

DevOps Interview Preparation (Quick Glance)

AWS

Check Availability Zone Availability

```
aws ec2 describe-instance-type-offerings \
  --location-type availability-zone \
  --filters "Name=instance-
type,Values=$instance_type"
"Name=location,Values=$1" \
  --region $region \
  --query
'InstanceTypeOfferings[?InstanceType=='${ins
tance_type}'].Location' \
  --output text
```

Key Pairs

Check if Key Pair Exists:

```
if ! aws ec2 describe-key-pairs --key-names
${key_pair_name} --region ${region}
&>/dev/null; then
```

Create Key Pair:

```
aws ec2 create-key-pair --key-name
${key_pair_name} --query 'KeyMaterial' --
output text --region ${region} >
CentosComplexKeyPair.pem
```

Set Permissions:

```
chmod 400 CentosComplexKeyPair.pem
```

VPC

Describe VPCs:

```
aws ec2 describe-vpcs --filters
"Name=cidr,Values=${vpc_cidr}" --query
'Vpcs[0].VpcId' --output text --region
${region}
```

Create VPC:

```
aws ec2 create-vpc --cidr-block ${vpc_cidr} -
-query 'Vpc.VpcId' --output text --region
${region}
```

Internet Gateway

Check if Internet Gateway Exists:

```
igw_id=$(aws ec2 describe-internet-gateways -
-filters "Name=attachment.vpc-
id,Values=${vpc_id}" --query
'InternetGateways[0].InternetGatewayId' --
output text --region ${region})
if [ "$igw_id" == "None" ]; then
```

Create Internet Gateway:

```
igw_id=$(aws ec2 create-internet-gateway --
query 'InternetGateway.InternetGatewayId' --
output text --region ${region})
```

Attach Internet Gateway:

```
aws ec2 attach-internet-gateway --internet-
gateway-id ${igw_id} --vpc-id ${vpc_id} --
region ${region}
```

Subnets

Check if Public Subnet 1 Exists:

```
public_subnet_id_1=$(aws ec2 describe-subnets
--filters "Name=vpc-id,Values=${vpc_id}"
"Name=cidr-
block,Values=${public_subnet_cidr_1}" --query
'Subnets[0].SubnetId' --output text --region
${region})
if [ "$public_subnet_id_1" == "None" ]; then
```

Create Public Subnet 1:

```
public_subnet_id_1=$(aws ec2 create-subnet --
vpc-id ${vpc_id} --cidr-block
${public_subnet_cidr_1} --availability-zone
${available_zone_1} --query 'Subnet.SubnetId'
--output text --region ${region})
```

Route Tables

Check if Route Table for Public Subnet 1 Exists:

```
public_route_table_id_1=$(aws ec2 describe-
route-tables --filters "Name=vpc-
id,Values=${vpc_id}"
"Name=association.subnet-
id,Values=${public_subnet_id_1}" --query
```

```
'RouteTables[0].RouteTableId' --output text -
-region ${region})
if [ "$public_route_table_id_1" == "None" ];
then
```

Create Route Table for Public Subnet 1:

```
public_route_table_id_1=$(aws ec2 create-
route-table --vpc-id ${vpc_id} --query
'RouteTable.RouteTableId' --output text --
region ${region})
```

Associate Route Table with Public Subnet 1:

```
aws ec2 associate-route-table --route-table-
id ${public_route_table_id_1} --subnet-id
${public_subnet_id_1} --region ${region}
```

Create Route in Route Table for Public Subnet 1:

```
aws ec2 create-route --route-table-id
${public_route_table_id_1} --destination-
cidr-block 0.0.0.0/0 --gateway-id ${igw_id} -
-region ${region}
```

NAT Gateway

Allocate Elastic IP:

```
eip_allocation_id_1=$(aws ec2 allocate-
address --domain vpc --query 'AllocationId' -
-output text --region ${region})
```

Create NAT Gateway:

```
nat_gateway_id_1=$(aws ec2 create-nat-gateway
--subnet-id ${public_subnet_id_1} --
allocation-id ${eip_allocation_id_1} --query
'NatGateway.NatGatewayId' --output text --
region ${region})
```

Update Private Route Table 1:

```
aws ec2 create-route --route-table-id
${private_route_table_id_1} --destination-
cidr-block 0.0.0.0/0 --nat-gateway-id
${nat_gateway_id_1} --region ${region}
echo "Updated Private Route Table 1 to use
NAT Gateway 1"
```

Security Groups

Check if Bastion Security Group Exists:

```
bastion_security_group_id=$(aws ec2 describe-
security-groups --filters "Name=vpc-
id,Values=${vpc_id}" "Name=group-
name,Values=${bastion_security_group_name}" -
-query 'SecurityGroups[0].GroupId' --output
text --region ${region})
if [ "$bastion_security_group_id" == "None"
]; then
```

Create Bastion Security Group:

```
bastion_security_group_id=$(aws ec2 create-
security-group --group-name
${bastion_security_group_name} --description
"Bastion security group" --vpc-id ${vpc_id} -
-query 'GroupId' --output text --region
${region})
```

Add Inbound Rules to Bastion Security Group:

```
aws ec2 authorize-security-group-ingress --
group-id ${bastion_security_group_id} --
protocol tcp --port 22 --cidr 0.0.0.0/0 --
region ${region}
```

Check if Application Security Group Exists:

```
app_security_group_id=$(aws ec2 describe-
security-groups --filters "Name=vpc-
id,Values=${vpc_id}" "Name=group-
name,Values=${app_security_group_name}" --
query 'SecurityGroups[0].GroupId' --output
text --region ${region})
if [ "$app_security_group_id" == "None" ];
then
```

Create Application Security Group:

```
app_security_group_id=$(aws ec2 create-
security-group --group-name
${app_security_group_name} --description
"Application security group" --vpc-id
${vpc_id} --query 'GroupId' --output text --
region ${region})
```

Add Inbound Rules to Application Security Group:

```
aws ec2 authorize-security-group-ingress --group-id ${app_security_group_id} --protocol tcp --port 22 --source-group ${bastion_security_group_id} --region ${region}
aws ec2 authorize-security-group-ingress --group-id ${app_security_group_id} --protocol tcp --port 80 --cidr 0.0.0.0/0 --region ${region}
```

IAM Role

Trust Policy:

```
cat > trust-policy.json <<EOF
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "ec2.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    },
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
EOF
```

