Cloud-Native Web Voting Application with Kubernetes

A combination of technologies is used in the development of this cloud-native web app. Users can vote for their favorite programming language from a list of six options—C#, Python, JavaScript, Go, Java, and NodeJS—by accessing it online.

Technical Stack

- 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.
- Frontend: The frontend of this application is built using React and JavaScript. It provides a responsive and user-friendly interface for casting votes.

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.
- Deployment: Kubernetes deployments specify the number of instances that the application should run and offer scalability and update instructions.
- Secret: Kubernetes secrets store sensitive information, such as API keys or credentials, required by the application securely.
- Service: Kubernetes services ensure that users can access the application by directing incoming traffic to the appropriate instances.
- Stateful: For components requiring statefulness, such as the MongoDB replica set, Kubernetes StatefulSets are employed to maintain order and unique identities.
- PersistentVolume and PersistentVolumeClaim:By managing the storage needed for the application, these Kubernetes resources guarantee data scalability and persistence.

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: Learn how to package apps and their dependencies by working with containerization tools such as Docker.
2. Kubernetes Orchestration: Discover how to effectively manage, grow, and deploy containerized apps in a production environment.
3. Microservices Architecture: Examine the advantages and difficulties of a microservices architecture, which separates and independently scales the front end and back end.
4. Database Replication: To ensure high availability and data redundancy, learn how to configure and maintain a MongoDB replica set.
5. Security and Secrets Management: Discover the best ways to use Kubernetes secrets to protect sensitive data.
6. Stateful Applications: Learn about the subtleties of putting stateful apps in place in an orchestration environment for containers.
7. Persistent Storage: Recognize how Kubernetes provides and administers persistent storage for stateful applications.

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

Create EKS cluster with NodeGroup (2 nodes of t2.medium instance type) Create EC2 Instance t2.micro (Optional)

##IAM role for ec2

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

```
            ],
            "Resource": "*"
        }]
}
```

Install Kubectl:

```
curl -O
https://s3.us-west-2.amazonaws.com/amazon-eks/1.24.11/2023-03-17/bin/linux/amd6
4/kubectl
chmod +x ./kubectl
sudo cp ./kubectl /usr/local/bin
export PATH=/usr/local/bin:$PATH
```

Install AWScli:

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

Once the Cluster is ready run the command to set context:

```
aws eks update-kubeconfig --name EKS_CLUSTER_NAME --region us-west-2
```

To check the nodes in your cluster run

```
kubectl get nodes
```

If using EC2 and getting the "You must be logged in to the server (Unauthorized)" error, refer this: https://repost.aws/knowledge-center/eks-api-server-unauthorized-error

Clone the github repo

```
git clone https://github.com/N4si/K8s-voting-app.git
```

Create CloudChamp Namespace

```
kubectl create ns cloudchamp

kubectl config set-context --current --namespace cloudchamp
```

MONGO Database Setup

To create Mongo statefulset with Persistent volumes, run the command in manifests folder:

```
kubectl apply -f mongo-statefulset.yaml
```

Mongo Service

```
kubectl apply -f mongo-service.yaml
```

Create a temporary network utils pod. Enter into a bash session within it. In the terminal run the following command:

```
kubectl run --rm utils -it --image praqma/network-multitool -- bash
```

Within the new utils pod shell, execute the following DNS queries:

```
for i in {0..2}; do nslookup mongo-$i.mongo; done
```

Note: This confirms that the DNS records have been created successfully and can be resolved within the cluster, 1 per MongoDB pod that exists behind the Headless Service - earlier created.

Exit the utils container

```
exit
```

On the `mongo-0` pod, initialise the Mongo database Replica set. In the terminal run the following command:

```
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
```

Note: Wait until this command completes successfully, it typically takes 10-15 seconds to finish, and completes with the message: bye

To confirm run this in the terminal:

```
kubectl exec -it mongo-0 -- mongo --eval "rs.status()" | grep
"PRIMARY\|SECONDARY"
```

Load the Data in the database by running this command:

Note: use langdb not langdb() as shown in the video

```
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, "homepage" :
"https://dotnet.microsoft.com/learn/csharp", "download" :
"https://dotnet.microsoft.com/download/", "votes" : 0}});
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}});
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}});
db.languages.insert({"name" : "go", "codedetail" : { "usecase" : "system, web,
server-side", "rank" : 12, "compiled" : true, "homepage" :
"https://golang.org", "download" : "https://golang.org/dl/", "votes" : 0}});
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}});
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}});

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

Create Mongo secret:

```
kubectl apply -f mongo-secret.yaml
```

API Setup

Create GO API deployment by running the following command:

```
kubectl apply -f api-deployment.yaml
```

Expose API deployment through service using the following command:

```
kubectl expose deploy api \
 --name=api \
 --type=LoadBalancer \
 --port=80 \
 --target-port=8080
```

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
}
```

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:

```
curl -s $API_ELB_PUBLIC_FQDN/languages | jq .
curl -s $API_ELB_PUBLIC_FQDN/languages/go | jq .
curl -s $API_ELB_PUBLIC_FQDN/languages/java | jq .
curl -s $API_ELB_PUBLIC_FQDN/languages/nodejs | jq .
```

If everything works fine, go ahead with Frontend setup.

```
{
API_ELB_PUBLIC_FQDN=$(kubectl get svc api
-ojsonpath="{.status.loadBalancer.ingress[0].hostname}")
```

```
echo API_ELB_PUBLIC_FQDN=$API_ELB_PUBLIC_FQDN
}
```

Frontend setup

Create the Frontend Deployment resource. In the terminal run the following command:

```
kubectl apply -f frontend-deployment.yaml
```

Create a new Service resource of LoadBalancer type. In the terminal run the following command:

```
kubectl expose deploy frontend \
 --name=frontend \
 --type=LoadBalancer \
 --port=80 \
 --target-port=8080
```

Confirm that the Frontend ELB is ready to recieve HTTP traffic. In the terminal run the following command:

```
{
FRONTEND_ELB_PUBLIC_FQDN=$(kubectl get svc frontend
-ojsonpath="{.status.loadBalancer.ingress[0].hostname}")
until nslookup $FRONTEND_ELB_PUBLIC_FQDN >/dev/null 2>&1; do sleep 2 && echo
waiting for DNS to propagate...; done
curl -I $FRONTEND_ELB_PUBLIC_FQDN
}
```

Generate the Frontend URL for browsing. In the terminal run the following command:

```
echo http://$FRONTEND_ELB_PUBLIC_FQDN
```

Test the full end-to-end cloud native application

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()"
```

Summary

In this Project, you learnt how to deploy a cloud native application into EKS. Once deployed and up and running, you used your local workstation's browser to test out the application. You later confirmed that your activity within the application generated data which was captured and recorded successfully within the MongoDB ReplicaSet back end within the cluster.