Kubernetes Networking

RA Owned by Raghavendra Addanki •••
Last updated: May 15, 2023

Kubernetes Internal Cluster Networking

- Kubernetes creates an Internal Cluster with a specific IP Subnet.
- The IPs in the cluster are accessible by all PODs in the Cluster.

How to find this Cluster IP Range

`kubectl describe cm kubeadm-config -n kube-system |grep Subnet`

```
linuxadmin@master:~$ kubectl describe cm kubeadm-config -n kube-system |grep Subnet
  serviceSubnet: 10.96.0.0/12
```

- When you create a Kubernetes Cluster, The Master Node that runs the API Server, will take the First available IP and sets that IP as Kubernetes Cluster IP.
- If any Node or POD want to communicate with Kubernetes API Server, they will use this IP.

How to see the Kubernetes Default Cluster IP

`kubectl get service --all-namespaces`

```
linuxadmin@master:~$ kubectl get service --all-namespaces
NAMESPACE     NAME         TYPE        CLUSTER-IP   EXTERNAL-IP   PORT(S)                  AGE
default       kubernetes   ClusterIP   10.96.0.1    <none>        443/TCP                  179m
kube-system   kube-dns     ClusterIP   10.96.0.10   <none>        53/UDP,53/TCP,9153/TCP   179m

```

- As you can see kubernetes API Service is using IP Address: `ClusterIP  10.96.0.1`

Internally this IP is mapped to the actual Master Node IP:

`kubectl get endpoints kubernetes`

```
linuxadmin@master:~$ kubectl get endpoints kubernetes
NAME         ENDPOINTS           AGE
kubernetes   192.168.1.50:6443   3h7m
```

Kubernetes POD Networking

- POD have their own IP Subnet Ranges.
- They are not part of this Kubernetes Service Subnet: `serviceSubnet: 10.96.0.0/12`
- Each Worker Node is allotted a POD Network Range.
- When a POD is created on a particular Worker Node, The POD inherits one of the IP in the POD IP Range.

How to find POD IP Range

`kubectl get ipamblocks.crd.projectcalico.org`

```
linuxadmin@master:~$ kubectl get ipamblocks.crd.projectcalico.org
NAME               AGE
172-16-104-0-26    57m
172-16-166-128-26  57m
172-16-219-64-26   58m
```

- But we the above output did not tell us which POD has got what IP Range.

How to find individual Worker Node POD IP Range

`kubectl get ipamblocks.crd.projectcalico.org -o jsonpath="{range .items[*]}{'podNetwork: '}{.spec.cidr}{'\t NodeIP: '}{.spec.affinity}{'\n'}"`

```
linuxadmin@master:~$ kubectl get ipamblocks.crd.projectcalico.org -o jsonpath="{range .items[*]}{'podNetwork: '}{
podNetwork: 172.16.104.0/26      NodeIP: host:node2
podNetwork: 172.16.166.128/26    NodeIP: host:node1
podNetwork: 172.16.219.64/26     NodeIP: host:master
```

- Any POD created on **node1** will get an IP Address in the Range: `172.16.166.128/26`

`podNetwork: 172.16.166.128/26    NodeIP: host:node1`

- Any POD created on **node2** will get an IP Address in the Range: `172.16.104.0/26`

`podNetwork: 172.16.104.0/26    NodeIP: host:node2`

Create some PODs and verify

First POD:

`sudo nano ssl-website1.yaml`

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-ssl-website1
spec:
  containers:
    - name: con-ssl-website1
      image: tanvisinghny/ssl-website
      ports:
        - containerPort: 80
        - containerPort: 443
```

```
kubectl create -f ssl-website1.yaml
```

Second POD:

`sudo nano ssl-website2.yaml`

```
apiVersion: v1
kind: Pod
metadata:
  name: pod-ssl-website2
spec:
  containers:
    - name: con-ssl-website2
      image: tanvisinghny/ssl-website
      ports:
        - containerPort: 80
        - containerPort: 443
```

```
kubectl create -f ssl-website2.yaml
```

Get the Individual POD IPs

```
linuxadmin@master:~$ kubectl get pods -o wide
NAME              READY   STATUS    RESTARTS   AGE   IP               NODE    NOMINATED NODE   READINESS GATES
pod-ssl-website1   1/1     Running   0          16s   172.16.166.130   node1   <none>           <none>
pod-ssl-website2   1/1     Running   0          11s   172.16.104.2     node2   <none>           <none>
```

POD on node1 gets IP address: `172.16.166.130`

POD on node2 gets IP address: `172.16.104.2`

Compare with POD IP Range we discussed earlier:

```
podNetwork: 172.16.104.0/26      NodeIP: host:node2
podNetwork: 172.16.166.128/26    NodeIP: host:node1
```

Go Inside the POD

```
kubectl exec --stdin --tty pod-ssl-website1 -- /bin/bash
```

Kubernetes will inject the Kubernetes Cluster IP and Port Number into the Environment Variables of every POD, so that Each POD can communicate with Kubernetes API Service:

Within the POD run the `printenv` command:

```
root@pod-ssl-website1:/# printenv

---
KUBERNETES_PORT_443_TCP_PROTO=tcp
KUBERNETES_PORT_443_TCP_ADDR=10.96.0.1
KUBERNETES_SERVICE_HOST=10.96.0.1
KUBERNETES_PORT=tcp://10.96.0.1:443
KUBERNETES_PORT_443_TCP_PORT=443
```

That is how a POD knows How to talk to Kubernetes API Server.

Exposing the Container to External World

- By default PODs and the containers inside are accessible only within POD Network.

While still inside the **POD1** we connected earlier try to access the Website hosted on **POD2**: `172.16.104.2`

```
apt update
apt install curl -y

curl http://172.16.104.2
```

POD-to-POD Communication successful!

But How to access this Website from External World?

- By creating a Service Object

Creating a Service Object that acts as a Broker between POD-network > Cluster-network > External-network

```
 kubectl expose pod pod-ssl-website1 --type=NodePort --port=443 --target-port=443
```

Error

```
linuxadmin@master:~$ kubectl expose pod pod-ssl-website1 --type=NodePort --port=443 --target-port=443
error: couldn't retrieve selectors via --selector flag or introspection: the pod has no labels and cannot be expo
```

Without Label Service Object can not be created.

Find Labels of both the PODS:

```
kubectl get pod pod-ssl-website1 -n default --show-labels

linuxadmin@master:~$ kubectl get pod pod-ssl-website1 -n default --show-labels
NAME               READY   STATUS    RESTARTS   AGE   LABELS
pod-ssl-website1   1/1     Running   0          29m   <none>

linuxadmin@master:~$ kubectl get pod pod-ssl-website2 -n default --show-labels
NAME               READY   STATUS    RESTARTS   AGE   LABELS
pod-ssl-website2   1/1     Running   0          29m   <none>
```

NOTE: We will discuss LABELS in next topic, but assigning a LABEL to a POD is mandatory.

Label both the PODS:

```
linuxadmin@master:~$ kubectl label pods pod-ssl-website2 -n default name=app2
pod/pod-ssl-website2 labeled

linuxadmin@master:~$ kubectl label pods pod-ssl-website1 -n default name=app1
pod/pod-ssl-website1 labeled
```

Show the labels:

```
kubectl get pods --show-labels

linuxadmin@master:~$ kubectl get pods --show-labels
NAME               READY   STATUS    RESTARTS   AGE   LABELS
pod-ssl-website1   1/1     Running   0          36m   name=app1
pod-ssl-website2   1/1     Running   0          36m   name=app2
```

Now create a Kubernetes Service that will expose the PODS to External World:

Exposing the First POD only to test.

```
kubectl expose pod pod-ssl-website1 --type=NodePort --name=svc-pod-ssl-website1

linuxadmin@master:~$ kubectl expose pod pod-ssl-website1 --type=NodePort --name=svc-pod-ssl-website1
service/svc-pod-ssl-website1 exposed
```

Find the newly created Service:

```
linuxadmin@master:~$ kubectl get services
NAME                  TYPE        CLUSTER-IP     EXTERNAL-IP   PORT(S)                       AGE
kubernetes            ClusterIP   10.96.0.1      <none>        443/TCP                       4h6m
svc-pod-ssl-website1  NodePort    10.105.83.26   <none>        80:30686/TCP,443:30727/TCP   7s
```

`svc-pod-ssl-website1   NodePort    10.105.83.26   <none>        80:30686/TCP,443:30727/TCP   7s`

That is not Enough!

You can see that External IP is <none>.

Patch the Service with the node1 physical IP = 192.168.1.51

`kubectl patch svc svc-pod-ssl-website1 -n default -p '{"spec": {"type": "NodePort", "externalIPs": ["192.168.1.51"]}}'`

```
linuxadmin@master:~$ kubectl patch svc svc-pod-ssl-website1 -n default -p '{"spec": {"type": "NodePort", "externa
service/svc-pod-ssl-website1 patched
```

Again find Service details:

```
kubectl get services

linuxadmin@master:~$ kubectl get services
NAME                  TYPE        CLUSTER-IP     EXTERNAL-IP    PORT(S)                      AGE
kubernetes            ClusterIP   10.96.0.1      <none>         443/TCP                      4h11m
svc-pod-ssl-website1  NodePort    10.105.83.26   192.168.1.51   80:30686/TCP,443:30727/TCP   5m7s

```

Now the External IP is reflected.

Access the Website now!