

Securing Your K8s Infrastructure With Tigera Secure



18 July 2019

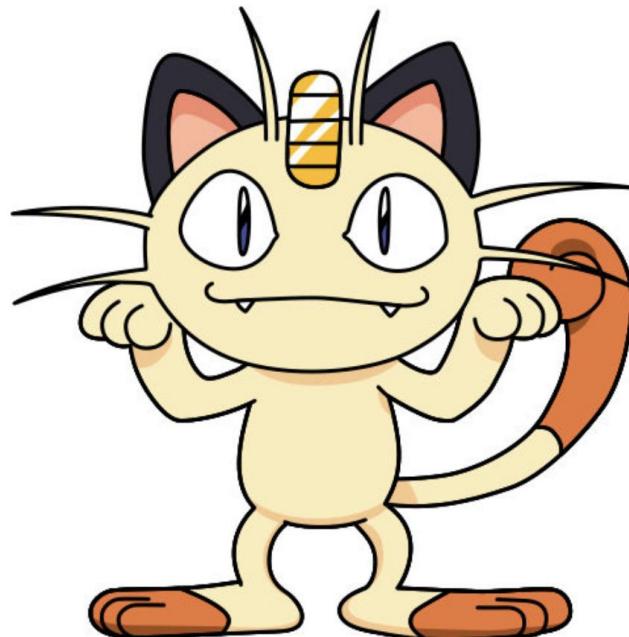
Drew Oetzel - Senior Technical Solutions Engineer



- 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!

Who is Tigera?

A super secret Pokemon?



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

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



Tigera Strategic Partnerships



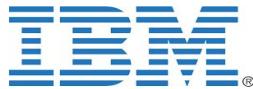
Integrated with ACS Engine, Collaborating on Azure Kubernetes Service (AKS) and Windows



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

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

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

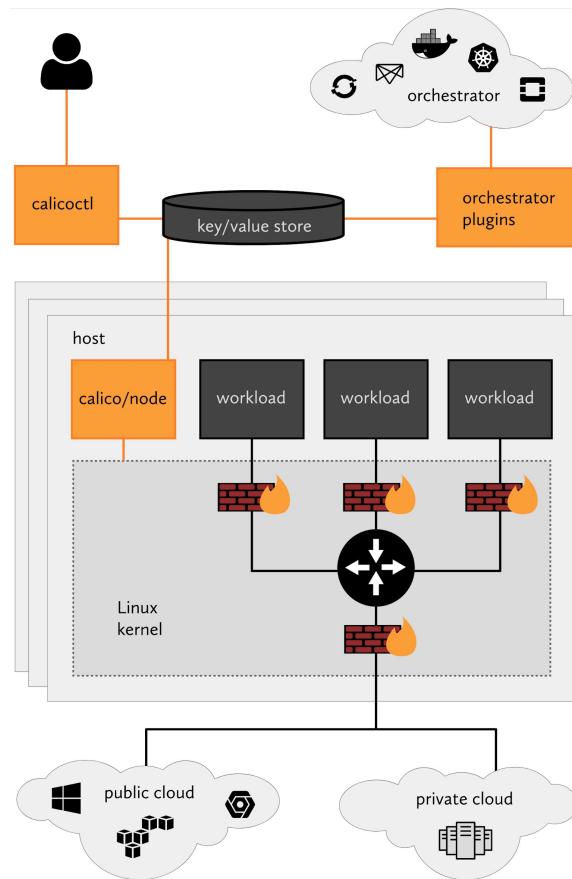
Secure your inner K8s networking with policies.

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

Calico Review



High Level Calico Architecture: Summary

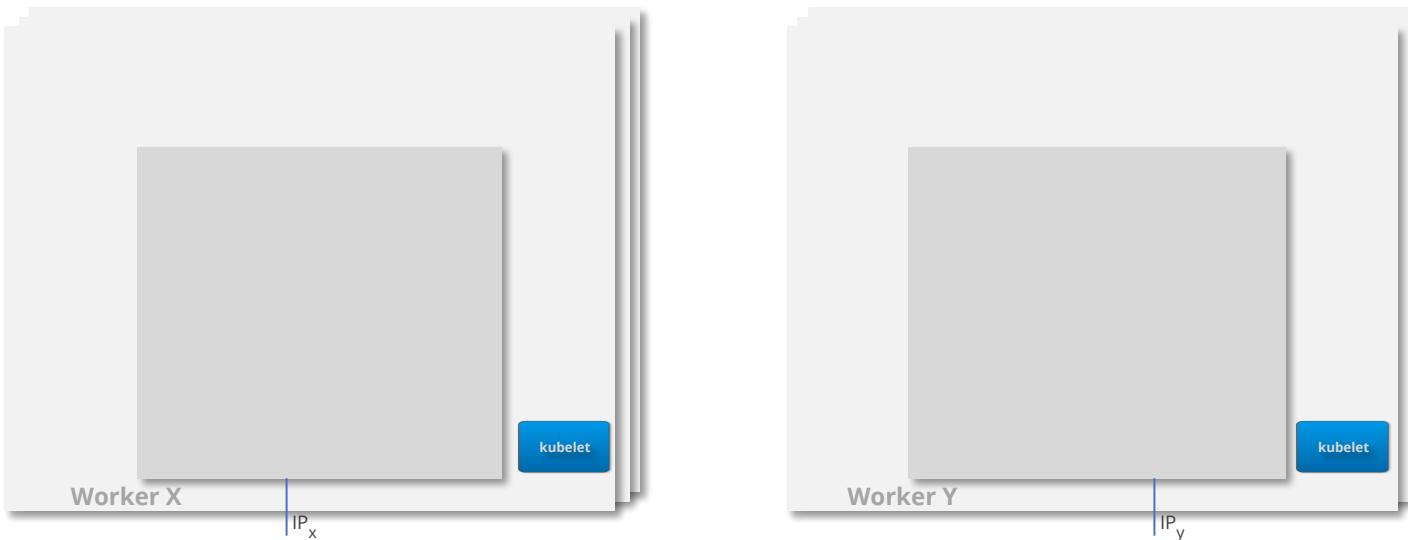
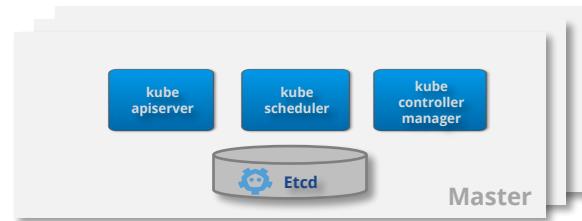


