**1. Explain Kubernetes Architecture / Explain the Kubernetes architecture and the roles of its components.**

**============================================================================**

* **Master Node Components:**
  + **API Server: Frontend to Kubernetes control plane, exposes Kubernetes API.**
  + **etcd: Distributed key-value store to hold cluster state data.**
  + **Controller Manager: Runs controllers like Node Controller, Replication Controller.**
  + **Scheduler: Assigns pods to nodes based on resource availability.**
* **Worker Node Components:**
  + **Kubelet: Agent running on nodes, ensures containers are running in Pods.**
  + **Kube-proxy: Network proxy for service abstraction, manages network rules.**
  + **Container Runtime: Runs containers (Docker, containerd, CRI-O).**

**Step-by-step:**

* **API Server processes REST commands from CLI or UI.**
* **Scheduler places pods on nodes.**
* **Controllers ensure desired state is met (like replica count).**
* **Kubelet communicates with API server to manage pods on nodes.**

**2. Explain Pod Lifecycle and Management / Describe the Pod lifecycle and how Kubernetes manages Pods.**

**====================================================================================**

* **Pod phases: Pending → Running → Succeeded/Failed → Unknown**
* **Kubelet creates the pod containers using container runtime.**
* **If pod dies, Controllers recreate pods (if under ReplicaSet/Deployment).**
* **Liveness and readiness probes help Kubernetes know if pod is healthy and ready.**

**3. How do you perform rolling updates and rollbacks in Kubernetes?**

**====================================================================================**

* **Use kubectl apply -f deployment.yaml to update deployment.**
* **Kubernetes creates new ReplicaSets with the new version.**
* **Rolling update replaces pods gradually (default maxUnavailable=25%, maxSurge=25%).**
* **Rollback: kubectl rollout undo deployment/<deployment-name>**

**Example Step-by-step:**

* **Modify image/version in deployment manifest.**
* **Apply changes → new pods start before old pods terminate.**
* **Monitor rollout status: kubectl rollout status deployment/<name>**
* **Undo if needed to previous stable version.**

**+++++ Explanation ++++++**

* **Rolling Update: Gradually replaces Pods of the old version with Pods of the new version without downtime.**
* **Rollback: Reverts the Deployment to a previous version in case the new version has issues.**

