Herding Cats: Calico Policies for Kubernetes

31 July 2019

TIGERA

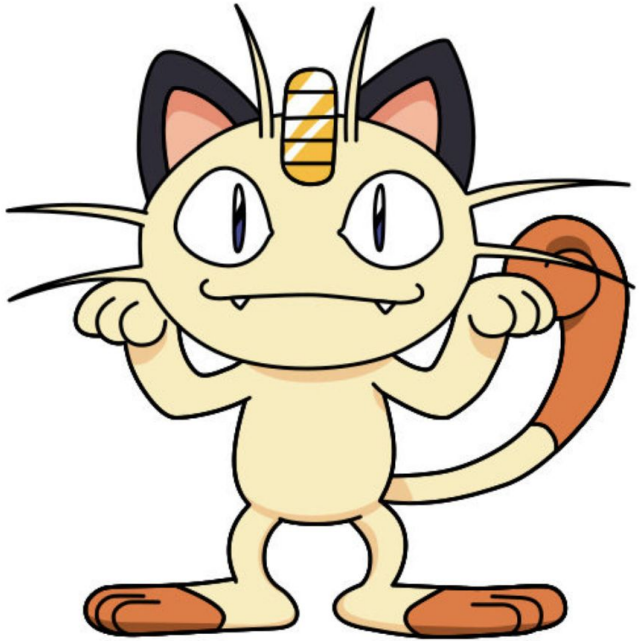
Drew Oetzel - Senior Technical Solutions Engineer

- Working with enterprise software since the late 90s
- 7 years at Splunk, honing his security skills
- 2.5 years at Mesosphere, then Heptio mastering the art of distributed systems, containers, and all that goes along with them
- Outside of tech ask him about history, gardening, or what he’s doing to try to curb his Reddit addiction!

TIGERA

Who is Tigera?

A super secret Pokemon?



No, but we do have a conference room named after Meowth!

TIGERA

Who Is Tigera?

Tigera Secure

- Zero Trust Network Security
- Visibility, Traceability and Remediation for Dynamic Applications
- Continuous Compliance & Enterprise Controls
- Security that Spans Multi-Cloud and Legacy Environments

Tigera Calico

- Open-source
- Scalable, distributed control plane
- Policy-driven network security
- No overlay required
- Widely deployed, and proven at scale

Bloomberg GitHub salesforce NTT Communications IBM CommonwealthBank orange virtustream EMC² YAHOO! box BBVA

SQUARESPACE Giant Swarm ebay idealo USA TODAY YAHOO! JAPAN at&t CenturyLink Ford JPMorganChase

TIGERA

Tigera Strategic Partnerships

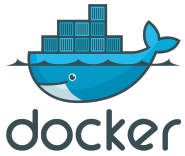
Integrated with ACS Engine, Collaborating on Azure Kubernetes Service (AKS) and Windows

Default network policy for Azure Container Service for Kubernetes (EKS) & Heptio/AWS Quick Start

Partnered to support network policy in Google Container Engine (GKE)

Partner to implement connectivity and security for IBM Kubernetes Service, IBM Cloud Private

OpenShift primed partner. Certified integration with OpenShift Container Platform



Default networking and policy for Docker EE Kubernetes

TIGERA

What is Calico

“Networking” for containers.

Open source project with worldwide contributors backed by the commercial enterprise Tigera, Inc.

Most used CNI plugin for Kubernetes worldwide.

www.projectcalico.org

TIGERA

Ok, But What Can Calico Do For Me?

Efficiently hand out IPs to your pods. (IPAM)

- Can use different subnets per namespace or rack / VPC
- Can assign static IP addresses to pods

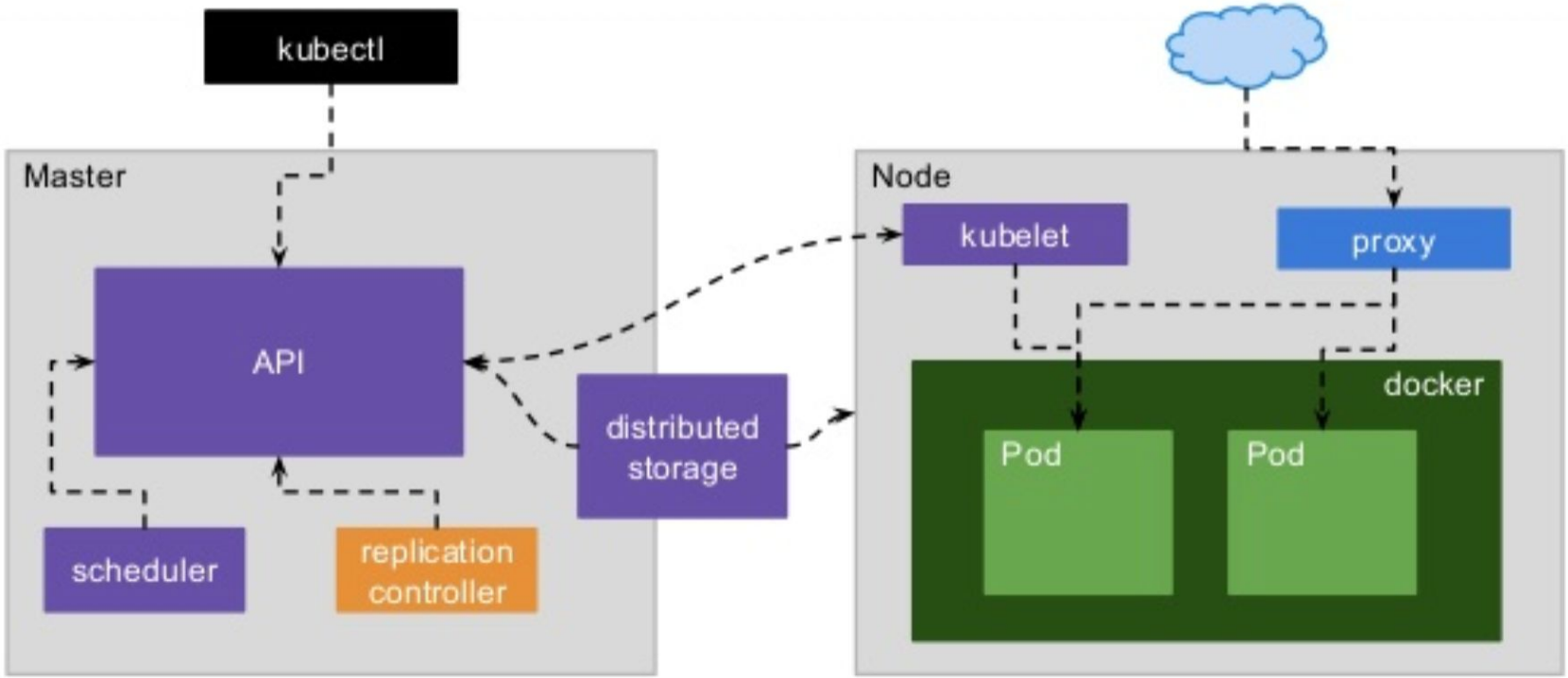
Secure your inner K8s networking with policies.

