==========EKS SETUP==========

1. Creating the EKS service IAM role
 a. Create a EKSIAM role for VPC (option:Allows access to other AWS service resources that are required to operate clusters managed by EKS) and attach these 2 policies and administratoraccess
  AmazonEKSClusterPolicy
  AmazonEKSServicePolicy
 b. Create another EKSCluster role in the same way and keep only AmazonEKSClusterPolicy for Cluster Creation

2. Creating the VPC infrastructure using CloudFormation
 - VPC, RouteTable, SecurityGroup, 3 Subnets
 * Create stack with following s3 location as template, keep all the options as it is
  https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-10-29/amazon-eks-vpc-sample.yaml

 Note: While creating the stack, under the permissions page if the IAMRole created doesnt showup then
  a. Goto IAMRole, "Trust relationships" tab, If cloudformation.amazonaws.com isn't listed as a trusted entity in the bottom, then choose "Edit trust relationship"
  b. In the Policy Document editor, enter the following AWS CloudFormation service role trust policy:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Service": "cloudformation.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

  c. Choose "Update Trust Policy"

 Note: If you get error while creating VPC/IGW give full admin policy access to the IAM Role

3. Creating a ControlPlane & cluster in the AWS Management Console
 - Choose the IAM Role / VPC / Subnet / SecuringGroup created above

4. Configuring kubectl for EKS
 - Create a EC2 workstation for EKS
 - Install Kubectl (https://docs.aws.amazon.com/eks/latest/userguide/install-kubectl.html)
 $ curl -o kubectl https://amazon-eks.s3.us-west-2.amazonaws.com/1.17.11/2020-09-18/bin/linux/amd64/kubectl
 $ chmod +x kubectl && sudo mv kubectl /usr/bin/kubectl
 $ kubectl version --short --client

 - Install AWS CLI
 $ curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
 $ sudo apt install unzip && unzip awscliv2.zip
 $ sudo ./aws/install --bin-dir /usr/bin --install-dir /usr/bin/aws-cli --update
 $ aws --version
 --- RUN as UBUNTU user ---------
 $ aws configure
 $ aws eks list-clusters
 $ aws eks update-kubeconfig --name wezvatechQA
Added new context arn:aws:eks:ap-south-1:184583194367:cluster/DemoEKSCluster to /home/ubuntu/.kube/config

```
 $ kubectl config view
 $ kubectl cluster-info
Kubernetes master is running at https://4748266046766D1BC244740EE296AA39.sk1.ap-south-1.eks.amazonaws.com
CoreDNS is running at https://4748266046766D1BC244740EE296AA39.sk1.ap-south-1.eks.amazonaws.com/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy

5. Provisioning Worker Nodes
 - Choose the VPC, EKS Cluster, SG & 3 subnets created
  https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-10-29/amazon-eks-nodegroup.yaml
 - For NodeImageId, choose the AMI based on K8s version & AWS region (ami-0cd8562db082e8c1a)
  https://docs.aws.amazon.com/eks/latest/userguide/eks-optimized-ami.html
  (/aws/service/eks/optimized-ami/1.21/amazon-linux-2/recommended/image_id)
 * Once the Nodes stack is created make a note of NodeInstanceRole from outputs:    arn:aws:iam::184583194367:role/DemoEKSNodes-NodeInstanceRole-1XFAAKMTKNJJ

 * Apply the aws-auth ConfigMap to your cluster (https://docs.aws.amazon.com/eks/latest/userguide/add-user-role.html)
 $ curl -o aws-auth-cm.yaml https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-08-12/aws-auth-cm.yaml
 curl -o aws-auth-cm.yaml https://amazon-eks.s3.us-west-2.amazonaws.com/cloudformation/2020-10-29/aws-auth-cm.yaml
 - Edit the file & add the NodesInstanceRole ARN in it
 $ kubectl apply -f aws-auth-cm.yaml
 $ kubectl get nodes --watch

* Install Helm
$ curl https://get.helm.sh/helm-v3.2.3-linux-amd64.tar.gz > helm.tar.gz
$ tar xzvf helm.tar.gz
$ sudo mv linux-amd64/helm /usr/local/bin

==========[ INGRESS Setup ]==========

1. Create a IAM Policy
$ curl -o iam_policy.json https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.2.0/docs/install/iam_policy.json
$ aws iam create-policy \
   --policy-name AWSLoadBalancerControllerIAMPolicy \
   --policy-document file://iam_policy.json

2. Create a rbac-role.yml file with the following content & update with the above policy ARN at the end in Serviceaccount manifest
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  labels:
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
rules:
  - apiGroups:
      - ""
      - extensions
    resources:
```

```
      - configmaps
      - endpoints
      - events
      - ingresses
      - ingresses/status
      - services
      - pods/status
    verbs:
      - create
      - get
      - list
      - update
      - watch
      - patch
  - apiGroups:
      - ""
      - extensions
    resources:
      - nodes
      - pods
      - secrets
      - services
      - namespaces
    verbs:
      - get
      - list
      - watch
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  labels:
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: aws-load-balancer-controller
subjects:
  - kind: ServiceAccount
    name: aws-load-balancer-controller
    namespace: kube-system
---
apiVersion: v1
kind: ServiceAccount
metadata:
  labels:
    app.kubernetes.io/name: aws-load-balancer-controller
  name: aws-load-balancer-controller
  namespace: kube-system
  annotations:
    eks.amazonaws.com/role-arn: <your-iam-role-arn-for-alb-ingress-here>

$ kubectl apply -f rbac-role.yml
```

