Kubernetes Learning Series

Part 10: Ingress (Practical Implementation)

A Step-by-Step Beginner-Friendly Hands-On Guide
with Retrospectives, Debugging, and Learnings

*By **Sarthak Srivastava***

Host-based Ingress with NGINX - Beginner-friendly guide

This short guide shows exactly how to route **blue.example.com** and **green.example.com** to **two tiny apps** using an **NGINX Ingress controller** in a **KodeKloud playground Kubernetes cluster**. Follow the steps in the "**Quick steps**" section to get **Hello from Blue!** and **Hello from Green!** working, then read the "**Common mistakes & fixes**" and the compact **troubleshooting** checklist.

What you'll get

- Copy-paste-ready commands to run in order.
- Beginner-friendly explanations (what and *why*).
- The common mistakes I made and how to avoid them.
- A short checklist for fast debugging.

Prerequisites

- kubectl configured to your cluster.
- A playground or local Kubernetes cluster (KodeKloud, minikube, kind).
- Edit access to /etc/hosts on your machine or VM that will run curl.

Quick steps (copy & run to produce the correct output)

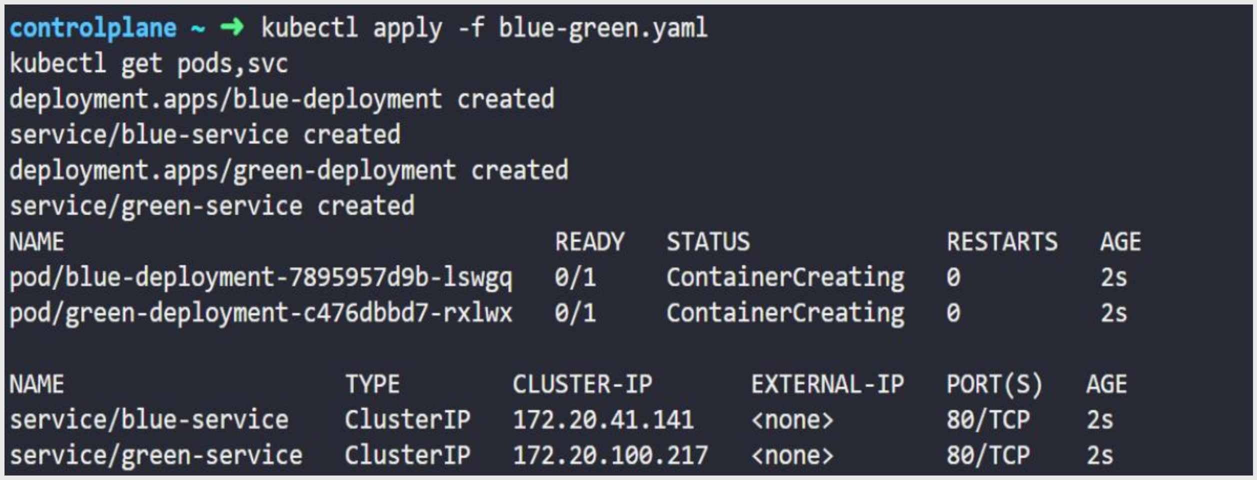
Run these commands in order on a fresh cluster. After step 6 you'll see Hello from Blue! and Hello from Green!

1) Deploy the apps (Blue & Green)

Save and run the following (creates deployments + services):

```
cat > blue-green.yaml <<'EOF'

apiVersion: apps/v1

kind: Deployment

metadata:
  name: blue-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: blue
  template:
    metadata:
      labels:
        app: blue
    spec:
      containers:
      - name: blue
        image: hashicorp/http-echo
        args:
          - "-text=Hello from Blue!"
        ports:
        - containerPort: 5678
---
apiVersion: v1
kind: Service
metadata:
  name: blue-service
spec:
  selector:
    app: blue
  ports:
  - port: 80
    targetPort: 5678
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: green-deployment
spec:
  replicas: 1
  selector:
    matchLabels:
      app: green
  template:
    metadata:
      labels:
        app: green
    spec:
      containers:
      - name: green
        image: hashicorp/http-echo
        args:
          - "-text=Hello from Green!"

        ports:

        - containerPort: 5678

---

apiVersion: v1

kind: Service

metadata:

  name: green-service

spec:

  selector:

    app: green

  ports:

  - port: 80

    targetPort: 5678

EOF
```

$ kubectl apply -f blue-green.yaml
$ kubectl get pods,svc



```
controlplane ~ ➜ kubectl apply -f blue-green.yaml
kubectl get pods,svc
deployment.apps/blue-deployment created
service/blue-service created
deployment.apps/green-deployment created
service/green-service created
NAME                                  READY   STATUS              RESTARTS   AGE
pod/blue-deployment-7895957d9b-lswgq  0/1     ContainerCreating   0          2s
pod/green-deployment-c476dbbd7-rxlwx  0/1     ContainerCreating   0          2s

NAME                    TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
service/blue-service    ClusterIP   172.20.41.141    <none>        80/TCP    2s
service/green-service   ClusterIP   172.20.100.217   <none>        80/TCP    2s
```

Quick verification (confirms the app is healthy):

# replace <blue-pod> with the actual pod name
$ kubectl port-forward pod/<blue-pod> 8081:5678 &
$ curl http://127.0.0.1:8081 # -> **Hello from Blue**!

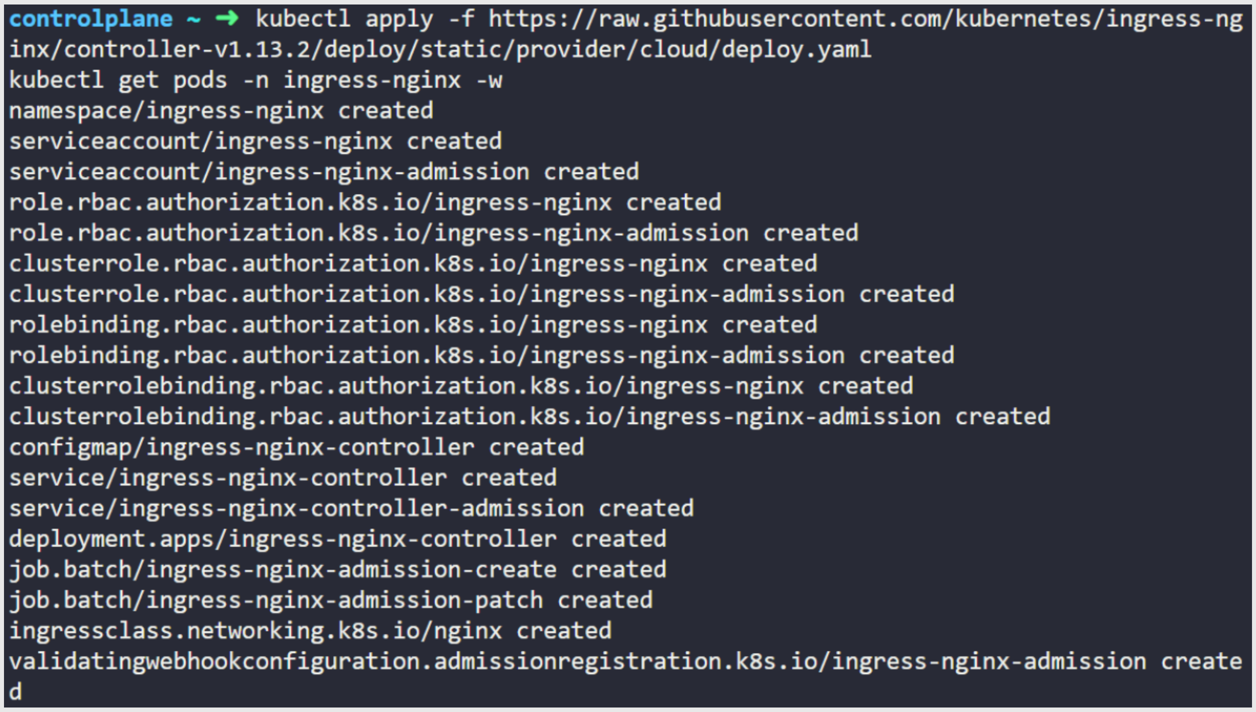
# stop the port-forward after test
$ pkill -f "kubectl port-forward pod/<blue-pod>"

If the curl shows **Hello from Blue**!, the app and service are working.

---

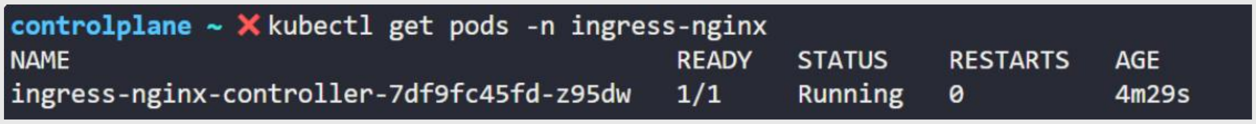
2) Install NGINX Ingress controller

$ kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.13.2/deploy/static/provider/cloud/deploy.yaml



```
controlplane ~ ➜ kubectl apply -f https://raw.githubusercontent.com/kubernetes/ingress-ng
inx/controller-v1.13.2/deploy/static/provider/cloud/deploy.yaml
kubectl get pods -n ingress-nginx -w
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 create
d
```

$ kubectl get pods -n ingress-nginx



```
controlplane ~ ✗ kubectl get pods -n ingress-nginx
NAME                                       READY   STATUS    RESTARTS   AGE
ingress-nginx-controller-7df9fc45fd-z95dw  1/1     Running   0          4m29s
```

# wait until controller pod is **1/1 Running**

Check the controller service:

$ kubectl get svc -n ingress-nginx

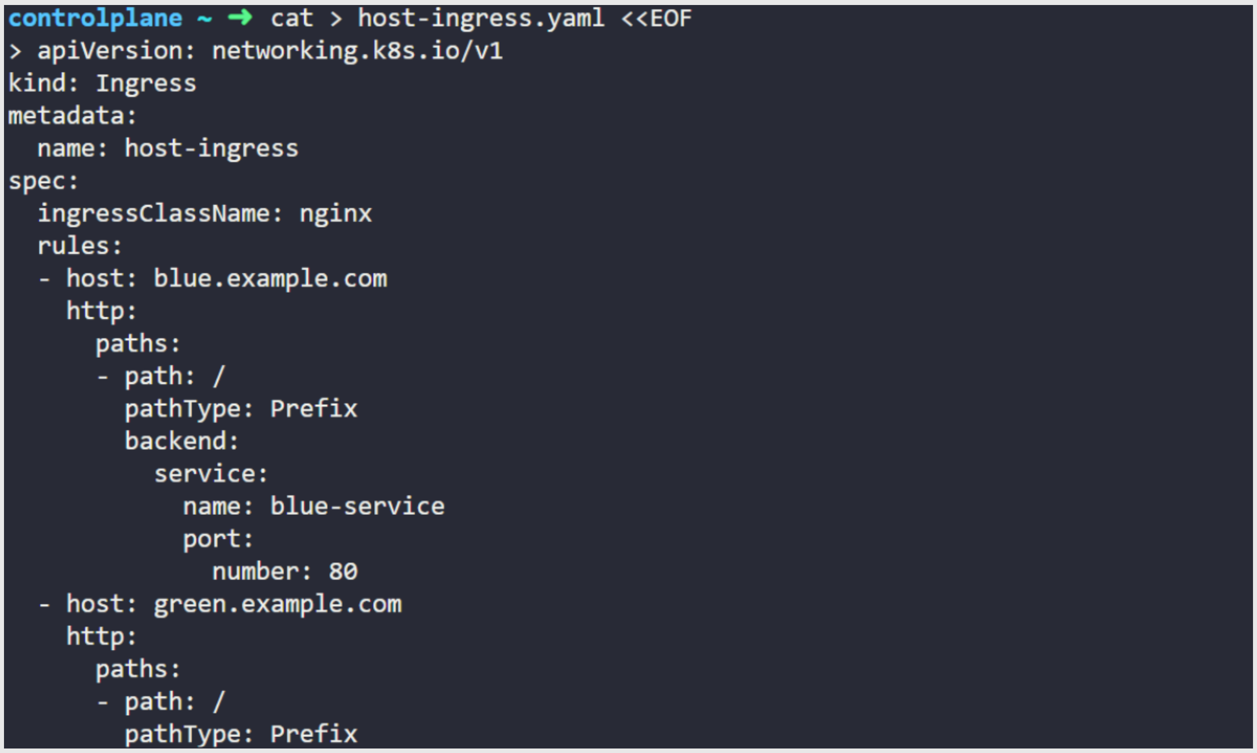
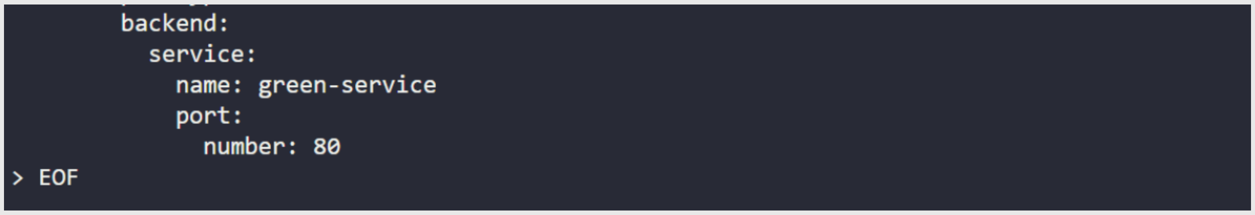
Note: In playground/non-cloud environments **LoadBalancer** usually shows EXTERNAL-IP <pending>, that's expected. Keep going.

---

3) Create the host-based Ingress

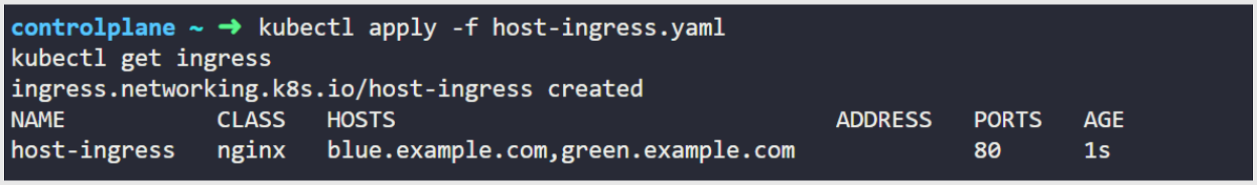
Save and apply:

```
cat > host-ingress.yaml <<'EOF'
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: host-ingress
spec:
  ingressClassName: nginx
  rules:
  - host: blue.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: blue-service
            port:
              number: 80
  - host: green.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: green-service
            port:
              number: 80

EOF
```

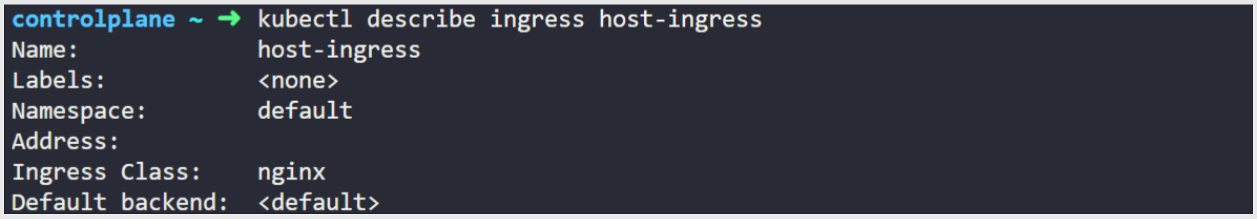
```
controlplane ~ ➜  cat > host-ingress.yaml <<EOF
> apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: host-ingress
spec:
  ingressClassName: nginx
  rules:
  - host: blue.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: blue-service
            port:
              number: 80
  - host: green.example.com
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
            name: green-service
            port:
              number: 80
> EOF
```

$ kubectl apply -f host-ingress.yaml
$ kubectl get ingress

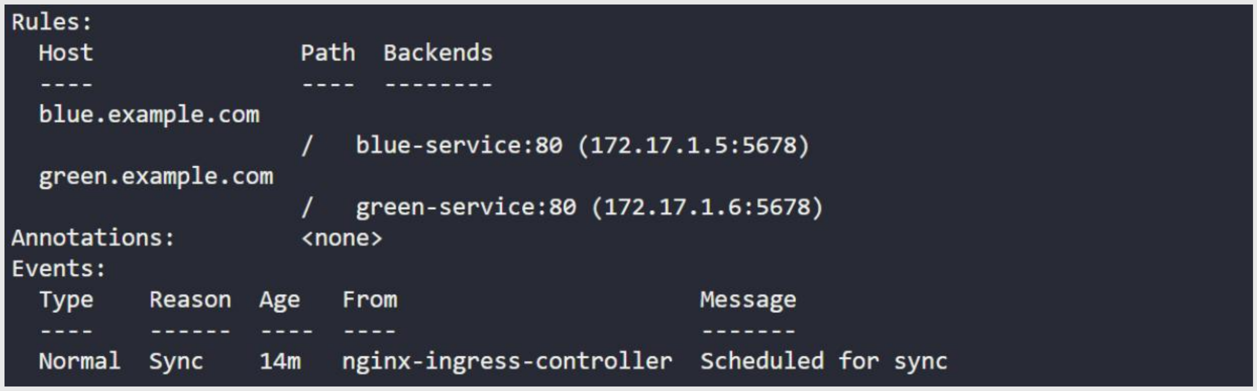


```
controlplane ~ ➜  kubectl apply -f host-ingress.yaml
kubectl get ingress
ingress.networking.k8s.io/host-ingress created
NAME           CLASS   HOSTS                                ADDRESS   PORTS   AGE
host-ingress   nginx   blue.example.com,green.example.com             80      1s
```

$ kubectl describe ingress host-ingress



```
controlplane ~ ➜  kubectl describe ingress host-ingress
Name:             host-ingress
Labels:           <none>
Namespace:        default
Address:
Ingress Class:    nginx
Default backend:  <default>
```


```
Rules:
  Host              Path  Backends
  ----              ----  --------
  blue.example.com
                    /   blue-service:80 (172.17.1.5:5678)
  green.example.com
                    /   green-service:80 (172.17.1.6:5678)
Annotations:        <none>
Events:
  Type    Reason  Age   From                      Message
  ----    ------  ----  ----                      -------
  Normal  Sync    14m   nginx-ingress-controller  Scheduled for sync
```

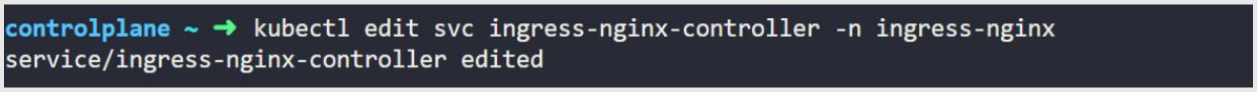
describe should show **blue-service**:80 (IP:5678) etc.

---

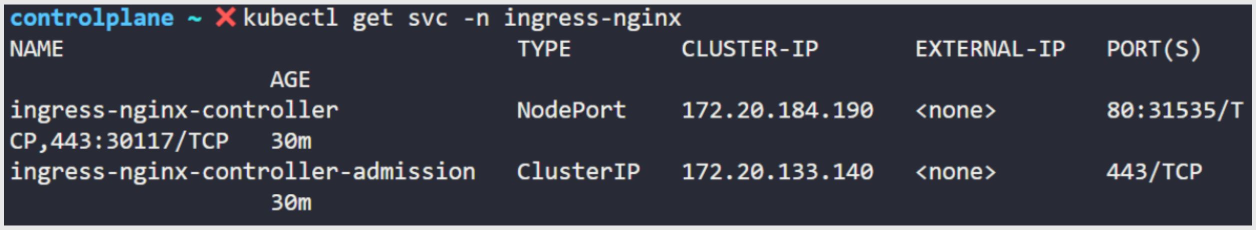
4) If EXTERNAL-IP is <pending>, convert to NodePort

In many labs the controller service will be type **LoadBalancer** but **no external IP** is provided. Convert it to **NodePort**:

$ kubectl edit svc ingress-nginx-controller -n ingress-nginx
# change: type: LoadBalancer -> type: NodePort


```
controlplane ~ ➜ kubectl edit svc ingress-nginx-controller -n ingress-nginx
service/ingress-nginx-controller edited
```

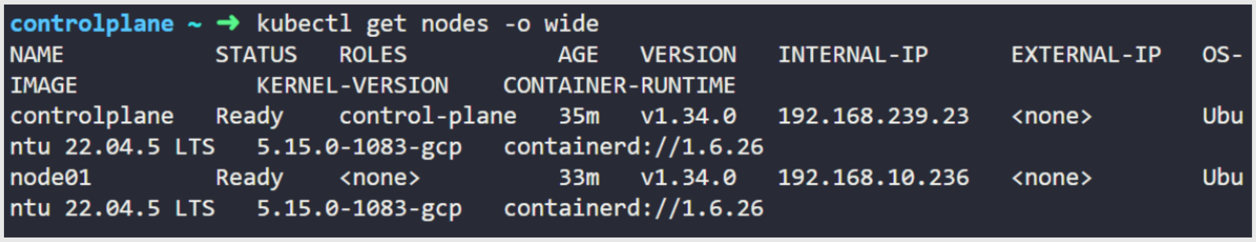
$ kubectl get svc -n ingress-nginx
# note **nodePort** for port **80**, e.g. **80:31535/TCP** -> **nodePort = 31535**


```
controlplane ~ ✗ kubectl get svc -n ingress-nginx
NAME                                 TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)
                  AGE
ingress-nginx-controller             NodePort    172.20.184.190   <none>        80:31535/T
CP,443:30117/TCP   30m
ingress-nginx-controller-admission   ClusterIP   172.20.133.140   <none>        443/TCP
                   30m
```

---

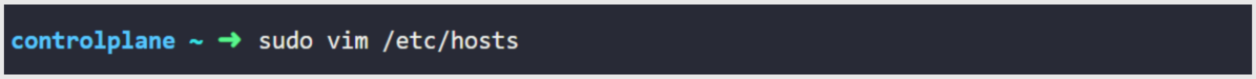
5) Find node IP and map hostnames

$ kubectl get nodes -o wide
# copy INTERNAL-IP (e.g. 192.168.239.23)


```
controlplane ~ ➜ kubectl get nodes -o wide
NAME           STATUS   ROLES           AGE   VERSION   INTERNAL-IP      EXTERNAL-IP   OS-
IMAGE             KERNEL-VERSION    CONTAINER-RUNTIME
controlplane   Ready    control-plane   35m   v1.34.0   192.168.239.23   <none>        Ubu
ntu 22.04.5 LTS   5.15.0-1083-gcp   containerd://1.6.26
node01         Ready    <none>          33m   v1.34.0   192.168.10.236   <none>        Ubu
ntu 22.04.5 LTS   5.15.0-1083-gcp   containerd://1.6.26
```

Edit **/etc/hosts** on the machine where you'll run **curl** and add:

$ sudo vim /etc/hosts



```
controlplane ~ ➜ sudo vim /etc/hosts
```

192.168.239.23 blue.example.com
192.168.239.23 green.example.com

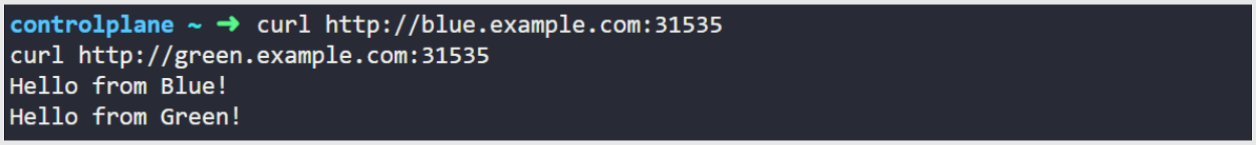
(Replace 192.168.239.23 with your node IP.)

---

6) Final test (this must return the Hello messages)

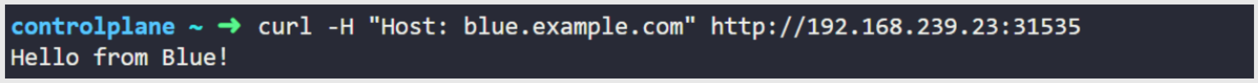
# use the **nodePort** noted earlier (example 31535)
$ curl http://blue.example.com:31535
$ curl http://green.example.com:31535

# -> Hello from Blue!
# -> Hello from Green!



```
controlplane ~ ➜ curl http://blue.example.com:31535
curl http://green.example.com:31535
Hello from Blue!
Hello from Green!
```

# alternative using Host header to node IP:
# curl -H "Host: blue.example.com" http://192.168.239.23:31535



```
controlplane ~ ➜ curl -H "Host: blue.example.com" http://192.168.239.23:31535
Hello from Blue!
```

If these print the **Hello messages**, **congratulations** , host-based Ingress is working.

II. Common mistakes — what they look like, why they happen, and the exact fix

For each item: **What you see → Why it's happening → How to fix it (commands / steps) → Tip to avoid it**

---

1) LoadBalancer shows EXTERNAL-IP: <pending>

What you see

kubectl get svc -n ingress-nginx

```
NAME                      TYPE          CLUSTER-IP    EXTERNAL-IP   PORT(S)

ingress-nginx-controller    LoadBalancer   172.20.x.x     <pending>     80:xxxxx/TCP
```

Why it happens

- Cloud providers assign an external IP to **LoadBalancer** services.
- Playground / local / on-prem clusters (minikube, kind, KodeKloud) usually don't have a cloud load balancer available, so the IP stays <**pending**>.

Fix (exact steps)

1. Change the **controller service** to **NodePort**:

$ kubectl edit svc ingress-nginx-controller -n ingress-nginx

# in the editor: change `type: **LoadBalancer**` -> `type: **NodePort**`, save

2. **Verify**:

$ kubectl get svc -n ingress-nginx

# note nodePort shown as **80:31535/TCP** — **nodePort** is **31535**

Tip to avoid

- If you're on a local/playground cluster, plan to use **NodePort** (or minikube/kind-specific add-ons) instead of expecting **LoadBalancer**.

---

2) Hostnames mapped to 127.0.0.1 instead of the node IP

What you see

- curl http://blue.example.com:31535 returns nothing or hits the wrong service.
- Your **/etc/hosts** has:

127.0.0.1 blue.example.com

Why it happens

- **127.0.0.1** is **localhost** points to your **current machine's loopback interface**.
- The ingress **NodePort** listens on the **node's network IP** (e.g., 192.168.239.23), not necessarily 127.0.0.1.

Fix (exact steps)

1. Find **node IP**:

$ kubectl get nodes -o wide

# Use INTERNAL-IP, e.g. 192.168.239.23

2. Edit **/etc/hosts** and add:

192.168.239.23 blue.example.com

192.168.239.23 green.example.com

3. **Test**:

```
$ curl http://blue.example.com:31535
```

# should return **Hello from Blue**!

Tip to avoid

- Always map your **test hostnames** to the **node IP** that exposes the **NodePort**

---

3) Not sending the Host: header (or testing incorrectly with port-forward)

What you see

- **curl http://127.0.0.1:8080/** returns a default page.
- **Ingress routing** does not send your request to the **right backend**.

Why it happens

- **Ingress matches** rules by the **HTTP Host: header**. If the header doesn't match **blue.example.com**, the controller won't route to **blue-service**.
- **Port-forwarding to a service** vs. pod can sometimes **return the controller's own admin/default page or hit HTTPS**, causing **confusing** output.

Fix (exact steps)

1. Either set **host mapping** or use the domain:

```
# with /etc/hosts mapping in place:
$ curl http://blue.example.com:31535
```

Or include the **Host header** explicitly in the curl request:

```
$ curl -H "Host: blue.example.com" http://127.0.0.1:8080
```

2. For **port-forward testing**, prefer forwarding to the **pod**:

```
$ kubectl port-forward -n ingress-nginx pod/<controller-pod> 8080:80
$ curl -H "Host: blue.example.com" http://127.0.0.1:8080
```

Tip to avoid

- Always think "what Host header will NGINX see?" If you're not sure, pass it explicitly with -H.

---

4) Service selector / port mismatch → "no active Endpoint"

What you see
Controller log or kubectl describe shows:

```
Service "default/blue-service" does not have any active Endpoint
```

Or kubectl get endpoints shows empty for that service.

Why it happens

- Service **selector labels** don't match **Deployment labels**, or
- Service **targetPort** doesn't match container **containerPort**.

Fix (exact steps)

1. **Inspect service** and **deployment**:

$ kubectl get svc blue-service -o yaml

$ kubectl get deploy blue-deployment -o yaml

2. Ensure **spec.selector** in the **Service equals** the **pod labels** (in Deployment → template → metadata → labels).
3. Ensure service has **ports**: - **port:** 80, **targetPort:** 5678 if the container listens on **5678**.
4. Verify **endpoints**:

$ kubectl get endpoints blue-service

# should show podIP:**5678**

Tip to avoid

- Always name labels deliberately (e.g., **app: blue**) and keep service selectors consistent.

Learnings - simple principles to remember

1. **Ingress is a traffic director, not a server.**
   It **routes external HTTP(S) into internal services** based on **Host** and **Path** rules.
2. **Host-based routing depends on the HTTP Host: header.**
   If that header doesn't match an **Ingress rule**, routing won't happen.
3. **Service types depend on environment.**
   LoadBalancer works on **cloud providers** (they give an IP). Local labs usually need **NodePort**.
4. **NodePort = nodeIP + port.**
   You must reach your **node's IP** at the **nodePort** to hit the **controller** from **outside** the cluster.
5. **Start small: pod → service → ingress.**
   Always confirm the pod works first, then the service, then ingress. This keeps debugging simple.

Troubleshooting - short, step-by-step flow you can follow

Use this exact ordered flow. After each step, stop and confirm the expected output before moving on.

1. Pod & service health

```
$ kubectl get pods

$ kubectl get svc blue-service green-service

$ kubectl get endpoints blue-service
```

Expected: pods **Running**, services **present**, endpoints list podIP:**5678**.

2. **Direct pod test**

```
$ kubectl port-forward pod/<blue-pod> 8081:5678 &   # background

curl http://127.0.0.1:8081

# -> Hello from Blue!
```

If this fails, fix the Deployment/Container first.

3. **Ingress resource check**

```
$ kubectl describe ingress host-ingress
```

Expected: rules show **blue.example.com** → **blue-service:80** (podIP:**5678**).

4. **Controller alive?**

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

$ kubectl logs -n ingress-nginx <controller-pod> | tail -n 50
```

Look for:

- Found valid **IngressClass**
- successfully validated configuration, accepting
- Backend **successfully reloaded**

If the controller logs mention no active Endpoint, go back to step 1.

5. **Is controller reachable externally?**

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

If **EXTERNAL-IP is <pending>** and you're not on cloud: change **LoadBalancer** → **NodePort** (see earlier fix).

6. **Node IP & /etc/hosts**

```
$ kubectl get nodes -o wide   # pick INTERNAL-IP

# add to /etc/hosts:

# 192.168.x.x blue.example.com
```

7. **Final test**

```
curl http://blue.example.com:<nodePort>   # Should be Hello from Blue!
```

curl http://green.example.com:<nodePort> # Should be Hello from Green!

Retrospective

We set up two tiny web apps and an NGINX Ingress controller to route traffic by hostname. The apps were fine from the start; we verified that by port-forwarding to the pod. The real problems were **environment-related**: the Ingress controller was configured as a **LoadBalancer** (which needs a **cloud provider** to get a **public IP**), our test **hostnames** were pointing at **localhost** instead of the **node IP**, and we sometimes tested without the **Host: heade**r. Changing the controller to **NodePort**, mapping hostnames to the node's IP in **/etc/hosts**, and testing with the **correct Host header** got everything working.

The **key** lesson: most Ingress "**failures**" are actually **network** or testing mistakes, not incorrect YAML.