

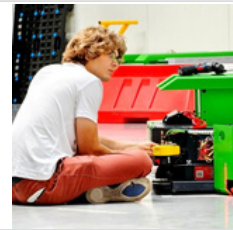
Kubernetes Ingress: Deploy python app with NGINX Ingress, docker & helm

Check GitHub for helpful DevOps tools:

Michael Robotics

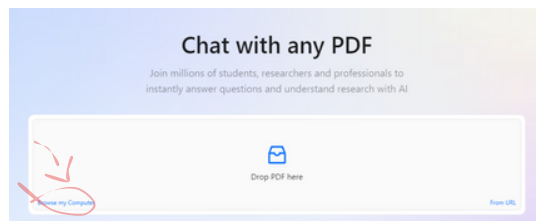
Hi, I'm Michal. I'm a Robotics Engineer and DevOps enthusiast. My mission is to create skill-learning platform that combats information overload by adhering to the set of principles: simplify, prioritize, and execute.

 <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/KubernetesIngress.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 Ingress: intro to NGINX Ingress

1) What is kubernetes Ingress?

An Ingress Controller in Kubernetes is a specialized component that manages external access to services within a Kubernetes cluster. It interprets and applies rules defined by Ingress resources to control how incoming HTTP and HTTPS traffic is routed to different services based on the rules specified. Here's a breakdown of its core functions:

2) Why to use kubernetes Ingress?

An Ingress Controller in Kubernetes provides efficient, cost-effective, and flexible traffic management by centralizing access to services within a cluster:

1. **Cost Savings:** It minimizes expenses by allowing multiple services to share a single external load balancer, unlike the costly LoadBalancer service provided by cloud providers.
2. **Efficient Traffic Routing:** Supports advanced routing, SSL termination, and load balancing, simplifying complex routing needs under a single IP address.
3. **Enhanced Security:** Manages HTTPS, SSL certificates, and access control, adding a secure layer around Kubernetes applications.
4. **Centralized Management:** Enables scalable, consistent rule management across services, ideal for complex architectures.

An Ingress Controller streamlines access, enhances security, and reduces cloud costs, making it a practical alternative to individual LoadBalancer services.

3) How it works?

Ingress refers to external traffic that enters your Kubernetes cluster. But how is this traffic managed within Kubernetes?

Kubernetes use ingress resource, which is defined by the user through yaml file:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: minimal-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx-example
  rules:
  - http:
      paths:
      - path: /testpath
        pathType: Prefix
        backend:
          service:
            name: test
            port:
              number: 80
```

Deploying an Ingress resource in Kubernetes sets rules for routing external traffic to services in your cluster, managed by the Ingress Controller.

By default, the Ingress Controller requests a cloud-provided load balancer, but changing its service type to NodePort allows it to create a load balancer within the cluster instead.

Kubernetes Ingress: Setup on killerkoda

1) Configure docker registry

navigate to website:

Develop faster. Run anywhere.

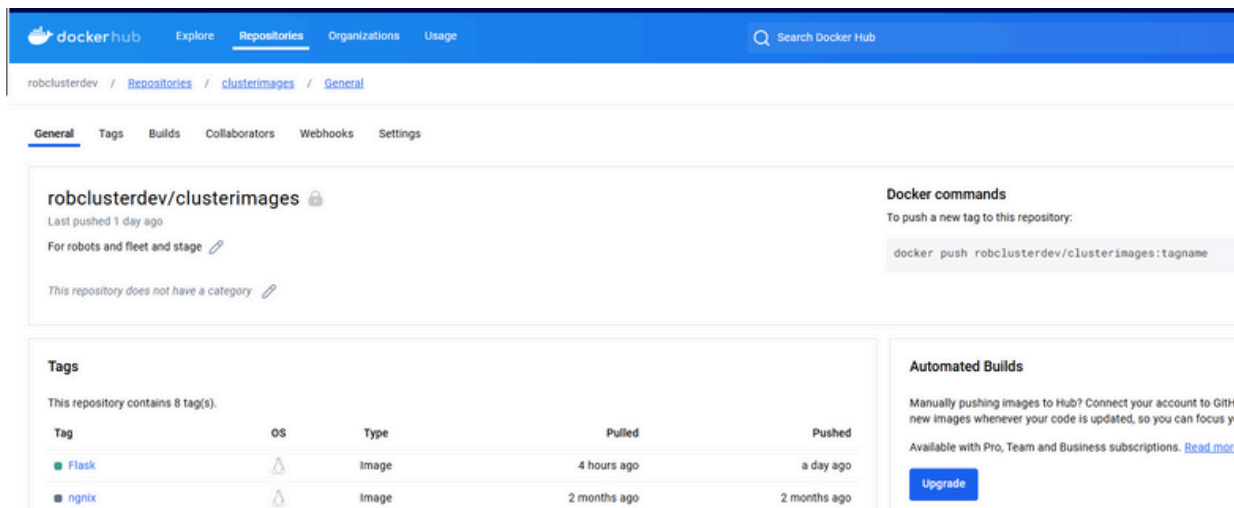
Build with the #1 most-used developer tool

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



And configure your private repo. Each user can create 1 free private repository.

Most important part to remember is your repo name, your username and password.



The screenshot shows the Docker Hub interface for the repository `robclusterdev/clusterimages`. The page includes a navigation bar with links to Explore, Repositories, Organizations, and Usage. The repository page has tabs for General, Tags, Builds, Collaborators, Webhooks, and Settings. The General tab is active, showing the repository name, last pushed time (1 day ago), and a description: "For robots and fleet and stage". It also displays a table of tags.

Tag	OS	Type	Pulled	Pushed
Flask	Linux	Image	4 hours ago	a day ago
nginx	Linux	Image	2 months ago	2 months ago

On the right side, there are sections for "Docker commands" (showing `docker push robclusterdev/clusterimages:tagname`) and "Automated Builds" (with an "Upgrade" button).

My repo name is: **robclusterdev/clusterimages**

Full app image tag: **robclusterdev/clusterimages:Flask**


2) Sign on killercoda

Go to killercoda website and enter your lab.

follow steps provided in link:

1.31 | Two Node

This playground will always have the latest Kubeadm Kubernetes

 <https://killercoda.com/playgrounds/course/kubernetes-playgrounds/two-node>

K L L K
C O D A

3) Download app

Go to my GitHub or just clone:

```
git clone https://github.com/MichaelRobotics/Kubernetes.git
```

4) Build image in directory where Dockerfile is located /Kubernetes/Ingress/Flask:

my docker repo is named: robclusterdev/clusterimages

```
docker build -t <repo_name_with_tag> .
```

in my case:

```
docker build -t robclusterdev/clusterimages:Flask .
```

Login to docker registry and follow instructions:

```
docker login
```

Push image to your repo:

```
docker push <repo_name_with_tag>
```

5) Configure kubernetes secrets and login

To allow Kubernetes to pull images from your private Docker repository, create a secret with your registry credentials.

```
kubectl create secret docker-registry my-dockerhub-secret \
  --docker-username=<your-username> \
  --docker-password=<your-password> \
  --docker-email=<your-email>
```

Modify deployment file accordingly:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
  labels:
    app: hello-world
spec:
  replicas: 1
  selector:
    matchLabels:
      app: hello-world
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
        - name: hello-world
          image: robclusterdev/clusterimages:Flask
          ports:
            - containerPort: 80
      imagePullSecrets:
        - name: my-dockerhub-secret
```

We added your Docker registry & secret name

6) Deploy deployment and service

Deployment should go accordingly:

```
controlplane $ ls
Flask deployment.yaml ingress.yaml minimal-ingress.yaml secret.sh service.yaml
controlplane $ kubectl apply -f deployment.yaml
deployment.apps/hello-world created
controlplane $ kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
hello-world-96c86ccd9-hfdjl        0/1     ContainerCreating   0           6s
controlplane $ kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
hello-world-96c86ccd9-hfdjl        0/1     ContainerCreating   0           8s
controlplane $ kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
hello-world-96c86ccd9-hfdjl        0/1     ContainerCreating   0           9s
controlplane $ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hello-world-96c86ccd9-hfdjl        1/1     Running   0           11s
```

Then deploy service. Important to notice is a selector. It should point towards deployed container.

```
apiVersion: v1
kind: Service
metadata:
  name: hello-world
spec:
  selector:
    app: hello-world
  ports:
    - protocol: TCP
      port: 80
      targetPort: 80
```

Then deploy service manifest file and check its functionality. Should go as follows:

```
controlplane $ kubectl apply -f service.yaml
service/hello-world created
controlplane $ kubectl get svc
NAME      TYPE        CLUSTER-IP    EXTERNAL-IP  PORT(S)    AGE
hello-world ClusterIP   10.97.2.130   <none>       80/TCP     12s
kubernetes ClusterIP   10.96.0.1     <none>       443/TCP    25d
controlplane $ curl 10.97.2.130
Hello, World!controlplane $
```

7) Create ingress resource

Deploy ingress manifest file:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: hello-world
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
  - host: "example.com"
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: hello-world
            port:
              number: 80
```

As specified in the manifest file, the NGINX controller routes traffic to the hello-world service when the DNS example.com is accessed.

