

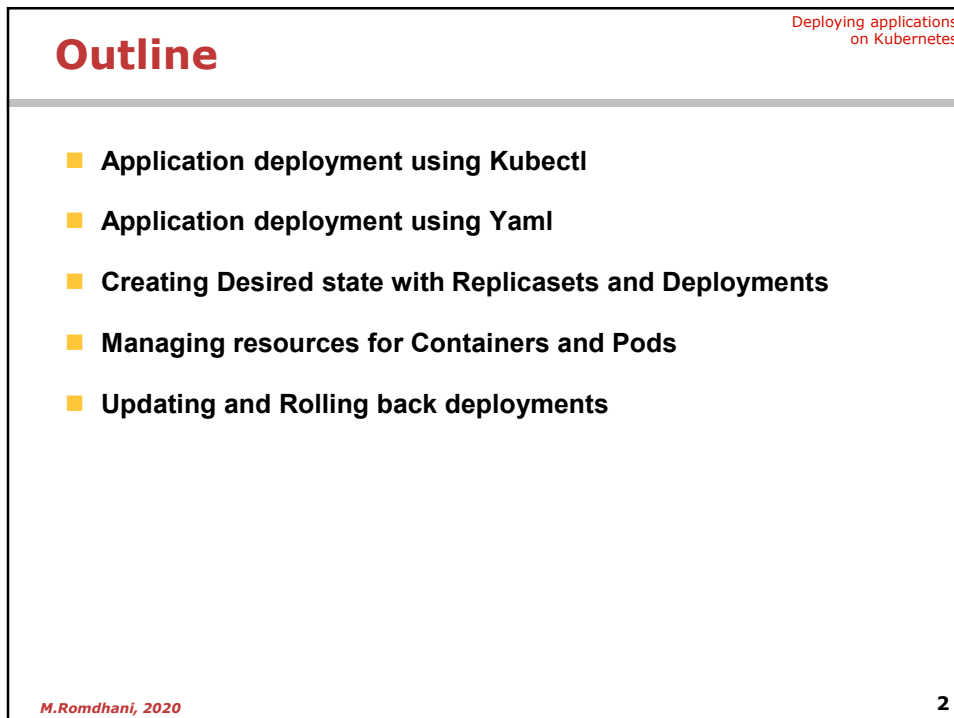
The slide features a purple header and footer bar. The header bar contains a small collage of images on the left. The main content area is white and contains the Kubernetes logo (a blue ship's wheel) at the top center, followed by the text "Unit 2" in red. Below this is the main title "Deploying applications on Kubernetes" in a large, bold, red font. In the bottom right corner, there are three small icons (a circle, a square, and a triangle) above the text "Business Training".

Unit 2

Deploying applications on Kubernetes

Business Training

1



The slide has a white background with a red header bar. The header bar contains the word "Outline" in a large, bold, red font on the left, and the text "Deploying applications on Kubernetes" in a smaller, red font on the right. Below the header bar is a list of five items, each preceded by a yellow square bullet point. At the bottom left, there is a small red text "M.Romdhani, 2020", and at the bottom right, there is a small red text "2".

Outline

Deploying applications on Kubernetes

- Application deployment using Kubectl
- Application deployment using Yaml
- Creating Desired state with Replicasets and Deployments
- Managing resources for Containers and Pods
- Updating and Rolling back deployments