Calico CNI and IPAM in Action



Pod Lifecycle

Lets start with baseline K8s nodes



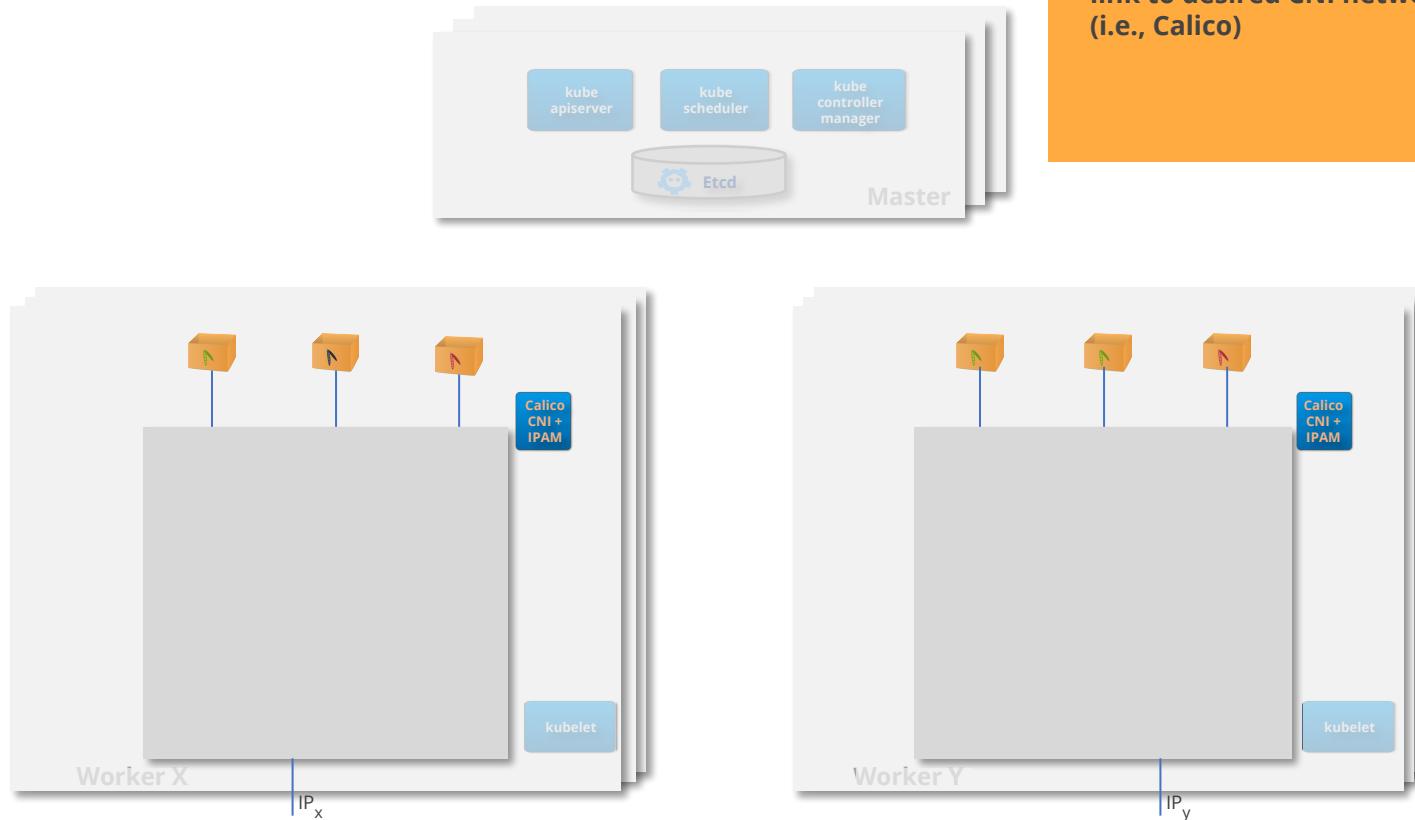
Pod Lifecycle



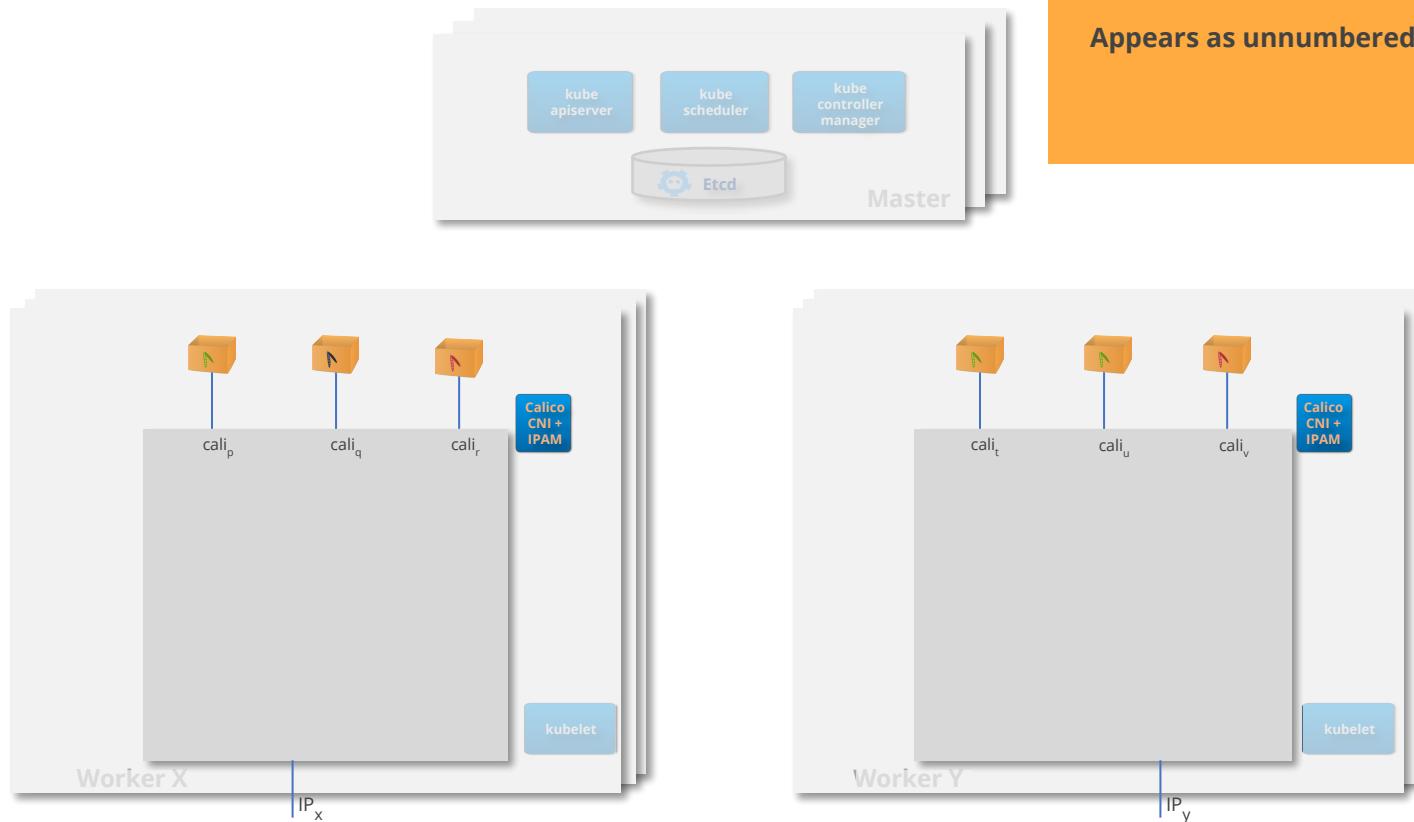
Pod Lifecycle: CNI



