K8S (5) Service

Service:

In order to connect to application, we need to connect to pod (remember, the container is inside pod). Because the pod is ephemeral (Any time we create new replicaSet, the old pods are gone. New pod will have new IP address), we cannot assign static IP to pods.
=> Use Service to group pods and access them. The grouping is by Labels and Selectors.
Service is also a K8S object, coordinating (1) the communication between Micro service inside cluster (2) communication with external world
Service will maintain (1) DNS (2) Load balancer

```
# service.yaml
# This frontend-svc Service selects all pods with label "app: frontend"
# The default type of Service is CluserIP so this Service will be assigned a IP (only
works inside cluster)
# Anyone connect to this Service, the traffic will be forward to one of the pods.
(Load Balancer will select which pod)
#  Every pod has its own IP address, adding targetPort will become this pod's
endpoint
# Endpoints are created and managed by Service, not K8S admin
apiVersion: v1
kind: Service
metadata:
  name: frontend-svc
spec:
  selector:
    app: frontend
  ports:
  - protocol: TCP
    port: 80 # Service listen on port:80 and receive traffic from outside
    targetPort: 5000 # traffic will be forward to pod on port:5000. The open port of
pod should match this
```

kube-proxy:

Runs on every node (CP node and worker node), implement service's configuration based on API server. Any create/update/remove command from API server will relay to kube-procy.
kube-proxy maintains a iptable to capture the request to ClusterIP.

Downside of kube-proxy:

kube-proxy will main the iptable, any request to Service will be routed to the application endpoints. But this load balancing is random, which means there is no control which pod will receive the request. There might be some pod replicas running in other node, kube-proxy might choose this away pod and ignore there is

a pod running on the same node with kube-proxy (which is closer and faster). We use Traffic Policy to config how kube-proxy routing.
Two types of Traffic Routing (1) Cluster: kube-proxy will look for any ready endpoints (2) Local: Only target the endpoint on the same node with request pod. Notice that the Local option is local-only, kube-proxy will not route request to pods on any other node.

Service discovery:

Service handles the communication of containerized applications, two methods the get the service at runtime:
(1) Environment variables: When pod is created, the kubelet on the same node will set environment variable. Record the service config.
Notice that this if the service is created after the pod, kubelet will not capture this.
(2) DNS (Recommended): Every service has a full FQDN (Fully qualified domain name, like **my-svc.my-namespace.svc.cluster.local**).

Service types:

Define scope of service, either (1) Only accessible in cluster (2) Accessible in cluster and external (3) Service mapping to other entities
Several types:
(1) ClusterIP: Service is assigned a Virtual IP (ClusterIP). Inside cluster ony.
(2) NodePort: Adding high-port ([30000:32767])to ClusterIP. Open to external world.
(3) Load Balancer: The LB type Service will create both ClusterIP and NodePort. There is a static port exposing this Service on every node. The Service is exposed to external world by LoadBalancer.
(4) External IP: Service is mapping to an External IP address. This requires external cloud provider(GCP or AWS).
(5) External Name: No selectors and no endpoint defined. Use CNAME to represent Service.

Multi-Port Services:

This is handy when you have multiple pods in your service. These pods and containers listens to different port.

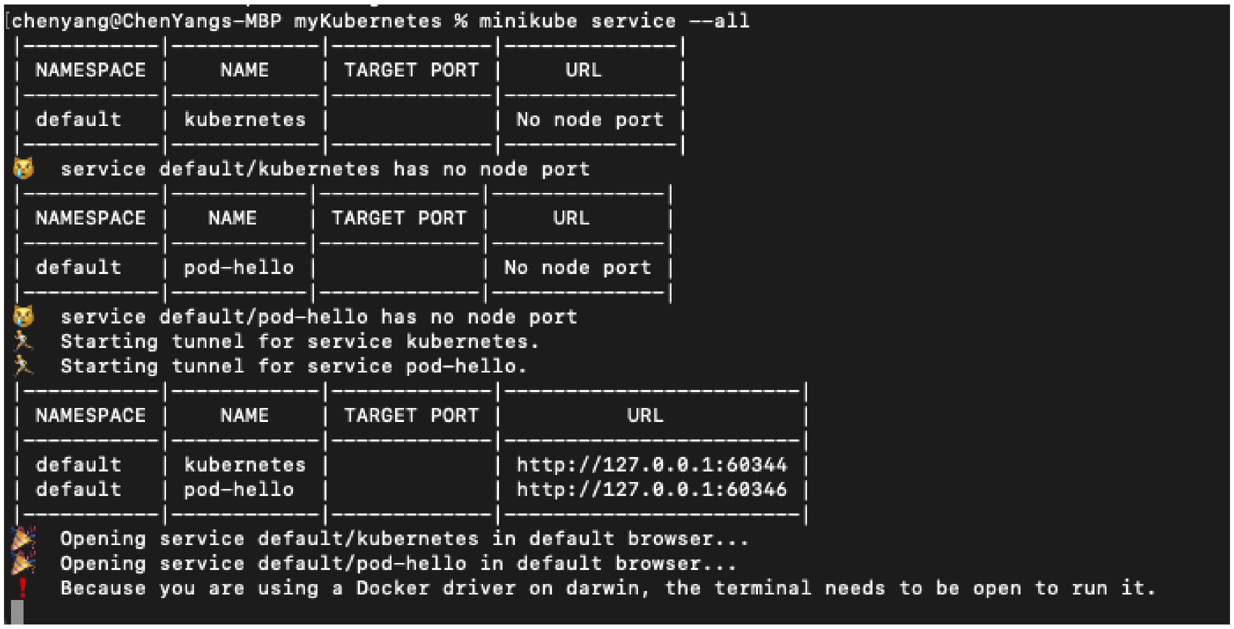
```
# multiPortService.yaml
# The **my-service** Service exposes Pods labeled **app==myapp** (there might be many app==myapp pods)
# One container listening on ports **80** another is listening on **443** (defined in targetPort, the container should have the same port listening)
# In clulster, **my-service** Service is accessible through **ClusterIP:8080** and **ClusterIP:8443** (defined in port)
# From external, **my-service** Service is accessible on nodePort **31080** and **31443**
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: myapp
  type: NodePort
  ports:
  - name: http
    protocol: TCP
    port: 8080
    targetPort: 80
    nodePort: 31080
  - name: https
    protocol: TCP
    port: 8443
    targetPort: 443
    nodePort: 31443
```

Demo of ClusterIP and NodePort type:

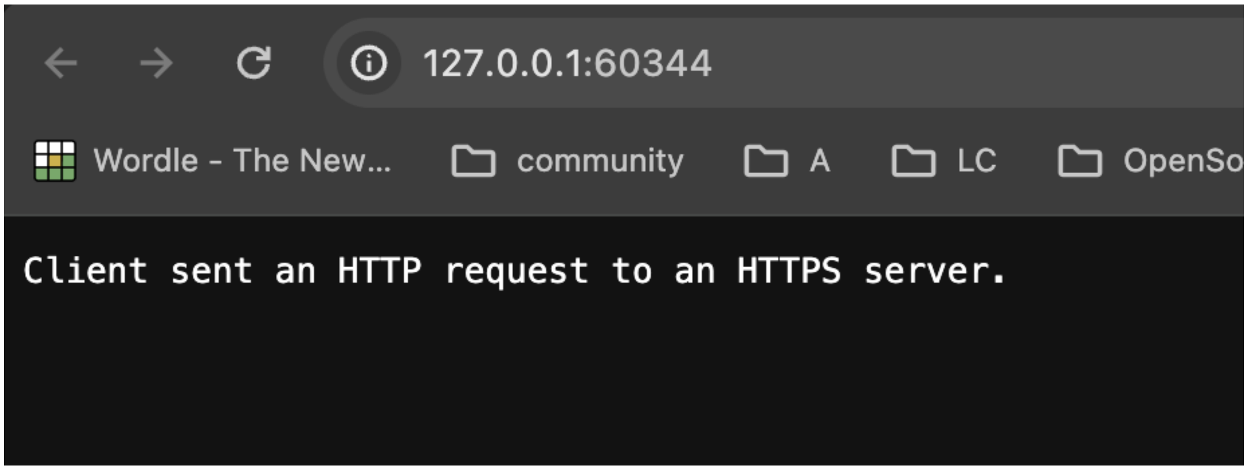
```
$ kubectl run pod-hello --image=pbitty/hello-from:latest --port=80 --expose=true
// Run a pod. Also comes with a service since we choose to expose the pod.
service/pod-hello created
```

```
pod/pod-hello created

$ kubectl get pod,svc,ep --show-labels
// check the label, the pod's label is run=pod-hello
NAME READY STATUS RESTARTS AGE LABELS
pod/pod-hello 1/1 Running 0 2m5s run=pod-hello

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE LABELS
service/kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 36h
component=apiserver,provider=kubernetes
service/pod-hello ClusterIP 10.100.218.44 <none> 80/TCP 2m4s <none>

NAME ENDPOINTS AGE LABELS
endpoints/kubernetes 192.168.49.2:8443 36h endpointslice.kubernetes.io/skip-
mirror=true
endpoints/pod-hello 10.244.151.5:80 2m4s <none>

$ kubectl describe svc pod-hello | grep -i selector // Verify the the selector of
service set to our pod's label run=pod-hello
Selector: run=pod-hello


$ kubectl describe pod pod-hello | grep -i podip:
// Verify the endpoint of pod-hello (10.244.151.5:80) will inherit the IP address of
pod/pod-hello
// Dump the detail of pod/pod-hello
// In the pod definition, the address matches the endpoint IP (10.244.151.5:80)
cni.projectcalico.org/podIP: 10.244.151.5/32

$ minikube service --all
// list all available service in cluster
// Since the pod-hello is ClusterIP type, it is not expose to external
// In order to validate the service works, we need to log in to the node
```

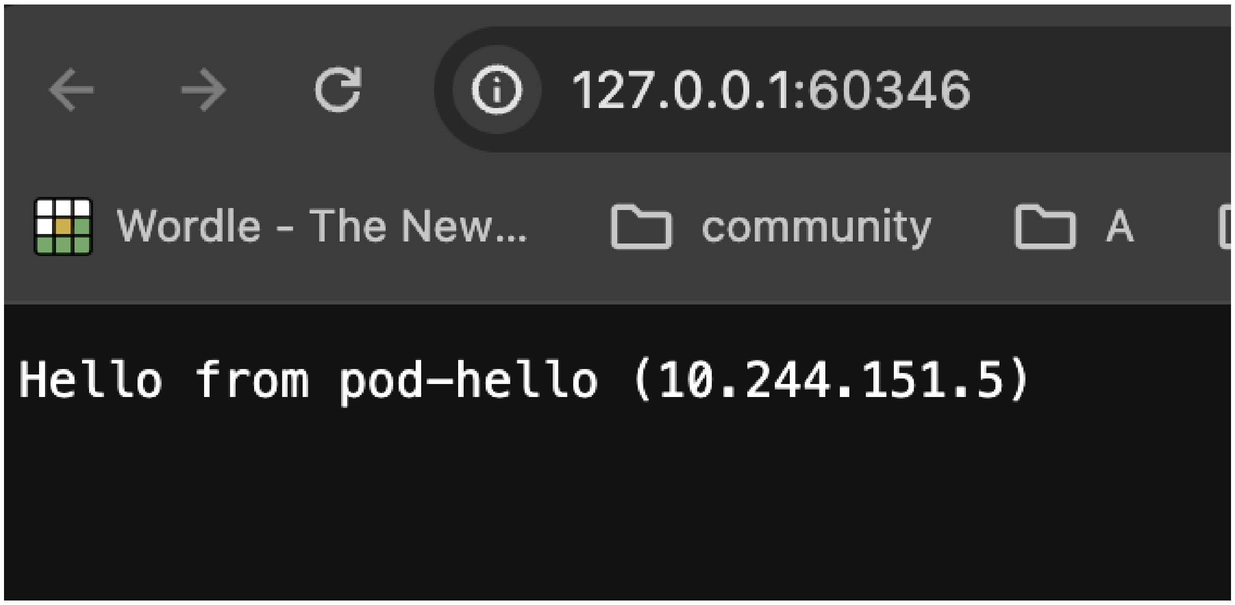
But in my situation, the minikube create a tunnel and provide a URL at http://127.0.0.1:60346



```
[chenyang@ChenYangs-MBP myKubernetes % minikube service --all
|-----------|------------|-------------|--------------|
| NAMESPACE |    NAME    | TARGET PORT |     URL      |
|-----------|------------|-------------|--------------|
| default   | kubernetes |             | No node port |
|-----------|------------|-------------|--------------|
😿  service default/kubernetes has no node port
|-----------|-----------|-------------|--------------|
| NAMESPACE |   NAME    | TARGET PORT |     URL      |
|-----------|-----------|-------------|--------------|
| default   | pod-hello |             | No node port |
|-----------|-----------|-------------|--------------|
😿  service default/pod-hello has no node port
🏃  Starting tunnel for service kubernetes.
🏃  Starting tunnel for service pod-hello.
|-----------|------------|-------------|------------------------|
| NAMESPACE |    NAME    | TARGET PORT |          URL           |
|-----------|------------|-------------|------------------------|
| default   | kubernetes |             | http://127.0.0.1:60344 |
| default   | pod-hello  |             | http://127.0.0.1:60346 |
|-----------|------------|-------------|------------------------|
🎉  Opening service default/kubernetes in default browser...
🎉  Opening service default/pod-hello in default browser...
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```

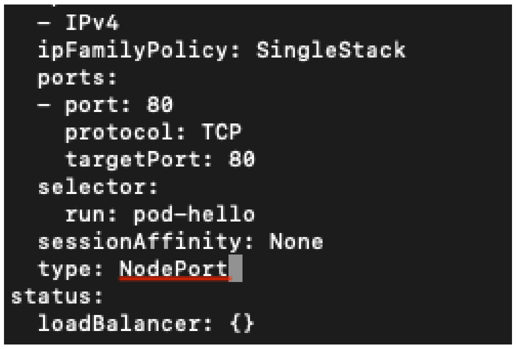


```
Client sent an HTTP request to an HTTPS server.
```



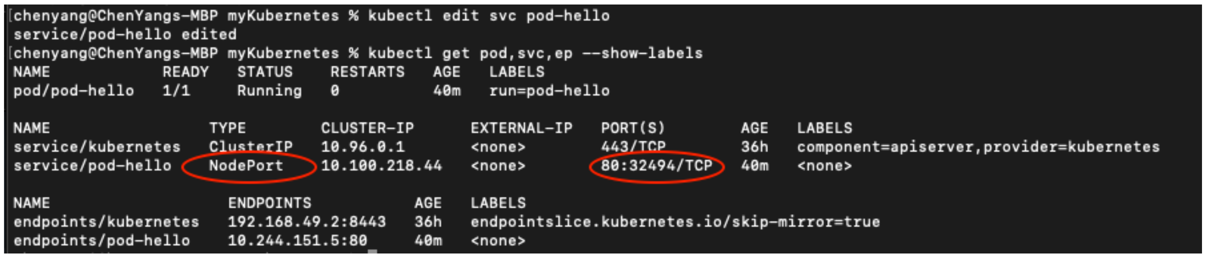
```
Hello from pod-hello (10.244.151.5)
```

```
% minikube ssh
docker@minikube:~$ curl 10.244.151.5
Hello from pod-hello (10.244.151.5)
docker@minikube:~$ curl 10.244.151.5
Hello from pod-hello (10.244.151.5)
docker@minikube:~$ curl 10.244.151.5
Hello from pod-hello (10.244.151.5)
docker@minikube:~$ curl 10.244.151.5
Hello from pod-hello (10.244.151.5)
// ssh into the node and curl the endpoint IP address, get response from the pod

docker@minikube:~$ curl 10.100.218.44
Hello from pod-hello (10.244.151.5)
docker@minikube:~$ curl 10.100.218.44
Hello from pod-hello (10.244.151.5)
docker@minikube:~$ curl 10.100.218.44
Hello from pod-hello (10.244.151.5)
docker@minikube:~$ curl 10.100.218.44
Hello from pod-hello (10.244.151.5)
// curl the Service IP address, also get the same reponse (response from the pod)
// At this point, we can confirm the Service is sending the request to its endpoint
// But only within cluster, we need to change service type to NodePort to gain
access from external

$ kubectl edit svc pod-hello // update the service
service/pod-hello edited
```



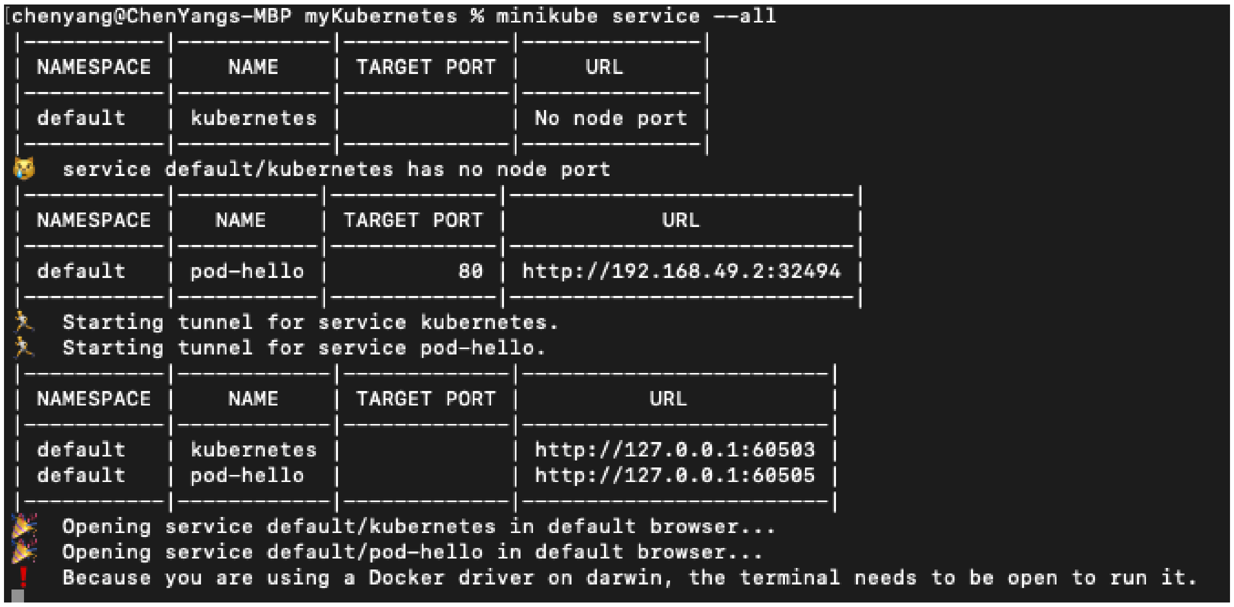
```
  - IPv4
  ipFamilyPolicy: SingleStack
  ports:
  - port: 80
    protocol: TCP
    targetPort: 80
  selector:
    run: pod-hello
  sessionAffinity: None
  type: NodePort
status:
  loadBalancer: {}
```

List out the Service. You can see the type has been update, and a high-port is assigned to service pod-hello.

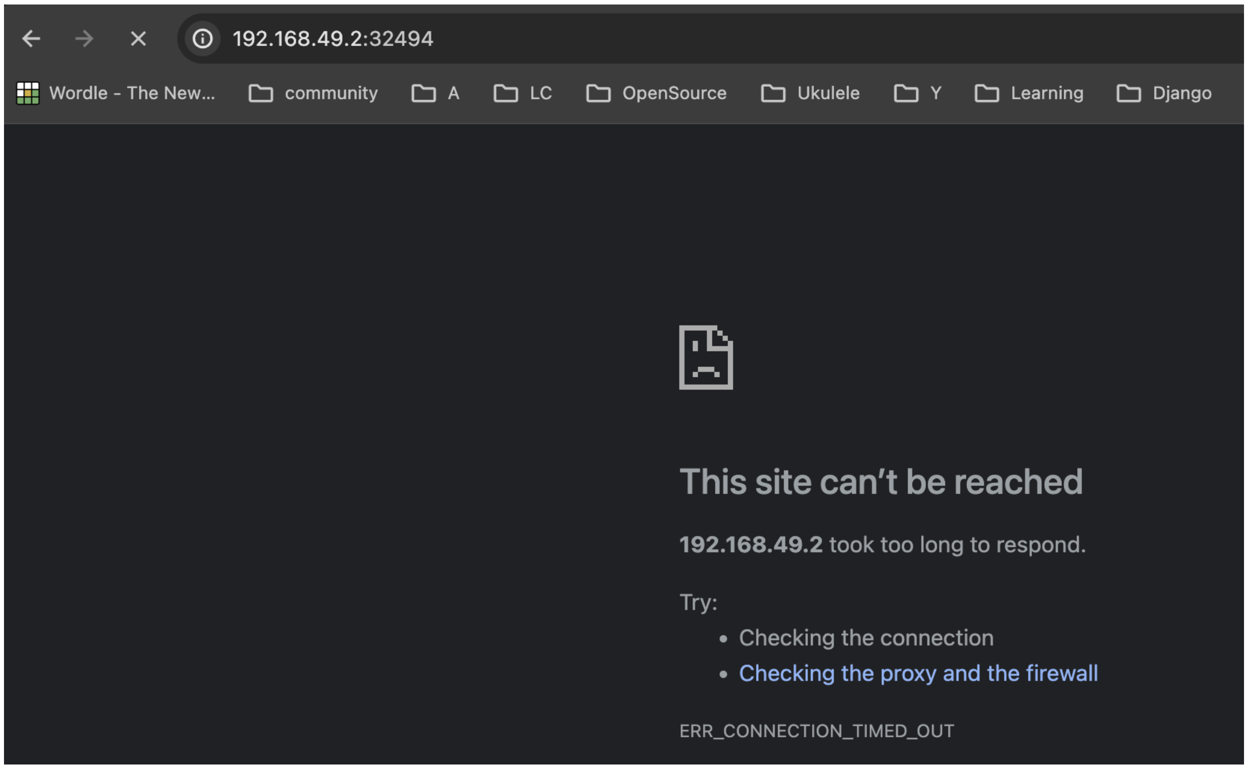

