Lab: Kubernetes Network Policy

Introduction:

In Kubernetes, **Network Policies** provide a way to control the communication between **Pods** and **services** within a cluster. They are used to define rules about how pods can communicate with each other and with resources outside the cluster, based on the pod's labels, IP addresses, and namespaces.

Network policies are an essential part of Kubernetes security because they enable you to restrict and secure pod communication, preventing unintended or unauthorized access between pods and services in your Kubernetes cluster.

Objective:

- **Create a Network Policy to Deny All Traffic to an Application**
- **Create a Network Policy to Allow All Traffic to an Application**
- **Create a Network Policy to Allow Traffic Only to a Specific Port of an Application**
- **Cleanup**

[Ensure that you have logged in as **root** user on **eoc-controller** node.]

1 Create A Network Policy Deny all Traffic to An Application

1.1 Let's **create** a Nginx **Pod** and **expose** it via port 80 for HTTP traffic.

```
# kubectl run webserver --image nginx --labels app=web
```

Output:

```
[root@eoc-controller ~]#kubectl run webserver --image nginx --labels app=web
pod/webserver created
```

```
# kubectl expose pod webserver --name=demo-service \
--labels app=web --port=80
```

Output:

```
[root@eoc-controller ~]#kubectl expose pod webserver --name=demo-service \
> --labels app=web --port=80
service/demo-service exposed
```

1.2 Let's **list** the **Pod** and **Service** by executing the below command.

```
# kubectl get pods,svc
```

Output:

```
[root@eoc-controller ~]#kubectl get pods,svc
NAME            READY   STATUS    RESTARTS   AGE
pod/webserver   1/1     Running   0          67s

NAME                   TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)   AGE
service/demo-service   ClusterIP   10.101.225.146   <none>        80/TCP    40s
service/kubernetes     ClusterIP   10.96.0.1        <none>        443/TCP   13d
```

1.3 Let's run a **temporary Pod** and make a request to **demo-service** by executing below command.

```
# kubectl run --rm -i -t --image=busybox:1.28.0 test-pod
```

Output:

```
[root@eoc-controller ~]#kubectl run --rm -i -t --image=busybox:1.28.0 test-pod -- sh
If you don't see a command prompt, try pressing enter.
/ #
```

```
# wget -qO- --timeout=2 http://demo-service
```

Output:

```
/ # wget -qO- --timeout=2 http://demo-service
<!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>
```

Note: The output above confirms access to the webserver application

```
# exit
```

Output:

```
/ # exit
Session ended, resume using 'kubectl attach test-pod -c test-pod -i -t' command when the pod is running
pod "test-pod" deleted
```

1.4 Let's **create** and **apply** a **NetworkPolicy Yaml** that denies any ingress traffic to Pods with the label **app=web**.

```
# cat > web-deny.yml << EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: web-deny-all
spec:
  podSelector:
     matchLabels:
       app: web
  ingress: []
EOF
```

```
# kubectl apply -f web-deny.yml
```

Output:

```
[root@eoc-controller ~]#kubectl apply -f web-deny.yml
networkpolicy.networking.k8s.io/web-deny-all created
```

1.5 Let's **list** the created **Network policy** by executing the below command.

```
# kubectl get netpol
```

Output:

```
[root@eoc-controller ~]#kubectl get netpol
NAME           POD-SELECTOR   AGE
web-deny-all   app=web        28s
```

1.6 Let's **describe** Network policy by executing the below command.

```
# kubectl describe networkpolicies.networking.k8s.io \
web-deny-all
```

Output:

```
[root@eoc-controller ~]#kubectl describe networkpolicies.networking.k8s.io \
> web-deny-all
Name:         web-deny-all
Namespace:    default
Created on:   2023-09-13 03:37:59 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     app=web
  Allowing ingress traffic:
    <none> (Selected pods are isolated for ingress connectivity)
  Not affecting egress traffic
  Policy Types: Ingress
```

1.7 Let's run a **temporary Pod** and make a request to **demo-service** by executing below command

```
# kubectl run  --rm -i -t --image=busybox:1.28.0 test-pod
\
-- sh
# wget -qO- --timeout=2 http://demo-service
```

Output:

```
[root@eoc-controller ~]#kubectl run  --rm -i -t --image=busybox:1.28.0 test-pod -- sh
If you don't see a command prompt, try pressing enter.
/ # wget -qO- --timeout=2 http://demo-service
wget: download timed out
```

1.8 Type **exit** to come out from the shell.

```
# exit
```

Output:

```
/ # exit
Session ended, resume using 'kubectl attach test-pod -c test-pod -i -t' command when the pod
is running
pod "test-pod" deleted
```