Create Role:

```
aws iam create-role --role-name ${role_name} --assume-role-policy-document file://trust-policy.json --region ${region}
```

Attach Policy:

```
aws iam attach-role-policy --role-name ${role_name} --policy-arn ${policy_arn} --region ${region}
```

Create Instance Profile:

```
aws iam create-instance-profile --instance-profile-name ${instance_profile_name} --region ${region}
```

Add Role to Instance Profile:

```
aws iam add-role-to-instance-profile --instance-profile-name ${instance_profile_name} --role-name ${role_name} --region ${region}
```

Launch EC2 Instance with Instance Profile:

```
aws ec2 run-instances --image-id ami-0abcdef1234567890 --count 1 --instance-type t2.micro --iam-instance-profile Name=MyInstanceProfile --region us-west-2
```

Placement Group

Create Placement Group:

```
aws ec2 create-placement-group --group-name ${placement_group_name} --strategy spread --region ${region}
```

- --group-name \${placement_group_name} : Specifies the name of the placement group.

strategy spread : Specifies the placement strategy (spread in this case).

- --region \${region} : Specifies the AWS region.

Cluster Placement Group

Use Case: High-performance computing (HPC) applications, big data workloads, and applications that require high network throughput.

```
aws ec2 create-placement-group --group-name my-cluster-group --strategy cluster --region us-west-2
```

Spread Placement Group

Use Case: Applications that require high availability and need to be isolated from failures, such as critical applications.

```
aws ec2 create-placement-group --group-name my-spread-group --strategy spread --region us-west-2
```

Partition Placement Group

Use Case: Large distributed and replicated workloads, such as Hadoop, Cassandra, and Kafka.

```
aws ec2 create-placement-group --group-name my-partition-group --strategy partition --partition-count 3 --region us-west-2
```

This command creates a partition placement group named my-partition-group with 3 partitions in the us-west-2 region.

Launch Instances in the Partition Placement Group:

```
aws ec2 run-instances --image-id ami-0abcdef1234567890 --count 3 --instance-type t2.micro --placement "GroupName=my-partition-group,PartitionNumber=0" --region us-west-2
aws ec2 run-instances --image-id ami-0abcdef1234567890 --count 3 --instance-type t2.micro --placement "GroupName=my-partition-group,PartitionNumber=1" --region us-west-2
aws ec2 run-instances --image-id ami-0abcdef1234567890 --count 3 --instance-type t2.micro --placement "GroupName=my-partition-group,PartitionNumber=2" --region us-west-2
```

S3 Bucket

Create S3 Bucket:

```
aws s3api create-bucket --bucket ${bucket_name} --region ${region} --create-bucket-configuration LocationConstraint=${region}
```

Create Sample File:

```
echo "This is a sample file for S3 bucket." > sample_file.txt
```

Upload Sample File:

```
aws s3 cp sample_file.txt s3://${bucket_name}/sample_file.txt --region ${region}
```

RDS

Create RDS Instance:

```
aws rds create-db-instance \
  --db-instance-identifier ${db_instance_identifier} \
  --db-instance-class ${db_instance_class} \
  --engine ${engine} \
  --master-username ${master_username} \
  --master-user-password ${master_user_password} \
  --allocated-storage 20 \
  --db-name ${db_name} \
  --vpc-security-group-ids ${app_security_group_id} \
  --db-subnet-group-name ${db_subnet_group_name} \
  --multi-az \
  --no-publicly-accessible \
  --region ${region}
```

Wait for Availability:

```
aws rds wait db-instance-available --db-instance-identifier ${db_instance_identifier} --region ${region}
```

Get RDS Endpoint:

```
db_endpoint=$(aws rds describe-db-instances -
-db-instance-identifier
${db_instance_identifier} --query
'DBInstances[0].Endpoint.Address' --output
text --region ${region})
echo "RDS instance endpoint: ${db_endpoint}"
```

Create DB Subnet Group:

```
aws rds create-db-subnet-group \
--db-subnet-group-name
${db_subnet_group_name} \
--db-subnet-group-description "My DB
Subnet Group" \
--subnet-ids ${private_subnet_id_1}
${private_subnet_id_2} \
--region ${region}
```

AWS CloudWatch

Create CloudWatch Alarm:

```
aws cloudwatch put-metric-alarm --alarm-name
${alarm_name} \
--metric-name CPUUtilization --namespace
AWS/EC2 \
--statistic Average --period 300 --
threshold 80 \
--comparison-operator
GreaterThanOrEqualToThreshold \
--dimensions
Name=InstanceId,Value=${instance_ids[0]} \
--evaluation-periods 2 --alarm-actions
${sns_topic_arn} \
--region ${region}
```

Launch Instances User Data Script:

```
cat > userDataCentOsComplex.sh <<EOF
#!/bin/bash
# Install httpd, unzip, and aws-cli
yum update -y
yum install -y httpd unzip aws-cli

# Start httpd service
systemctl start httpd
s
# Enable httpd service to start on boot
systemctl enable httpd

# Create a sample log file
echo "This is a sample log file." >
./sample_log.txt

# Upload the log file to S3 bucket
bucket_name=$(grep bucket_name
./resource_ids_centos.txt | cut -d'=' -f2)
aws s3 cp ./sample_log.txt
s3://${bucket_name}/sample_log.txt

# Download and unzip the website files
cd /var/www/html
wget
https://www.tooplate.com/download/2137_barist
a_cafe -O barista_cafe.zip
EOF
```

Launch Instances:

```
aws ec2 run-instances \
--image-id ami-0abcdef1234567890 \
--count 2 \
--instance-type t3.micro \
--key-name ${key_pair_name} \
--security-group-ids
${app_security_group_id} \
--subnet-id ${private_subnet_id_2} \
--user-data
file://userDataCentOsComplex.sh \
--tag-specifications
'ResourceType=instance,Tags=[{Key=Name,Value=
'${instance_name}_2'}]' \
--region ${region} \
--monitoring "Enabled=false" \
```

```
--iam-instance-profile
Name=${instance_profile_name} \
--block-device-mappings
'[{ "DeviceName": "/dev/sdh", "Ebs": { "VolumeSize
": 8, "DeleteOnTermination": true } }]' \
--placement
"AvailabilityZone=${available_zone_2},GroupNa
me=${placement_group_name}" \
--instance-initiated-shutdown-behavior
"terminate" \
--query 'Instances[*].InstanceId' --
output text
```