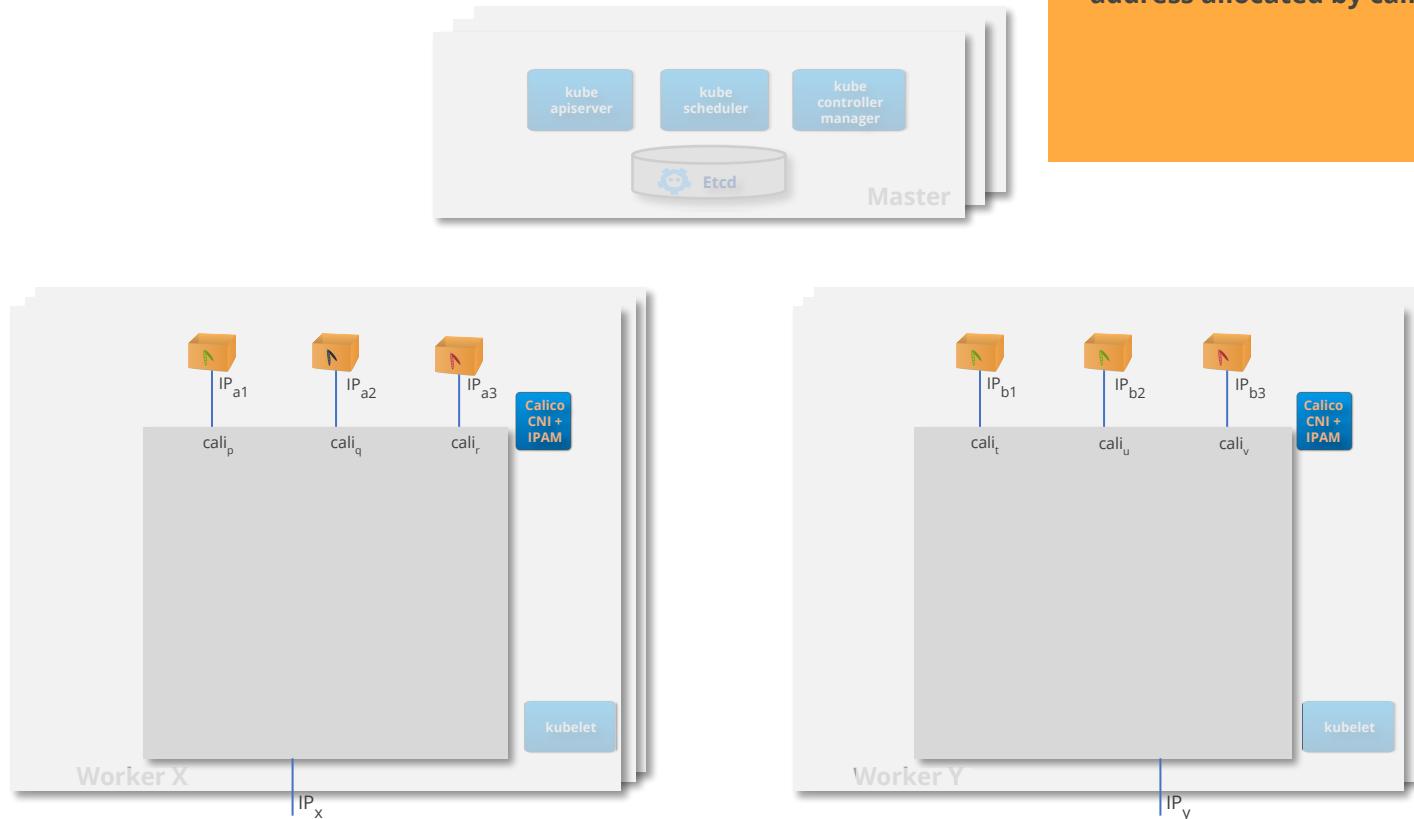
Pod Lifecycle: CNI



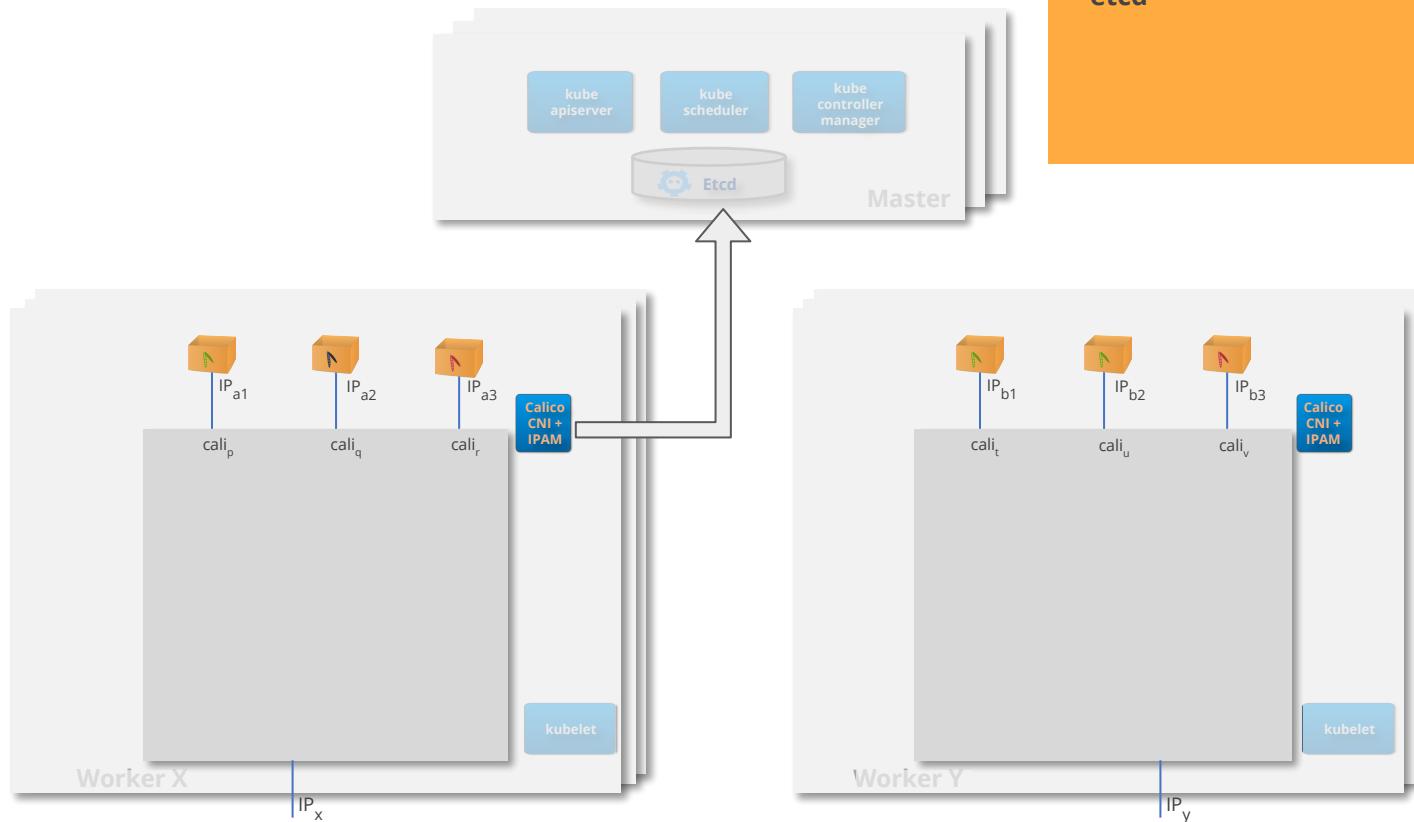
Pod Lifecycle: CNI



