

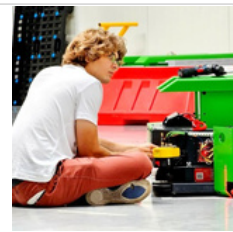
Kubernetes Deployment Strategies: Choose Rolling Update, Canary, or Blue-Green.

Check GitHub for helpful DevOps tools:

Michael Robotics

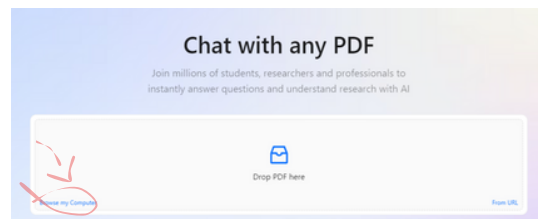
Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

 <https://github.com/MichaelRobotics>



Ask Personal AI Document assistant to learn interactively (FASTER)!

- 1 Download PDF
 - 2 Go to website
 - 3 Browse file
 - 4 Chat with Document
- 1 <https://github.com/MichaelRobotics/DevOpsTools/blob/main/KubernetesDeploy.pdf>
- 2 Click there to go to ChatPdf website
- 3
- 4 Ask questions about document!



Completly new to Linux and Networking?

Essential for this PDF is a thorough knowledge of networking. I highly recommend the HTB platform's networking module, which offers extensive information to help build a comprehensive understanding.

HTB - Your Cyber Performance Center

We provide a human-first platform creating and maintaining high performing cybersecurity individuals and organizations.

 <https://www.hackthebox.com/>



What is Kubernetes?

Kubernetes is an open-source platform that automates the deployment, scaling, and management of containerized applications. It helps manage clusters of nodes running containers, ensuring efficient and reliable operation.

How Kubernetes clusters are made?

Kubernetes clusters consist of a control plane and multiple worker nodes. The control plane manages cluster operations, while worker nodes run the actual container workloads.

Why and When use Kubernetes

Kubernetes is ideal for deploying scalable, resilient, and automated containerized applications. It is used when managing multiple containers across different environments is necessary.

Example: Running a microservices-based e-commerce platform that scales up during peak hours.

System Requirements

- RAM: 2 GB per node (1 GB can work for testing but may lead to limited performance)
- 10 GB free storage
- Ubuntu

Kubernetes: Main components & packages

- **kube-apiserver:** Central management component that exposes the Kubernetes API; acts as the front-end for the cluster.
- **etcd:** Distributed key-value store for storing all cluster data, ensuring data consistency across nodes.
- **kube-scheduler:** Assigns pods to available nodes based on resource requirements and policies.
- **kube-controller-manager:** Manages core controllers that handle various functions like node status, replication, and endpoints.
- **kubelet:** Agent that runs on each node, responsible for managing pods and their containers.
- **kube-proxy:** Manages networking on each node, ensuring communication between pods and services within the cluster.

Kubernetes Deployment Strategies: Rolling Update

1)Intro

Kubernetes (K8s) deployment strategies are crucial for managing application updates with minimal disruption, aiming for near-zero downtime

Common K8s deployment strategies include:

- **Rolling Updates:** Updates pods one by one, keeping the app available.
- **Blue-Green Deployments:** Runs new and old versions side-by-side for quick switching after verification.
- **Canary Releases:** Diverts a small user segment to the new version for testing before full deployment.

2)Rolling update

Rolling update is default kubernetes deployment strategy. During a rolling update, Kubernetes manages the rollout by creating new pods before terminating old ones. The process is controlled by `maxUnavailable` and `maxSurge` parameters:

- **maxUnavailable:** The maximum number of pods that can be unavailable during the update.
- **maxSurge:** The maximum number of extra pods that can be created above the desired number of replicas during the update.