1.9 Let's delete **pod**, **service** and **NetworkPolicy** by executing below command.

```
# kubectl delete pod webserver

# kubectl delete service demo-service

# kubectl delete networkpolicy web-deny-all
```

Output:

```
[root@eoc-controller ~]#kubectl delete pod webserver
pod "webserver" deleted
[root@eoc-controller ~]#kubectl delete service demo-service
service "demo-service" deleted
[root@eoc-controller ~]#kubectl delete networkpolicy web-deny-all
networkpolicy.networking.k8s.io "web-deny-all" deleted
```

2.Create a Network Policy to Allow All Traffic to an Application

Use Case: After applying a deny-all policy that blocks all non-whitelisted traffic to the application, you now need to allow access to the application from all pods within the current namespace.

2.1 Let's **Create** a Pod and **Service** in a default namespace by executing the below comamnd.

```
# kubectl run webserver --image=nginx --labels=app=web \
--expose --port 80
```

Output:

```
[root@eoc-controller ~]#kubectl run webserver --image=nginx --labels=app=web \
> --expose --port 80
service/webserver created
pod/webserver created
```

2.2 Let's **create** and **apply NetworkPolicy** that allow all ingress traffic to the pods having **labels** app=web.

```
# cat > web-allow.yml << EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: web-allow-all
  namespace: default
spec:
  podSelector:
     matchLabels:
       app: web
  ingress:
  - {}
EOF
```

```
# kubectl apply -f web-allow.yml
```

Output:

```
[root@eoc-controller ~]#kubectl apply -f web-allow.yml
networkpolicy.networking.k8s.io/web-allow-all created
```

2.3 Let's **list** the created **Network Policy** by executing the below command.

```
# kubectl get netpol
```

Output:

```
[root@eoc-controller ~]#kubectl get netpol
NAME             POD-SELECTOR   AGE
web-allow-all    app=web        67s
```

2.4 Let's **describe** the **Network Policy** by executing the below command.

```
# kubectl describe networkpolicies.networking.k8s.io \

web-allow-all
```

Output:

```
[root@eoc-controller ~]#kubectl describe networkpolicies.networking.k8s.io   web-allow-all
Name:         web-allow-all
Namespace:    default
Created on:   2023-09-13 03:44:40 -0400 EDT
Labels:       <none>
Annotations:  <none>
Spec:
  PodSelector:     app=web
  Allowing ingress traffic:
    To Port: <any> (traffic allowed to all ports)
    From: <any> (traffic not restricted by source)
  Not affecting egress traffic
  Policy Types: Ingress
```

2.5 Let's run a **temporary** Pod and make a request to demo-service by executing below command

```
# kubectl run test-pod --rm -i -t --image=alpine -- sh

# wget -qO- --timeout=2 http://webserver
```

Output:

```
[root@eoc-controller ~]#kubectl run test-pod --rm -i -t --image=alpine -- sh
If you don't see a command prompt, try pressing enter.
/ # wget -qO- --timeout=2 http://webserver
<!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>
```

2.6 Type **exit** to come out from the shell.

```
# exit
```

Output:

```
/ # exit
Session ended, resume using 'kubectl attach test-pod -c test-pod -i -t' command when the pod is ru
nning
pod "test-pod" deleted
```

2.7 Let's delete **Pod**, **Service** and **NP** by executing the below command.

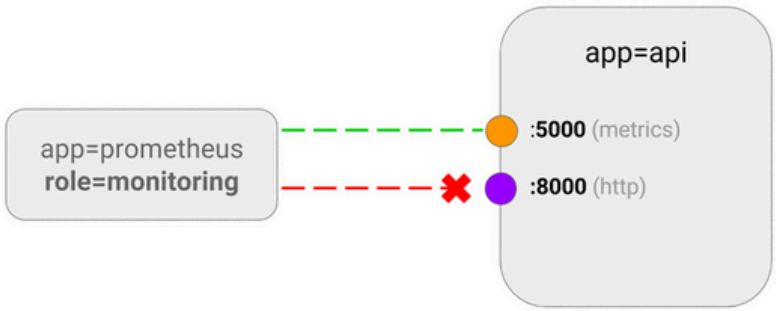
```
# kubectl delete pod webserver # kubectl
delete svc webserver # kubectl delete
networkpolicy web-allow-all
```

Output:

```
[root@eoc-controller ~]#kubectl delete pod webserver
pod "webserver" deleted
[root@eoc-controller ~]#kubectl delete svc webserver
service "webserver" deleted
[root@eoc-controller ~]#kubectl delete networkpolicy web-allow-all
networkpolicy.networking.k8s.io "web-allow-all" deleted
```

3. Create a Network Policy to Allow Traffic Only to a Specific Port of an Application

"This type of NetworkPolicy allows you to define ingress rules for specific ports of an application. If no port is specified in the ingress rule, the rule applies to all ports of the application by default."