- Secure your East/West traffic
- Prevent malicious behavior from spreading
- Block insider threat / insider error

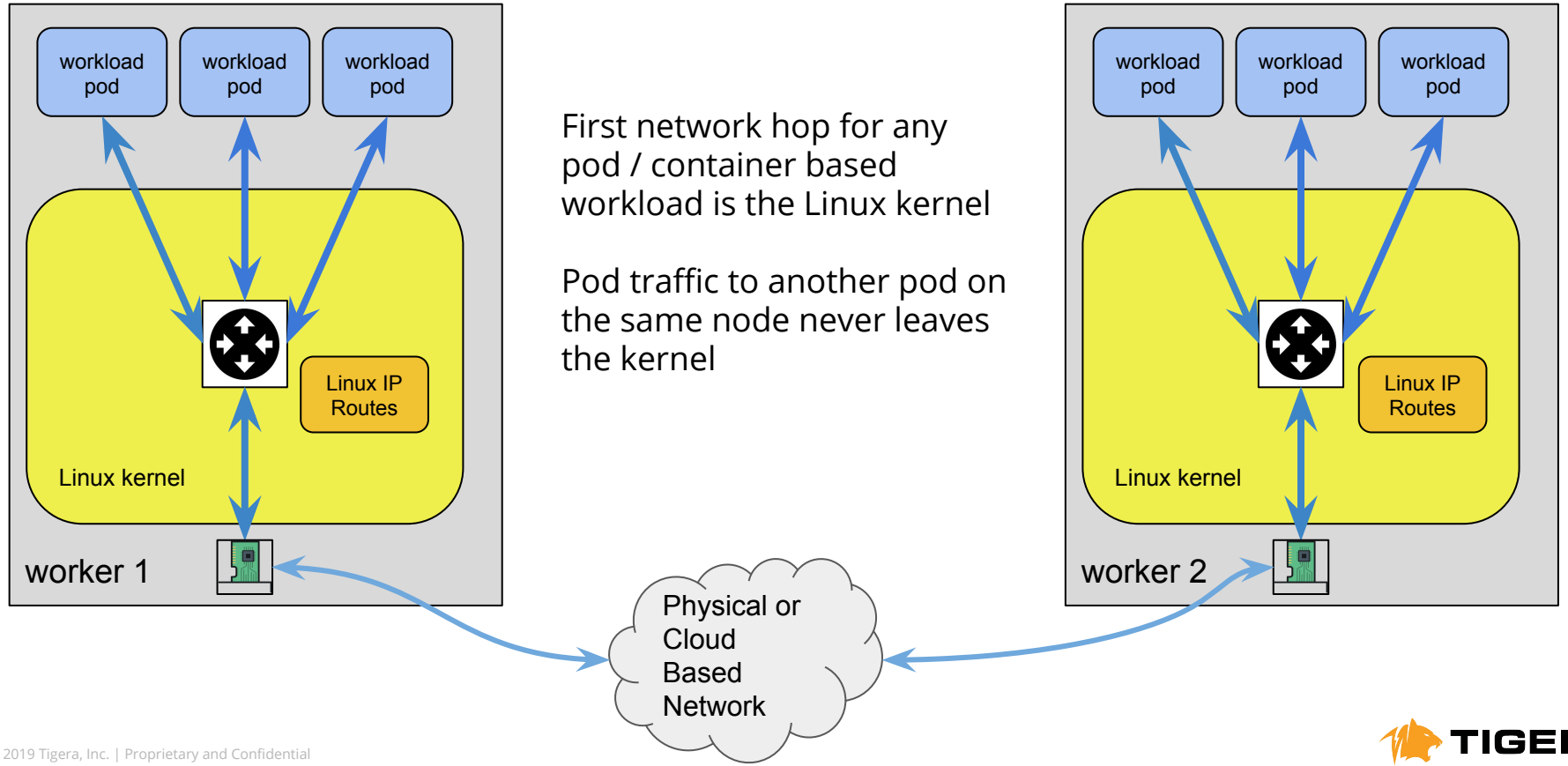
Calico Review

TIGERA

Kubernetes architecture



Intra Pod and Intra Node Networking

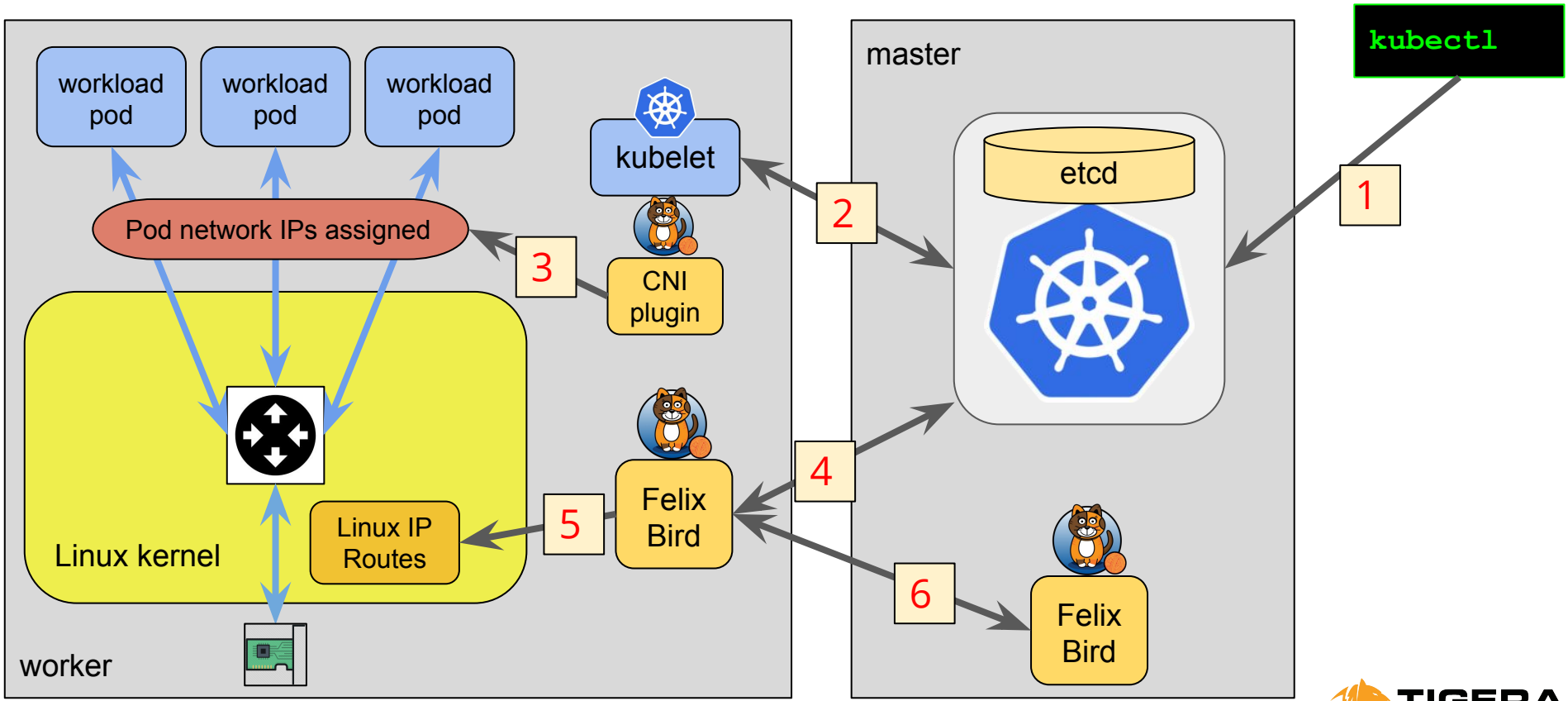


First network hop for any pod / container based workload is the Linux kernel

Pod traffic to another pod on the same node never leaves the kernel

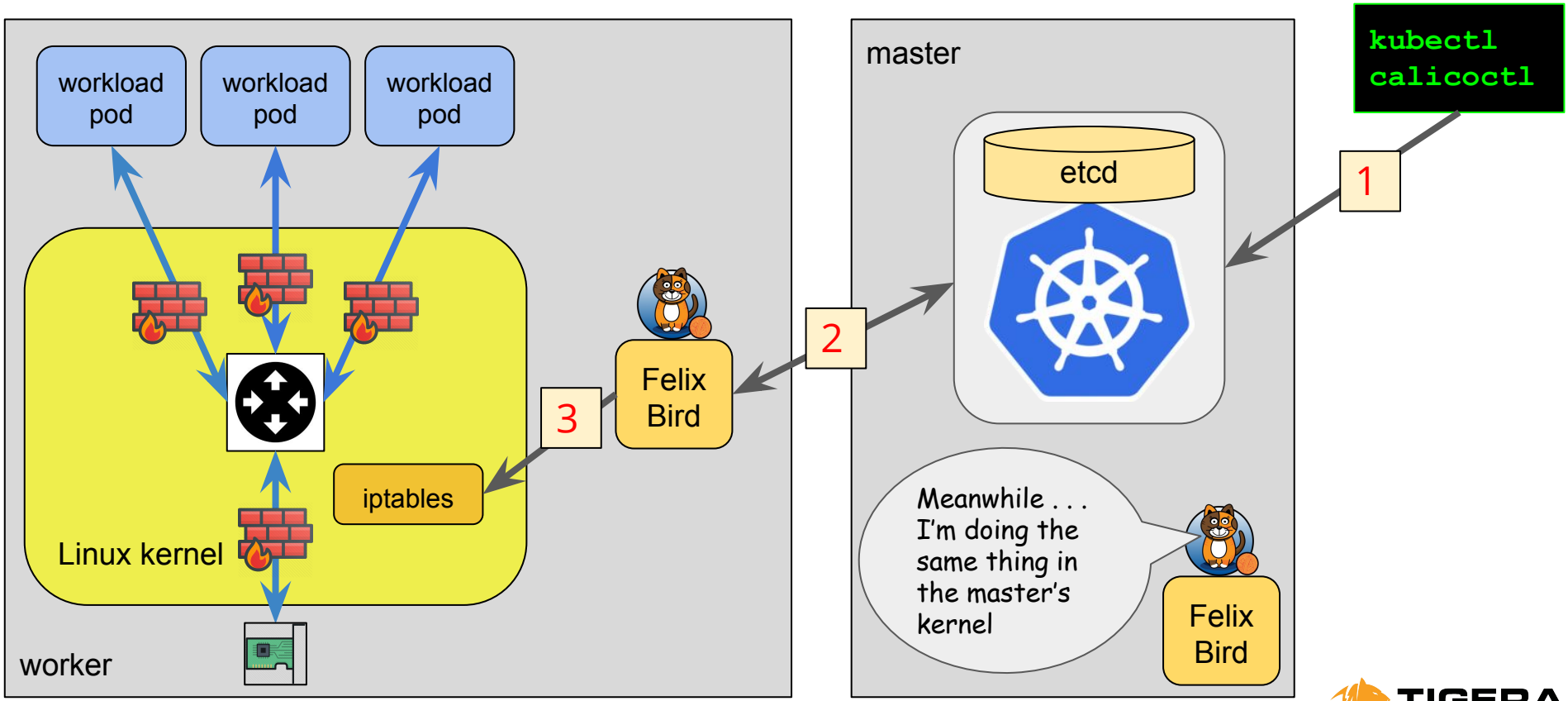
TIGERA

