# **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==`'${instance\_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\_barista\_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=${available\_zone\_2},GroupName=${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},Version=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\_scaling\_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\_scaling\_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 <file-name>.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.
  + Syntax: kubectl get events [-n <namespace>] [-o <output\_format>]
  + Example: **kubectl get events -n my-namespace --sort-by='.lastTimestamp'**
* **kubectl config view**: Displays current kubeconfig settings.
  + Syntax: kubectl config view [--minify] [--flatten]
  + Example: **kubectl config view --minify**
* **kubectl cluster-info**: Displays endpoint information about the master and services in the cluster.
  + Syntax: kubectl cluster-info
  + 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 speci c 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 con gure an Ingress to route tra c to different services.**

De ne 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 kubecon g 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].hostname}')

**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/AmazonEKSClusterPolicy"

}

**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.certificate\_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.

**-**-------------------------------------------

**Priority Order of Learning Kubernetes Resources (Quickie)**

**--------------------------------------------**

**Priority 1: Must-Know Kubernetes Resources for Interviews**

**--------------------------------------------**

Pod

Deployment

Service

ConfigMap

Secret

PersistentVolume

PersistentVolumeClaim

Namespace

StatefulSet

Ingress

HorizontalPodAutoscaler

**--------------------------------------------**

**Priority 2: Nice-to-Know Resources (Learn if You Have Time)**

**--------------------------------------------**

Replicaset

DaemonSet

Job and CronJob

NetworkPolicy

ServiceAccount

ResourceQuota

LimitRange

**--------------------------------------------**

**Priority 3: Skip for Now (Unless Specialized)**

**--------------------------------------------**

VerticalPodAutoscaler

PodDisruptionBudget

CustomResourceDefinition

StorageClass

Endpoints

Roles

RoleBindings

ClusterRoles

ClusterRoleBindings

**-------------------------------------------**

**Priority 1: Must-Know Kubernetes Resources for Interviews**

**Pod**

**The smallest and simplest Kubernetes object. A Pod represents a single instance of a running process in your cluster.**

apiVersion: v1

kind: Pod

metadata:

name: simple-pod

labels:

app: my-app

spec:

containers:

- name: app-container

image: nginx:latest

ports:

- containerPort: 80

resources:

requests:

cpu: "100m"

memory: "128Mi"

limits:

cpu: "500m"

memory: "256Mi"

**Deployment**

**Provides declarative updates for Pods and ReplicaSets.**

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

resources:

requests:

memory: "256Mi"

cpu: "200m"

limits:

memory: "512Mi"

cpu: "500m"

livenessProbe:

httpGet:

path: /health

port: 80

initialDelaySeconds: 30

periodSeconds: 10

readinessProbe:

httpGet:

path: /ready

port: 80

initialDelaySeconds: 5

periodSeconds: 5

env:

- name: ENVIRONMENT

value: "production"

**Service**

**An abstraction which defines a logical set of Pods and a policy by which to access them - like loadbalancers.**

apiVersion: v1

kind: Service

metadata:

name: my-service

spec:

selector:

app: my-app

ports:

- protocol: TCP

port: 80

targetPort: 8080

name: http

type: LoadBalancer

**ConfigMap**

**Used to store configuration data in key-value pairs which can be consumed by pods.**

apiVersion: v1

kind: ConfigMap

metadata:

name: my-config

data:

app.env: "production"

config.file: |

key1=value1

key2=value2

**Secret**

**Manages sensitive information, like passwords, OAuth tokens, and ssh keys, which can be referenced in pod definitions.**

apiVersion: v1

kind: Secret

metadata:

name: my-secret

type: Opaque

data:

username: YWRtaW4= *# "admin"*

password: UEA1NXcwcmQ= *# "P@55w0rd"*

**PersistentVolume**

**A piece of storage in the cluster that has been provisioned by an administrator or dynamically provisioned using Storage Classes.**

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**

**Requests storage resources defined by a PersistentVolume.**

apiVersion: v1

kind: PersistentVolumeClaim

metadata:

name: my-claim

spec:

accessModes:

- ReadWriteOnce

resources:

requests:

storage: 5Gi

storageClassName: standard

**Namespace**

**Provides a scope for names. Resources like Pods, Services, and Deployments can be isolated within namespaces.**

apiVersion: v1

kind: Namespace

metadata:

  name: my-namespace

  labels:

    environment: production

**StatefulSet**

**Manages the deployment and scaling of a set of Pods, and provides guarantees about the ordering and uniqueness of these Pods.**

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**

**Manages external access to the services in a cluster, typically HTTP.**

apiVersion: networking.k8s.io/v1

kind: Ingress

metadata:

  name: my-ingress

  annotations:

    kubernetes.io/ingress.class: "nginx"

spec:

  rules:

    - host: example.com

      http:

        paths:

          - path: /app

            pathType: Prefix

            backend:

              service:

                name: my-service

                port:

                  number: 80

**HorizontalPodAutoscaler**

**Scales a Deployment, ReplicaSet, or ReplicationController based on observed CPU utilization or other select metrics.**

apiVersion: autoscaling/v2

kind: HorizontalPodAutoscaler

metadata:

  name: my-hpa

spec:

  scaleTargetRef:

    apiVersion: apps/v1

    kind: Deployment

    name: my-deployment

  minReplicas: 2

  maxReplicas: 10

  metrics:

    - type: Resource

      resource:

        name: cpu

        target:

          type: Utilization

          averageUtilization: 70

**Priority 2: Nice-to-Know Resources (Learn if You Have Time)**

**Replicaset**

**Ensures a specified number of pod replicas are running at any given time.**

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

**DaemonSet**

**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.**

apiVersion: apps/v1

kind: DaemonSet

metadata:

  name: my-daemonset

spec:

  selector:

    matchLabels:

      app: my-app

  template:

    metadata:

      labels:

        app: my-app

    spec:

      nodeSelector:

        kubernetes.io/role: worker

      tolerations:

        - key: "node-role.kubernetes.io/control-plane"

          effect: "NoSchedule"

      containers:

        - name: my-container

          image: nginx:1.14.2

**Job**

**Creates one or more Pods and ensures that a specified number of them successfully terminate. Good for batch processes.**

apiVersion: batch/v1

kind: Job

metadata:

  name: my-job

spec:

  completions: 5

  parallelism: 2

  backoffLimit: 4

  template:

    spec:

      containers:

        - name: my-job-container

          image: busybox

          command: ["/bin/sh", "-c", "echo Hello, Kubernetes!"]

      restartPolicy: OnFailure

**CronJob**

**Manages time-based Jobs, similar to cron in Unix-like systems.**

apiVersion: batch/v1

kind: CronJob

metadata:

  name: my-cronjob

spec:

  schedule: "0 \*/1 \* \* \*"  # Every hour

  concurrencyPolicy: Forbid

  jobTemplate:

    spec:

      template:

        spec:

          containers:

            - name: my-cronjob-container

              image: busybox

              command: ["/bin/sh", "-c", "echo Hello"]

          restartPolicy: OnFailure

**NetworkPolicy**

**Specifies how groups of pods are allowed to communicate with each other and other network**

**endpoints.**

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

      ports:

        - protocol: TCP

          port: 3306

**ServiceAccount**

**Provides an identity for processes that run in a Pod, which can be used for authenticating to the API server.**

apiVersion: v1

kind: ServiceAccount

metadata:

  name: my-service-account

  namespace: devops-interview

imagePullSecrets:

  - name: regcred

**ResourceQuota**

**Provides constraints that limit aggregate resource consumption per namespace.**

apiVersion: v1

kind: ResourceQuota

metadata:

  name: my-quota

  namespace: devops-interview

spec:

  hard:

    pods: "10"

    requests.cpu: "4"

    requests.memory: "6Gi"

    limits.cpu: "10"

    limits.memory: "10Gi"

**LimitRange**

**Constrains resource allocations (to Pods or Containers) in a namespace.**

apiVersion: v1

kind: LimitRange

metadata:

  name: my-limitrange

  namespace: devops-interview

spec:

  limits:

    - type: Container

      max:

        cpu: "1"

        memory: "512Mi"

      min:

        cpu: "100m"

        memory: "64Mi"

      default:

        cpu: "500m"

        memory: "512Mi"

      defaultRequest:

        cpu: "200m"

        memory: "256Mi"

**Priority 3: Skip for Now (Unless Specialized)**

**VerticalPodAutoscaler**

**Automatically adjusts the compute resources of pods based on usage.**

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: "128Mi"

        maxAllowed:

          cpu: "2"

          memory: "4Gi"

**PodDisruptionBudget**

**Ensures that a specified number of pods are available even during voluntary disruptions like node drains or upgrades.**

apiVersion: policy/v1

kind: PodDisruptionBudget

metadata:

  name: my-pdb

spec:

  minAvailable: 2

  selector:

    matchLabels:

      app: my-app

**CustomResourceDefinition**

**Allows users to create new types of resources without adding another API server.**

apiVersion: apiextensions.k8s.io/v1

kind: CustomResourceDefinition

metadata:

  name: crontabs.stable.example.com

spec:

  group: stable.example.com

  scope: Namespaced

  names:

    plural: crontabs

    singular: crontab

    kind: CronTab

  versions:

    - name: v1

      served: true

      storage: true

      schema:

        openAPIV3Schema:

          type: object

          properties:

            spec:

              type: object

              properties:

                cronSpec:

                  type: string

                image:

                  type: string

**StorageClass**

**Describes different classes or profiles of storage in the cluster.**

apiVersion: storage.k8s.io/v1

kind: StorageClass

metadata:

  name: standard

provisioner: kubernetes.io/aws-ebs

parameters:

  type: gp2

reclaimPolicy: Retain

allowVolumeExpansion: true

volumeBindingMode: WaitForFirstConsumer

**Endpoints**

**Exposes the IP addresses of a service's backing pods.**

apiVersion: v1

kind: Endpoints

metadata:

  name: my-endpoints

spec:

  subsets:

    - addresses:

        - ip: 192.168.1.1

      ports:

        - port: 80

          name: http

**Roles**

**Define permissions for users or service accounts within a namespace.**

apiVersion: rbac.authorization.k8s.io/v1

kind: Role

metadata:

  name: pod-reader

  namespace: devops-interview

rules:

  - apiGroups: [""]

    resources: ["pods"]

    verbs: ["get", "list", "watch"]

**RoleBindings**

**Define permissions for users or service accounts within a namespace.**

apiVersion: rbac.authorization.k8s.io/v1

kind: RoleBinding

metadata:

  name: read-pods

  namespace: devops-interview

subjects:

  - kind: User

    name: my-user

    apiGroup: rbac.authorization.k8s.io

roleRef:

  kind: Role

  name: pod-reader

  apiGroup: rbac.authorization.k8s.io

**ClusterRoles**

**Similar to Roles but cluster-wide, not namespace-specific.**

apiVersion: rbac.authorization.k8s.io/v1

kind: ClusterRole

metadata:

  name: secret-reader

rules:

  - apiGroups: [""]

    resources: ["secrets"]

    verbs: ["get", "list", "watch"]

**ClusterRoleBindings**

**Similar to Roles but cluster-wide, not**

**namespace-specific**

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