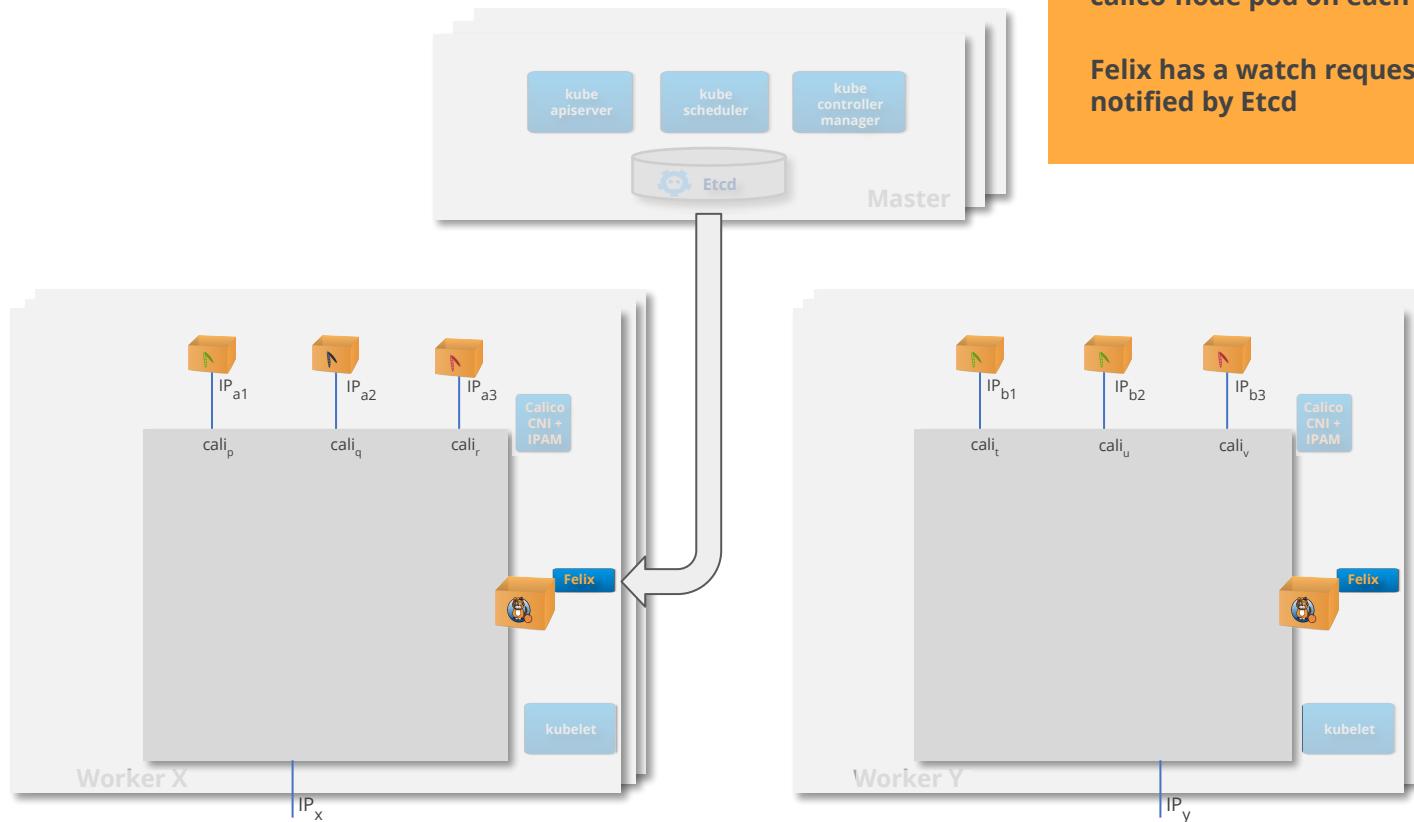
Pod Lifecycle: Address Block Assignment



Pod Lifecycle



Pod Lifecycle