High Level Calico CNI Architecture



TIGERA

Calico High Level Policy Architecture



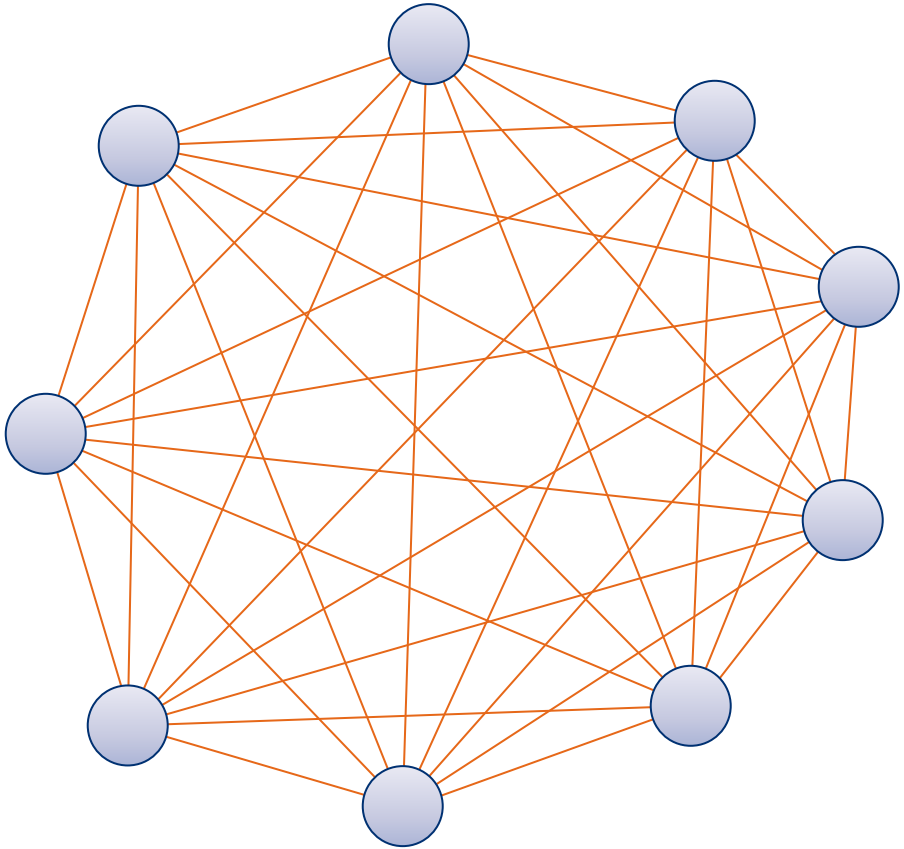
TIGERA

Declarative Network Policy

TIGERA

Motivation For Network Policy

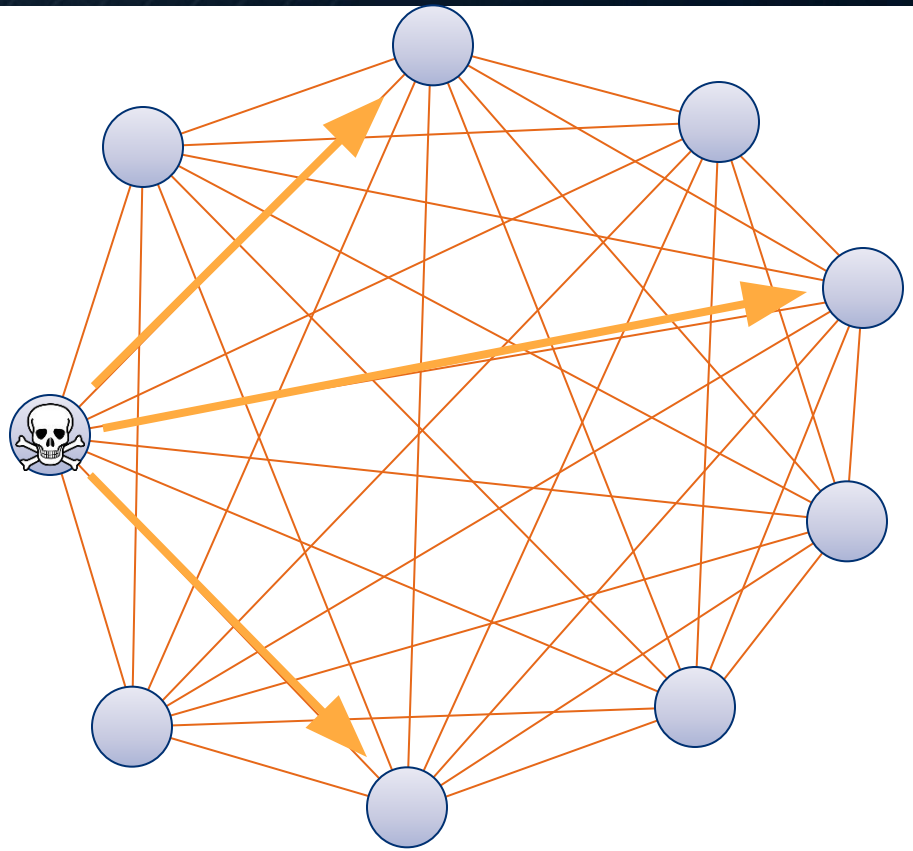
Consider a Kubernetes cluster of n services. Of the n^2 possible connections between all your services, only a tiny, tiny fraction of them are actually useful for your application.



TIGERA

Pathways for Attack

The rest are only useful for an attacker.

