Network Policy

Introduction:

Network policies are used to control the traffic flow at the IP address and port level. By default, all the traffic among pods is allowed and we can restrict it using network policies.

We can use the following identifier inside a Yaml file to control the traffic.

1. Pod: using the Pod's labels
2. Namespace: using the Namespace's labels
3. IP Blocks: using a specific IP Address

Objectives:

1. Blocking the access to the pods in a namespace using Network Policy
2. Default Deny all Ingress Traffic
3. Allow all Ingress traffic
4. Default Deny all Egress Traffic
5. Allow all Egress traffic
6. Default Deny all Ingress and All Egress traffic

1. Blocking the access to the pods in a namespace using Network Policy:

As we have already discussed that by default all Ingress and all Egress traffic is allowed. We can control the traffic as per our scenario. Let's take an example to block the Ingress traffic from one namespace to another.

Let's create two Namespaces named as **network1** and **network2.**

```
kubectl create ns network1
```

```
kubectl create ns network2
```

Using above command, we have created two namespaces. Let's create some pods inside these namespaces.

```
kubectl run network1-pod --image nginx -n network1
```

```
kubectl run network2-pod --image nginx -n network2
```

Using the above command, we have created **network1-pod** in **network1** namespace whereas **network2-pod** in **network2** namespace.

```
root@master:~#
root@master:~# kubectl run network1-pod --image nginx -n network1
pod/network1-pod created
root@master:~#
root@master:~# kubectl run network2-pod --image nginx -n network2
pod/network2-pod created
root@master:~#
root@master:~# kubectl get pods -o wide -n network1
NAME           READY   STATUS    RESTARTS   AGE   IP               NODE      NOMINATED NODE   READINESS GATES
network1-pod   1/1     Running   0          29s   192.168.189.100   worker2   <none>           <none>
root@master:~#
root@master:~#
root@master:~# kubectl get pods -o wide -n network2
NAME           READY   STATUS    RESTARTS   AGE   IP               NODE      NOMINATED NODE   READINESS GATES
network2-pod   1/1     Running   0          22s   192.168.235.158   worker1   <none>           <none>
root@master:~#
```

We can see in above output that both the pods have received the IP Address.

Let's try to access the pods from default namespace. First, we are going to create a pod inside default namespace which will be used to access the pod inside network1-pod or network2-pod.

```
kubectl run nginx --image nginx
```

Using above command, we have created a pod named nginx in default namespace. Now we will be trying to access the pod **network1-pod** from **nginx** pod in **default** namespace.

```
kubectl exec -it nginx -- /bin/bash
```

Using above command, we are inside the nginx pod and we will try to curl the **network1-pod** using its IP address which is **192.168.189.100**

In below output, we can see that we are able to access the pod as there is no network policy configured.

```
root@master:~#
root@master:~# kubectl exec -it nginx -- /bin/bash
root@nginx:/#
root@nginx:/# curl 192.168.189.100
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@nginx:/#
```

Now let's create a scenario that only pods with specific labels which are a part of **network2** namespace can access **network1-pod**.

Create a network policy using the below Yaml file. Firstly, we need to check the label of the pod where we are going to apply our network policy and label of the namespace and pods which will be allowed to access the pod.

kubectl describe pod network1-pod -n network1

```
root@master:~#
root@master:~# kubectl describe pod network1-pod -n network1
Name:         network1-pod
Namespace:    network1
Priority:     0
Node:         worker2/172.31.0.82
Start Time:   Mon, 16 Jan 2023 06:24:57 +0000
Labels:       run=network1-pod
Annotations:  cni.projectcalico.org/containerID: 71604f7e43a6a78f139a13f7791ca9b3163ace81ff0e7666be71da1c7d49b6a5
              cni.projectcalico.org/podIP: 192.168.189.100/32
              cni.projectcalico.org/podIPs: 192.168.189.100/32
Status:       Running
IP:           192.168.189.100
```

kubectl describe ns network2

```
root@master:~#
root@master:~# kubectl describe ns network2
Name:         network2
Labels:       kubernetes.io/metadata.name=network2
Annotations:  <none>
Status:       Active

No resource quota.

No LimitRange resource.
```

```
kubectl describe pod network2-pod -n network2
```

```
root@master:~#
root@master:~# kubectl describe pod network2-pod -n network2
Name:         network2-pod
Namespace:    network2
Priority:     0
Node:         worker1/172.31.0.70
Start Time:   Mon, 16 Jan 2023 06:25:09 +0000
Labels:       run=network2-pod
Annotations:  cni.projectcalico.org/containerID: a8a30dca8a2a375767e8df878551502b0884c6215f3e0444d603f37009af0c2f
              cni.projectcalico.org/podIP: 192.168.235.158/32
              cni.projectcalico.org/podIPs: 192.168.235.158/32
Status:       Running
IP:           192.168.235.158
```

