🚀 Deployment Strategies – Theory & Step-by-Step Execution

Understanding deployment strategies helps ensure **high availability, low risk, and smooth releases** in production environments.

Below are the **4 most important deployment strategies**, explained with **theory + practical steps** 👇

🔷 Rolling/Rollout Deployment

📘 Theory:

Rolling deployment updates the application **incrementally** by replacing instances **one by one**, while keeping the service available.

Old and new versions run **simultaneously** during deployment.

⚠ Key Point:

- **Zero/minimal downtime**
- Default strategy in Kubernetes

🛠 Steps:

1. Create a Deployment with 3 replicas with any older image

```
controlplane:~$ kubectl create deployment mydep --image=nginx:1.24 --replicas=3
deployment.apps/mydep created
controlplane:~$ kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
mydep-7894647bfd-dfbct   1/1     Running   0          15s
mydep-7894647bfd-gc6js   1/1     Running   0          15s
mydep-7894647bfd-sxd7k   1/1     Running   0          15s
controlplane:~$
```

2. Now update or change the version of image in that Deployment

```
controlplane:~$ kubectl set image deployments mydep nginx=docker.io/nginx:1.25
deployment.apps/mydep image updated
```

3. And all pods will terminate one by one and new pods will be created

- As shown in below image

```
controlplane:~$ kubectl get pods -w
NAME                     READY   STATUS              RESTARTS   AGE
mydep-689c7c668b-q8zkq   0/1     ContainerCreating   0          6s
mydep-7894647bfd-dfbct   1/1     Running             0          73s
mydep-7894647bfd-gc6js   1/1     Running             0          73s
mydep-7894647bfd-sxd7k   1/1     Running             0          73s
mydep-689c7c668b-q8zkq   1/1     Running             0          8s
mydep-7894647bfd-sxd7k   1/1     Terminating         0          75s
mydep-7894647bfd-sxd7k   1/1     Terminating         0          75s
mydep-689c7c668b-zg4sm   0/1     Pending             0          0s
mydep-689c7c668b-zg4sm   0/1     Pending             0          0s
mydep-689c7c668b-zg4sm   0/1     ContainerCreating   0          0s
mydep-7894647bfd-sxd7k   1/1     Terminating         0          76s
mydep-689c7c668b-zg4sm   0/1     ContainerCreating   0          1s
mydep-7894647bfd-sxd7k   0/1     Completed           0          76s
mydep-689c7c668b-zg4sm   1/1     Running             0          2s
mydep-7894647bfd-sxd7k   0/1     Completed           0          77s
mydep-7894647bfd-sxd7k   0/1     Completed           0          77s
mydep-7894647bfd-dfbct   1/1     Terminating         0          77s
mydep-689c7c668b-kkzp9   0/1     Pending             0          0s
mydep-7894647bfd-dfbct   1/1     Terminating         0          77s
mydep-689c7c668b-kkzp9   0/1     Pending             0          0s
mydep-689c7c668b-kkzp9   0/1     ContainerCreating   0          1s
mydep-7894647bfd-dfbct   1/1     Terminating         0          78s
mydep-7894647bfd-dfbct   0/1     Completed           0          78s
mydep-689c7c668b-kkzp9   0/1     ContainerCreating   0          1s
mydep-7894647bfd-dfbct   0/1     Completed           0          78s
mydep-7894647bfd-dfbct   0/1     Completed           0          78s
mydep-689c7c668b-kkzp9   1/1     Running             0          2s
mydep-7894647bfd-gc6js   1/1     Terminating         0          79s
mydep-7894647bfd-gc6js   1/1     Terminating         0          79s
mydep-7894647bfd-gc6js   1/1     Terminating         0          80s
mydep-7894647bfd-gc6js   0/1     Completed           0          80s
mydep-7894647bfd-gc6js   0/1     Completed           0          80s
```

🔷 Recreate Deployment

📘 Theory:

Recreate deployment replaces the **entire old application version** with the **new version** by stopping all running instances first.
During deployment, the application remains **unavailable**.

