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_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

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**

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:Describe
LaunchTemplateVersions"
                        ],
                        "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.certi
ficate_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/AmazonEKSClusterPoli
cy"
}
```

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
```

Kubernetes

- **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.

```yaml
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.

```yaml
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.

```yaml
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.

```yaml
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.

```yaml
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.

```yaml
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.

```yaml
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.

```yaml
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"
```

Ansible

Program 1: Ansible Basics and Core Workflow

What is Ansible? (Core concepts, agentless, YAML, SSH)

Ansible is an open-source automation tool that uses an agentless architecture (no software installed on managed nodes), relies on SSH for communication, and uses YAML for configuration files like playbooks and inventory.

Ansible Inventory (Static vs. dynamic, host grouping)

The inventory defines the hosts Ansible manages. It can be static (a simple file) or dynamic (script-generated), with hosts organized into groups.

Ansible Ad-Hoc Commands (Basic usage, quick tasks)

Quick, one-line commands to perform tasks on hosts without writing a full playbook (e.g., ansible all -m ping).

Ansible Idempotence (Understanding the concept)

Ansible ensures tasks are idempotent, meaning running them multiple times produces the same result without unintended changes.

Step 1: Set Up a Static Inventory

Create a file named hosts.ini to define managed hosts and groups.

```
# File: hosts.ini
[webservers]
web1.example.com
web2.example.com

[dbservers]
db1.example.com

[all:vars]
ansible_user=admin
```

```
ansible_ssh_private_key_file=~/.ssh/id_rsa
```

Explanation:

- [webservers] and [dbservers] are host groups.
- ansible_user and ansible_ssh_private_key_file are variables for SSH access, showcasing Ansible's agentless nature (uses SSH, no agents needed).

Step 2: Run an Ad-Hoc Command

Use an ad-hoc command to check host uptime.

```
ansible -i hosts.ini all -m command -a "uptime"
```

Output (example):

```
web1.example.com | SUCCESS | rc=0 >>
 14:35:23 up 5 days,  3:12,  1 user,  load
average: 0.10, 0.15, 0.20
web2.example.com | SUCCESS | rc=0 >>
 14:35:23 up 3 days, 10:45,  2 users,  load
average: 0.05, 0.08, 0.12
db1.example.com | SUCCESS | rc=0 >>
 14:35:23 up 7 days,  1:23,  1 user,  load
average: 0.25, 0.30, 0.35
```

Explanation:

- -i hosts.ini: Specifies the inventory file.
- all: Targets all hosts in the inventory.
- -m command: Uses the command module to execute uptime.

This demonstrates quick tasks without a playbook and Ansible's SSH-based communication.

Step 3: Demonstrate Idempotence

Run a simple idempotent ad-hoc command multiple times.

```
ansible -i hosts.ini webservers -m file -a
"path=/tmp/test.txt state=touch"
```

First Run Output:

```
web1.example.com | CHANGED => {"changed": true,
"path": "/tmp/test.txt"}
web2.example.com | CHANGED => {"changed": true,
"path": "/tmp/test.txt"}
```

Second Run Output:

```
web1.example.com | SUCCESS => {"changed": false,
"path": "/tmp/test.txt"}
web2.example.com | SUCCESS => {"changed": false,
"path": "/tmp/test.txt"}
```

Explanation:

- The file module with state=touch creates /tmp/test.txt if it doesn't exist.
- First run: File is created (changed: true).
- Second run: File already exists, so no change (changed: false), proving idempotence.

Step 4: Tie It Together

Ansible's core concepts are shown:

- Agentless: No software installed on web1, web2, or db1; SSH handles everything.
- YAML: Inventory uses a simple, readable format (though not strict YAML here, it's YAML-compatible).
- SSH: Connection relies on SSH keys defined in the inventory.

Key Takeaways for Notes:

- Inventory organizes hosts and groups for targeting.
- Ad-hoc commands are fast, playbook-free ways to manage systems.
- Idempotence ensures consistent results, a core Ansible principle.

Execution Command:

```
# Check connectivity
ansible -i hosts.ini all -m ping
# Run uptime command
ansible -i hosts.ini all -m command -a "uptime"
# Test idempotence
ansible -i hosts.ini webservers -m file -a
"path=/tmp/test.txt state=touch"
```

Program 2: Ansible Playbooks and Task Management

Topics Included:

- Ansible Playbooks (Structure, purpose, basic syntax)
  - Playbooks are YAML files defining a series of tasks to automate workflows.
- Ansible Modules (Common modules like command, shell, copy, service, package, file, template)
  - Modules are reusable units of work (e.g., copy for files, service for managing services).
- Ansible Loops (loop, basic iteration)
  - Loops allow repeating tasks over a list of items.
- Ansible Conditionals (when, basic operators)
  - Conditionals control task execution based on conditions (e.g., OS type).
- Ansible Tags (Purpose, usage, running specific tasks)
  - Tags label tasks for selective execution.
- Ansible Blocks (Basic usage, grouping tasks)
  - Blocks group related tasks for better organization or error handling.
- Ansible Command Module vs. Shell Module (Differences, when to use which)
  - command: Runs simple commands without shell features.
  - shell: Runs commands with shell capabilities (e.g., pipes).

Step 1: Create a Playbook

Create a file named setup_webserver.yml.