Wait for Running State:

```
aws ec2 wait instance-running --instance-ids
${instance_ids} --region ${region}
```

Wait for Status Checks to Pass:

```
aws ec2 wait instance-status-ok --instance-
ids ${instance_ids} --region ${region}
```

Load Balancers

Create Load Balancer:

```
load_balancer_arn=$(aws elbv2 create-load-
balancer \
--name my-load-balancer \
--subnets ${public_subnet_id_1}
${public_subnet_id_2} \
--security-groups
${app_security_group_id} \
--query
'LoadBalancers[0].LoadBalancerArn' --output
text --region ${region})
```

Create Target Group:

```
target_group_arn=$(aws elbv2 create-target-
group \
--name my-target-group \
--protocol HTTP \
--port 80 \
--vpc-id ${vpc_id} \
--query 'TargetGroups[0].TargetGroupArn'
--output text --region ${region})
```

AutoScaling Group

Create Launch Template:

```
launch_template_id=$(aws ec2 create-launch-
template \
--launch-template-name
${launch_template_name} \
--version-description "v1" \
--launch-template-data '{
"ImageId": "${image_id}",
"InstanceType": "t3.micro",
"KeyName": "${key_pair_name}",
"SecurityGroupIds":
["${app_security_group_id}"],
"IamInstanceProfile": {"Name":
"${instance_profile_name}"},
"UserData": "${(base64 -w 0
./userDataCentOsComplex.sh) }",
"BlockDeviceMappings": [{
"DeviceName": "/dev/sdh",
"Ebs": {
"VolumeSize": 8,
"DeleteOnTermination": true
}
}]}
}' --query
'LaunchTemplate.LaunchTemplateId' --output
text --region ${region})
```

Create Auto Scaling Group:

```
aws autoscaling create-auto-scaling-group \
--auto-scaling-group-name
${auto_scaling_group_name} \
--launch-template
"LaunchTemplateId=${launch_template_id},Versi
on=1" \
--min-size ${min_size} \
--max-size ${max_size} \
```

```
--desired-capacity ${desired_capacity} \
--vpc-zone-identifier "${subnet_ids}" \
--region ${region}
```

Scale Up Policy:

```
scale_up_policy_arn=$(aws autoscaling put-
scaling-policy \
  --auto-scaling-group-name
${auto_scaling_group_name} \
  --policy-name ScaleUpPolicy \
  --scaling-adjustment 1 \
  --adjustment-type ChangeInCapacity \
  --region ${region} \
  --query 'PolicyARN' --output text)
```

Scale Down Policy:

```
scale_down_policy_arn=$(aws autoscaling put-
scaling-policy \
  --auto-scaling-group-name
${auto_scaling_group_name} \
  --policy-name ScaleDownPolicy \
  --scaling-adjustment -1 \
  --adjustment-type ChangeInCapacity \
  --region ${region} \
  --query 'PolicyARN' --output text)
```

High CPU Utilization Alarm:

```
aws cloudwatch put-metric-alarm \
  --alarm-name HighCPUUtilization \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --statistic Average \
  --period 300 \
  --threshold 80 \
  --comparison-operator
GreaterThanOrEqualToThreshold \
  --dimensions
Name=AutoScalingGroupName,Value=${auto_scalin
g_group_name} \
  --evaluation-periods 2 \
  --alarm-actions ${scale_up_policy_arn} \
  --region ${region}
```

Low CPU Utilization Alarm:

```
aws cloudwatch put-metric-alarm \
  --alarm-name LowCPUUtilization \
  --metric-name CPUUtilization \
  --namespace AWS/EC2 \
  --statistic Average \
  --period 300 \
  --threshold 20 \
  --comparison-operator
LessThanOrEqualToThreshold \
  --dimensions
Name=AutoScalingGroupName,Value=${auto_scalin
g_group_name} \
  --evaluation-periods 2 \
  --alarm-actions ${scale_down_policy_arn}
\
  --region ${region}
```

Kubernetes

Pod

```
apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  labels:
    app: my-app
spec:
  containers:
    - name: my-container
      image: nginx:1.14.2
      ports:
        - containerPort: 80
      resources:
        requests:
          cpu: "100m"
          memory: "128Mi"
        limits:
```

```
cpu: "500m"
memory: "256Mi"
readinessProbe:
  httpGet:
    path: /
    port: 80
    initialDelaySeconds: 5
    periodSeconds: 10
livenessProbe:
  httpGet:
    path: /healthz
    port: 80
    initialDelaySeconds: 15
    periodSeconds: 20
restartPolicy: Always
nodeSelector:
  disktype: ssd
tolerations:
- key: "key"
  operator: "Equal"
  value: "value"
  effect: "NoSchedule"
```

ReplicaSet

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: my-replicaset
  labels:
    app: my-app
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-deployment
  labels:
    app: my-app
spec:
  replicas: 3
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 1
      maxUnavailable: 0
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: nginx:1.14.2
          ports:
            - containerPort: 80
          imagePullPolicy: IfNotPresent
```

Service

```
apiVersion: v1
kind: Service
```

```

metadata:
  name: my-service
  annotations:
    service.beta.kubernetes.io/aws-load-
balancer-type: "nlb"
spec:
  selector:
    app: my-app
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
      name: http
      type: LoadBalancer
      sessionAffinity: ClientIP
      externalTrafficPolicy: Local

```

ConfigMap

```

apiVersion: v1
kind: ConfigMap
metadata:
  name: my-config
data:
  config.property: "some-value"
  another.property: |
    line1
    line2
binaryData:
  binaryFile: <base64 encoded>

```

Secret

```

apiVersion: v1
kind: Secret
metadata:
  name: my-secret
type: Opaque
data:
  username: dXNlcm5hbWU= # base64 encoded
  password: cGFzc3dvcmQ= # base64 encoded
stringData:
  config.yaml: |
    apiUrl: "https://myapi.com"
    token: "my-token"

```

PersistentVolume

```

apiVersion: v1
kind: PersistentVolume
metadata:
  name: pv0001
spec:
  capacity:
    storage: 5Gi
  accessModes:
    - ReadWriteOnce
  persistentVolumeReclaimPolicy: Retain
  storageClassName: standard
  nfs:
    server: nfs-server.example.com
    path: "/exports"

```

PersistentVolumeClaim

```

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: my-claim
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 5Gi
  storageClassName: standard
  volumeMode: Filesystem
  volumeName: pv0001 # optional, binds to
a specific PV

```

Namespace

```

apiVersion: v1
kind: Namespace
metadata:

```

```

name: my-namespace
labels:
  environment: development

```

DaemonSet

```

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: my-daemonset
  namespace: kube-system
spec:
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      nodeSelector:
        node-
role.kubernetes.io/master: ""
      tolerations:
        - key: "node-
role.kubernetes.io/master"
          effect: NoSchedule
      containers:
        - name: my-container
          image: nginx:1.14.2

```

Job

```

apiVersion: batch/v1
kind: Job
metadata:
  name: my-job
spec:
  completions: 5
  parallelism: 2
  backoffLimit: 6
  template:
    spec:
      containers:
        - name: my-job-container
          image: busybox
          command: ["/bin/sh", "-c",
"echo Hello, Kubernetes! && sleep 30"]
          restartPolicy: OnFailure

```

CronJob

```

apiVersion: batch/v1beta1
kind: CronJob
metadata:
  name: my-cronjob
spec:
  schedule: "*/1 * * * *"
  concurrencyPolicy: Forbid
  failedJobsHistoryLimit: 1
  successfulJobsHistoryLimit: 3
  suspend: false
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: my-cronjob-
container
              image: busybox
              command:
                - /bin/sh
                - -c
                - date; echo Hello
from the Kubernetes cron job
              restartPolicy: OnFailure

```

StatefulSet

```

apiVersion: apps/v1
kind: StatefulSet
metadata:
  name: my-statefulset

```



```
spec:
  serviceName: "my-service"
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      containers:
        - name: my-container
          image: nginx:1.14.2
          volumeMounts:
            - name: www
              mountPath: /usr/share/nginx/html
          volumeClaimTemplates:
            - metadata:
                name: www
              spec:
                accessModes: ["ReadWriteOnce"]
                resources:
                  requests:
                    storage: 1Gi
```

Ingress

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: my-ingress
  annotations:
    kubernetes.io/ingress.class: "nginx"
    nginx.ingress.kubernetes.io/rewrite-target: /$2
spec:
  rules:
    - host: example.com
      http:
        paths:
          - path: /path/(.*)
            pathType: Prefix
            backend:
              service:
                name: my-service
                port:
                  number: 80
```

HorizontalPodAutoscaler

```
apiVersion: autoscaling/v2beta2
kind: HorizontalPodAutoscaler
metadata:
  name: my-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-deployment
  minReplicas: 1
  maxReplicas: 10
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 50
    - type: Pods
      pods:
        metric:
          name: packets-per-second
        target:
          type: AverageValue
          averageValue: 1k
```

VerticalPodAutoscaler

Requires additional installation of Vertical Pod Autoscaler:

```
apiVersion: autoscaling.k8s.io/v1
kind: VerticalPodAutoscaler
metadata:
  name: my-vpa
spec:
  targetRef:
    apiVersion: "apps/v1"
    kind: Deployment
    name: my-deployment
  updatePolicy:
    updateMode: "Auto"
  resourcePolicy:
    containerPolicies:
      - containerName: '*'
        minAllowed:
          cpu: 250m
          memory: 64Mi
        maxAllowed:
          cpu: 2
          memory: 4Gi
```

NetworkPolicy

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
spec:
  podSelector:
    matchLabels:
      role: db
  policyTypes:
    - Ingress
    - Egress
  ingress:
    - from:
        - podSelector:
            matchLabels:
              role: frontend
      ports:
        - protocol: TCP
          port: 6379
  egress:
    - to:
        - ipBlock:
            cidr: 10.0.0.0/24
            except:
              - 10.0.0.0/28
      ports:
        - protocol: TCP
          port: 5978
```

ServiceAccount

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-service-account
  namespace: my-namespace
secrets:
  - name: my-secret
imagePullSecrets:
  - name: regcred
```

Endpoints

```
apiVersion: v1
kind: Endpoints
metadata:
  name: my-endpoints
subsets:
  - addresses:
      - ip: 192.168.1.1
        nodeName: worker1
    ports:
      - port: 80
        name: http
```

ResourceQuota

```
apiVersion: v1
```

```
kind: ResourceQuota
metadata:
  name: my-quota
spec:
  hard:
    pods: "10"
    requests.cpu: "4"
    requests.memory: 6Gi
    limits.cpu: "10"
    limits.memory: 10Gi
    configmaps: "10"
    secrets: "10"
    services: "5"
    services.loadbalancers: "1"
```

LimitRange

```
apiVersion: v1
kind: LimitRange
metadata:
  name: my-limitrange
spec:
  limits:
    - type: Pod
      max:
        cpu: "2"
        memory: 1Gi
      min:
        cpu: 200m
        memory: 6Mi
    - type: Container
      default:
        cpu: 500m
        memory: 512Mi
      defaultRequest:
        cpu: 100m
        memory: 128Mi
```

Roles and RoleBindings

```
# Role
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: my-namespace
  name: pod-reader
rules:
- apiGroups: [""]
  resources: ["pods"]
  verbs: ["get", "watch", "list"]

# RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: my-namespace
subjects:
- kind: User
  name: my-user
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

ClusterRoles and ClusterRoleBindings

```
# ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]

# ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
```

```
metadata:
  name: read-secrets-global
subjects:
- kind: User
  name: my-user
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

CustomResourceDefinition

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  name: crontabs.stable.example.com
spec:
  group: stable.example.com
  versions:
    - name: v1
      served: true
      storage: true
      schema:
        openAPIV3Schema:
          type: object
          properties:
            spec:
              type: object
              properties:
                cronSpec:
                  type: string
                image:
                  type: string
                replicas:
                  type: integer

          subresources:
            status: {}
  scope: Namespaced
  names:
    plural: crontabs
    singular: crontab
    kind: CronTab
    shortNames:
      - ct
```

StorageClass

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
  name: standard
provisioner: kubernetes.io/aws-ebs
parameters:
  type: gp2
  zones: us-west-2a, us-west-2b
reclaimPolicy: Retain
allowVolumeExpansion: true
mountOptions:
- debug
volumeBindingMode: WaitForFirstConsumer
```

PodDisruptionBudget

```
apiVersion: policy/v1beta1
kind: PodDisruptionBudget
metadata:
  name: my-pdb
spec:
  minAvailable: 2
  selector:
    matchLabels:
      app: my-app
```

Use `kubectl apply -f <filename>.yaml` to deploy them to your Kubernetes cluster.

