ISOVALENT now part of CISCO

# Introduction to Cilium Network Policies (Part 1)

Paul Arah | Published: Mar 10, 2024 | Updated: Apr 02, 2024 | Cilium

Table of contents

Imagine a busy city with no traffic rules. Cars, bicycles, and pedestrians would be free to roam wherever possible. This scenario is akin to what a Kubernetes cluster looks like without network policies: uncontrolled and unpredictable network communication! Network Policies provides a solution for enforcing network segmentation controls within Kubernetes clusters. Operators can define rules that control how pods communicate with each other and other internal and external resources, improving the security posture of Kubernetes clusters.

This blog post is part of a three-part series where we dive into Cilium Network Policies and the Network Policy Editor. In this first part of the series, we'll introduce Cilium Network Policies and compare them to the standard Kubernetes network policies; in the second part, we'll explore Cilium Network Policies through the lens of some common user stories, and in the third part of this series we will explore how we can use the Network Policy Editor to ease the cognitive overhead of writing network policies to secure your Kubernetes clusters.

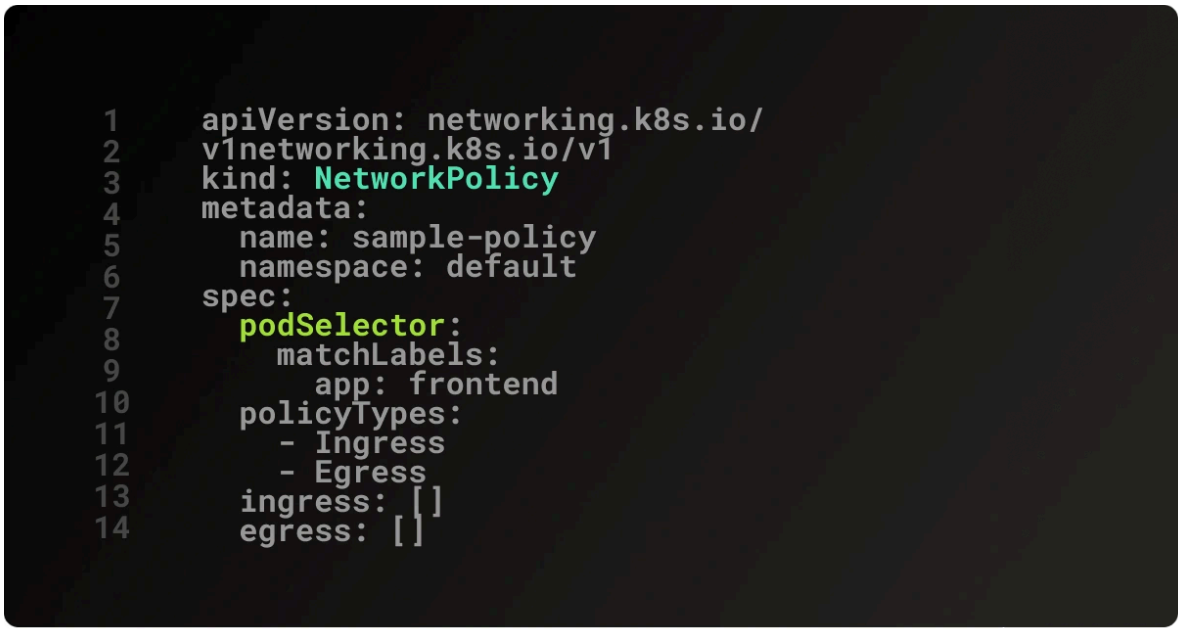
Before diving into Cilium Network Policies, it would be helpful to take a step backward and understand the standard Kubernetes network policy and how Cilium enhances it.

## Introduction to Kubernetes Network Policy

Kubernetes, by default, employs a flat networking topology, allowing pod-to-pod communication without any restrictions. The Kubernetes network model specifies that each pod gets its unique IP address, and every pod can communicate with every other pod on every node without NAT. This approach logically translates to simplicity for operators since each pod (group of containers) can be treated much like a VM. However, attackers can exploit this default absence of network segmentation controls to execute lateral movement attacks, among other possible vulnerabilities.