Lets deploy 4 nginx replicas:

```
kubectl create deployment nginx-deployment --image=nginx:latest --replicas=4
```

Now check deployment configuration:

```
kubectl edit deployment
```

```
  app: nginx
  name: nginx
  namespace: default
  resourceVersion: "3485"
  uid: 1fe616fd-d8c9-4514-93e2-896c998bcc47
spec:
  progressDeadlineSeconds: 600
  replicas: 4
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: nginx
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
```

As we can see, maxSurge and maxUnavailable is set to 25% for default. Lets change image to httpd and check how pod will be updated:

```
spec:
  progressDeadlineSeconds: 600
  replicas: 4
  revisionHistoryLimit: 10
  selector:
    matchLabels:
      app: nginx-deployment
  strategy:
    rollingUpdate:
      maxSurge: 25%
      maxUnavailable: 25%
    type: RollingUpdate
  template:
    metadata:
      creationTimestamp: null
    labels:
      app: nginx-deployment
    spec:
      containers:
        - image: httpd
```

Pods are getting created. Parameter maxUnavailable is set to 25%, so in case of 4 replicas one of them can be unavailable. Parameter maxSurge set to 25% also, so there can be 5 replicas available during update:

NAME	READY	STATUS	RESTARTS	AGE
nginx-deployment-bc64dffbd-c4sr7	1/1	Running	0	40s
nginx-deployment-bc64dffbd-nznqt	1/1	Running	0	38s
nginx-deployment-bc64dffbd-s4l7g	1/1	Running	0	40s
nginx-deployment-c7c784555-nzxrv	0/1	ContainerCreating	0	2s
nginx-deployment-c7c784555-vj9x5	0/1	ContainerCreating	0	2s
controlplane \$ kubectl get pods				
NAME	READY	STATUS	RESTARTS	AGE
nginx-deployment-bc64dffbd-c4sr7	1/1	Running	0	41s
nginx-deployment-c7c784555-f5qmw	0/1	ContainerCreating	0	1s
nginx-deployment-c7c784555-nzxrv	1/1	Running	0	3s
nginx-deployment-c7c784555-vj9x5	1/1	Running	0	3s
nginx-deployment-c7c784555-wn6wf	0/1	ContainerCreating	0	1s
controlplane \$ kubectl get pods				
NAME	READY	STATUS	RESTARTS	AGE
nginx-deployment-bc64dffbd-c4sr7	0/1	Completed	0	43s
nginx-deployment-c7c784555-f5qmw	1/1	Running	0	3s
nginx-deployment-c7c784555-nzxrv	1/1	Running	0	5s
nginx-deployment-c7c784555-vj9x5	1/1	Running	0	5s
nginx-deployment-c7c784555-wn6wf	1/1	Running	0	3s
controlplane \$ kubectl get pods				
NAME	READY	STATUS	RESTARTS	AGE
nginx-deployment-c7c784555-f5qmw	1/1	Running	0	4s
nginx-deployment-c7c784555-nzxrv	1/1	Running	0	6s
nginx-deployment-c7c784555-vj9x5	1/1	Running	0	6s
nginx-deployment-c7c784555-wn6wf	1/1	Running	0	4s

In case of invalid image tag:

```
CreationTimestamp: null
labels:
  app: nginx
spec:
  containers:
  - image: httpd:v4
    imagePullPolicy: Always
    name: nginx
    resources: {}
```

Created containers will loop in ImageLoopBackoff forever:

```
controlplane $ kubectl create deploy nginx --image=nginx
deployment.apps/nginx created
controlplane $ kubectl get pods
NAME          READY   STATUS             RESTARTS   AGE
nginx-676b6c5bbc-zmj7p  0/1     ContainerCreating   0           5s
controlplane $ kubectl get pods
NAME          READY   STATUS             RESTARTS   AGE
nginx-676b6c5bbc-zmj7p  0/1     ContainerCreating   0           8s
controlplane $ kubectl get pods
NAME          READY   STATUS   RESTARTS   AGE
nginx-676b6c5bbc-zmj7p  1/1     Running   0           9s
controlplane $ kubectl get pods
```

Kubernetes Deployment Strategies: Canary

1) Intro

Canary deployment is a strategy that allows new versions of software to be released gradually in a production environment, starting with a small subset of users (e.g., 20%). This minimizes risk by exposing potential issues before a full-scale rollout.

2) Why Not Just Use a Stage Environment?

Stage environments can't fully replicate production because they lack real user interactions and traffic complexities. This means some issues only appear when the update is live.

How It Works

1. **Initial Rollout:** A small percentage of users are routed to the new version (v2) while others stay on the stable version (v1). This is managed by load balancers.
2. **Monitoring:** Key metrics and user feedback are monitored for errors or performance issues.
3. **Decision Point:** If v2 performs well, it's gradually rolled out to more users. If issues arise, the update can be paused or rolled back.
4. **Scaling Up:** Once proven stable, the deployment expands to all users.

3) Management and Tools

Load balancers, monitoring tools (like Prometheus or Datadog), and automated rollback systems ensure a smooth process. This approach enables real-world testing and continuous improvement based on actual user feedback.

4) Deployment

Create your main deployment and service

```
curl -o app-v1.yaml
```

```
https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/DeploymentStrategies/canary/app-v1.yaml
```

This is the main deployment of your application with the service that will be used to route to it

```
# Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: production
  labels:
    app: production
spec:
  replicas: 1
  selector:
    matchLabels:
      app: production
  template:
    metadata:
      labels:
        app: production
[...]
kind: Service
metadata:
  name: production
  labels:
    app: production
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
      name: http
  selector:
    app: production
```


apply

```
kubectl apply -f app-v1.yaml
```

Create the canary deployment and service.

```
curl -o ingress-canary.yaml
```

```
https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/DeploymentStrategies/canary/ingress-canary.yaml
```

This is the canary deployment that will take a weighted amount of requests instead of the main deployment

```
# Deployment
apiVersion: apps/v1
kind: Deployment
metadata:
  name: canary
  labels:
    app: canary
spec:
  replicas: 1
  selector:
[...]
```

```
# Service
apiVersion: v1
kind: Service
metadata:
  name: canary
  labels:
    app: canary
spec:
  ports:
    - port: 80
      targetPort: 80
      protocol: TCP
      name: http
  selector:
    app: canary
```

apply

```
kubectl apply -f ingress-canary.yaml
```

Create Ingress Pointing To Your Main Deployment

```
curl -o canary-deployment.yaml
```

```
https://raw.githubusercontent.com/user/repository/branch/path/to/canary-deployment.yaml
```

Next you will need to expose your main deployment with an ingress resource, note there are no canary specific annotations on this ingress

```
# Ingress
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: production
  annotations:
spec:
  ingressClassName: nginx
  rules:
  - host: echo.prod.mydomain.com
    http:
      paths:
      - pathType: Prefix
        path: /
        backend:
          service:
            name: production
            port:
              number: 80
```

apply

```
kubectl apply -f canary-deployment.yaml
```

Create Ingress Pointing To Your Canary Deployment

```
curl -o ingress-canary.yaml
```

```
https://raw.githubusercontent.com/MichaelRobotics/Kubernetes/main/DeploymentStrategies/canary/ingress-canary.yaml
```

- The host name is identical to the main ingress host name
- The `nginx.ingress.kubernetes.io/canary: "true"` annotation is required and defines this as a canary annotation (if you do not have this the Ingresses will clash)
- The `nginx.ingress.kubernetes.io/canary-weight: "50"` annotation dictates the weight of the routing, in this case there is a "50%" chance a request will hit the canary deployment over the main deployment

```
# Ingress
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: canary
  annotations:
    nginx.ingress.kubernetes.io/canary: \"true\"
    nginx.ingress.kubernetes.io/canary-weight: \"50\"
spec:
  ingressClassName: nginx
  rules:
  - host: echo.prod.mydomain.com
    http:
      paths:
      - pathType: Prefix
        path: /
        backend:
          service:
            name: canary
            port:
              number: 80
```

uncomment delete `\"` in `nginx.ingress.kubernetes.io/canary` and `canary-weight`

Testing your setup

You can use the following command to test your setup (replacing INGRESS_CONTROLLER_IP with your ingress controllers IP Address)

```
for i in $(seq 1 10); do curl -s --resolve echo.prod.mydomain.com:80:$INGRESS_CONTROLLER_IP echo.prod.mydomain.com | grep "Hostname"; done
```

