

## KUBERNETES NETWORKING

<b>Kubernetes networking fundamentals.....</b>	<b>2</b>
Investigating k8s networking.....	2
Get all Nodes and their IP information, INTERNAL-IP is the real IP of the Node.....	2
Deploy a basic workload, hello-world with 3 replicas. ....	2
Verify if a pod has its unique IP address .....	3
Access to pod shell and check its networking configuration .....	3
SSH to the worker node and check the network information.....	4
Exploring cluster DNS.....	2
Get k8s service in kube-system .....	5
Get detail info about CoreDNS deployment.....	5
Discover the CoreDNS configuration and default forwarder .....	7
Configure Pod DNS client Configuration .....	8
Check the DNS configuration of the normal pod and custom pod.....	8
DNS discovering.....	9
<b>Configuring and managing application access with services .....</b>	<b>9</b>
<b>ClusterIP.....</b>	<b>9</b>
<b>NodePort.....</b>	<b>10</b>
<b>LoadBalancer .....</b>	<b>11</b>
<b>Service Discovery .....</b>	<b>12</b>
<b>Configuring and managing application access with ingress.....</b>	<b>13</b>

# I. Kubernetes networking fundamentals

## 1. Investigating k8s networking

### a. Get all Nodes and their IP information, INTERNAL-IP is the real IP of the Node

```
kubectl get nodes -o wide
```

NAME	STATUS	ROLES	AGE	VERSION	INTERNAL-IP	EXTERNAL-IP	OS-IMAGE	KERNEL-VERSION	CONTAINER-RUNTIME
gke-cluster-1-default-pool-487a6374-lj8l	Ready	<none>	2d23h	v1.16.11-gke.5	10.148.0.10		Container-Optimized OS from Google	4.19.112+	docker://19.3.1
gke-cluster-1-default-pool-487a6374-nz29	Ready	<none>	2d23h	v1.16.11-gke.5	10.148.0.9		Container-Optimized OS from Google	4.19.112+	docker://19.3.1
gke-cluster-1-default-pool-487a6374-x2nh	Ready	<none>	2d23h	v1.16.11-gke.5	10.148.0.8		Container-Optimized OS from Google	4.19.112+	docker://19.3.1

### b. Deploy a basic workload, hello-world with 3 replicas.

```
https://github.com/hungtran84/k8s-cka/blob/master/d3_networking/01_networking_fundamentals/Deployment.yaml
kubectl apply -f Deployment.yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: hello-world
spec:
  replicas: 10
  selector:
    matchLabels:
      app: hello-world
  template:
    metadata:
      labels:
        app: hello-world
    spec:
      containers:
        - name: hello-world
          image: gcr.io/google-samples/hello-app:1.0
          ports:
            - containerPort: 8080
          resources:
```

```

requests:
  memory: 128M
  cpu: 100m
limits:
  memory: 128M
  cpu: 100m
---
apiVersion: v1
kind: Service
metadata:
  name: hello-world
spec:
  selector:
    app: hello-world
  ports:
    - port: 80
      protocol: TCP
      targetPort: 8080

deployment.apps/hello-world created
service/hello-world created

```

### c. Verify if a pod has its unique IP address

```

kubectl get pods -o wide

```

NAME	READY	STATUS	RESTARTS	AGE	IP	NODE
hello-world-5b76c5697b-7cd48	1/1	Running	0	5m39s	10.48.0.5	gke-cluster-1-default-pool-487a6374-x2nh
hello-world-5b76c5697b-8mjn4	1/1	Running	0	5m39s	10.48.1.5	gke-cluster-1-default-pool-487a6374-nz29
hello-world-5b76c5697b-cvpcv	1/1	Running	0	5m39s	10.48.2.11	gke-cluster-1-default-pool-487a6374-lj8l

### d. Access to pod shell and check its networking configuration

```

PODNAME=$(kubectl get pods --selector=app=hello-world -o jsonpath='{ .items[0].metadata.name }')
kubectl exec -it $PODNAME -- ip addr