Now we are aware about the labels used. Let’s create a network policy using these labels.

```
vi network-policy.yaml
```

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: network-policy
  namespace: network1
spec:
  podSelector:
    matchLabels:
          run: network1-pod
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
                  kubernetes.io/metadata.name: network2
          podSelector:
            matchLabels:
                  run: network2-pod
      ports:
        - protocol: TCP
          port: 80
```

From above file, we can clearly see that understand below points.

1. We are creating a network policy in **network1** namespace which has been mentioned in metadata field.
2. We are applying this policy on **network1-pod** in **network1** namespace using the label (**run: network1-pod**) which has been mentioned under specs podSelector field.
3. We are creating an Ingress policy which will impact only incoming traffic on network1-pod
4. Under Ingress rule, we are selecting a pod **network2-pod** with label **run=network2-pod** under namespace network2 having label **kubernetes.io/metadata.name= network2**.
5. Under ports field, we are mentioning that only port 80 is allowed.

Let's apply the definition file and test it.

```
kubectl apply -f network-policy.yaml
```

We can get the details of this network policy using the below command.

```
kubectl get networkpolicy -n network1
```

```
kubectl describe networkpolicy -n network1
```

From the below output, you can easily understand that the policy has been configured as per our scenario.

As we have created this policy for Ingress only so it won't be impacting the egress traffic.

```
root@master:~# kubectl apply -f network-policy.yaml
networkpolicy.networking.k8s.io/network-policy created
root@master:~#
root@master:~#
root@master:~# kubectl get networkpolicy -n network1
NAME             POD-SELECTOR       AGE
network-policy   run=network1-pod   61s
root@master:~#
root@master:~# kubectl describe networkpolicy -n network1
Name:         network-policy
Namespace:    network1
Created on:   2023-01-16 07:21:21 +0000 UTC
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     run=network1-pod
  Allowing ingress traffic:
    To Port: 80/TCP
    From:
      NamespaceSelector: kubernetes.io/metadata.name=network2
      PodSelector: run=network2-pod
  Not affecting egress traffic
  Policy Types: Ingress
root@master:~#
```

Let's try to test the impact of this rule using one of our nginx pod which is in default namespace.

```
root@master:~#
root@master:~# kubectl exec -it nginx -- /bin/bash
root@nginx:/#
root@nginx:/# curl 192.168.189.100
```

From above output, we can see that our pod is not able to access **network1-pod**.

Let's try the same from **network2-pod** which is a part of **network2** namespace.

```
kubectl exec -it network2-pod -n network2 -- /bin/bash
```

From below output, we can witness that our pod is reachable as per our Yaml file.

```
root@master:~# kubectl exec -it network2-pod -n network2 -- /bin/bash
root@network2-pod:/#
root@network2-pod:/# curl 192.168.189.100
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
root@network2-pod:/#
```

Now we can delete this network policy using below command.

```
kubectl delete networkpolicy network-policy -n network1
```

2. **Default Deny all Ingress Traffic:**

   We can use below Yaml file to deny all the Ingress traffic.

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

3. **Allow all Ingress traffic:**

   Use the below Yaml file to allow access to Ingress traffic.

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

4. **Default Deny all Egress Traffic:**

We can use below Yaml file to Deny all Egress traffic.

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

5. **Allow all Egress traffic:**

Use the below Yaml file to allow all Egress traffic.

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-all-egress
spec:
  podSelector: {}
  egress:
  - {}
  policyTypes:
  - Egress
```

6. **Default Deny all Ingress and All Egress traffic**

Use the below Yaml file to deny all Ingress and all Egress traffic.

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