Note: A port may be either a numerical or named port on a pod.

3.1 Let's **run** a web server by executing the below command

```
# kubectl run webserver-pod \
--image=ahmet/app-on-two-ports --labels=app=apiserver
```

Output:

```
[root@eoc-controller ~]#kubectl run webserver-pod \
> --image=ahmet/app-on-two-ports --labels=app=apiserver
pod/webserver-pod created
```

3.2 Let's **create** a Service and **expose** it using the ClusterIP service type.

```
# kubectl create service clusterip apiserver \
--tcp 8001:8000 --tcp 5001:5000
```

Output:

```
[root@eoc-controller ~]#kubectl create service clusterip apiserver \
> --tcp 8001:8000 --tcp 5001:5000
service/apiserver created
```

Note: "Network Policies do not have knowledge of the specific port numbers exposed by the application, such as 8001 and 5001. This is because Network Policies control inter-pod traffic, while when you expose a Pod as a Service, ports are remapped to the Service's defined ports."

"Therefore, you need to use the container port numbers (e.g., 8000 and 5000) in the NetworkPolicy specification. A less error-prone alternative is to refer to the port names (e.g., metrics and http)."

3.3 Let's **create** the **NetworkPolicy as** api-allow-5000.yaml and **apply it.**

```
# cat > api-allow-5000.yml << EOF
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: api-allow-5000
spec:
  podSelector:
    matchLabels:
      app: apiserver
  ingress:
  - ports:
    - port: 5000
    from:
    - podSelector:
        matchLabels:
          role: monitoring
EOF
```

```
# kubectl apply -f api-allow-5000.yml
```

Output:

```
[root@eoc-controller ~]#kubectl apply -f api-allow-5000.yml
networkpolicy.networking.k8s.io/api-allow-5000 created
```

3.4 Let's **run** a temporary Pod with no custom labels and observe that traffic to ports 5000 and 8000 is **blocked**.

```
# kubectl run test-pod --rm -i -t --image=alpine -- sh

# wget -qO- --timeout=2 http://apiserver:8001

# wget -qO- --timeout=2 http://apiserver:5001/metrics
```

Output:

```
[root@eoc-controller ~]#kubectl run test-pod --rm -i -t --image=alpine -- sh
If you don't see a command prompt, try pressing enter.
/ # wget -qO- --timeout=2 http://apiserver:8001
wget: download timed out
/ # wget -qO- --timeout=2 http://apiserver:5001/metrics
wget: download timed out
```

3.5 Type **exit** to exit the shell.

```
# exit
```

Output:

```
/ # exit
Session ended, resume using 'kubectl attach test-pod -c test-pod -i -t' command when the pod is runnin
g
pod "test-pod" deleted
```

3.6 Let's run a temporary pod with the **role=monitoring** label, and observe that traffic to port 5000 is allowed, but port 8000 is still inaccessible.

```
# kubectl run test-pod --labels=role=monitoring --rm -i \
-t --image=alpine -- sh

# wget -qO- --timeout=2 http://apiserver:8001

# wget -qO- --timeout=2 http://apiserver:5001/metrics
```

Output:

```
[root@eoc-kubemaster ~]#kubectl run test-pod --labels=role=monitoring --rm -i \
> -t --image=alpine -- sh
If you don't see a command prompt, try pressing enter.
/ # wget -qO- --timeout=2 http://apiserver:8001
wget: download timed out
/ # wget -qO- --timeout=2 http://apiserver:5001/metrics
http.requests=1
go.goroutines=5
go.cpus=4
```

3.7 Type **exit** to exit the shell.

```
# exit
```

Output:

```
/ # exit
Session ended, resume using 'kubectl attach test-pod -c test-pod -i -t' command when the pod is runnin
g
pod "test-pod" deleted
```

4. Cleanup

4.1 Let's **delete** the **Pod** by executing the below command.

```
# kubectl delete pod webserver-pod
```

Output:

```
[root@eoc-controller ~]#kubectl delete pod webserver-pod
pod "webserver-pod" deleted
```

4.2 Let's **delete** the **Service** apiserver by executing the below command

```
# kubectl delete service apiserver
```

Output:

```
[root@eoc-controller ~]#kubectl delete service apiserver
service "apiserver" deleted
```