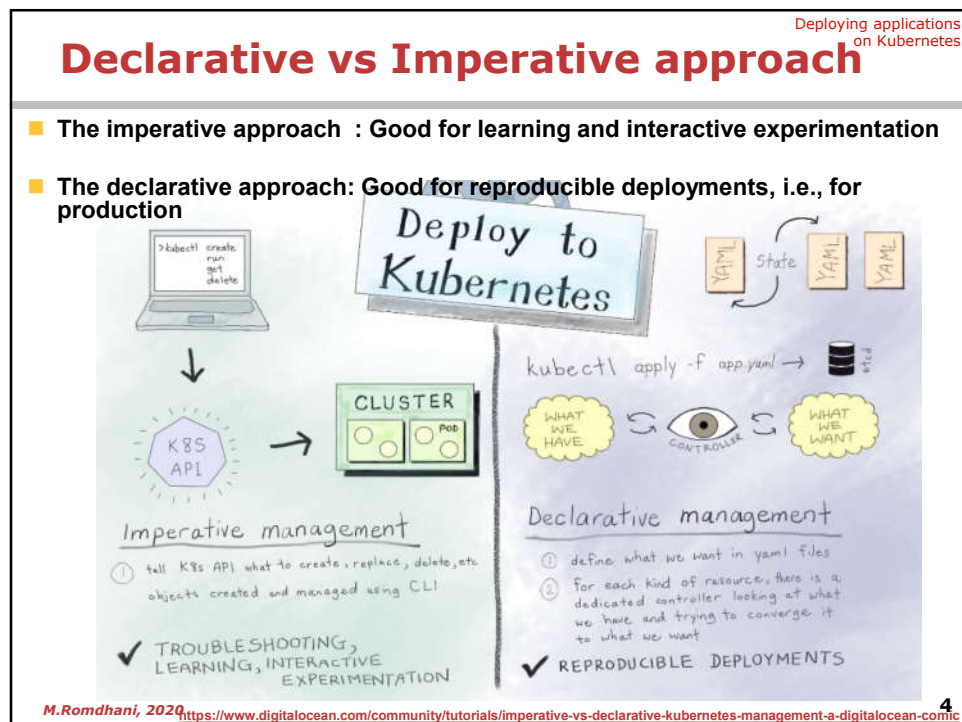
M.Romdhani, 2020

2

2

Application deployment using Kubectl

3



4

Managing Kubernetes Objects Using Imperative Commands

Deploying applications
on Kubernetes

■ Creating objects using Kubectl /imperative

- **run**: Create a new Deployment object to run Containers in one or more Pods.
- **expose**: Create a new Service object to load balance traffic across Pods.
- **autoscale**: Create a new Autoscaler object to automatically horizontally scale
- **create** <objecttype> [<subtype>] <instancename>. Some objects types have subtypes that you can specify in the create command. For example, the Service object has several subtypes including ClusterIP, LoadBalancer, and NodePort.

■ Updating objects using Kubectl /imperative

- **scale**: Horizontally scale a controller to add or remove Pods by updating the replica count of the controller.
- **annotate**: Add or remove an annotation from an object.
- **label**: Add or remove a label from an object.
- **set <field>**: Set an aspect of an object.
- **edit**: Directly edit the raw configuration of a live object by opening its configuration in an editor.
- **patch**: Directly modify specific fields of a live object by using a patch string.

■ Deleting objects using Kubectl /imperative

- **delete** <type>/<name>

M.Romdhani, 2020

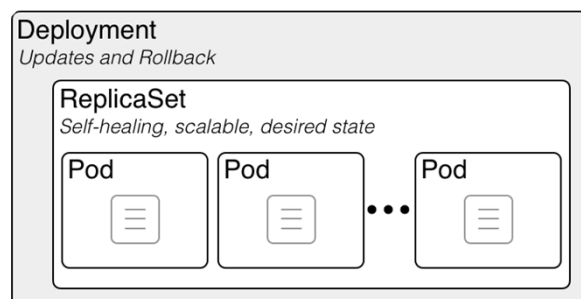
5

5

Deployment Objects

Deploying applications
on Kubernetes

- **Pods** are the basic, atomically deployable unit
- **ReplicaSet** are responsible for achieving and reconciling **the desired state** of an application service.
- **The Deployment** augments a **ReplicaSet** by providing rolling update and rollback functionality on top of it.



M.Romdhani, 2020

6

6

Create Pod using Kubectl/imperative

Deploying applications
on Kubernetes

■ To create a pod use **kubectl run** command

Create a pod named nginx-pod with image nginx

```
kubectl run nginx-pod --generator=run-pod/v1 --image=nginx
```

- This command generates pods that are not bound to a ReplicaSet or Deployment. These are called **Naked Pods**.
- `--generator=run-pod/v1` specifies the behavior of `kubectl run` command. If not specified `kubectl` generates a deployment rather than a pod.
- Hint for generating the yaml manifest `-o yaml,--dry-run` and output redirection

```
kubectl run nginx-pod7 --generator=run-pod/v1 --image=nginx
-o yaml --dry-run | Out-File mypod2.yml
```

■ Don't use naked Pods !