- **Pod:** `kubectl run my-pod --image=nginx --port=80 --restart=Never --dry-run=client -o yaml > pod.yaml`
- **Service:** `kubectl expose deployment my-deployment --type=LoadBalancer --`

port=80 --target-port=8080 --name=my-service

- **ConfigMap:** `kubectl create configmap my-config --from-file=config.properties`
- **Secret:** `kubectl create secret generic my-secret --from-literal=username=user --from-literal=password=pass`
- **Namespace:** `kubectl create namespace my-namespace`
- **Deployment:** `kubectl create deployment my-deployment --image=nginx --replicas=3 --dry-run=client -o yaml > deploy.yaml`
- **HorizontalPodAutoscaler:** `kubectl autoscale deployment my-deployment --min=1 --max=10 --cpu-percent=50`

Important Kubernetes Commands:

- **kubectl get pods:** Lists all pods in the current namespace.
 - Syntax: `kubectl get pods [-n <namespace>] [-o <output_format>]`
 - Example: `kubectl get pods -n default -o wide`
- **kubectl get nodes:** Shows all nodes in the cluster.
 - Syntax: `kubectl get nodes [-o <output_format>]`
 - Example: `kubectl get nodes -o json`
- **kubectl get services:** Lists all services in the current namespace.
 - Syntax: `kubectl get services [-n <namespace>] [-o <output_format>]`
 - Example: `kubectl get services -n kube-system`
- **kubectl describe pod :** Provides detailed information about a specific pod.
 - Syntax: `kubectl describe pod <pod-name> [-n <namespace>]`
 - Example: `kubectl describe pod my-pod -n my-namespace`
- **kubectl logs :** Retrieves logs from a container in a pod.
 - Syntax: `kubectl logs <pod-name> [-c <container-name>] [--previous] [-f]`
 - Example: `kubectl logs my-pod -c my-container --previous`
- **kubectl exec -it -- /bin/bash:** Opens an interactive shell into a container within a pod.
 - Syntax: `kubectl exec -it <pod-name> [-c <container-name>] -- <command>`
 - Example: `kubectl exec -it my-pod -c main-container -- /bin/bash`
- **kubectl apply -f .yaml:** Applies a configuration to a resource by filename or stdin.
 - Syntax: `kubectl apply -f <filename>.yaml [-n <namespace>]`
 - Example: `kubectl apply -f deployment.yaml`
- **kubectl delete pod :** Deletes a pod.
 - Syntax: `kubectl delete pod <pod-name> [-n <namespace>]`

◦ Example: `kubectl delete pod my-pod`

- **kubectl scale --replicas=3 deployment/:** Scales the number of pods for a deployment.
 - Syntax: `kubectl scale --replicas=<number> deployment/<deployment-name> [-n <namespace>]`
 - Example: `kubectl scale --replicas=3 deployment/my-app`
- **kubectl rollout status deployment/:** Checks the status of a deployment rollout.
 - Syntax: `kubectl rollout status deployment/<deployment-name> [-n <namespace>]`
 - Example: `kubectl rollout status deployment/my-deployment`
- **kubectl rollout undo deployment/:** Rolls back to the previous deployment revision.
 - Syntax: `kubectl rollout undo deployment/<deployment-name> [-n <namespace>]`
 - Example: `kubectl rollout undo deployment/my-deployment`
- **kubectl create deployment --image=:** Creates a new deployment with the specified image.
 - Syntax: `kubectl create deployment <deployment-name> --image=<image-name> [-n <namespace>]`
 - Example: `kubectl create deployment nginx --image=nginx`
- **kubectl get deployments:** Lists all deployments in the current namespace.
 - Syntax: `kubectl get deployments [-n <namespace>] [-o <output_format>]`
 - Example: `kubectl get deployments -o yaml`
- **kubectl port-forward ::** Forwards traffic from a local port to a port on the pod.
 - Syntax: `kubectl port-forward <pod-name> <local-port>:<pod-port> [-n <namespace>]`
 - Example: `kubectl port-forward my-pod 8080:80`
- **kubectl label nodes =:** Adds or updates a label on a node.
 - Syntax: `kubectl label nodes <node-name> <key>=<value> [--overwrite]`
 - Example: `kubectl label nodes worker1 disktype=ssd`
- **kubectl taint nodes =::** Adds a taint on a node, which can repel pods unless they tolerate the taint.
 - Syntax: `kubectl taint nodes <node-name> <key>=<value>:<effect> [--overwrite]`
 - Example: `kubectl taint nodes worker2 apptype=legacy:NoSchedule`
- **kubectl get events:** Shows all events in the current namespace.

- o Syntax: `kubectl get events [-n <namespace>] [-o <output_format>]`
- o Example: `kubectl get events -n my-namespace --sort-by='.lastTimestamp'`
- **kubectl config view:** Displays current kubeconfig settings.
 - o Syntax: `kubectl config view [--minify] [--flatten]`
 - o Example: `kubectl config view --minify`
- **kubectl cluster-info:** Displays endpoint information about the master and services in the cluster.
 - o Syntax: `kubectl cluster-info`
 - o Example: `kubectl cluster-info`

How do you mount a ConfigMap as an environment variable or volume in a Pod?

For environment variables:

```
env:
- name: SPECIAL_LEVEL_KEY
  valueFrom:
    configMapKeyRef:
      name: special-config
      key: special.how
```

For volumes:

```
volumes:
- name: config-volume
  configMap:
    name: special-config
volumeMounts:
- mountPath: /etc/config
  name: config-volume
```

How would you securely use Secrets in a Pod?

Mount Secrets as files in a volume for minimal exposure or use them as environment variables. For file mounts:

```
volumes:
- name: secret-volume
  secret:
    secretName: mysecret
volumeMounts:
- name: secret-volume
  readOnly: true
  mountPath: "/etc/secrets"
```

For environment variables:

```
env:
- name: SECRET_USERNAME
  valueFrom:
    secretKeyRef:
      name: mysecret
      key: username
```

How can you schedule Pods on specific nodes?

Use nodeSelector in the pod spec to match node labels:

```
nodeSelector:
  disktype: ssd
```

Or use nodeAffinity for more complex rules. Taints and tolerations can also be used to repel or attract pods to nodes.

Describe how you would configure an Ingress to route traffic to different services.

Define rules in the Ingress resource:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
spec:
  rules:
  - host: example.com
    http:
```

```
paths:
- path: /api
  pathType: Prefix
  backend:
    service:
      name: api-service
      port:
        number: 80
- path: /
  pathType: Prefix
  backend:
    service:
      name: web-service
      port:
        number: 80
```

How do you implement a NetworkPolicy to restrict pod communication?

Define a NetworkPolicy with selectors and rules for ingress/egress:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-from-namespace
spec:
  podSelector:
    matchLabels:
      role: frontend
  policyTypes:
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          environment: production
    ports:
    - protocol: TCP
      port: 80
```

How can you bind a ServiceAccount to a Role or ClusterRole?

Use RoleBindings or ClusterRoleBindings:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
- kind: ServiceAccount
  name: my-service-account
  namespace: default
roleRef:
  kind: Role
  name: pod-reader
  apiGroup: rbac.authorization.k8s.io
```

How would you set up ResourceQuotas to prevent a namespace from using too many resources?

Define a ResourceQuota in the namespace:

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: compute-resources
spec:
  hard:
    pods: "4"
    requests.cpu: "1"
    requests.memory: 1Gi
    limits.cpu: "2"
    limits.memory: 2Gi
```

Update kubeconfig for EKS:

```
aws eks update-kubeconfig --name my-cluster --region ap-south-1
```

The script sets up port forwarding for Prometheus to access it locally.

```
kubectl port-forward $(kubectl get pods -l
app=prometheus -o
jsonpath='{.items[0].metadata.name}')
9090:9090 > /dev/null 2>&1 &
```

mysql-secret.yaml

```
apiVersion: v1
kind: Secret
metadata:
  name: mysql-secret
type: Opaque
data:
  MYSQL_ROOT_PASSWORD: cGFzc3dvcmQ= #
base64 encoded value of "password"
```

backend-service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: backend-service
spec:
  selector:
    app: backend
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
  type: LoadBalancer
```

backenddeployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: backend
  template:
    metadata:
      labels:
        app: backend
    spec:
      initContainers:
        - name: init-mysql
          image: mysql:8.0
          env:
            - name: MYSQL_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mysql-
secret
                  key:
MYSQL_ROOT_PASSWORD
            - name: DB_HOST
              value: "${db_host}"
            - name: DB_PORT
              value: "${db_port}"
          volumeMounts:
            - name: init-sql
              mountPath: /docker-
entrypoint-initdb.d
              command: [ "sh", "-c", "mysql
-h ${db_host} -P ${db_port} -u admin -
p${MYSQL_ROOT_PASSWORD} < /docker-entrypoint-
initdb.d/init.sql" ]
            containers:
              - name: backend
                image:
jeevan2001/backend:latest
                env:
                  - name: DB_HOST
                    value: "${db_host}"
                  - name: DB_PORT
                    value: "${db_port}"
                  - name: MYSQL_ROOT_PASSWORD
                    valueFrom:
```

```
secretKeyRef:
  name: mysql-
```

```
secret
```

```
key:
```

```
MYSQL_ROOT_PASSWORD
```

```
ports:
```

```
- containerPort: 3000
```

```
volumes:
```

```
- name: init-sql
```

```
configMap:
```

```
name: init-sql-config
```

Get the Backend LoadBalancer DNS

```
export BACKEND_LOADBALANCER_DNS=$(kubectl get
service backend-service -o
jsonpath='{.status.loadBalancer.ingress[0].ho
stname}')
```

frontendservice.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: frontend-service
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
  type: LoadBalancer
```

frontenddeployment.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend
spec:
  replicas: 2
  selector:
    matchLabels:
      app: frontend
  template:
    metadata:
      labels:
        app: frontend
    spec:
      containers:
        - name: frontend
          image:
jeevan2001/frontend:latest
          ports:
            - containerPort: 80
          imagePullPolicy: Always
```

hpa-backend.yaml

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: hpa-backend
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: backend-deployment
  minReplicas: 1
  maxReplicas: 10
  targetCPUUtilizationPercentage: 50
```

cluster-autoscaler.yaml

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: cluster-autoscaler
  namespace: kube-system
  labels:
    app: cluster-autoscaler
spec:
  replicas: 1
  selector:
```

```

    matchLabels:
      app: cluster-autoscaler
  template:
    metadata:
      labels:
        app: cluster-autoscaler
    spec:
      containers:
        - name: cluster-autoscaler
          image:
            k8s.gcr.io/autoscaling/cluster-autoscaler:v1.20.0
          command:
            - ./cluster-autoscaler
            - --v=4
            - --stderrthreshold=info
            - --cloud-provider=aws
            - --skip-nodes-with-local-storage=false
            - --expander=least-waste
            - --nodes=1:10:my-node-group
          env:
            - name: AWS_REGION
              value: ap-south-1
          resources:
            limits:
              cpu: 100m
              memory: 300Mi
            requests:
              cpu: 100m
              memory: 300Mi
          volumeMounts:
            - name: ssl-certs
              mountPath:
                /etc/ssl/certs/ca-certificates.crt
              readOnly: true
          volumes:
            - name: ssl-certs
              hostPath:
                path: /etc/ssl/certs/ca-certificates.crt

```

cluster-autoscaler-policy.json

```

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

```

Terraform

AWS Provider

```

provider "aws" {
  region = "ap-south-1"
}

```

Kubernetes Provider

```

provider "kubernetes" {

```

```

  host =
    aws_eks_cluster.my_cluster.endpoint
  cluster_ca_certificate =
    base64decode(aws_eks_cluster.my_cluster.certificate_authority[0].data)
  token =
    data.aws_eks_cluster_auth.my_cluster.token
}

```

Data Sources

aws_eks_cluster_auth

```

data "aws_eks_cluster_auth" "my_cluster" {
  name = aws_eks_cluster.my_cluster.name
}

```

aws_availability_zones

```

data "aws_availability_zones" "available" {}

```

Network Resources

aws_vpc

```

resource "aws_vpc" "eks_vpc" {
  cidr_block = "10.0.0.0/16"
}

```

aws_subnet

```

resource "aws_subnet" "eks_public_subnet" {
  count = 3
  vpc_id =
    aws_vpc.eks_vpc.id
  cidr_block =
    cidrsubnet(aws_vpc.eks_vpc.cidr_block, 8, count.index)
  availability_zone =
    element(data.aws_availability_zones.available.names, count.index)
  map_public_ip_on_launch = true
}

```

aws_subnet (Private)

```

resource "aws_subnet" "eks_private_subnet" {
  count = 3
  vpc_id =
    aws_vpc.eks_vpc.id
  cidr_block =
    cidrsubnet(aws_vpc.eks_vpc.cidr_block, 8, count.index + 3)
  availability_zone =
    element(data.aws_availability_zones.available.names, count.index)
  map_public_ip_on_launch = false
}

```

aws_internet_gateway

```

resource "aws_internet_gateway" "eks_igw" {
  vpc_id = aws_vpc.eks_vpc.id
}

```

aws_route_table

```

resource "aws_route_table" "eks_public_route_table" {
  vpc_id = aws_vpc.eks_vpc.id

  route {
    cidr_block = "0.0.0.0/0"
    gateway_id =
      aws_internet_gateway.eks_igw.id
  }
}

```

aws_route_table_association

```

resource "aws_route_table_association" "eks_public_route_table_association" {
  count = 3
  subnet_id =
    element(aws_subnet.eks_public_subnet[*].id, count.index)
  route_table_id =
    aws_route_table.eks_public_route_table.id
}

```

aws_nat_gateway

```

resource "aws_nat_gateway" "eks_nat_gateway" {
  count = 3

```

```

        allocation_id =
aws_eip.nat_eip[count.index].id
        subnet_id      =
element(aws_subnet.eks_public_subnet[*].id,
count.index)
}

```

aws_eip

```

resource "aws_eip" "nat_eip" {
    count    = 3
    domain  = "vpc"
}

```

aws_route_table (Private)

```

resource "aws_route_table"
"eks_private_route_table" {
    vpc_id = aws_vpc.eks_vpc.id

    route {
        cidr_block      = "0.0.0.0/0"
        nat_gateway_id =
element(aws_nat_gateway.eks_nat_gateway[*].id
, 0)
    }
}

```

aws_route_table_association (Private)

```

resource "aws_route_table_association"
"eks_private_route_table_association" {
    count    = 3
    subnet_id =
element(aws_subnet.eks_private_subnet[*].id,
count.index)
    route_table_id =
aws_route_table.eks_private_route_table.id
}

```

Security

aws_security_group

```

resource "aws_security_group"
"eks_security_group" {
    vpc_id = aws_vpc.eks_vpc.id

    egress {
        from_port    = 0
        to_port      = 0
        protocol     = "-1"
        cidr_blocks = ["0.0.0.0/0"]
    }

    ingress {
        from_port    = 3306
        to_port      = 3306
        protocol     = "tcp"
        cidr_blocks = ["10.0.0.0/16"]
    }
}

```

Database

aws_db_instance

```

resource "aws_db_instance" "mydb" {
    allocated_storage    = 20
    storage_type         = "gp2"
    engine               = "mysql"
    engine_version       = "8.0"
    instance_class       = "db.t3.micro"
    db_name              = "mydatabase"
    username             = "admin"
    password             = "password"
    db_subnet_group_name =
aws_db_subnet_group.mydb_subnet_group.name
    vpc_security_group_ids =
[aws_security_group.rds_security_group.id]
    skip_final_snapshot = true
}

```

aws_db_subnet_group

```

resource "aws_db_subnet_group"
"mydb_subnet_group" {
    name = "mydb-subnet-group"
}

```

```

        subnet_ids =
aws_subnet.eks_private_subnet[*].id
}

```

IAM

aws_iam_role

```

resource "aws_iam_role" "eks_cluster_role" {
    name = "eks-cluster-role"

    assume_role_policy = jsonencode({
        Version = "2012-10-17"
        Statement = [
            {
                Effect = "Allow"
                Principal = {
                    Service =
"eks.amazonaws.com"
                }
                Action = "sts:AssumeRole"
            },
        ]
    })
}

```

aws_iam_role_policy_attachment

```

resource "aws_iam_role_policy_attachment"
"eks_cluster_role_attachment" {
    role      =
aws_iam_role.eks_cluster_role.name
    policy_arn =
"arn:aws:iam::aws:policy/AmazonEKSClusterPolicy"
}

```

EKS

aws_eks_cluster

```

resource "aws_eks_cluster" "my_cluster" {
    name     = "my-cluster"
    role_arn =
aws_iam_role.eks_cluster_role.arn

    vpc_config {
        subnet_ids =
aws_subnet.eks_public_subnet[*].id
        security_group_ids =
[aws_security_group.eks_security_group.id]
    }
}

```

aws_eks_node_group

```

resource "aws_eks_node_group" "my_node_group"
{
    cluster_name =
aws_eks_cluster.my_cluster.name
    node_group_name = "my-node-group"
    node_role_arn =
aws_iam_role.eks_node_role.arn
    subnet_ids =
aws_subnet.eks_private_subnet[*].id

    scaling_config {
        desired_size = 5
        max_size     = 7
        min_size     = 3
    }

    instance_types = ["t3.small"]

    remote_access {
        ec2_ssh_key = "my-key"
    }

    tags = {
        Name = "eks-node-group"
    }
}

```

Local Resources and Data

local_file

```
resource "local_file"
"website_content_configmap" {
  content =
data.template_file.website_content_configmap.
rendered
  filename = "${path.module}/website-
content-configmap.yaml"
}
```

data.template_file

```
data "template_file"
"website_content_configmap" {
  template = file("${path.module}/website-
content-configmap.tpl.yaml")
  vars = {
    db_host =
aws_db_instance.mydb.endpoint
  }
}
```

kubernetes_config_map

```
resource "kubernetes_config_map"
"init_sql_config" {
  metadata {
    name = "init-sql-config"
  }
  data = {
    "init.sql" =
file("${path.module}/init.sql")
  }
}
```

VPC

```
resource "aws_vpc" "eks_vpc" {
  cidr_block = "10.0.0.0/16"
}

resource "aws_subnet" "eks_public_subnet" {
  count                = 3
  vpc_id              =
aws_vpc.eks_vpc.id
  cidr_block          =
cidrsubnet(aws_vpc.eks_vpc.cidr_block, 8,
count.index)
  availability_zone    =
element(data.aws_availability_zones.available
.names, count.index)
  map_public_ip_on_launch = true
}

resource "aws_subnet" "eks_private_subnet" {
  count                = 3
  vpc_id              =
aws_vpc.eks_vpc.id
  cidr_block          =
cidrsubnet(aws_vpc.eks_vpc.cidr_block, 8,
count.index + 3)
  availability_zone    =
element(data.aws_availability_zones.available
.names, count.index)
}
```