Kubernetes network policies provide an application-centric construct for defining network segmentation rules at various levels in the cluster. We can define network security policies at layers 3 and 4 of the OSI model, i.e., IP, TCP, UDP, and SCTP. Although the network policy spec is part of Kubernetes, the implementation is done by a Container Network Interface (CNI), so using a CNI that supports network policies is imperative. At a high level, the Kubernetes Network Policy API allows us to filter traffic based on the pod labels, namespace labels, and IP/CIDR range.

The Kubernetes network policy resource specifies four important fields: The *podSelector* specifies which pod(s) a policy should be applied on, the *policyTypes* field indicates whether a policy defines rules for ingress traffic, egress traffic, or a combination of both, and the *ingress* and *egress* fields define the ingress and egress traffic rules respectively.



```
apiVersion: networking.k8s.io/
v1networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: sample-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      app: frontend
  policyTypes:
    - Ingress
    - Egress
  ingress: []
  egress: []
```

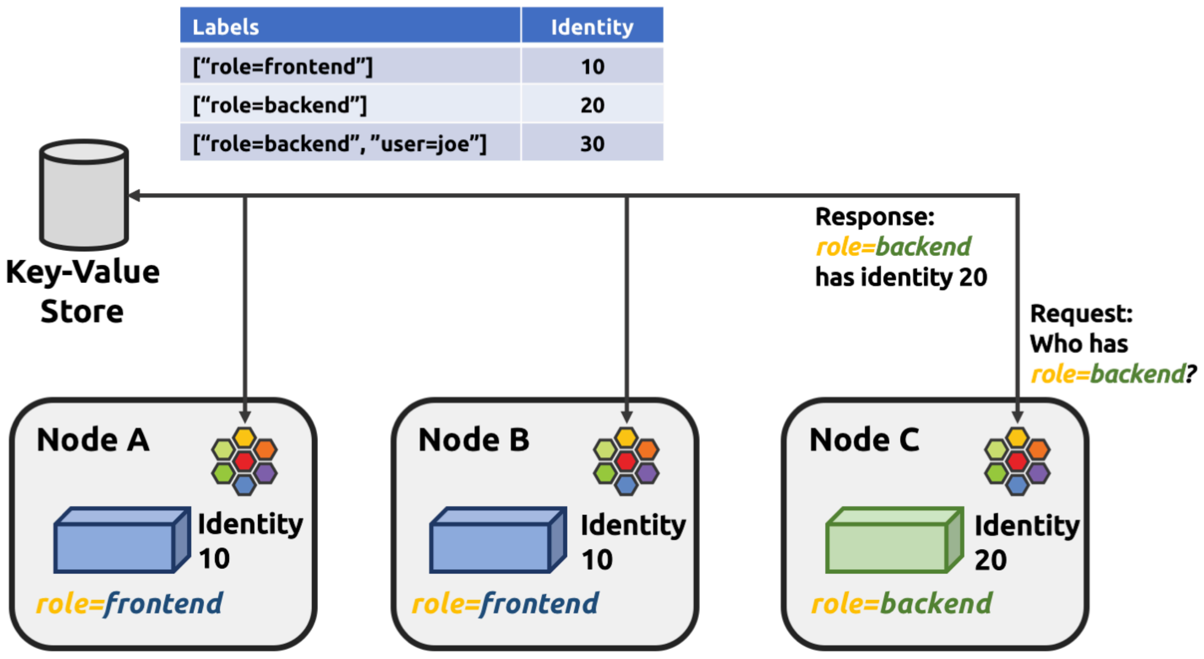
While Kubernetes network policy provides network isolation, it is still limited in granularity, flexibility, and features. Cilium extends the Kubernetes network policy model, providing a more advanced feature set that caters to operators' everyday and niche use cases. Let's dive into Cilium!

## Decoupling Identity from IP addresses: Cilium Endpoint and Identity

To fully grasp Cilium Network Policies, it is essential to start with two basic building blocks: The Cilium Endpoint and the Cilium Identity.