- Naked Pods will not be rescheduled in the event of a node failure.

M.Romdhani, 2020

7

7

Create ReplicaSet using Kubectl/imperative

Deploying applications
on Kubernetes

■ There is no way to create a ReplicaSet using the imperative approach. We should use the declarative approach.

- No generator exists for Replicasets [\[Ref\]](#).
 - Possible values for `--generator` flag : `--generator=run-pod/v1` for Pods, `--generator=run/v1` for Replication controllers, `--generator=deployment/v1beta1` for Deployments. Apart from `run-pod/v1` all other generators are deprecated since Kubernetes 1.12
- It is possible to create a Replication Controller using the imperative approach

```
kubectl run nginx-rc --generator=run/v1 --image=nginx --replicas=4
```

■ What is a Replication controller ?

- A replication controller is a resource that ensures a specified number of pod replicas are running at any one time. The Replication Controller is the original form of replication in Kubernetes. It has been replaced by Replica Sets.
- ReplicaSet have more options for the selector.
- A **Deployment** that configures a **ReplicaSet** is now the recommended way to set up replication. Replication Controllers are deprecated

M.Romdhani, 2020

8

8

Create Deployment using Kubectl/imperative

Deploying applications
on Kubernetes

■ To create a Deployment use **kubectl run** command

- # create a deployment named nginx-deploy with image nginx and 3 replicas
kubectl run nginx-deploy --image=nginx --replicas=3
- The Deployment creates by default the Replicaset and the pods

■ Examples

- # Start a single instance of hazelcast and let the container expose port 5701 .
kubectl run hazelcast --image=hazelcast --port=5701
- # Start a single instance of hazelcast and set environment variables "DNS_DOMAIN=cluster" and "POD_NAMESPACE=default" in the container.
kubectl run hazelcast --image=hazelcast --env="DNS_DOMAIN=cluster" --env="POD_NAMESPACE=default"
- # Start a single instance of hazelcast and set labels "app=hazelcast" and "env=prod" in the container.
kubectl run hazelcast --image=hazelcast --labels="app=hazelcast,env=prod"
- # Start a single instance of nginx, but overload the spec of the deployment with a partial set of values parsed from JSON.
kubectl run nginx --image=nginx --overrides='{ "apiVersion": "v1", "spec": { ... } }'
- # Start a pod of busybox and keep it in the foreground, don't restart it if it exits.
kubectl run -i -t busybox --image=busybox --restart=Never

M.Romdhani, 2020

9

9

Application deployment using Yaml

10

Deploying applications on Kubernetes

Yaml manifest structure

■ **Required fields**

- **apiVersion** - Which version of the Kubernetes API you're using to create this object
- **kind** - What kind of object you want to create
- **metadata** - Data that helps uniquely identify the object, including a name string, UID, and optional namespace
- **spec** - What state you desire for the object. The precise format of the object spec is different for every Kubernetes object

■ **The status field**

- While spec describes the desired state, the status describes the current state. It is added and updated continuously by K8s control plane.

```
kubectl get deploy mydepl -o yaml
```

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 2
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

11

M.Romdhani, 2020

11

Deploying applications on Kubernetes

kubectl apply vs create

■ **kubectl create -f whatever.yaml**

- creates resources if they don't exist
- if resources already exist, don't alter them (and display error message)

■ **kubectl apply -f whatever.yaml**

- creates resources if they don't exist
- if resources already exist, update them (to match the definition provided by the YAML file)
- stores the manifest as an annotation in the resource

12

M.Romdhani, 2020

12

Simple Pod Deployment

Deploying applications
on Kubernetes

Deployment steps

1. Describe the app using Kubernetes YAML (**my-nginx-pod.yaml**)
2. Run the deployment Command
kubectl apply -f my-nginx-pod.yaml
3. Make sure the pod has been created
kubectl get pods
4. Tear down your app
kubectl delete -f my-nginx-pod.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: mynginxapp
  labels:
    name: mynginxapp
spec:
  containers:
    - name: mynginxapp
      image: nginx
      ports:
        - containerPort: 80
```

M.Romdhani, 2020

13

13

Creating multiple resources

Deploying applications
on Kubernetes

- The manifest can contain multiple resources separated by ---

```
kind: ...
apiVersion: ...
metadata: ...
  name: ...
...
---
kind: ...
apiVersion: ...
metadata: ...
  name: ...
...
```

M.Romdhani, 2020

14

14

Simple Pod with namespace and labels

Deploying applications
on Kubernetes

■ 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.

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

M.Romdhani, 2020

15

15

A Multi container Pod: Main Container with Side Car Container

Deploying applications
on Kubernetes

■ Main Container and the Side Car Container share a Volume

```
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

16

16

Creating Desired State with replicaset and Deployments

17

Deploying applications
on Kubernetes

ReplicaSet Deployment

- **Saving this manifest into frontend.yaml and submitting it to a Kubernetes cluster will create the defined ReplicaSet and the Pods that it manages**
 - You can then get the current ReplicaSets deployed:
`kubectl get rs`
 - You can then get the current pods deployed:
`kubectl get pods`
- **If we try to delete a pod, the ReplicaSet will create a new one**
- **It is not recommended to manipulate replicaset directly, use deployment instead**

```

apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: frontend
  labels:
    app: guestbook
    tier: frontend
spec:
  # modify replicas according to your case
  replicas: 3
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      labels:
        tier: frontend
    spec:
      containers:
        - name: php-redis
          image: gcr.io/google_samples/gb-frontend:v3

```

Pod Template

M.Romdhani, 2020
18

18

Using Deployments

Deploying applications
on Kubernetes

- 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`

```
# 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
```

M.Romdhani, 2020

19

19

Updating the deployment

Deploying applications
on Kubernetes

- You can update the deployment by applying a new YAML file. This YAML file specifies that the deployment should be updated to use nginx 1.16.1

```
# 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:1.16.1
          ports:
            - containerPort: 80
```

M.Romdhani, 2020

20

20

Scaling the application by increasing the replica count

Deploying applications
on Kubernetes

- You can increase the number of pods in your Deployment by applying a new YAML file. This YAML file sets replicas to 4, which specifies that the Deployment should have four pods:

```
# for versions before 1.9.0 use apps/v1beta2
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
spec:
  selector:
    matchLabels:
      app: nginx
  replicas: 4 # Update the replicas from 2 to 4
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.16.1
          ports:
            - containerPort: 80
```

M.Romdhani, 2020

21

21

Managing resources for Containers and Pods

22

Managing Resources for Containers

Deploying applications
on Kubernetes

Why managing resources is important ?

- Within Kubernetes, containers are scheduled as pods. By default, a pod in Kubernetes will run with no limits on CPU and memory in a default namespace. This can create several problems related to contention for resources.

When you specify a Pod, you can optionally specify how much of each resource a Container needs.

- The most common resources to specify are CPU and memory (RAM)

Requests and Limits

- The **requests** is the amount guaranteed by the control plane.
 - Requests affect scheduling decisions !
- The **limits** are "hard limits". The container is not allowed to use more of that resource than the limit.

```
resources:
  requests:
    cpu: 100m
    memory: 300Mi
  limits:
    cpu: 1
    memory: 300Mi
```

M.Romdhani, 2020

23

23

Pod quality of service

Deploying applications
on Kubernetes

Each pod is assigned a QoS class (visible in status.qosClass).

■ If limits = requests:

- as long as the container uses less than the limit, it won't be affected
- If all containers in a pod have (limits=requests), QoS is considered "**Guaranteed**"

■ If requests < limits:

- as long as the container uses less than the request, it won't be affected
- otherwise, it might be killed/evicted if the node gets overloaded
- if at least one container has (requests<limits), QoS is considered "**Burstable**"

■ If a pod doesn't have specified any request nor limit, QoS is considered "**BestEffort**"

■ When a node is overloaded, BestEffort pods are killed first. Then, Burstable pods that exceed their limits.

■ If we only use Guaranteed pods, no pod should ever be killed (as long as they stay within their limits)

M.Romdhani, 2020

24

24

Don't use Memory Swap !

Deploying applications
on Kubernetes

- **The semantics of memory and swap limits on Linux cgroups are complex**
 - In particular, it's not possible to disable swap for a cgroup (the closest option is to reduce "swappiness")
- **The architects of Kubernetes wanted to ensure that Guaranteed pods never swap**
 - The only solution was to disable swap entirely !
- **If you don't care that pods are swapping, you can enable swap**

M.Romdhani, 2020

25

25

Managing Resources for Containers

Deploying applications
on Kubernetes

- **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.

M.Romdhani, 2020

```
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"
```

26

Managing Resources for Containers

Deploying applications on Kubernetes

A cluster node:

4x vCPUs 16GB RAM

1. The pod effective request
400MB of Memory
300Mi + 100Mi

2. Kubernetes assigns
1024 shares per core.
 $1024 / 0.1 = 1024$ shares
 $1024 / 0.5 = 512$ shares

The pod - Deployment.yaml

```
kind: Deployment
apiVersion: extensions/v1beta1
metadata:
  name: redis
  labels:
    name: redis-deployment
    app: example-voting-app
spec:
  replicas: 1
  selector:
    matchLabels:
      name: redis
      role: redisdb
  app: example-voting-app
  template:
    spec:
      containers:
        - name: redis
          image: redis:5.0.3-alpine
          resources:
            limits:
              memory: 600Mi
            requests:
              cpu: 1
              memory: 300Mi
              cpu: 500m
        - name: busybox
          image: busybox:1.28
          resources:
            limits:
              memory: 200Mi
              cpu: 300m
            requests:
              memory: 100Mi
              cpu: 100m
```

3. Will be killed if allocates > 600MB.
The whole Pod will fail.

4. Will be throttled if uses more than "1 Core".
1 core = 1000 millicores = 1000m =
100ms of computing time every 100 real ms
Full computing time of the node:
4 vCPUs * 100 real ms =
400ms of computing time = 4000m

5. Killed if allocates > 200MB.

6. Throttled if uses > 30ms of computing time in 100ms

27

27

Requests and Limits default values

Deploying applications on Kubernetes

- If we specify a limit without a request, the request is set to the limit.
- If we specify a request without a limit, there will be no limit (which means that the limit will be the size of the node)
 - Unless there are default values defined for our namespace!
- If we don't specify anything, the request is zero and the limit is the size of the node.
 - This is generally not what we want. A container without a limit can use up all the resources of a node
 - if the request is zero, the scheduler can't make a smart placement decision.

28

M.Romdhani, 2020

28

Defining min, max, and default resources using LimitRange

Deploying applications
on Kubernetes

- 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

```
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

29

29

Namespace Quotas

Deploying applications
on Kubernetes

- Quotas are enforced by creating a **ResourceQuota** object
- **ResourceQuota** objects are namespaced, and apply to their namespace only
- We can have multiple **ResourceQuota** objects in the same namespace
- The most restrictive values are used
- When a **ResourceQuota** is created, we can see how much of it is used:


```
kubectl describe resourcequota my-resource-quota
```

```
apiVersion: v1
kind: ResourceQuota
metadata:
  name: a-little-bit-of-compute
spec:
  hard:
    requests.cpu: "10"
    requests.memory: 10Gi
    limits.cpu: "20"
    limits.memory: 20Gi
```

M.Romdhani, 2020

30

30

Updating and Rolling back deployments

31

Creating a Deployment

Deploying applications
on Kubernetes

- Deployment provide **rollback functionality and update control**.

- Updates are managed through the pod-template-hash label.
- Each iteration creates a unique label that is assigned to both the ReplicaSet and subsequent Pods

- Creating a declarative deployment of nginx 1.14.2

- kubectl apply -f deploy1-14-2.yaml
- To see the Deployment Rollout status, run : `kubectl rollout status deploy nginx-deployment`
- To see the Replicaset(rs) run: `kubectl get rs`
- To see the labels automatically generated for each Pod, run: `kubectl get pods --show-labels`.

- The **pod-template-hash** label is added by the Deployment controller to every ReplicaSet that a Deployment creates or adopts.

M.Romdhani, 2020

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deployment
  labels:
    app: nginx
spec:
  replicas: 3
  selector:
    matchLabels:
      app: nginx
  template:
    metadata:
      labels:
        app: nginx
    spec:
      containers:
        - name: nginx
          image: nginx:1.14.2
          ports:
            - containerPort: 80
```

32

Updating a Deployment

Deploying applications
on Kubernetes

■ Let's follow the steps given below to update your Deployment:

1. Let's update the nginx Pods to use the nginx:1.16.1 image instead of the nginx:1.14.2 image.

```
kubectl set image deployment/nginx-deployment nginx=nginx:1.16.1 --record
```

- Alternatively, you can edit the Deployment and change

```
kubectl edit deployment.v1.apps/nginx-deployment
```

2. To see the rollout status, run:

```
kubectl rollout status deployment.v1.apps/nginx-deployment
```

- Get details of your Deployment and look at the events section

```
kubectl describe deployments
```

- When you updated the Deployment, it created a new and scaled it up to 1 and then scaled down the old ReplicaSet to 2, so that at least 2 Pods were available and at most 4 Pods were created at all times.
- It then continued scaling up and down the new and the old ReplicaSet, with the same rolling update strategy.
- Finally, you'll have 3 available replicas in the new ReplicaSet, and the old ReplicaSet is scaled down to 0

M.Romdhani, 2020

33

33

Recording deployment actions

Deploying applications
on Kubernetes

■ Some commands that modify a Deployment accept an optional **--record** flag

- Example: `kubectl set image deployment worker worker=alpine --record`

■ The flag will store the command line in the Deployment

- Technically, using the annotation `kubernetes.io/change-cause`
- It gets copied to the corresponding ReplicaSet (Allowing to keep track of which command created or promoted this ReplicaSet)

■ We can view this information with `kubectl rollout history`

■ Updating the annotation directly

```
kubectl annotate deployment worker kubernetes.io/change-cause="Just for fun"
```

- Check that our annotation shows up in the change history:

```
kubectl rollout history deployment worker
```

M.Romdhani, 2020

34

34

Pausing & Resuming

Deploying applications
on Kubernetes

- You can pause a Deployment before triggering one or more updates and then resume it.
 - This allows you to apply multiple fixes in between pausing and resuming without triggering unnecessary rollouts
 - Use Case example :
 - Pause a running deployment
 - Update the image (no rollout started)
 - Resume the Deployment and observe a new ReplicaSet coming up with all the new updates
- Pause by running the following command:


```
kubectl rollout pause deployment.v1.apps/nginx-deployment
```
- Resume the Deployment and observe a new ReplicaSet coming up with all the new updates:


```
kubectl rollout resume deployment.v1.apps/nginx-deployment
```

M.Romdhani, 2020

35

35

Deploying DaemonSets

Deploying applications
on Kubernetes

- 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

```
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

36

36

Rolling Back a Deployment

Deploying applications
on Kubernetes

- You may want to rollback a Deployment; for example, when the Deployment is not stable, such as crash looping.
 - By default, all of the Deployment's rollout history is kept in the system so that you can rollback anytime you want.
- Suppose that you made a typo while updating the Deployment, by putting the image name as `nginx:1.161` instead of `nginx:1.16.1`:


```
kubectl set image deployment.v1.apps/nginx-deployment nginx=nginx:1.161 --record=true
```

 - The output is similar to this:


```
deployment.apps/nginx-deployment image updated
```
 - The rollout gets stuck. You can verify it by checking the rollout status:


```
kubectl rollout status deployment.v1.apps/nginx-deployment
```
- Rolling Back to a Previous Revision
 - First, check the revisions of this Deployment:


```
kubectl rollout history deployment.v1.apps/nginx-deployment
```
 - Now you've decided to undo the current rollout and rollback to the previous revision:


```
kubectl rollout undo deployment.v1.apps/nginx-deployment
```
 - Check if the rollback was successful and the Deployment is running as expected, run:


```
kubectl get deployment nginx-deployment
```

M.Romdhani, 2020

37

37

Deployment Updates Strategies

Deploying applications
on Kubernetes

- **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** cycles through updating the Pods according to the parameters: **maxSurge** and **maxUnavailable**
- **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%**.
- **maxUnavailable**
 - Optional field that specifies the maximum number of Pods that can be unavailable during the update process. The default value is **25%**.

```
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

38

38