```
# File: setup_webserver.yml
---
- name: Set up a basic web server
    hosts: webservers
    tasks:
        # Block for package installation
        - name: Install required packages
            block:
                - name: Install httpd and unzip
                    ansible.builtin.package:
                        name: "{{ item }}"
                        state: present
                    loop:
                        - httpd
                        - unzip
                    tags: install

        # Task with conditional
```

```
        - name: Copy index.html to web server
            ansible.builtin.copy:
                src: ./files/index.html
                dest: /var/www/html/index.html
                mode: '0644'
            when: ansible_os_family == "RedHat"
            tags: configure

        # Task comparing command vs shell
        - name: Check httpd version with command
module
            ansible.builtin.command: httpd -v
            register: httpd_version_cmd
            tags: check

        - name: Check disk usage with shell module
            ansible.builtin.shell: df -h | grep
/dev
            register: disk_usage
            tags: check

        # Service management
        - name: Ensure httpd is running
            ansible.builtin.service:
                name: httpd
                state: started
                enabled: yes
            tags: service
```

Explanation:

- **Playbook Structure:** Starts with ---, defines a play targeting webservers.
- **Modules:** Uses package, copy, command, shell, and service.
- **Loops:** Installs multiple packages (httpd, unzip) with loop.
- **Conditionals:** Copies index.html only on RedHat-based systems.
- **Tags:** Labels tasks as install, configure, check, or service.
- **Blocks:** Groups package installation tasks.
- **Command vs. Shell:** command runs httpd -v (no shell needed); shell runs df -h | grep /dev (needs pipe).

Step 2: Prepare Supporting Files

Create a simple index.html file in a files/ directory.

```
<!-- File: files/index.html -->
<h1>Welcome to My Web Server</h1>
```

Step 3: Use an Inventory

Reuse the hosts.ini from Program 1 (assuming webservers group exists).

```
# File: hosts.ini
[webservers]
web1.example.com
web2.example.com

[all:vars]
ansible_user=admin
ansible_ssh_private_key_file=~/.ssh/id_rsa
```

Step 4: Run the Playbook

Execute the full playbook:

```
ansible-playbook -i hosts.ini setup_webserver.yml
```

Run specific tagged tasks:

```
ansible-playbook -i hosts.ini setup_webserver.yml
--tags "install,configure"
```

Output (example):

```
TASK [Install httpd and unzip] ************
changed: [web1.example.com] => (item=httpd)
changed: [web1.example.com] => (item=unzip)
TASK [Copy index.html to web server] ******
changed: [web1.example.com]
TASK [Check httpd version with command module]
****
changed: [web1.example.com]
TASK [Check disk usage with shell module] ****
changed: [web1.example.com]
TASK [Ensure httpd is running] ************
changed: [web1.example.com]
```

Step 5: Verify Results

Check outputs stored in register: Add a debug task (optional) to see httpd_version_cmd and disk_usage:

```
- name: Debug outputs
    ansible.builtin.debug:
        var: httpd_version_cmd.stdout

- name: Debug disk usage
    ansible.builtin.debug:
        var: disk_usage.stdout
```

Rerun to see idempotence (most tasks show changed: false on second run).

Key Takeaways for Notes:

- **Playbooks:** Automate multi-step workflows in YAML.
- **Modules:** Building blocks for tasks (e.g., copy for files, service for daemons).
- **Loops:** Simplify repetitive tasks.
- **Conditionals:** Add logic to adapt to environments.
- **Tags:** Enable selective task execution.
- **Blocks:** Organize related tasks.
- **Command vs. Shell:** Use command for simple tasks, shell for complex shell features.

Execution Commands:

```
# Run full playbook
ansible-playbook -i hosts.ini setup_webserver.yml

# Run only installation and configuration
ansible-playbook -i hosts.ini setup_webserver.yml
--tags "install,configure"

# Run checks only
ansible-playbook -i hosts.ini setup_webserver.yml
--tags "check"
```

Program 3: Advanced Playbook Features and Reusability

Topics Included:

- **Ansible Roles** (Organization, reusability, basic structure)
    - Roles organize tasks, variables, and files into reusable units.
- **Ansible Variables** (Types, scope, usage)
    - Variables store dynamic data (e.g., package names) with different scopes (play, role, host).
- **Ansible Facts** (Purpose, usage, basic facts)
    - Facts are system details (e.g., OS, IP) gathered from managed nodes.
- **Ansible Handlers** (Purpose, usage, notify)
    - Handlers are tasks triggered by notify when changes occur (e.g., restart a service).
- **Ansible Templates** (Jinja2, basic usage)

- Templates use Jinja2 to generate dynamic files (e.g., config files).

Step 1: Set Up a Role Structure

Create a role named webserver with the standard directory layout.

```
mkdir -p
roles/webserver/{tasks,handlers,templates,vars,fil
es}
```

Explanation: Roles organize code into tasks/ (main logic), handlers/ (triggered tasks), templates/ (dynamic files), vars/ (variables), and files/ (static files).

Step 2: Define Role Components

Main Tasks (roles/webserver/tasks/main.yml):

```
---
- name: Install web server package
    ansible.builtin.package:
        name: "{{ web_package }}"
        state: present
    notify: Restart web service

- name: Copy static index.html
    ansible.builtin.copy:
        src: index.html
        dest: "{{ web_doc_root }}/index.html"
        mode: '0644'

- name: Generate httpd.conf from template
    ansible.builtin.template:
        src: httpd.conf.j2
        dest: /etc/httpd/conf/httpd.conf
        mode: '0644'
    notify: Restart web service

- name: Ensure web service is running
    ansible.builtin.service:
        name: "{{ web_service }}"
        state: started
        enabled: yes
```

Variables (roles/webserver/vars/main.yml):

```
---
web_package: httpd
web_service: httpd
web_doc_root: /var/www/html
```

Handlers (roles/webserver/handlers/main.yml):

```
---
- name: Restart web service
    ansible.builtin.service:
        name: "{{ web_service }}"
        state: restarted
```

Template (roles/webserver/templates/httpd.conf.j2):

```
Listen {{ ansible_default_ipv4.address }}:80
ServerName {{ ansible_hostname }}
DocumentRoot "{{ web_doc_root }}"
<Directory "{{ web_doc_root }}">
        AllowOverride All
        Require all granted
</Directory>
```

Static File (roles/webserver/files/index.html):

```
<h1>Hello from {{ ansible_hostname }}!</h1>
```

Step 3: Create a Playbook to Use the Role

Create deploy_web.yml:

```
---
- name: Deploy web server using role
    hosts: webservers
    pre_tasks:
        - name: Gather facts
            ansible.builtin.setup:
        - name: Debug OS and IP
            ansible.builtin.debug:
                msg: "Running on {{
ansible_os_family }} with IP {{
ansible_default_ipv4.address }}"
    roles:
        - webserver
```

Explanation:

- Roles: The webserver role is applied to webservers.
- Variables: web_package, web_service, etc., are defined in the role's vars/.
- Facts: ansible_os_family, ansible_hostname, and ansible_default_ipv4.address are used dynamically.
- Handlers: Notified when the package or config changes.
- Templates: httpd.conf.j2 uses Jinja2 to insert facts like IP and hostname.

Step 4: Use an Inventory

Reuse hosts.ini from previous programs:

```
# File: hosts.ini
[webservers]
web1.example.com
web2.example.com

[all:vars]
ansible_user=admin
ansible_ssh_private_key_file=~/.ssh/id_rsa
```

Step 5: Run the Playbook

Execute the playbook:

```
ansible-playbook -i hosts.ini deploy_web.yml
```

Output (example):

```
TASK [Debug OS and IP] ************
ok: [web1.example.com] => {
        "msg": "Running on RedHat with IP
192.168.1.10"
}
TASK [webserver : Install web server package] ****
changed: [web1.example.com]
TASK [webserver : Copy static index.html] ****
changed: [web1.example.com]
TASK [webserver : Generate httpd.conf from
template] ****
changed: [web1.example.com]
TASK [webserver : Ensure web service is running]
****
changed: [web1.example.com]
HANDLER [webserver : Restart web service] ****
changed: [web1.example.com]
```

Step 6: Verify Results

On web1.example.com, check:

- curl http://192.168.1.10: Should show "Hello from web1!" (hostname from facts).
- /etc/httpd/conf/httpd.conf: Contains the IP and hostname from the template.

Key Takeaways for Notes:

- Roles: Modularize tasks for reusability (e.g., webserver role can be reused across projects).
- Variables: Define constants (e.g., web_package) in vars/ for flexibility.
- Facts: Automatically gather system info (e.g., ansible_hostname) for dynamic configs.
- Handlers: Trigger actions (e.g., service restart) only when needed.
- Templates: Use Jinja2 to create dynamic files based on facts and variables.

Execution Command:

```
ansible-playbook -i hosts.ini deploy_web.yml
```

Program 4: Security and Operational Control

Topics Included:

- **Ansible Vault** (Basic encryption, usage)
  - Vault encrypts sensitive data (e.g., passwords) in files.
- **Ansible Privilege Escalation** (become, become_user)
  - become escalates privileges (e.g., to root) for tasks requiring elevated access.
- **Ansible Check Mode (Dry Run)** (--check)
  - Check mode simulates tasks without making changes.
- **Ansible Best Practices** (Organization, security, readability)
  - Best practices include clear naming, modular structure, and secure handling of secrets.

Step 1: Encrypt Sensitive Data with Ansible Vault

Create an encrypted file secrets.yml for sensitive variables.

```
ansible-vault create secrets.yml
```

Enter a vault password (e.g., mypassword) when prompted, then add:

```
# File: secrets.yml
db_password: "securepass123"
```

Explanation: Vault encrypts secrets.yml to protect db_password.

Step 2: Create a Playbook with Security Features

Create secure_setup.yml:

```
---
- name: Securely set up a database server
    hosts: dbservers
    vars_files:
        - secrets.yml  # Include encrypted
variables
    tasks:
        - name: Install MariaDB package
            ansible.builtin.package:
                name: mariadb-server
                state: present
            become: yes  # Escalate privileges to
root
            become_user: root
            tags: install

        - name: Ensure MariaDB service is running
            ansible.builtin.service:
                name: mariadb
                state: started
                enabled: yes
            become: yes
            become_user: root
            tags: service

        - name: Set database root password
            ansible.builtin.shell: mysqladmin -u
root password "{{ db_password }}"
            when: ansible_os_family == "RedHat"
            become: yes
            become_user: root
            tags: configure
            no_log: true  # Hide sensitive output
(best practice)
```

Explanation:

- **Vault:** secrets.yml provides db_password.
- **Privilege Escalation:** become: yes and become_user: root allow installing packages and managing services.
- **Check Mode:** Can be tested with --check.
- **Best Practices:**
  - Clear task names (e.g., "Install MariaDB package").
  - no_log: true hides sensitive data in logs.
  - Modular structure with tags (install, service, configure).

Step 3: Use an Inventory

Reuse or adapt hosts.ini:

```
# File: hosts.ini
[dbservers]
db1.example.com
[all:vars]
ansible_user=admin
ansible_ssh_private_key_file=~/.ssh/id_rsa
```

Step 4: Run the Playbook

Dry Run (Check Mode):

```
ansible-playbook -i hosts.ini secure_setup.yml --
check --ask-vault-pass
```

Enter the vault password (mypassword) when prompted.

Output (example):

```
TASK [Install MariaDB package] ************
ok: [db1.example.com] => (skipped, in check mode)
TASK [Ensure MariaDB service is running] ****
ok: [db1.example.com] => (skipped, in check mode)
TASK [Set database root password] *********
ok: [db1.example.com] => (skipped, in check mode)
Full Execution:
ansible-playbook -i hosts.ini secure_setup.yml --
ask-vault-pass
Output (example):
TASK [Install MariaDB package] ************
changed: [db1.example.com]
TASK [Ensure MariaDB service is running] ****
changed: [db1.example.com]
TASK [Set database root password] *********
changed: [db1.example.com]
```

Step 5: Verify Results

On db1.example.com:

- Check if mariadb-server is installed (rpm -q mariadb-server).
- Verify MariaDB is running (systemctl status mariadb).
- Test the root password (mysql -u root -p with securepass123).

Key Takeaways for Notes:

- **Vault:** Encrypts sensitive data (e.g., db_password) for security.
- **Privilege Escalation:** become ensures tasks requiring root access succeed.
- **Check Mode:** --check previews changes without applying them.
- **Best Practices:**
  - Use descriptive names and tags.
  - Hide sensitive output with no_log.
  - Store secrets in Vault, not plaintext.

Execution Commands:

```
# Create/edit Vault file
ansible-vault edit secrets.yml --ask-vault-pass
# Dry run
ansible-playbook -i hosts.ini secure_setup.yml --
check --ask-vault-pass
# Full run
ansible-playbook -i hosts.ini secure_setup.yml --
ask-vault-pass
```

Program 5: Debugging and Validation

Topics Included:

- **Ansible Debugging** (Basic techniques, -v, debug module)
  - Debugging tools like verbose mode (-v) and the debug module help troubleshoot issues.
- **Ansible Check Mode (Dry Run)** (--check)
  - Check mode simulates playbook execution without applying changes.

Step 1: Create a Playbook for Debugging

Create debug_validate.yml:

```
---
- name: Debug and validate system setup
    hosts: webservers
    tasks:
        - name: Gather facts
            ansible.builtin.setup:
            tags: facts

        - name: Debug system OS and memory
            ansible.builtin.debug:
                msg: "OS: {{ ansible_os_family }}, 
Free Memory: {{ ansible_memfree_mb }} MB"
            tags: debug

        - name: Install httpd package
            ansible.builtin.package:
                name: httpd
                state: present
            register: install_result  # Store task 
output
            tags: install

        - name: Debug installation result
            ansible.builtin.debug:
                var: install_result
            when: install_result is defined
            tags: debug

        - name: Ensure httpd is running
            ansible.builtin.service:
                name: httpd
                state: started
            register: service_result
            tags: service

        - name: Debug service status
            ansible.builtin.debug:
                msg: "Service changed: {{ 
service_result.changed }}, State: {{ 
service_result.state }}"
            when: service_result is defined
            tags: debug
```

Explanation:

- **Debugging:** Uses debug module to print facts (e.g., OS, memory) and task results.
- **Check Mode:** Can simulate package installation and service management.
- **Register:** Captures task outputs (install_result, service_result) for inspection.

Step 2: Use an Inventory

Reuse hosts.ini from previous programs:

```
# File: hosts.ini
[webservers]
web1.example.com

[all:vars]
ansible_user=admin
ansible_ssh_private_key_file=~/.ssh/id_rsa
```

Step 3: Run the Playbook with Debugging

Verbose Mode (Basic):

```
ansible-playbook -i hosts.ini debug_validate.yml -
v
```

Verbose Mode (Detailed):

```
ansible-playbook -i hosts.ini debug_validate.yml -
vvv
```

Output (example with -v):