* Install eksctl on Ubuntu

```
$ curl --silent --location "https://github.com/weaveworks/eksctl/releases/latest/download/eksctl_$(uname -s)_amd64.tar.gz" | tar xz -C /tmp
$ sudo mv /tmp/eksctl /usr/local/bin
$ eksctl version
```

3. Create IAM OIDC provider:

```
$ eksctl utils associate-iam-oidc-provider \
    --region ap-south-1 \
    --cluster wezvatechQA  \
    --approve
```

4. Create IAM Service Account & refer to the policy created at 1

```
$ eksctl create iamserviceaccount \
  --cluster wezvatechQA \
  --namespace kube-system \
  --name aws-load-balancer-controller \
  --attach-policy-arn arn:aws:iam::249397932281:policy/AWSLoadBalancerControllerIAMPolicy \
  --override-existing-serviceaccounts \
  --approve
$ eksctl get iamserviceaccount --cluster wezvatechQA
$ kubectl describe sa aws-load-balancer-controller -n kube-system
```

5. Install the TargetGroupBinding CRDs

```
$ kubectl apply -k github.com/aws/eks-charts/stable/aws-load-balancer-controller/crds?ref=master
$ kubectl get crd
```

6. Deploy AWS Load Balancer Controller (ALB Ingress Controller)

```
$ helm repo add eks https://aws.github.io/eks-charts
$ helm upgrade -i aws-load-balancer-controller \
    eks/aws-load-balancer-controller \
    -n kube-system \
    --set clusterName=wezvatechQA \
    --set serviceAccount.create=false \
    --set serviceAccount.name=aws-load-balancer-controller \

$ kubectl -n kube-system rollout status deployment aws-load-balancer-controller
$ kubectl get pods -n kube-system
```

7. Deploy sample application

```
$ curl -o demoapp.yml https://raw.githubusercontent.com/kubernetes-sigs/aws-load-balancer-controller/v2.2.0/docs/examples/2048/2048_full.yaml
 -- change the replica to 2
$ kubectl apply -f demoapp.yml
$ kubectl get ingress/ingress-2048 -n game-2048
```