You will get the following output showing that your canary setup is working as expected:

```
Hostname: production-5c5f65d859-phqzc
Hostname: canary-6697778457-zkfjf
Hostname: canary-6697778457-zkfjf
Hostname: production-5c5f65d859-phqzc
Hostname: canary-6697778457-zkfjf
Hostname: production-5c5f65d859-phqzc
Hostname: production-5c5f65d859-phqzc
Hostname: production-5c5f65d859-phqzc
Hostname: canary-6697778457-zkfjf
Hostname: production-5c5f65d859-phqzc
```

Kubernetes Deployment Strategies: Blue & Green

1) Blue-Green Deployment

Blue-Green Deployment involves maintaining two identical environments: "Blue" (current version) and "Green" (new version). Traffic is switched from Blue to Green once the new version is fully tested. If issues arise, traffic can quickly be switched back to the Blue environment, ensuring minimal downtime.

2) Difference from Canary Deployment

- Traffic Routing: Blue-Green switches all traffic at once to the new version, while Canary gradually shifts traffic (e.g., 10% to the new version).
- Risk Management: Blue-Green provides an immediate rollback option to the old version, whereas Canary manages risk incrementally.
- Cost: Blue-Green is more expensive due to the need to run two environments simultaneously.

3) When to Use Blue-Green

- Fast Recovery: Ideal for companies needing quick recovery in case of failures. It ensures zero downtime and rapid rollback.
- Costly: Requires double the infrastructure, making it more expensive than other deployment strategies like Canary.

4) Deploy Blue green

Follow canary instructions, but don't download and apply canary-ingress.

All operation is based on changing service, to which ingress points:

Blue:

```
# Ingress
apiVersion: networking.k8s.io/v1
kind: Ingress
[...]
- pathType: Prefix
  path: /
  backend:
    service:
      name: canary
      port:
        number: 80
```

Green:

```
# Ingress
apiVersion: networking.k8s.io/v1
kind: Ingress
[...]
- pathType: Prefix
  path: /
  backend:
    service:
      name: production
      port:
        number: 80
```

common troubleshooting

1) Canary Deployment Not Routing Traffic Correctly

Cause: Misconfigured annotations or wrong traffic weight.

Solution: Check canary ingress annotations (nginx.ingress.kubernetes.io/canary: "true" and canary-weight). Use `kubectl describe ingress <ingress-name>` to verify configuration.

2) Blue-Green Deployment Failing to Switch Traffic

Cause: Routing or load balancer issues.

Solution: Verify routing with `kubectl get services` and ensure the ingress is directing traffic to the green environment. Check load balancer configuration.

3) Rollout Stuck During Deployment

Cause: Pod readiness/liveness probe failures or resource limits.

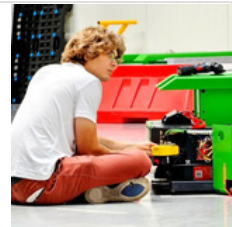
Solution: Check pod status (`kubectl get pods`) and use `kubectl describe pod <pod-name>` to identify issues. Adjust probe and resource configurations as needed.

4) Check my Kubernetes Troubleshooting series:

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

<https://github.com/MichaelRobotics>




Learn more about Kubernetes

Check Kubernetes and piyushsachdeva - great docs!

Setup a Multi Node Kubernetes Cluster

kubeadm is a tool to bootstrap the Kubernetes cluster

 <https://github.com/piyushsachdeva/CKA-2024/tree/main/Resources/Day27>



Kubernetes Documentation

This section lists the different ways to set up and run Kubernetes

 <https://kubernetes.io/docs/setup/>



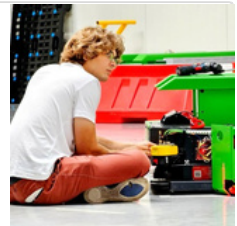
Share, comment, DM and check GitHub for scripts & playbooks created to automate process.

Check my GitHub

Michael Robotics

Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats skill information overload by adhering to the set of principles: simplify, prioritize, and execute.

<https://github.com/MichaelRobotics>



PS.

If you need a playbook or bash script to manage KVM on a specific Linux distribution, feel free to ask me in the comments or send a direct message!