```
TASK [Debug system OS and memory] ************
ok: [web1.example.com] => {
        "msg": "OS: RedHat, Free Memory: 2048 MB"
}
TASK [Install httpd package] ************
changed: [web1.example.com] => {"changed": true, 
"name": "httpd"}
TASK [Debug installation result] ************
ok: [web1.example.com] => {
        "install_result": {"changed": true, 
"name": "httpd", "state": "present"}
}
```

Explanation:

- -v shows task outputs; -vvv adds detailed execution info (e.g., SSH commands).

Step 4: Run in Check Mode

Simulate execution:

```
ansible-playbook -i hosts.ini debug_validate.yml -
-check
```

Output (example):

```
TASK [Debug system OS and memory] ************
ok: [web1.example.com] => {
        "msg": "OS: RedHat, Free Memory: 2048 MB"
}
TASK [Install httpd package] ************
ok: [web1.example.com] => (skipped, in check mode)
TASK [Debug installation result] ************
skipping: [web1.example.com]  # Skipped because 
install_result isn't set in check mode
TASK [Ensure httpd is running] ************
ok: [web1.example.com] => (skipped, in check mode)
```

Explanation: Check mode runs debug tasks but skips changes (e.g., package install).

Step 5: Verify Debugging Output

Rerun with tags to focus on debugging:

```
ansible-playbook -i hosts.ini debug_validate.yml -
-tags "debug" -v
```

Output (example):

```
TASK [Debug system OS and memory] ************
ok: [web1.example.com] => {
        "msg": "OS: RedHat, Free Memory: 2048 MB"
}
TASK [Debug installation result] ************
ok: [web1.example.com] => {
        "install_result": {"changed": false, 
"name": "httpd"}
}
TASK [Debug service status] ************
ok: [web1.example.com] => {
        "msg": "Service changed: false, State: 
started"
}
```

Key Takeaways for Notes:

- **Debugging:**
  - -v to -vvv: Increases verbosity for troubleshooting.
  - debug module: Prints variables, facts, or task results (e.g., ansible_memfree_mb).
- **Check Mode:** --check validates playbook logic without altering systems.
- Combine register with debug to inspect task outcomes.

Execution Commands:

```
# Run with basic verbosity
ansible-playbook -i hosts.ini debug_validate.yml -
v
# Run with maximum verbosity
ansible-playbook -i hosts.ini debug_validate.yml -
vvv
# Run in check mode
ansible-playbook -i hosts.ini debug_validate.yml -
-check
```

```
# Run debug tasks only
ansible-playbook -i hosts.ini debug_validate.yml --tags "debug"
```

Program 6: Ansible Ecosystem and Reusable Content Management

Topics Included:

- **Ansible Galaxy** (Purpose, usage, finding roles)
- **Ansible Collections** (Purpose, benefits, basic usage)
- Ansible Playbook Includes and Imports (Differences, usage)

Rationale: Ansible Galaxy and Ansible Collections are both part of Ansible's ecosystem for managing reusable content (roles and collections). Galaxy is a hub for finding roles, while Collections extend this concept with modular, reusable code including roles, modules, and plugins. Playbook Includes and Imports tie into this by allowing you to integrate Galaxy roles or Collection content into your playbooks dynamically (import_role, include_tasks) or statically. s Program Example: A playbook that pulls a role from Galaxy (e.g., configuring an Nginx server), uses a Collection for additional utilities (e.g., community.general), and demonstrates import_role vs. include_tasks for modularity.

Program:

```
- name: Deploy Nginx using Galaxy Role and Collections
    hosts: webservers
    tasks:
        - name: Import Nginx role from Galaxy
            ansible.builtin.import_role:
                name: geerlingguy.nginx  # Fetched via ansible-galaxy
        - name: Use Collection module for additional setup
            community.general.package_facts:
                manager: apt
        - name: Include dynamic tasks
            ansible.builtin.include_tasks: setup_firewall.yml
```

Program 7: Data Manipulation and Dynamic Playbooks

Topics Included:

- **Ansible Filters** (Basic usage, data manipulation)
- **Ansible Lookup Plugins** (Basic understanding, usage)
- **Ansible Playbook Variables Precedence** (Understanding the order)

Rationale: Filters and Lookup Plugins are tools for manipulating and retrieving data dynamically within playbooks. Filters transform data (e.g., | json_query), while Lookups fetch external data (e.g., lookup('file', 'path')). Variables Precedence is critical here because it determines how variables (used in filters or lookups) are overridden or prioritized (e.g., playbook vars vs. role vars). Program Example: A playbook that reads data from a file using a lookup, manipulates it with filters, and respects variable precedence for customization.

Program:

```
- name: Process server data dynamically
    hosts: all
    vars:
        default_port: 80
    tasks:
        - name: Read config from file using lookup
            ansible.builtin.set_fact:
                config_data: "{{ lookup('file', 'config.json') | from_json }}"
        - name: Filter and transform data
            ansible.builtin.debug:
                msg: "Server: {{ config_data.servers | map(attribute='name') | join(', ') }}"
        - name: Show variable precedence (playbook vars override defaults)
            ansible.builtin.debug:
                msg: "Port: {{ port | default(default_port) }}"
```

Program 8: Robust Automation with Error Handling and Scaling

Topics Included:

- **Ansible Dynamic Inventory** (Basic concept, benefits)
- **Ansible Error Handling** (ignore_errors, failed_when)
- Ansible Forks (Basic understanding)

Rationale: Dynamic Inventory allows Ansible to adapt to changing environments (e.g., cloud instances), which pairs well with Forks for parallel execution across multiple hosts. Error Handling ensures robustness by managing failures (e.g., ignoring non-critical errors or defining custom failure conditions).

Program Example: A playbook that uses a dynamic inventory (e.g., AWS EC2), handles errors gracefully, and scales with forks.

Program:

```
- name: Manage cloud servers with error handling
    hosts: all
    # Dynamic inventory assumed (e.g., ec2.py script)
    forks: 10  # Parallel execution
    tasks:
        - name: Install package with error handling
            ansible.builtin.package:
                name: httpd
                state: present
            ignore_errors: yes  # Continue despite failures
        - name: Check service status
            ansible.builtin.command: systemctl status httpd
            register: result
            failed_when: "'running' not in result.stdout"  # Custom failure condition
        - name: Debug result
            ansible.builtin.debug:
                msg: "Service is {{ 'up' if 'running' in result.stdout else 'down' }}"
```

Program 9: Controlled Deployment with Rolling Updates

Topics Included:

- **Ansible Rolling Updates** (serial)

Rationale: Rolling Updates (serial) is a standalone but critical concept for managing deployments in production environments, ensuring minimal downtime by updating hosts in batches. This can sbe a dedicated program as it's often used independently or combined with other features (e.g., error handling from Set 3).

Program Example: A playbook that updates a web application across multiple servers in batches.

Program:

```
- name: Perform rolling update on web servers
    hosts: webservers
    serial: 2  # Update 2 hosts at a time
    tasks:
        - name: Update application package
            ansible.builtin.package:
                name: myapp
                state: latest
        - name: Restart service
            ansible.builtin.service:
                name: myapp
                state: restarted
        - name: Verify application
            ansible.builtin.uri:
                url: "http://{{ inventory_hostname
}}/health"
                status_code: 200
```

GitHub Actions: (Definitely Needed)

Revision Notes: GitHub Actions CI/CD Pipeline

Program: .github/workflows/ci-cd.yml

```
name: CI-CD Pipeline  # Step 1: Naming the
workflow

on:  # Step 1: Basic trigger setup
    push:
        branches:  # Step 3: Branch filters
            - main
            - 'feature/*'
    pull_request:
        branches:
            - main

env:  # Step 5: Environment variables
    NODE_ENV: test  # Global env var for
consistency

jobs:
    lint:  # Step 4: First job in a multi-job
setup
        runs-on: ubuntu-latest  # Step 1: Runner
specification
        steps:
            - name: Checkout code  # Step 2:
Accessing repo code
                uses: actions/checkout@v4

            - name: Set up Node.js  # Step 2:
Preparing environment
                uses: actions/setup-node@v4
                with:
                    node-version: '20'

            - name: Install dependencies  # Step
2: Running a script
                run: npm install

            - name: Run linting  # Step 2:
Executing a task
                run: npm run lint

    test:  # Step 4: Second job with dependency
        needs: lint  # Step 4: Job dependency
        runs-on: ubuntu-latest
        steps:
            - name: Checkout code  # Step 2:
Repeated for isolation
                uses: actions/checkout@v4

            - name: Set up Node.js
                uses: actions/setup-node@v4
                with:
                    node-version: '20'

            - name: Install dependencies
                run: npm install

            - name: Run tests  # Step 3: Adding
testing
                run: npm test

    deploy:  # Step 4: Third job with dependency
        needs: test  # Step 4: Depends on test
passing
        if: github.ref == 'refs/heads/main'  #
Step 5: Conditional deployment
        runs-on: ubuntu-latest
        steps:
            - name: Checkout code
                uses: actions/checkout@v4

            - name: Set up Node.js
                uses: actions/setup-node@v4
                with:
                    node-version: '20'

            - name: Install dependencies
                run: npm install

            - name: Build site  # Step 5: Using
env vars
                run: npm run build
                env:
                    DEPLOY_ENV: production  # Step
5: Job-specific env var

            - name: Deploy to GitHub Pages  # Step
5: Deployment with secrets
                uses: peaceiris/actions-gh-
pages@v3
                with:
                    github_token: ${{
secrets.GITHUB_TOKEN }}  # Step 5: Secret usage
                    publish_dir: ./dist
```

Required Repo Files

package.json

```
{
    "name": "my-project",
    "version": "1.0.0",
    "scripts": {
        "lint": "eslint .",
        "test": "jest",
        "build": "mkdir -p dist && echo
'<h1>Deployed!</h1>' > dist/index.html"
    },
    "devDependencies": {
        "eslint": "^8.57.0",
        "jest": "^29.7.0"
    }
}
```

index.test.js

