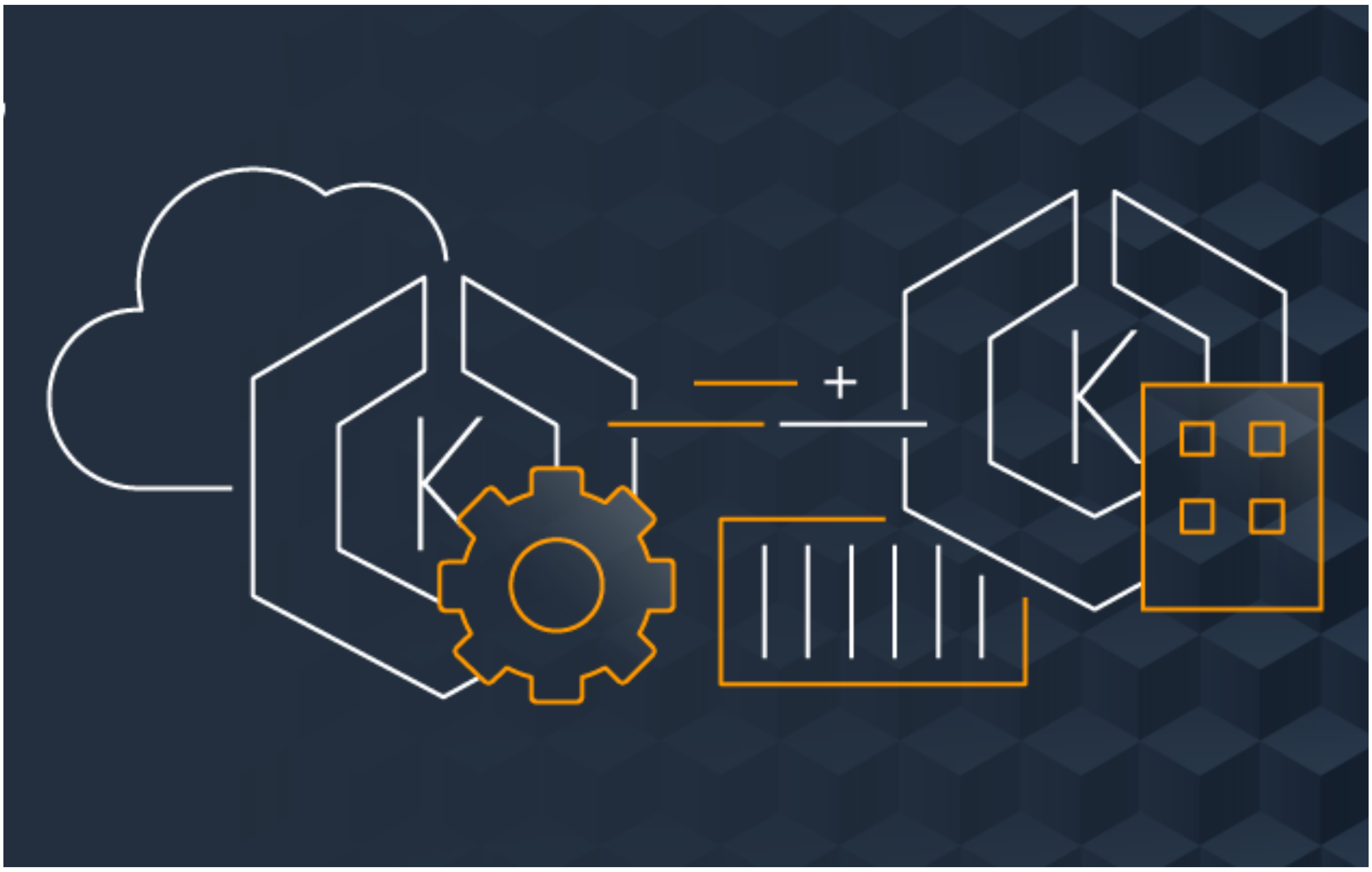


Name : Jitesh Jayendra Kale

Project : DevOps Capstone Project

Project Directory : https://bit.ly/37SrxjW

Github repo : https://github.com/iamG2kale/devops_capstone.git

Infrastructure and Code files : https://bit.ly/3wk29vR

Task 1 Screens : https://bit.ly/3y6Cmtq

Task 2 Screens : https://bit.ly/3MNXhFS

Task 3 Screens : https://bit.ly/38LByj9

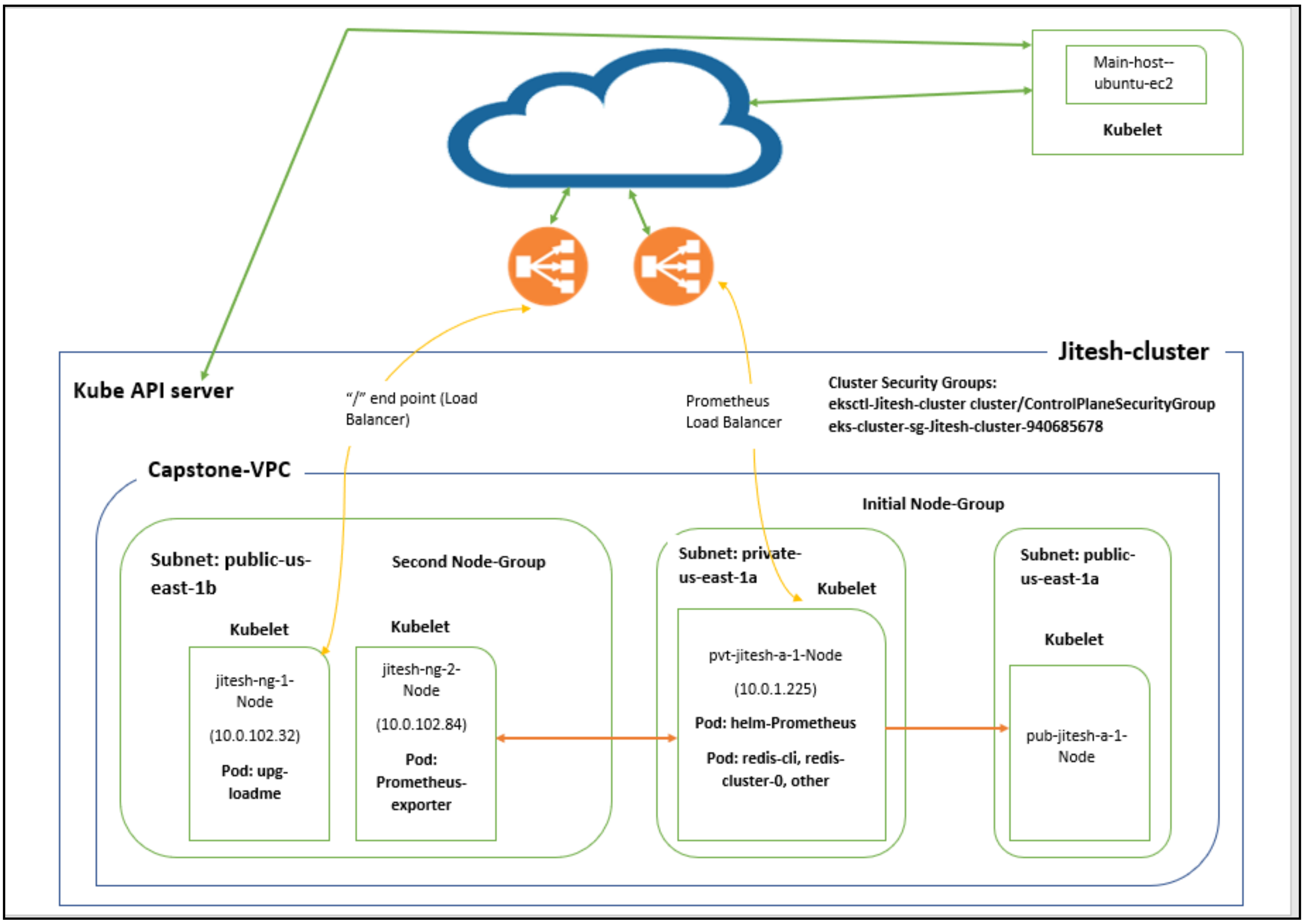
Task 4 Screens : https://bit.ly/3kwpLI5

DevOps Final Module

Capstone Project

Objective

To successfully create a POC capstone project around the setup of Kubernetes cluster on AWS cloud and mapping applications on Kubernetes. The final goal of this project is to Setup VPC and infrastructure using terraform for Kubernetes cluster on AWS, use ekstcl to setup EKS cluster, manage stateless and stateful workloads, implement metric systems and autoscale nodes and workload on Kubernetes with keeping budget management in mind.