**Step 1: Create an initial Deployment**

**Let's create a Deployment with an NGINX container version 1.14.2.**

**# nginx-deployment.yaml**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: nginx-deployment**

**spec:**

**replicas: 3**

**selector:**

**matchLabels:**

**app: nginx**

**template:**

**metadata:**

**labels:**

**app: nginx**

**spec:**

**containers:**

**- name: nginx**

**image: nginx:1.14.2**

**ports:**

**- containerPort: 80**

**Apply this Deployment:**

**kubectl apply -f nginx-deployment.yaml**

**Check pods:**

**kubectl get pods -l app=nginx**

**Step 2: Perform a rolling update**

**Update the image version from nginx:1.14.2 to nginx:1.16.1.**

**You can edit the YAML or do a direct command:**

**kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1**

**Kubernetes will start replacing pods gradually with new pods using the new image version without downtime.**

**Check rollout status:**

**kubectl rollout status deployment/nginx-deployment**

**Step 3: Verify the update**

**Check the pods' image versions:**

**kubectl get pods -l app=nginx -o jsonpath="{.items[\*].spec.containers[0].image}"**

**You should see all pods running nginx:1.16.1.**

**Step 4: Rollback to previous version if needed**

**If the new version has problems, rollback to the previous version:**

**kubectl rollout undo deployment/nginx-deployment**

**Check rollout status again:**

**kubectl rollout status deployment/nginx-deployment**

**check rollout history and status:**

* **View rollout history:**

**kubectl rollout history deployment/nginx-deployment**

* **View details about a specific revision:**

**kubectl rollout history deployment/nginx-deployment --revision=1**

**4. Networking in Kubernetes / Q: Explain how networking works in Kubernetes.**

**=============================================================================**

**K8 Networking Aspect What it Does Example Resource**

**Pod-to-Pod communication Direct IP communication between Pods Pod**

**Stable network endpoint Expose group of Pods via DNS/IP Service (ClusterIP)**

**External access Access Service from outside Service (NodePort/LoadBalancer)**

**Network control Restrict traffic flow between Pods NetworkPolicy**

**============================================================================**

**External access Uses NodePort/LoadBalancer; still may involve kube-proxy but for outside-in traffic.**

**Stable network endpoint Handled via Services. Kube-proxy manages IP routing to backends**

**+++++++++++**

**STEP 1 -- Pod-to-Pod Direct Communication**

**List all pods and get their IPs:**

**kubectl get pods -o wide**

**Example output:**

**NAME READY STATUS RESTARTS AGE IP NODE**

**pod-a 1/1 Running 0 10m 10.244.1.5 node-1**

**pod-b 1/1 Running 0 8m 10.244.2.7 node-2**

**Here, pod-a has IP 10.244.1.5, and pod-b has IP 10.244.2.7.**

**Pick a pod, exec into it:**

**kubectl exec -it <pod-name> -- sh**

**kubectl exec -it pod-a -- sh You are now “inside” pod-a’s container.**

**Try ping or curl another pod using its IP:**

**ping <other-pod-ip>**

**ex 🡪 ping 10.244.2.7**

**You should see responses like:**

**PING 10.244.2.7 (10.244.2.7): 56 data bytes**

**64 bytes from 10.244.2.7: icmp\_seq=0 ttl=64 time=0.123 ms**

**64 bytes from 10.244.2.7: icmp\_seq=1 ttl=64 time=0.100 ms**

**Or**

**if it’s an HTTP server running on pod-b, you can:**

**ex 🡪 curl** [**http://10.244.2.7:8080**](http://10.244.2.7:8080) **🡪 you will get a response directly**

**You’ll see that pod-to-pod communication works within the cluster without NAT.**

**Step 2 🡪 Service Discovery / Services Provide Stable Network Endpoints**

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

** Pods are ephemeral and may be recreated with different IPs.**

** Kubernetes Services provide a stable IP and DNS name to access a group of Pods.**

** Services load balance traffic to Pods matching a label selector.**

**Create a Pod running nginx:**

**# nginx-pod.yaml**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: nginx-pod**

**labels:**

**app: nginx**

**spec:**

**containers:**

**- name: nginx**

**image: nginx**

**ports:**

**- containerPort: 80**

**Create a Service to expose nginx Pod inside the cluster:**

**# nginx-service.yaml**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: nginx-service**

**spec:**

**selector:**

**app: nginx**

**ports:**

**- protocol: TCP**

**port: 80 # Service port**

**targetPort: 80 # Pod container port**

**Apply both:**

**kubectl apply -f nginx-pod.yaml**

**kubectl apply -f nginx-service.yaml**

**Check Service IP:**

**kubectl get svc nginx-service**

**Output:**

**NAME TYPE CLUSTER-IP PORT(S) AGE**

**nginx-service ClusterIP 10.96.25.100 80/TCP 1m**

**You can now access nginx via this stable IP or DNS name nginx-service from other Pods inside the cluster.**

**Accessing Services from Inside the Cluster**

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

**Create a Pod with a curl client to test access:**

**# curl-pod.yaml**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: curl-pod**

**spec:**

**containers:**

**- name: curl**

**image: curlimages/curl**

**command: ["sleep", "3600"]**

**Apply: kubectl apply -f curl-pod.yaml**

**Exec into curl-pod and curl nginx service by DNS name:**

**kubectl exec -it curl-pod – curl http://nginx-service**

**You should get the nginx welcome page HTML output.**

**Step 3 🡪 Exposing Services Outside the Cluster**

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

* **To allow access from outside, change Service type to NodePort or LoadBalancer.**

**Example NodePort service:**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: nginx-nodeport**

**spec:**

**selector:**

**app: nginx**

**ports:**

**- port: 80**

**targetPort: 80**

**nodePort: 30080**

**type: NodePort**

**kubectl apply -f nginx-nodeport.yaml**

**kubectl get nodes -o wide**

**Now you can access nginx from outside by hitting:**

**http://<Node-IP>:30080**

**Step 4 🡪 Controlling Traffic with Network Policies**

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

**By default, all Pods can talk to each other.**

**You can create NetworkPolicies to restrict which Pods can talk to which.**

**Example allowing only Pods labeled role=frontend to access nginx on port 80:**

**apiVersion: networking.k8s.io/v1**

**kind: NetworkPolicy**

**metadata:**

**name: allow-frontend**

**spec:**

**podSelector:**

**matchLabels:**

**app: nginx**

**ingress:**

**- from:**

**- podSelector:**

**matchLabels:**

**role: frontend**

**ports:**

**- protocol: TCP**

**port: 80**

**5. Explain how Kubernetes handles storage?**

**================================================================**

**Kubernetes supports Volumes which persist data beyond the lifecycle of a Pod.**

**There is different types of volumes: emptyDir, hostPath, persistentVolumeClaim (PVC), etc.**

**PersistentVolumes (PV) represent actual storage resources. It is a piece of storage in the cluster provisioned by an admin or dynamically provisioned using a StorageClass (e.g., SSD vs HDD, reclaim policy, etc.).**

