KUBERNETES

Core

Part 2

Replication Control

A single pod is running an application- if it fails, it also crashes.

To prevent this, we want more than one POD running.
Replication Control helps run multiple instances of a single POD in the Kubernetes cluster- providing high availability.

- Can be run with only one POD.
- Make sure there is a specific number of PODs up.

Also, it can create multiple PODs and share the load across them.

Create Replication Controller

$ kubectl create -f rc-definition.yaml

```yaml
apiVersion: v1
kind: ReplicationController
metadata:
  name: myapp-rc
  labels:
    app: my-app
    type: front-end

spec:
  template:

    metadata:
    name: myapp-rc
      labels:
        app: my-app
        type: front-end
    spec:
      containers:
        - name: nginx-container
          image: nginx:1.15.5
replicas: 3

```

View created RC

$ kubectl get replication controller

Replica Set

```yaml
apiVersion: apps/v1
kind: Replication
metadata:
  name: myapp-replicaset
  labels:
    app: myapp
    type: front-end

spec:
  template:
    metadata:
      name: myapp-pod
      labels:
        app: myapp
        type: front-end
    spec:
      containers:
        - name: nginx-container
          image: nginx

replicas: 3

selector:
  matchLabels:
    app: myapp
    type: front-end

```

Create Replication Set

$ kubectl create -f replicaset-definition.yaml

Deployment

Update, scale and rollback the enviorment

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
 name: my-app-deployment
 labels:
   app: myapp
   type: front-end
spec:
 template:
  metadata:
   name: myapp-pod
    labels:
     app: myapp
     type: front-end
  spec:
   containers:
   - name: nginx-container
     image: nginx

replicas: 3
selector:
 matchLabels:
  type: front-end
```

Services

Enable communication between various components within and outside of the app.

Help to connect other apps or users

Connect the Front-end to the users
Connect the Back-end to the Front-end pods
Connect to external data source

3 main services

NodePort
ClusterIP – Default
LoadBalancer

See all services
$ kubectl get servicesl

NodePort

Listen to a port on the node and forward the request on that port, to a port in a POD running the app.
Multiple PODs in service- use the same selector
Multiple PODs in PODs- use the same labels
Also, work on multiple PODs- different nodes with the same ports- Auto

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-app-service

spec:
  type: NodePort
  ports:
    - targetPort: 8080
      port: 80
      nodePort: 30000 # Range 30000-32767
  selector:
    app: my-app
    type: front-end
```

ClusterIP

The front end communicates with the back end and itself with Redis services.
All nodes have a non-static IP.
Kubernetes helps us to provide a single interface to access the pods in a group.

- A Service created for Back-end PODs will help group all the Back-end PODs together and provide a single interface for other PODs to access this service- the request is sent randomly.
- Each service get an IP and a name.

ClusterIP ".yaml" file

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-app-service

spec:
  type: ClusterIP
  ports:
    - targetPort: 80
      port: 80

  selector:
    app: my-app
    type: front-end

```

LoadBalancer

Can use a cloud load balancer, such as AWS, Google, and Azure.

```yaml
apiVersion: v1
kind: Service
metadata:
  name: my-app-service

spec:
  type: LoadBalancer
  ports:
    - targetPort: 80
      port: 80
      nodePort: 30008

```

Namespaces

- PODs, clusters, services, and nods are created in a namespace by default.
- Kubernetes creates a set of PODs and services for internal purposes, such as those required by networking solutions, the DNS service, etc.
- To isolate the user and prevent you from deleting/modifying the services and create them in other cluster startup name kube-system.
- Resources that should be made available to all users are created in kube-public.
- Policies can be assigned to namespaces for permissions purposes.
- Can assign ‘quota’ of resources for each namespace- each namespace grants a certain amount of nodes and won’t use more then it’s allowed.

- Resources in the same namespace refer to each other with names.
- In other namespaces they refer as:

<service-name>.<namespace>.<service>.<domain>

- Can view PODs in other namespace:

$ kubectl get pods --namespace=<namespace-name>

- Can create PODs in other namespace:

$ kubectl create -f file.yaml--namespace=<namespace-name>

Create Namespace

- <file>.yaml

```yaml
apiVersion: v1
kind: Namespace
metadata:
  name: dev

```

- Command:
  $ kubectl create namespace <name>

- Switch default namespace:
  $ kubectl config set-context $(kubectl current-context)
  --namespace=<name>

Imperative vs Declarative

Imperative: Step by step.

- A list of commands.
- To create use: ‘run’, ‘create’, and ‘expose’.
- To update: ‘edit’, ‘scale’, ‘replace’, and ‘set’.

Declarative: End goal

- Uses the ‘apply’ command.

  $ kubectl apply

  - Take into consideration: the local configuration file, the live object definition on Kubernetes, and the last applied configuration.
  - When using the command the <file>.yaml transform to <file>.json and store the last applied configuration.