Traditionally, security enforcement has relied on IP address filters. However, with the emergence of highly dynamic cloud native container orchestration environments such as Kubernetes, relying on IP addresses for security enforcement constrains scalability and flexibility in these environments. In Kubernetes, for instance, every pod is assigned a unique IP address and when security enforcement is based on IP address, then IP address filters (iptables rules defined by kube-proxy) are placed on each node in the cluster. Every time a pod is started or stopped, the IP address filters on every node in the cluster has to be updated. In a large cluster, depending on the churn rate of the deployed pods, updating these IP address based security rules could imply updating thousands of cluster nodes multiple times per second! This also implies that starting some new pods has to be delayed until the relevant updates have been made to the security rules on the cluster's nodes. This approach is inflexible, fragile, and does not scale well.

Cilium completely decouples security from network addressing using workload identity derived from a combination of user and system-defined labels and metadata. When a pod is created, Cilium creates an endpoint representing the pod on the network. The endpoint is assigned an internal IPv4 and IPv6 address. The endpoint's identity is derived from the pod labels; if the labels change, Cilium updates the endpoint's identity accordingly. The identity is used to enforce basic connectivity between endpoints. Pods now only have to resolve their identities from a key-value store, which is much simpler than updating security rules for each node in the cluster.



## What's Different With Cilium Network Policies?

To begin, Cilium implements the standard Kubernetes network policy spec. Your Kubernetes network policies work out of the box with Cilium without any additional changes. However, Cilium also takes it further and extends the standard Kubernetes network policy spec via Cilium network policies, providing more flexibility, granularity, and advanced features. With Cilium network policies, we can implement fine-grained rules at Layers 3, 4, and 7 of the OSI model, catering to broader and more relevant use cases than the standard Kubernetes network policy.

Cilium introduces two new Kubernetes custom resource definitions(CRDs): CiliumNetworkPolicy and CiliumClusterwideNetworkPolicy. With these CRDs, we can define security policies with Cilium as Kubernetes objects, and Kubernetes automatically distributes these policies to all the nodes in the cluster. The *CiliumNetworkPolicy* CRD is namespaced scoped, while the *CiliumClusterwideNetworkPolicy* CRD works similarly to the *CiliumNetworkPolicy* but is cluster-scoped.

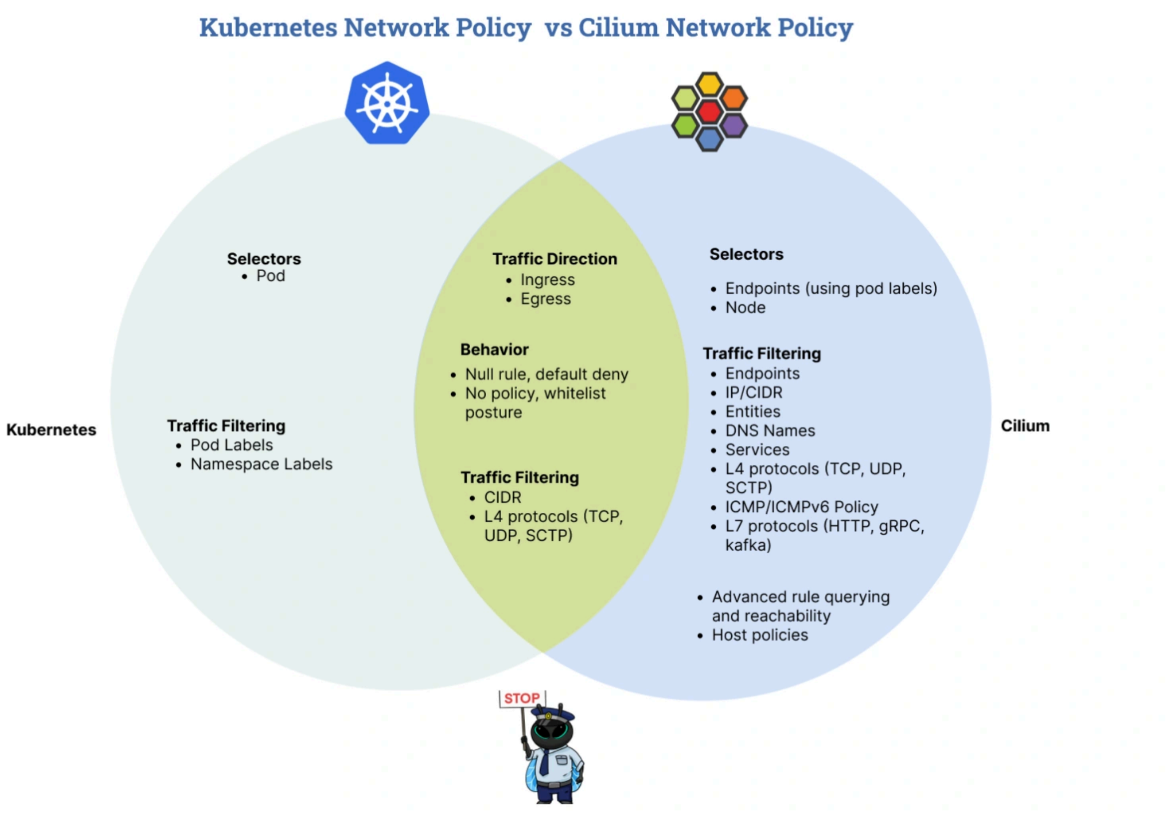
At Layer 3, Cilium's endpoint-based policies can define connectivity rules based on pod labels. Service-based policies use orchestration systems services like Kubernetes service endpoints to define connectivity rules.

