

# Herding Cats: Deep Dive Into Calico Policies for Kubernetes

Minneapolis Cloud Native Meetup  
May 23, 2019



© 2018 Tigera, Inc. | Proprietary and Confidential

# WELCOME



## Your Presenter:

Drew Oetzel  
Senior Field Engineer @ Tigera

✉ [drew@tigera.io](mailto:drew@tigera.io)

 <https://www.linkedin.com/in/andrewoetzel/>

 <https://github.com/drew-tigera/HerdingCats>



# WHO IS TIGERA?

An 80s hair metal band???

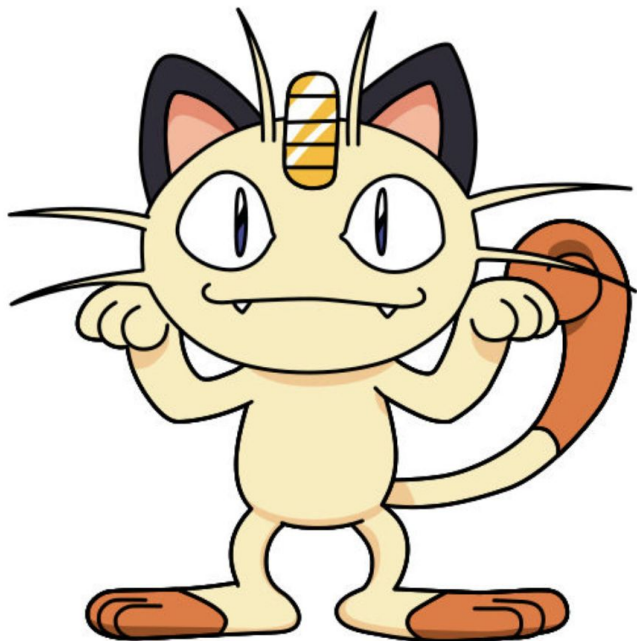


No . . . that's:

**PANTERA**

# WHO IS TIGERA?

A super secret Pokemon?



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

# WHO IS TIGERA?



**Large cloud &  
enterprise customers**



**Calico: Open Source,  
widely adopted**



**Tigera Secure  
Enterprise Edition**





# 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



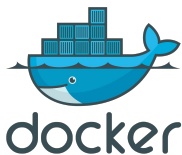
Partnered to support network policy in Google Container Engine (GKE)



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

# 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](http://www.projectcalico.org)**

# 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.**

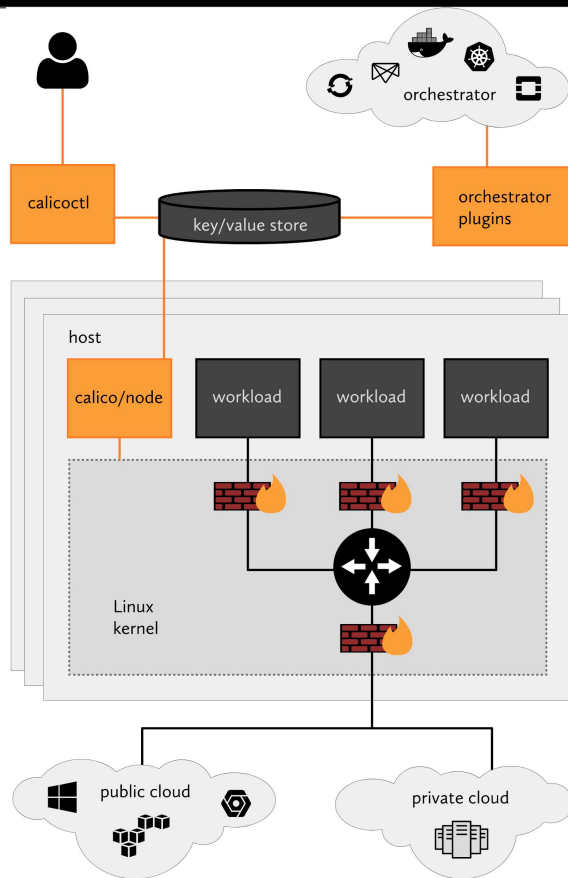
- Secure your East/West traffic
- Prevent malicious behavior from spreading
- Block insider threat / insider error



# CALICO REVIEW



# HIGH LEVEL CALICO ARCHITECTURE: SUMMARY

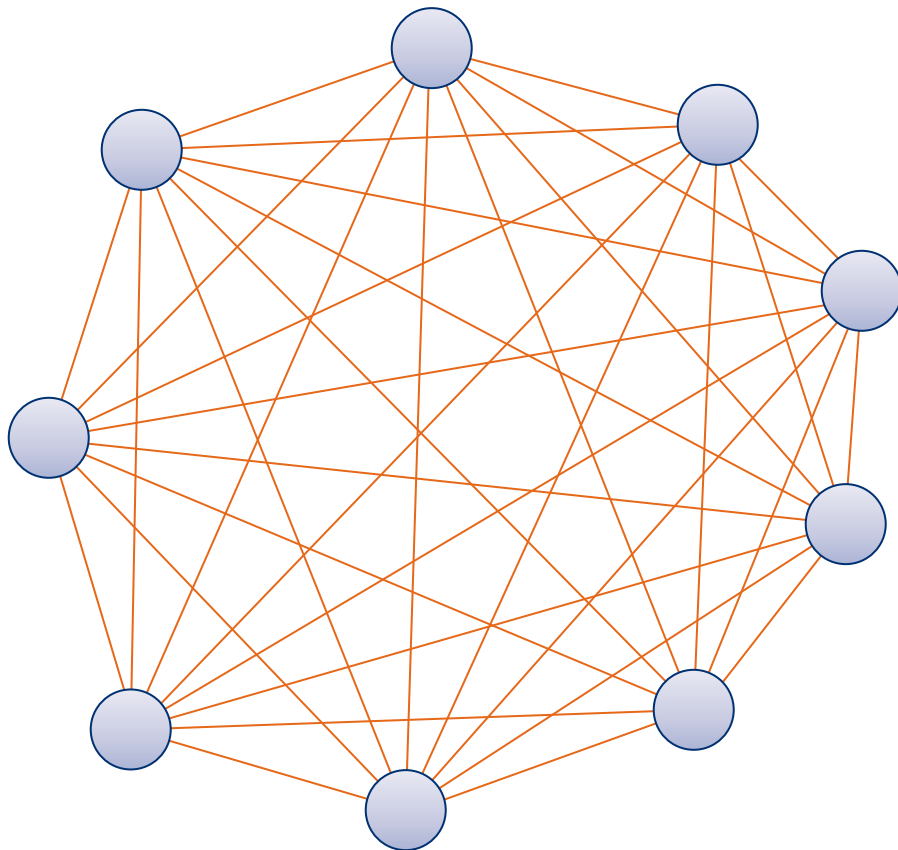


# Declarative Network Policy

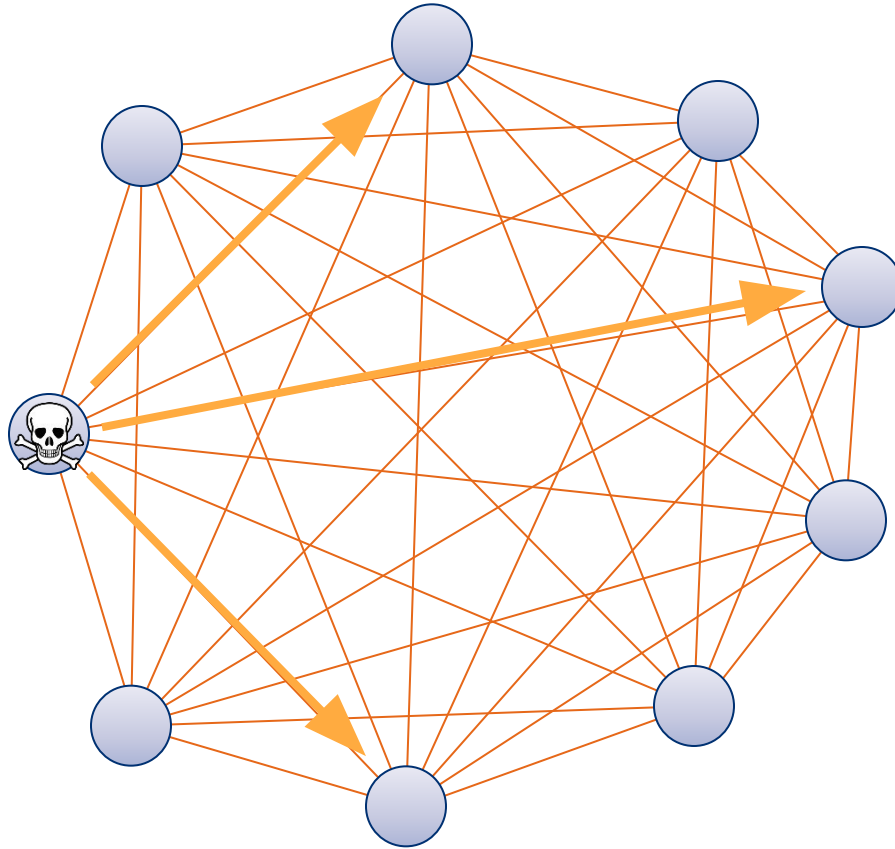


# MOTIVATION FOR NETWORK POLICY

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.