```

```

1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever

```

```
3: eth0@if8: <BROADCAST,MULTICAST,UP,LOWER_UP,M-DOWN> mtu 1460 qdisc noqueue
state UP
    link/ether 26:e3:bb:fe:f1:cc brd ff:ff:ff:ff:ff:ff
    inet 10.48.0.5/24 brd 10.48.0.255 scope global eth0
        valid_lft forever preferred_lft forever
```

#### e. SSH to the worker node and check the network information

```
ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc mq state UP group
default qlen 1000
    link/ether 42:01:0a:94:00:0a brd ff:ff:ff:ff:ff:ff
    inet 10.148.0.10/32 scope global dynamic eth0
        valid_lft 3500sec preferred_lft 3500sec
    inet6 fe80::4001:aff:fe94:a/64 scope link
        valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN group default
    link/ether 02:42:f8:56:08:2c brd ff:ff:ff:ff:ff:ff
    inet 169.254.123.1/24 brd 169.254.123.255 scope global docker0
        valid_lft forever preferred_lft forever
4: cbr0: <BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP> mtu 1460 qdisc htb state
UP group default qlen 1000
    link/ether 02:61:37:4a:3c:09 brd ff:ff:ff:ff:ff:ff
    inet 10.48.2.1/24 brd 10.48.2.255 scope global cbr0
        valid_lft forever preferred_lft forever
    inet6 fe80::61:37ff:fe4a:3c09/64 scope link
        valid_lft forever preferred_lft forever
5: veth016d847c@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc noqueue
master cbr0 state UP group default
    link/ether c2:3c:99:13:c9:d5 brd ff:ff:ff:ff:ff:ff link-netnsid 0
    inet6 fe80::c03c:99ff:fe13:c9d5/64 scope link
        valid_lft forever preferred_lft forever
6: vethd910d1ef@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc noqueue
master cbr0 state UP group default
    link/ether 16:81:fc:23:6d:33 brd ff:ff:ff:ff:ff:ff link-netnsid 1
    inet6 fe80::1481:fcff:fe23:6d33/64 scope link
        valid_lft forever preferred_lft forever
```

```
8: vethdc6beca3@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc noqueue
master cbr0 state UP group default
    link/ether 7e:77:11:cc:9f:9f brd ff:ff:ff:ff:ff:ff link-netnsid 3
    inet6 fe80::7c77:11ff:fecc:9f9f/64 scope link
        valid_lft forever preferred_lft forever
14: veth79a85604@if3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1460 qdisc
noqueue master cbr0 state UP group default
    link/ether 8a:98:f0:e7:67:29 brd ff:ff:ff:ff:ff:ff link-netnsid 2
    inet6 fe80::8898:f0ff:fee7:6729/64 scope link
        valid_lft forever preferred_lft forever

route
Kernel IP routing table
Destination    Gateway      Genmask      Flags Metric Ref    Use Iface
default        10.148.0.1   0.0.0.0      UG    1024  0      0 eth0
10.48.2.0      0.0.0.0      255.255.255.0 U    0      0      0 cbr0
10.148.0.1     0.0.0.0      255.255.255.255 UH   1024  0      0 eth0
169.254.123.0 0.0.0.0      255.255.255.0 U    0      0      0 docker0

brctl show
bridge name    bridge id          STP enabled    interfaces
cbr0           8000.0261374a3c09  no             veth016d847c
               veth79a85604
               vethd910d1ef
               vethdc6beca3
docker0        8000.0242f856082c  no
```

## 2. Exploring cluster DNS

### a. Get k8s service in kube-system

```
kubectl get service --namespace kube-system
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
dashboard-metrics-scraper	ClusterIP	10.0.20.20	<none>	8000/TCP	35m
kube-dns	ClusterIP	10.0.0.10	<none>	53/UDP,53/TCP	35m
kubernetes-dashboard	ClusterIP	10.0.50.224	<none>	443/TCP	35m
metrics-server	ClusterIP	10.0.61.133	<none>	443/TCP	35m

### b. Get detail info about CoreDNS deployment

```
kubectl describe deployment coredns --namespace kube-system
```

```
Name:                coredns
Namespace:            kube-system
CreationTimestamp:    Sat, 18 Jul 2020 16:37:43 +0700
Labels:               addonmanager.kubernetes.io/mode=Reconcile
```

