ChatGPT

Cluster Architecture, Installation & Configuration

Question 1: Create ClusterRole, ServiceAccount and Binding

- **Context:** You have an existing Kubernetes cluster. You need to create a ClusterRole for a deployment pipeline and bind it to a ServiceAccount in a specific namespace.
- **Task:** Create a new ClusterRole named `deployment-clusterrole` that allows **only** creation of Deployments, StatefulSets, and DaemonSets. Create a ServiceAccount `cicd-token` in namespace `app-team1`. Bind the ClusterRole to the ServiceAccount (scoped to `app-team1`).
- **Solution:** Use `kubectl create` commands or YAML manifests. For example:

```
# Create the ClusterRole with limited verbs
kubectl create clusterrole deployment-clusterrole \
  --resource=deployments,statefulsets,daemonsets \
  --verb=create

# Create the ServiceAccount in app-team1 namespace
kubectl create sa cicd-token -n app-team1

# Bind the ClusterRole to the ServiceAccount (namespace-scoped)
kubectl create clusterrolebinding cicd-deploy-binding \
  --clusterrole=deployment-clusterrole \
  --serviceaccount=app-team1:cicd-token
```

  This ensures `cicd-token` in `app-team1` can create Deployments, StatefulSets, and DaemonSets only.
- **Commands & Tips:** You can also write these as YAML (ClusterRole, ServiceAccount, ClusterRoleBinding) and apply with `kubectl apply -f`. Verify with `kubectl get clusterrole deployment-clusterrole` and `kubectl describe clusterrolebinding cicd-deploy-binding`. Ensure you use the correct namespace when creating the ServiceAccount and in the binding (format `<namespace>:<name>`). No additional verbs (like list/get) are needed here since only creation is required.

Question 3: Upgrade Master Node to v1.22.2

- **Context:** An existing cluster's control plane runs version 1.20.0. You must upgrade all control-plane components on the master node only, to version 1.22.2.
- **Task:** Drain the master node, perform the upgrade of the API server, controller-manager, scheduler, kubelet, and kubectl, then uncordon the node. Do not upgrade any worker nodes or addons.
- **Solution:** Follow the kubeadm upgrade process. First drain and cordon the master:

```
kubectl drain master-node-0 --ignore-daemonsets
```

Then upgrade kubeadm and apply the new version:

```
sudo apt-get update
sudo apt-get install -y kubeadm=1.22.2-00
sudo kubeadm upgrade plan v1.22.2
sudo kubeadm upgrade apply v1.22.2
```

After control-plane is upgraded, update kubelet and kubectl:

```
sudo apt-mark unhold kubelet kubectl
sudo apt-get update
sudo apt-get install -y kubelet=1.22.2-00 kubectl=1.22.2-00
sudo apt-mark hold kubelet kubectl
```

Then reload and restart kubelet:

```
sudo systemctl daemon-reload
sudo systemctl restart kubelet
```

Finally, uncordon the node:

```
kubectl uncordon master-node-0
```

This brings the master back to schedulable state [1].

- **Commands & Tips:** The sequence is important: **drain** before upgrade and **uncordon** after (as shown in the official docs [2] [1]). Use `kubectl get nodes` to verify the master returns to `Ready`. If `kubeadm upgrade apply` fails, it is idempotent – re-running it can recover. For detailed steps see the kubeadm upgrade guide [2] [1]. Ensure your package versions match the target (1.22.2) exactly, including the `-00` suffix on Debian/Ubuntu.

Question 4: Restore etcd from a Snapshot

- **Context:** The control plane's etcd data needs recovery. A previous snapshot file is available at `/var/lib/backup/etcd-snapshot-previous.db`.
- **Task:** Restore the etcd snapshot to recover cluster state.
- **Solution:** Stop the kube-apiserver (so etcd is not in use), then use `etcdctl snapshot restore` to populate a new data directory, reconfigure etcd to use it, and restart services. For example:

```
# Stop API server (if run as static pod, remove or rename /etc/kubernetes/
manifests/kube-apiserver.yaml)
mv /etc/kubernetes/manifests/kube-apiserver.yaml /root/
```

```
# Restore snapshot into new data-dir
ETCDCTL_API=3 etcdctl snapshot restore /var/lib/backup/etcd-snapshot-
previous.db \
  --data-dir /var/lib/etcd-new

# Update etcd manifest to point to /var/lib/etcd-new (or move data)
mv /var/lib/etcd /var/lib/etcd-old
mv /var/lib/etcd-new /var/lib/etcd

# Restart API server (move the manifest back)
mv /root/kube-apiserver.yaml /etc/kubernetes/manifests/

# The kubelet will restart etcd and API server pods automatically.
```

This restores etcd from the snapshot [3]. Then verify cluster health with `kubectl get nodes`.

- **Commands & Tips:** Ensure the new `--data-dir` is not the one used by the running etcd (use a temporary directory). The KodeKloud notes explain that after restoring to a new dir you must update the etcd data directory and restart etcd/API server [3]. Always supply cert flags if your etcd is secured. If using static pods, moving or editing the manifest will cause kubelet to launch etcd with the restored data.

Workloads & Scheduling

Question 2: Drain and Cord on a Node

- **Context:** There's a node named `ek8s-node-0` in the cluster.
- **Task:** Make `ek8s-node-0` unschedulable and reschedule its pods elsewhere (simulate maintenance).
- **Solution:** Use `kubectl drain` to cordon and evict pods in one command:

```
kubectl drain ek8s-node-0 --ignore-daemonsets
```

This marks the node unschedulable and evicts pods (except DaemonSet-managed) [4] [2]. If you needed the node back online later, run `kubectl uncordon ek8s-node-0` [5].

- **Commands & Tips:** Using `--ignore-daemonsets` avoids blocking on daemonset pods. The `drain` command will wait for graceful pod termination before completing [6]. If pods with emptyDir need deletion, add `--delete-emptydir-data`. Always verify pods have moved (`kubectl get pods -o wide`).

Question 7: Scale Deployment to 3 Replicas

- **Context:** A Deployment named `presentation` exists.
- **Task:** Scale the `presentation` Deployment to 3 replicas.
- **Solution:** Use `kubectl scale`. For example:

```
kubectl scale --replicas=3 deployment/presentation
```

This sets `.spec.replicas=3` on the Deployment. Confirm with `kubectl get deployment presentation`.

- **Commands & Tips:** The `kubectl scale` command works on deployments, etc. (e.g. `kubectl scale --replicas=3 deployment/my-deploy` [7]). You can also `kubectl edit deployment/presentation` and change `spec.replicas: 3`. After scaling, verify with `kubectl get pods` to see 3 pods running.

Question 8: Pod on SSD Node

- **Context:** You need to create a Pod that should run on a node labeled `disk=ssd`.
- **Task:** Schedule a pod with Name `nginx-kusc00401`, using image `nginx`, and constrain it to nodes with label `disk=ssd`.
- **Solution:** Define a Pod manifest with a `nodeSelector`. For example:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-kusc00401
spec:
  containers:
  - name: nginx
    image: nginx
  nodeSelector:
    disk: ssd
```

Apply it with `kubectl apply -f`. This ensures the Pod only schedules on a node with `disk=ssd` [8] .

- **Commands & Tips:** The `nodeSelector` field ties Pods to nodes by label [8] . You could also label a node yourself (`kubectl label node nodename disk=ssd`). If no node has that label, the Pod will remain Pending. Use `kubectl get pods -o wide` to see which node it runs on. (Affinity could also be used for more complex rules.)

Question 9: Multi-Container Pod

- **Context:** You need a Pod with two containers.
- **Task:** Schedule a Pod named `kucc8` with 2 application containers: one running `nginx` and the other `consul`.
- **Solution:** Write a Pod YAML with both container specs. For example:

```
apiVersion: v1
kind: Pod
metadata:
  name: kucc8
spec:
  containers:
  - name: nginx
    image: nginx
  - name: consul
    image: consul
```

Apply it with `kubectl apply -f`. This creates one Pod with both `nginx` and `consul` containers.

- **Commands & Tips:** Ensure both container names are unique. Verify with `kubectl get pod kucc8 -o yaml` or `kubectl describe pod kucc8` to see both containers. No special scheduling constraint is needed beyond the standard scheduling.

Question 20: Pod with Redis and Consul

- **Context:** (Similar to Q9.) Schedule a Pod named `kucc1`.
- **Task:** Create a Pod `kucc1` with 2 containers: one with image `redis` and one with `consul`.
- **Solution:** Use a Pod spec like:

```
apiVersion: v1
kind: Pod
metadata:
  name: kucc1
spec:
  containers:
  - name: redis
    image: redis
  - name: consul
    image: consul
```

Then `kubectl apply -f` this manifest.

- **Commands & Tips:** (Same as Q9) Check `kubectl get pods -l run=kucc1` to see the Pod. Make sure container names (`redis`, `consul`) are valid identifiers.

Question 21: Pod on Spinning Disk Node

- **Context:** Place a Pod on nodes labeled `disk=spinning`.
- **Task:** Schedule a pod named `nginx-kusc00401` (image `nginx`) with `nodeSelector: disk=spinning`.
- **Solution:** Similar to Q8. Pod YAML:

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-kusc00401
spec:
  containers:
  - name: nginx
    image: nginx
  nodeSelector:
    disk: spinning
```

Apply with `kubectl apply -f`. The Pod will only schedule on a node with label `disk=spinning`.
- **Commands & Tips:** If no such node exists, the Pod stays Pending. The nodeSelector ensures placement constraints [8]. Use `kubectl get nodes --show-labels` to verify node labels.

Question 22: Scale Deployment "guestbook" to 5 Replicas

- **Context:** A Deployment named `guestbook` exists.
- **Task:** Scale `guestbook` to 5 pods.
- **Solution:** Use `kubectl scale`:

```
kubectl scale --replicas=5 deployment/guestbook
```

This sets `.spec.replicas=5`. Check with `kubectl get deployment guestbook`.
- **Commands & Tips:** (Same method as Q7.) Confirm 5 pods with `kubectl get pods`. If the Deployment has a label selector, you will see 5 pods matching that selector. Use the syntax shown in the kubectl reference [7].

Services & Networking

Question 5: NetworkPolicy to Allow Port-Restricted Traffic

- **Context:** In namespace `fubar`, we want to allow only specific traffic.
- **Task:** Create a NetworkPolicy `allow-port-from-namespace` in `fubar` namespace so that **only** Pods in namespace `internal` can connect to port 9000 of Pods in `fubar`. All other ingress (or other ports) should be denied.
- **Solution:** Example NetworkPolicy YAML:

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-port-from-namespace
  namespace: fubar
spec:
  podSelector: {}        # applies to all pods in fubar
  policyTypes:
```

```
  - Ingress
  ingress:
  - from:
    - namespaceSelector:
        matchLabels:
          name: internal     # assuming 'internal' ns is labeled
name=internal
    ports:
    - protocol: TCP
      port: 9000
```

This policy only allows ingress on port 9000 from Pods in namespaces labeled `name=internal`. All other traffic is implicitly denied.

- **Commands & Tips:** Replace the namespace selector label with an actual label on the `internal` namespace (e.g., `kubectl label ns internal name=internal`). Ensure the NetworkPolicy's `namespaceSelector` matches `internal`'s label. No explicit deny rule is needed; Kubernetes defaults to deny for anything not allowed. Verify with `kubectl get networkpolicy -n fubar`.

Question 6: Create Service with NodePort

- **Context:** There is an existing Deployment `front-end` with container port `80` named `http`.
- **Task:** Create a Service named `front-end` that exposes the deployment's container port `http` (port 80) and also assigns a `NodePort` so the pods are accessible via node's IP.
- **Solution:** For example:

```
# Create a headless or ClusterIP Service exposing port 80
kubectl create service clusterip front-end --tcp=80:80

# Alternatively, use YAML:
```

Or better, YAML:

```
apiVersion: v1
kind: Service
metadata:
  name: front-end
spec:
  type: NodePort
  selector:
    app: front-end   # label matching the deployment pods
  ports:
  - name: http
    port: 80         # service port
    targetPort: 80   # container port
    nodePort: 30080  # example NodePort (will be in range 30000-32767)
```

Apply with `kubectl apply -f`. This exposes `front-end:80` to outside on one of the node ports (here 30080).

- **Commands & Tips:** You can run `kubectl expose deployment front-end --type=NodePort --port=80` to auto-create a NodePort service. Verify with `kubectl get svc front-end`. To reach a pod, use `curl http://<node-ip>:<NodePort>`. Pick a specific nodePort if required by adding `--node-port=30080`. Make sure the `selector` matches your deployment's labels.

Question 16: Create Ingress "pong" in ing-internal

- **Context:** Namespace `ing-internal` exists. We need an NGINX Ingress.
- **Task:** Create an Ingress named `pong` in namespace `ing-internal` that routes path `/hello` to service `hello` on port `5678`. (When accessed at `/hello`, it should return "hello".)
- **Solution:** First ensure the namespace exists. Then:

```
kubectl create ingress pong -n ing-internal --rule="/hello=hello:5678"
```

This creates an Ingress where requests to `/hello` are forwarded to service `hello`'s port 5678 [9]. Make sure the Ingress controller (nginx) is installed. Test with `curl http://<INGRESS-IP>/hello`.

- **Commands & Tips:** The `kubectl create ingress` shortcut uses the default Ingress class (usually nginx). If it's not supported, you can write a YAML Ingress:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: pong
  namespace: ing-internal
spec:
  rules:
  - http:
      paths:
      - path: /hello
        pathType: Prefix
        backend:
          service:
            name: hello
            port:
              number: 5678
```

Note the pathType and backend format for v1 API. Verify route with `kubectl get ingress -n ing-internal`.

Question 18: Create Ingress "ping" in ing-internal

- **Context:** Similar to Q16.

- **Task:** Create an Ingress named `ping` in namespace `ing-internal` that routes path `/hi` to service `hi` on port `5678`.
- **Solution:** For example:

  ```
  kubectl create ingress ping -n ing-internal --rule="/hi=hi:5678"
  ```

  Or YAML:

  ```
  apiVersion: networking.k8s.io/v1
  kind: Ingress
  metadata:
    name: ping
    namespace: ing-internal
  spec:
    rules:
    - http:
        paths:
        - path: /hi
          pathType: Prefix
          backend:
            service:
              name: hi
              port:
                number: 5678
  ```

  This ensures requests to `/hi` hit the `hi` service on port 5678 [9].
- **Commands & Tips:** As above, ensure the `hi` service exists and an Ingress controller is running. Check the Ingress with `kubectl describe ingress ping -n ing-internal`. Test with `curl http://<INGRESS-IP>/hi`.

Storage

Question 11: Create PV and Add Logging Sidecar

- **Context:** You have a running Pod `big-corp-app` that writes logs to `/var/log/big-corp-app.log`. You need a sidecar to stream the log, and a PersistentVolume for `/srv/app-data`.
- **Task:**
- Create a PersistentVolume named `app-data` of size 2Gi, access mode `ReadWriteOnce`, hostPath `/srv/app-data`.
- Modify the existing Pod `big-corp-app`: add a sidecar container named `sidecar` (image `busybox`) that runs `tail -f /var/log/big-corp-app.log`, using a shared volume mounted at `/var/log`.

- **Solution:**

- PersistentVolume YAML example:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: app-data
spec:
  capacity:
    storage: 2Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: /srv/app-data
```

  Apply with `kubectl apply -f pv-app-data.yaml`.

- To add the sidecar, you can `kubectl edit pod big-corp-app` (or redeploy a Pod). For example, if you have the Pod YAML, update it:

```
spec:
  volumes:
  - name: log-vol
    emptyDir: {}
  containers:
  - name: big-corp-app
    image: your-app-image
    volumeMounts:
    - name: log-vol
      mountPath: /var/log
  - name: sidecar
    image: busybox
    command: ["sh", "-c", "tail -f /var/log/big-corp-app.log"]
    volumeMounts:
    - name: log-vol
      mountPath: /var/log
```

  This uses an `emptyDir` volume (shared in the Pod) so both containers see the same `/var/log`. The sidecar will then print new log lines. Apply changes (you may need to recreate the Pod).

- **Commands & Tips:** Verify the PV with `kubectl get pv app-data`. If using the PV, you may also need a PersistentVolumeClaim depending on your use case. The sidecar pattern (a second container reading logs via a shared volume) is a common logging solution [3]. Ensure the existing container writes logs into the shared path. After modifying, `kubectl get pods` should show both containers running in `big-corp-app`.

Troubleshooting

Question 15: Pod with Highest CPU Usage

- **Context:** Pods are labeled `name=overloaded-cpu`. You need to identify which one uses the most CPU.
- **Task:** Find the pod with label `name=overloaded-cpu` that has the highest CPU usage, and write its name to `/opt/KUBET00401/KUBET00401.txt`.
- **Solution:** Use `kubectl top pods` (requires Metrics Server). For example:

```
kubectl top pods -l name=overloaded-cpu --sort-by=cpu | head -n1 | awk
'{print $1}' > /opt/KUBET00401/KUBET00401.txt
```

  This lists pods with the label, sorts by CPU usage, takes the top line's name, and writes it to the file.
- **Commands & Tips:** `kubectl top` shows live resource use (CPU/memory). The `--sort-by=cpu` flag orders by CPU usage. Ensure Metrics Server is installed (otherwise `kubectl top` won't work). You could also use JSONPath:

```
kubectl top pods -l name=overloaded-cpu --no-headers --sort-by=cpu | awk
'{print $1}' > /opt/KUBET00401/KUBET00401.txt
```

  Check the file content afterwards. (If multiple namespaces, add `-n <ns>` as needed.)

Question 23: List Pods on Zone=zone-1

- **Context:** Nodes have a label `zone=zone-1`.
- **Task:** List all Pods running on nodes with `zone=zone-1` and save their names to `/opt/CKA/zone1-pods.txt`.
- **Solution:** Use `kubectl get pods` with a node selector. One approach:

```
kubectl get pods --all-namespaces \
  -l zone=zone-1 \
  -o custom-columns=NAME:.metadata.name --no-headers \
  > /opt/CKA/zone1-pods.txt
```

  This uses the `-l zone=zone-1` label selector and outputs only pod names. If pods aren't labeled, you could label nodes and use a field selector:

```
kubectl get pods --all-namespaces --field-selector spec.nodeName=<node-
name> -o name
```

  But here we assume pods were labeled by node zone.

- **Commands & Tips:** The `-l` flag filters by pod label [10]. If pods themselves aren't labeled by zone, you may first need to label pods or use `--field-selector` on `spec.nodeName`. For example, label each node (`kubectl label node <name> zone=zone-1`) and then use:

  ```
  kubectl get pods --all-namespaces -l zone=zone-1 -o name > /opt/CKA/zone1-
  pods.txt
  ```

  Finally, verify with `cat /opt/CKA/zone1-pods.txt`. Each line should contain a pod name (prefix "pod/" if using `-o name`, or just name if custom columns).

Total questions processed: 17.

---

[1] [2] [5] Upgrading kubeadm clusters | Kubernetes
https://kubernetes.io/docs/tasks/administer-cluster/kubeadm/kubeadm-upgrade/

[3] Backup and Restore Methods - KodeKloud Notes
https://notes.kodekloud.com/docs/CKA-Certification-Course-Certified-Kubernetes-Administrator/Cluster-Maintenance/Backup-and-Restore-Methods

[4] [6] kubectl drain | Kubernetes
https://kubernetes.io/docs/reference/kubectl/generated/kubectl_drain/

[7] kubectl scale | Kubernetes
https://kubernetes.io/docs/reference/kubectl/generated/kubectl_scale/

[8] Assigning Pods to Nodes | Kubernetes
https://kubernetes.io/docs/concepts/scheduling-eviction/assign-pod-node/

[9] Certified Kubernetes Administrator CKA | by Mohammad Abu Humaidan | Medium
https://medium.com/@abu7midan/certified-kubernetes-administrator-cka-ff7958af62bd

[10] Labels and Selectors | Kubernetes
https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/