15. Networking

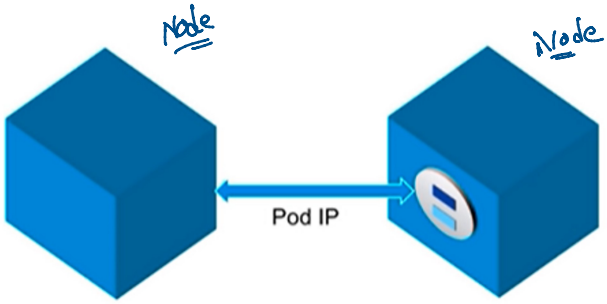
01 March 2022 01:29

Networking

Kubernetes Networking Basics

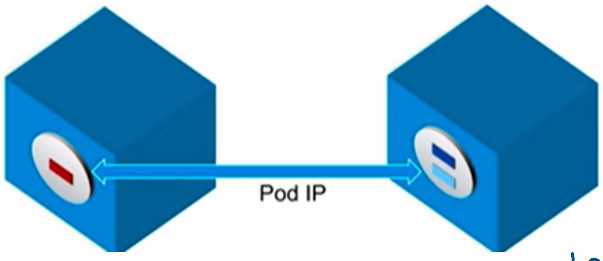
→ each pod one unique ip
→ container can use local host

Kubernetes Networking Basics



Node can reach pod using Pod IP

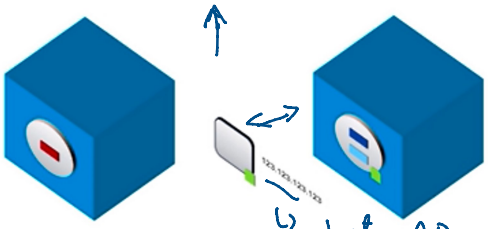
Kubernetes Networking Basics



- Pod can reach other pod using pod IP
- killed and restarted on diff IP
- replicas diff IP

service can be used based on labels.

- service can use env variable (or) DNS



↳ cluster IP only reachable with in the cluster.

Node port → is opened in every node of the cluster any req. to node port will go to the cluster IP.

Load balancer type → exposed to outside by using cloud provider load balancer.

Kubernetes Network Policy

Rules for controlling network access to Pods

Similar to security groups controlling access to virtual machines

Scoped to Namespace

↳ we can creat network policy independently to each namespace.

Network Policy Caveat

Kubernetes network plugin must support Network Policy

↑ with no

Kubernetes network plugin must support Network Policy

→ per cluster

→ otherwise though we successfully creat Network policy ↑ with no error.
there is no way to enforce the policy

Non-Isolated vs. Isolated Pods

→ once selected by network policy using labels its called isolated.

Non-isolated Isolated

```
ubuntu@ip-10-0-128-5:~$ kubectl get pods -n kube-system
NAME                                                              READY   STATUS    RESTARTS   AGE
calico-etcd-zb28f                                                 1/1     Running   2          11d
calico-kube-controllers-5dcc459c88-74h7v                          1/1     Running   3          11d
calico-node-cftgz                                                 2/2     Running   2          12m
calico-node-dbv2z                                                 2/2     Running   1          12m
calico-node-ms9qb                                                 2/2     Running   7          11d
coredns-86c58d9df4-cmc8n                                          1/1     Running   2          11d
coredns-86c58d9df4-n2vpg                                          1/1     Running   2          11d
etcd-ip-10-0-5-100.us-west-2.compute.internal                     1/1     Running   2          11d
kube-apiserver-ip-10-0-5-100.us-west-2.compute.internal           1/1     Running   2          11d
kube-controller-manager-ip-10-0-5-100.us-west-2.compute.internal  1/1     Running   2          11d
kube-proxy-8tqkj                                                  1/1     Running   0          12m
kube-proxy-cd7rc                                                  1/1     Running   2          11d
kube-proxy-d9sk2                                                  1/1     Running   0          12m
kube-scheduler-ip-10-0-5-100.us-west-2.compute.internal           1/1     Running   2          11d
kubernetes-dashboard-59756dbcc8-whkmt                             1/1     Running   2          11d
metrics-server-68d85f76bb-g4dhf                                   1/1     Running   0          10d
ubuntu@ip-10-0-128-5:~$
```