Overall Architecture

Tech Stack

AWS, S3, ECS, EKS, Load balancers, terraform, Kubernetes, ekstcl, helm, aws-iam-authenticator, kubectl, helm, docker, apache benchmark, Kubernetes-aws add-ons

Task 1

Link to folder : https://bit.ly/37SrxjW

Task 1 involved four subtasks each focusing on building infrastructure and setting up architecture for the project. Task 1 focuses on infrastructure via terraform and creation of EKS cluster using eksctl tool along with add ons.

Sub-task 1 :

1. AWS-CLI was installed on ec2 ubuntu machine and credentials, account details where configured using aws configure with region us-east-1.

2. S3 bucket was initialised manually with name “capstone-infra” to store terraform state file that was used dynamically to create entire infrastructure.

Sub-task 2 :

1. AWS provider was used to setup AWS with us-east-1 region and S3 to store terraform.state file dynamically.

2. VPC module was used to create VPC [cidr = “10.0.0.0/16"] with name “capstone-vpc” along with 2 public ["10.0.101.0/24", "10.0.102.0/24"] and private subnets ["10.0.1.0/24", "10.0.2.0/24"] in two availability zones ["us-east-1a", “us-east-1b”] with one NAT gateway attached to private subnets, 1 Internet Gateway attached to public subnet with properly configured route tables.

3. Correct tagging was done on public and private subnets for AWS EKS and creating load balancer with load balancer controller add-on.

Sub-task 3 :

1. Eksctl tool was installed on ubuntu ec2 machine.

2. A cluster with name “Jitesh-cluster’ was creating using eksctl with the help of YAML file.

3. Correct VPC and Subnet ids were used in clusterConfig yaml file along with OIDC association as true for access IAM and policies.

4. Two node-groups were created with t2.medium configuration and were tagged properly to use with cluster-autoscaler as per requirement.

Sub-task 4 :

1. Three add-ons were installed on top of Kubernetes cluster so as to use with load balancer, apache bench and helm in later tasks.

2. Kubernetes and AWS provided YAML files were used to create deployments and install mentioned add-ons.

EKS cluster security groups and configuration

Cluster created in total 10 security groups for communication between Kubernetes master components and worker components.

GroupName	eks-cluster-sg-Jitesh-cluster-940685678
VpcId	vpc-0f3d59dad874ea5c7
Description	EKS created security group applied to ENI that is attached to EKS Control Plane master nodes, as well as any managed workloads.
InboundRulesCount	2
OutboundRulesCount	1

GroupName	eksctl-Jitesh-cluster-nodegroup-jitesh-ng-2-SG-1Q1KL1QXZF8FQ
VpcId	vpc-0f3d59dad874ea5c7
Description	Communication between the control plane and worker nodes in group jitesh-ng-2
InboundRulesCount	5
OutboundRulesCount	1

GroupName	eksctl-Jitesh-cluster-nodegroup-pub-jitesh-a-1-SG-1QXW2STX64IUW
VpcId	vpc-0f3d59dad874ea5c7
Description	Communication between the control plane and worker nodes in group pub-jitesh-a-1
InboundRulesCount	2
OutboundRulesCount	1

GroupName	eksctl-Jitesh-cluster-cluster-ControlPlaneSecurityGroup-1T58A49KJT6OA
VpcId	vpc-0f3d59dad874ea5c7
Description	Communication between the control plane and worker nodegroups
InboundRulesCount	4
OutboundRulesCount	8

GroupName	eksctl-Jitesh-cluster-nodegroup-pvt-jitesh-a-1-SG-VE1XW7961E0K
VpcId	vpc-0f3d59dad874ea5c7
Description	Communication between the control plane and worker nodes in group pvt-jitesh-a-1
InboundRulesCount	4
OutboundRulesCount	1

GroupName	eksctl-Jitesh-cluster-nodegroup-jitesh-ng-1-SG-BZLI186TAJZ2
VpcId	vpc-0f3d59dad874ea5c7
Description	Communication between the control plane and worker nodes in group jitesh-ng-1
InboundRulesCount	4
OutboundRulesCount	1

GroupName	eksctl-Jitesh-cluster-cluster-ClusterSharedNodeSecurityGroup-1X73LUXXMZUPT
VpcId	vpc-0f3d59dad874ea5c7
Description	Communication between all nodes in the cluster
InboundRulesCount	2
OutboundRulesCount	1

GroupName	k8s0-traffic-Jiteshcluster-4123dc325c
VpcId	vpc-0f3d59dad874ea5c7
Description	[k8s] shared backend SecurityGroup for LoadBalancer
InboundRulesCount	0
OutboundRulesCount	1

GroupName	k8s-elb-a212a9de0856d42c19f13ec30923ff6c
VpcId	vpc-0f3d59dad874ea5c7
Description	ecurity group for Kubernetes ELB a212a9de0856d42c19f13ec30923ff6c (demo/prometheus-kube-prometheus-prometheus)
InboundRulesCount	2
OutboundRulesCount	1

GroupName	k8s-demo-upgloadm-5873e21567
VpcId	vpc-0f3d59dad874ea5c7
Description	k8s] Managed SecurityGroup for LoadBalancer
InboundRulesCount	1
OutboundRulesCount	1

Task 2

Link to folder : https://bit.ly/3MNXhFS

Task 2 involved three subtasks each focusing on setting up ECR repository, building/running upg-load-me app image/container, pushing the same to ECR repo, creating additional node-groups with given config and creating deployment, ingress and services for the load-me app.

Sub-task 1 :

1. ECR repository with name “loadme” was creating to store image of the load-me app creating using docker.

2. Dockerfile was written for node load-me app.

3. Image was tagged and pushed into ECR repo and same was tested on the local machine on the browser.

Sub-task 2 :

1. Node-group was created using eksctl command line with the help of written YAML file.

2. Correct type of instances with desired count, min, max configuration was set along with correct taint mechanism to be used with nodes.

Sub-task 3 :

1. “Demo” namespace was created in Kubernetes.

2. Upg-load-me deployment, service and ingress with load balancer was created and deployed using YAML files.

3. Correct Tolerations and Affinity configuration was mapped in deployment YAML so as to work with taints created in node-group.

4. Deployment used docker image from ECR stored in earlier sub-task and app was run and accessed via load balancer successfully at endpoint “/“.

Task 3

Link to folder : https://bit.ly/38LByj9

Task 3 involved three subtasks each focusing on deploying redis-server with statefulset, PVC and configmap, deploying redis-cli with redis docker image with restartPolicy configuration, connect redis service from cli and do required operations all in demo namespace.

Sub-task 1 :

1. Statefulset Redis server was created along with service, configMap, PVC and required attributes.
2. Service created used clusterIP to access redis-server pods on port 6379.
3. Official redis:latest docker image was used.

Sub-task 2 :

1. Single redis for redis-cli was deployed in demo namespace using redis docker image.
2. Mentioned commands were passed with restart policy always attribute in redis deployment.
3. Kubectl exec command was used to sh into pod of redis-cli deployment.

Sub-task 3 :

1. Redis-cli was connected to redis service using IP using command line.
2. Key : foo, value :1 was stored into redis server and output was also shown using get foo statement.
3. Redis-cli pod was restarted and reconnected to test if key-value pair is present.
4. After doing ‘get foo’ on reconnected service, output as 1 was printed and data was present.

Task 4

Link to folder : https://bit.ly/3kwpLI5

Task 4 involved three subtasks each focusing on deploying Horizontal Pod Scaler for upg-loadme app, installing prometheus and grafana using helm, Load test the HPA pods using apache benchmark tool and visualise load.

Sub-task 1 :

1. HPA for upg-loadme app was deployed with given configuration using YAML file.

Sub-task 2 :

1. Prometheus and Grafana was installed using Helm charts and Prometheus port was forwarded as required,

2. Also, Prometheus service was changed from clusterIP to use load balancer with automatic internal port mapping.

Sub-task 3 :

1. Apache Benchmark was install and used to create load on the app with scale as 100 using load balancer dns and load parameter.

2. Autoscaling (Scaling Up and Down) based on the load > 50 % and duration was captured using top pods command.

3. App was again tested with load as 9000 to for better visualisation of the same.

4. Graphs were captured in prometheus during load testing. Total cpu usage, memory usage and scaling up and down scaling graphs were captured