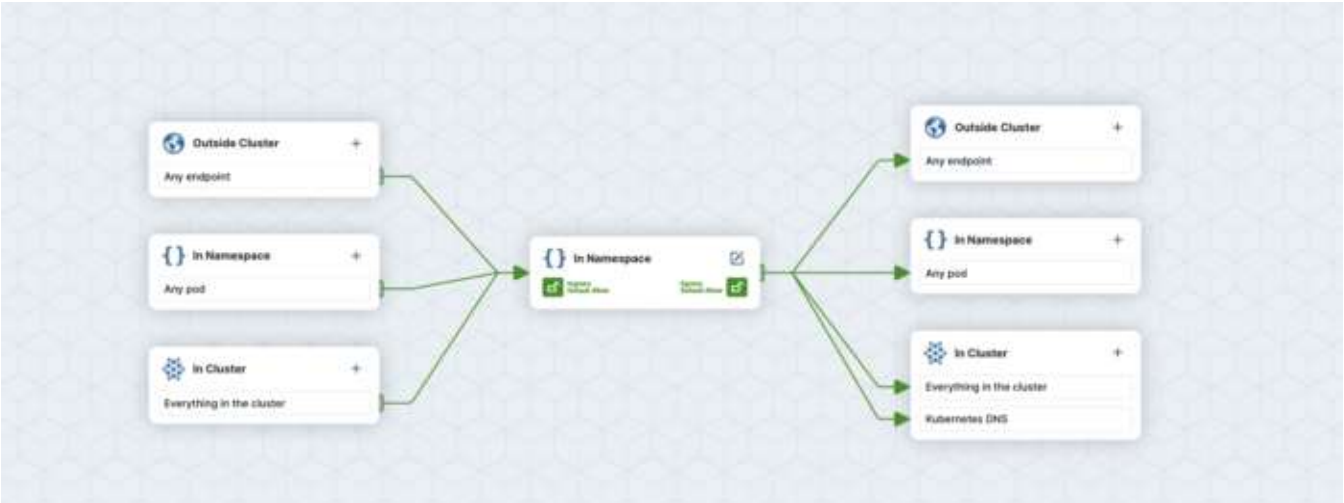
Network Policies demystified in Kubernetes

Pavan Kumar
Apr 10 · 6 min read

Visualize network policies in Kubernetes using Cilium Editor

You might have a couple of microservices running on Kubernetes. Considering a simple architecture you might have a web server (The frontend) and a database server (The backend) and a couple of other microservices too (Ex: A messaging queue like Kafka, RabbitMQ, etc). You might want that the database server should only be accessible by Kafka, RabbitMQ Pods. The messaging broker pods (i.e. Kafka) should receive traffic only from the web-server pods. How is all of this achieved? How can we control the Ingress and Egress traffic to/from the Kubernetes resources? **Network Policies** in Kubernetes to our rescue here. **Network Policies** are the Kubernetes resources that control the traffic between Kubernetes resources like Pods, network endpoints. It defines the network access policies for communication between Pods, network endpoints. Network policies allow us to specify the entities to which a Pod is authorized to connect. The aforementioned problem with the limited communication between pods can be solved by enforcing a networking policy on the database pod, that would allow traffic from the pods that have the label app=messaging-server, and the same with the Kafka/RabbitMQ pods to allow traffic from the pods that have the label app=web-server.



Network Policy visualizing using cilium editor

What is the entire story all about? (TLDR)

1. Understanding various networking policies in Kubernetes.
2. Visualize these networking policies using the *cilium editor.*

Prerequisites

1. A Kubernetes cluster (Can be either On-Prem, AKS, EKS, GKE, Kind).

Story Resources

1. GitHub Link: https://github.com/pavan-kumar-99/medium-manifests
2. GitHub Branch: network-policies

Creating network policies

> *Network policies are implemented by the network plugin. To use network policies, you must be using a networking solution that supports NetworkPolicy.*

By default Pods are non-isolated. That means that

1. Each Pod can communicate to every other pod in the same namespace.
2. All the pods from one namespace can communicate to every other pod in a different namespace.
3. No Ingress or Egress policies are applied to the Pods. By default, the Ingress and Egress traffic is allowed to and from the Pod.

Let us get started by creating 3 deployments. In the scope of this demo, all three deployments will use the same image (i.e. Nginx) as the primary intention of this article is to understand the concept of network policies but not to develop an application from scratch. Let us create the web-server components.

```
$ kubectl create ns netpol

$ kubectl create deploy web-server --image=nginx
```

```
$ kubectl expose deploy web-server --port=80
```

Lets us now create the Kafka components

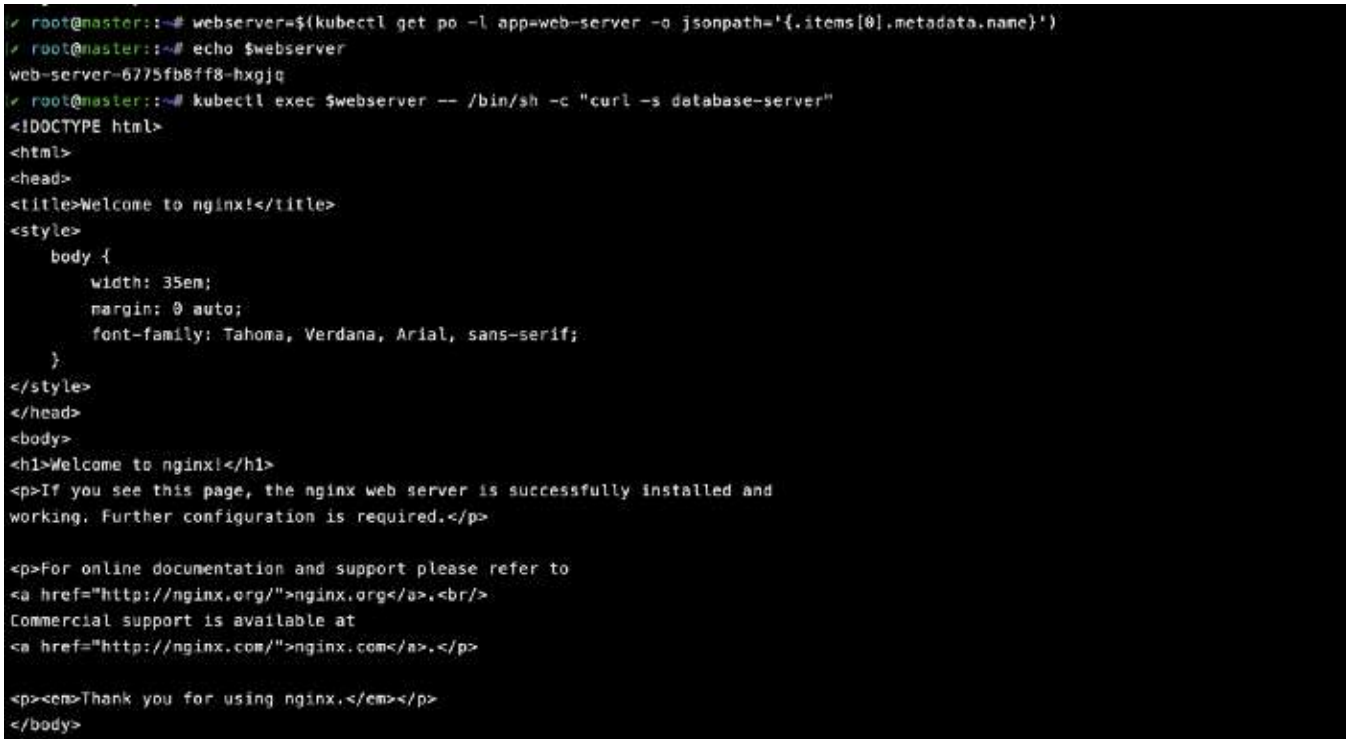
```
$ kubectl create deploy kafka-server --image=nginx

$ kubectl expose deploy kafka-server --port=80
```

Let us now create the database components

```
$ kubectl create deploy database-server --image=nginx

$ kubectl expose deploy database-server --port=80
```

We will now try to reach the database-server from the web-server.

```
$ export role="web-server"

$ server_name=$(kubectl get po -l app=$role -o
jsonpath='{.items[0].metadata.name}')

$ kubectl exec $server_name -- /bin/sh -c "curl -s database-server"
```



Communication between web-server and database-server

You should notice that the web-server can communicate with the database-server. This is the same with the other pods as well. You can check that by changing the role to kafka-server or any other server in the above command.

So, no we don't want the database-server to be accessible by every pod in the namespace. Instead only the kafka-server should be accessing it. i.e. only the pods with the label app=kafka-server should be able to communicate to the database-server (i.e. with the label app=database-server).

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: access-db-from-kafka
  namespace: netpol
spec:
  podSelector:
    matchLabels:
      app: database-server
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: kafka-server
      ports:
        - port: 80
        - port: 443
```

access-db-from-kafka-netpol.yaml hosted with ♡ by GitHub view raw

Let us now take a look at the manifests closely.

> *spec.podSelector: The labels of the pod to which this network policy is to be applied.*
>
> *ingress.from.podSelector: The labels of the pods from which the ingress traffic is allowed.*
>
> *ingress.ports: The ports to which the ingress communication is allowed.*

Let us now apply the NetworkPolicy to see the change.

```
$ git clone https://github.com/pavan-kumar-99/medium-manifests.git \
-b network-policies

$ cd medium-manifests/

$ kubectl apply -f access-db-from-kafka-netpol.yaml
```

Hurrah, our first policy is now applied. Let us try to access the database servers from 2 different pods i.e.

1. From the web-server pod (This should fail).
2. From the Kafka Pod (This should Pass).

```
$ export role="web-server"

$ server_name=$(kubectl get po -l app=$role -o
jsonpath='{.items[0].metadata.name}')

$ kubectl exec $server_name -- /bin/sh -c "curl -s database-server"

command terminated with exit code 7
```

The request will get timed out now. Let us now try the same from the Kafka server.

```
$ export role="kafka-server"

$ server_name=$(kubectl get po -l app=$role -o
jsonpath='{.items[0].metadata.name}')

$ kubectl exec $server_name -- /bin/sh -c "curl -s database-server"
```

```
<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@master:~/netpol#
```

The curl command is now successful

You should now be able to connect to the database-server pod from the kafka-server pod.

We should also restrict the traffic to the kafka-server pod to allow the ingress traffic only from the web-server pod. But this time let's make it a bit more interactive by visualizing the networking policies.

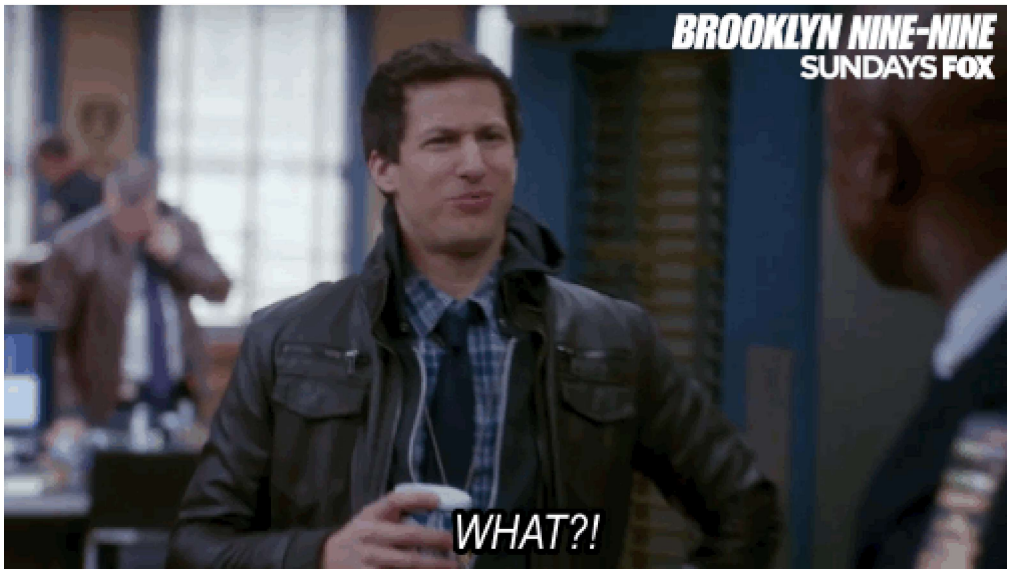


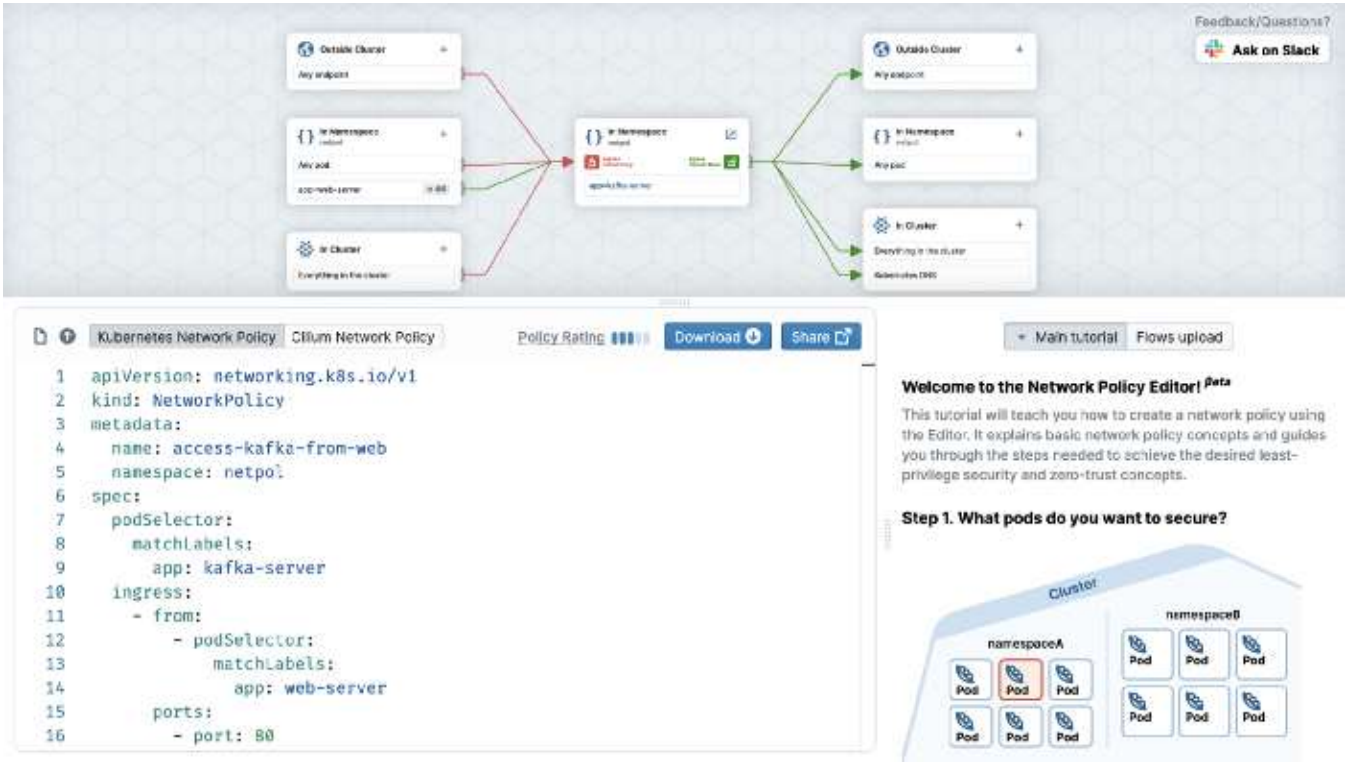
Image Credits: Google

Yes, you've read it right. **editor.cilium.io** makes it easy to build and visualize Network Policies, which can then be downloaded as YAML and run in any Kubernetes cluster. The YAML's can also be uploaded to visualize them with a proper GUI.

Cilium Editor

The box in the center represents the resource to which you are trying to apply the network policy to. The resource to your left represents the ingress policies and the resources to your right represent the egress policies. We should now restrict the traffic to the kafka-server pod to allow the ingress traffic only from the web-server pod.

Once you create the policy you should find the policy yaml created on the screen.



Network policy from cilium editor

We will now download and apply the policy to our Kubernetes cluster. I have already downloaded this to my GitHub repo.

```
$ git clone https://github.com/pavan-kumar-99/medium-manifests.git \
-b network-policies

$ cd medium-manifests/

$ kubectl apply -f access-kafka-from-web.yaml
```

```yaml
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: access-kafka-from-web
  namespace: netpol
spec:
```

```
  podSelector:
    matchLabels:
      app: kafka-server
  ingress:
    - from:
        - podSelector:
            matchLabels:
              app: web-server
      ports:
        - port: 80
```

access-kafka-from-web.yaml hosted with ♡ by GitHub view raw

Hurrah, our second policy is also applied. Let us try to access the kafka servers from 2 different pods i.e.

1. From the web-server pod (This should Pass).
2. From the Database Pod (This should fail).

```
$ export role="database-server"

$ server_name=$(kubectl get po -l app=$role -o
jsonpath='{.items[0].metadata.name}')

$ kubectl exec $server_name -- /bin/sh -c "curl -s kafka-server"

command terminated with exit code 7
```

We will now try to access the kafka server from the web-server

```
$ export role="web-server"

$ server_name=$(kubectl get po -l app=$role -o
jsonpath='{.items[0].metadata.name}')

$ kubectl exec $server_name -- /bin/sh -c "curl -s kafka-server"
```

```
root@master:~/netpol# export role="web-server"
root@master:~/netpol# server_name=$(kubectl get po -l app=$role -o jsonpath='{.items[0].metadata.name}')
root@master:~/netpol# kubectl exec $server_name -- /bin/sh -c "curl -s kafka-server"
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
    body {
        width: 35em;
        margin: 0 auto;
```

Network policy works as expected

Conclusion

With the help of tools like https://editor.cilium.io/ understanding and visualizing, network policies are now made easier. Many more complicated network policies include selectors like CIDR block, namespaces, etc. I will try to cover them in my next article with an actual golang application.

Until next time…..

Recommended

AutoScaling in Kubernetes (HPA / VPA)

Autoscale your applications in Kubernetes using Vertical Pod Autoscaler (VPA) and Horizontal Pod Autoscaler (HPA)

medium.com

Reference

Network Policies

If you want to control traffic flow at the IP address or port level (OSI layer 3 or 4), then you might consider using...

kubernetes.io