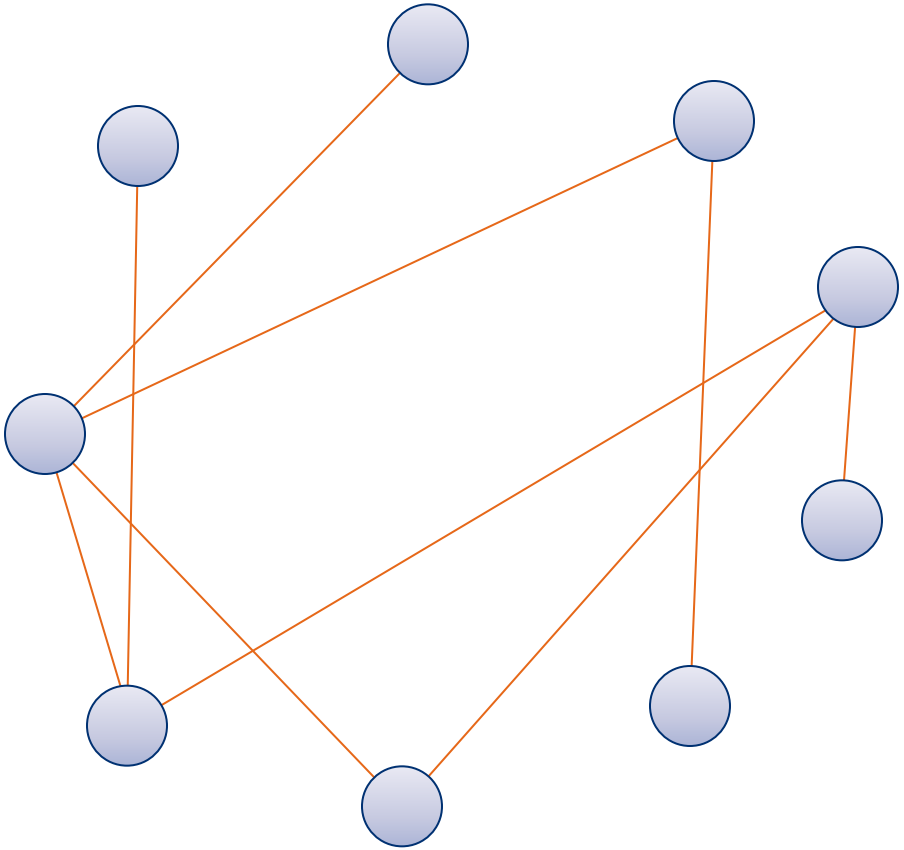


TIGERA

Identifying Unused Connections

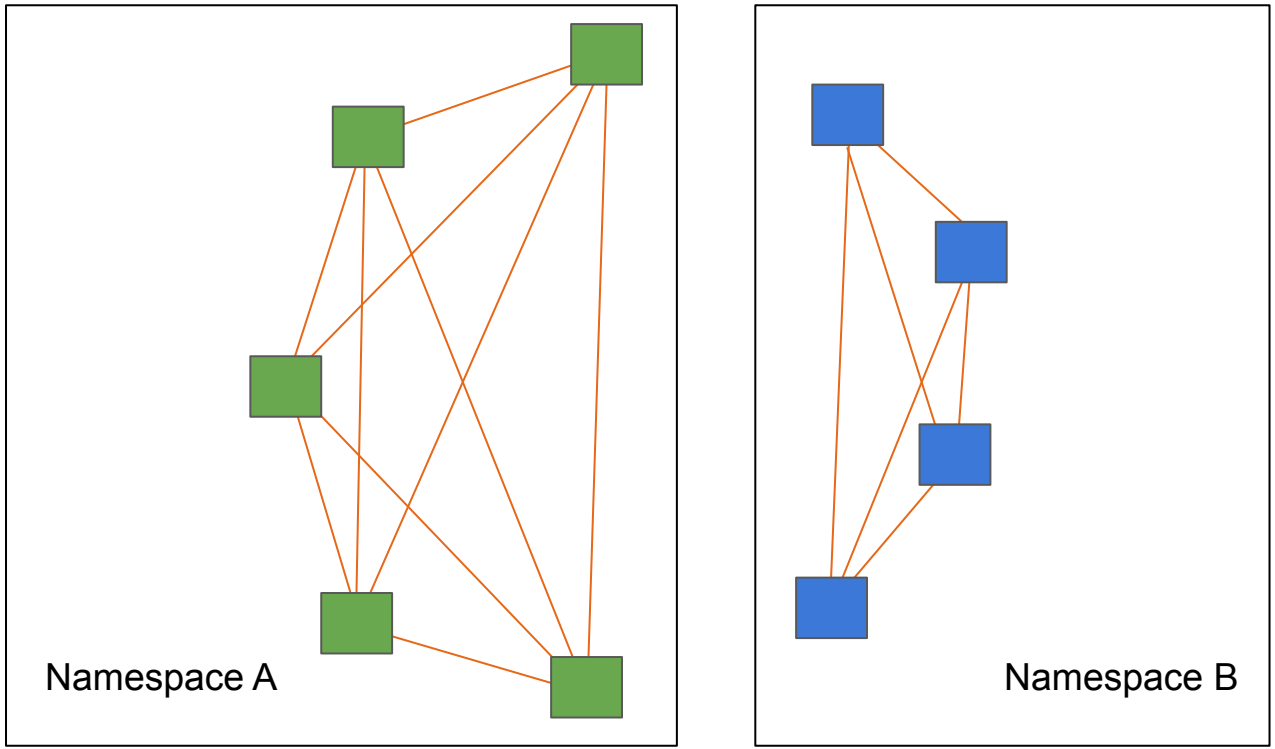
We know that our frontend load balancers don’t connect directly to the backend database. Dev containers do not connect to prod, and so on.

Each connection path that we can eliminate will reduce complexity and improve security.



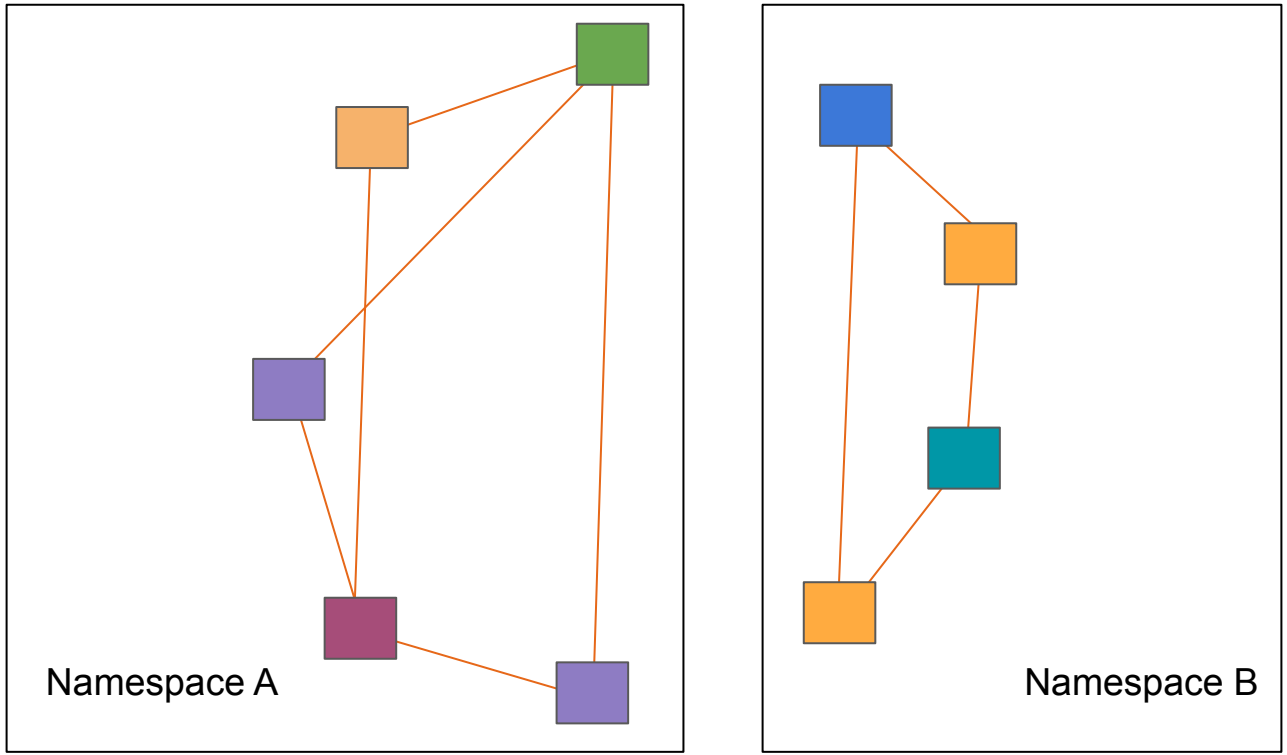
TIGERA

Namespace Isolation



TIGERA

Custom Isolation



TIGERA

Example Network Policy Use Cases

- Stage separation - isolate dev / test / prod instances
- Translation of traditional firewall rules - e.g. 3-tier
- “Tenant” separation - e.g. typically use namespaces for different teams within a company - but without network policy, they are not network isolated
- Fine-grained firewalls - reduce attack surface within microservice-based applications
- Compliance - e.g. PCI, HIPAA
- Any combination of the above

Key Kubernetes Concept: Labels

- Labels are key/value pairs that are attached to objects, such as pods.
- Labels are intended to be used to specify identifying attributes of objects that are meaningful and relevant to users, but do not directly imply semantics to the core system.
- Labels can be used to organize and to select subsets of objects.
- Labels can be attached to objects at creation time and subsequently added and modified at any time.
- Each Key must be unique for a given object.

```
"labels": {
  "key1" : "value1",
  "key2" : "value2"
}
```

Source: https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

Key Kubernetes Concept: Label Selectors

Via a *label selector*, the client/user can identify a set of objects. The label selector is the **core grouping primitive in Kubernetes.**

- Equality-based (==, !=)
- Set membership (in, notin, exists)

```
environment = production
tier != frontend
```

```
environment in (production, qa)
tier notin (frontend, backend)
partition
!partition
```


