Contents

[Kubernetes Ingress 2](#_Toc14791290)

[Setup Nginx Ingress Controller 2](#_Toc14791291)

[Setup LoadBalancer Service For Ingress Controller 2](#_Toc14791292)

[Map A Domain Name To Loadbalancer IP 3](#_Toc14791293)

[Single DNS Mapping: 3](#_Toc14791294)

[Wildcard DNS Mapping: 3](#_Toc14791295)

[Setup A Demo Application 4](#_Toc14791296)

[Create Kubernetes Ingress Object 5](#_Toc14791297)

# Kubernetes Ingress

## Setup Nginx Ingress Controller

There are two nginx ingress controllers.

* Nginx ingress controller by kubernetes community
* Nginx ingress controller by Nginx Inc

We will be using the Nginx controller from the kubernetes community.

Ingress controller needs a specific namespace, service account, cluster role bindings, configmaps etc. You can create all the kubernetes objects mentioned using the [yaml file from official ingress repo](https://github.com/kubernetes/ingress-nginx/tree/master/deploy" \t "_blank)

Let’s deploy the ingress controller using mandatory.yaml file from the official repo. It has the consolidated list of [kubernetes objects](https://devopscube.com/kubernetes-deployment-tutorial/" \t "_blank) required for the Nginx controller.

Lets create the Nginx controller deployment using kubectl.

kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/master/deploy/static/mandatory.yaml

Check the ingress controller pods to make sure if it is setup correctly.

kubectl get pods -n ingress-nginx

## Setup LoadBalancer Service For Ingress Controller

Next step is to create a service of Type Loadbalancer to expose the nginx controller deployment outside the cluster.

**Step 1:** Create a project directory locally and switch to that directory.

mkdir ingress-deployment && cd ingress-deployment

**Step 2:** Create a file named nginx-ingress.yaml

vi nginx-ingress.yaml

**Step 3:** Copy the following contents to the file.

**Note**: The annotations under the labels are very important for integrating with the nginx controller deployment.

kind: Service

apiVersion: v1

metadata:

  name: ingress-nginx

  namespace: ingress-nginx

  labels:

    app.kubernetes.io/name: ingress-nginx

    app.kubernetes.io/part-of: ingress-nginx

spec:

  externalTrafficPolicy: Local

  type: LoadBalancer

  selector:

    app.kubernetes.io/name: ingress-nginx

    app.kubernetes.io/part-of: ingress-nginx

  ports:

    - name: http

      port: 80

      targetPort: http

    - name: https

      port: 443

      targetPort: https

**Step 4:** Create the ingress service.

kubectl apply -f nginx-ingress.yaml

**Step 5:** Check the created service if it is attached to the external load balancer.

kubectl get svc -n ingress-nginx

### Map A Domain Name To Loadbalancer IP

To make our ingress settings work, we need to map a domain name to the load balancer IP. You can do it in two ways.

### Single DNS Mapping:

You can map single domain directly as a A record to the load balancer IP. Using this you can have only one domain for the ingress controller and multiple path based traffic routing.

For example,

www.example.com --> Loadbalancer IP

You can have path based routing using this model.

Few examples,

http://www.example.com/app1

http://www.example.com/app2

http://www.example.com/app1/api

<http://www.example.com/app2/api>

### Wildcard DNS Mapping:

If you map a wildcard DNS to the load balancer, you can have dynamic DNS end points through ingress.

For example,

\*.apps.example.com

This way you can have multiple dynamic subdomains through single ingress controller and each DNS can have its own path based routing.

For example,

#URL one

http://demo1.apps.example.com/api

http://demo1.apps.example.com/api/v1

http://demo1.apps.example.com/api/v2

#URL two

http://demo2.apps.example.com/api

http://demo2.apps.example.com/api/v1

<http://demo2.apps.example.com/api/v2>

For demo purposes, we have mapped a wildcard DNS to the LoadBalancer IP. Based on your DNS provider, you can do this setting.

## Setup A Demo Application

For testing purposes, we will deploy a demo application and add a ClusterIp service to it.

**Step 1:**create a namespace named dev

kubectl create namespace dev

**Step 2:** Create a file named hello-app.yaml

**Step 3:** Copy the following contents and save the file.

apiVersion: apps/v1

kind: Deployment

metadata:

  name: hello-app

  namespace: dev

spec:

  selector:

    matchLabels:

      app: hello

  replicas: 3

  template:

    metadata:

      labels:

        app: hello

    spec:

      containers:

      - name: hello

        image: "gcr.io/google-samples/hello-app:2.0"

**Step 4:** Create the deployment using kubectl

kubectl create -f hello-app.yaml

Check the deployment status.

kubectl get deployments -n dev

**Step 5:** Create a file named hello-app-service.yaml

**Step 6:** Copy the following contents and save the file.

apiVersion: v1

kind: Service

metadata:

  name: hello-service

  namespace: dev

  labels:

    app: hello

spec:

  type: ClusterIP

  selector:

    app: hello

  ports:

  - port: 80

    targetPort: 8080

    protocol: TCP

**Step 7:** Create the service using kubectl.

kubectl create -f hello-app-service.yaml

Check the service status:

kubectl get svc -n dev

## Create Kubernetes Ingress Object

Now let’s create an ingress object to access our hello app using a DNS. An ingress object is nothing but a setup of routing rules.

If you are wondering how ingress object is connected to Nginx controller, the ingress controller pod connects to the Ingress API to check for rules and it updates its nginx.conf accordingly.

**Step 1:** Create a file named ingress.yaml

**Step 2:** Copy the following contents and save the file.

Replace test.apps.example.info with your domain name. Here the assumption is that you have a wildcard DNS in the format \*.apps.example.info

apiVersion: extensions/v1beta1

kind: Ingress

metadata:

  name: test-ingress

  namespace: dev

spec:

  rules:

  - host: test.apps.example.info

    http:

      paths:

      - backend:

          serviceName: hello-service

          servicePort: 80

**Step 3:** Describe created ingress object created to check the configurations.

kubectl describe ingress  -n dev

Now if you try to access test.apps.example.info domain (replace it with your domain name), you should be able to access our sample app deployed.