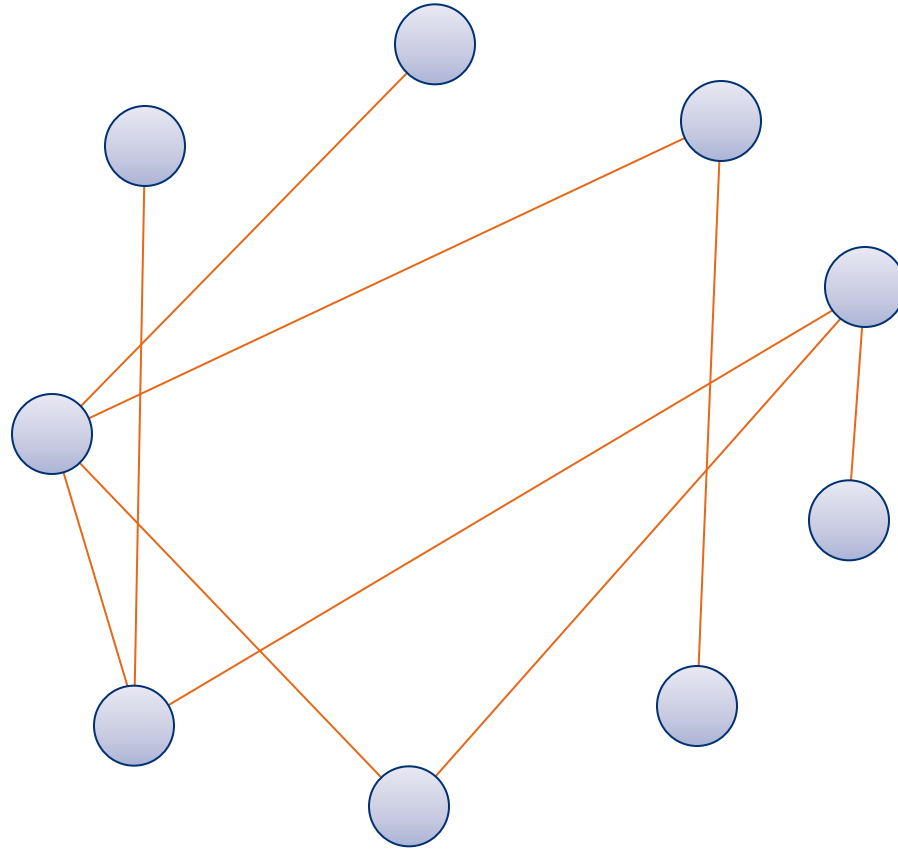


The rest are only  
useful for an  
attacker.



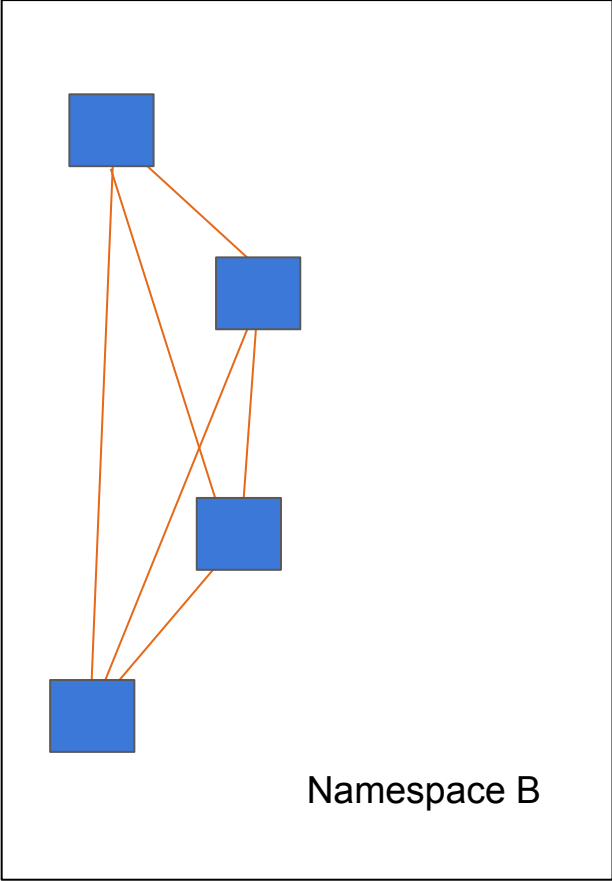
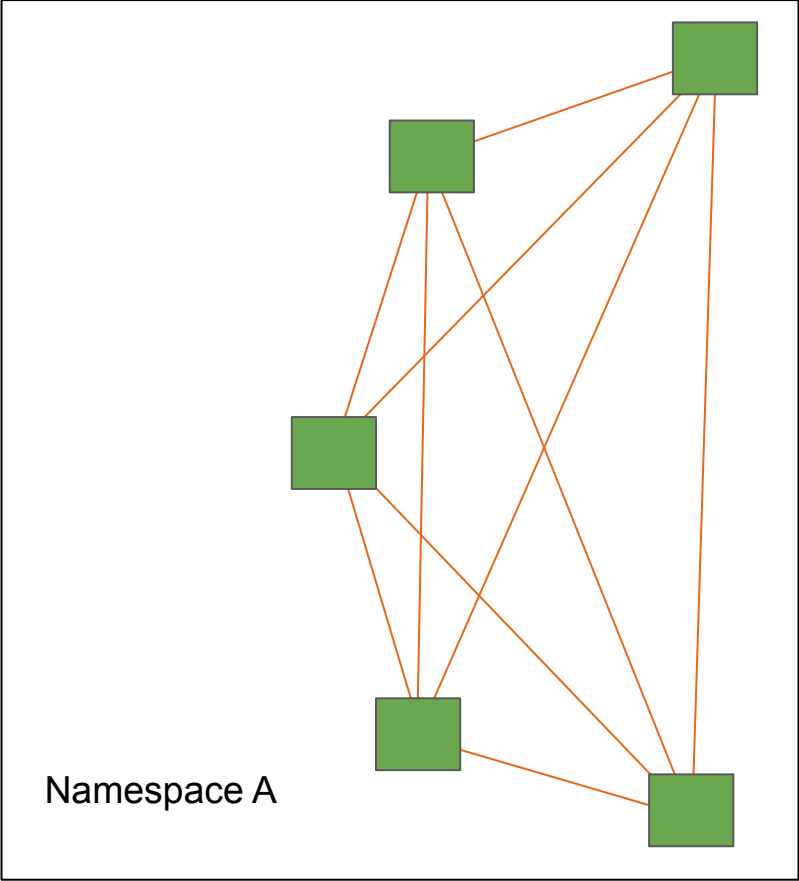
We know that our frontend load balancers don't connect directly to the backend database. dev containers do not connect to prod, and so on.

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

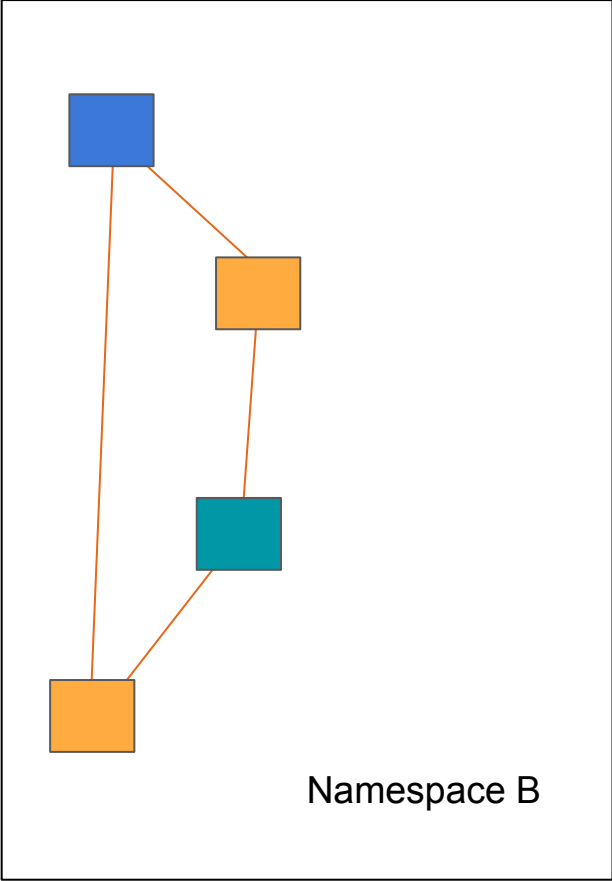
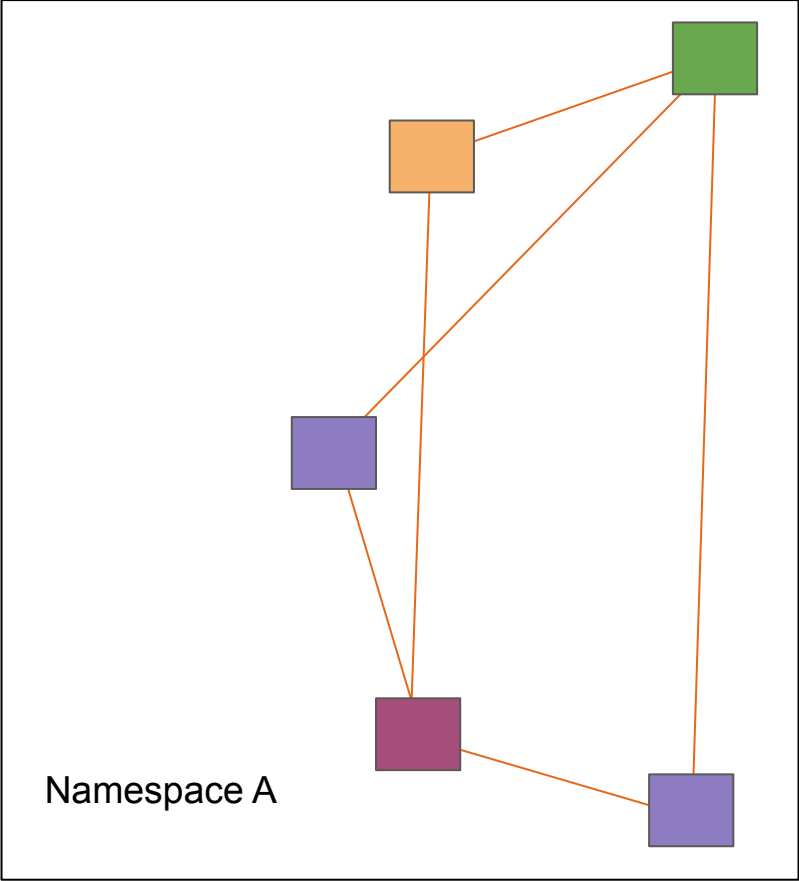




Namespace Isolation



Custom Isolation



# 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 (==, !=)

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

> Set membership (in, notin, exists)

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

# KUBERNETES NetworkPolicy RESOURCE

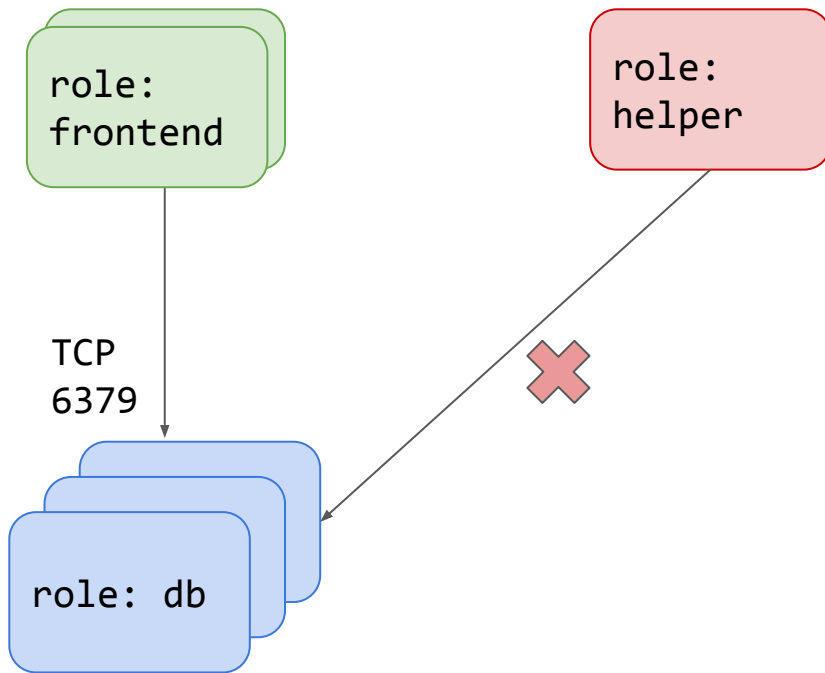
- > Specifies how groups of pods are allowed to communicate with each other and other network endpoints using:
  - Pod label selector
  - Namespace label selector
  - Protocol + Ports
  - More to come in the future
- > Pods selected by a NetworkPolicy:
  - Are isolated by default
  - Are allowed incoming 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
```



# 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:
```