TIGERA

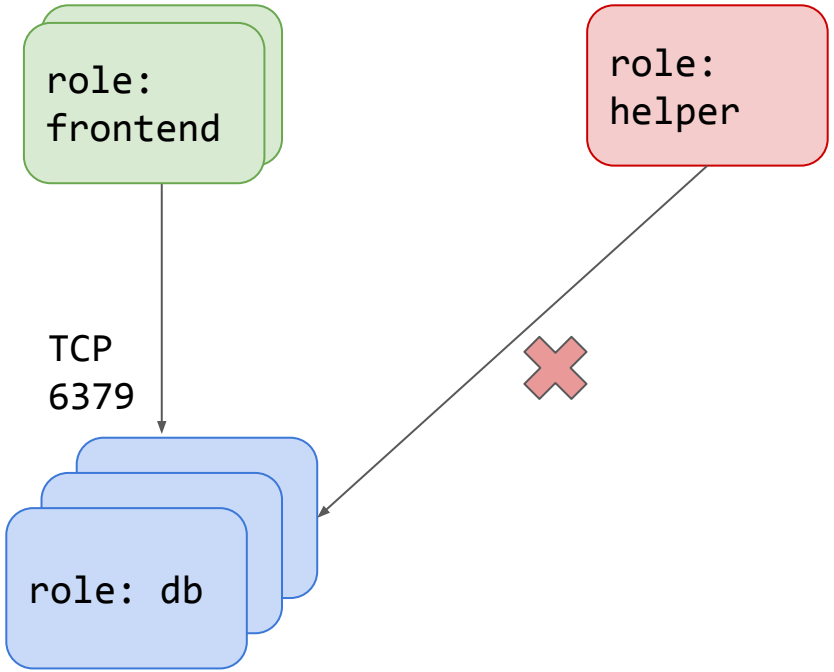
Kubernetes NetworkPolicy Resource

- Specifies how groups of pods are allowed to communicate with each other and other network endpoints using:
  - Pod label selector
  - Namespace label selector
  - Protocol + Ports
  - More to come in the future
- Pods selected by a NetworkPolicy:
  - Are isolated by default
  - Are allowed incoming traffic if that traffic matches a NetworkPolicy ingress rule.
- Requires a controller to implement the API

https://kubernetes.io/docs/concepts/services-networking/network-policies/

Kubernetes Network Policy Example

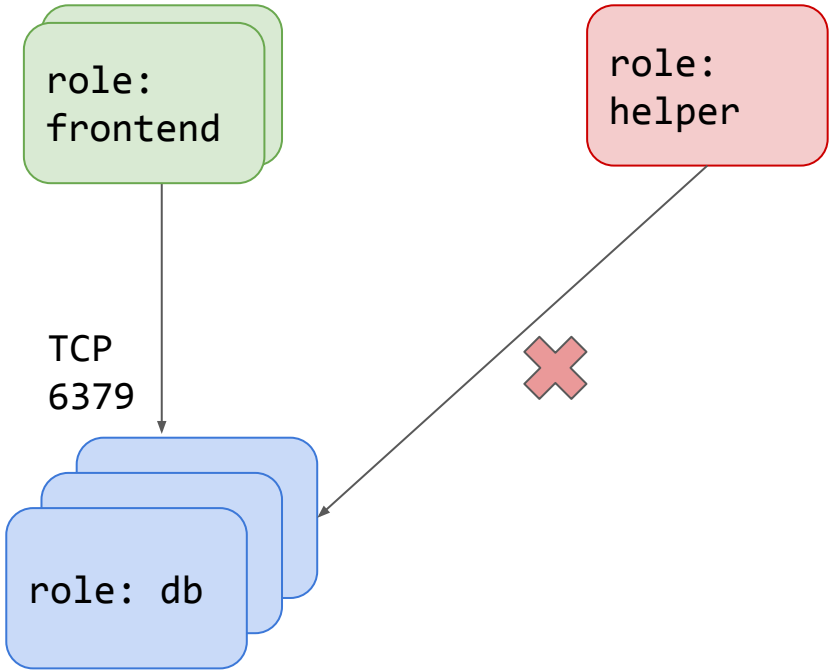
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
```



TIGERA

Kubernetes Network Policy Example

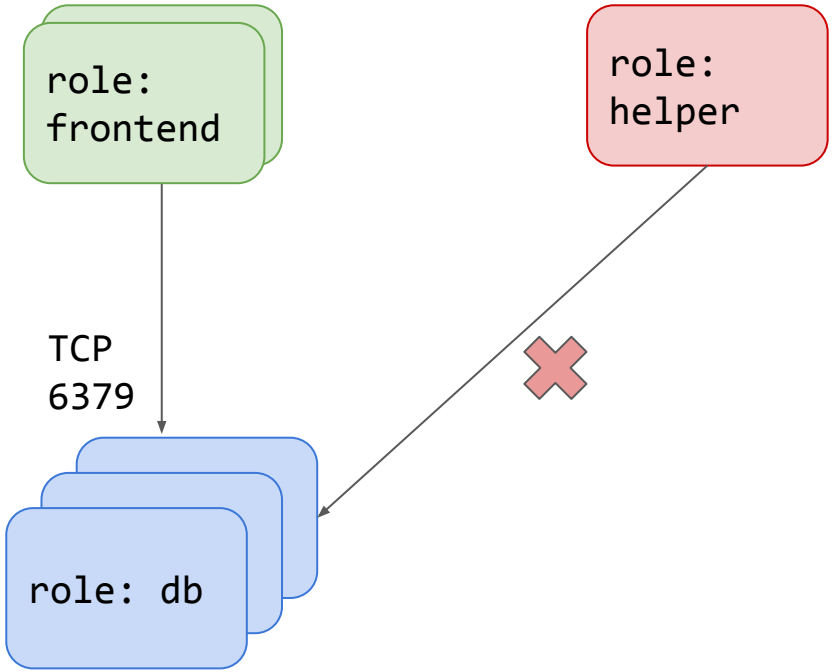
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
```