🛠️ Steps:

1. Create a YAML file with following details

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: mydep
  name: mydep
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mydep
  strategy:
    type: Recreate
  template:
    metadata:
      labels:
        app: mydep
    spec:
      containers:
      - image: docker.io/nginx:1.24
        name: nginx
        resources: {}
status: {}
```

- Add strategy type = Recreate in spec.

2. Apply the YAML file and create a Deployment

```
controlplane:~$ kubectl apply -f mydep.yaml
deployment.apps/mydep configured
controlplane:~$ kubectl get pods
NAME                     READY   STATUS    RESTARTS   AGE
mydep-54f78b9b99-bvgm6   1/1     Running   0          6s
mydep-54f78b9b99-kbhsv   1/1     Running   0          6s
mydep-54f78b9b99-qd8rd   1/1     Running   0          101s
controlplane:~$
```

3. Now update the image with new version in Deployment

```
controlplane:~$ kubectl set image deployments mydep nginx=nginx:1.25
deployment.apps/mydep image updated
controlplane:~$
```

4. Check pods by “kubectl get pods -w”

- All pod will get terminated at once and New pod will get created

```
controlplane:~$ kubectl get pods -w
NAME                     READY   STATUS              RESTARTS   AGE
mydep-7894647bfd-hzw9k   1/1     Running             0          15s
mydep-7894647bfd-t78xf   1/1     Running             0          15s
mydep-7894647bfd-zqfp4   1/1     Running             0          15s
mydep-7894647bfd-hzw9k   1/1     Terminating         0          23s
mydep-7894647bfd-zqfp4   1/1     Terminating         0          23s
mydep-7894647bfd-t78xf   1/1     Terminating         0          23s
mydep-7894647bfd-hzw9k   1/1     Terminating         0          23s
mydep-7894647bfd-zqfp4   1/1     Terminating         0          23s
mydep-7894647bfd-t78xf   1/1     Terminating         0          23s
mydep-7894647bfd-hzw9k   1/1     Terminating         0          23s
mydep-7894647bfd-hzw9k   0/1     Completed           0          23s
mydep-7894647bfd-t78xf   1/1     Terminating         0          23s
mydep-7894647bfd-zqfp4   1/1     Terminating         0          23s
mydep-7894647bfd-t78xf   0/1     Completed           0          23s
mydep-7894647bfd-zqfp4   0/1     Completed           0          23s
mydep-7894647bfd-zqfp4   0/1     Completed           0          24s
mydep-7894647bfd-zqfp4   0/1     Completed           0          24s
mydep-7894647bfd-hzw9k   0/1     Completed           0          24s
mydep-7894647bfd-hzw9k   0/1     Completed           0          24s
mydep-7894647bfd-t78xf   0/1     Completed           0          24s
mydep-7894647bfd-t78xf   0/1     Completed           0          24s
mydep-867d69ffc5-kpzsl   0/1     Pending             0          0s
mydep-867d69ffc5-kpzsl   0/1     Pending             0          0s
mydep-867d69ffc5-zh9js   0/1     Pending             0          0s
mydep-867d69ffc5-7wr8x   0/1     Pending             0          0s
mydep-867d69ffc5-zh9js   0/1     Pending             0          0s
mydep-867d69ffc5-kpzsl   0/1     ContainerCreating   0          0s
mydep-867d69ffc5-7wr8x   0/1     Pending             0          0s
mydep-867d69ffc5-zh9js   0/1     ContainerCreating   0          0s
mydep-867d69ffc5-7wr8x   0/1     ContainerCreating   0          0s
mydep-867d69ffc5-kpzsl   0/1     ContainerCreating   0          0s
mydep-867d69ffc5-zh9js   0/1     ContainerCreating   0          1s
mydep-867d69ffc5-7wr8x   0/1     ContainerCreating   0          1s
mydep-867d69ffc5-7wr8x   1/1     Running             0          2s
mydep-867d69ffc5-zh9js   1/1     Running             0          2s
mydep-867d69ffc5-kpzsl   1/1     Running             0          2s
```

⚠ **Key Point:**

- Causes **downtime**
- Simple but risky for production

🔷 **Canary Deployment**

📘 **Theory:**

Canary deployment releases the new version to a **small group of users first**, then gradually increases traffic if everything works correctly.

🛠️ **Steps:**

1. Create a YAML file with following details

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: mycan1
  name: mycan1
spec:
  replicas: 7
  selector:
    matchLabels:
      app: mycan1
      class: unnati
  strategy: {}
  template:
    metadata:
      labels:
        app: mycan1
        class: unnati
    spec:
      containers:
      - image: hashicorp/http-echo
        name: http-echo
        args:
        - "-text=version1"
        - "-listen=:5678"
        resources: {}
status: {}
```

- **Class = unnati**
- **args = version1**
- **Replicas = 7**

2. Apply the YAML file and create a deployment “mycan1”

```
controlplane:~$ kubectl apply -f mycan1.yaml
deployment.apps/mycan1 unchanged
controlplane:~$ kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
mycan1-797f74f8d-494n6    1/1     Running   0          17s
mycan1-797f74f8d-h2zf5    1/1     Running   0          17s
mycan1-797f74f8d-jcwsx    1/1     Running   0          17s
mycan1-797f74f8d-szb8p    1/1     Running   0          17s
mycan1-797f74f8d-tmtxj    1/1     Running   0          17s
mycan1-797f74f8d-v5thv    1/1     Running   0          17s
mycan1-797f74f8d-xmzxt    1/1     Running   0          17s
controlplane:~$
```

3. Create a 2nd YAML file with the same details only change in name and args use same class = unnati.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  labels:
    app: mycan2
  name: mycan2
spec:
  replicas: 3
  selector:
    matchLabels:
      app: mycan2
      class: unnati
  strategy: {}
  template:
    metadata:
      labels:
        app: mycan2
        class: unnati
    spec:
      containers:
      - image: hashicorp/http-echo
        name: http-echo
        args:
        - "-text=version2"
        - "-listen=:5678"
        resources: {}
status: {}
```

- **Replicas = 2**
- **Args = version2**
- **Class = unnati**

```
controlplane:~$ kubectl apply -f mycan2.yaml
deployment.apps/mycan2 created
controlplane:~$ kubectl get pods
NAME                      READY   STATUS    RESTARTS   AGE
mycan1-797f74f8d-494n6    1/1     Running   0          5m24s
mycan1-797f74f8d-h2zf5    1/1     Running   0          5m24s
mycan1-797f74f8d-jcwsx    1/1     Running   0          5m24s
mycan1-797f74f8d-szb8p    1/1     Running   0          5m24s
mycan1-797f74f8d-tmtxj    1/1     Running   0          5m24s
mycan1-797f74f8d-v5thv    1/1     Running   0          5m24s
mycan1-797f74f8d-xmzxt    1/1     Running   0          5m24s
mycan2-84c5d47fb8-4d9w2   1/1     Running   0          7s
mycan2-84c5d47fb8-t5vmz   1/1     Running   0          7s
mycan2-84c5d47fb8-v57bg   1/1     Running   0          7s
controlplane:~$
```

4. Create SVC with following details.

```yaml
apiVersion: v1
kind: Service
metadata:
  labels:
    class: unnati
  name: mycansvc
spec:
  ports:
  - port: 5678
    protocol: TCP
    targetPort: 5678
  selector:
    class: unnati
status:
  loadBalancer: {}
```

- **Give label as class = unnati**

```
controlplane:~$ kubectl apply -f svc.yaml
service/mycansvc created
controlplane:~$
```

4. Copy the SVC ip and "curl" it and you can see that 70% traffic git hitting version1 (mycan1) and 30% to version2 (mycan2).

```
controlplane:~$ kubectl get svc
NAME         TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)    AGE
kubernetes   ClusterIP   10.96.0.1       <none>        443/TCP    20d
mycansvc     ClusterIP   10.105.113.65   <none>        5678/TCP   2m28s
controlplane:~$ curl 10.105.113.65:5678
version1
controlplane:~$ curl 10.105.113.65:5678
version2
controlplane:~$ curl 10.105.113.65:5678
version1
controlplane:~$ curl 10.105.113.65:5678
version1
controlplane:~$ curl 10.105.113.65:5678
version1
controlplane:~$ curl 10.105.113.65:5678
version1
controlplane:~$ curl 10.105.113.65:5678
version2
controlplane:~$ curl 10.105.113.65:5678
version2
controlplane:~$ curl 10.105.113.65:5678
version2
controlplane:~$ curl 10.105.113.65:5678
version2
controlplane:~$ curl 10.105.113.65:5678
version1
controlplane:~$ curl 10.105.113.65:5678
version1
controlplane:~$ curl 10.105.113.65:5678
version1
controlplane:~$ curl 10.105.113.65:5678
version1
controlplane:~$ curl 10.105.113.65:5678
version1
controlplane:~$ curl 10.105.113.65:5678
version1
controlplane:~$
```

⚠ **Key Point:**

- **Lowest risk strategy**
- Needs monitoring & traffic control tools

🔷 Blue-Green Deployment

📘 Theory:

Blue-Green deployment maintains **two identical environments**:

- **Blue** → current production version
- **Green** → new application version

Traffic is switched to Green **only after successful testing**.

🛠️ Steps:

1. Create Blue deployment with following details

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: blue
spec:
  replicas: 2
  selector:
    matchLabels:
      app: httpd
      version: blue
  template:
    metadata:
      labels:
        app: httpd
        version: blue
    spec:
      containers:
      - name: httpd
        image: httpd:2.4
        ports:
        - containerPort: 80
        volumeMounts:
        - name: html-volume
          mountPath: /usr/local/apache2/htdocs/index.html
          subPath: index.html
      volumes:
      - name: html-volume
        configMap:
          name: blue-html
```

2. Create Green deployment with following details.

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: green
spec:
  replicas: 2
  selector:
    matchLabels:
      app: httpd
      version: green
  template:
    metadata:
      labels:
        app: httpd
        version: green
    spec:
      containers:
      - name: httpd
        image: httpd:2.4
        ports:
        - containerPort: 80
        volumeMounts:
        - name: html-volume
          mountPath: /usr/local/apache2/htdocs/index.html
          subPath: index.html
      volumes:
      - name: html-volume
        configMap:
          name: green-html
```

3. Create config map for blue deployment with following details.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: blue-html
data:
  index.html: |
    <html>
      <body>
        <h1 style="color:blue">Welcome to BLUE Deployment!</h1>
      </body>
    </html>
```

4. Create config map for green deployment with following details.

```yaml
apiVersion: v1
kind: ConfigMap
metadata:
  name: green-html
data:
  index.html: |
    <html>
      <body>
        <h1 style="color:green">Welcome to GREEN Deployment!</h1>
      </body>
    </html>
```

5. Get SVC ip and do "curl" you will able to see the Blue deployment is working

```
controlplane:~$ kubectl get svc
NAME         TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE
httpd-svc    LoadBalancer   10.97.2.2       <pending>     80:31119/TCP   6m3s
kubernetes   ClusterIP      10.96.0.1       <none>        443/TCP        20d
mycansvc     ClusterIP      10.105.113.65   <none>        5678/TCP       34m
controlplane:~$ curl 10.97.2.2
<html>
  <body>
    <h1 style="color:blue">Welcome to BLUE Deployment!</h1>
  </body>
</html>
controlplane:~$
```

6. Now do changes in SVC change the selector to "green"

```yaml
spec:
  allocateLoadBalancerNodePorts: true
  clusterIP: 10.97.2.2
  clusterIPs:
  - 10.97.2.2
  externalTrafficPolicy: Cluster
  internalTrafficPolicy: Cluster
  ipFamilies:
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - nodePort: 31119
    port: 80
    protocol: TCP
    targetPort: 80
  selector:
    app: httpd
    version: green
  sessionAffinity: None
  type: LoadBalancer
status:
  loadBalancer: {}
```

7. Now you "curl" the same sec ip and you will able to see Green deployment.

```
controlplane:~$ kubectl edit svc httpd-svc
service/httpd-svc edited
controlplane:~$ curl 10.97.2.2
<html>
  <body>
    <h1 style="color:green">Welcome to GREEN Deployment!</h1>
  </body>
</html>
controlplane:~$
```

⚠ **Key Point:**

- **Instant rollback**
- Requires double infrastructure

🎯 **Conclusion:**

Each deployment strategy has its own **use case**:

- Recreate → Simplicity
- Rolling → Continuous availability
- Blue-Green → Safe & fast rollback
- Canary → Controlled risk

A good DevOps engineer chooses the **right strategy at the right time**.