Security Groups

AWS Security Group:

```
resource "aws_security_group"
"eks_security_group" {
  vpc_id = aws_vpc.eks_vpc.id

  ingress {
    from_port = 80
    to_port   = 80
    protocol  = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  egress {
    from_port = 0
    to_port   = 0
    protocol  = "-1"
  }
}
```

```
cidr_blocks = ["0.0.0.0/0"]
}
```

Kubernetes Network Policy:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-web
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: web
  ingress:
    - from:
      - podSelector:
          matchLabels:
            app: frontend
      ports:
        - protocol: TCP
          port: 80
```

EKS Cluster

EKS Cluster:

```
resource "aws_eks_cluster" "my_cluster" {
  name     = "my-cluster"
  role_arn =
aws_iam_role.eks_cluster_role.arn

  vpc_config {
    subnet_ids =
[aws_subnet.eks_public_subnet.*.id]
  }
}
```

IAM Role for EKS Cluster:

```
resource "aws_iam_role" "eks_cluster_role" {
  name = "eks-cluster-role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Principal = {
          Service =
"eks.amazonaws.com"
        }
        Action = "sts:AssumeRole"
      },
    ]
  })
}

resource "aws_iam_role_policy_attachment"
"eks_cluster_policy" {
  role =
aws_iam_role.eks_cluster_role.name
  policy_arn =
"arn:aws:iam::aws:policy/AmazonEKSClusterPoli
cy"
}
```

AWS & Kubernetes Integration with Terraform

```
provider "aws" {
  region = "ap-south-1"
}

provider "kubernetes" {
  host =
aws_eks_cluster.my_cluster.endpoint
  cluster_ca_certificate =
base64decode(aws_eks_cluster.my_cluster.certi
ficate_authority[0].data)
  token =
data.aws_eks_cluster_auth.my_cluster.token
}
```



```
resource "aws_eks_cluster" "my_cluster" {
  name      = "my-cluster"
  role_arn =
aws_iam_role.eks_cluster_role.arn

  vpc_config {
    subnet_ids =
[aws_subnet.eks_public_subnet.*.id]
  }
}
```

Code Example:

ConfigMap:

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: db-config
data:
  DB_HOST: mydb.example.com
  DB_PORT: "3306"
```

Secret:

```
apiVersion: v1
kind: Secret
metadata:
  name: db-secret
type: Opaque
data:
  DB_PASSWORD: cGFzc3dvcmQ= # base64 encoded
password
```

Using ConfigMap and Secret in a Pod:

```
apiVersion: v1
kind: Pod
metadata:
  name: my-app
spec:
  containers:
  - name: my-app-container
    image: my-app-image
    env:
    - name: DB_HOST
      valueFrom:
        configMapKeyRef:
          name: db-config
          key: DB_HOST
    - name: DB_PORT
      valueFrom:
        configMapKeyRef:
          name: db-config
          key: DB_PORT
    - name: DB_PASSWORD
      valueFrom:
        secretKeyRef:
          name: db-secret
          key: DB_PASSWORD
```

Autoscaling using Kubernetes and AWS

AWS Auto Scaling Group:

```
resource "aws_autoscaling_group" "example" {
  launch_configuration =
aws_launch_configuration.example.id
  min_size             = 1
  max_size             = 5
  desired_capacity     = 2
  vpc_zone_identifier =
[aws_subnet.eks_public_subnet.*.id]
}
```

Kubernetes HPA:

```
apiVersion: autoscaling/v1
kind: HorizontalPodAutoscaler
metadata:
  name: my-app-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: my-app
  minReplicas: 1
```

maxReplicas: 10

targetCPUUtilizationPercentage: 50

- **Pods:** The smallest and simplest Kubernetes object. A Pod represents a single instance of a running process in your cluster.
- **ReplicaSets:** Ensures a specified number of pod replicas are running at any given time.
- **Deployments:** Provides declarative updates for Pods and ReplicaSets.
- **Services:** An abstraction which defines a logical set of Pods and a policy by which to access them - like load-balancers.
- **ConfigMaps:** Used to store configuration data in key-value pairs which can be consumed by pods.
- **Secrets:** Manages sensitive information, like passwords, OAuth tokens, and ssh keys, which can be referenced in pod definitions.
- **PersistentVolumes (PV):** A piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes.
- **PersistentVolumeClaims (PVC):** Requests storage resources defined by a PersistentVolume.
- **Namespaces:** Provides a scope for names. Resources like Pods, Services, and Deployments can be isolated within namespaces.
- **Nodes:** A worker machine in Kubernetes, either virtual or physical, where containers will be launched by Kubernetes.
- **DaemonSets:** Ensures that all (or some) Nodes run a copy of a Pod. As nodes are added to the cluster, Pods are added to them. As nodes are removed from the cluster, those Pods are garbage collected.
- **Jobs:** Creates one or more Pods and ensures that a specified number of them successfully terminate. Good for batch processes.
- **CronJobs:** Manages time-based Jobs, similar to cron in Unix-like systems.
- **StatefulSets:** Manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods.
- **Ingress:** Manages external access to the services in a cluster, typically HTTP.
- **HorizontalPodAutoscaler:** Scales a Deployment, ReplicaSet, or ReplicationController based on observed CPU utilization or other select metrics.
- **VerticalPodAutoscaler:** Automatically adjusts the compute resources of pods based on usage.
- **NetworkPolicies:** Specifies how groups of pods are allowed to communicate with each other and other network endpoints.

- **ServiceAccounts:** Provides an identity for processes that run in a Pod, which can be used for authenticating to the API server.
- **Endpoints:** Exposes the IP addresses of a service's backing pods.
- **ResourceQuotas:** Provides constraints that limit aggregate resource consumption per namespace.
- **LimitRanges:** Constrains resource allocations (to Pods or Containers) in a namespace.
- **Roles and RoleBindings (for RBAC - Role-Based Access Control):** Define permissions for users or service accounts within a namespace.
- **ClusterRoles and ClusterRoleBindings:** Similar to Roles but cluster-wide, not namespace-specific.
- **CustomResourceDefinitions (CRDs):** Allows users to create new types of resources without adding another API server.
- **StorageClasses:** Describes different classes or profiles of storage in the cluster.
- **PodDisruptionBudgets:** Ensures that a specified number of pods are available even during voluntary disruptions like node drains or upgrades.