@github.com:N4si/K8s-voting-app.git
Cloning into 'K8s-voting-app'...
```

Our nodes are ready now we will create pods for mongo db through mongo manifest file:



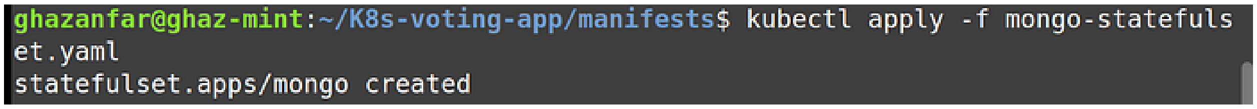
```
api-deployment.yaml       frontend-service.yaml  mongo-statefulset.yaml
api-service.yaml          mongo-secret.yaml
frontend-deployment.yaml  mongo-service.yaml
```

But first create the new namespace in cluster:

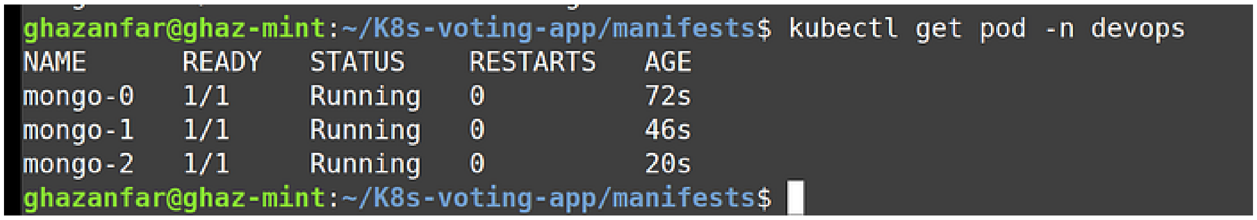


```
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$ kubectl create ns devops
namespace/devops created
```

Now apply the manifest file of mongo it will create three pods along with three persistent volumes of pods,

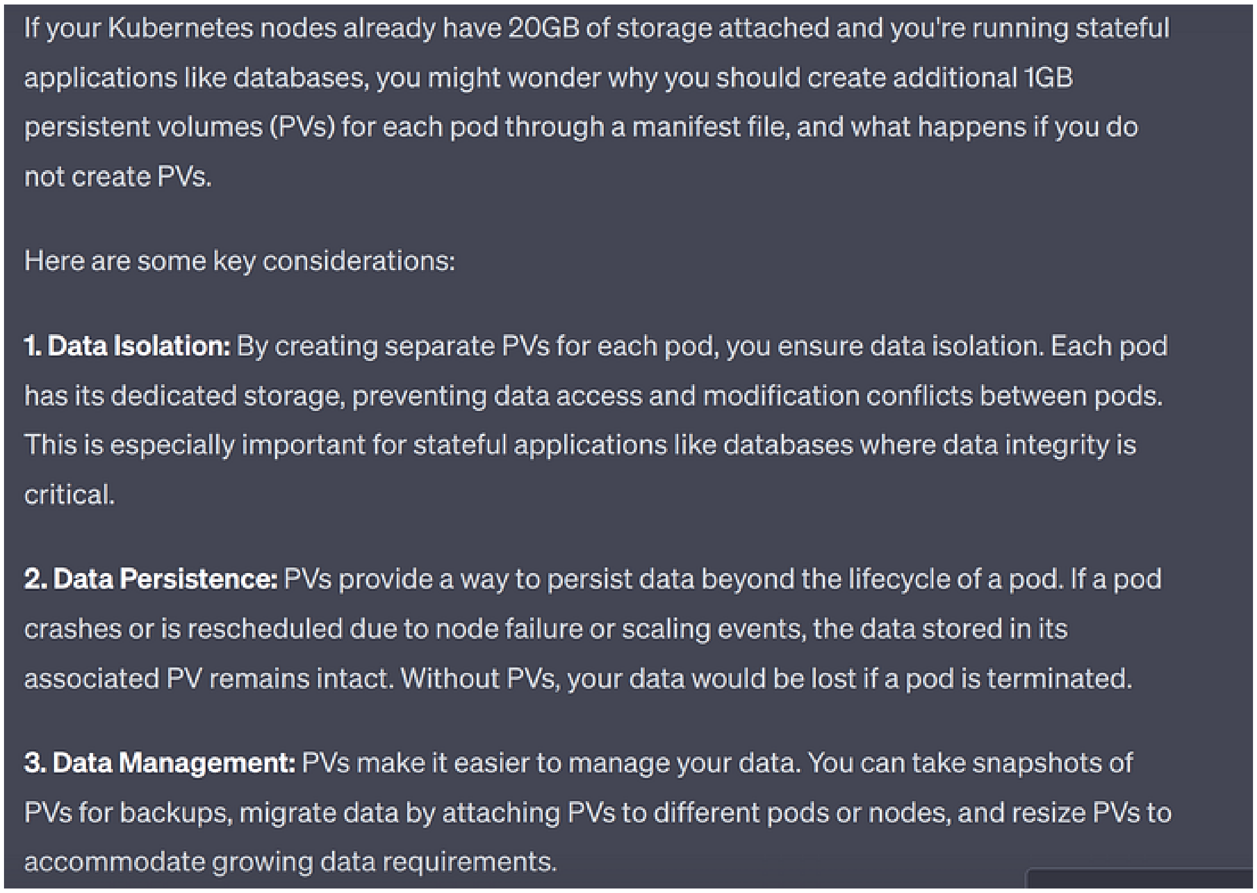


```
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$ kubectl apply -f mongo-statefuls
et.yaml
statefulset.apps/mongo created
```



