06: Kubernetes Deployments

- Components to Deployments

1. the spec
2. the controller

Spec is declarative YMAL object. Describe desired state of a stateless app

Deployment controller implements and mages it. Controller operates as a background loop

1. start with a stateless app
2. package it as a container
3. define it in a Pod template

This stateless app doesn't scale, self-heal, updates or rollbacks autmatically.

4. Wrap them in a Deployment object

The container holds the application, the Pod augments the container with labels, annotations and other metadata, and **Deployment further augments things with scaling and updates**.

POST the deployment object to the API Server where Kubernetes implemetns it and the Deployment controller watches it.

- A deployment object only manages a single Pod template.
- A deployment can manage multiple replicas of the same Pod.

ReplicaSets

- Not recommended manage ReplicaSets directly -> Manage via Deployment controller
- CONTAINERS: Great way to package apps and dependencies
- PODS: allow containers to run on Kubernetes and enable schaduling and other stuff.
- ReplicaSets: Manage Pods and bring self-healing and scaling.
- Deployments: Manage ReplicaSets and add rollouts and rollbacks.

Self-healing and Scalability

- CONTAINERS: Co-locate containers, share volumes, memory, simple networking, etc.

If NODE fails, the POD IS LOST. Deployments:

- self-healing: replace Pods at fail
- scaled: Increase or decrease load

All About the STATE

- Desired state
- Observed State
- Reconciliation

Imperative model has no concept of desired state, it's just a list of steps and structions. Kubernetes prefers declarative model.

Reconciliation

- ReplicaSets are implemented as a controller running as a background reocncilation loop checking the right number of Pod replicas are present on the cluster.

Rolling updates with deployments

- Zero-downtime rolling-updates (rollouts) of stateless apps
- Requerimients from microservices apps:
    - Loose coupling via APIs
    - Backwards and forwards compatibility

Each Deploymenmt describes:

- How many Pod replicas
- What images to use for Pod's containers
- What network ports to expose
- Details about how to perform rolling updates

The repolica seets enters on a watch loop suring observed state and desired state are in agreement. A deployment objects sits above the RSet, governing its configuration and providing mechanisms for rollouts and rollbacks.

New configuration YAML files creates new replicaSets than controlls new Pods with the updated configuration. When old config Pods are zero, all new pods has the new configuration. Old replicaSets config still remains for rollbacks to reverting to previous versions. This process is the oposite to the described rollout.

Create a Deployment

YAML snippet from deploy.yml file. Defines a single-container Pod wrapped in a Deployment object.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deploy
  replicas: 10 # Number of Pods to deploy/mng
  selector:
    matchLabels:
      app: hello-world
  revisionHistoryLimit: 5
  progressDeadlineSeconds: 300
  minReadySeconds: 10
  strategy: # Controls how updates happen
    type: RollingUpdate
      maxUnavailable: 1
      maxSurge: 1
  template: # Pod template
```

```
    metadata:
    labels:
    app: hello-world
  spec:
    containers:
    - name: hello-pod
    image: nigelpoulton/k8sbook:1.0
    ports:
    - containerPort: 8080
```

- spec.selector is a list of labels that pods must have in order for the deployment to manage them. The Deployment selector matches the labels assigned to the Pod lower in the Pod template
- spec.revisionHistoryLimit How many older versions / replicaSets t keep.
- spec.progressDeadlineSeconds How long to wait during a rollout for each new replica to come online in minutes. The clk is rst for each new repolica.
- spec.strategy how to update the Pods when a rollout occurs

```
# explore
kubectl get deploy hello-deploy

kubectl describe deploy hello-deploy
```

Deployments automatically create associated ReplicaSets

```
kubectl get rs
```

Toe get detailed information about the replica set use `kubectl describe rs hello-deploy-XXXXXXXXXXX`

Accessing the App

To access to all the replicas of the application you need a Kubernetes Service object.

The YAML defines a Service that works with Pod repolicas depoloyed.

```
apiVersion: v1
kind: Service
metadata:
  name: hello-svc
  labels:
    app: hello-world
  spec:
    type: NodePort
    ports:
    - port: 8080
      nodePort: 30001
```

```
    protocol: TCP
  selector:
    app: hello-world
```

```
$ kubectl apply -f svc.yml
servic e/hello-svc created
```

With a Service deployed, it can access tha app by hitting any of the cluster nodes on port 30001.

Scalling Operations

Imperatively

```
kubectl scale
```

declaratively updating a YAML file and re-posting it to the API server.

```
kubectl scale deploy hello-deploy --replicas 5
```

The current state no longer matches the manifest. This can cause issues. __You should always keep the YAML manifests in sync with live enviroments. -> the correct wayy, edity YAML file, set a different number of replicas and run `kubectl apply -f deploy.yml`

Rolling Update

Rollout: a new version of the app has already been created and containerized as an image with the 2.0 tag... You have to perform a rollout. We will ignore real-world CI/CD workflows and version control tools.

1. Update the image version in the Deployments resource manifest.

```
nigelpoulton/k8sbook:1.0
nigelpoulton/k8sbook:2.0
```

The .spec section contains settings governing the updates

```
revisionHistoryLimit: 5
progressDeadlineSeconds: 300
minReadySeconds: 10
strategy:
  type: RollingUpdate
  rollingUpdate:
```

```
    maxUnavailable: 1
    maxSruge: 1
```

- revisionHistoryLimit: keep 5 previous releases. This means the previous 5 ReplicaSet objects will be kept to easily rollback.
- progressDeadlineSeconds: 5 minute windows
- minReadySeconds: rate at which replicas are replaced. The example tells that any new replica must be up and running for 10 serconds without issues before replace the next one in sequence. **In real world the value must be large enought to trap common failures**.
- strategy:
    - Rolling update strategy
    - Never have more than one Pod below desired state -> maxUnavailable -> NEVER HAVE MORE THAN 11 (if 10 desired)
    - Never have more than one Pod above desired state. -> maxSurge -> NEVER HAVE LESS THAN 9 REPLCAS (if 10 desired)

The Deployment file has a selector block. This is a list of labels the Deployment controller looks for when finding Pods to update during rollout operation.

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-deploy
spec:
  selector: # THE DEPLOY MANAGE
    matchLabels: # ALL REPLICAS CON THE CLUSTER
      app: hello-world # WITH THIS LABEL
...
```

```
kubectl apply -f deploy.yml
deployment.apps/hello-deploy configure
```

```
kubectl rollout status deployment hello-deploy
>> Waiting for depoyment "hello-deploy"
>> rollout... 4 out of 10 new replicas...
>> ...
```

```
kubectl get depoloy hello-deploy
```

```
kubectl rollout pause deploy hello-deploy

kubectl describe deploy hello-deploy

kubectl rollout resume depoloy hello-deploy

kubectl get deploy hello-deploy
```

Rollback

```
kubectl rollout history deployment hello-deploy
REVISION CHANGE-CAUSE
1        <none>
2        kubectl apply --filmename-deploy.yml

kubectl get rs
NAME  READY AGE DESIRED CURRENT

kubectl describe rs hello-deploy-XXXXXX
```

A rollback follows the same logic and rules as an update/rollout.

```
kubectl rollout undo deployment hello-deploy --to-revision=1
```

The rollback was imperative. The current state of the cluster no longer matches with sou8rce YAML files. This is a fumental flaw with the imperative appoach and a major reason to only uupdate the cluster declaratively by updating YAML files and reposting them.

Clean-up

```
kubectl delete -f deploy.yml
kbuectl delete -f svc.yml
```