Entity-based policies allow categorizing remote peers without knowing their IP addresses. IP/CIDR-based policies define connectivity rules for external services using hardcoded IP addresses or subnets. DNS-based policies can define connectivity rules based on DNS names resolved to IP addresses.

At Layer 4, Cilium network policies can define rules that allow an endpoint to emit and receive packets on a specific port using a specific protocol. Finally, at Layer 7, Cilium network policies allow defining rules for API-level security for common Layer 7 protocols such as HTTP, Kafka, gRPC, etc.

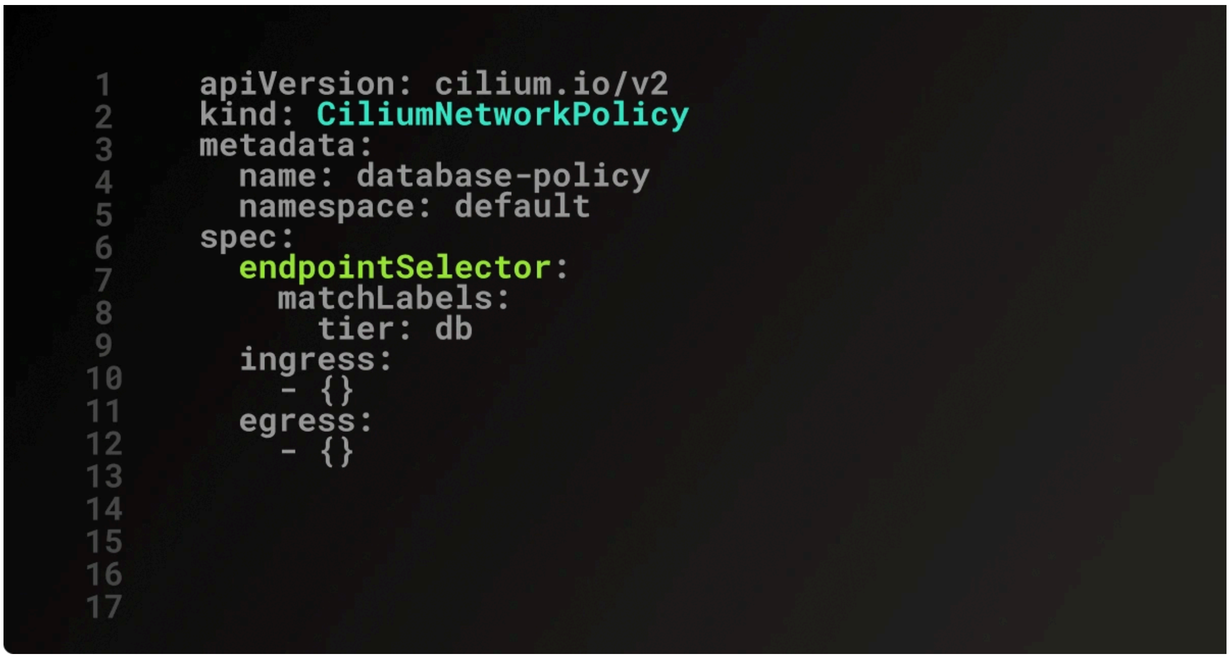
We can apply Cilium network policies to a single pod, a group of pods with matching labels, an entire namespace, or the entire cluster. This provides the ability to apply the same consistent network security model across various levels of the entire infrastructure.

Cilium also supports host policies with the CiliumClusterWideNetworkPolicy CRD. This feature essentially is a host-level firewall for the selected nodes in the cluster, extending the same consistent Kubernetes network security policy model to the hosts. An example of this would be blocking SSH access to node(s) and ICMP pings to the node(s). Combining Cilium network policy and Cilium cluster-wide network policy can simplify managing security policies. Baseline security policies that apply to all the pods can be specified as Cilium cluster-wide policies, and pod(s) specific policies can be specified as Cilium network policies. An example baseline policy would be a default deny rule for all pods and allowing egress traffic to kube-dns for all pods in the cluster.


Kubernetes Network Policy vs Cilium Network Policy
Selectors
• Pod
Kubernetes
Traffic Filtering
• Pod Labels
• Namespace Labels
Traffic Direction
• Ingress
• Egress
Behavior
• Null rule, default deny
• No policy, whitelist posture
Traffic Filtering
• CIDR
• L4 protocols (TCP, UDP, SCTP)
Selectors
• Endpoints (using pod labels)
• Node
Traffic Filtering
• Endpoints
• IP/CIDR
• Entities
• DNS Names
• Services
• L4 protocols (TCP, UDP, SCTP)
• ICMP/ICMPv6 Policy
• L7 protocols (HTTP, gRPC, kafka)
• Advanced rule querying and reachability
• Host policies
Cilium
STOP

## Anatomy of the CiliumNetworkPolicy CRD

The CiliumNetworkPolicy custom resource definition (CRD) can be broken down into three parts: the endpoint selector, ingress, and egress policies. The endpoints selector field selects endpoints based on the pod labels and specifies which pod(s) a policy should be applied to. With the CiliumClusterWideNetworkPolicy resource, we replace the endpoint selector field with the node selector field, and the node labels are used to specify which node a policy applies to. The ingress and egress section allows us to define ingress and egress traffic rulesets respectively.



```
apiVersion: cilium.io/v2
kind: CiliumNetworkPolicy
metadata:
  name: database-policy
  namespace: default
spec:
  endpointSelector:
    matchLabels:
      tier: db
  ingress:
    - {}
  egress:
    - {}
```

## Summary

By implementing network policies, you can transform your Kubernetes cluster from a chaotic city with no rules into a well-organized and secure environment. Cilium network policy provides more granularity, flexibility, and advanced features than the standard Kubernetes network policy. Cilium supports defining granular rulesets at Layers 3, 4, and 7 of the OSI model. At layer 3 of the OSI model, policies can be endpoint-based, entity-based, DNS-based, IP/CIDR-based, or service-based. At layer 7 of the OSI model, Cilium supports defining rulesets for common protocols such as HTTP, gRPC, and Kafka. Cilium implements two CRDs for defining security policies: CiliumNetworkPolicy and CiliumClusterWideNetworkPolicy. These CRDs can be used individually or together to achieve the ideal security requirements for your Kubernetes clusters. Similar to how clear traffic regulations ensure smooth, safe, and predictable movement in a city, network policies allow you to establish order and control within your containerized applications. So, take the wheel and implement Cilium network policies to create a secure and predictable network landscape for your Kubernetes cluster!

In the next blog post in this series, we will explore Cilium network policy through the lens of some common user stories.