```
chenyang@ChenYangs-MBP myKubernetes % kubectl edit svc pod-hello
service/pod-hello edited
chenyang@ChenYangs-MBP myKubernetes % kubectl get pod,svc,ep --show-labels
NAME            READY   STATUS    RESTARTS   AGE   LABELS
pod/pod-hello   1/1     Running   0          40m   run=pod-hello

NAME                   TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)        AGE   LABELS
service/kubernetes     ClusterIP   10.96.0.1       <none>        443/TCP        36h   component=apiserver,provider=kubernetes
service/pod-hello      NodePort    10.100.218.44   <none>        80:32494/TCP   40m   <none>

NAME                    ENDPOINTS           AGE   LABELS
endpoints/kubernetes    192.168.49.2:8443   36h   endpointslice.kubernetes.io/skip-mirror=true
endpoints/pod-hello     10.244.151.5:80     40m   <none>
```

```
$ minikube service --all
// There is a url to access service
```

But in my case, still opening a accessible tunnel and the URL is no response



```
[chenyang@ChenYangs-MBP myKubernetes % minikube service --all
|-----------|------------|-------------|--------------|
| NAMESPACE |    NAME    | TARGET PORT |     URL      |
|-----------|------------|-------------|--------------|
| default   | kubernetes |             | No node port |
|-----------|------------|-------------|--------------|
😿  service default/kubernetes has no node port
|-----------|-----------|-------------|---------------------------|
| NAMESPACE |   NAME    | TARGET PORT |            URL            |
|-----------|-----------|-------------|---------------------------|
| default   | pod-hello |          80 | http://192.168.49.2:32494 |
|-----------|-----------|-------------|---------------------------|
🏃  Starting tunnel for service kubernetes.
🏃  Starting tunnel for service pod-hello.
|-----------|------------|-------------|------------------------|
| NAMESPACE |    NAME    | TARGET PORT |          URL           |
|-----------|------------|-------------|------------------------|
| default   | kubernetes |             | http://127.0.0.1:60503 |
| default   | pod-hello  |             | http://127.0.0.1:60505 |
|-----------|------------|-------------|------------------------|
🎉  Opening service default/kubernetes in default browser...
🎉  Opening service default/pod-hello in default browser...
❗  Because you are using a Docker driver on darwin, the terminal needs to be open to run it.
```



This site can't be reached

192.168.49.2 took too long to respond.

Try:

- Checking the connection
- Checking the proxy and the firewall

ERR_CONNECTION_TIMED_OUT

The same, didn't expose the IP to external. Only the tunnel works. Seems like this is known issue for Docker Desktop: https://github.com/kubernetes/minikube/issues/11193#issuecomment-826331511
Tried to use Qeum as driver to continue the demo...

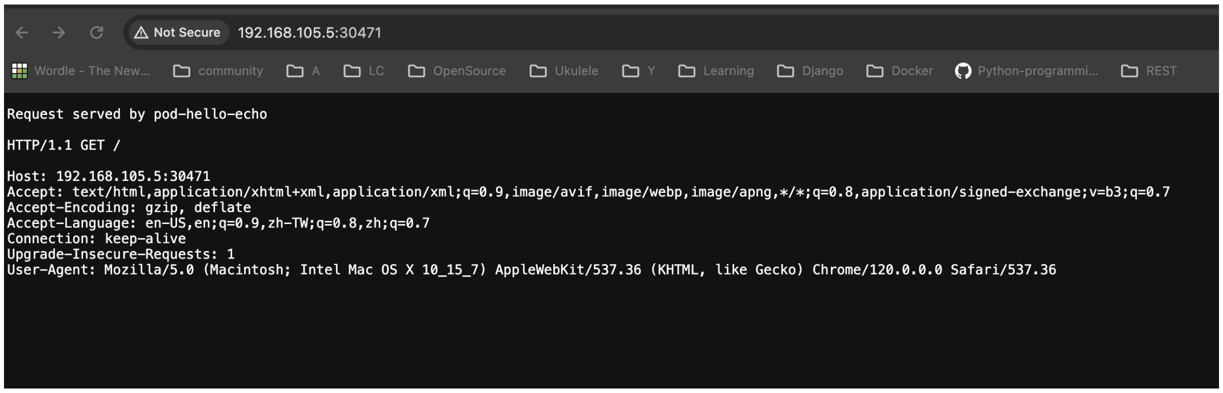
Demo of ClusterIP and NodePort type in Qemu:

Change to qemu and redo everything. Setup refer to
https://devopscube.com/minikube-mac/
We don't use the **pbitty/hello-from:latest** because it doesn't support on arm64 chip. Using **kicbase/echo-server:1.0** instead.

```
$ minikube start --driver=qemu --network socket_vmnet --network-plugin=cni --
cni=calico
😄  minikube v1.32.0 on Darwin 14.2 (arm64)
✨  Using the qemu2 driver based on user configuration
❗  With --network-plugin=cni, you will need to provide your own CNI. See --cni flag
as a user-friendly alternative
👍  Starting control plane node minikube in cluster minikube
🔥  Creating qemu2 VM (CPUs=4, Memory=2200MB, Disk=20000MB) ...
🐳  Preparing Kubernetes v1.28.3 on Docker 24.0.7 ...
▪ Generating certificates and keys ...
▪ Booting up control plane ...
▪ Configuring RBAC rules ...
🔗  Configuring Calico (Container Networking Interface) ...
▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
🔎  Verifying Kubernetes components...
🌟  Enabled addons: default-storageclass, storage-provisioner
🏄  Done! kubectl is now configured to use "minikube" cluster and "default" namespace
by default

$ kubectl run pod-hello-echo --image=kicbase/echo-server:1.0 --port=8080 --
expose=true
service/pod-hello created
pod/pod-hello created

$ kubectl get pod,svc,ep --show-labels
NAME READY STATUS RESTARTS AGE LABELS
pod/pod-hello-echo 1/1 Running 0 2s run=pod-hello-echo
NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE LABELS
service/pod-hello-echo ClusterIP 10.99.122.5 <none> 8080/TCP 2s <none>
NAME ENDPOINTS AGE LABELS
endpoints/pod-hello-echo 10.244.120.69:8080 2s <none>

$ kubectl describe svc pod-hello-echo | grep -i selector
Selector: run=pod-hello-echo
$ kubectl describe pod pod-hello-echo | grep -i podip:
cni.projectcalico.org/podIP: 10.244.120.69/32
// Everything in place. pods
// Verify the endpoint of pod-hello (10.244.120.69:8080) will inherit the IP address
of pod/pod-hello,
// and the service will select the pod/pod-hello
```

```
_ _
_ _ ( ) ( )
___ ___ (_) ___ (_)| |/') _ _ | |_ __
/' _ ` _ `\| |/' _ `\| || , < ( ) ( )| '_`\ /'__`\
| ( ) ( ) || || ( ) || || |\`\ | (_) || |_) )( ___/
(_) (_) (_)(_)(_) (_)(_)(_) (_)`\___/'(_,__/'`\____)