---CLEANUP-----

```
$ kubectl delete -f demoapp.yml
$ helm uninstall aws-load-balancer-controller -n kube-system
$ kubectl delete -k github.com/aws/eks-charts/stable/aws-load-balancer-controller/crds?ref=master
$ eksctl delete iamserviceaccount \
    --cluster wezvatechQA \
    --name aws-load-balancer-controller \
    --namespace kube-system \
    --wait
```

```
$ aws iam delete-policy \
   --policy-arn arn:aws:iam::249397932281:policy/AWSLoadBalancerControllerIAMPolicy

==========================================[ Cluster Backup ]==========================================
1. Create a S3 bucket
2. Install Velero Client
$ wget https://github.com/vmware-tanzu/velero/releases/download/v1.3.2/velero-v1.3.2-linux-amd64.tar.gz
$ tar -xvf velero-v1.3.2-linux-amd64.tar.gz -C /tmp
$ sudo mv /tmp/velero-v1.3.2-linux-amd64/velero /usr/local/bin
$ velero version

3. Install Velero on EKS
$ velero install \
   --provider aws \
   --plugins velero/velero-plugin-for-aws:v1.0.1 \
   --bucket <S3BucketName> \
   --backup-location-config region=ap-south-1 \
   --snapshot-location-config region=ap-south-1 \
   --secret-file /home/ubuntu/.aws/credentials

$ kubectl logs -f deployment/velero -n velero

4. Deploy test app
$ kubectl create ns demo
$ kubectl apply -f testpod.yml

5. Create Backup
$ velero backup create backup.1 --include-namespaces demo
$ velero backup describe backup.1
- Verify the backup folder inside s3 bucket
$ velero backup get

6. Create a disaster scenario i.e delete namespace
$ kubectl delete ns demo

7. Restore from backup
$ velero restore create --from-backup backup.1
$ kubectl get ns

----CleanUp----
$ velero backup delete backup.1
$ kubectl delete ns demo

==================================================[ CLUSTER AUTOSCALER SETUP ]==================================================
1. Deploy Metrics server
$ kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.5.0/components.yaml
$ kubectl get pods -n kube-system
$ kubectl top pods or kubectl top nodes

2. Configure Autoscaling Group
- Make a note of existing Autoscaling Group created by EKS Worker Stack
- Edit IAM role associated to EKS node & add an inline IAM policy with the following document
{
```

```
  "Version": "2012-10-17",
  "Statement": [
    {
      "Action": [
        "autoscaling:DescribeAutoScalingGroups",
        "autoscaling:DescribeAutoScalingInstances",
        "autoscaling:DescribeLaunchConfigurations",
        "autoscaling:DescribeTags",
        "autoscaling:SetDesiredCapacity",
        "autoscaling:TerminateInstanceInAutoScalingGroup",
        "ec2:DescribeLaunchTemplateVersions"
      ],
      "Resource": "*",
      "Effect": "Allow"
    }
  ]
}
```

3. Download the yaml & add the Cluster name at the end, apply the yaml

```
$ wget https://raw.githubusercontent.com/kubernetes/autoscaler/master/cluster-autoscaler/cloudprovider/aws/examples/cluster-autoscaler-autodiscover.yaml
          - --nodes=1:6:<Autoscaling Group Name>
$ kubectl apply -f cluster-autoscaler-autodiscover.yaml
$ kubectl get pods -n kube-system
$ kubectl logs -f <podname> -n kube-system
```

4. Deploy sample application and increase the replicas to add load

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-to-scaleout
spec:
  replicas: 1
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        service: nginx
        app: nginx
    spec:
      containers:
      - image: nginx
        name: nginx-to-scaleout
        resources:
          limits:
            cpu: 500m
            memory: 512Mi
          requests:
            cpu: 500m
            memory: 512Mi
```

---CLEANUP-----

```
$ kubectl delete -f cluster-autoscaler-autodiscover.yaml
```

- Remove the inline policy from the IAM Role for the worker node group
$ kubectl delete -f https://github.com/kubernetes-sigs/metrics-server/releases/download/v0.5.0/components.yaml