```
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$ kubectl get pod -n devops
NAME      READY   STATUS    RESTARTS   AGE
mongo-0   1/1     Running   0          72s
mongo-1   1/1     Running   0          46s
mongo-2   1/1     Running   0          20s
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$
```

Benefit of using PVs (persistent volume for database pods)



If your Kubernetes nodes already have 20GB of storage attached and you're running stateful applications like databases, you might wonder why you should create additional 1GB persistent volumes (PVs) for each pod through a manifest file, and what happens if you do not create PVs.

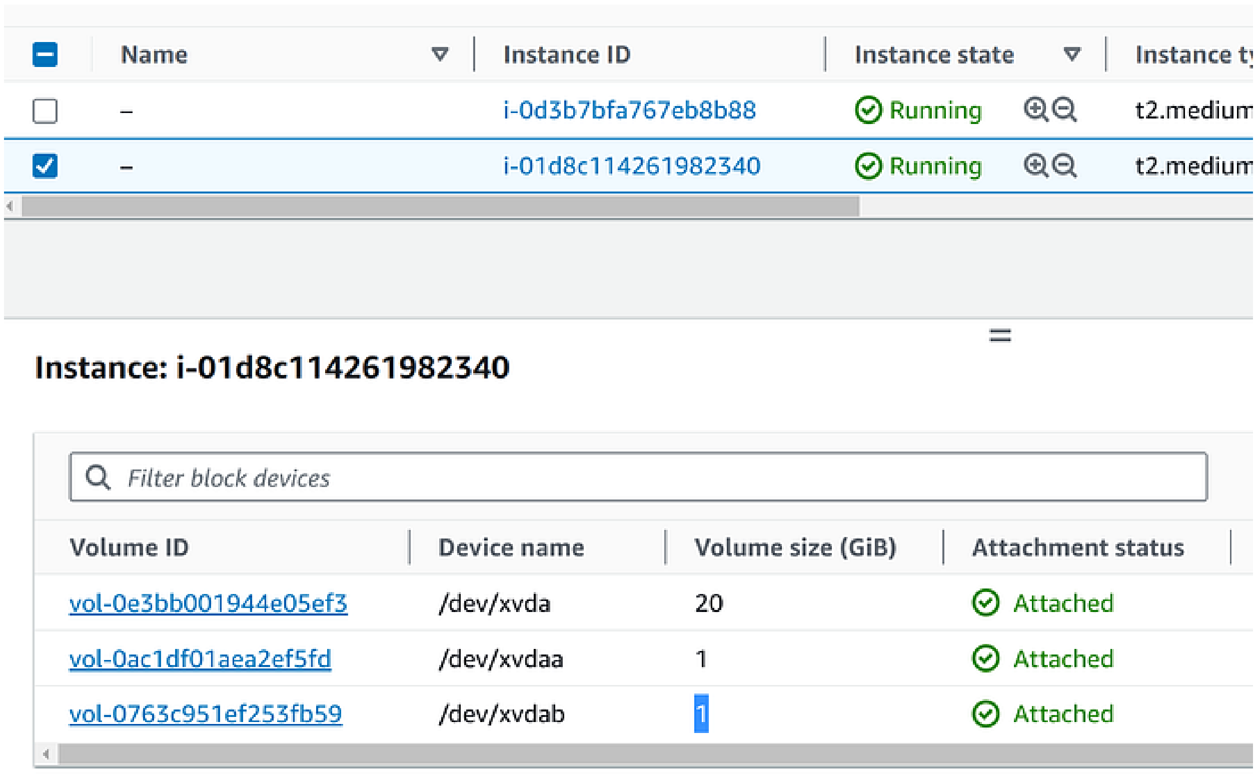
Here are some key considerations:

**1. Data Isolation:** By creating separate PVs for each pod, you ensure data isolation. Each pod has its dedicated storage, preventing data access and modification conflicts between pods. This is especially important for stateful applications like databases where data integrity is critical.

**2. Data Persistence:** PVs provide a way to persist data beyond the lifecycle of a pod. If a pod crashes or is rescheduled due to node failure or scaling events, the data stored in its associated PV remains intact. Without PVs, your data would be lost if a pod is terminated.

**3. Data Management:** PVs make it easier to manage your data. You can take snapshots of PVs for backups, migrate data by attaching PVs to different pods or nodes, and resize PVs to accommodate growing data requirements.

Once pods are created you will notice that extra 1 gb persistent volume also created for each pod, kubernetes scheduler created two pods in one 1 node and 1 pod in other:

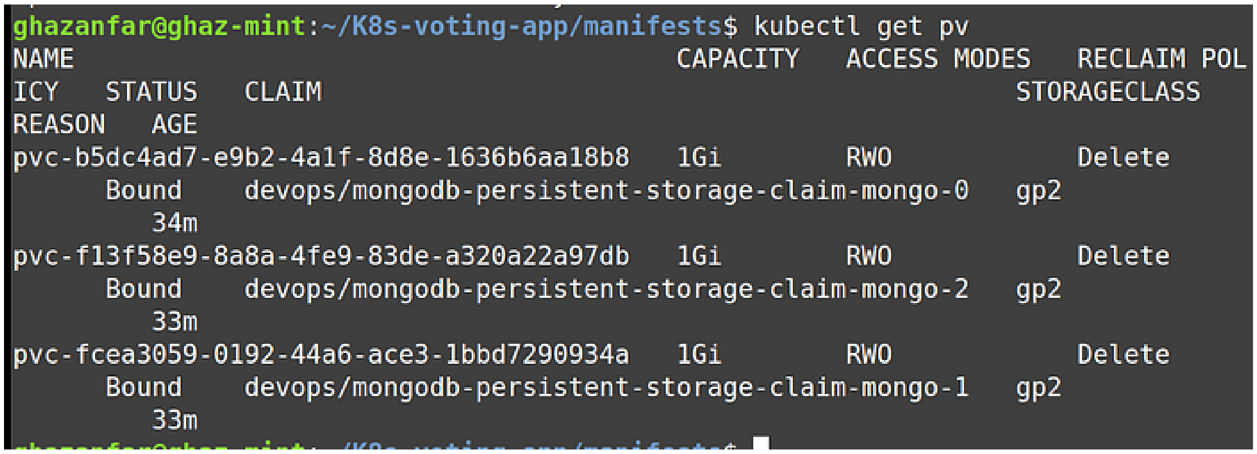


|  | Name | Instance ID | Instance state |  | Instance ty |
|---|---|---|---|---|---|
| ☐ | – | i-0d3b7bfa767eb8b88 | ✓ Running |  | t2.medium |
| ✓ | – | i-01d8c114261982340 | ✓ Running |  | t2.medium |

Instance: i-01d8c114261982340

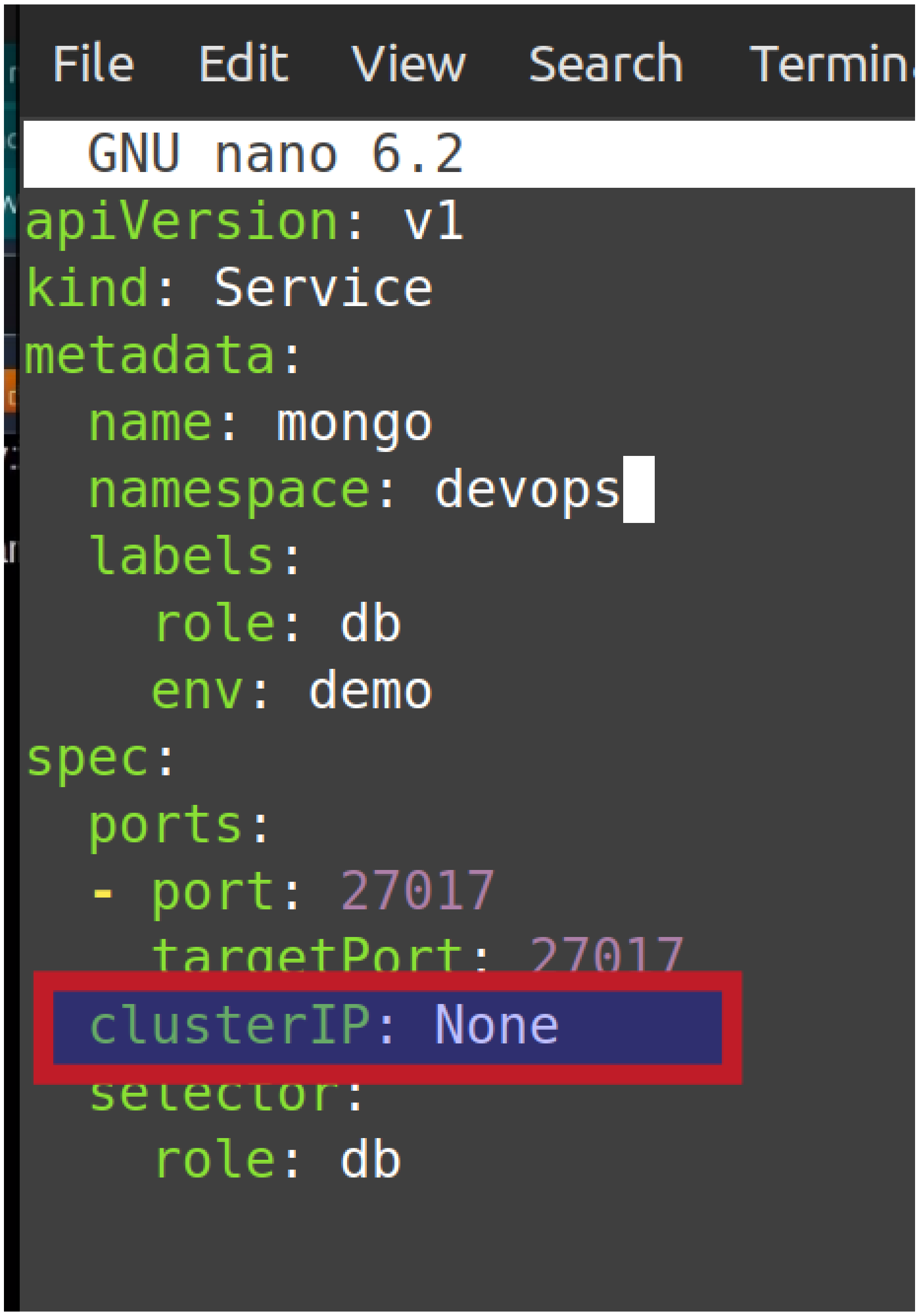
Filter block devices

| Volume ID | Device name | Volume size (GiB) | Attachment status |
|---|---|---|---|
| vol-0e3bb001944e05ef3 | /dev/xvda | 20 | ✓ Attached |
| vol-0ac1df01aea2ef5fd | /dev/xvdaa | 1 | ✓ Attached |
| vol-0763c951ef253fb59 | /dev/xvdab | 1 | ✓ Attached |



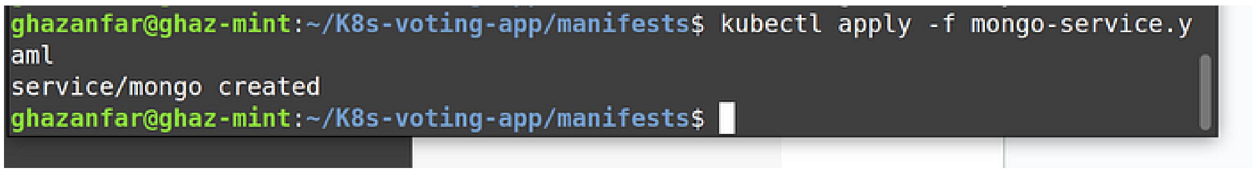
```
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$ kubectl get pv
NAME                                       CAPACITY   ACCESS MODES   RECLAIM POL
ICY   STATUS   CLAIM                                        STORAGECLASS
REASON   AGE
pvc-b5dc4ad7-e9b2-4a1f-8d8e-1636b6aa18b8   1Gi        RWO            Delete
      Bound    devops/mongodb-persistent-storage-claim-mongo-0   gp2
   34m
pvc-f13f58e9-8a8a-4fe9-83de-a320a22a97db   1Gi        RWO            Delete
      Bound    devops/mongodb-persistent-storage-claim-mongo-2   gp2
   33m
pvc-fcea3059-0192-44a6-ace3-1bbd7290934a   1Gi        RWO            Delete
      Bound    devops/mongodb-persistent-storage-claim-mongo-1   gp2
   33m
```

Now we will expose our mongo db pods using service, we will create headless service because we are not gonna expose mongo database to external world, just to expose it for rest API:

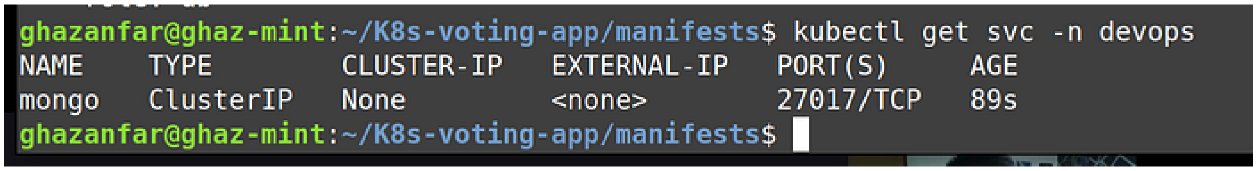


```
apiVersion: v1
kind: Service
metadata:
  name: mongo
  namespace: devops
  labels:
    role: db
    env: demo
spec:
  ports:
  - port: 27017
    targetPort: 27017
  clusterIP: None
  selector:
    role: db
```

Create service for mongodb:

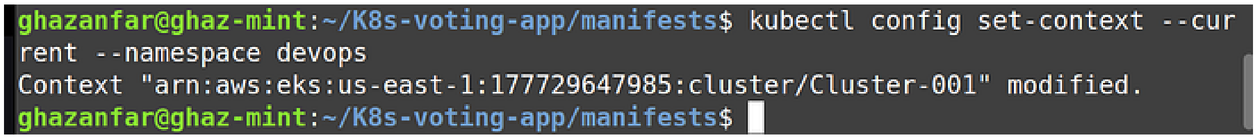


```
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$ kubectl apply -f mongo-service.y
aml
service/mongo created
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$
```



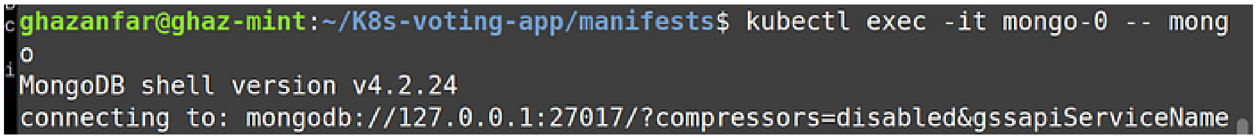
```
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$ kubectl get svc -n devops
NAME    TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)     AGE
mongo   ClusterIP   None         <none>        27017/TCP   89s
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$
```

If you do not want to type namespace again and again with kubectl command use below command:



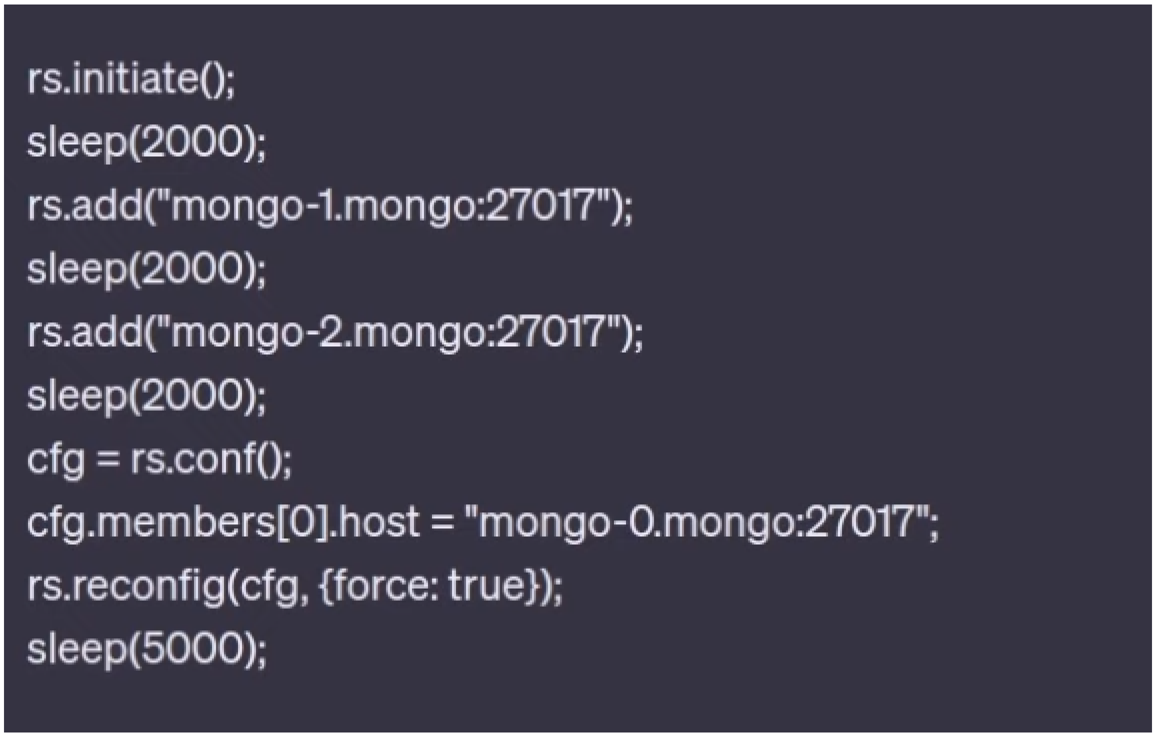
```
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$ kubectl config set-context --cur
rent --namespace devops
Context "arn:aws:eks:us-east-1:177729647985:cluster/Cluster-001" modified.
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$
```

Now we we will set mongo 0 to our primary pod and other two as secondary:

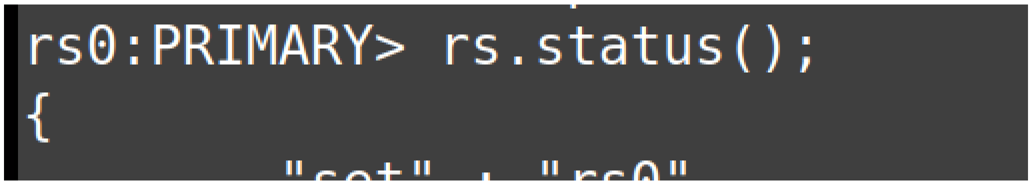


```
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$ kubectl exec -it mongo-0 -- mong
o
MongoDB shell version v4.2.24
connecting to: mongodb://127.0.0.1:27017/?compressors=disabled&gssapiServiceName
```

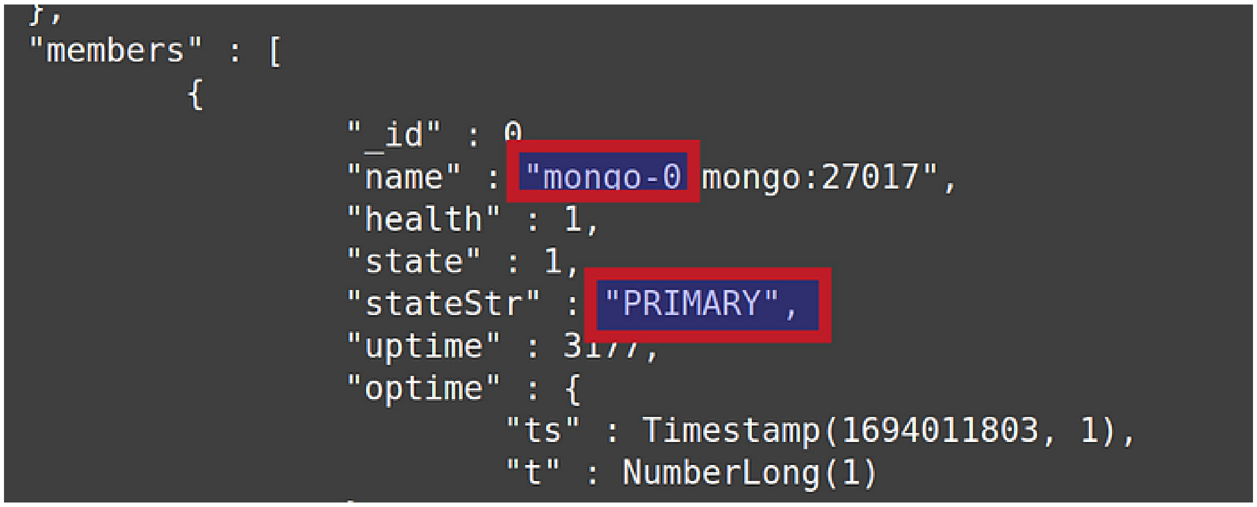
Use these all command in the mongo shell to make the mongo-0 pod as primary and mongo-1, mongo-2 as secondary:



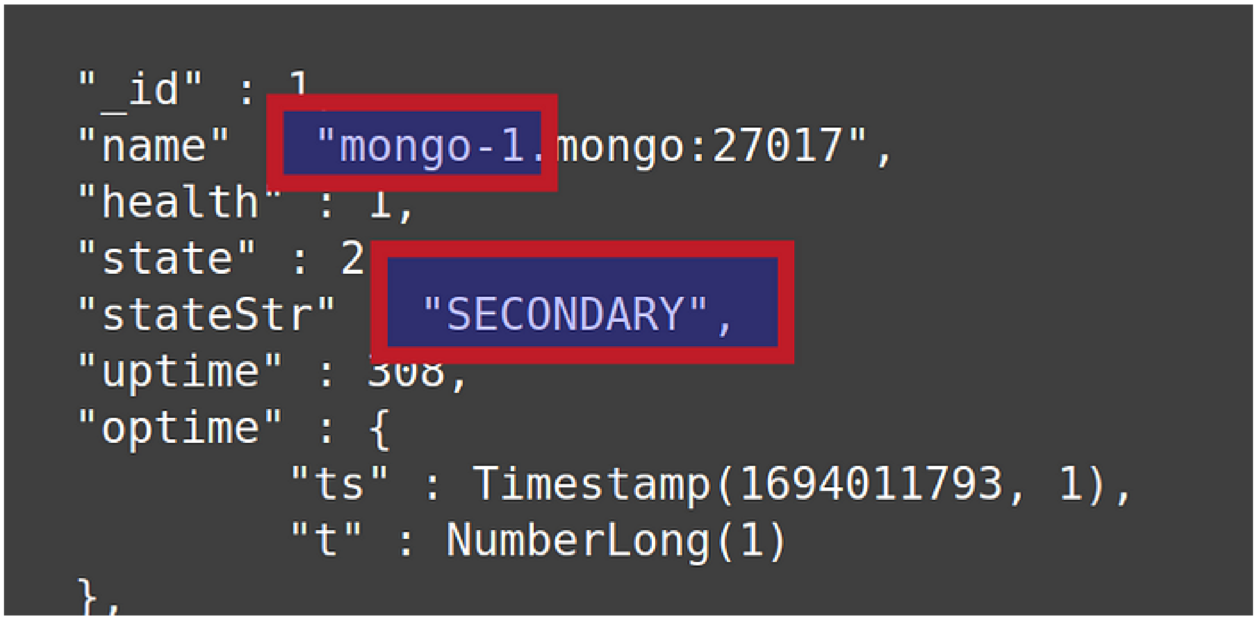
```
rs.initiate();
sleep(2000);
rs.add("mongo-1.mongo:27017");
sleep(2000);
rs.add("mongo-2.mongo:27017");
sleep(2000);
cfg = rs.conf();
cfg.members[0].host = "mongo-0.mongo:27017";
rs.reconfig(cfg, {force: true});
sleep(5000);
```



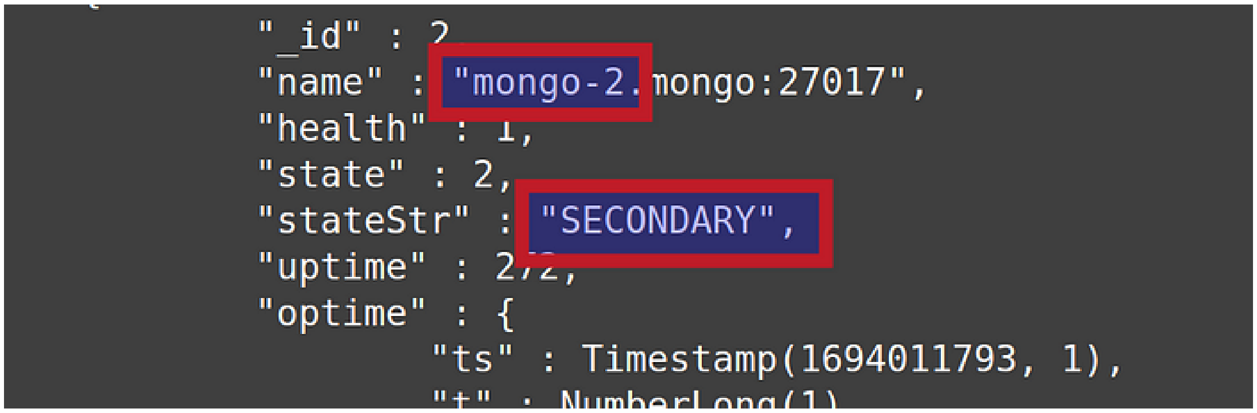
```
rs0:PRIMARY> rs.status();
{
```



```
"members" : [
        {
                "_id" : 0,
                "name" : "mongo-0.mongo:27017",
                "health" : 1,
                "state" : 1,
                "stateStr" : "PRIMARY",
                "uptime" : 3177,
                "optime" : {
                        "ts" : Timestamp(1694011803, 1),
                        "t" : NumberLong(1)
```

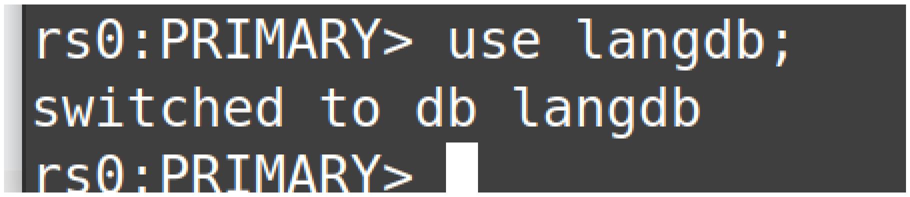


```
"_id" : 1,
"name" : "mongo-1.mongo:27017",
"health" : 1,
"state" : 2,
"stateStr" : "SECONDARY",
"uptime" : 308,
"optime" : {
        "ts" : Timestamp(1694011793, 1),
        "t" : NumberLong(1)
},
```



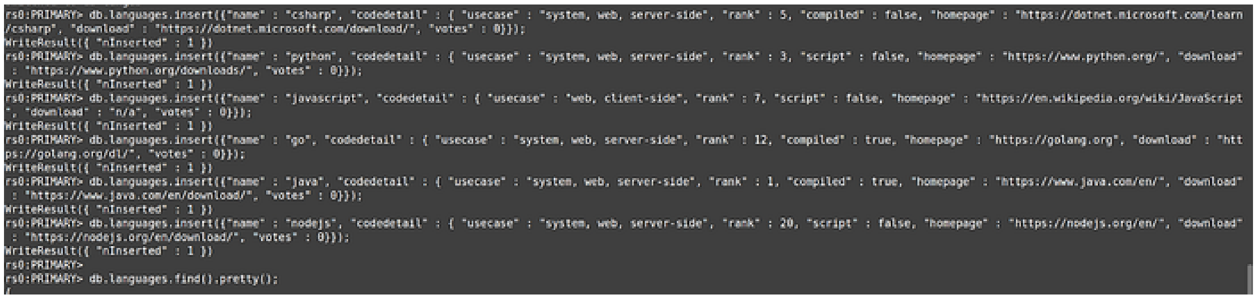
```
        "_id" : 2
        "name" : "mongo-2.mongo:27017",
        "health" : 1,
        "state" : 2,
        "stateStr" : "SECONDARY",
        "uptime" : 272,
        "optime" : {
                "ts" : Timestamp(1694011793, 1),
                "t" : NumberLong(1)
```

Now we will create database in mongo-db so that we can insert data there:



```
rs0:PRIMARY> use langdb;
switched to db langdb
rs0:PRIMARY>
```

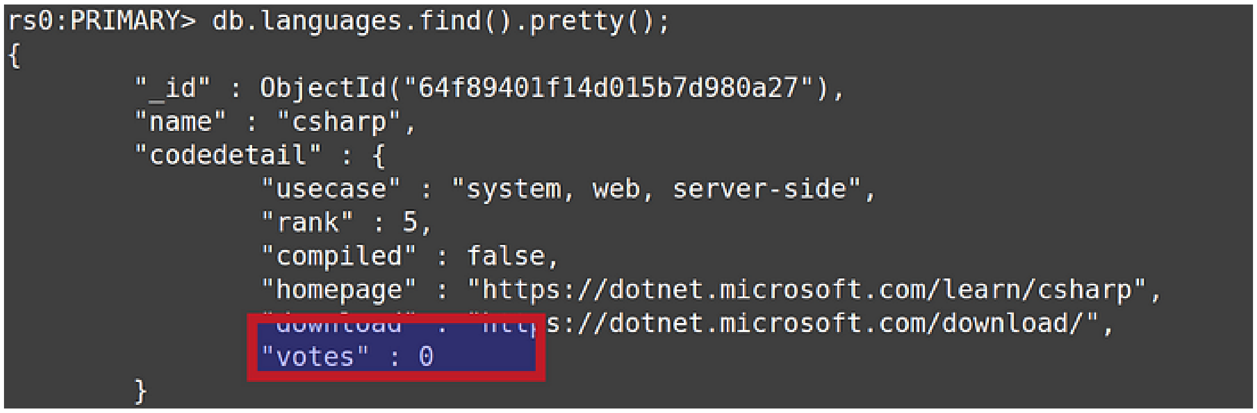
I will put data of each programing language which will be in voting app:



```
rs0:PRIMARY> db.languages.insert({"name" : "csharp", "codedetail" : { "usecase" : "system, web, server-side", "rank" : 5, "compiled" : false, "homepage" : "https://dotnet.microsoft.com/learn
/csharp", "download" : "https://dotnet.microsoft.com/download/", "votes" : 0}});
WriteResult({ "nInserted" : 1 })
rs0:PRIMARY> db.languages.insert({"name" : "python", "codedetail" : { "usecase" : "system, web, server-side", "rank" : 3, "script" : false, "homepage" : "https://www.python.org/", "download"
 : "https://www.python.org/downloads/", "votes" : 0}});
WriteResult({ "nInserted" : 1 })
rs0:PRIMARY> db.languages.insert({"name" : "javascript", "codedetail" : { "usecase" : "web, client-side", "rank" : 7, "script" : false, "homepage" : "https://en.wikipedia.org/wiki/JavaScript
", "download" : "n/a", "votes" : 0}});
WriteResult({ "nInserted" : 1 })
rs0:PRIMARY> db.languages.insert({"name" : "go", "codedetail" : { "usecase" : "system, web, server-side", "rank" : 12, "compiled" : true, "homepage" : "https://golang.org", "download" : "htt
ps://golang.org/dl/", "votes" : 0}});
WriteResult({ "nInserted" : 1 })
rs0:PRIMARY> db.languages.insert({"name" : "java", "codedetail" : { "usecase" : "system, web, server-side", "rank" : 1, "compiled" : true, "homepage" : "https://www.java.com/en/", "download"
 : "https://www.java.com/en/download/", "votes" : 0}});
WriteResult({ "nInserted" : 1 })
rs0:PRIMARY> db.languages.insert({"name" : "nodejs", "codedetail" : { "usecase" : "system, web, server-side", "rank" : 20, "script" : false, "homepage" : "https://nodejs.org/en/", "download"
 : "https://nodejs.org/en/download/", "votes" : 0}});
WriteResult({ "nInserted" : 1 })
rs0:PRIMARY>
rs0:PRIMARY> db.languages.find().pretty();
```

These all commands can be find in readme section of github link:

Write now we have zero vote in each language:

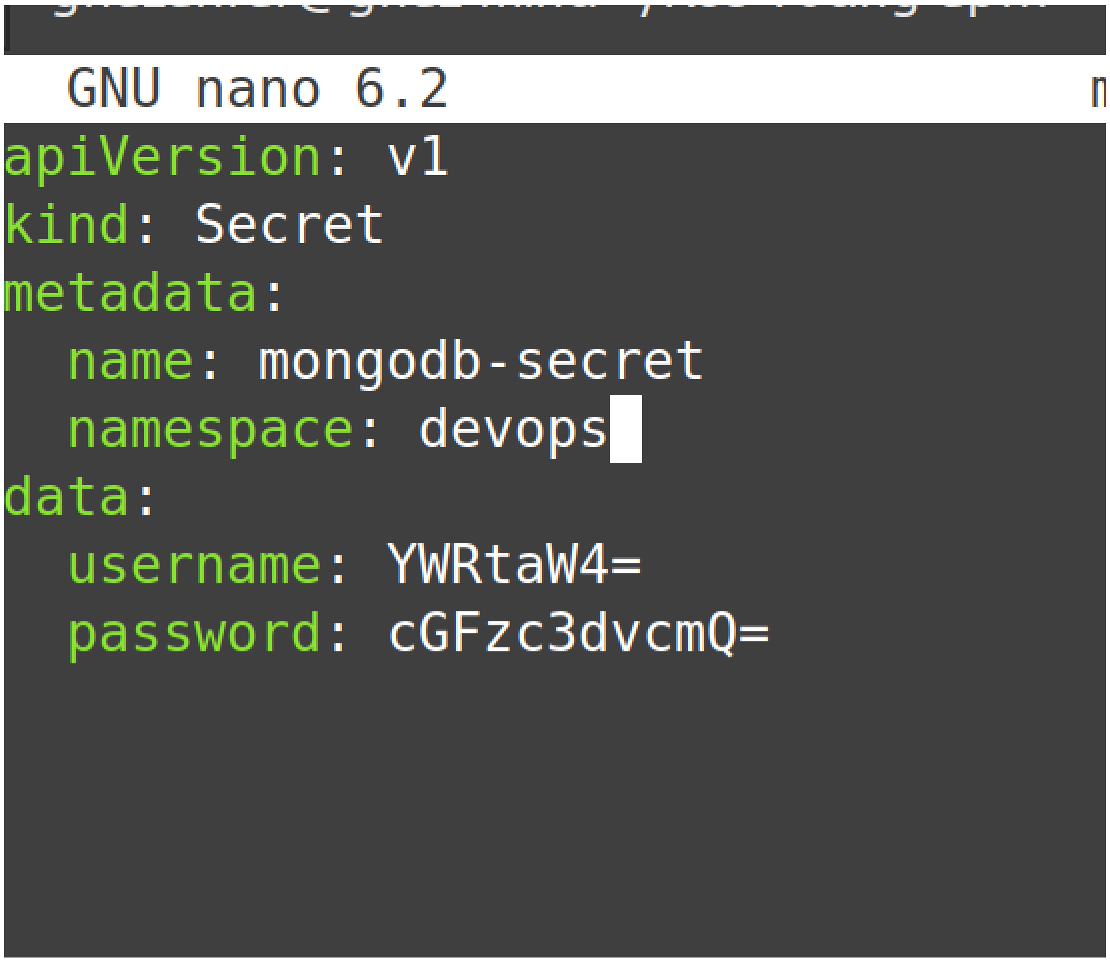


```
rs0:PRIMARY> db.languages.find().pretty();
{
        "_id" : ObjectId("64f89401f14d015b7d980a27"),
        "name" : "csharp",
        "codedetail" : {
                "usecase" : "system, web, server-side",
                "rank" : 5,
                "compiled" : false,
                "homepage" : "https://dotnet.microsoft.com/learn/csharp",
                "download" : "https://dotnet.microsoft.com/download/",
                "votes" : 0
        }
```

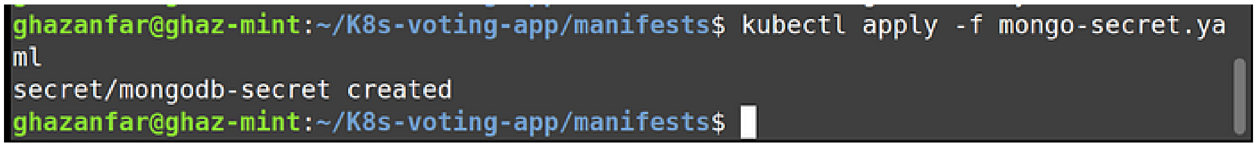
So we will create front end so that when ever someone cast vote in font end it is deploy in mongodb through API.

First deploy the API:

To deploy the API we will create the secret file that hold the password and username for the database:

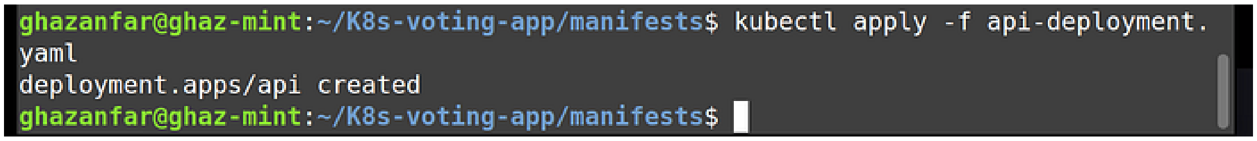


```
  GNU nano 6.2
apiVersion: v1
kind: Secret
metadata:
  name: mongodb-secret
  namespace: devops
data:
  username: YWRtaW4=
  password: cGFzc3dvcmQ=
```



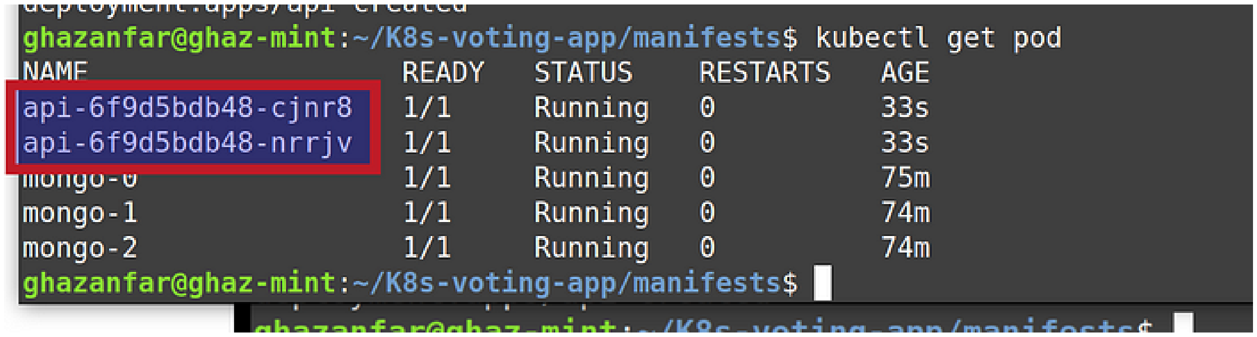
```
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$ kubectl apply -f mongo-secret.ya
ml
secret/mongodb-secret created
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$
```

Now we are ready to deploy the API as well:



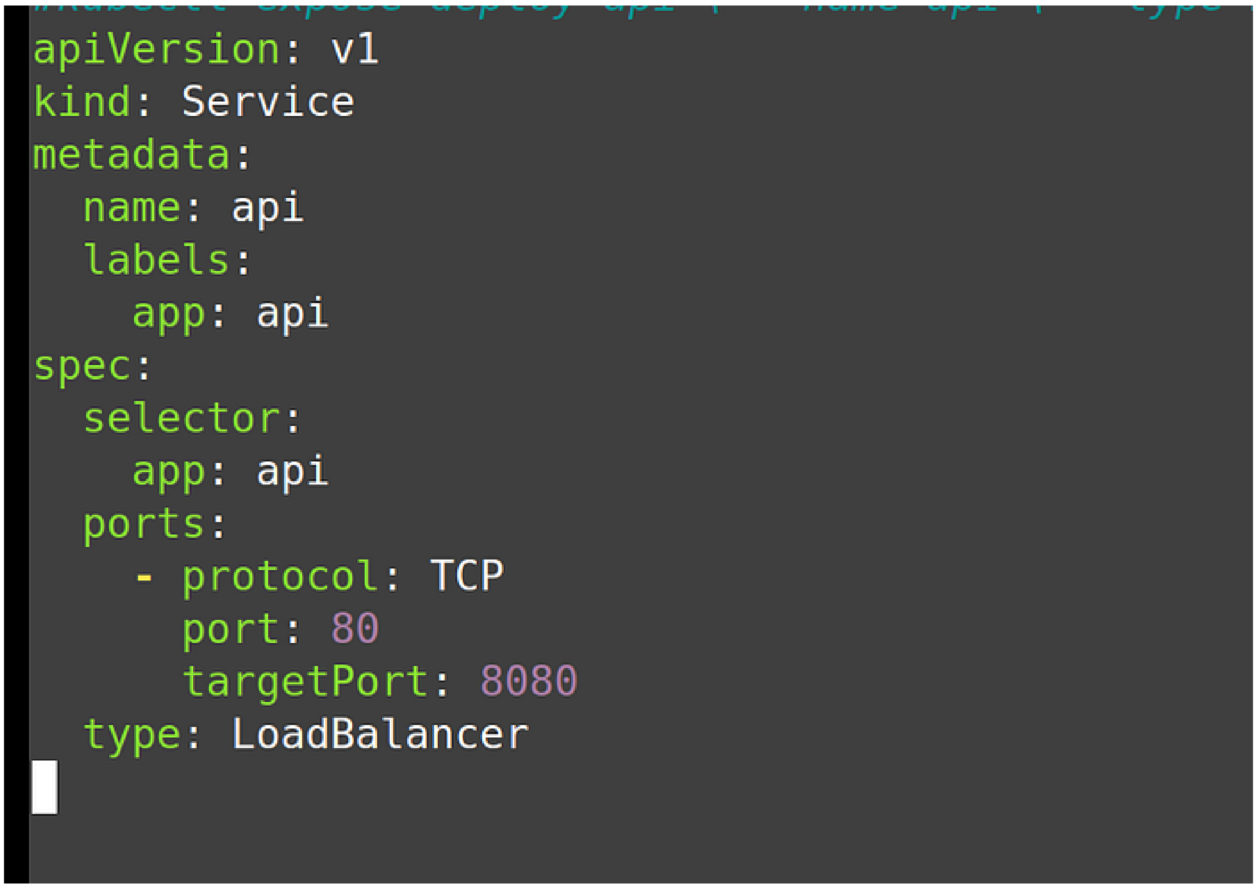
```
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$ kubectl apply -f api-deployment.
yaml
deployment.apps/api created
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$
```

It will create two pods for API:

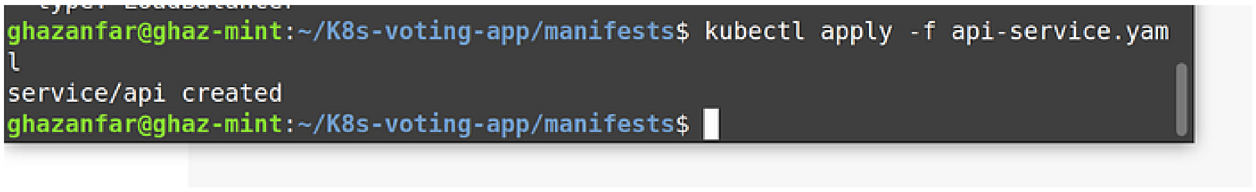


```
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$ kubectl get pod
NAME                   READY   STATUS    RESTARTS   AGE
api-6f9d5bdb48-cjnr8   1/1     Running   0          33s
api-6f9d5bdb48-nrrjv   1/1     Running   0          33s
mongo-0                1/1     Running   0          75m
mongo-1                1/1     Running   0          74m
mongo-2                1/1     Running   0          74m
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$
```

We need to expose these API by creating the load balancer service because need to expose API to outside cluster:

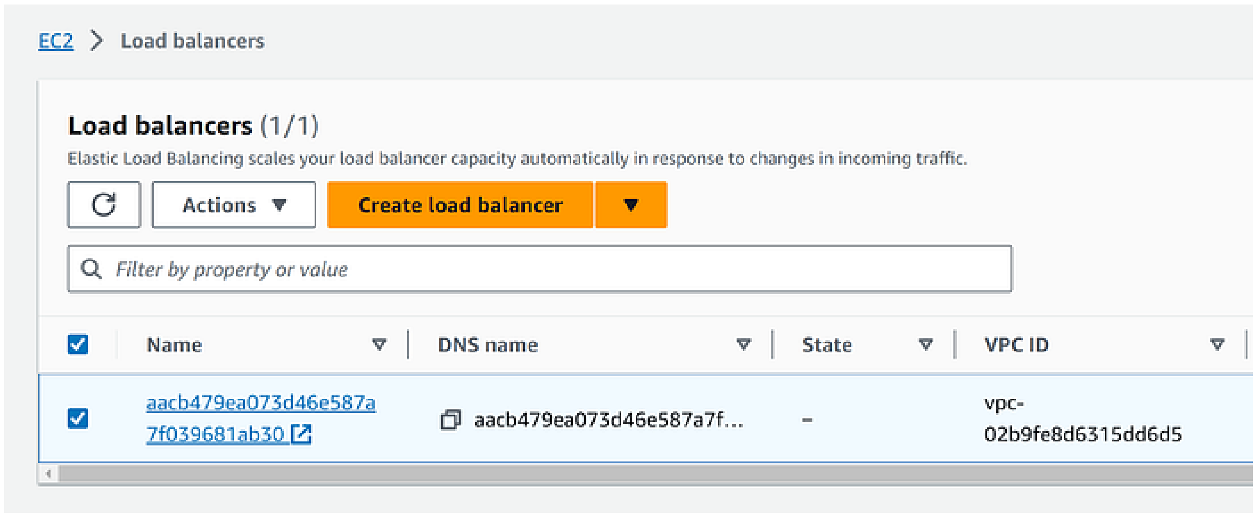


```
apiVersion: v1
kind: Service
metadata:
  name: api
  labels:
    app: api
spec:
  selector:
    app: api
  ports:
    - protocol: TCP
      port: 80
      targetPort: 8080
  type: LoadBalancer
```



```
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$ kubectl apply -f api-service.yam
l
service/api created
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$
```

It have created load balancer for us:

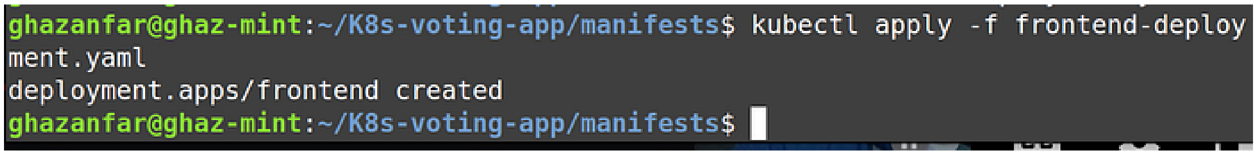

EC2 > Load balancers

Load balancers (1/1)

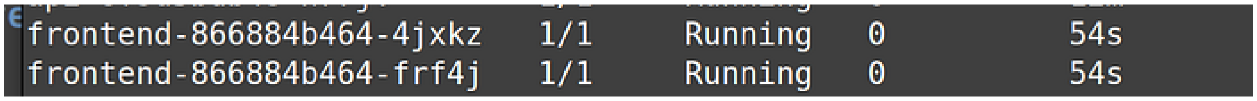
Elastic Load Balancing scales your load balancer capacity automatically in response to changes in incoming traffic.

| Name | DNS name | State | VPC ID |
|---|---|---|---|
| aacb479ea073d46e587a7f039681ab30 | aacb479ea073d46e587a7f... | – | vpc-02b9fe8d6315dd6d5 |

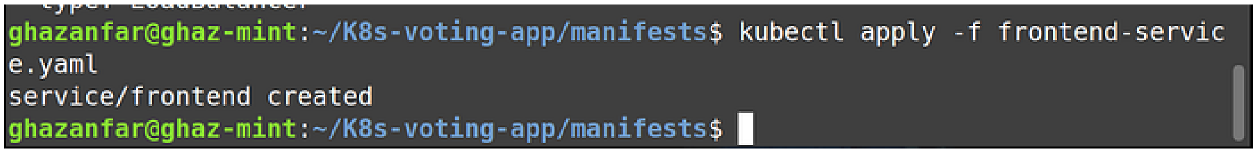
Now we will deploy our front end which will get votes from the users:


```
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$ kubectl apply -f frontend-deploy
ment.yaml
deployment.apps/frontend created
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$
```

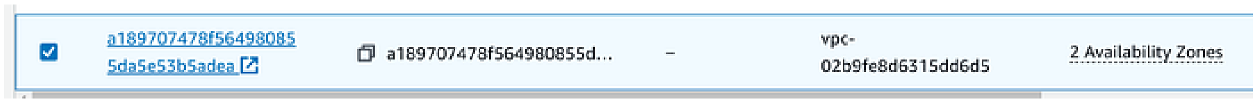
It has created two pods for front-end also:


```
frontend-866884b464-4jxkz   1/1     Running   0          54s
frontend-866884b464-frf4j   1/1     Running   0          54s
```

Now expose the front end using load balancer service:

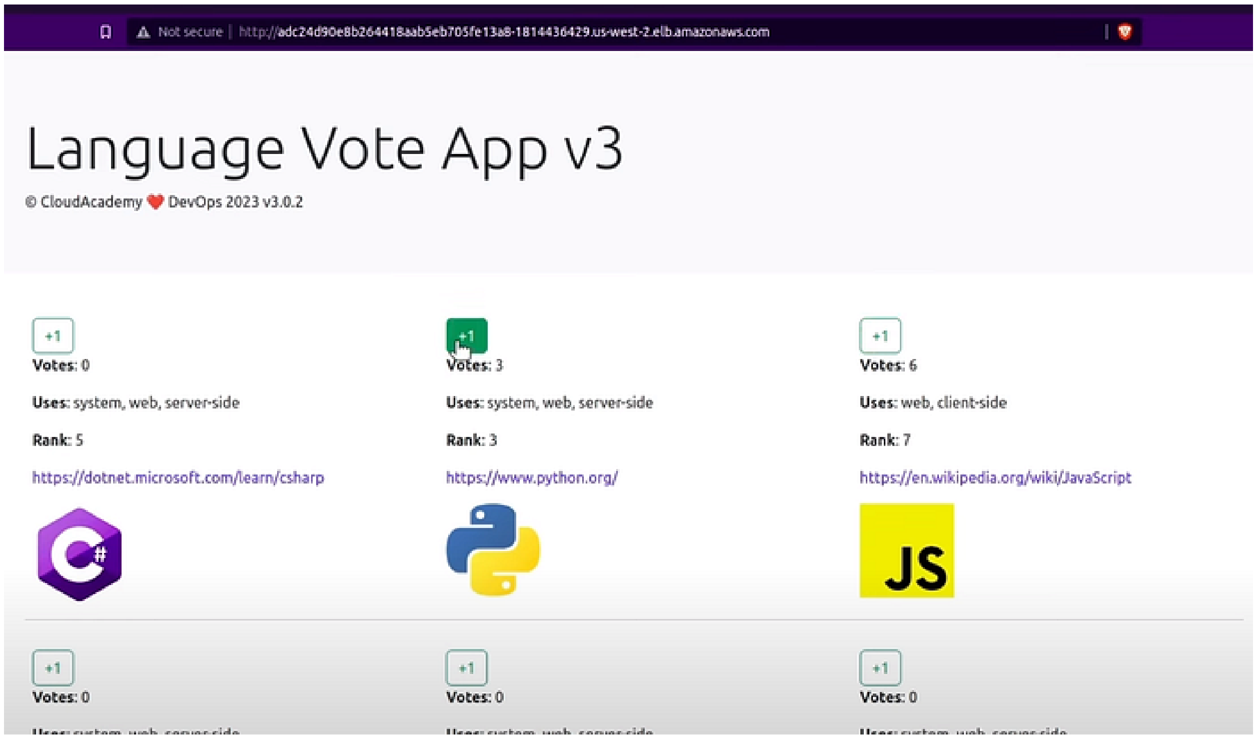

```
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$ kubectl apply -f frontend-servic
e.yaml
service/frontend created
ghazanfar@ghaz-mint:~/K8s-voting-app/manifests$
```

It also created load balancer for front end:



| Name | DNS name | State | VPC ID | Availability Zones |
|---|---|---|---|---|
| a189707478f564980855da5e53b5adea | a189707478f564980855d... | – | vpc-02b9fe8d6315dd6d5 | 2 Availability Zones |

Now access the front-end load balancer and cast some votes:

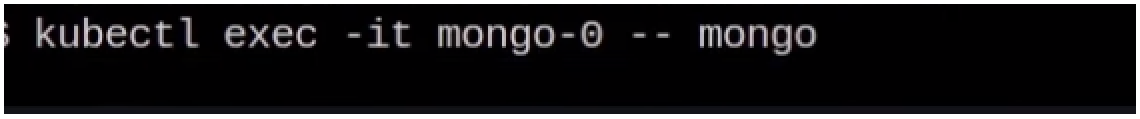

Not secure | http://adc24d90e8b264418aab5eb705fe13a8-1814436429.us-west-2.elb.amazonaws.com

Language Vote App v3

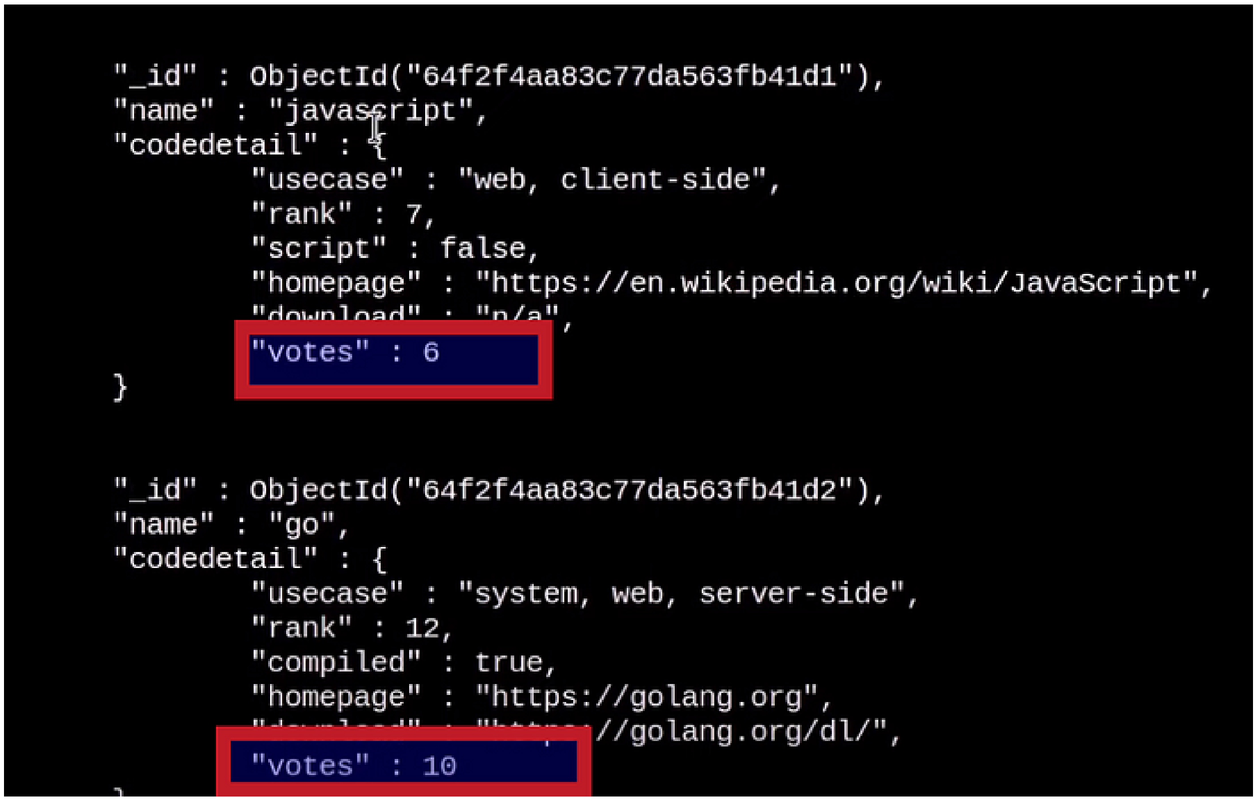
© CloudAcademy ❤️ DevOps 2023 v3.0.2

| | | |
|---|---|---|
| +1 | +1 | +1 |
| **Votes:** 0 | **Votes:** 3 | **Votes:** 6 |
| **Uses:** system, web, server-side | **Uses:** system, web, server-side | **Uses:** web, client-side |
| **Rank:** 5 | **Rank:** 3 | **Rank:** 7 |
| https://dotnet.microsoft.com/learn/csharp | https://www.python.org/ | https://en.wikipedia.org/wiki/JavaScript |
| +1 | +1 | +1 |
| **Votes:** 0 | **Votes:** 0 | **Votes:** 0 |

These vote will be seen in database:



```
$ kubectl exec -it mongo-0 -- mongo
```



```
"_id" : ObjectId("64f2f4aa83c77da563fb41d1"),
"name" : "javascript",
"codedetail" : {
        "usecase" : "web, client-side",
        "rank" : 7,
        "script" : false,
        "homepage" : "https://en.wikipedia.org/wiki/JavaScript",
        "download" : "n/a",
        "votes" : 6
}


"_id" : ObjectId("64f2f4aa83c77da563fb41d2"),
"name" : "go",
"codedetail" : {
        "usecase" : "system, web, server-side",
        "rank" : 12,
        "compiled" : true,
        "homepage" : "https://golang.org",
        "download" : "https://golang.org/dl/",
        "votes" : 10
```

Thats all in this project.