NGINX Ingress Controller®

Mastering NGINX as a Kubernetes Ingress Controller: A Step-by-Step Guide

Davoud Azari

Introduction

Setting up a **Kubernetes cluster** with multiple services can be exciting, but managing incoming traffic is always a challenge.

How do you expose services properly? Should you configure a separate **LoadBalancer** for each one? How do you handle **SSL/TLS**? How can you set up routing rules without adding unnecessary complexity?

One solution is using an **Ingress Controller**. Among the available options, **NGINX Ingress Controller** provides a way to simplify HTTP(S) request routing while improving security.

In this guide, we'll go through the **installation, configuration, and management of NGINX Ingress Controller in Kubernetes**. We'll also explore some practical approaches for traffic management, security, and monitoring.

In This Guide, We'll Cover:

- What NGINX and an Ingress Controller are
- Why you need an Ingress Controller
- How to set up NGINX as an Ingress Controller
- Configuring routing rules for services
- Best practices for managing ingress in production environments

Understanding NGINX and Kubernetes Ingress

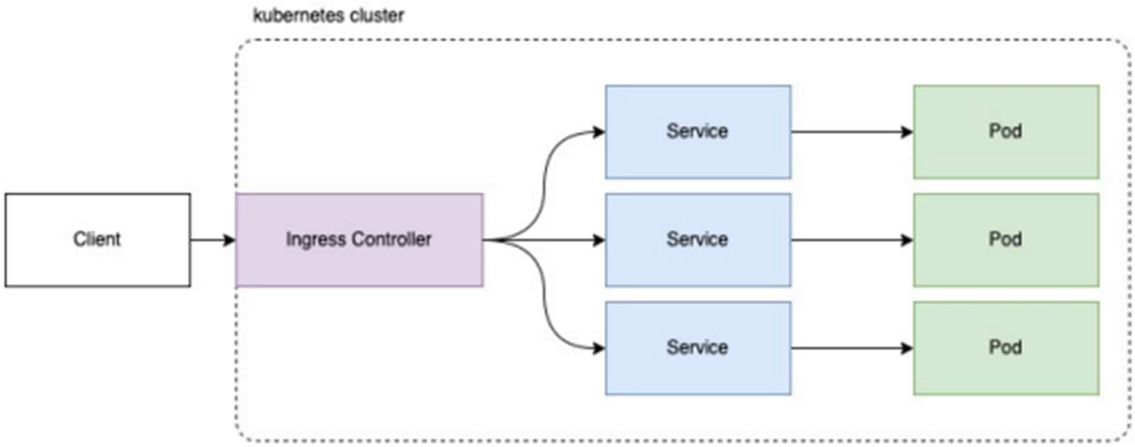
What is NGINX?

NGINX is a high-performance, open-source web server that functions as a:
- Reverse proxy
- Load balancer
- API gateway
- Caching mechanism

Its versatility makes it a perfect fit for Kubernetes environments.

Link: https://www.f5.com/glossary/nginx

What is an Ingress Controller?



Ingress controller and services relation

An Ingress Controller is a traffic manager that directs external HTTP(S) requests to services inside a Kubernetes cluster based on predefined routing rules.

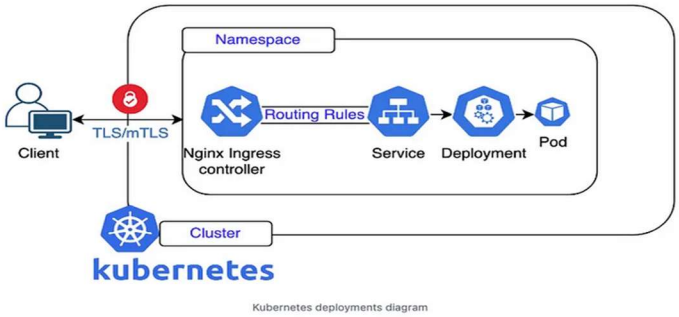
Without an Ingress Controller:
- Each service must be exposed separately using NodePorts or LoadBalancers (not scalable).
- Managing SSL/TLS certificates manually is complex.
- Routing and network traffic must be manually configured for every service.

NGINX Ingress Controller simplifies all of this by managing the flow of traffic through a single entry point.

Setting Up NGINX as a Kubernetes Ingress Controller

Prerequisites:

- A running Kubernetes cluster (Minikube, K3s, or a cloud-managed Kubernetes service)
- kubectl and Helm installed



Kubernetes deployments diagram

Step 1: Installing the NGINX Ingress Controller

The easiest way to install NGINX Ingress Controller is using Helm:

```
helm repo add ingress-nginx https://kubernetes.github.io/ingress-nginx
helm repo update
helm install my-nginx ingress-nginx/ingress-nginx
```

Once installed, check if the NGINX pods are running:

```
kubectl get pods -n default
```

Ensure the controller is exposed correctly:

```
kubectl get svc -n default
```

Step 2: Deploying Sample Services

Now, let's create a simple FastAPI-based application that NGINX will route traffic to.

Dockerfile for FastAPI Application

```
FROM python:3.9
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8080"]
```

requirements.txt (Dependencies)

```
fastapi
pydantic
uvicorn
```

main.py (FastAPI Application)

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"message": "Hello from FastAPI service"}
```

Now, build and push the Docker image:

```
docker build -t my-fastapi-app .
docker tag my-fastapi-app myrepo/my-fastapi-app:v1
docker push myrepo/my-fastapi-app:v1
```

Step 3: Configuring the Ingress Resource

Now, let's define an Ingress resource to route requests to the correct backend service.

Example Ingress Configuration (ingress.yaml)

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: fastapi-ingress
spec:
  ingressClassName: nginx
  rules:
  - host: fastapi.local
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: fastapi-service
            port:
              number: 80
```

Step 4: Testing the Ingress Controller

To test if NGINX is correctly routing traffic, update your /etc/hosts file:

```
127.0.0.1 fastapi.local
```

Then, run:

```
curl http://fastapi.local
```

Best Practices for Managing NGINX Ingress

To optimize NGINX performance and security, follow these best practices:

- Enable HTTPS/TLS with Cert-Manager for automatic SSL certificate handling.
- Implement Rate Limiting & Security to prevent DDoS attacks.
- Ensure Load Balancing & High Availability by running multiple NGINX replicas.
- Monitor Traffic & Logs with Prometheus and Grafana.

Troubleshooting Common Issues

If your Ingress isn't working correctly, check these common issues:

- Ensure the Ingress Controller pod is running:

```
kubectl get pods -n ingress-nginx
```

- Check if the NGINX service is exposed properly:

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

- View Ingress rules to verify correct configuration:

```
kubectl get ingress -n default
```

- Debug logs for issues:

```
kubectl logs -n ingress-nginx -l app.kubernetes.io/name=ingress-nginx
```

Conclusion

* Why Kubernetes needs an Ingress Controller
* How to set up NGINX as an Ingress Controller
* Configuring and testing an Ingress resource
* Best practices for managing NGINX in production environments