```
test('basic test', () => {
    expect(1 + 1).toBe(2);
});
```

.eslintrc.json

Basic config—use eslint:recommended.

Concept Explanations (Your Revision Notes)

Step 1: Basic Workflow Setup

What: Defines the workflow's name (name) and trigger (on).

In the Code: name: CI-CD Pipeline and on: push/pull_request start the pipeline on code pushes or PRs.

Relatable Explanation: "It's like setting an alarm clock—tells GitHub when to wake up and run my tasks, like pushing code is the trigger."

Why It Matters: Every pipeline needs a starting point—interviewers ask this first.

Step 2: Checking Out Code and Running a Script

What: Uses actions/checkout@v4 to access repo files and run to execute commands (e.g., npm install, node index.js).

In the Code: Each job has uses: actions/checkout@v4 and runs scripts like npm install or npm run lint.

Relatable Explanation: "Imagine borrowing a book from the library (checkout) and then reading it aloud (run)—I need the code before I can do anything with it."
Why It Matters: Without this, the runner's a blank slate—core to any task.
Step 3: Adding Testing and Branch Filters
What: Runs tests (npm test) and limits triggers to specific branches (branches: [main, 'feature/*']).
In the Code: test job runs npm test, and on: push: branches filters to main and feature/*.
Relatable Explanation: "It's like only studying for specific exams (branches) and then taking a quiz (test) to check my work—keeps things focused."
Why It Matters: Testing ensures quality; filters save resources—standard CI stuff.
Step 4: Multiple Jobs with Dependencies
What: Splits tasks into jobs (lint, test, deploy) with needs to enforce order.
In the Code: lint runs first, test needs lint, and deploy needs test—a chain of tasks.
Relatable Explanation: "Think of a relay race—lint passes the baton to test, then test to deploy. No one runs until the previous runner's done."
Why It Matters: Shows you can organize complex workflows—mid-level skill.
Step 5: Env Vars, Secrets, and Deployment
What: Uses env for configuration, secrets for sensitive data, and deploys (e.g., to GitHub Pages).
In the Code: env: NODE_ENV: test globally, DEPLOY_ENV in deploy, secrets.GITHUB_TOKEN for auth, and peaceiris/actions-gh-pages@v3 for deployment.
Relatable Explanation: "It's like setting the thermostat (env), locking my diary (secrets), and mailing a package (deploy)—configures, secures, and ships my app."
Why It Matters: Real-world pipelines need these—interviewers test this often.

How It Works (Big Picture)

- Push to feature/*: Lint, then test—no deploy.
- Push to main: Lint, test, deploy to GitHub Pages if all pass.
- PR to main: Lint and test as a check before merging.
- Env/Secrets: NODE_ENV sets test mode; GITHUB_TOKEN securely authenticates deployment.

Revision Tips

- Memorize the Flow: Trigger → Lint → Test → Deploy (if main).
- Key Lines: on: push, uses: actions/checkout@v4, needs:, secrets.GITHUB_TOKEN.
- Practice Explaining: Use the relatable analogies—interviewers love clarity.

Github Actions (Add on concepts):

Below, I've created a comprehensive revision note for your GitHub Actions learning, merging Steps 6 through 10 into a single, cohesive program where possible. Since some concepts (like self-hosted runners) can't fully merge into a single YAML file without real infrastructure, I'll provide a main workflow with most features and a separate note for self-hosted runners. Each section includes the code and a relatable explanation tailored for your revision—think of it as a cheat sheet you can revisit before interviews!

Revision Notes: GitHub Actions Master Workflow

Goal

This is a production-ready CI/CD pipeline for a Node.js project that tests across environments, deploys dynamically, and handles errors—covering Steps 6-10.

Main Workflow Program: .github/workflows/ci-cd.yml

```
name: Advanced CI-CD Pipeline

# Triggers (Step 9: Dynamic Workflows)
on:
    push:
        branches: [main, 'feature/*']
    pull_request:
        branches: [main]
    workflow_dispatch:
        inputs:
            environment:
                description: 'Deploy environment
(staging/production)'
                required: true
                default: 'staging'
            log-level:
                description: 'Log verbosity'
                default: 'info'

# Global Env Vars
env:
    NODE_ENV: production

jobs:
    # Step 6: Matrix Builds
    test:
        runs-on: ubuntu-latest  # Could be self-
hosted (Step 8)
        strategy:
            matrix:
                node-version: [18, 20]
            fail-fast: false  # Step 10: Error
Handling
        container:  # Step 8: Docker
            image: node:${{ matrix.node-version }}
        steps:
            - name: Checkout Code
                uses: actions/checkout@v4

            # Step 7: Caching
            - name: Cache Node Modules
                id: cache-npm
                uses: actions/cache@v3
                with:
                    path: ~/.npm
                    key: ${{ runner.os }}-node-${{
matrix.node-version }}-${{ hashFiles('**/package-
lock.json') }}
                    restore-keys: ${{ runner.os
}}-node-${{ matrix.node-version }}-

            - name: Install Dependencies
                if: steps.cache-npm.outputs.cache-
hit != 'true'  # Step 7: Conditionals
                run: npm install

            - name: Run Tests
```

```
                run: npm test
                continue-on-error: true  # Step
10: Error Handling

            # Step 6: Custom Action
            - name: Custom Failure Alert
                if: failure()  # Step 7:
Conditionals, Step 10: Error Handling
                uses: ./.github/actions/failure-
alert
                with:
                    message: 'Tests failed on Node
${{ matrix.node-version }}'

            # Step 7: Artifacts
            - name: Upload Test Logs
                if: always()  # Step 7:
Conditionals
                uses: actions/upload-artifact@v3
                with:
                    name: test-logs-node-${{
matrix.node-version }}
                    path: ./test-logs/*.log

    # Step 10: Status Checks
    status-check:
        needs: test
        if: github.event_name == 'pull_request'
        runs-on: ubuntu-latest
        steps:
            - name: Verify Status
                run: |
                    if [ "${{ job.status }}" ==
"success" ]; then
                        echo "All tests passed!"
                        exit 0
                    else
                        echo "Tests failed - check
logs."
                        exit 1
                    fi

    # Step 9: Dynamic Workflows + Step 10:
Advanced Deployment
    deploy:
        needs: test
        if: (github.ref == 'refs/heads/main' ||
github.event_name == 'workflow_dispatch') &&
success()
        runs-on: ubuntu-latest
        environment: ${{
github.event.inputs.environment || 'production' }}
        steps:
            - name: Checkout Code
                uses: actions/checkout@v4

            - name: Set Up Node.js
                uses: actions/setup-node@v4
                with:
                    node-version: '20'

            - name: Install Dependencies
                run: npm install

            # Step 9: Dynamic Step
            - name: Generate Dynamic Build Command
                id: dynamic-build
                run: |
                    echo "build-cmd=npm run build
-- --env ${{ github.event.inputs.environment ||
'production' }}" >> $GITHUB_OUTPUT

            - name: Build Site
                run: ${{ steps.dynamic-
build.outputs.build-cmd }}

            - name: Deploy to GitHub Pages
                uses: peaceiris/actions-gh-
pages@v3
                with:
                    github_token: ${{
secrets.GITHUB_TOKEN }}
                    publish_dir: ./dist

            # Step 10: Error Handling Post-Deploy
            - name: Verify Deployment
                run: |
                    if [ $? -eq 0 ]; then
                        echo "Deployed to ${{
github.event.inputs.environment || 'production' }}
successfully!"
                    else
                        echo "Deployment failed!"
&& exit 1
```

Custom Action: .github/actions/failure-alert/action.yml

```
name: 'Failure Alert'
description: 'Logs a failure message'
inputs:
    message:
        description: 'Failure message'
        required: true
runs:
    using: 'node16'
    main: 'index.js'
```

.github/actions/failure-alert/index.js

```
const core = require('@actions/core');
const message = core.getInput('message');
console.log(`ALERT: ${message}`);
```

Supporting Files (Assumptions for Revision)

package.json:

```
{
    "scripts": {
        "test": "jest --outputFile=./test-
logs/test.log",
        "build": "mkdir -p dist && echo '<h1>Built
for $DEPLOY_ENV</h1>' > dist/index.html"
    },
    "devDependencies": { "jest": "^29.7.0" }
}
```

Repo Setup: Ensure test-logs/ exists and branch protection rules require status-check.

Concept-by-Concept Explanation

Step 6: Matrix Builds and Custom Actions

- Code: strategy: matrix runs tests on Node 18 and 20; custom action at .github/actions/failure-alert.
- Explanation: Matrix builds test all combos at once, like a factory QC check. The custom action alerts when something breaks.
- Interview Bit: "I used a matrix to ensure compatibility and a custom action to alert on failures—keeps things modular."

Step 7: Conditionals, Caching, and Artifacts

- Code: if: steps.cache-npm.outputs.cache-hit != 'true', actions/cache@v3, and actions/upload-artifact@v3.
- Explanation: Caching speeds up builds, conditionals skip redundant steps, and artifacts let you debug later.
- Interview Bit: "Caching speeds up builds, conditionals skip redundant steps, and artifacts let me debug later."

Step 8: Docker and Self-Hosted Runners

- Code: container: image: node:${{ matrix.node-version }} (Docker); runs-on: self-hosted (not fully merged—see below).
- Explanation: Docker ensures a consistent environment; self-hosted

runners give control for special cases.

- Interview Bit: "Docker ensures my env is consistent; self-hosted runners give me control for special cases."

Separate Note for Self-Hosted:

- Replace runs-on: ubuntu-latest with runs-on: self-hosted after configuring a runner in repo settings.

Step 9: Dynamic Workflows and Reusability

- Code: workflow_dispatch with inputs, dynamic step via echo ... >> $GITHUB_OUTPUT.
- Explanation: Dynamic workflows let you customize runs manually; reusability keeps code DRY.
- Interview Bit: "Dynamic workflows let me customize runs manually; reusability keeps code DRY."

Step 10: Error Handling, Status Checks, and Advanced Deployment

- Code: continue-on-error, fail-fast: false, status-check job, environment: with post-deploy check.
- Explanation: Error handling ensures tests can stumble but still finish. Status checks enforce PR quality. Advanced deployment targets and verifies delivery.
- Interview Bit: "I handle errors gracefully, enforce PR quality, and deploy with precision."