Deployments

Overview

- Deployments help us manage a set of identical pods through a single object.
- Through the YAML file of a deployment we specify the state.

```
kubectl run nginx-deployment --image=nginx --replicas=3
```

Rolling updates

- When a configuration changes, for instance, by updating the image used by the pods, Kubernetes will try to reconcile this change in different ways.
- This is specified in the **strategy.type** field.

Rolling updates strategies

- **Recreate:** All running pods in the deployment are killed, and new pods are created in their place.
    - Does not guarantee zero downtime!
    - Useful in case you are not able to have multiple versions of your application running simultaneously.
- **RollingUpdate:** Gracefully updates pods one at a time to prevent your application from going down.
    - **maxUnavailable:** max number of pods to kill.
    - **maxSurge:** max number of new pods.

Rolling back

- Kubernetes keeps track of the rollout history, so you can easily rollback to a previously deployed version at any time.
- `kubectl rollout undo deployment nginx-deployment --to-revision=1`

Services

Overview

- Many tutorials tend to merge these two, but they are different!
- Services are responsible of enabling access to a set of pods.
- Deployments are responsible for keeping the pods alive.

But wait, do we need services?

- We could simply enable network access to each pod, right?
- Sadly, pods can be created and destroyed mercilessly, so this is not an option.
- Thanks to services, networking rules are defined in terms of *labels*, not IPs.

Routing a request to a services

- A request comes to the external IP address at the port assigned to the service.
- Once inside of the cluster, the request is routed to the **service's** clusterIP address.
- The service then decides a pod to send the request to, among those that have the matching label.

What about other kinds of backends (not in GKE cluster)?

- You can use services without selectors:
  - You want to have an external database cluster in production, but in your test environment you use your own databases.
  - You are migrating a workload to Kubernetes. Whilst evaluating the approach, you run only a proportion of your backends in Kubernetes.

Services without selector

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  ports:
    - protocol: TCP
      port: 80
      targetPort: 9376
```

```yaml
# An Endpoint is needed because GKE will not create it
apiVersion: v1
kind: Endpoints
metadata:
  name: my-service
subsets:
  - addresses:
      - ip: 192.0.2.42
    ports:
      - port: 9376
```

Types of services

- Cluster IP is the default approach when creating a Kubernetes Service.
- The service is allocated an internal IP that other components can use to access the pods.
- Having a single IP allows the service to be load balanced across multiple Pods.

ClusterIP

```yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp1-clusterip-svc
  labels:
    app: webapp1-clusterip
spec:
  ports:
  - port: 80
  selector:
    app: webapp1-clusterip
```

ExternalIP

- Make a service available outside of the cluster

```yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp1-externalip-svc
  labels:
    app: webapp1-externalip
spec:
  ports:
  - port: 80
  externalIPs:
  - HOSTIP
  selector:
    app: webapp1-externalip
```

TargetPort

- Separate available/listening ports.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp1-clusterip-targetport-svc
  labels:
    app: webapp1-clusterip-targetport
spec:
  ports:
  - port: 8080
    targetPort: 80
  selector:
    app: webapp1-clusterip-targetport
```

NodePort

- Exposes the service on *each* node's IP in the specified port.
- Service is then available on all nodes at that port.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp1-nodeport-svc
  labels:
    app: webapp1-nodeport
spec:
  type: NodePort
  ports:
  - port: 80
    nodePort: 30080
  selector:
    app: webapp1-nodeport
```

LoadBalancer

- For cloud-managed clusters, the cloud provider issues a public IP.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: webapp1-loadbalancer-svc
  labels:
    app: webapp1-loadbalancer
spec:
  type: LoadBalancer
  ports:
  - port: 80
  selector:
    app: webapp1-loadbalancer
```

Demo

- Deployments and Services Together
- **Further reference:** Official docs for K8s Services