Project: Cloud-Native Web Voting Application with Kubernetes

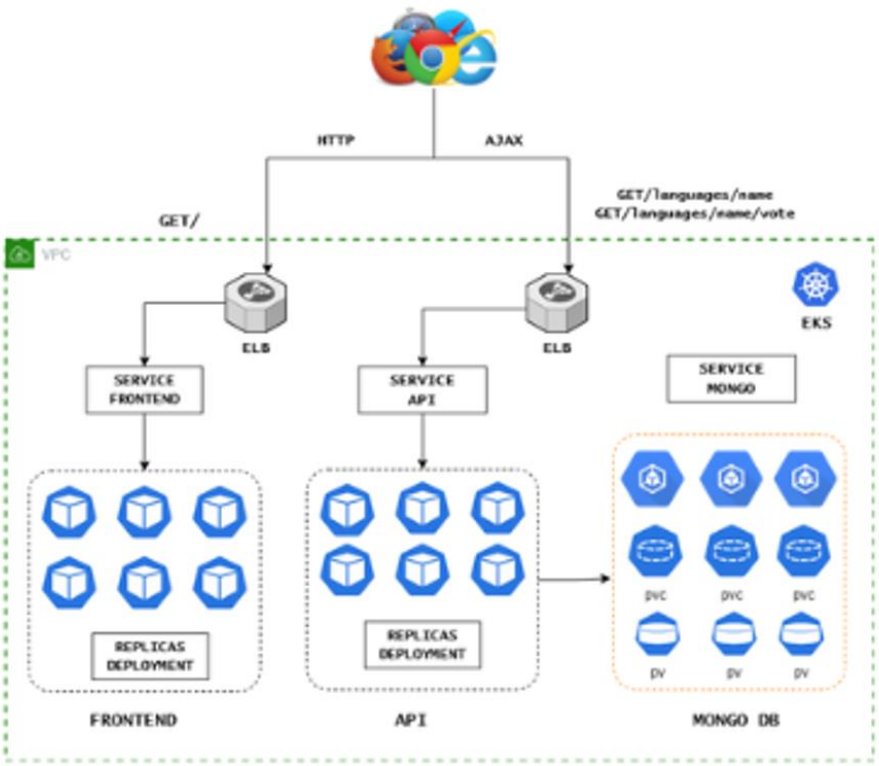
Introduction:

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.

Project Architecture:



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.

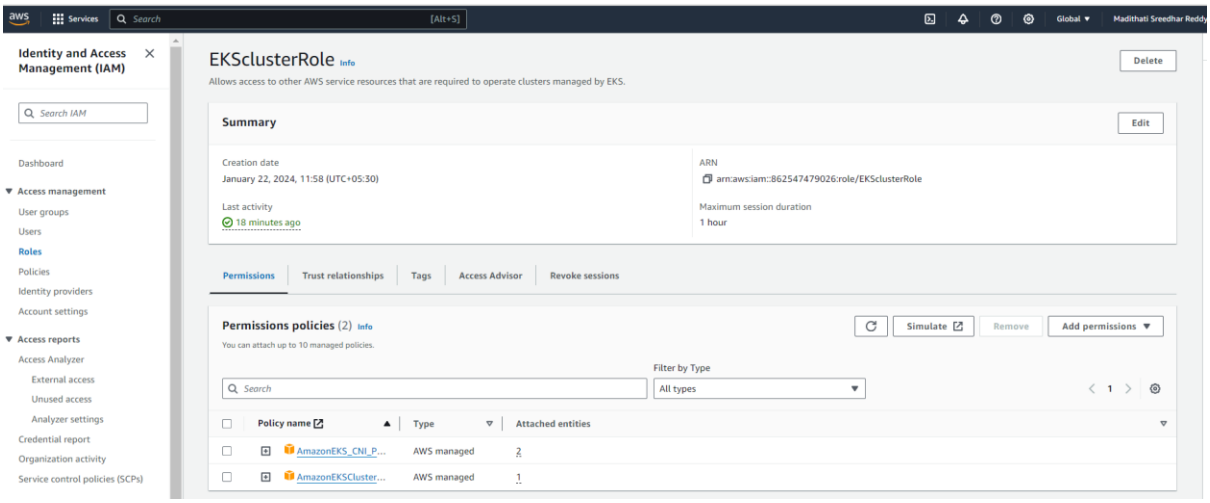
By working through this project, you'll develop a deeper understanding of cloud-native application development, containerization, Kubernetes, and the various technologies involved in building and deploying modern web applications.

Steps to Deploy:

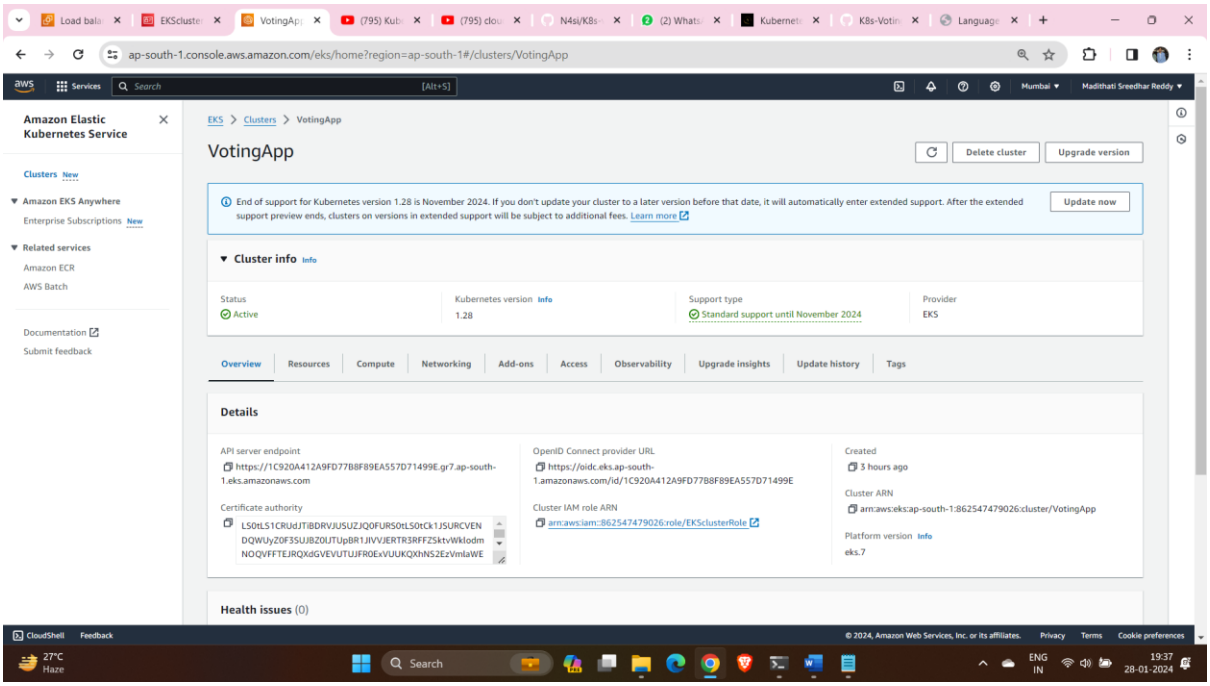
Creating EKS Cluster:

1. Navigate to Your Aws Console
2. Click the “Search” field and search For EKS or select directly Elastic Kubernetes Service on the Recently visited tab
3. Click “Add cluster” and then “Create”
4. Click the “Name” field and enter a unique name for the cluster that is anything you want. For example, I used Cloud and version 1.27
5. Click on Amazon EKS User Guide for New IAM role creation.
   Redirect to the IAM dashboard

6. Click “Roles” → “Create role”→ “Allow AWS services like EC2, Lambda, or others to perform actions in this account.”→ “Choose a service or use case” and Type “EKS”
   Click this radio button with EKS-Cluster
   Click “Next” and you will directly redirect to policy and click Next we have only one policy for it and it selects by default for EKS) that is AmazonEKSClusterPolicy
   Click the “Role name” field and provide the name (EKSClusterRole) and then “Create role”
   Click “EKSClusterRole” that is created at Cluster Service Role and then “Next”

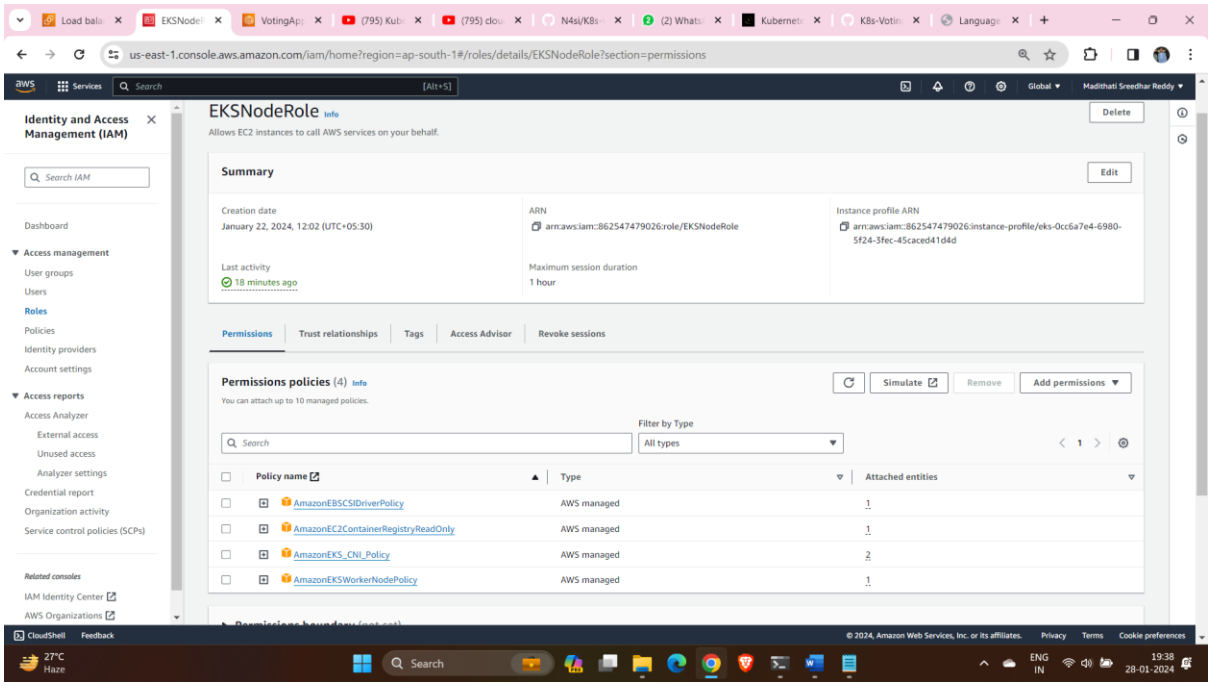


7. Click “Select security groups” and use the existing security group or create a new security Group and click “Next”
8. No changes Click “Next” (Default no need to change anything) then “Create”
9. In your cluster Click “Add-ons”
   Click “Get more add-ons”
   Click this checkbox. with Amazon EBS CSI Driver
   No changes Click “Next” (Default no need to change anything)
10. Click “Create”
11. Once your Cluster up to active status

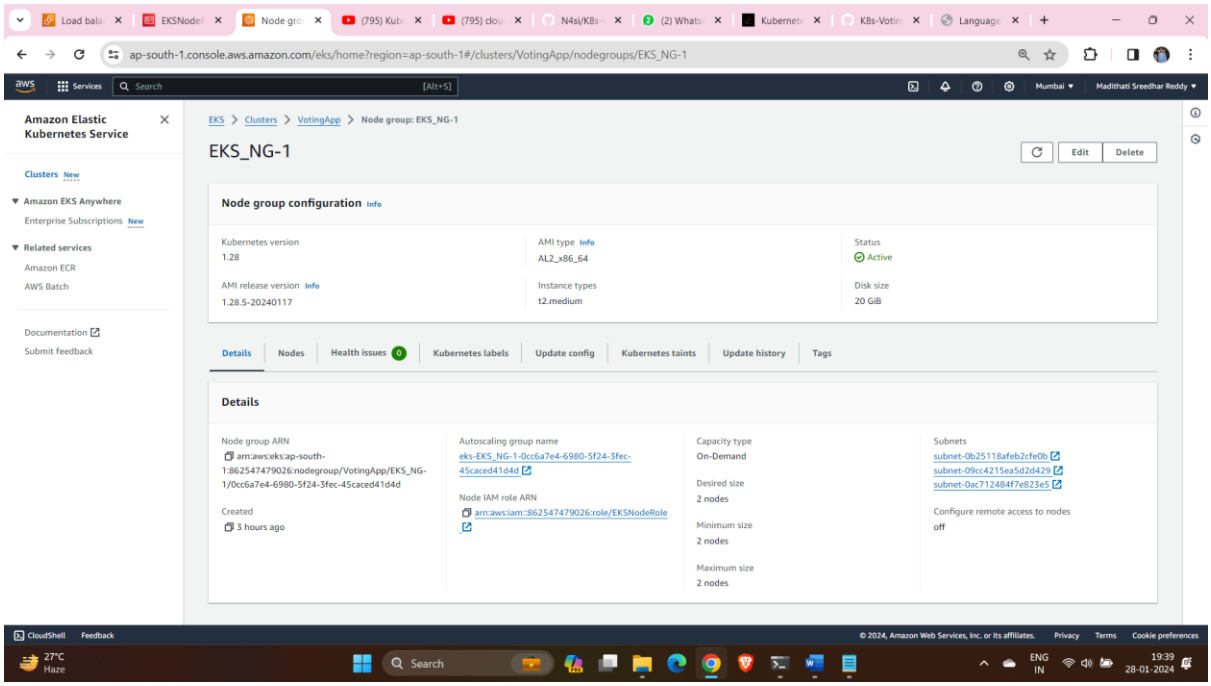


Creating EKS-Node Group:

1. Click “Compute”
2. Click on “Add node group” and then “Name” field.
   Write any Name you want (NodeGroup)
3. Click “Select role” and click on the IAM console
   Click “Create role”→“Allow AWS services like EC2, Lambda, or others to perform actions in this account.”→ “Choose a service or use case” → “EC2” and “Next”→ “Search” field.
4. Search these Policy Names and make it check
   AmazonEC2ContainerRegistryReadOnly
   AmazonEKS_CNI_Policy
   AmazonEBSCSIDriverPolicy
   AmazonEKSWorkerNodePolicy
   Click “Next” and then “Role name” field.
   Add Role name as EKSNodeRole
   Click “Create role”
   Add a role that was created before “EKSNodeRole”



5. Click “Next”
6. On the next page remove t3.medium and add t2.medium as instance type.
   Select t2.medium
7. Click “Next” →“Create”

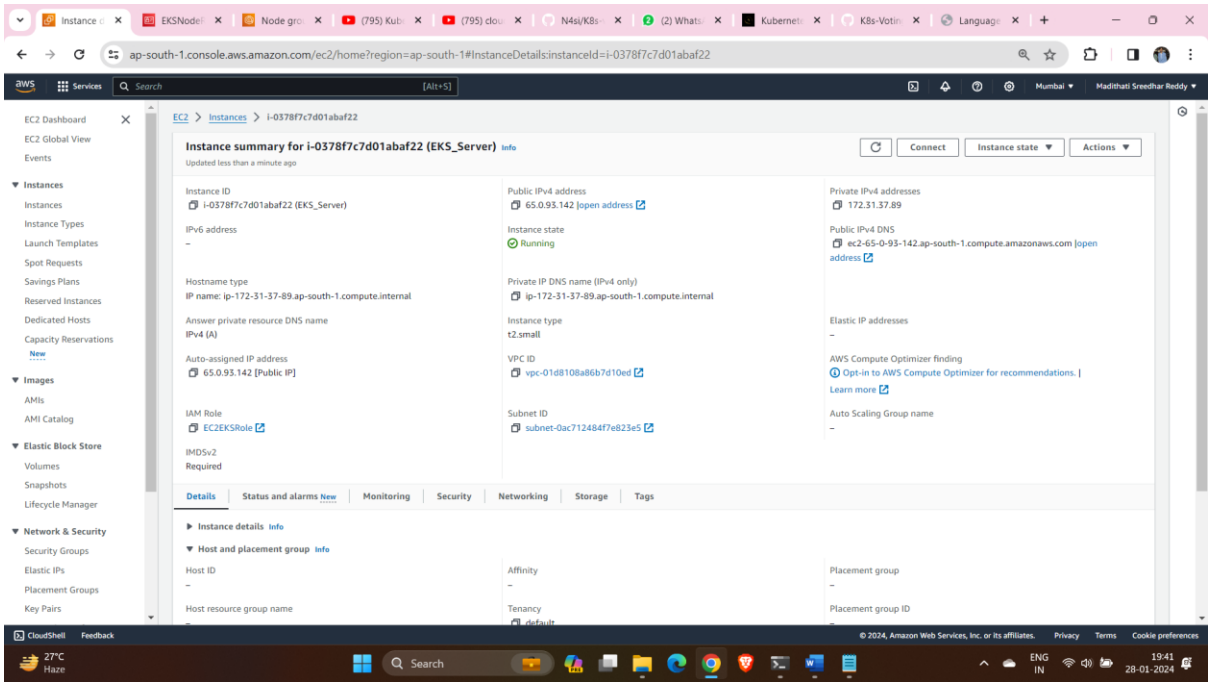


Create EC2-Server:

1. Node Groups will take some time to create, Click “EC2” or Search for Ec2
2. Click “Launch instance”
   Add Name and AMI as Ubuntu
   Take instance type as t2.micro and select keypair with default security Group.
3. Click “Advanced details”
4. Click on the IAM instance Profile and Create a New IAM profile
5. Click “Create role” → “Choose a service or use case”→ “EC2”→ “Search” field.
   Type “EBS”
   Click this checkbox with the policy name AmazonEBSCSIDriverPolicy. → “Next”
6. Click the “Role name” field and provide the name as EKSaccess → “Create role”→ newly created role “EKSaccess” →“Add permissions”→ “Create inline policy”→ “JSON"
7. REMOVE EVERYTHING FROM THE POLICY EDITOR
   And add this

```
{
        "Version": "2012-10-17",
        "Statement": [{
                "Effect": "Allow",
                "Action": [
                        "eks:DescribeCluster",
                        "eks:ListClusters",
                        "eks:DescribeNodegroup",
                        "eks:ListNodegroups",
                        "eks:ListUpdates",
                        "eks:AccessKubernetesApi"
                ],
                "Resource": "*"
        }]
}
```

Click “Next”→“Policy name” field and add the name as eksaccesspolicy→ create policy.
Add That Role to your instance and launch the instance.



Installing Required Tools:

- Once the instance comes up copy the SSH client to connect to Putty.
- Install Kubectl on the instance

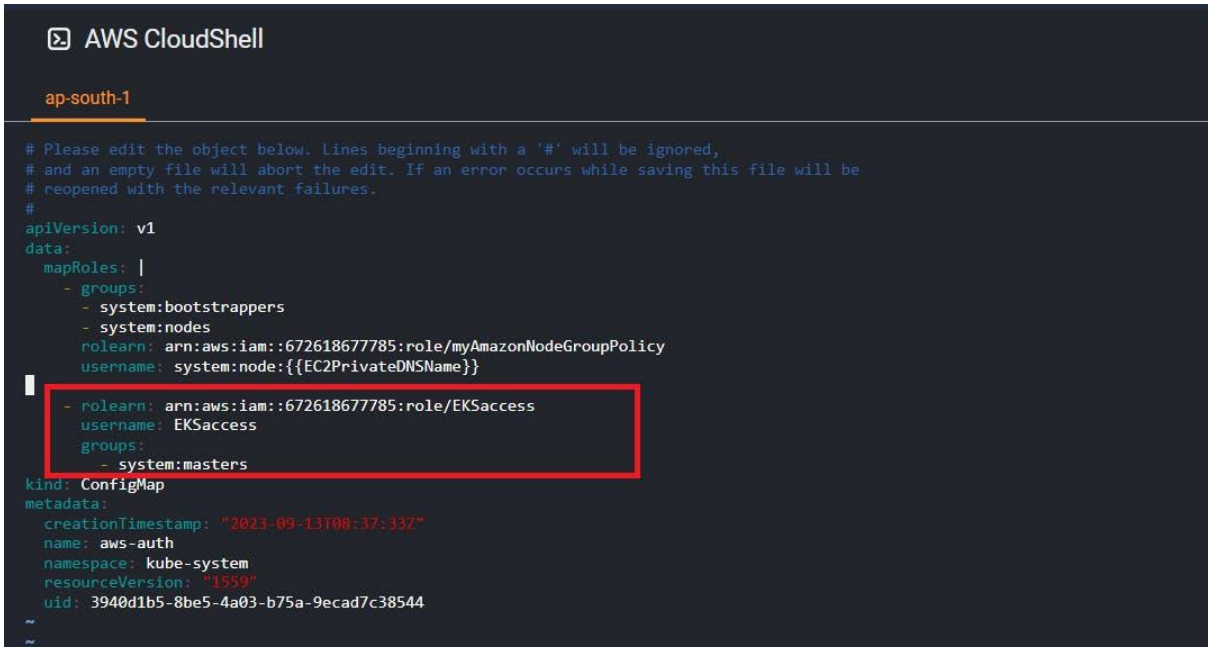
```
#installs kubectl on instance
curl -O https://s3.us-west-2.amazonaws.com/amazon-eks/1.24.11/2023-03-17/bin/linux/am
chmod +x ./kubectl
sudo cp ./kubectl /usr/local/bin
export PATH=/usr/local/bin:$PATH
```

- Install AWS CLI on the instance

```
#install awscli
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2.zip
sudo ./aws/install
```

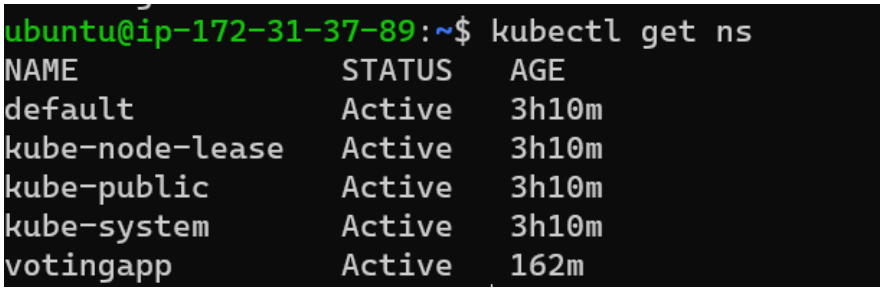
- Now check whether nodes are up or not
  kubectl get nodes
  You will get a refused error because we haven’t set up the context yet. lets set context
  aws eks update-kubeconfig --name EKS_CLUSTER_NAME --region CLUSTER_REGION
- Let’s check again whether nodes are up or not from instance.
  We will get an error that You must be logged in to the server (unauthorized)
  The error message “You must be logged in to the server (Unauthorized)” in Kubernetes indicates that the user or service account trying to access the cluster does not have the necessary permissions. This error typically occurs when the authentication and authorization mechanisms in Kubernetes deny access.
- Let’s Resolve the issue;

1. Go to Aws console
2. Click on the AWS cloud shell icon on the top right
3. Click on “Connect”
4. First set context by providing the following command
   aws eks update-kubeconfig --name EKS_CLUSTER_NAME --region CLUSTER_REGION
5. Edit the config map for access
   kubectl edit configmap aws-auth --namespace kube-system
6. Go to your Iam roles and copy the arn of iam role of ec2 instance that is attached
   Add your Role arn to the config map



   EKSaccess role is added to the Instance (While creating the instance)
7. Check now whether nodes are up or not

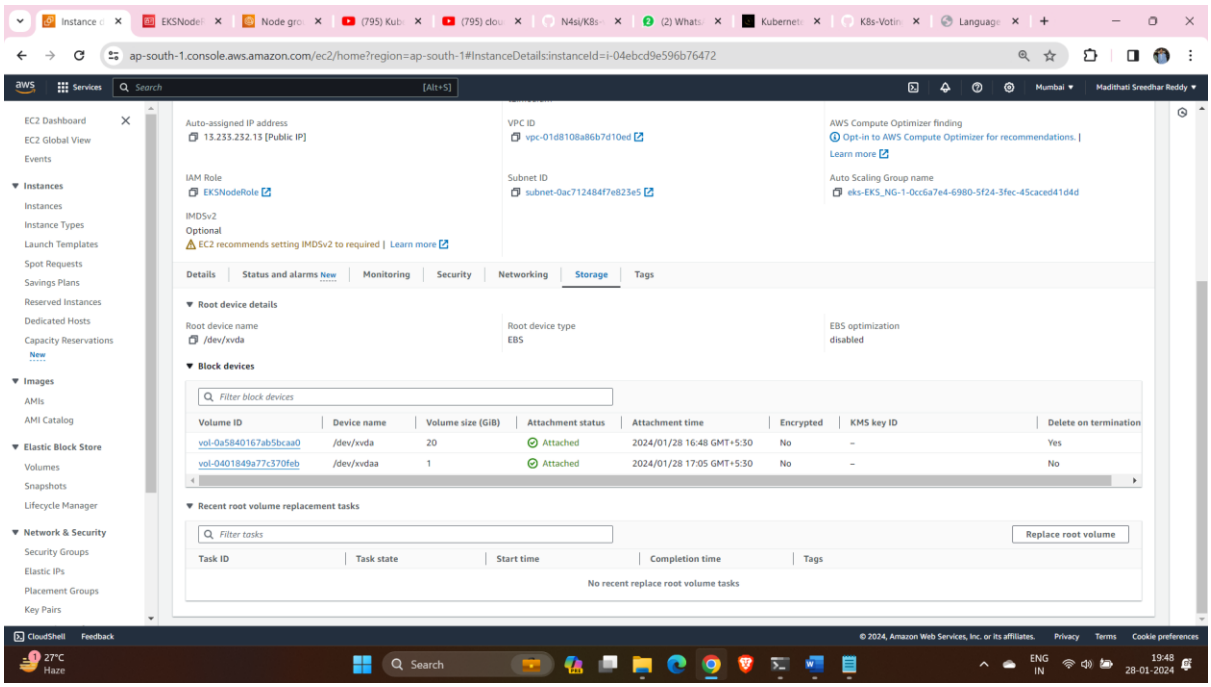
- Let’s clone our Project Repository
  git clone <repo-url>
- Go inside the K8s-voting app once it is cloned.
  In the API deployment, we used a namespace as cloudchamp. By default we get 4 namespaces only, we have to create votingapp namespace.
  kubectl create ns votingapp



- When you want to work within a specific namespace for your Kubernetes operations.
  We have to set our namespace as current
  kubectl config set-context --current --namespace cloudchamp

MONGO Database Setup:

- To create a Mongo stateful set with Persistent volumes, run the command in the manifests folder:
  #to apply manifest file
  kubectl apply -f mongo-statefulset.yaml
  #to check pods
  kubectl get pods
- Go to Aws console and click on nodes and storage You can see now new 1Gb storage has been added to both nodes



Create Mongo Service:
kubectl apply -f mongo-service.yaml
kubectl get svc

- Now let's go inside the mongo-0 pod and we have to initialise the Mongo database Replica set.
  kubectl get pods
  kubectl exec -it mongo-0 -- mongo
- Load the Data in the database by running this command:
  use langdb

```
cat << EOF | kubectl exec -it mongo-0 -- mongo
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);
EOF
```

db.languages.find().pretty();

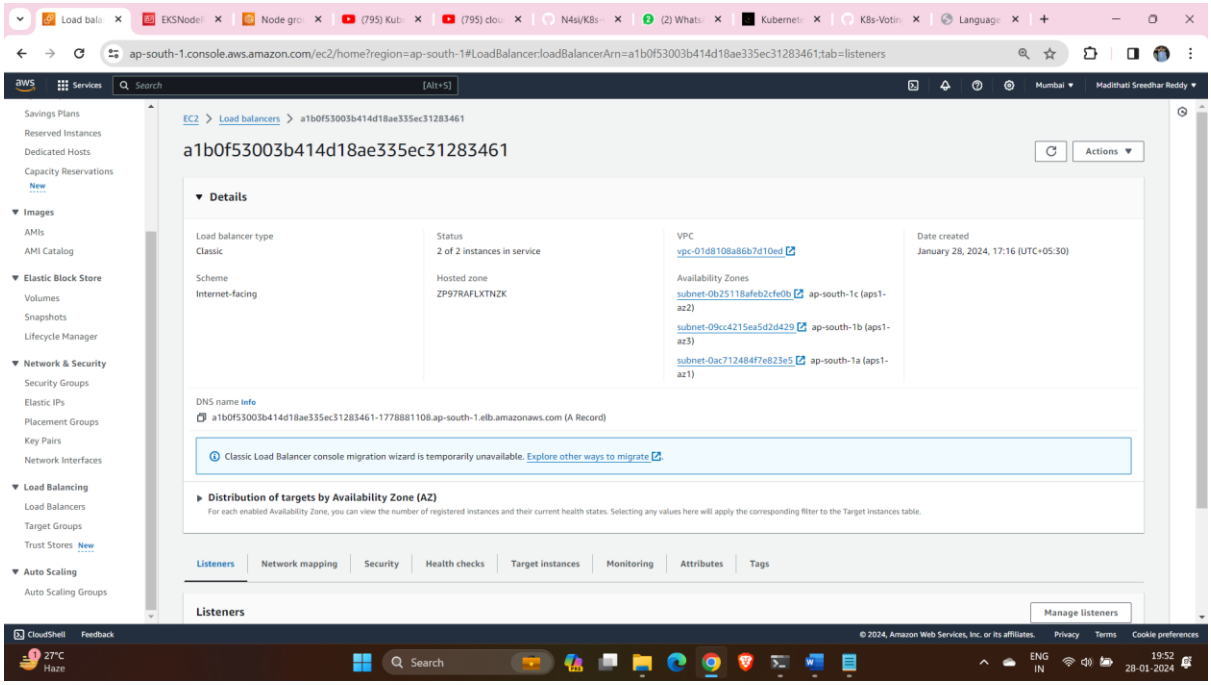
```
cat << EOF | kubectl exec -it mongo-0 -- mongo
use langdb;
db.languages.insert({"name" : "csharp", "codedetail" : { "usecase" : "system, web, server-side", "rank" : 5, "compiled" : false
db.languages.insert({"name" : "python", "codedetail" : { "usecase" : "system, web, server-side", "rank" : 3, "script" : false,
db.languages.insert({"name" : "javascript", "codedetail" : { "usecase" : "web, client-side", "rank" : 7, "script" : false, "hom
db.languages.insert({"name" : "go", "codedetail" : { "usecase" : "system, web, server-side", "rank" : 12, "compiled" : true, "h
db.languages.insert({"name" : "java", "codedetail" : { "usecase" : "system, web, server-side", "rank" : 1, "compiled" : true, "
db.languages.insert({"name" : "nodejs", "codedetail" : { "usecase" : "system, web, server-side", "rank" : 20, "script" : false,

db.languages.find().pretty();
EOF
```

exit #exit from conatiner

- Create Mongo secret:
  kubectl apply -f mongo-secret.yaml

API Setup:

1. Create GO API deployment by running the following command:
   kubectl apply -f api-deployment.yaml
   kubectl get all
2. Expose API deployment through service using the following command
   kubectl expose deploy api \
   --name=api \
   --type=LoadBalancer \
   --port=80 \
   --target-port=8080
   kubectl get svc
3. One load Balancer will be created in your AWS account

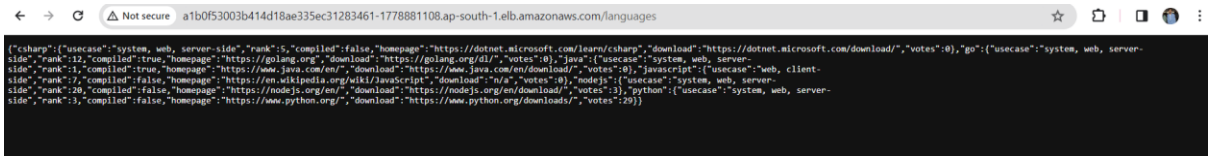


4. Next, set the environment variable:
   {
   API_ELB_PUBLIC_FQDN=$(kubectl get svc api -ojsonpath="{.status.loadBalancer.ingress[0].hostname}")

until nslookup $API_ELB_PUBLIC_FQDN >/dev/null 2>&1; do sleep 2 && echo waiting for DNS to propagate...; done
curl $API_ELB_PUBLIC_FQDN/ok
echo
}

5. Test and confirm that the API route URL /languages, and /languages/{name} endpoints can be called successfully. In the terminal run any of the following commands:

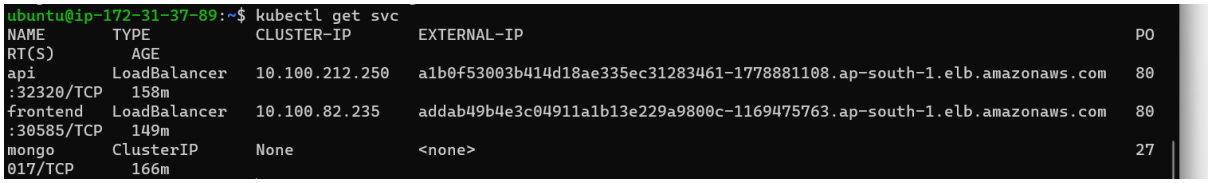
<api loadbalancer ip/languages> #in browser



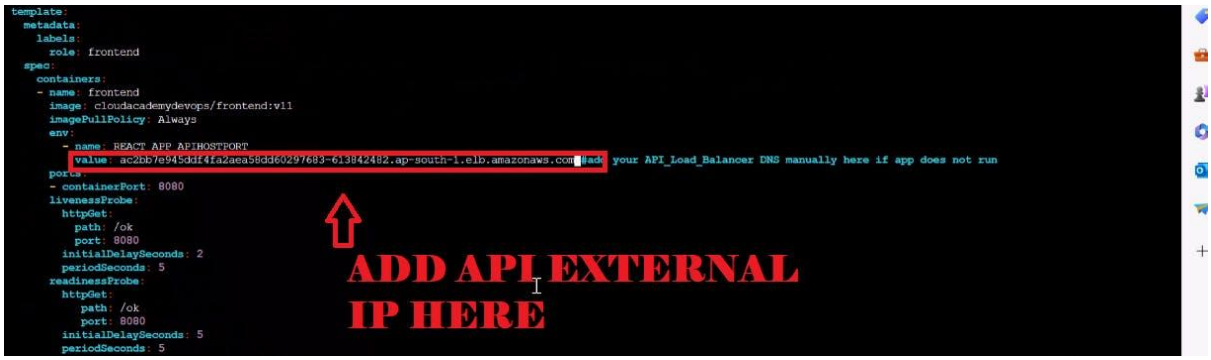
6. In the browser, you have to use your external IP of Api to see this output
7. If everything works fine, go ahead with the Frontend setup.

Frontend setup:

1. Now copy your API External service ip
   kubectl get svc



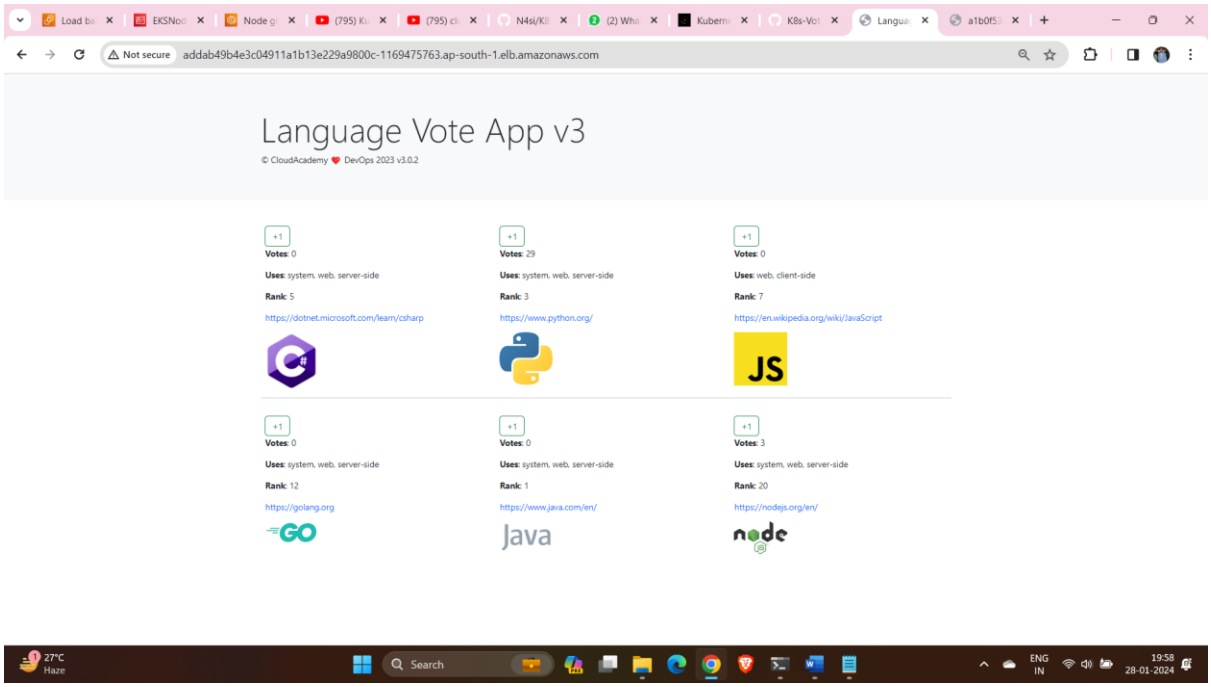
2. Now open your frontend-deployment. yaml file
   sudo vi frontend-deployment.yaml
   Update the frontend-deployment.yaml file with your api-ip



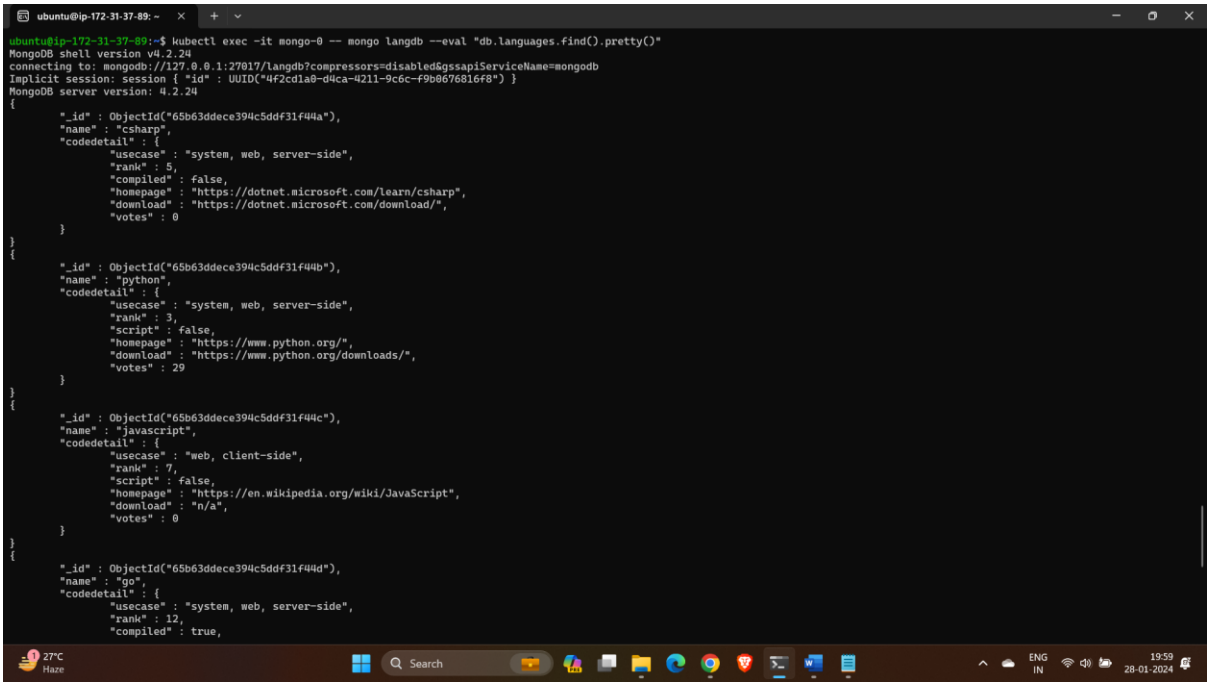
3. Now deploy the frontend
   kubectl apply -f frontend-deployment.yaml
4. And now expose the frontend-service:
   kubectl expose deploy frontend \
   --name=frontend \
   --type=LoadBalancer \
   --port=80 \
   --target-port=8080

```
ubuntu@ip-172-31-37-89:~$ kubectl get all
NAME                           READY   STATUS    RESTARTS   AGE
pod/api-6dd74f6f5d-8vdkh       1/1     Running   0          161m
pod/api-6dd74f6f5d-gkzkj       1/1     Running   0          161m
pod/frontend-74486f6498-9hhbb  1/1     Running   0          151m
pod/frontend-74486f6498-tx6lc  1/1     Running   0          151m
pod/mongo-0                    1/1     Running   0          172m
pod/mongo-1                    1/1     Running   0          171m
pod/mongo-2                    1/1     Running   0          171m

NAME               TYPE           CLUSTER-IP       EXTERNAL-IP
      PORT(S)        AGE
service/api        LoadBalancer   10.100.212.250   a1b0f53003b414d18ae335ec31283461-1778881108.ap-south-1.elb.amazonaws.com   80:32320
/TCP   160m
service/frontend   LoadBalancer   10.100.82.235    addab49b4e3c04911a1b13e229a9800c-1169475763.ap-south-1.elb.amazonaws.com   80:30585
/TCP   151m
service/mongo      ClusterIP      None             <none>
      27017/TCP      168m

NAME                       READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/api        2/2     2            2           161m
deployment.apps/frontend   2/2     2            2           151m

NAME                                  DESIRED   CURRENT   READY   AGE
replicaset.apps/api-6dd74f6f5d        2         2         2       161m
replicaset.apps/frontend-74486f6498   2         2         2       151m

ubuntu@ip-172-31-37-89:~$
statefulset.apps/mongo   3/3     172m
ubuntu@ip-172-31-37-89:~$
```

Copy your external ip of the frontend service and paste it into the browser You will get an application like this:



- Using your local workstation's browser – browse to the URL created in the previous output.
- After the voting application has loaded successfully, vote by clicking on several of the +1 buttons, This will generate AJAX traffic which will be sent back to the API via the API's assigned ELB.
- Query the MongoDB database directly to observe the updated vote data. In the terminal execute the following command:
  kubectl exec -it mongo-0 -- mongo langdb --eval "db.languages.find().pretty()"



- If you got the data both in the console and in the UI then:

Congratulations, you have successfully completed the project

Happy Learning..:)