calico network plugin installed.

```
ubuntu@ip-10-0-128-5:~$ kubectl get pods -n network-policy -L app,region
NAME       READY   STATUS    RESTARTS   AGE   APP      REGION
client-1   1/1     Running   0          97s   client   us-east
client-2   1/1     Running   0          97s   client   us-west
server     1/1     Running   0          97s   server   us-central
ubuntu@ip-10-0-128-5:~$
```

two clients on diff regions sends requests to server

```
ubuntu@ip-10-0-128-5:~$ kubectl logs -n network-policy --tail=1 -f client-1
Received request
Received request
Received request
Received request
^C
ubuntu@ip-10-0-128-5:~$ kubectl logs -n network-policy --tail=1 -f client-2
Received request
Received request
```

server receiving the requests

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-us-east
  namespace: network-policy
spec:
  # Which Pods the policy applies to
  podSelector:
    matchLabels:
      app: server
  policyTypes:
  - Ingress
  - Egress
  ingress:
  - from:
    # Supports podSelector, namespaceSelector, and ipBlock
    - podSelector:
        matchLabels:
          region: us-east
    ports: # If not present allows all ports
    - protocol: TCP
      port: 8888
  egress:
  - to: # Allows all traffic when empty
```

→ network policies are available in the networking api's

→ allowing us-east

if podselector empty then the policy applies to all the pods.

→ incoming traffic
→ outgoing traffic. we can include one or both.

→ 3 supports

if from is ommited traffic from all sources is allowed

if port is ommitted traffic from all ports is allowed

allows to all outbound traffic.

```
ubuntu@ip-10-0-128-5:~$ kubectl logs -n network-policy --tail=0 -f client-1
Received request
Received request
Received request
^C
```

only us east requests are coming.

```
ubuntu@ip-10-0-128-5:~$ kubectl logs -n network-policy --tail=0 -f client-1
Received request
Received request
Received request
^C
ubuntu@ip-10-0-128-5:~$ kubectl logs -n network-policy --tail=0 -f client-2
^C
ubuntu@ip-10-0-128-5:~$
```

only we can see requests are coming.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: block-one-ip
  namespace: network-policy
spec:
  podSelector:
    matchLabels:
      app: server
  policyTypes:
  - Egress
  egress:
  - to:
    - ipBlock:
        cidr: 0.0.0.0/0
      except:
      - 192.168.134.72/32
```

→ sets range of allowed IP's
→ This means all ip address are allowed.
optional
Black list.

- This is ip of client 1 for demonstration purpose.
- In general we should use labels to refer to pods in your cluster, because ip add. changes, labels remains the same.

```
ubuntu@ip-10-0-128-5:~$ kubectl logs -n network-policy --tail=0 -f client-2
^C
ubuntu@ip-10-0-128-5:~$ kubectl logs -n network-policy --tail=0 -f client-1
Received request
Received request
Received request
Received request
^C
ubuntu@ip-10-0-128-5:~$ clear
```

How we are getting response though we blocked.

//** If one policy allows the traffic though other deny it. The traffic is allowed.

1st policy allowed egress to all
2nd denied to client1
} allowed to client 1

```
ubuntu@ip-10-0-128-5:~$ kubectl logs -n network-policy --tail=0 -f client-1
Received request
Received request
Received request
Received request
^C
```

If we delete the old network policy & keep only one, still we see traffic coming in.

Because the rules apply to new connection. As old connection still exists traffic is still flowing.

```
ubuntu@ip-10-0-128-5:~$ kubectl exec -n network-policy server -it ping 192.168.134.72
PING 192.168.134.72 (192.168.134.72): 56 data bytes
^C
--- 192.168.134.72 ping statistics ---
6 packets transmitted, 0 packets received, 100% packet loss
command terminated with exit code 1
ubuntu@ip-10-0-128-5:~$ kubectl exec -n network-policy server -it ping 192.168.233.198
PING 192.168.233.198 (192.168.233.198): 56 data bytes
64 bytes from 192.168.233.198: seq=0 ttl=63 time=0.076 ms
64 bytes from 192.168.233.198: seq=1 ttl=63 time=0.058 ms
64 bytes from 192.168.233.198: seq=2 ttl=63 time=0.057 ms
64 bytes from 192.168.233.198: seq=3 ttl=63 time=0.058 ms
^C
--- 192.168.233.198 ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max = 0.057/0.062/0.076 ms
ubuntu@ip-10-0-128-5:~$
```

pinging from server to client 1
we see there is no response because trying establish a new connection here.

CONCLUSION

- Networking review
- Types of Kubernetes services
- Network Policy