Azure Kubernetes Service (AKS) – Practical Assignment

This document provides a comprehensive solution to key Kubernetes concepts using Azure Kubernetes Service (AKS), including deployments, service types, volumes, scaling, and health probes.

---

1. Deploy ReplicaSet, ReplicationController, and Deployment on AKS

In AKS, we use `kubectl` to interact with the Kubernetes API and deploy resources.

1.1 ReplicaSet

```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: azure-replicaset
spec:
  replicas: 3
  selector:
    matchLabels:
      app: demo-app
  template:
    metadata:
      labels:
        app: demo-app
    spec:
      containers:
      - name: demo-container
        image: nginx
```

Usage in AKS:

- Run with: `kubectl apply -f replicaset.yaml`
- Use for ensuring a stable set of Pods.

Advantages:

- Ensures desired number of Pods are running at all times.
- Good for basic workload scaling.
- Self-healing: replaces failed Pods automatically.

Disadvantages:

- No support for rolling updates.
- Lacks native version tracking or rollback support.
- Typically managed indirectly by a Deployment.

1.2 ReplicationController

```
apiVersion: v1
kind: ReplicationController
metadata:
  name: azure-rc
spec:
  replicas: 2
  selector:
    app: demo-app
  template:
    metadata:
      labels:
        app: demo-app
    spec:
      containers:
      - name: demo-container
        image: nginx
```

Advantages:

- Maintains a consistent number of Pods.
- Was foundational to Kubernetes replication logic.

Disadvantages:

- Deprecated in favor of ReplicaSet and Deployment.
- No support for rolling updates or version management.
- Limited in flexibility and scalability.

1.3 Deployment

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: azure-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
```

```
      app: demo-app
  template:
    metadata:
      labels:
        app: demo-app
    spec:
      containers:
      - name: demo-container
        image: nginx:1.21
```

Advantages:

- Automates rollouts and rollbacks.
- Provides version tracking and declarative updates.
- Works well with CI/CD pipelines.
- Abstracts and manages ReplicaSets efficiently.

Disadvantages:

- More complex than lower-level controllers.
- Requires careful configuration for update strategies.

---

2. Kubernetes Service Types in AKS

2.1 ClusterIP

- Default service type.
- Used for internal communication between AKS Pods.

2.2 NodePort

- Exposes the service on a port on each AKS node.
- Access from outside via `<NODE-IP>:<NodePort>`.

2.3 LoadBalancer

- AKS integrates directly with Azure Load Balancer.

```
apiVersion: v1
kind: Service
metadata:
  name: azure-lb
spec:
  type: LoadBalancer
  selector:
    app: demo-app
```

```
  ports:
  - protocol: TCP
    port: 80
    targetPort: 80
```

Result: Azure provisions a public IP via LoadBalancer.

---

3. PersistentVolume (PV) and PersistentVolumeClaim (PVC) in AKS

Azure uses Azure Disks or Azure Files for storage.

3.1 PersistentVolumeClaim

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: azure-pvc
spec:
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 1Gi
  storageClassName: default
```

3.2 Pod using PVC

```
apiVersion: v1
kind: Pod
metadata:
  name: azure-pvc-pod
spec:
  containers:
  - name: nginx
    image: nginx
    volumeMounts:
    - mountPath: "/mnt/storage"
      name: volume
  volumes:
  - name: volume
    persistentVolumeClaim:
      claimName: azure-pvc
```

---

4. Managing AKS Clusters

4.1 Creating AKS Cluster

```
az aks create \
  --resource-group myResourceGroup \
  --name myAKSCluster \
  --node-count 2 \
  --enable-addons monitoring \
  --generate-ssh-keys
```

4.2 Get Credentials

```
az aks get-credentials --resource-group myResourceGroup --name myAKSCluster
```

4.3 Scale Cluster

```
az aks scale --resource-group myResourceGroup --name myAKSCluster --node-count 3
```

4.4 Upgrade Cluster

```
az aks upgrade --resource-group myResourceGroup --name myAKSCluster --
kubernetes-version 1.29.2
```

---

5. Configure Liveness and Readiness Probes

```
livenessProbe:
  httpGet:
    path: /healthz
    port: 80
  initialDelaySeconds: 5
  periodSeconds: 10

readinessProbe:
  httpGet:
    path: /ready
    port: 80
  initialDelaySeconds: 5
  periodSeconds: 10
```

These ensure Azure Load Balancer only routes traffic to healthy Pods.

---

6. Taints and Tolerations in AKS

Add Taint

```
kubectl taint nodes <node-name> role=infra:NoSchedule
```

Add Toleration in Pod

```
spec:
  tolerations:
  - key: "role"
    operator: "Equal"
    value: "infra"
    effect: "NoSchedule"
```

Used to restrict workloads to specific node types (e.g., GPU nodes).

---

7. Attach PVCs to Pods (Azure Disk or Azure File)

Persistent Volume Claims allow you to dynamically request storage backed by Azure-managed services like Azure Disks or Azure Files.

Step-by-Step:

1. Create a PersistentVolumeClaim (as shown in Section 3.1).
2. In the Pod spec, mount the volume using the claim name:

```
volumes:
  - name: volume
    persistentVolumeClaim:
      claimName: azure-pvc
```

1. Choose a proper `storageClassName`:
2. For Azure Disks: `default`
3. For Azure Files: `azurefile`

This allows data to persist even if the Pod is deleted or rescheduled.

---

8. Configure Health Probes in AKS

Probes help the AKS scheduler and load balancer determine if your application is healthy and ready.

Liveness Probe:

Checks if the container is alive. If it fails, Kubernetes restarts the container.

```
livenessProbe:
  httpGet:
    path: /healthz
    port: 8080
  initialDelaySeconds: 10
  periodSeconds: 15
```

Readiness Probe:

Checks if the application is ready to accept traffic. If it fails, the Pod is removed from the Service's endpoints.

```
readinessProbe:
  httpGet:
    path: /ready
    port: 8080
  initialDelaySeconds: 5
  periodSeconds: 10
```

These probes are critical in AKS for:

- Smooth rolling updates
- Avoiding traffic to unready Pods
- Detecting crashes proactively

---

9. Autoscaling in AKS (Horizontal Pod Autoscaler)

Purpose:

Horizontal Pod Autoscaler (HPA) automatically adjusts the number of pod replicas based on CPU utilization or custom metrics.

Steps to Configure HPA:

1. **Ensure Metrics Server is Installed:**

```
kubectl apply -f https://github.com/kubernetes-sigs/metrics-server/releases/
latest/download/components.yaml
```

1. **Create an HPA:**

```
kubectl autoscale deployment azure-deployment \
  --cpu-percent=50 \
  --min=1 \
  --max=5
```

1. **Monitor HPA Status:**

```
kubectl get hpa
```

This will show current CPU usage, target thresholds, and number of replicas.

Important Notes:

- AKS must have the metrics server running to support HPA.
- HPA helps manage resource usage efficiently and handles load spikes gracefully.
- You can also use custom metrics for autoscaling beyond CPU (e.g., memory or application-level metrics).