```

    k8s-app=kube-dns
    kubernetes.io/cluster-service=true
    kubernetes.io/name=CoreDNS
    version=v20
Annotations:      deployment.kubernetes.io/revision: 1
Selector:          k8s-app=kube-dns,version=v20
Replicas:          2 desired | 2 updated | 2 total | 2 available | 0 unavailable
StrategyType:      RollingUpdate
MinReadySeconds:   0
RollingUpdateStrategy: 1 max unavailable, 25% max surge
Pod Template:
  Labels:           k8s-app=kube-dns
                   kubernetes.io/cluster-service=true
                   version=v20
  Annotations:      prometheus.io/port: 9153
  Service Account:  coredns
  Containers:
    coredns:
      Image:         mcr.microsoft.com/oss/kubernetes/coredns:1.6.6
      Ports:         53/UDP, 53/TCP, 9153/TCP
      Host Ports:    0/UDP, 0/TCP, 0/TCP
      Args:
        -conf
        /etc/coredns/Corefile
      Limits:
        memory: 170Mi
      Requests:
        cpu:      100m
        memory:   70Mi
      Liveness:    http-get http://:8080/health delay=60s timeout=5s period=10s #success=1
                  #failure=5
      Environment: <none>
      Mounts:
        /etc/coredns from config-volume (ro)
        /etc/coredns/custom from custom-config-volume (ro)
        /tmp from tmp (rw)
  Volumes:
    config-volume:
      Type:      ConfigMap (a volume populated by a ConfigMap)
      Name:      coredns
      Optional:  false
    custom-config-volume:
      Type:      ConfigMap (a volume populated by a ConfigMap)
      Name:      coredns-custom
      Optional:  true

```

```

tmp:
  Type:          EmptyDir (a temporary directory that shares a pod's lifetime)
  Medium:
  SizeLimit:     <unset>
  Priority Class Name: system-node-critical
Conditions:
  Type           Status Reason
  ----           -
  Available      True   MinimumReplicasAvailable
  Progressing    True   NewReplicaSetAvailable
OldReplicaSets: <none>
NewReplicaSet:  coredns-544d979687 (2/2 replicas created)
Events:
  Type    Reason          Age   From          Message
  ----    -
  Normal  ScalingReplicaSet 38m   deployment-controller Scaled up replica set coredns-544d979687 to 1
  Normal  ScalingReplicaSet 35m   deployment-controller Scaled up replica set coredns-544d979687 to 2

```

### c. Discover the CoreDNS configuration and default forwarder

```
kubectl get configmaps --namespace kube-system coredns -o yaml
```

```

apiVersion: v1
data:
  Corefile: |
    .:53 {
      errors
      health
      kubernetes cluster.local in-addr.arpa ip6.arpa {
        pods insecure
        upstream
        fallthrough in-addr.arpa ip6.arpa
      }
      prometheus :9153
      forward . /etc/resolv.conf
      cache 30
      loop
      reload
      loadbalance
      import custom/*.override
    }
    import custom/*.server
kind: ConfigMap

```

```

metadata:
  annotations:
    kubectl.kubernetes.io/last-applied-configuration: |
      {"apiVersion":"v1","data":{"Corefile":".:53 {\n  errors\n  health\n  kubernetes cluster.local
in-addr.arpa ip6.arpa {\n  pods insecure\n  upstream\n  fallthrough in-addr.arpa ip6.arpa\n
}\n  prometheus :9153\n  forward . /etc/resolv.conf\n  cache 30\n  loop\n  reload\n
loadbalance\n  import custom/*.override\n}\nimport
custom/*.server\n"},"kind":"ConfigMap","metadata":{"annotations":{"labels":{"addonmanager.
kubernetes.io/mode":"Reconcile","k8s-app":"kube-dns","kubernetes.io/cluster-
service":"true"},"name":"coredns","namespace":"kube-system"}}
  creationTimestamp: "2020-07-18T09:37:43Z"
  labels:
    addonmanager.kubernetes.io/mode: Reconcile
    k8s-app: kube-dns
    kubernetes.io/cluster-service: "true"
  name: coredns
  namespace: kube-system
  resourceVersion: "110"
  selfLink: /api/v1/namespaces/kube-system/configmaps/coredns
  uid: fca1d50a-198a-4570-be14-2786a8b202ba
    
