Kubernetes Ingress & Ingress Controller - Complete Setup

In kubernetes, the importance of controlling traffic in and out of a cluster is very crucial. This is where kubernetes ingress can offer many benefits.

In this article we will focus on what is kubernetes Ingress and Ingress controller and how it works in detail.

Overview:

- **What is an Kubernetes Ingress**
- **What is an Ingress Controller**
- **How to create an Ingress resource**
- **How to create an Ingress controller**

What is an Kubernetes Ingress ?

In kubernetes, an ingress is an object that allows access to your kubernetes services from outside the kubernetes cluster. You can configure access by creating a collection of rules that define which inbound connection reach which services. The key components of ingress are as follows:

- **Ingress Controller**: This is a specialized load balancer that watches the Ingress resource and processes the rules defined within it. It is responsible for routing incoming traffic to the appropriate services, based on the Ingress rules.
- **Ingress Resource**: This is a Kubernetes resource that defines the rules for routing incoming traffic. It specifies the hostname, path, and destination services for different types of requests.

- **Ingress Rules**: These are the rules defined within the Ingress resource that determine how incoming traffic should be routed. Rules can be based on host, path, or different criteria.

What is an Ingress controller ?

An ingress controller is an specialized load balancer for kubernetes that manages external access to services within a cluster. It watches for the ingress resources and configures the necessary routing rules. Depending on the configuration it supports path based or host based routing, SSL/TLS termination.

Let us take an example and see how kubernetes ingress works.

Example:

Here we will create kubernetes manifests such as deployment, service and ingress resources.

Pre-requisites:

- **kubectl**
- **docker**
- **minikube**

Let us create the manifest files.

1. **Create a deployment.yaml file and paste the below content**

```
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: balav8/hello-world:latest
        ports:
        - containerPort: 80
```

Save it and apply it using below command

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

2. **Create a service.yaml file and paste the below content**

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

Save it and apply it using below command

```
kubectl apply -f service.yaml
```

3. **Check if the deployment and service is running**

```
kubectl get all
```

```
NAME                                 READY   STATUS    RESTARTS   AGE
pod/hello-world-7bfdd9576d-ppwcr     1/1     Running   0          19m

NAME                  TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
service/hello-world   ClusterIP   10.105.155.144   <none>        80/TCP    19m
service/kubernetes    ClusterIP   10.96.0.1        <none>        443/TCP   26m

NAME                          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/hello-world   1/1     1            1           19m

NAME                                       DESIRED   CURRENT   READY   AGE
replicaset.apps/hello-world-7bfdd9576d     1         1         1       19m
```

We can see the pod is running and we have a service running too, now we can check if the service is running.

4. **To check if the service is running use the below command**

```
curl 10.105.155.144
```

```
ubuntu@ip-172-31-39-161:~/CKA-2024/Resources/Day33/Flask/k8s$ curl 10.105.155.144
Hello, World!ubuntu@ip-172-31-39-161:~/CKA-2024/Resources/Day33/Flask/k8s$
```

Our application is **accessible**.

5. **Create an ingress resource using ingress.yaml file and paste the below content.**

```
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
```

Save it and apply using below command

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

6. **Check the ingress resource**

```
kubectl get ing
```

You can see that the address field is empty, that is because we haven't created an ingress controller.

Once the ingress controller is installed, it will check for the ingress resource and it will assign the address accordingly.

7. **To install nginx ingress controller run the below command**

```
kubectl apply -f https://raw.githubusercontent.com/kubernetes/i
```

```
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
serviceaccount/ingress-nginx-admission created
role.rbac.authorization.k8s.io/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrole.rbac.authorization.k8s.io/ingress-nginx-admission created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx-admission created
configmap/ingress-nginx-controller created
service/ingress-nginx-controller created
service/ingress-nginx-controller-admission created
deployment.apps/ingress-nginx-controller created
job.batch/ingress-nginx-admission-create created
job.batch/ingress-nginx-admission-patch created
ingressclass.networking.k8s.io/nginx created
validatingwebhookconfiguration.admissionregistration.k8s.io/ingress-nginx-admission created
```

Now your nginx ingress controller is installed.

8. **Wait for sometime and check the ingress again.**

```
kubectl get ing
```

```
ubuntu@ip-172-31-39-161:~/CKA-2024/Resources/Day33/Flask/k8s$ kubectl get ing
NAME          CLASS   HOSTS         ADDRESS        PORTS   AGE
hello-world   nginx   example.com   10.107.181.7   80      3m44s
```

As you can see, the address field is populated with the IP address, let us check if the ingress is working.

9. **Let us see if the ingress is working or not.**

```
curl 10.107.181.7
```

```
ubuntu@ip-172-31-39-161:~/CKA-2024/Resources/Day33/Flask/k8s$ curl 10.107.181.7
<html>
<head><title>404 Not Found</title></head>
<body>
<center><h1>404 Not Found</h1></center>
<hr><center>nginx</center>
</body>
</html>
```

Oops, you can see the 404 not found here. This is because we can given in the ingress that it should **only be accessible on the host which is example.com.**

So we have to resolve example.com to the ingress IP address. Let us see how to do that in the next step.

10. **Go to /etc/hosts file and add the below line**

```
10.107.181.7 example.com
```

Add your ingress IP space the host name. Save it and try to run curl command again.

11. **Run curl command**

```
curl example.com
```

```
ubuntu@ip-172-31-39-161:~/CKA-2024/Resources/Day33/Flask/k8s$ curl example.com
Hello, World!ubuntu@ip-172-31-39-161:~/CKA-2024/Resources/Day33/Flask/k8s$
```

Finally we our application is accessible again using ingress.

Conclusion: