CSYE 7220 DevOps

Group:- 4
Group Members:-
Sapna Patel (001023371)
Shivam Thakkar (001023257)

For CI/CD implementation with Circle.ci

How to run the project

1. Infra setup

Here we are using Amazon Elastic Kubernetes Service as Infrastructure where the Kubernetes pods will be made run.

Go to Terraform folder and run the below steps:-

bash
$ terraform init

bash
$ terraform apply -auto-approve

2. CI/CD automation using CircleCI

Once the infrastructure is UP and running >>>

Then go to [CircleCI](https://circleci.com/) and link it with your Github account. Add following environment variables -

1. AWS_ACCESS_KEY_ID - Your AWS account access key
2. AWS_DEFAULT_REGION - AWS account default region e.g us-east-1
3. AWS_REGION - similar to AWS_DEFAULT_REGION
4. AWS_SECRET_ACCESS_KEY - Your AWS account secret access key
5. DockerUsername - your docker hub repo username
6. DockerPassword - your docker hub repo password

Configuring local machine with the infrastructure created
bash
$ terraform output kubeconfig > C:/Users/<username>/.kube/config-terraform-eks-demo
$ terraform output config_map_aws_auth > ./config-map-aws-auth.yml

Take the backup of already present kube config file
bash
$ cp C:/Users/<username>/.kube/config C:/Users/<username>/.kube/config.bak

bash
$ cp C:/Users/<username>/.kube/config-terraform-eks-demo C:/Users/<username>/.kube/config

To check your local machine connected with EKS cluster
bash
$ kubectl get nodes

Now these files (config-map-aws-auth.yml and config) needs to be present in the root folder of github repository. Push this files to github repository
bash
$ git add --all
$ git commit -m "Adding kube config & config-map-aws-auth.yml"
$ git push origin <branch name>

It will automatically trigger CircleCI pipeline. And the dashboard of CircleCI will help you understand the code build stats very crystal and clear.

Detailed work implemented in config.yml (.cicleci folder) -

After any application code update pushed in Github repo will initiate the steps below:

Workflow 1:

* Code checkout from Github
* Installions
* Creating docker image for backend and pushing it to docker hub image repo
* Kubectl configuration to apply deployments and load balancer to back-end layer
* Similarly, creating docker image for frontend and pushing to docker hub
* Kubectl configuration to apply deployments and load balancer to front-end layer which will be automatically linked with back-end layer

For Load testing with AKS using Apache Bench and Jmeter

Go to AKS repository where *main.tf* file resides.

Step 1:-

bash
$ az login
$ terraform init
$ terraform plan
$ terraform apply
Note:- Using the azure service principle credentials

Step 2:- Configuring kubectl for AKS
For output config file, for windows apply

bash
$ terraform output kube_config > C:\Users\cvam6\.kube\config-terraform-aks-demo

Take a backup of your current config file and then open the generated file and remove the *EOD* from the start of the file and end of the file. And after that copy all from that file and paste into the config file in location(for us)
C:\Users\cvam6\.kube\config

Step 3:-
Horizontal autoscalling
Apply combo file of deployment and service

bash
$ kubectl apply -f app-combo.yaml
$ kubectl describe deploy final-rockstar-app

Step 4:-
Auto scaling via YAML
$ kubectl create -f app-scaler.yaml
$ kubectl describe hpa final-rockstar-app

Step 5:-
Create a namespace
$ kubectl create namespace monitoring

Step 6:-
Cluster Role
$ kubectl create -f clusterRole.yaml

Step 7:-
Create permissions using Config-map
$ kubectl create -f config-map.yaml

Step8:-
Prometheus Deployment
$ kubectl create -f prometheus-deployment.yaml --namespace=monitoring
$ kubectl create -f prometheus-service.yaml --namespace=monitoring

Get prometheus external IP with this command
$ kubectl get svc --namespace=monitoring

Step 9:-
$ kubectl get svc
And get the External IP for the final-rockstar-app

For Apache Bench load testing
$ ab -c 50 -n 10000 <external_ip_final-rockstar-app>

And for Jmeter testing
Set number of threads to 50
Loop count to 10000
And run a test.

To monitor the Horizontal Pod Autoscaling,
bash
$ kubectl get hpa

This will give the result of a number of pods increment and target load.

To see the working of the whole application, we need to deploy the service of all backend applications. For that, we need to perform the following commands for all application deployment and services.

For flask application,
$ kubectl apply -f .\sa-flask-logic-deployment.yml
$ kubectl apply -f .\sa-flask-logic-service.yml

For Java Application,
$ kubectl apply -f .\sa-java-webapp-deployment.yml
$ kubectl apply -f .\sa-java-webapp-service.yml

For Dotnet application,
$ kubectl apply -f .\rs-dotnet-api-deployment.yml
$ kubectl apply -f .\rs-dotnet-api-service.yml

Then run,
$ kubectl get svc
To get external IP of all applications

And then go to the browser and go to the below link,
http://<final-rockstar-app_extrenal_ip>?javaWebApp=http://<sa-java-webapp_external_ip>&dotnetWebApi=http://<rs-dotnet-api_external_ip>

And you can go through the whole working full-stack application.