```

#### d. Configure Pod DNS client Configuration

```

https://github.com/hungtran84/k8s-cka/blob/master/d3\_networking/01\_networking\_fundamentals/DeploymentCustomDns.yaml#L10
kubectl apply -f DeploymentCustomDns.yaml
deployment.apps/hello-world-customdns created
service/hello-world-customdns created
    
```

#### e. Check the DNS configuration of the normal pod and custom pod

```

CUSTOM_PODNAME=$(kubectl get pods --selector=app=hello-world-customdns -o jsonpath='{
.items[0].metadata.name }')

kubectl exec -it $CUSTOM_PODNAME -- cat /etc/resolv.conf
nameserver 9.9.9.9

PODNAME=$(kubectl get pods --selector=app=hello-world -o jsonpath='{
.items[0].metadata.name }')

kubectl exec -it $PODNAME -- cat /etc/resolv.conf
nameserver 10.0.0.10
search default.svc.cluster.local svc.cluster.local cluster.local
options ndots:
    
```



## f. DNS discovering

```
# Run a busybox pod in the same namespace and test DNS resolving
```

```
kubectl run -it --rm bb --image busybox -- bin/sh
```

```
/ # nslookup hello-world
```

```
Server:      10.0.0.10
```

```
Address:     10.0.0.10:53
```

```
Name:  hello-world.default.svc.cluster.local
```

```
Address: 10.0.222.248
```

```
# Run another busybox pod in a different namespace
```

```
kubectl create ns myns
```

```
namespace/myns created
```

```
kubectl run -n myns -it --rm bb1 --image busybox -- bin/sh
```

```
/ # nslookup hello-world
```

```
Server:      10.0.0.10
```

```
Address:     10.0.0.10:53
```

```
** server can't find hello-world.myns.svc.cluster.local: NXDOMAIN
```

```
/ # nslookup hello-world.default.svc.cluster.local
```

```
Server:      10.0.0.10
```

```
Address:     10.0.0.10:53
```

## II. Configuring and managing application access with services

### 1. ClusterIP

```
#Imperative, create a deployment with one replica
```

```
kubectl create deployment hello-world-clusterip \
```

```
--image=gcr.io/google-samples/hello-app:1.0
```

```
deployment.apps/hello-world-clusterip created
```

```
#If you don't define a type, the default is ClusterIP
```

```
kubectl expose deployment hello-world-clusterip \
```

```
--port=80 --target-port=8080 --type ClusterIP
```

```
service/hello-world-clusterip exposed
```

```
#Get a list of services, examine the Type, CLUSTER-IP and Port
```

```
kubectl get svc
```

```
#Get the Service's ClusterIP and store that for reuse.
```

```
SERVICEIP=$(kubectl get service hello-world-clusterip -o jsonpath='{ .spec.clusterIP }')
```

```
echo $SERVICEIP
```

10.0.1.249

#Access the service inside the cluster

```
kubectl run bb -it --rm \
  --image radial/busyboxplus:curl \
  --restart Never \
  -- curl http://\$SERVICEIP
```

#Get a list of the endpoints for a service.

```
kubectl get endpoints hello-world-clusterip
```

NAME	ENDPOINTS	AGE
hello-world-clusterip	10.24.2.8:8080	23m

#Scale the deployment, new endpoints are registered automatically

```
kubectl scale deployment hello-world-clusterip --replicas=6
```

deployment.apps/hello-world-clusterip scaled

```
kubectl get endpoints hello-world-clusterip
```

NAME	ENDPOINTS	AGE
hello-world-clusterip	10.24.0.5:8080,10.24.0.6:8080,10.24.1.10:8080 + 3 more...	29m

## 2. NodePort

#Imperative, create a deployment with one replica

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

deployment.apps/hello-world-nodeport created

#If you don't define a type, the default is ClusterIP

```
kubectl expose deployment hello-world-nodeport \
  --port=80 --target-port=8080 --type NodePort
```

service/hello-world-nodeport exposed

#Get a list of services, examine the Type, CLUSTER-IP and NodePort...