$ curl 10.244.120.69:8080
Request served by pod-hello-echo

HTTP/1.1 GET /

Host: 10.244.120.69:8080
Accept: */*
User-Agent: curl/7.79.1

$ curl 10.99.122.5:8080
Request served by pod-hello-echo

HTTP/1.1 GET /

Host: 10.99.122.5:8080
Accept: */*
User-Agent: curl/7.79.1

// curl Service IP and endpoint IP, both resposne

$ minikube service list
|-------------|----------------|--------------|-----|
| NAMESPACE | NAME | TARGET PORT | URL |
|-------------|----------------|--------------|-----|
| default | kubernetes | No node port | |
| default | pod-hello-echo | No node port | |
| kube-system | kube-dns | No node port | |
|-------------|----------------|--------------|-----|
$ minikube service pod-hello-echo
|-----------|----------------|-------------|--------------|
| NAMESPACE | NAME | TARGET PORT | URL |
|-----------|----------------|-------------|--------------|
| default | pod-hello-echo | | No node port |
|-----------|----------------|-------------|--------------|
😿 service default/pod-hello-echo has no node port

// Access only within cluster, we need to change service type to NodePort to gain
access from external

$ kubectl edit svc pod-hello-echo // update the service
```

```
service/pod-hello-echo edited

$ kubectl get pod,svc,ep --show-labels
NAME READY STATUS RESTARTS AGE LABELS
pod/pod-hello-echo 1/1 Running 0 7m23s run=pod-hello-echo

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE LABELS
service/kubernetes ClusterIP 10.96.0.1 <none> 443/TCP 28m
component=apiserver,provider=kubernetes
service/pod-hello-echo NodePort 10.99.122.5 <none> 8080:30471/TCP 7m23s <none>

NAME ENDPOINTS AGE LABELS
endpoints/kubernetes 192.168.105.5:8443 28m endpointslice.kubernetes.io/skip-
mirror=true
endpoints/pod-hello-echo 10.244.120.69:8080 7m23s <none>

$ minikube service list
|-------------|----------------|--------------|----------------------------|
| NAMESPACE | NAME | TARGET PORT | URL |
|-------------|----------------|--------------|----------------------------|
| default | kubernetes | No node port | |
| default | pod-hello-echo | 8080 | http://192.168.105.5:30471 |
| kube-system | kube-dns | No node port | |
|-------------|----------------|--------------|----------------------------|
//

$ minikube service pod-hello-echo
|-----------|----------------|-------------|----------------------------|
| NAMESPACE | NAME | TARGET PORT | URL |
|-----------|----------------|-------------|----------------------------|
| default | pod-hello-echo | 8080 | http://192.168.105.5:30471 |
|-----------|----------------|-------------|----------------------------|
🎉 Opening service default/pod-hello-echo in default browser...
```

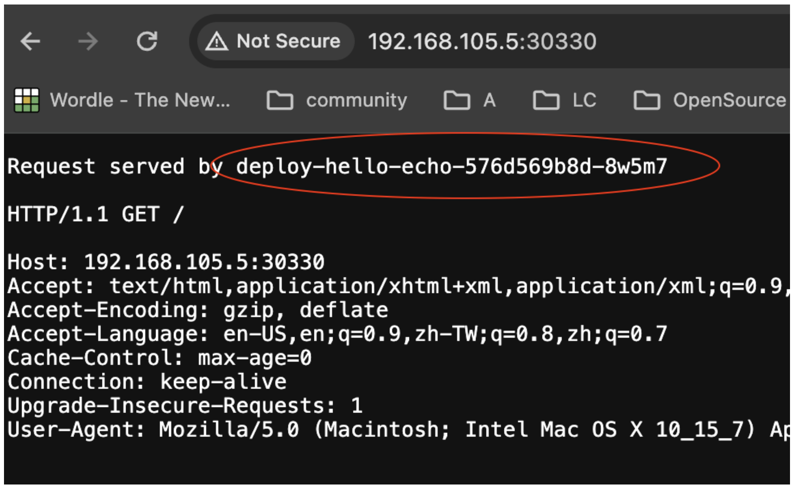


```
Request served by pod-hello-echo

HTTP/1.1 GET /

Host: 192.168.105.5:30471
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,zh-TW;q=0.8,zh;q=0.7
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/120.0.0.0 Safari/537.36
```

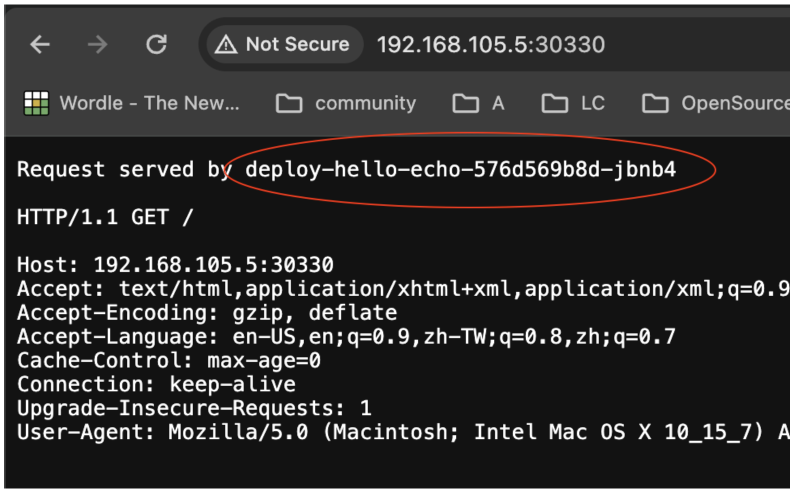
Try it on multiple replicas application. Expose deployment through Service.

