Review 2

Deploying Applications on Kubernetes

Business Training

1

Simple Pod with namespace and labels

- **Additional information**
  - **Namespace**: Namespaces provide a scope for Kubernetes resources, splitting the cluster in smaller units.
  - **Labels**: Labels are intended to be used to specify identifying attributes of objects that are meaningful and relevant to users, but do not directly imply semantics to the core system.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: mynginxapp
  namespace: default
  labels:
    name: mynginxapp
    profile: dev
spec:
  containers:
  - name: mynginxapp
    image: nginx
    ports:
      - containerPort: 80
```


2

Review Session 2

A Multi container Pod: Main Container with Side Car Container

- **Main Container and the Side Car Container share a Volume**

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: pod-with-sidecar
spec:
  # Create a volume called 'shared-logs' that the pp and sidecar share.
  volumes:
  - name: shared-logs
    emptyDir: {}
  containers:
  - name: app-container # Main application container
    # Simple application: write the current date  to the log file every 5 seconds
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "while true; do date >> /var/log/app.txt; sleep 5;done"]
    volumeMounts: # Mount the pod's shared log file into the app container
    - name: shared-logs
      mountPath: /var/log

  - name: sidecar-container # Sidecar container
    image: nginx:1.7.9
    ports:
      - containerPort: 80
    volumeMounts: # Mount the pod's shared log file into the sidecar
    - name: shared-logs
      mountPath: /usr/share/nginx/html # nginx-specific mount path
```

M.Romdhani, 2020 3

3

Review Session 2

Using Deployments

- **Saving this manifest into `nginxdeploy.yaml` and submitting it to a Kubernetes cluster will create the defined Deployment, ReplicaSet and the Pods**
  - You can then get the current Deployments deployed:

    `kubectl get deployments`
  - You can then get the current ReplicaSets deployed:

    `kubectl get rs`
  - You can then get the current pods deployed:

    `kubectl get pods`

```yaml
# for versions before 1.9.0 use apps/v1beta2
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2 # tells deployment to run 2 pods
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
      - name: nginx
        image: nginx:1.14.2
        ports:
        - containerPort: 80
```

Pod Template

M.Romdhani, 2020 4

4

Review Session 2

Managing Resources for Containers

- **Meaning of CPU units**
  - One cpu, in Kubernetes, is equivalent to 1 vCPU/Core for cloud providers and 1 hyperthread on bare-metal Intel processors.
  - Fractional requests are allowed. The expression 0.1 is equivalent to the expression 100m, which can be read as “one hundred millicpu”
- **Meaning of Memory units**
  - You can express memory as a plain integer or as a fixed-point integer using one of these suffixes: E, P, T, G, M, K. You can also use the power-of-two equivalents: Ei, Pi, Ti, Gi, Mi, Ki.
- **The following Pod has two Containers.**
  - Each Container has a request of **0.25 cpu and 64MiB** and a limit of **0.5 cpu and 128MiB** of memory.

```yaml
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "password"
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: wp
    image: wordpress
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

M.Romdhani, 2020

5

Review Session 2

Defining min, max, and default resources using LimitRange

- **We can create LimitRange objects to indicate any combination of:**
  - min and/or max resources allowed per pod
  - default resource limits
  - default resource requests
  - maximal burst ratio (limit/request)
- **LimitRange objects are namespaced**
- **They apply to their namespace only**

```yaml
apiVersion: v1
kind: LimitRange
metadata:
  name: my-very-detailed-limitrange
spec:
  limits:
  - type: Container
    min:
      cpu: "100m"
    max:
      cpu: "2000m"
      memory: "1Gi"
    default:
      cpu: "500m"
      memory: "250Mi"
    defaultRequest:
      cpu: "500m"
```

M.Romdhani, 2020

6

6

Review Session 2

Deployment Updates Stategies

- **Strategy: describes the method used to update the deployment**
  - **Recreate** is pretty self explanatory, All existing Pods are killed before new ones are created
  - **RollingUpdate** (The default Strategy)cycles through updating the Pods according to the parameters: **maxSurge** and **maxUnavailable**
- **maxUnavailable**
  - Optional field that specifies the maximum number of Pods that can be unavailable during the update process. The default value is 25%.
- **maxSurge**
  - Optional field that specifies the maximum number of Pods that can be created over the desired number of Pods. The default value is 25%.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: deploy-example
spec:
  replicas: 3
  revisionHistoryLimit: 3
  selector:
    matchLabels:
      app: nginx
      env: prod
  strategy:
    type: RollingUpdate
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
  template:
    <pod template>
```

M.Romdhani, 2020 7

7

Review Session 2

Managing Rollout / HowTO

- **How to update an image ?**
  - **kubectl set image** deployment/my-deploy **www=image:v2**
- **How to check the Rollout Status ?**
  - **kubectl rollout status** deployment/my-deploy
- **How to check the Rollout History ?**
  - **kubectl rollout history** deployment/my-deploy
- **How to Roolback to the previous version ?**
  - **kubectl rollout undo** deployment/my-deploy
- **How to Roolback to the previous given version ?**
  - **kubectl rollout undo** deployment/my-deploy **--to-revision=2**

M.Romdhani, 2020 8

8

Review Session 2

Deploying DaemonSets

- **A DaemonSet ensures that all (or some) Nodes run a copy of a Pod.**
  - As nodes are added to the cluster, Pods are added to them.
  - As nodes are removed from the cluster, those Pods are garbage collected. Deleting a DaemonSet will clean up the Pods it created.
  - Typical uses of a DaemonSet are:
    - Running a cluster storage daemon, such as glusterd, ceph, on each node.
    - Running a logs collection daemon on every node, such as fluentd or filebeat.
    - Running a node monitoring daemon on every node, such as Prometheus Node Exporter

```yaml
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: my-daemonset
  namespace: my-namespace
  Labels:
    key: value
spec:
  template:
    metadata:
      labels:
        name: my-daemonsetcontainer
    ...
  selector:
    matchLabels:
      name: my-daemonsetcontainer
```

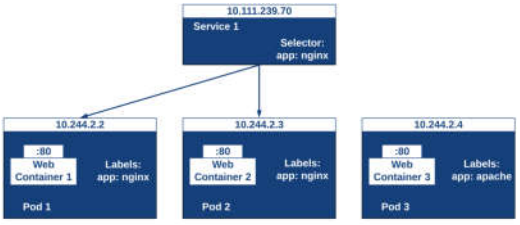
M.Romdhani, 2020 9

9

Review Session 2

Services

- **Stable resource to access to Pods / Services are configured using kube-proxy**
  - **Have Static IP**
  - **Havec Static DNS Name**
  - **Use Selecters to specify the controlled Pods**
    - **List of endpoints**
  - **Provide Load balancing**
- **3 Types : ClusterIP, NodePort, and LoadBalancer**
  - **Local access** : ClusterIP
  - **RemoteAccess**: NodePort, LoadBalancer



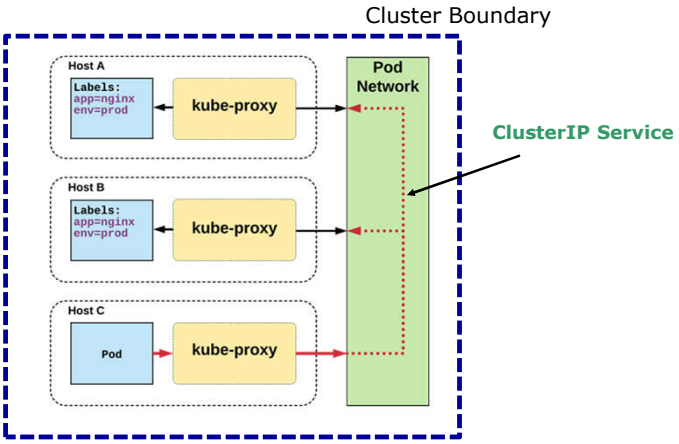
M.Romdhani, 2020 10

10

Review Session 2

ClusterIP Services

- **Local access Only**



M.Romdhani, 2020

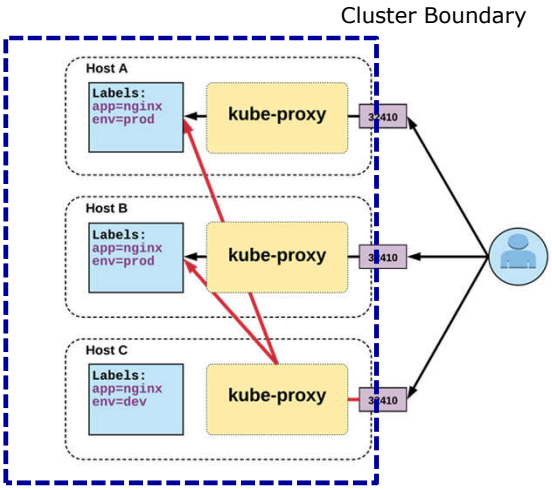
11

11

Review Session 2

NodePort Services

- **Remote Access possible through direct access to Nodes**



M.Romdhani, 2020

12

12

LoadBalancer Services

- **Remote access managed by an External load balancer**