```
controlplane $ kubectl apply -f deployment.yaml
deployment.apps/hello-world created
controlplane $ kubectl get pods
NAME                                READY   STATUS             RESTARTS   AGE
hello-world-96c86ccd9-btpx2        0/1     ContainerCreating   0           8s
controlplane $ kubectl get pods
NAME                                READY   STATUS    RESTARTS   AGE
hello-world-96c86ccd9-btpx2        1/1     Running   0           10s
```

8) Install helm and ingress controller

install helm

```
$ curl -fsSL -o get_helm.sh
https://raw.githubusercontent.com/helm/helm/main/scripts/get-helm-3
$ chmod 700 get_helm.sh
$ ./get_helm.sh
```

Install Ingress

```
helm upgrade --install ingress-nginx ingress-nginx \
--repo https://kubernetes.github.io/ingress-nginx \
--namespace ingress-nginx --create-namespace
```

Proper installation ends like this:

```
# This section is only required if TLS is to be enabled for the Ingress
tls:
  - hosts:
    - www.example.com
    secretName: example-tls

If TLS is enabled for the Ingress, a Secret containing the certificate and key must also be provided:

apiVersion: v1
kind: Secret
metadata:
  name: example-tls
  namespace: foo
data:
  tls.crt: <base64 encoded cert>
  tls.key: <base64 encoded key>
type: kubernetes.io/tls
controlplane $ █
```

9) Configure ingress controller

Enter ingress controller configuration and change its service type to NodePort from LoadBalancer

to get service name:

```
kubectl get svc -n ingress-nginx
```

to edit service:

```
kubectl edit svc -n ingress-nginx
```

Then change Loadbalancer type to NodePort

```
protocol: TCP
targetPort: https
selector:
  app.kubernetes.io/component: controller
  app.kubernetes.io/instance: ingress-nginx
  app.kubernetes.io/name: ingress-nginx
sessionAffinity: None
type: NodePort
status:
  loadBalancer: {}
- apiVersion: v1
  kind: Service
  metadata:
```

Additionally, check if your service has an assigned IP address. If it does, your ingress controller is configured correctly.

```
controlplane $ kubectl get svc -n ingress-nginx
NAME                                TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)                                AGE
ingress-nginx-controller            NodePort    10.99.24.123   <none>         80:31862/TCP,443:31573/TCP           18m
ingress-nginx-controller-admission ClusterIP    10.96.140.225 <none>         443/TCP                               18m
controlplane $
```

nginx-controller got CLUSTER-IP so our loadbalancer works

10) Resolve DNS

curl loadbalancer CLUSTER-IP:

```
controlplane $ kubectl get svc -n ingress-nginx
NAME                                TYPE        CLUSTER-IP    EXTERNAL-IP    PORT(S)          AGE
ingress-nginx-controller            NodePort    10.99.24.123   <none>         80:31862/TCP,443:31573/TCP 18m
ingress-nginx-controller-admission ClusterIP    10.96.140.225 <none>         443/TCP          18m
controlplane $ curl 10.99.24.123
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
controlplane $
```

Accessing the service IP won't yield a response from the load balancer, as it only responds when example.com is reached. We need to ensure that example.com resolves to this IP address.

Now curl it with proper resolve:

```
curl --resolve example.com:80:10.99.24.123 http://example.com
```

Now everything should work:

```
controlplane $ curl 10.99.24.123
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
controlplane $ curl --resolve example.com:80:10.99.24.123 http://example.com
Hello, World!controlplane $
```

To make change permanent, edit **/etc/hosts** and add DNS record:

```
127.0.0.1 localhost

# The following lines are desirable for IPv6 capable hosts
::1 ip6-localhost ip6-loopback
fe00::0 ip6-localnet
ff00::0 ip6-mcastprefix
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
ff02::3 ip6-allhosts
127.0.0.1 ubuntu
127.0.0.1 host01
127.0.0.1 controlplane
172.30.2.2 node01
10.99.24.123 example.com
```

restart network-manager

```
sudo systemctl restart network-manager
```

You can access loadbalancer through example.com now!

```
curl http://example.com
```

common troubleshooting

1) Ingress Route Not Working / 404 Error

Cause: Incorrect Ingress resource configuration or missing hostname.

2) Ingress Controller Not Deploying

Cause: Incorrect Helm chart installation or lack of permissions.

3) CrashLoopBackOff for Python App Pods

Cause: Application misconfiguration or compatibility issue.

4) Service Not Exposed by Ingress

Cause: Incorrect service type or misconfigured Ingress resource.

5) SSL Certificate Not Working

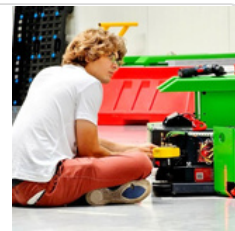
Cause: Missing TLS secret or hostname mismatch in certificate.

6) 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/>



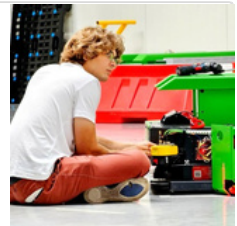
Share, comment, DM and check GitHub for scripts & playbooks created to automate process.

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!