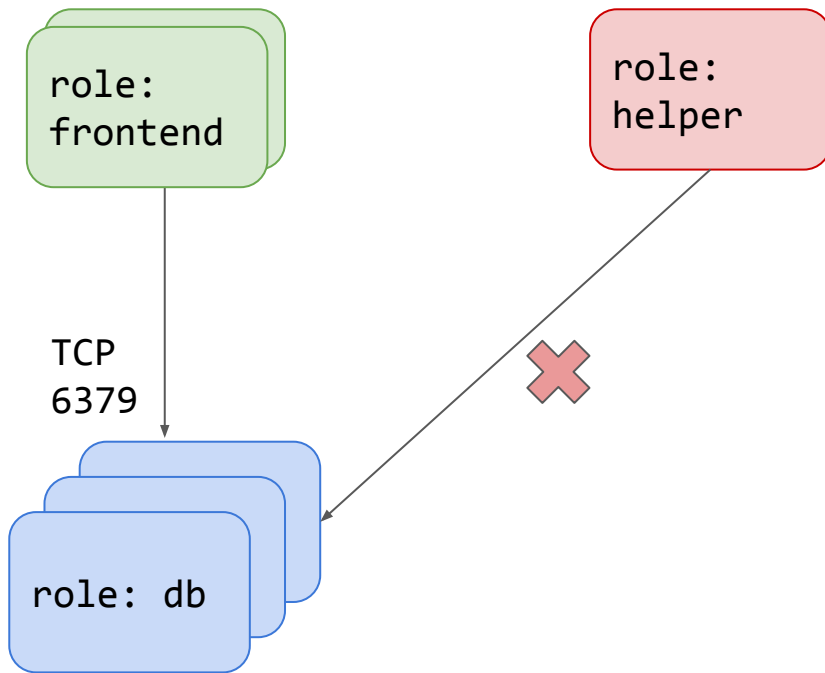
```
        matchLabels:
```

```
          role: frontend
```

```
    ports:
```

```
      - protocol: TCP
```

```
        port: 6379
```



# 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:
```

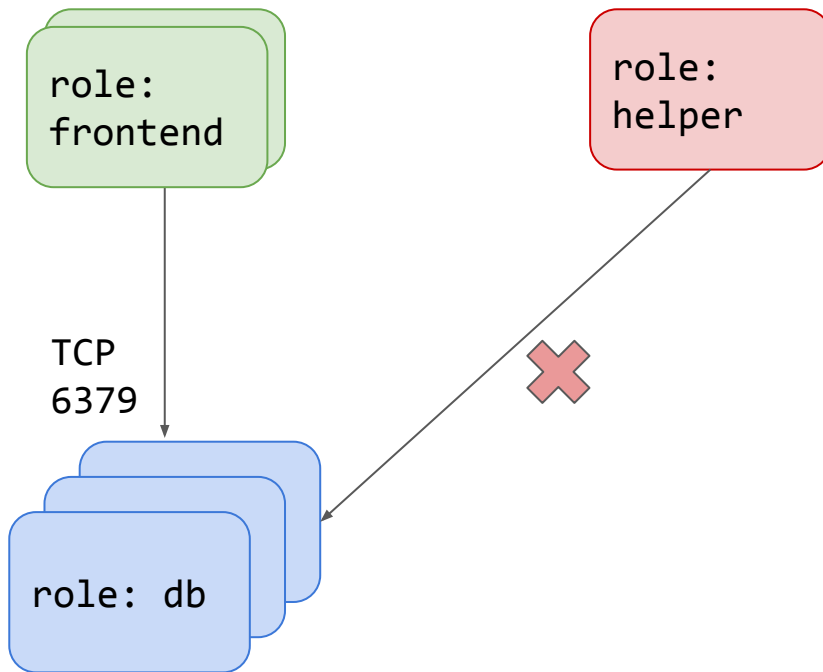
```
      matchLabels:
```

```
        role: frontend
```

```
  ports:
```

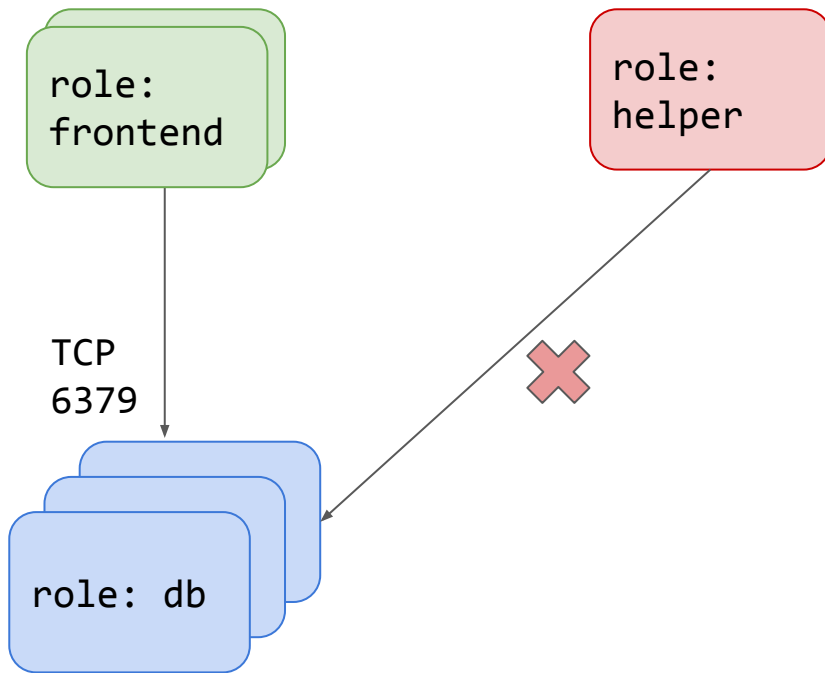
```
  - protocol: TCP
```

```
    port: 6379
```



# Kubernetes Network Policy Example

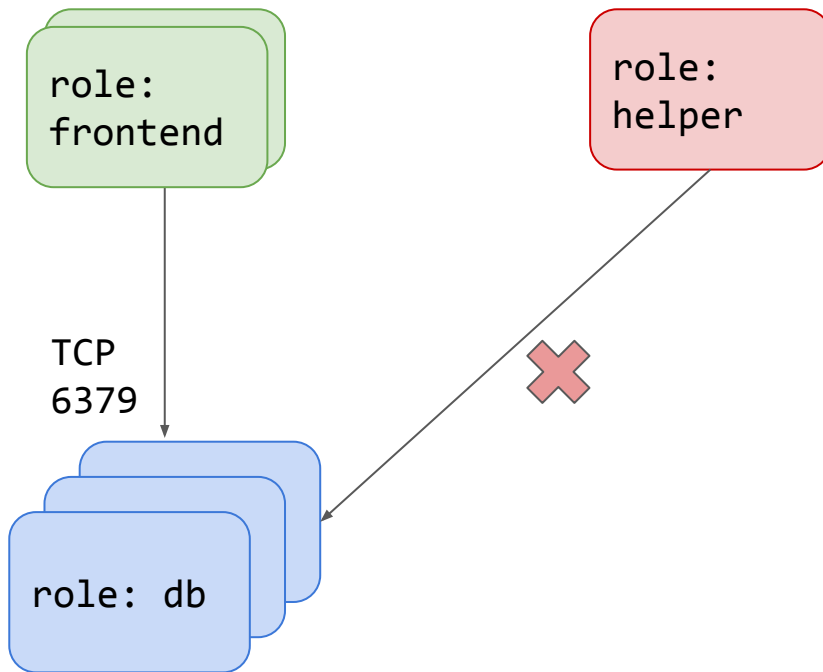
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
    - from:
      - podSelector:
          matchLabels:
            role: frontend
        ports:
          - protocol: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



# 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
```



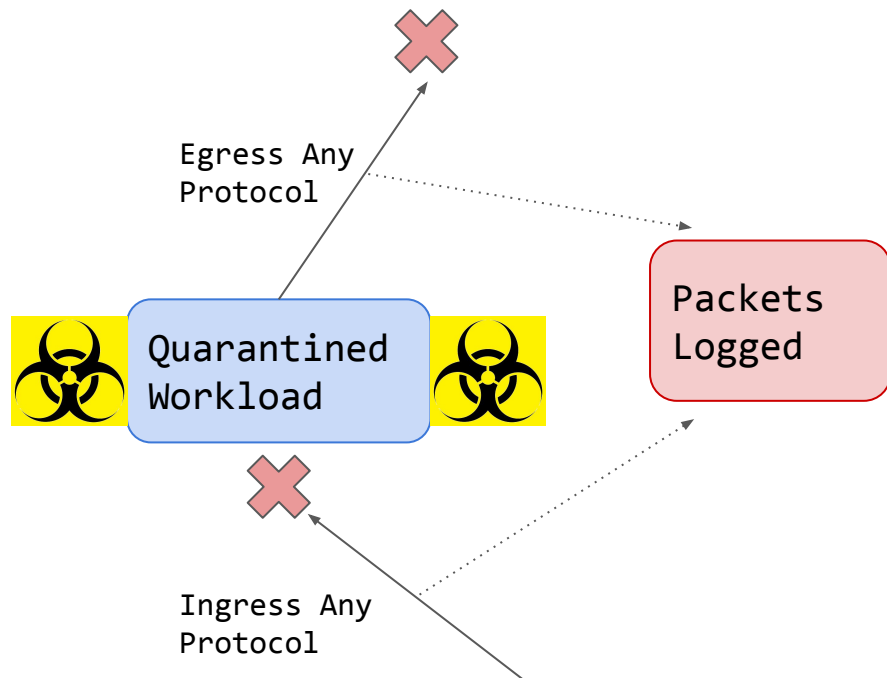
# CALICO NETWORK POLICY

- > Calico supports more advanced use cases beyond Kubernetes Network Policy
  - **global** network policies (rather than being limited to apply to pods in a single namespace)
  - the 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)
  - the 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
```



# Calico Network Policy Example

```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: security-operations.quarantine
```

## spec:

```
  order: 200
```

```
  selector: quarantine == "true"
```

## ingress:

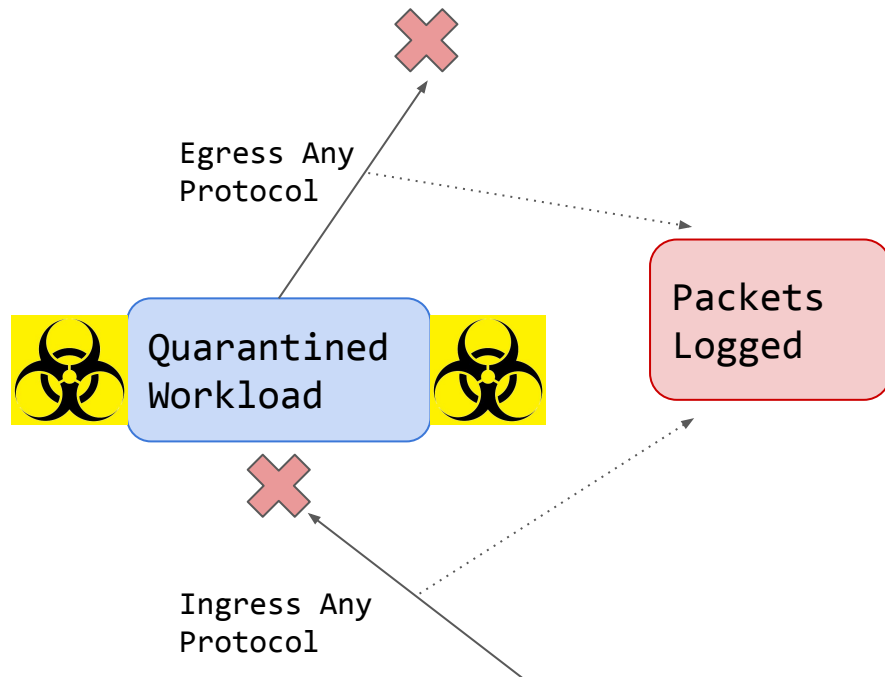
- action: Log  
 source: {}  
 destination: {}
- action: Deny  
 source: {}  
 destination: {}

## egress:

- action: Log  
 source: {}  
 destination: {}
- action: Deny  
 source: {}  
 destination: {}

## types:

- Ingress
- Egress



# 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: {}

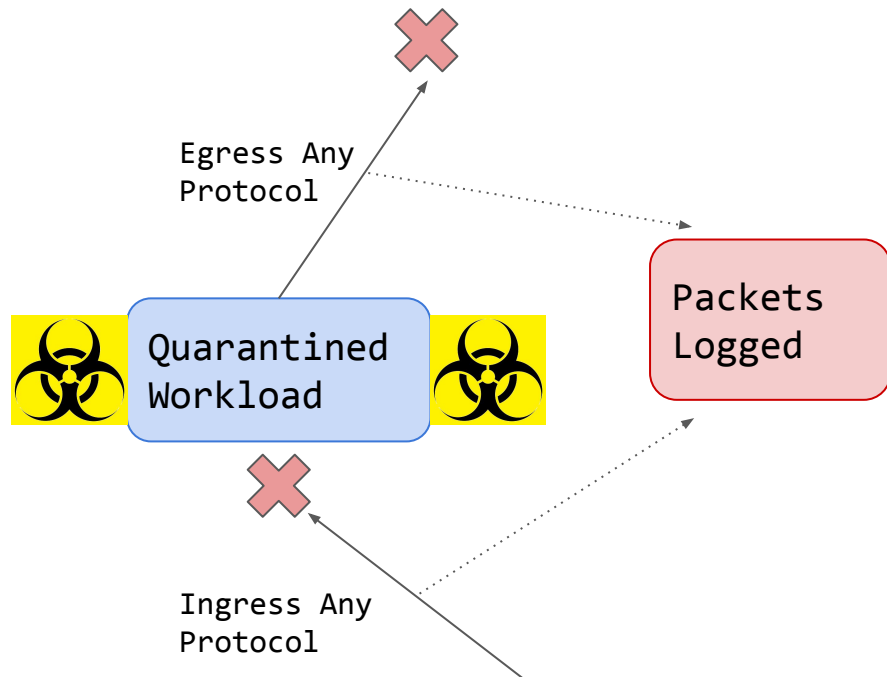
## egress:

- action: Log  
source: {}  
destination: {}
- action: Deny  
source: {}  
destination: {}

## types:

- Ingress

- Egress



# 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!

# Thank you!