**PersistentVolumeClaim (PVC): A user's request for storage.**

**1. Define a StorageClass**

**# storage-class.yaml**

**apiVersion: storage.k8s.io/v1**

**kind: StorageClass**

**metadata:**

**name: standard**

**provisioner: kubernetes.io/aws-ebs # Use a provisioner (e.g., aws-ebs, gce-pd, csi)**

**parameters:**

**type: gp2**

**reclaimPolicy: Retain**

**volumeBindingMode: WaitForFirstConsumer**

**Apply it:**

**kubectl apply -f storage-class.yaml**

**2. Create a PersistentVolumeClaim (PVC)**

**# pvc.yaml**

**apiVersion: v1**

**kind: PersistentVolumeClaim**

**metadata:**

**name: my-pvc**

**spec:**

**accessModes:**

**- ReadWriteOnce**

**resources:**

**requests:**

**storage: 1Gi**

**storageClassName: standard**

**Apply it:**

**kubectl apply -f pvc.yaml**

**This triggers dynamic provisioning using the StorageClass. Kubernetes creates a PersistentVolume behind the scenes.**

**Now Create a Pod that Uses the PVC**

**# pod-using-pvc.yaml**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: pvc-demo-pod**

**spec:**

**containers:**

**- name: app**

**image: busybox**

**command: ["sleep", "3600"]**

**volumeMounts:**

**- mountPath: "/data"**

**name: storage**

**volumes:**

**- name: storage**

**persistentVolumeClaim:**

**claimName: my-pvc**

**Apply it:**

**kubectl apply -f pod-using-pvc.yaml**

**Now your container can write to /data, and data will persist as long as the volume exists, even if the Pod is deleted.**

**kubectl get pvc**

**kubectl get pv**

**kubectl get pod pvc-demo-pod -o yaml**

**6. Explain the difference between Deployment, ReplicaSet, StatefulSet, and DaemonSet Controllers.**

**====================================================================================**

**Answer:**

* **Deployment: Manages stateless apps, handles rolling updates and rollbacks.**
* **ReplicaSet: Ensures specified number of pod replicas are running, usually managed by Deployments.**
* **StatefulSet: Manages stateful apps, provides stable identities and persistent storage.**
* **DaemonSet: Ensures a copy of a pod runs on all (or some) nodes, e.g., for logging or monitoring agents.**

**7. What are Controllers in Kubernetes?**

**====================================================**

**Answer:**

**Controllers monitor the state of the cluster via the API server and make or request changes to move the current state towards the desired state.**

**They automate tasks such as:**

* **Ensuring a certain number of pods are running**
* **Restarting failed containers**
* **Managing updates**
* **Scaling applications**

**Types of Built-in Controllers**

1. **ReplicaSet Controller – Ensures a specific number of Pod replicas are running.**
2. **Job Controller – Ensures Pods complete a specific task to completion.**
3. **DaemonSet Controller – Ensures a copy of a Pod runs on all or some Nodes.**
4. **StatefulSet Controller – Manages stateful applications (with persistent identity).**
5. **CronJob Controller – Manages time-based jobs.**

**Step 1: Define the ReplicaSet Controller**

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

**This YAML defines a Deployment that manages 3 replicas of an NGINX web server.**

**# nginx-deployment.yaml**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: nginx-deployment**

**spec:**

**replicas: 3**

**selector:**

**matchLabels:**

**app: nginx**

**template:**

**metadata:**

**labels:**

**app: nginx**

**spec:**

**containers:**

**- name: my-nginx**

**image: nginx:1.25**

**ports:**

**- containerPort: 80**

**Apply the YAML to Create the Deployment**

**kubectl apply -f nginx-deployment.yaml**

**Check the Deployment Status**

**kubectl get deployments**

**You’ll see something like:**

**NAME READY UP-TO-DATE AVAILABLE AGE**

**nginx-deployment 3/3 3 3 10s**

**This means the Deployment controller created 3 Pods via a ReplicaSet.**

**Verify the Pods**

**kubectl get pods -l app=nginx**

**Should return 3 pods like:**

**nginx-deployment-xxxxx Running**

**nginx-deployment-yyyyy Running**

**nginx-deployment-zzzzz Running**

**Step 2 🡪 Define StatefulSet Controller**

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

**StatefulSet Controller manages stateful applications — those needing stable identities, persistent storage, and ordered deployment.**

* **Each pod gets a unique name and persistent volume.**
* **Useful for databases (like MySQL, Cassandra, etc.)**