TIGERA

Kubernetes Network Policy Example

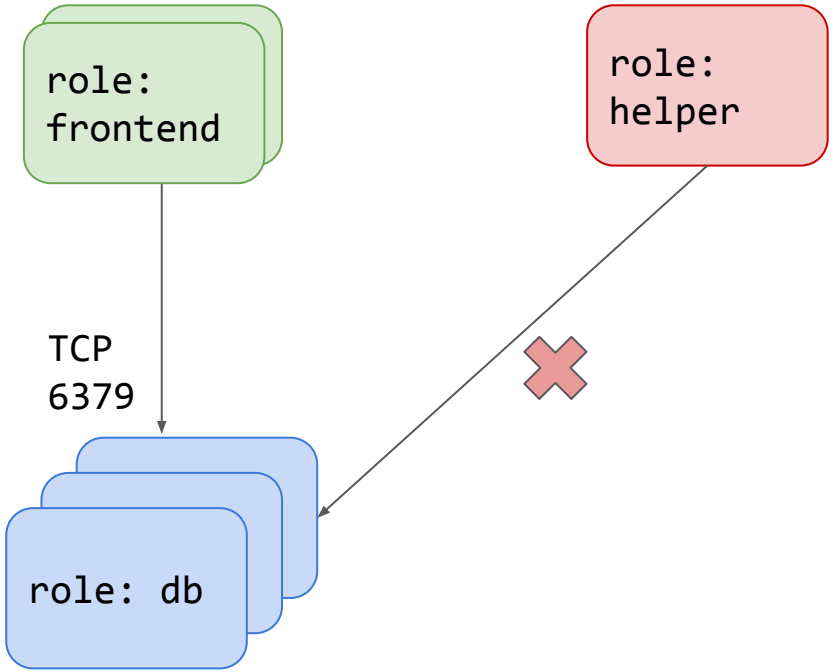
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
```



TIGERA

Kubernetes Network Policy Example

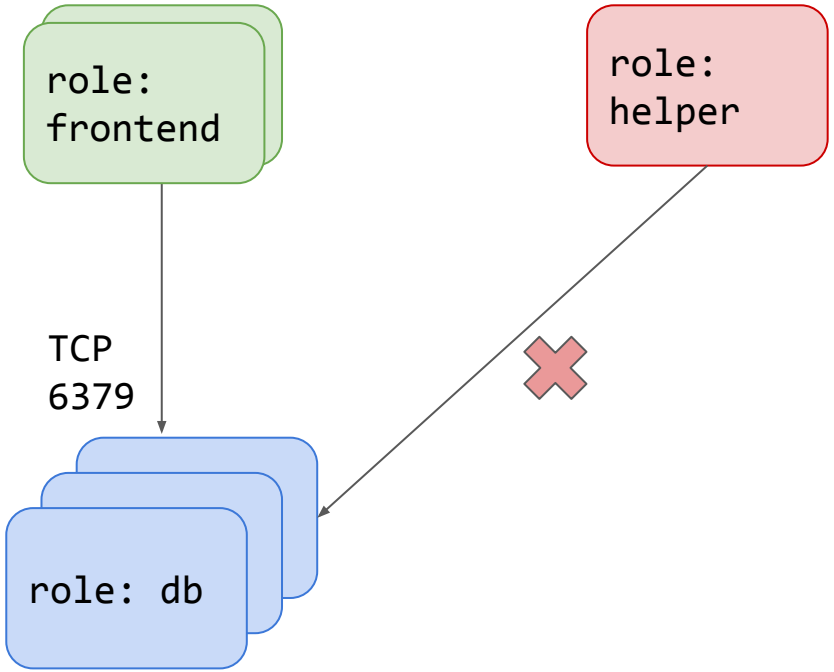
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
```



TIGERA

Kubernetes Network Policy Example

```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
  - from:
    - podSelector:
        matchLabels:
          role: frontend
    ports:
    - protocol: TCP
      port: 6379
```



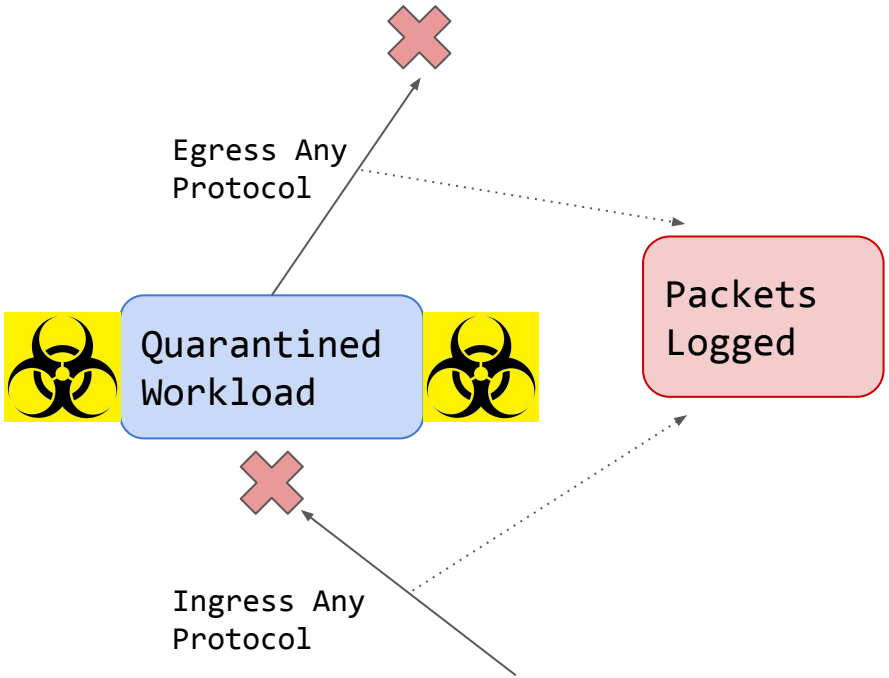
TIGERA

Advanced use cases beyond Kubernetes Network Policy

- **Global** network policies (rather than being limited to apply to pods in a single namespace)
- Ability to specify policy rules that **reference labels of pods in other namespaces** (rather than only being allowed to allow/deny all pods from another namespace)
- Ability to specify policy rules that **reference service account labels**
- **Richer label expressions** and **traffic specification** (e.g. port ranges, protocols other than tcp/udp, negative matching on protocol type, ICMP type)
- **Deny** rules
- **Policy order**/prioritization (which becomes necessary when you have mix of deny and allow rules)
- **Network sets** - ability to apply labels to a set of IP addresses, so they can be selected by label selector expressions
- Support for **non-Kubernetes nodes** (e.g. standalone hosts) within the policy domain,
- Policy options specific to **host endpoints** (apply-on-forward, pre-DNAT, doNotTrack)
- If you need these capabilities, use Calico directly instead of via Kubernetes Network Policy API
- https://docs.projectcalico.org/v3.7/security/calico-network-policy