```
kubectl get svc hello-world-nodeport
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
hello-world-nodeport	NodePort	10.0.2.236	<none>	80:30768/TCP	51s

#Get the Service's ClusterIP, NodePort and Port and store that for reuse.

```
CLUSTERIP=$(kubectl get service hello-world-nodeport -o jsonpath='{ .spec.clusterIP }')
PORT=$(kubectl get service hello-world-nodeport -o jsonpath='{ .spec.ports[].port }')
NODEPORT=$(kubectl get service hello-world-nodeport -o jsonpath='{ .spec.ports[].nodePort }')
```

#Access the service inside the cluster

```
kubectrl run bb -it --rm \
  --image radial/busyboxplus:curl \
  --restart Never \
  -- curl http://\$CLUSTERIP:\$PORT
```

#And we can access the service by hitting the node port on ANY node in the cluster on the Node's Real IP or Name.

```
curl http://NODE_IP:$NODEPORT
```

### 3. LoadBalancer

#Imperative, create a deployment with one replica

```
kubectrl create deployment hello-world-loadbalancer \
  --image=gcr.io/google-samples/hello-app:1.0
deployment.apps/hello-world-loadbalancer created
```

#If you don't define a type, the default is ClusterIP

```
kubectrl expose deployment hello-world-loadbalancer \
  --port=80 --target-port=8080 --type LoadBalancer
service/hello-world-loadbalancer exposed
```

#Get a list of services, examine the Type, CLUSTER-IP, NodePort, External IP

```
kubectrl get svc hello-world-loadbalancer
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
hello-world-loadbalancer	LoadBalancer	10.0.12.59	35.240.171.179	80:30482/TCP	8m1s

#Access the application from Internet

```
LOADBALANCERIP=$(kubectrl get service hello-world-loadbalancer -o jsonpath='{
.status.loadBalancer.ingress[.ip ]}')
curl http://$LOADBALANCERIP:$PORT
Hello, world!
Version: 1.0.0
Hostname: hello-world-loadbalancer-56c49fc47f-jg79g
```

#Have a look at the cloud load balancer

Network services

[Load balancer details](#)
[EDIT](#)
[DELETE](#)

Load balancing

Cloud DNS

Cloud CDN

Cloud NAT

Traffic Director

Service Directory

aaf6018cec29a4ece9d5ae3399b58fc5

Frontend

Protocol ^	IP:Port	Network Tier ?
TCP	35.240.171.179:80	Premium

Backend

Name: aaf6018cec29a4ece9d5ae3399b58fc5 Region: asia-southeast1 Session affinity: None Health check: k8s-470d598ae74a2d6b-node

Instances ^	Zone	35.240.171.179
gke-cluster-2-calico-default-pool-2bb75e10-0ztm	asia-southeast1-a	✓
gke-cluster-2-calico-default-pool-2bb75e10-rk6f	asia-southeast1-a	✓
gke-cluster-2-calico-default-pool-2bb75e10-sf0r	asia-southeast1-a	✓

#### 4. Service Discovery

#Create a deployment with its clusterIP service

[https://github.com/hungtran84/k8s-cka/blob/master/d3\\_networking/02\\_services/service-hello-world-clusterip.yaml](https://github.com/hungtran84/k8s-cka/blob/master/d3_networking/02_services/service-hello-world-clusterip.yaml)

kubectl apply -f service-hello-world-clusterip.yaml

deployment.apps/hello-world-clusterip created

service/hello-world-clusterip created

#Get the environment variables for the pod

PODNAME=\$(kubectl get pods -o jsonpath='{ .items[].metadata.name }')

kubectl exec -it \$PODNAME -- env | sort

HELLO\_WORLD\_CLUSTERIP\_PORT=tcp://10.0.14.193:80

HELLO\_WORLD\_CLUSTERIP\_PORT\_80\_TCP=tcp://10.0.14.193:80

HELLO\_WORLD\_CLUSTERIP\_PORT\_80\_TCP\_ADDR=10.0.14.193

HELLO\_WORLD\_CLUSTERIP\_PORT\_80\_TCP\_PORT=80

HELLO\_WORLD\_CLUSTERIP\_PORT\_80\_TCP\_PROTO=tcp

HELLO\_WORLD\_CLUSTERIP\_SERVICE\_HOST=10.0.14.193

HELLO\_WORLD\_CLUSTERIP\_SERVICE\_PORT=80

```
HOME=/root
HOSTNAME=hello-world-clusterip-5c77dccc4-6bqmp
KUBERNETES_PORT=tcp://10.0.0.1:443
KUBERNETES_PORT_443_TCP=tcp://10.0.0.1:443
KUBERNETES_PORT_443_TCP_ADDR=10.0.0.1
KUBERNETES_PORT_443_TCP_PORT=443

#Create an externalName
https://github.com/hungtran84/k8s-cka/blob/master/d3\_networking/02\_services/service-externalname.yaml
kubectl apply -f service-externalname.yaml
service/hello-world-api created

kubectl get svc hello-world-api
NAME          TYPE          CLUSTER-IP  EXTERNAL-IP          PORT(S)  AGE
hello-world-api  ExternalName  <none>       hello-world.api.example.com  <none>   24s

#Verify the CNAME created at last step
kubectl run bb -it --rm --image busybox -- bin/sh
/ # nslookup hello-world-api.default.svc.cluster.local 10.0.0.10
Server:      10.0.0.10
Address:      10.0.0.10:53

hello-world-api.default.svc.cluster.local    canonical name = hello-world.api.example.com
```

### III. Configuring and managing application access with ingress

```
#Create nginx ingress on your cloud provider (azure/gcp/aws...)
https://github.com/hungtran84/k8s-cka/blob/master/d3\_networking/03\_ingress/nginx-ingress.yaml
kubectl apply -f nginx-ingress.yaml
namespace/ingress-nginx created
serviceaccount/ingress-nginx created
configmap/ingress-nginx-controller created
clusterrole.rbac.authorization.k8s.io/ingress-nginx created
clusterrolebinding.rbac.authorization.k8s.io/ingress-nginx created
role.rbac.authorization.k8s.io/ingress-nginx created
rolebinding.rbac.authorization.k8s.io/ingress-nginx created
```

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

#Check the status of the pods to see if the ingress controller is online.

```
kubectl get pods --namespace ingress-nginx
```

NAME	READY	STATUS	RESTARTS	AGE
ingress-nginx-admission-create-hnlsq	0/1	Completed	0	3m6s
ingress-nginx-admission-patch-bds42	0/1	Completed	1	3m5s
ingress-nginx-controller-69d6546d6d-b6lvp	1/1	Running	0	3m18s

#Now let's check to see if the service is online

```
kubectl get services --namespace ingress-nginx
```

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
ingress-nginx-controller	LoadBalancer	10.0.134.85	52.227.171.191	80:30697/TCP,443:30217/TCP	4m49s
ingress-nginx-controller-admission	ClusterIP	10.0.138.37	<none>	443/TCP	4m50s

#Create a deployment, scale it to 2 replicas and expose it as a service.

#This service will be ClusterIP and we'll expose this service via the Ingress.

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

```
kubectl scale deployment hello-world-service-single --replicas=2
```

```
kubectl expose deployment hello-world-service-single --port=80 --target-port=8080 --
type=ClusterIP
```

```
deployment.apps/hello-world-service-single created
```

```
deployment.apps/hello-world-service-single scaled
```

```
service/hello-world-service-single exposed
```

#Create a single Ingress routing to the one backend service on the service port 80

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

```
ingress.networking.k8s.io/ingress-single created
```

#Get the status of the ingress. It's routing for all host names on that public IP on port 80

kubectl get ingress

NAME	HOSTS	ADDRESS	PORTS	AGE
ingress-single	*	52.227.171.191	80	105s

kubectl get services --namespace ingress-nginx

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
ingress-nginx-controller	LoadBalancer	10.0.134.85	52.227.171.191	80:30697/TCP,443:30217/TCP	10m
ingress-nginx-controller-admission	ClusterIP	10.0.138.37	<none>	443/TCP	10m

kubectl describe ingress ingress-single

