What is a rolling update in Kubernetes?

A **rolling update** in **Kubernetes** is a deployment strategy used to **gradually update** your application with **zero downtime** (or minimal downtime), by **replacing pods one at a time** with the new version.

How it works:

When you update a Deployment (for example, changing the container image version), Kubernetes:

1. **Creates a new pod** with the updated configuration.
2. **Waits** until the new pod is running and healthy.
3. **Terminates one of the old pods**.
4. **Repeats** the process until all old pods are replaced.

This way, your app **remains available** throughout the update.

Benefits:

- **No downtime** during updates.
- **Easy rollback** if something goes wrong.
- Supports controlled rollout using settings like:
    - `maxUnavailable`: max number of pods that can be down during update.
    - `maxSurge`: max number of extra pods that can be created during update.

🔧 Step 1: Create an initial deployment

Save this YAML as `deployment-v1.yaml`:

```
yaml
----------
apiVersion: apps/v1
kind: Deployment
metadata:
  name: myapp-deployment
spec:
  replicas: 3
  selector:
    matchLabels:
      app: myapp
  template:
    metadata:
      labels:
        app: myapp
    spec:
      containers:
        - name: myapp-container
          image: nginx:1.19 # Initial version
          ports:
            - containerPort: 80
```

Apply it:

```
bash
----------
kubectl apply -f deployment-v1.yaml
```

Check pods:

```
bash
----------
kubectl get pods -l app=myapp

kubectl get pods -l app=myapp -o wide
```

Check version

kubectl describe deployment myapp-deployment | grep Image

🔄 Step 2: Trigger a rolling update

Now update the image to a newer version (`nginx:1.21`).

You can do this using the `kubectl set image` command:

```
bash
----------
kubectl set image deployment/myapp-deployment myapp-container=nginx:1.21
```

📈 Step 3: Watch the rolling update

To see the update in progress:

```
bash
----------
```

```
kubectl rollout status deployment/myapp-deployment
```

This will show you how Kubernetes replaces the old pods one-by-one with new ones running the updated image.

Check the pods again:

```
bash
----------
kubectl get pods -l app=myapp -o wide
```

You'll see the old version getting terminated and new ones taking over.

⬅️ BACK Step 4: Rollback (if needed)

If something goes wrong, you can easily rollback:

```
bash
----------
kubectl rollout undo deployment/myapp-deployment
```

1. What is the main purpose of a rolling update in Kubernetes?

A) To delete all old pods at once and start fresh
B) To update the cluster nodes automatically
C) To update pods gradually with minimal or no downtime
D) To restart the Kubernetes API server

✅ **Correct Answer: C**

---

2. Which Kubernetes object supports rolling updates by default?

A) Pod
B) Service
C) Deployment
D) Namespace

✅ **Correct Answer: C**

---

3. During a rolling update, what does Kubernetes do first?

A) Deletes all old pods
B) Creates new pods with the updated version
C) Updates the kubelet
D) Scales down the deployment to zero

✅ **Correct Answer: B**

---

4. What does the `maxSurge` parameter define in a rolling update strategy?

A) The number of pods that can be updated in parallel
B) The maximum number of unavailable pods during the update
C) The number of extra pods allowed during the update
D) The delay between pod updates

✅ **Correct Answer: C**