Calico Network Policy Example

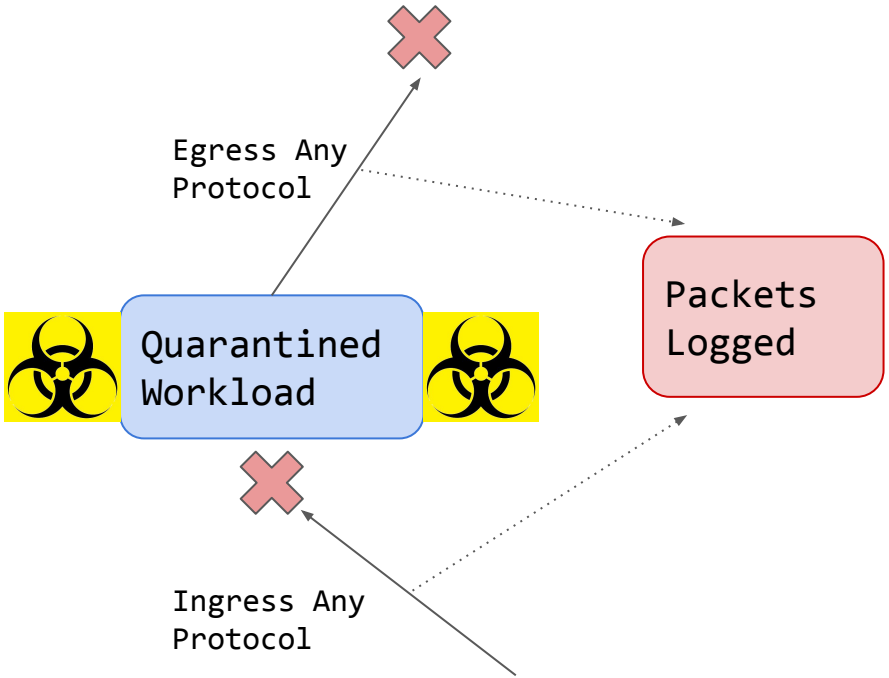
```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: security-operations.quarantine
spec:
  order: 200
  selector: quarantine == "true"
  ingress:
    - action: Log
      source: {}
      destination: {}
    - action: Deny
      source: {}
      destination: {}
  egress:
    - action: Log
      source: {}
      destination: {}
    - action: Deny
      source: {}
      destination: {}
    types:
    - Ingress
    - Egress
```



TIGERA

Calico Network Policy Example

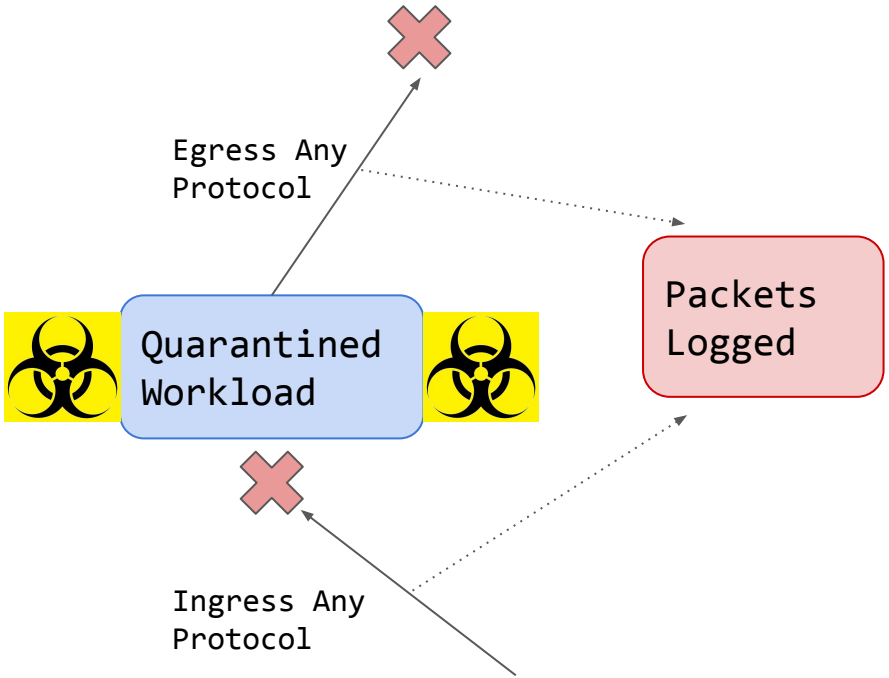
```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: security-operations.quarantine
spec:
  order: 200
  selector: quarantine == "true"
  ingress:
    - action: Log
      source: {}
      destination: {}
    - action: Deny
      source: {}
      destination: {}
  egress:
    - action: Log
      source: {}
      destination: {}
    - action: Deny
      source: {}
      destination: {}
    types:
    - Ingress
    - Egress
```



TIGERA

Calico Network Policy Example

```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: security-operations.quarantine
spec:
  order: 200
  selector: quarantine == "true"
  ingress:
    - action: Log
      source: {}
      destination: {}
    - action: Deny
      source: {}
      destination: {}
  egress:
    - action: Log
      source: {}
      destination: {}
    - action: Deny
      source: {}
      destination: {}
  types:
  - Ingress
  - Egress
```



TIGERA

Calicoctl: The Calico Command Line

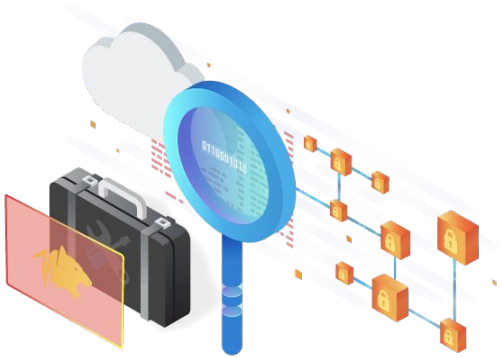
- Calicoctl is the command line utility for advanced Calico configuration
- Download: https://github.com/projectcalico/calicoctl/releases
- Reference: https://docs.projectcalico.org/latest/reference/calicoctl/

FELIX TIP: Calicoctl is available for Linux, MacOS and Windows. Download it for your favorite desktop operating system!

Tigera Secure Enables Key Security Capabilities

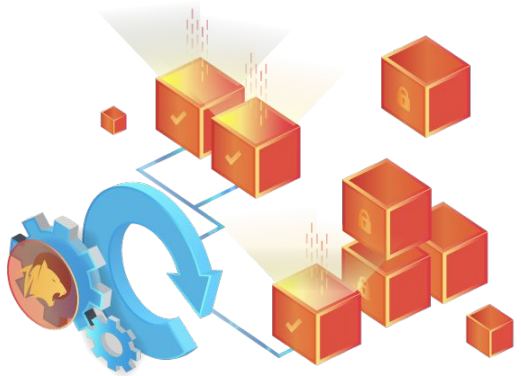
Zero Trust Network Security

Extend your security controls to Kubernetes



Visibility & Threat Detection

Monitor traffic, detect and prevent threats



Continuous Compliance

Continuous reporting, alert on non-compliance

More Tigera Webinars

NEXT Webinar

Kubernetes, Helm, and Network Security - Best Practices at Scale

https://www.brighttalk.com/webcast/17114/362773

'MUST SEE' ON-DEMAND WEBINAR

Atlassian Case Study - Moving to Kubernetes on AWS with Zero Trust Security

https://www.tigera.io/webinars/aws-tigera-atlassian

TIGERA

Q&A

Schedule a Demo: tigera.io/demo

TIGERA

Follow us on: