CKAD HANDS-ON PRACTICE QUESTIONS

Part 1: For your YAML files, use Kubernetes Documentation or generate them from the CLI

---

1. Resource Requests and Limits

Task: Set CPU and memory requests and limits for a container in a Pod.

Question:

1. Create a Pod named `resource-limited-pod` with a single container running the `nginx` image.
2. Set the following resource limits for the `nginx` container:
    - Request: 100m CPU, 128Mi memory
    - Limit: 200m CPU, 256Mi memory
3. Verify that the resource requests and limits are set correctly.

---

2. Deployment with Rolling Updates

Task: Create a Deployment and perform a rolling update.

Question:

1. Create a Deployment named `my-app` that runs 3 replicas of the `nginx` image.
2. Expose the Deployment via a `ClusterIP` service named `my-app-service`.
3. Perform a rolling update to change the NGINX image version to `nginx:1.19`.
4. Verify that the rollout was successful, and ensure that the new version of the image is running.
5. Roll back to the previous version of the Deployment.

---

3. ConfigMaps and Secrets

Task: Use ConfigMaps and Secrets with Pods.

Question:

1. Create a ConfigMap named `my-config` with two key-value pairs: `app_name=myapp` and `app_version=1.0`.
2. Create a Secret named `my-secret` with the following data:

    - `username: admin`
    - `password: password123`
3. Create a Pod named `config-secrets-pod` using the `nginx` image, and mount the ConfigMap and Secret as environment variables.
4. Verify that the Pod can access both the ConfigMap and the Secret.

---

4. Namespaces

Task: Create and use namespaces.

Question:

1. Create a new namespace named `dev` in your cluster.
2. Create a Pod named `nginx-pod-dev` in the `dev` namespace, running the `nginx` image.
3. Create a second Pod named `nginx-pod-prod` in the `prod` namespace, running the `nginx` image.
4. Verify that the Pods are running in their respective namespaces.

---

5. StatefulSet

Task: Create a StatefulSet and use persistent storage.

Question:

1. Create a StatefulSet named `my-statefulset` with 3 replicas running the `nginx` image.
2. Ensure that each Pod in the StatefulSet has its own PersistentVolume (PV) using `hostPath` for storage.
3. Verify that the PersistentVolumes are created and attached to each Pod in the StatefulSet.
4. Delete one of the Pods in the StatefulSet and verify that it is recreated with the same PersistentVolume.

---

6. Job and CronJob

Task: Create a Job and a CronJob.

Question:

1. Create a Job named `backup-job` that runs a container using the `busybox` image with the command `echo "Backup completed"`.

2. Set the Job to run to completion only once.
3. Create a CronJob that runs the backup Job every day at midnight.
4. Verify that the CronJob runs as expected and the Job completes successfully.

---

7. NetworkPolicies

Task: Implement NetworkPolicies to control traffic.

Question:

1. Create two Pods named `pod-a` and `pod-b` in the same namespace.
2. Create a `NetworkPolicy` that only allows traffic from `pod-a` to `pod-b`.
3. Verify that traffic from `pod-b` to `pod-a` is blocked and traffic from `pod-a` to `pod-b` is allowed.

---

8. Debugging Pods

Task: Troubleshoot Pod issues.

Question:

1. Create a Pod named `nginx-pod` using the `nginx` image, but intentionally create a misconfiguration such as incorrect environment variables or missing files.
2. Use `kubectl` commands (e.g., `describe`, `logs`, `exec`) to debug and fix the issues with the Pod.
3. Once fixed, ensure the Pod is running correctly.

---

9. RBAC - Role-Based Access Control

Task: Implement RBAC policies.

Question:

1. Create a `Role` that allows reading Pods in the `default` namespace.
2. Create a `RoleBinding` that binds the `Role` to a service account named `read-only-sa`.
3. Create a Pod that uses the `read-only-sa` service account.
4. Verify that the Pod can read the Pod list, but cannot create or delete Pods.

1.

---

10. Helm - Install and Manage Packages

Task: Use Helm to deploy an application.

Question:

1. Install Helm and initialize it (if necessary).
2. Use Helm to deploy the `nginx` chart to your cluster.
3. Expose the application via a Service using Helm's chart configuration.
4. Upgrade the application to a newer version of the `nginx` chart.
5. Roll back the deployment to the previous version.

Part 2: For your YAML files, use Kubernetes Documentation or generate them from the CLI

1. Pod Creation and Management

1. Create a Pod with the name `nginx-pod` running the `nginx` image.
2. Create a Pod with the name `nginx-pod` that runs the `nginx` image with two containers: one running `nginx` and another running `busybox` with the command `sleep 3600`.
3. Set resource requests and limits for a Pod running `nginx:latest` with a CPU request of `100m` and memory request of `256Mi`.
4. Delete the Pod `nginx-pod` using `kubectl` command.
5. Scale the Pod `nginx-pod` to 3 replicas.

2. Deployments and Rollouts

1. Create a Deployment named `nginx-deployment` that runs the `nginx` image with 3 replicas.
2. Expose the `nginx-deployment` via a `ClusterIP` Service.
3. Perform a rolling update to change the `nginx` version from `1.18` to `1.19`.
4. Rollback the Deployment `nginx-deployment` to its previous revision.
5. Ensure the Deployment `nginx-deployment` has at least 2 replicas running at all times.

3. StatefulSets and Persistence

1. Create a StatefulSet named `web-statefulset` running `nginx` with 3 replicas.
2. Create a PersistentVolume and PersistentVolumeClaim and link it to a StatefulSet with the name `stateful-web-app`.
3. Scale the StatefulSet `web-statefulset` to 5 replicas and verify persistent storage behavior.
4. Delete one Pod from the StatefulSet and verify that it's recreated with the same PersistentVolume.
5. Implement and verify that each Pod in the StatefulSet gets a unique DNS name.

4. ConfigMaps and Secrets

1. Create a ConfigMap with the name `nginx-config` containing key-value pairs such as `nginx-port=8080`.
2. Create a Secret named `db-credentials` with data like `username=admin` and `password=secret`.
3. Mount a ConfigMap as an environment variable in a Pod running `nginx`.
4. Mount a Secret as a volume in a Pod and ensure sensitive data is protected.
5. Create a Pod that uses both a ConfigMap and Secret and verify that they are correctly injected into the Pod's containers.

5. Services

1. Create a Service of type `ClusterIP` to expose the `nginx-pod` on port 80.
2. Create a Service of type `LoadBalancer` for the `nginx-deployment`.
3. Create a NodePort service to expose an application on port `30001`.
4. Expose the `nginx` Deployment via a `ClusterIP` service and test access to the service from another Pod.

6. Networking and Network Policies

1. Create a NetworkPolicy to block all traffic to a Pod except from a specific `nginx-pod`.
2. Create a Pod `pod-a` and another `pod-b`. Apply a NetworkPolicy that allows only `pod-a` to communicate with `pod-b`.
3. Use the `kubectl port-forward` command to expose a Pod running `nginx` locally on your machine.
4. Implement a NetworkPolicy that allows traffic only from Pods in the same namespace.
5. Create a Pod that can only access a Service via its ClusterIP.

7. Horizontal Pod Autoscaling

1. Create a Deployment for `nginx` with 2 replicas and set up a Horizontal Pod Autoscaler (HPA) for it based on CPU usage.
2. Set a target CPU utilization of 50% for the HPA and verify its behavior.
3. Ensure that the minimum replicas for the HPA are 2 and maximum replicas are 4.
4. Monitor the scaling events and ensure that HPA scales the Deployment automatically based on load.

8. Ingress and TLS

1. Set up an Ingress controller and create an Ingress resource to expose the `nginx-deployment` via `myapp.example.com`.
2. Enable TLS termination on the Ingress and secure the connection using a self-signed certificate.
3. Create an Ingress resource that allows routing traffic to two different services based on URL path (e.g., `/app` and `/api`).
4. Configure an Ingress to route traffic to a Service using both HTTP and HTTPS.
5. Verify that your Ingress resource correctly routes traffic to the appropriate Pods.

9. Jobs and CronJobs

1. Create a Job named `backup-job` that runs a simple `echo "Backup Complete"` using the `busybox` image.
2. Create a CronJob that runs the `backup-job` every day at midnight.
3. Check the CronJob's logs and verify that it executes correctly.
4. Scale the Job to run multiple instances and verify that it completes successfully.
5. Create a CronJob that runs once every hour and executes a script that writes to a log file.

10. RBAC and ServiceAccount:

1. Create a ServiceAccount and assign it a role that only allows access to the `nginx` Pods.
2. Create a RoleBinding to assign the role to a specific ServiceAccount in a namespace.
3. Use the `kubectl` command to test the permissions granted by the ServiceAccount.