```

Name:          ingress-single
Namespace:     default
Address:       52.227.171.191
Default backend: hello-world-service-single:80 (10.244.0.14:8080,10.244.1.21:8080)
Rules:
  Host      Path  Backends
  ----      -
  *         *    hello-world-service-single:80 (10.244.0.14:8080,10.244.1.21:8080)
Annotations: Events:
  Type     Reason  Age   From              Message
  ----     -
  Normal   CREATE  6m52s  nginx-ingress-controller  Ingress default/ingress-single
  Normal   UPDATE  6m21s  nginx-ingress-controller  Ingress default/ingress-single

```

#Access the application via the exposed ingress on the public IP

```

INGRESSIP=$(kubectl get ingress -o jsonpath='{ .items[].status.loadBalancer.ingress[].ip }')
curl http://$INGRESSIP

```

Hello, world!

Version: 1.0.0

Hostname: hello-world-service-single-dc7d9bccf-jg9k8

#Create 2 additional services

```

kubectl create deployment hello-world-service-blue --image=gcr.io/google-samples/hello-app:1.0
kubectl create deployment hello-world-service-red --image=gcr.io/google-samples/hello-app:1.0

```

```

kubectl expose deployment hello-world-service-blue --port=4343 --target-port=8080 --
type=ClusterIP

```

```

kubectl expose deployment hello-world-service-red --port=4242 --target-port=8080 --
type=ClusterIP

```

deployment.apps/hello-world-service-blue created

```
deployment.apps/hello-world-service-red created
service/hello-world-service-blue exposed
service/hello-world-service-red exposed
```

```
#Create an ingress with paths each routing to different backend services.
```

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

```
https://github.com/hungtran84/k8s-cka/blob/master/d3\_networking/03\_ingress/ingress-path.yaml
ingress.networking.k8s.io/ingress-path created
```

```
kubectl get ing
```

NAME	HOSTS	ADDRESS	PORTS	AGE
ingress-path	path.example.com	52.227.171.191	80	78s
ingress-single	*	52.227.171.191	80	13m

```
# tada!!!
```

```
curl http://$INGRESSIP/red --header 'Host: path.example.com'
```

```
Hello, world!
```

```
Version: 1.0.0
```

```
Hostname: hello-world-service-red-56cc7b86b-gc76n
```

```
curl http://$INGRESSIP/blue --header 'Host: path.example.com'
```

```
Hello, world!
```

```
Version: 1.0.0
```

```
Hostname: hello-world-service-blue-7647475b7-x9z5w
```

```
#Add a backend to the ingress listening on path.example.com pointing to the single service
https://github.com/hungtran84/k8s-cka/blob/master/d3\_networking/03\_ingress/ingress-path-backend.yaml
```

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

```
ingress.networking.k8s.io/ingress-path configured
```

```
#Hit the default backend service, single
```

```
curl http://$INGRESSIP/ --header 'Host: path.example.com'
```

```
Hello, world!
```

```
Version: 1.0.0
```

```
Hostname: hello-world-service-single-dc7d9bccf-mq675
```

```
#Route traffic to the services using named based virtual hosts rather than paths
```

```
https://github.com/hungtran84/k8s-cka/blob/master/d3\_networking/03\_ingress/ingress-namebased.yaml
```

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

```
ingress.networking.k8s.io/ingress-namebased created
```

```
curl http://$INGRESSIP/ --header 'Host: red.example.com'
```

```
Hello, world!
```



Version: 1.0.0

Hostname: hello-world-service-red-56cc7b86b-dtprz

```
curl http://$INGRESSIP/ --header 'Host: blue.example.com'
```

Hello, world!

Version: 1.0.0

Hostname: hello-world-service-blue-7647475b7-hzcn9

**#Try a name based virtual host that doesn't exist**

```
curl http://$INGRESSIP/ --header 'Host: tel4vn.edu.vn'
```

Hello, world!

Version: 1.0.0

Hostname: hello-world-service-single-dc7d9bccf-jg9k8

**#TLS**

**#1 - Generate a certificate**

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 \
-keyout tls.key -out tls.crt -subj
"/C=VN/ST=HCM/L=HCMC/O=IT/OU=IT/CN=tls.example.com"
```

Generating a 2048 bit RSA private key

.....+++

.....+++

writing new private key to 'tls.key'

-----

**#2 - Create a secret with the key and the certificate**

```
kubectl create secret tls tls-secret --key tls.key --cert tls.crt
secret/tls-secret created
```

**#3 - Create an ingress using the certificate and key.**

```
kubectl apply -f ingress-tls.yaml
ingress.networking.k8s.io/ingress-tls created
```

```
curl https://tls.example.com:443 --resolve tls.example.com:443:$INGRESSIP --insecure --verbose
```

\* Added tls.example.com:443:52.227.171.191 to DNS cache

\* Hostname tls.example.com was found in DNS cache

\* Trying 52.227.171.191...

\* TCP\_NODELAY set

\* Connected to tls.example.com (52.227.171.191) port 443 (#0)

\* ALPN, offering h2

\* ALPN, offering http/1.1

\* successfully set certificate verify locations:

\* CAfile: /etc/ssl/cert.pem

CAPath: none

```

* TLSv1.2 (OUT), TLS handshake, Client hello (1):
* TLSv1.2 (IN), TLS handshake, Server hello (2):
* TLSv1.2 (IN), TLS handshake, Certificate (11):
* TLSv1.2 (IN), TLS handshake, Server key exchange (12):
* TLSv1.2 (IN), TLS handshake, Server finished (14):
* TLSv1.2 (OUT), TLS handshake, Client key exchange (16):
* TLSv1.2 (OUT), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (OUT), TLS handshake, Finished (20):
* TLSv1.2 (IN), TLS change cipher, Change cipher spec (1):
* TLSv1.2 (IN), TLS handshake, Finished (20):
* SSL connection using TLSv1.2 / ECDHE-RSA-AES128-GCM-SHA256
* ALPN, server accepted to use h2
* Server certificate:
* subject: C=VN; ST=HCM; L=HCMC; O=IT; OU=IT; CN=tls.example.com
* start date: Jul 20 10:57:21 2020 GMT
* expire date: Jul 20 10:57:21 2021 GMT
* issuer: C=VN; ST=HCM; L=HCMC; O=IT; OU=IT; CN=tls.example.com
* SSL certificate verify result: self signed certificate (18), continuing anyway.
* Using HTTP2, server supports multi-use
* Connection state changed (HTTP/2 confirmed)
* Copying HTTP/2 data in stream buffer to connection buffer after upgrade: len=0
* Using Stream ID: 1 (easy handle 0x7f9e9a80a200)
> GET / HTTP/2
> Host: tls.example.com
> User-Agent: curl/7.64.1
> Accept: */*
>
* Connection state changed (MAX_CONCURRENT_STREAMS == 128)!
< HTTP/2 200
< server: nginx/1.17.10
< date: Mon, 20 Jul 2020 11:01:03 GMT
< content-type: text/plain; charset=utf-8
< content-length: 82
< strict-transport-security: max-age=15724800; includeSubDomains
<
Hello, world!
Version: 1.0.0
Hostname: hello-world-service-single-dc7d9bccf-mq675
* Connection #0 to host tls.example.com left intact
* Closing connection 0

#Clean up from our demo
kubectl delete ingress ingress-path
kubectl delete ingress ingress-tls

```

```
kubectl delete ingress ingress-namebased  
kubectl delete deployment hello-world-service-single  
kubectl delete deployment hello-world-service-red  
kubectl delete deployment hello-world-service-blue  
kubectl delete service hello-world-service-single  
kubectl delete service hello-world-service-red  
kubectl delete service hello-world-service-blue  
kubectl delete secret tls-secret
```

**#Delete the ingress, ingress controller and other configuration elements**

```
kubectl delete -f nginx-ingress.yaml
```