```
$ kubectl create deployment deploy-hello-echo --image=kicbase/echo-server:1.0 --
port=8080 --replicas=3
deployment.apps/deploy-hello-echo created

//We can expose Deployment through Service, just not in the create command, run
another expose command
$ kubectl expose deployment deploy-hello-echo --type=NodePort
service/deploy-hello-echo exposed

$ kubectl get pod,svc,ep,deploy -l app=deploy-hello-echo --show-labels
// Only dump a certain label
// 3 pods in place as we request during creation
NAME READY STATUS RESTARTS AGE LABELS
pod/deploy-hello-echo-576d569b8d-8w5m7 1/1 Running 0 2m21s app=deploy-hello-
echo,pod-template-hash=576d569b8d
pod/deploy-hello-echo-576d569b8d-jbnb4 1/1 Running 0 2m21s app=deploy-hello-
echo,pod-template-hash=576d569b8d
pod/deploy-hello-echo-576d569b8d-tl47h 1/1 Running 0 2m21s app=deploy-hello-
echo,pod-template-hash=576d569b8d

NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S) AGE LABELS
service/deploy-hello-echo NodePort 10.108.65.196 <none> 8080:30330/TCP 30s
app=deploy-hello-echo

NAME ENDPOINTS AGE LABELS
endpoints/deploy-hello-echo 10.244.120.70:8080,10.244.120.71:8080,10.244.120.72:8080
30s app=deploy-hello-echo

NAME READY UP-TO-DATE AVAILABLE AGE LABELS
deployment.apps/deploy-hello-echo 3/3 3 3 2m21s app=deploy-hello-echo

$ minikube service list
|-------------|-------------------|--------------|----------------------------|
| NAMESPACE | NAME | TARGET PORT | URL |
|-------------|-------------------|--------------|----------------------------|
| default | deploy-hello-echo | 8080 | http://192.168.105.5:30330 |
| default | kubernetes | No node port | |
| default | pod-hello-echo | 8080 | http://192.168.105.5:30471 |
| kube-system | kube-dns | No node port | |
|-------------|-------------------|--------------|----------------------------|
```

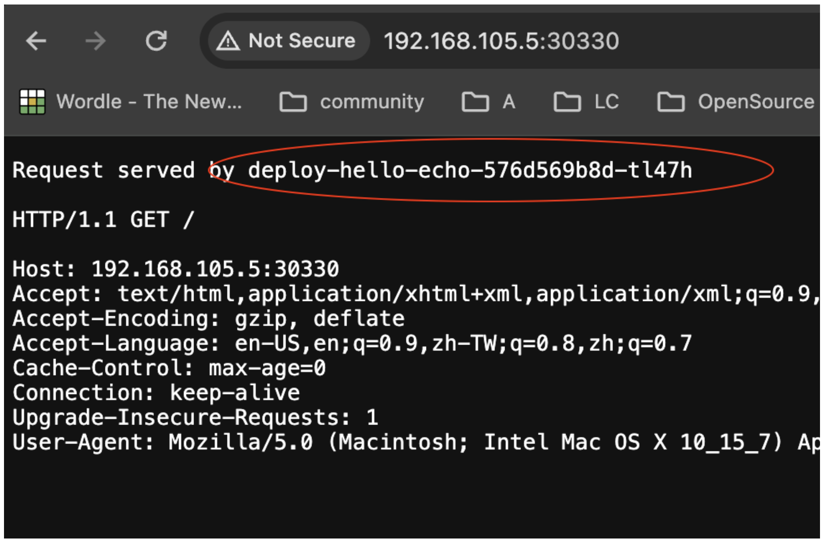
Visited http://192.168.105.5:30330/. The kicbase/echo-server didn't show the endpoint IP address but shows the name of the pod.
If you reload the URL couple of times, you can see the request is actually processed by different endpoints.
The Service is load balancing between service's 3 pods.



```
Request served by deploy-hello-echo-576d569b8d-8w5m7

HTTP/1.1 GET /

Host: 192.168.105.5:30330
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,zh-TW;q=0.8,zh;q=0.7
Cache-Control: max-age=0
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) Ap
```



```
Request served by deploy-hello-echo-576d569b8d-jbnb4

HTTP/1.1 GET /

Host: 192.168.105.5:30330
Accept: text/html,application/xhtml+xml,application/xml;q=0.9
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,zh-TW;q=0.8,zh;q=0.7
Cache-Control: max-age=0
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) A
```



```
Request served by deploy-hello-echo-576d569b8d-tl47h

HTTP/1.1 GET /

Host: 192.168.105.5:30330
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,
Accept-Encoding: gzip, deflate
Accept-Language: en-US,en;q=0.9,zh-TW;q=0.8,zh;q=0.7
Cache-Control: max-age=0
Connection: keep-alive
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) Ap
```