**Example YAML:**

**apiVersion: apps/v1**

**kind: StatefulSet**

**metadata:**

**name: web**

**spec:**

**serviceName: "nginx"**

**replicas: 3**

**selector:**

**matchLabels:**

**app: nginx**

**template:**

**metadata:**

**labels:**

**app: nginx**

**spec:**

**containers:**

**- name: nginx**

**image: nginx**

**volumeMounts:**

**- name: www**

**mountPath: /usr/share/nginx/html**

**volumeClaimTemplates:**

**- metadata:**

**name: www**

**spec:**

**accessModes: [ "ReadWriteOnce" ]**

**resources:**

**requests:**

**storage: 1Gi**

**Use-case: Databases, message queues, or anything needing stable storage and identity.**

**Step 3 🡪 DaemonSet Controller**

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

**Ensures that a copy of a Pod runs on every / selected Node in the cluster.**

* **Commonly use for logging, monitoring, or network agents.**

**Example YAML:**

**apiVersion: apps/v1**

**kind: DaemonSet**

**metadata:**

**name: node-exporter**

**spec:**

**selector:**

**matchLabels:**

**name: node-exporter**

**template:**

**metadata:**

**labels:**

**name: node-exporter**

**spec:**

**containers:**

**- name: node-exporter**

**image: prom/node-exporter**

**Use-case: Run agents like Fluentd, Prometheus Node Exporter, etc.**

**Step 4 🡪 Job Controller**

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

**Creates Pods that run to completion (once, not forever).**

* **Perfect for batch jobs or one-time tasks.**
* **Retries if Pods fail.**

**Example YAML:**

**apiVersion: batch/v1**

**kind: Job**

**metadata:**

**name: hello-job**

**spec:**

**template:**

**spec:**

**containers:**

**- name: hello**

**image: busybox**

**command: ["echo", "Hello from Job!"]**

**restartPolicy: Never**

**backoffLimit: 4**

**Use-case: One-off tasks like database migrations, data processing**

**Step 5 🡪 CronJob Controller**

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

**Runs Jobs on a schedule, like a Linux cron job.**

* **Can manage retries and failed runs.**
* **Syntax follows standard cron format.**

**Example YAML:**

**apiVersion: batch/v1**

**kind: CronJob**

**metadata:**

**name: hello-cron**

**spec:**

**schedule: "\*/1 \* \* \* \*" # Every minute**

**jobTemplate:**

**spec:**

**template:**

**spec:**

**containers:**

**- name: hello**

**image: busybox**

**command: ["echo", "Hello from CronJob!"]**

**restartPolicy: OnFailure**

**Use-case: Periodic backups, reporting, monitoring jobs.**

**8. How do you secure a Kubernetes cluster?**

**====================================================**

**Answer:**

* **Use RBAC (Role-Based Access Control) for permission management.**
* **Enable network policies to control pod-to-pod communication.**
* **Use TLS for communication between components.**
* **Use image scanning and trusted registries.**
* **Enable Pod Security Policies or Pod Security Admission for runtime restrictions.**
* **Encrypt secrets at rest.**
* **Regularly patch and update cluster components.**

**Step 1. Use Role-Based Access Control (RBAC)**

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

**RBAC use to limit who can perform which action in your cluster. For that we need to create RBAC role and bind it to the user.**

**Create a role that allows reading pods:**

**apiVersion: rbac.authorization.k8s.io/v1**

**kind: Role**

**metadata:**

**namespace: default**

**name: pod-reader**

**rules:**

**- apiGroups: [""]**

**resources: ["pods" , "services"]**

**verbs: ["get", "watch", "list"]**

**Bind role to a user:**

**apiVersion: rbac.authorization.k8s.io/v1**

**kind: RoleBinding**

**metadata:**

**name: read-pods-binding**

**namespace: default**

**subjects:**

**- kind: User**

**name: dev-group**

**apiGroup: rbac.authorization.k8s.io**

**roleRef:**

**kind: Role**

**name: pod-reader**

**apiGroup: rbac.authorization.k8s.io**

**Apply: kubectl apply -f role.yaml**

**kubectl apply -f rolebinding.yaml**

**Enable Authentication and Authorization using client certificates, OIDC, or cloud provider IAM.**

** Use OIDC with Google or Azure AD to authenticate users.**

** Configure --authentication-mode=Webhook and --authorization-mode=RBAC in the API server.**

**Step 2 🡪 Use Network Policies**

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

**Control traffic between pods and namespaces, by enabling a network plugin that supports policies (e.g., Calico, Cilium)**

**kubectl apply -f** [**https://docs.projectcalico.org/manifests/calico.yaml**](https://docs.projectcalico.org/manifests/calico.yaml) **🡪 Download and apply Calico manifests**

**kubectl get pods -n calico-system 🡪 Verify Calico pods are running**

**Block all traffic except from a specific app:**

**apiVersion: networking.k8s.io/v1**

**kind: NetworkPolicy**

**metadata:**

**name: allow-nginx**

**namespace: default**

**spec:**

**podSelector:**

**matchLabels:**

**app: nginx**

**ingress:**

**- from:**

**- podSelector:**

**matchLabels:**

**app: frontend**

**Step 3 🡪 Use TLS for communication between components.**

**As etcd contains all cluster secrets and configurations.**

** Encrypt etcd at rest.**

** Use TLS for etcd client and peer connections.**

** Limit access to etcd to the API server only.**

**Vi /etc/kubernetes/encryption-config.yaml**

**apiVersion: apiserver.config.k8s.io/v1**

**kind: EncryptionConfiguration**

**resources:**

**- resources:**

**- secrets**

**providers:**

**- aescbc:**

**keys:**

**- name: key1**

**secret: <base64-encoded-secret>**

**- identity: {}**

**Edit /etc/kubernetes/manifests/kube-apiserver.yaml file and add the --encryption-provider-config /etc/kubernetes/encryption-config.yaml**

**Restart the kube-apiserver if needed**

**Step 4 🡪 Scan Container Images**

**Use tools like Trivy, Clair, or Aqua Security scan the container image**

**Ex 🡪 trivy image nginx:latest**

**9. What strategies are available for deploying applications in Kubernetes?**

**Answer:**

* **Rolling Update: Incrementally updates pods with zero downtime.**
* **Blue/Green Deployment: Runs two environments and switches traffic from old to new.**
* **Canary Deployment: Gradually rolls out new versions to a subset of users.**
* **Recreate: Shuts down old version before starting the new one.**

**Rolling update 🡪 explained at above**

**Blue/green deployment 🡪**

**In case of blue/green deployment the downtime and risk are reduces by running two production environments (Blue and Green) in parallel. Once test is success we can delete old version.**

* + - **Lets think the Current version is running in the Blue environment.**
    - **Green is a new version of your app, deployed alongside Blue.**
    - **Tests/health checks are run against Green.**
    - **Traffic is switched from Blue to Green via Ingress or LoadBalancer.**
    - **Once confirmed, Blue is optionally deleted or kept for rollback.**

**To perform blue/green deployment there should be two deployment manifest files and a selector base service manifest file.**

**Vi deployment-blue.yaml**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: myapp-blue**

**spec:**

**replicas: 3**

**selector:**

**matchLabels:**

**app: myapp**

**version: blue**

**template:**

**metadata:**

**labels:**

**app: myapp**

**version: blue**

**spec:**

**containers:**

**- name: myapp**

**image: myapp:1.0**

**ports:**

**- containerPort: 80**

**Vi deployment-green.yaml**

**Same as above , but the version: green and image: new\_image\_with\_new\_version as below.**

**metadata:**

**name: myapp-green**

**...**

**version: green**

**image: myapp:2.0**

**vi selector\_base\_service.yaml**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: myapp-service**

**spec:**

**selector:**

**app: myapp**

**version: blue # or green (to be updated)**

**ports:**

**- protocol: TCP**

**port: 80**

**targetPort: 80**

**vi blue\_green\_deploy.yml ------------ it is in Ansible**

**- name: Blue-Green Deployment on Kubernetes**

**hosts: localhost**

**gather\_facts: no**

**vars:**

**new\_version: green # or blue**

**old\_version: blue # or green**

**image\_tag: "2.0"**

**tasks:**

**- name: Apply new version deployment**

**kubernetes.core.k8s:**

**state: present**

**definition: "{{ lookup('file', 'deployment-' + new\_version + '.yaml') }}"**

**- name: Wait for rollout to complete**

**command: >**

**kubectl rollout status deployment/myapp-{{ new\_version }}**

**register: rollout\_status**

**until: rollout\_status.stdout.find('successfully rolled out') != -1**

**retries: 10**

**delay: 10**

**- name: Run health checks (optional)**

**uri:**

**url: "http://{{ new\_version }}.myapp.internal/health"**

**status\_code: 200**

**register: health\_result**

**until: health\_result.status == 200**

**retries: 5**

**delay: 5**

**- name: Switch service to new version**

**kubernetes.core.k8s:**

**state: present**

**definition:**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: myapp-service**

**spec:**

**selector:**

**app: myapp**

**version: "{{ new\_version }}"**

**ports:**

**- protocol: TCP**

**port: 80**

**targetPort: 80**

**- name: (Optional) Delete old version deployment**

**kubernetes.core.k8s:**

**state: absent**

**kind: Deployment**

**name: myapp-{{ old\_version }}**

**Canary Deployment 🡪 Canary deployment is a process to roll out a new version of an application to a small subset of users before exposing it to the entire user base. This lets you test the new version in production with real traffic, minimizing risk by limiting exposure to potential issues.**

**ansi-canary-folder/**

**├── canary-deployment.yaml**

**├── stable-deployment.yaml**

**├── service.yaml**

**└── canary-playbook.yaml**

**For Canary deployment we need to follow below steps**

**Existing app version (stable).**

**New app version (canary) deployed alongside stable.**

**Traffic is split between stable and canary pods, e.g., 90% stable, 10% canary.**

**You monitor canary behavior.**

**If successful, shift 100% traffic to new version and remove old version.**

**stable-deployment.yaml**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: myapp-stable**

**spec:**

**replicas: 9**

**selector:**

**matchLabels:**

**app: myapp**

**version: stable**

**template:**

**metadata:**

**labels:**

**app: myapp**

**version: stable**

**spec:**

**containers:**

**- name: myapp**

**image: mydockerhub/myapp:v1**

**ports:**

**- containerPort: 80**

**canary-deployment.yaml**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: myapp-canary**

**spec:**

**replicas: 1**

**selector:**

**matchLabels:**

**app: myapp**

**version: canary**

**template:**

**metadata:**

**labels:**

**app: myapp**

**version: canary**

**spec:**

**containers:**

**- name: myapp**

**image: mydockerhub/myapp:v2**

**ports:**

**- containerPort: 80**

**service.yaml**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: myapp-service**

**spec:**

**selector:**

**app: myapp**

**ports:**

**- protocol: TCP**

**port: 80**

**targetPort: 80**

**type: LoadBalancer**

**Ansible Playbook - canary-playbook.yaml**

**- name: Canary Deployment on Kubernetes EKS**

**hosts: localhost**

**gather\_facts: no**

**tasks:**

**- name: Deploy Stable version**

**k8s:**

**state: present**

**definition: "{{ lookup('file', 'stable-deployment.yaml') }}"**

**- name: Deploy Canary version**

**k8s:**

**state: present**

**definition: "{{ lookup('file', 'canary-deployment.yaml') }}"**

**- name: Deploy Service exposing both versions**

**k8s:**

**state: present**

**definition: "{{ lookup('file', 'service.yaml') }}"**

**- name: Wait for stable deployment rollout to complete**

**k8s\_info:**

**kind: Deployment**

**namespace: default**

**name: myapp-stable**

**register: stable\_deploy**

**- name: Wait until stable replicas ready**

**k8s\_info:**

**kind: Pod**

**namespace: default**

**label\_selectors:**

**- "app=myapp,version=stable"**

**register: stable\_pods**

**until: stable\_pods.resources | selectattr('status.phase','equalto','Running') | list | length == 9**

**retries: 10**

**delay: 15**

**- name: Wait until canary pods ready**

**k8s\_info:**

**kind: Pod**

**namespace: default**

**label\_selectors:**

**- "app=myapp,version=canary"**

**register: canary\_pods**

**until: canary\_pods.resources | selectattr('status.phase','equalto','Running') | list | length == 1**

**retries: 10**

**delay: 15**

**Once real time testing complete by users then scaling Canary Up/Stable Down (Promoting Canary) using Ansible playbook**

**- name: Promote canary to stable by scaling**

**hosts: localhost**

**gather\_facts: no**

**tasks:**

**- name: Scale stable deployment down**

**k8s:**

**kind: Deployment**

**name: myapp-stable**

**namespace: default**

**replicas: 0**

**- name: Scale canary deployment up**

**k8s:**

**kind: Deployment**

**name: myapp-canary**

**namespace: default**

**replicas: 10**

**- name: Update canary label to stable (optional)**

**k8s:**

**kind: Deployment**

**name: myapp-canary**

**namespace: default**

**definition:**

**spec:**

**template:**

**metadata:**

**labels:**

**version: stable**

**10. How do you monitor a Kubernetes cluster?**

* **Use tools like Prometheus (metrics collection), Grafana (visualization).**
* **Use ELK/EFK stacks (Elasticsearch, Fluentd/Fluent Bit, Kibana) for logs.**
* **Kubernetes Dashboard or Lens IDE for cluster state.**
* **Use probes (liveness/readiness) for pod health.**

**11. Explain how you would troubleshoot a failing pod?**

* **Check pod status:**

**kubectl get pods and kubectl describe pod <pod-name>.**

* **Check pod logs:**

**kubectl logs <pod-name>**

**kubectl logs myapp-pod -c <container-name> 🡪 If multiple containers**

* **Inspect events for failures:**

**kubectl get events --sort-by='.lastTimestamp'**

* **Check container runtime status on the node ( MemoryPressure and DiskPressure will be True)**

**kubectl describe node nodeName**

* **Validate resource limits/requests and node capacity.**

**kubectl get pod <pod-name> -o yaml**

**ex🡪**

**resources:**

**requests:**

**cpu: "4"**

**memory: "8Gi"**

**If your node only has 2 CPUs, this pod will remain in Pending state.**

* **Check network connectivity and service endpoints.**

**kubectl get svc**

**kubectl get endpoints -------> it will give ip and port use that port in blow command**

**kubectl exec -it <pod-name> -- curl <service>:<port>**

**12. How do Kubernetes and Docker complement each other?**

**Answer:**

* **Docker packages apps into containers.**
* **Kubernetes orchestrates and manages container lifecycle, networking, scaling, and updates.**
* **Docker is the runtime used in many Kubernetes clusters (though alternatives exist).**

**13. What is Helm and why would you use it?**

**Answer:**

* **Helm is a package manager for Kubernetes, simplifying app deployment and management.**
* **It uses Charts to define, install, and upgrade Kubernetes applications.**
* **It helps manage complex apps and their dependencies declaratively.**

**14. What is a ConfigMap and Secret? How do you use them?**

**Answer:**

* **ConfigMap: Used to store non-sensitive configuration data as key-value pairs.**
* **Secret: Used to store sensitive data like passwords, tokens, or keys, base64-encoded.**
* **They can be consumed by Pods as environment variables or mounted as files.**

**Create configmap from yaml file**

**Vi testConfigMap.yaml**

**apiVersion: v1**

**kind: ConfigMap**

**metadata:**

**name: my-config**

**data:**

**APP\_ENV: "production"**

**APP\_DEBUG: "false"**

**DATABASE\_URL: "mysql://db:3306"**

**kubectl apply -f testConfigMap.yaml**

**Use this ConfigMap in a Pod as environment variables**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: configmap-demo**

**spec:**

**containers:**

**- name: app**

**image: myapp:latest**

**envFrom:**

**- configMapRef:**

**name: my-config**

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

**First create the encrypted data**

**echo –n ‘admin’ | base64**

**o/p 🡪 YWRtaW4**

**echo –n ‘suman@#345’ | base64**

**o/p 🡪 MWETY345DAV45AK**

**Create Secret from yaml file**

**Vi mySecretFile.yaml**

**apiVersion: v1**

**kind: Secret**

**metadata:**

**name: my-secret**

**type: Opaque**

**data:**

**DB\_PASSWORD: MWETY345DAV45AK**

**Use this Secret in a Pod as environment variables**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: secret-demo**

**spec:**

**containers:**

**- name: app**

**image: myapp:latest**

**env:**

**- name: DB\_PASSWORD**

**valueFrom:**

**secretKeyRef:**

**name: my-secret**

**key: DB\_PASSWORD**

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

**Create a property file**

**cat /etc/config/app.properties**

**APP\_ENV=production**

**APP\_DEBUG=false**

**DATABASE\_URL=mysql://db:3306**

**Create a Configmap yaml**

**Vi my\_configmap.yaml**

**apiVersion: v1**

**kind: ConfigMap**

**metadata:**

**name: app-config**

**data:**

**app.properties: |**

**APP\_ENV=production**

**APP\_DEBUG=false**

**DATABASE\_URL=mysql://db:3306**

**Define the Pod that mounts the ConfigMap as a volume**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: configmap-volume-pod**

**spec:**

**containers:**

**- name: app-container**

**image: busybox**

**command: [ "sleep", "3600" ]**

**volumeMounts:**

**- name: config-volume**

**mountPath: /etc/config # Files from ConfigMap(app.properties) is available here**

**volumes:**

**- name: config-volume**

**configMap:**

**name: app-config # Refers to the ConfigMap created earlier**

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

**First create the encrypted data**

**echo –n ‘admin’ | base64**

**o/p 🡪 YWRtaW4**

**echo –n ‘suman@#345’ | base64**

**o/p 🡪 MWETY345DAV45AK**

**Now create the manifest file**

**Vi mySecrete.yaml**

**apiVersion: v1**

**kind: Secret**

**metadata:**

**name: mysecret**

**type: Opaque**

**data:**

**username: YWRtaW4 🡪 Here username is the key and encrypted data is value**

**password: MWETY345DAV48AK 🡪 password is key and encrypted data is value**

**create pod manifest file**

**apiVersion: v1**

**kind: Pod**

**metadata:**

**name: secret-volume-pod**

**spec:**

**containers:**

**- name: secret-container**

**image: busybox**

**command: [ "sleep", "3600" ]**

**volumeMounts:**

**- name: secret-volume**

**mountPath: "/etc/secret-data"**

**readOnly: true**

**volumes:**

**- name: secret-volume**

**secret:**

**secretName: my-secret**

**Kubectl create -f mySecretPod.yaml**

**Kubectl exec secret-volume-pod ls /etc/secret-data 🡪 two files username , password will be there**

**15. Deploy NGINX App with Helm?**

**Step 1 🡪 Create the folder structure / Skeleton for Helm Chart**

**helm create my-nginx-test 🡪 it will create below folder structure**

**my-nginx-test/**

**Chart.yaml**

**values.yaml**

**templates/**

**deployment.yaml**

**service.yaml**

**\_helpers.tpl**

**Step 2 🡪 create values.yaml**

**replicaCount: 2**

**image:**

**repository: nginx**

**tag: latest**

**pullPolicy: IfNotPresent**

**service:**

**type: ClusterIP**

**port: 80**

**resources: {}**

**nodeSelector: {}**

**tolerations: []**

**affinity: {}**

**=========================================================================**

**my-nginx/ # Github repo**

**├── .github/**

**├── helm/**

**│ └── my-nginx-test/ # Helm chart folder**

**│ ├── Chart.yaml # Chart metadata**

**│ ├── values.yaml # Default configuration values**

**│ ├── templates/ # Kubernetes YAML templates**

**│ │ ├── deployment.yaml**

**│ │ ├── service.yaml**

**│ │ └── \_helpers.tpl**

**│ └── README.md # Chart-specific README**

**├── nginx.conf # NGINX configuration file**

**├── Dockerfile # To build custom NGINX image (optional)**

**├── README.md # Project overview and instructions**

**├── LICENSE # License file (e.g., MIT)**

**└── .dockerignore # Ignore files for Docker build**

**=========================================================================**

**Step 3 🡪 \_helpers.tpl is the file use to store reusable template helpers**

**{{/\* Generate chart name \*/}}**

**{{- define "my-nginx-test.name" -}}**

**{{- .Chart.Name -}}**

**{{- end }}**

**{{/\* Generate full resource name \*/}}**

**{{- define "my-nginx-test.fullname" -}}**

**{{- printf "%s-%s" .Release.Name .Chart.Name | trunc 63 | trimSuffix "-" -}}**

**{{- end }}**

**{{/\* Common labels for resources \*/}}**

**{{- define "my-nginx-test.labels" -}}**

**app.kubernetes.io/name: {{ include "my-nginx-test.name" . }}**

**app.kubernetes.io/instance: {{ .Release.Name }}**

**app.kubernetes.io/version: {{ .Chart.AppVersion }}**

**helm.sh/chart: {{ .Chart.Name }}-{{ .Chart.Version }}**

**{{- end }}**

**Step 4 🡪 deployment.yaml which manages pods, specifying things like replicas, container images, ports, and labels**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: {{ include "my-nginx-test.fullname" . }}**

**labels:**

**{{ include "my-nginx-test.labels" . | indent 4 }}**

**spec:**

**replicas: {{ .Values.replicaCount }}**

**selector:**

**matchLabels:**

**app.kubernetes.io/name: {{ include "my-nginx-test.name" . }}**

**app.kubernetes.io/instance: {{ .Release.Name }}**

**template:**

**metadata:**

**labels:**

**app.kubernetes.io/name: {{ include "my-nginx-test.name" . }}**

**app.kubernetes.io/instance: {{ .Release.Name }}**

**spec:**

**containers:**

**- name: nginx**

**image: "{{ .Values.image.repository }}:{{ .Values.image.tag }}"**

**imagePullPolicy: {{ .Values.image.pullPolicy }}**

**ports:**

**- containerPort: {{ .Values.service.port }}**

**livenessProbe:**

**httpGet:**

**path: /**

**port: {{ .Values.service.port }}**

**readinessProbe:**

**httpGet:**

**path: /**

**port: {{ .Values.service.port }}**

**resources:**

**{{- toYaml .Values.resources | nindent 12 }}**

**Step 5 🡪 service.yaml exposes your pods to other services or the outside world.**

**apiVersion: v1**

**kind: Service**

**metadata:**

**name: {{ include "my-nginx-test.fullname" . }}**

**labels:**

**app: {{ include "my-nginx-test.name" . }}**

**spec:**

**type: {{ .Values.service.type }}**

**ports:**

**- port: {{ .Values.service.port }}**

**targetPort: {{ .Values.service.port }}**

**protocol: TCP**

**name: http**

**selector:**

**app.kubernetes.io/name: {{ include "my-nginx-test.name" . }}**

**app.kubernetes.io/instance: {{ .Release.Name }}**

**Step 6 🡪 create Chart.yaml**

**apiVersion: v2 # Helm chart API version (v2 is for Helm 3+)**

**name: my-nginx-test # Name of the chart**

**description: A simple NGINX web server**

**type: application # Can be "application" or "library"**

**version: 1.0.0 # Chart version (used for Helm chart packages)**

**appVersion: "1.21.6" # Version of app being deployed (e.g. NGINX version)**

**keywords:**

**- nginx**

**- web**

**- http**

**maintainers:**

**- name: sumanta**

**email: sk@example.com**

**sources:**

**- https://github.com/example/my-nginx**

**home: https://nginx.org**

**Step 7 🡪 Install Your Chart**

**helm install nginx-app ./my-nginx-test-proj**

**Step 8 🡪 update replicaCount value to 3 in values.yaml we can upgrade using helm**

**helm upgrade nginx-app ./ my-nginx-test-proj**

**16. Explain Horizontal Pod Autoscaling (HPA).**

**Answer:**

* **HPA automatically scales the number of pods based on CPU/memory usage or custom metrics.**
* **It queries the Metrics Server for resource usage.**
* **Configuration involves defining min/max pod count and target utilization.**

**Example**

**Step 1: Deploy NGINX using deployment manifest file**

**apiVersion: apps/v1**

**kind: Deployment**

**metadata:**

**name: nginx-deployment**

**spec:**

**replicas: 1**

**selector:**

**matchLabels:**

**app: nginx**

**template:**

**metadata:**

**labels:**

**app: nginx**

**spec:**

**containers:**

**- name: nginx**

**image: nginx**

**resources:**

**requests:**

**cpu: 100m**

**limits:**

**cpu: 500m**

**Step 2 : create the HPA (Horizontal Pod Autoscaling) manifest file**

**apiVersion: autoscaling/v2**

**kind: HorizontalPodAutoscaler**

**metadata:**

**name: nginx-hpa**

**spec:**

**scaleTargetRef:**

**apiVersion: apps/v1**

**kind: Deployment**

**name: nginx-deployment ------------ it look for deployment manifest file**

**minReplicas: 1**

**maxReplicas: 5**

**metrics:**

**- type: Resource**

**resource:**

**name: cpu**

**target:**

**type: Utilization**

**averageUtilization: 50**

**kubectl apply -f nginx-deployment.yaml**

**kubectl apply -f nginx-hpa.yaml**

**kubectl get hpa nginx-hpa 🡪 Check HPA status**

**Generate load on NGINX to increase CPU usage (for example using kubectl exec to run a stress command inside a pod or an external load generator). The HPA will increase pod count automatically.**