RX-M Cloud Native Consulting

Kubernetes

Lab 10 – Network Policy

A network policy is a specification of how groups of pods are allowed to communicate with each other and other network endpoints. NetworkPolicy resources use labels to select pods and define rules which specify what traffic is allowed to the selected pods. Network policies are implemented by a CNI network plugin, so you must use a CNI networking solution which supports NetworkPolicy.

By default, pods are non-isolated; they accept traffic from any source. Pods become isolated by having a NetworkPolicy that selects them. Adding a NetworkPolicy to a namespace selecting a particular pod, causes that pod to become "isolated", rejecting any connections that are not explicitly allowed by a NetworkPolicy. Other pods in the namespace that are not selected by any NetworkPolicy will continue to accept all traffic.

In this lab you will try controlling pod connectivity with Kubernetes Network Policy.

Step 0 - Clean up your Kubernetes Cluster

Delete any non-essential user defined deployments, services and other resources if the cluster is preexisting.

Step 1 - Run a service and access it from a client

To begin let's run a pod and try connecting to it from another pod. The rxmllc/hostinfo image listens on port 9898 and responds to requests with the pod hostname. Run a pod with a hostinfo pod having the label "app=hi":

```
user@ubuntu:~$ kubectl run hi --generator=run-pod/v1 --image rxmllc/hostinfo --port 80 -l app=hi

pod/hi created

user@ubuntu:~$
```

Now run a client busybox pod and try accessing the hostinfo pod from within the busybox pod:

```
user@ubuntu:~$ kubectl run client --generator=run-pod/v1 --image busybox --command -- tail -f
/dev/null

pod/client created

user@ubuntu:~$ kubectl get pod

NAME     READY   STATUS    RESTARTS   AGE
client   1/1     Running   0          12s
hi       1/1     Running   0          34s

user@ubuntu:~$ kubectl get pod hi -o json | jq -r '.status | .podIP, .hostIP'

10.32.0.5
192.168.228.157

user@ubuntu:~$ kubectl exec -it client sh

/ # wget -qO - http://10.32.0.5:9898

hi 10.32.0.5

/ # exit

user@ubuntu:~$
```

So with no network policy we can access one pod from another.

Step 2 - Create a blocking network policy

For our first network policy we'll create a policy that denies all inbound connections to pods in the default namespace. Create the following policy:

```
user@ubuntu:~$ mkdir ~/np && cd ~/np

user@ubuntu:~/np$ nano np.yaml

user@ubuntu:~/np$ cat np.yaml
```

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: default-deny
spec:
  podSelector: {}
  policyTypes:
  - Ingress
```

```
user@ubuntu:~/np$ kubectl apply -f np.yaml

networkpolicy.networking.k8s.io/default-deny created

user@ubuntu:~/np$ kubectl get networkpolicy

NAME           POD-SELECTOR   AGE
default-deny   <none>         4s

user@ubuntu:~/np$
```

This policy selects all pods (`{}`) and has no ingress policies (`Ingress`) for them. By creating any network policy however, we automatically isolate all pods.

Retry connecting to the hostinfo pod:

```
user@ubuntu:~/np$ kubectl exec -it client -- wget -qO - http://10.32.0.5:9898

wget: can't connect to remote host (10.32.0.5): Connection timed out
command terminated with exit code 1

user@ubuntu:~/np$
```

The command should hang and you will have to press CTRL+C to cancel it. The presence of a network policy shuts down our ability to reach the other pod.

Step 3 - Create a permissive network policy

To enable our busybox pod to access our hostinfo pod we will need to create a network policy that selects the hostinfo pods and allows ingress from the busybox pod. The hostinfo pod has the label "app=hi" so we can use that to select the target pod and our busybox pod has the label "run=client" so we'll use that to identify the ingress pods allowed (anytime you create a pod with the `kubectl run` command, all of the resources created get assigned the label `run= <NAME>`, where NAME is the name of the pod).

Create the new policy:

```
user@ubuntu:~/np$ nano np-hi.yaml

user@ubuntu:~/np$ cat np-hi.yaml
```

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: access-hostinfo
spec:
  podSelector:
    matchLabels:
      app: hi
  ingress:
  - from:
    - podSelector:
        matchLabels:
          run: "client"
```

```
user@ubuntu:~/np$ kubectl apply -f np-hi.yaml

networkpolicy.networking.k8s.io/access-hostinfo created

user@ubuntu:~/np$ kubectl get networkpolicy

NAME              POD-SELECTOR   AGE
access-hostinfo   app=hi         3s
default-deny      <none>         6m18s

user@ubuntu:~/np$
```

Now retry access the hostinfo pod from the busybox pod:

```
user@ubuntu:~/np$ kubectl exec -it client -- wget -qO - http://10.32.0.5:9898

hi 10.32.0.5

user@ubuntu:~/np$
```

It works!

For more information on networkpolicy take a look at the resource reference:

https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.15/#networkpolicy-v1-networking-k8s-io

Cleanup

When you are finished exploring, tear down all of your file-deployed components by using `kubectl delete -f` on the ~/np directory:

```
user@ubuntu:~/np$ kubectl delete -f ~/np/.

networkpolicy.networking.k8s.io "access-hostinfo" deleted
networkpolicy.networking.k8s.io "default-deny" deleted

user@ubuntu:~/np$
```

And delete any resources you created with `kubectl run` (if you didn't already):

```
user@ubuntu:~/np$ kubectl delete deploy hi client

deployment.extensions "hi" deleted
deployment.extensions "client" deleted

user@ubuntu:~/np$ cd

user@ubuntu:~$
```

Congratulations you have completed the lab!