Create an Ingress Resource for External Access to Your Application

Table of Contents

Introduction

In Kubernetes, an **Ingress** resource defines rules that allow external access to services within a cluster. It acts as a gateway, managing traffic and routing requests based on defined rules. In this lab, you will set up an Ingress resource in a Minikube environment to provide external access to a sample web application.

Problem Statement

As applications within a Kubernetes cluster grow, the need for external access becomes paramount. Traditional methods, such as NodePort Services, can expose services, but they lack advanced routing capabilities. Ingress solves this problem by providing a single entry point to your applications, allowing for better management and easier access.

Prerequisites

Completion of all previous lab guides (up to Lab Guide-06) is required before proceeding with Lab Guide-07.

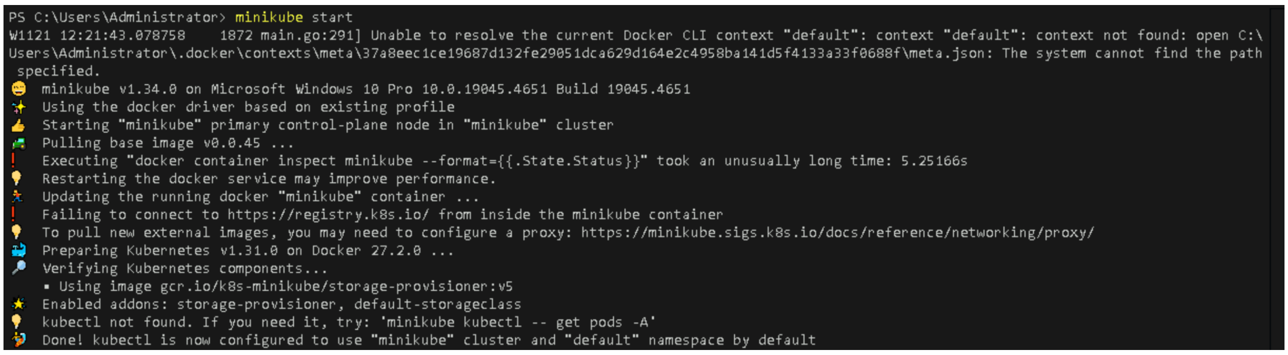
- **Minikube** installed on your Windows machine.
- **kubectl** command-line tool installed and configured to communicate with your Minikube cluster.
- Ensure your Minikube version is compatible (at least Kubernetes v1.19).

Setup Instructions

Step 1: Create a Minikube Cluster

1. **Start Minikube**
   Open PowerShell or Command Prompt as Administrator and run:

```
minikube start
```



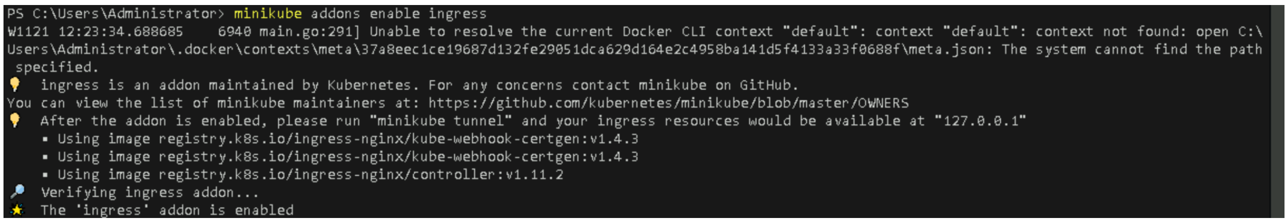
```
PS C:\Users\Administrator> minikube start
W1121 12:21:43.078758    1872 main.go:291] Unable to resolve the current Docker CLI context "default": context "default": context not found: open C:\
Users\Administrator\.docker\contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33f0688f\meta.json: The system cannot find the path
 specified.
😄  minikube v1.34.0 on Microsoft Windows 10 Pro 10.0.19045.4651 Build 19045.4651
✨  Using the docker driver based on existing profile
👍  Starting "minikube" primary control-plane node in "minikube" cluster
🚜  Pulling base image v0.0.45 ...
❗  Executing "docker container inspect minikube --format={{.State.Status}}" took an unusually long time: 5.25166s
💡  Restarting the docker service may improve performance.
🏃  Updating the running docker "minikube" container ...
❗  Failing to connect to https://registry.k8s.io/ from inside the minikube container
💡  To pull new external images, you may need to configure a proxy: https://minikube.sigs.k8s.io/docs/reference/networking/proxy/
🐳  Preparing Kubernetes v1.31.0 on Docker 27.2.0 ...
🔎  Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🌟  Enabled addons: storage-provisioner, default-storageclass
💡  kubectl not found. If you need it, try: 'minikube kubectl -- get pods -A'
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace by default
```

This command will create a local Kubernetes cluster.

Step 2: Enable the Ingress Controller

1. **Enable NGINX Ingress Controller**
   Run the following command in your terminal:

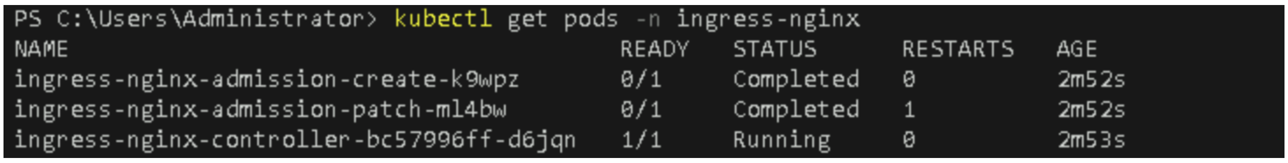
```
minikube addons enable ingress
```



```
PS C:\Users\Administrator> minikube addons enable ingress
W1121 12:23:34.688685    6940 main.go:291] Unable to resolve the current Docker CLI context "default": context "default": context not found: open C:\
Users\Administrator\.docker\contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33f0688f\meta.json: The system cannot find the path
 specified.
💡  ingress is an addon maintained by Kubernetes. For any concerns contact minikube on GitHub.
You can view the list of minikube maintainers at: https://github.com/kubernetes/minikube/blob/master/OWNERS
💡  After the addon is enabled, please run "minikube tunnel" and your ingress resources would be available at "127.0.0.1"
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.3
    ▪ Using image registry.k8s.io/ingress-nginx/kube-webhook-certgen:v1.4.3
    ▪ Using image registry.k8s.io/ingress-nginx/controller:v1.11.2
🔎  Verifying ingress addon...
🌟  The 'ingress' addon is enabled
```

2. **Verify the Ingress Controller is Running**
   After a few moments, verify that the Ingress controller is running with:

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



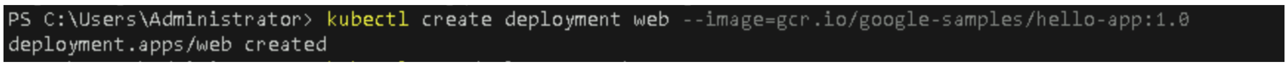
```
PS C:\Users\Administrator> kubectl get pods -n ingress-nginx
NAME                                        READY   STATUS      RESTARTS   AGE
ingress-nginx-admission-create-k9wpz        0/1     Completed   0          2m52s
ingress-nginx-admission-patch-ml4bw         0/1     Completed   1          2m52s
ingress-nginx-controller-bc57996ff-d6jqn    1/1     Running     0          2m53s
```

You should see the NGINX Ingress controller listed and running.

Step 3: Deploy a Hello World Application

1. **Create a Deployment**
   Deploy a sample Hello World application:

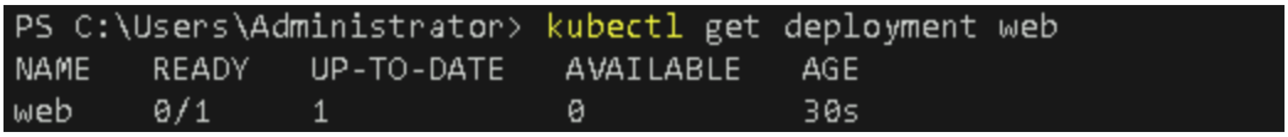
```
kubectl create deployment web --image=gcr.io/google-samples/hello-app:1.0
```



```
PS C:\Users\Administrator> kubectl create deployment web --image=gcr.io/google-samples/hello-app:1.0
deployment.apps/web created
```

Confirm the deployment is successful by running:

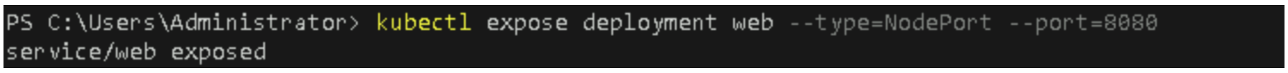
```
kubectl get deployment web
```



```
PS C:\Users\Administrator> kubectl get deployment web
NAME   READY   UP-TO-DATE   AVAILABLE   AGE
web    0/1     1            0           30s
```

2. **Expose the Deployment**
   Expose the deployment with NodePort:

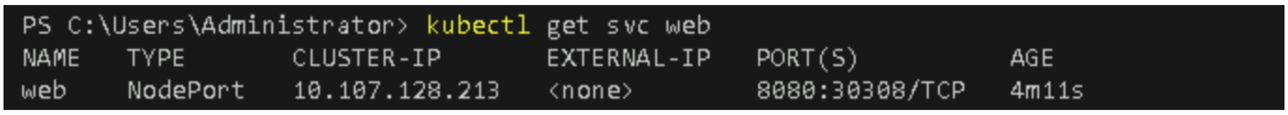
```
kubectl expose deployment web --type=NodePort --port=8080
```



```
PS C:\Users\Administrator> kubectl expose deployment web --type=NodePort --port=8080
service/web exposed
```

3. **Verify the Service**
   Check the service to ensure it's available:

```
kubectl get svc web
```



```
PS C:\Users\Administrator> kubectl get svc web
NAME   TYPE       CLUSTER-IP       EXTERNAL-IP   PORT(S)          AGE
web    NodePort   10.107.128.213   <none>        8080:30308/TCP   4m11s
```

   Note the `NodePort` assigned to the service.

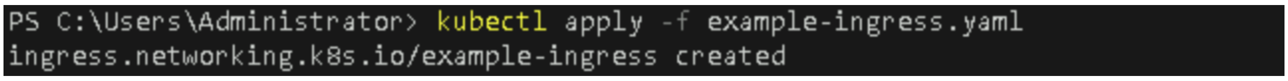
Step 4: Create the Ingress Resource

1. **Create an Ingress YAML File**
   Create a file named `example-ingress.yaml` with the following content:

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: example-ingress
  annotations:
    nginx.ingress.kubernetes.io/rewrite-target: /
spec:
  ingressClassName: nginx
  rules:
    - host: hello-world.example
      http:
        paths:
          - path: /
            pathType: Prefix
            backend:
              service:
                name: web
                port:
                  number: 8080
```

2. **Apply the Ingress Resource**
   Use the following command to create the Ingress:

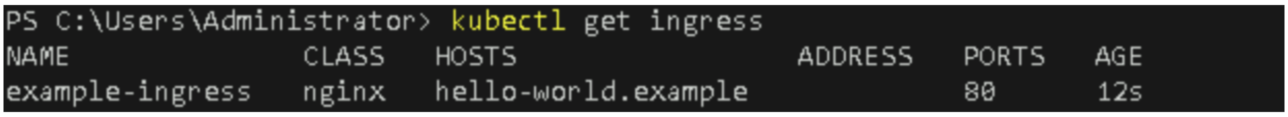
```
kubectl apply -f example-ingress.yaml
```



```
PS C:\Users\Administrator> kubectl apply -f example-ingress.yaml
ingress.networking.k8s.io/example-ingress created
```

3. **Verify the Ingress**
   Check the status of the Ingress:

```
kubectl get ingress
```



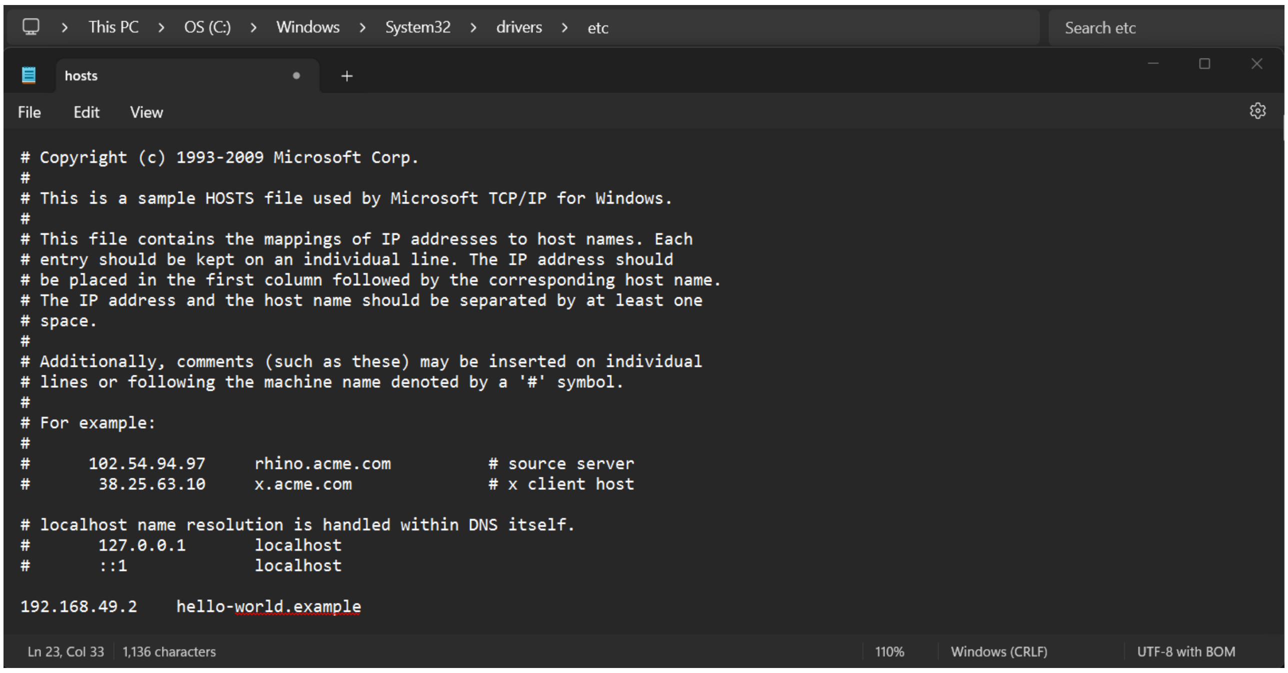
```
PS C:\Users\Administrator> kubectl get ingress
NAME              CLASS   HOSTS                 ADDRESS   PORTS   AGE
example-ingress   nginx   hello-world.example             80      12s
```

   Note: It may take a minute to assign an address.

Step 5: Test the Ingress

1. **Update the Hosts File**

   Open the `hosts` file with administrative rights (located at `C:\Windows\System32\drivers\etc\hosts`) and add the following line using the IP address returned by `minikube ip`:



```
# Copyright (c) 1993-2009 Microsoft Corp.
#
# This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
#
# This file contains the mappings of IP addresses to host names. Each
# entry should be kept on an individual line. The IP address should
# be placed in the first column followed by the corresponding host name.
# The IP address and the host name should be separated by at least one
# space.
#
# Additionally, comments (such as these) may be inserted on individual
# lines or following the machine name denoted by a '#' symbol.
#
# For example:
#
#      102.54.94.97     rhino.acme.com          # source server
#       38.25.63.10     x.acme.com              # x client host

# localhost name resolution is handled within DNS itself.
#       127.0.0.1       localhost
#       ::1             localhost

192.168.49.2    hello-world.example
```

This PC > OS (C:) > Windows > System32 > drivers > etc
Search etc
hosts
File Edit View
Ln 23, Col 33 | 1,136 characters
110%
Windows (CRLF)
UTF-8 with BOM

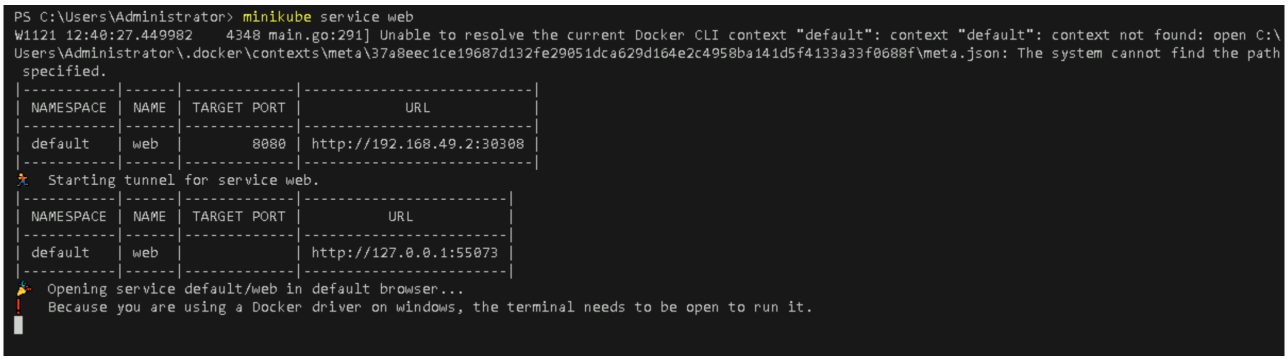
```
<minikube-ip> hello-world.example
```

Replace `<minikube-ip>` with the actual IP address from the previous command.

2. **Test the Ingress**

   Visit the Service via NodePort, using the `minikube service` command:

```
minikube service web
```

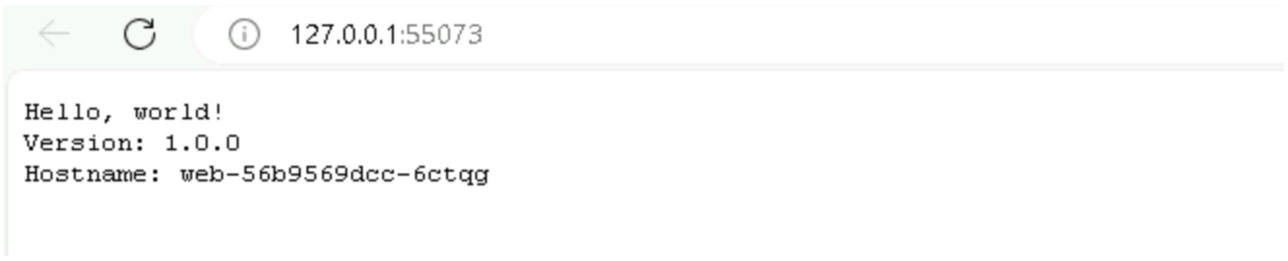


```
PS C:\Users\Administrator> minikube service web
W1121 12:40:27.449982    4348 main.go:291] Unable to resolve the current Docker CLI context "default": context "default": context not found: open C:\
Users\Administrator\.docker\contexts\meta\37a8eec1ce19687d132fe29051dca629d164e2c4958ba141d5f4133a33f0688f\meta.json: The system cannot find the path
 specified.
|-----------|------|-------------|---------------------------|
| NAMESPACE | NAME | TARGET PORT |            URL            |
|-----------|------|-------------|---------------------------|
| default   | web  |        8080 | http://192.168.49.2:30308 |
|-----------|------|-------------|---------------------------|
🏃  Starting tunnel for service web.
|-----------|------|-------------|------------------------|
| NAMESPACE | NAME | TARGET PORT |          URL           |
|-----------|------|-------------|------------------------|
| default   | web  |             | http://127.0.0.1:55073 |
|-----------|------|-------------|------------------------|
🎉  Opening service default/web in default browser...
❗  Because you are using a Docker driver on windows, the terminal needs to be open to run it.
```

The above command will open the `sample application` .

The output is similar to:

```
Hello, world!
Version: 1.0.0
Hostname: web-56b9569dcc-48zbn
```



```
Hello, world!
Version: 1.0.0
Hostname: web-56b9569dcc-6ctqg
```

> **Note:** On Linux System we can test ingress using `Minikube IP` address instead of Localhost i.e., `127.0.0.1`

References

- Kubernetes Ingress Documentation
- Minikube Ingress Documentation