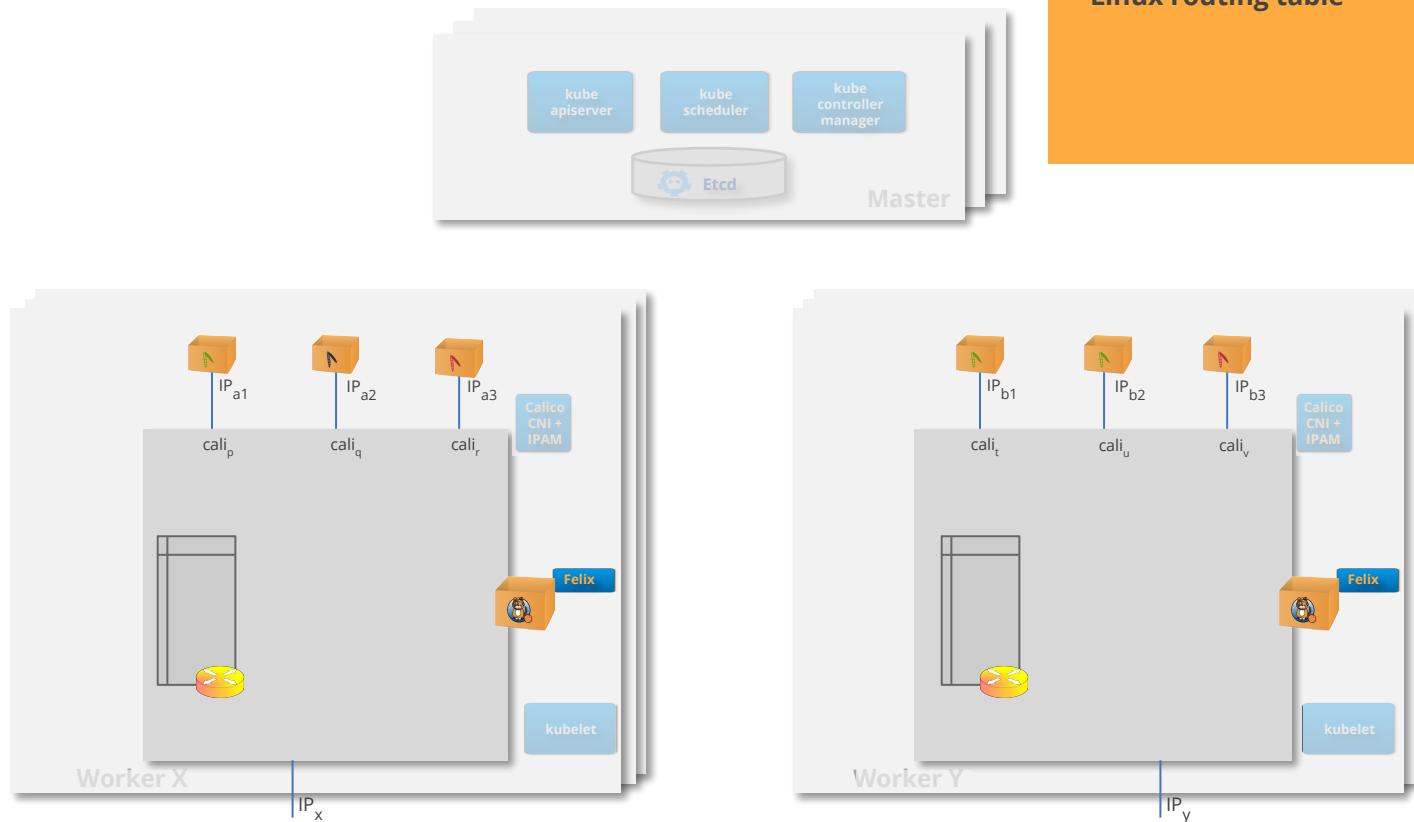


Felix daemon process runs inside the calico-node pod on each host

Felix has a watch requesting to be notified by Etcd

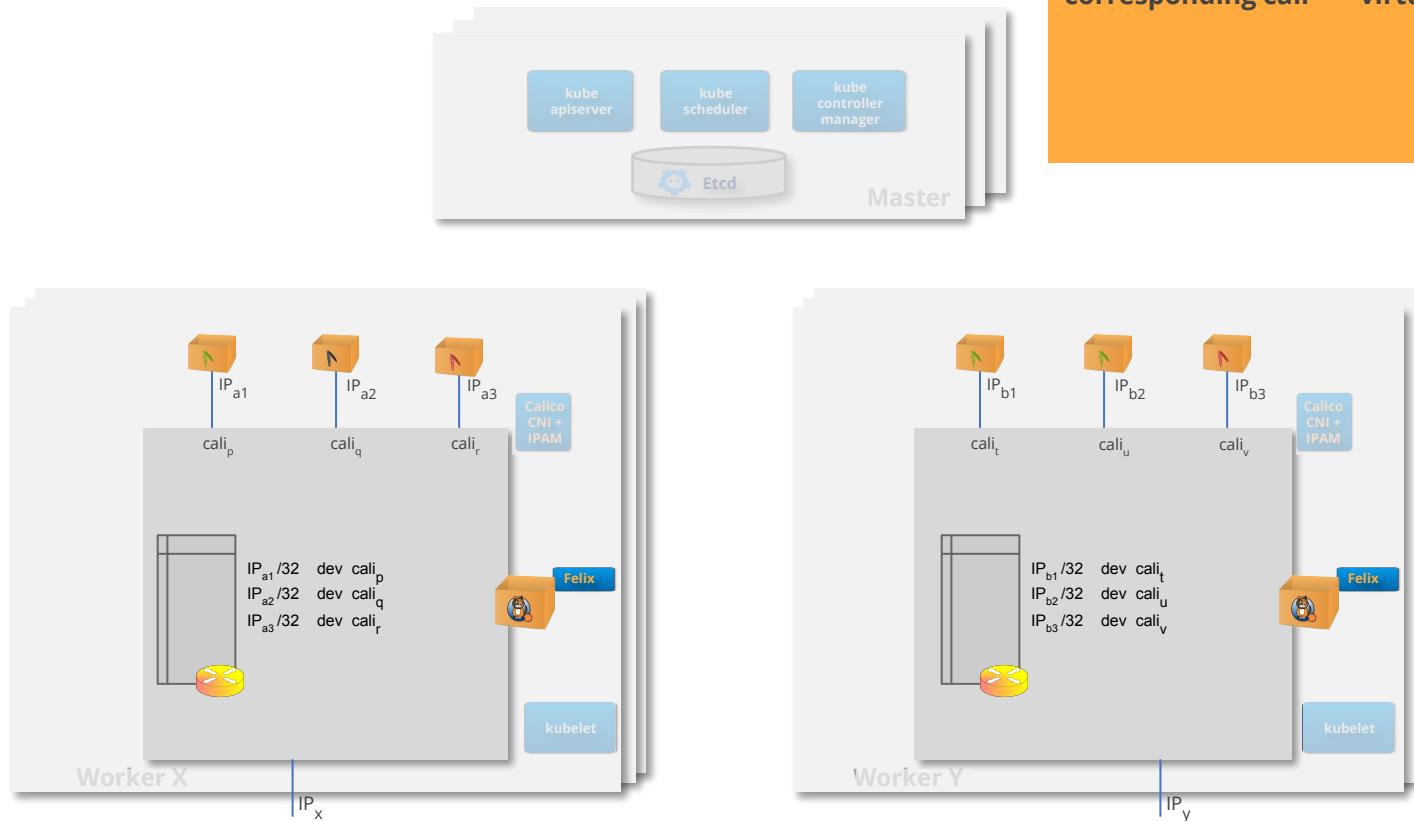


Pod Lifecycle

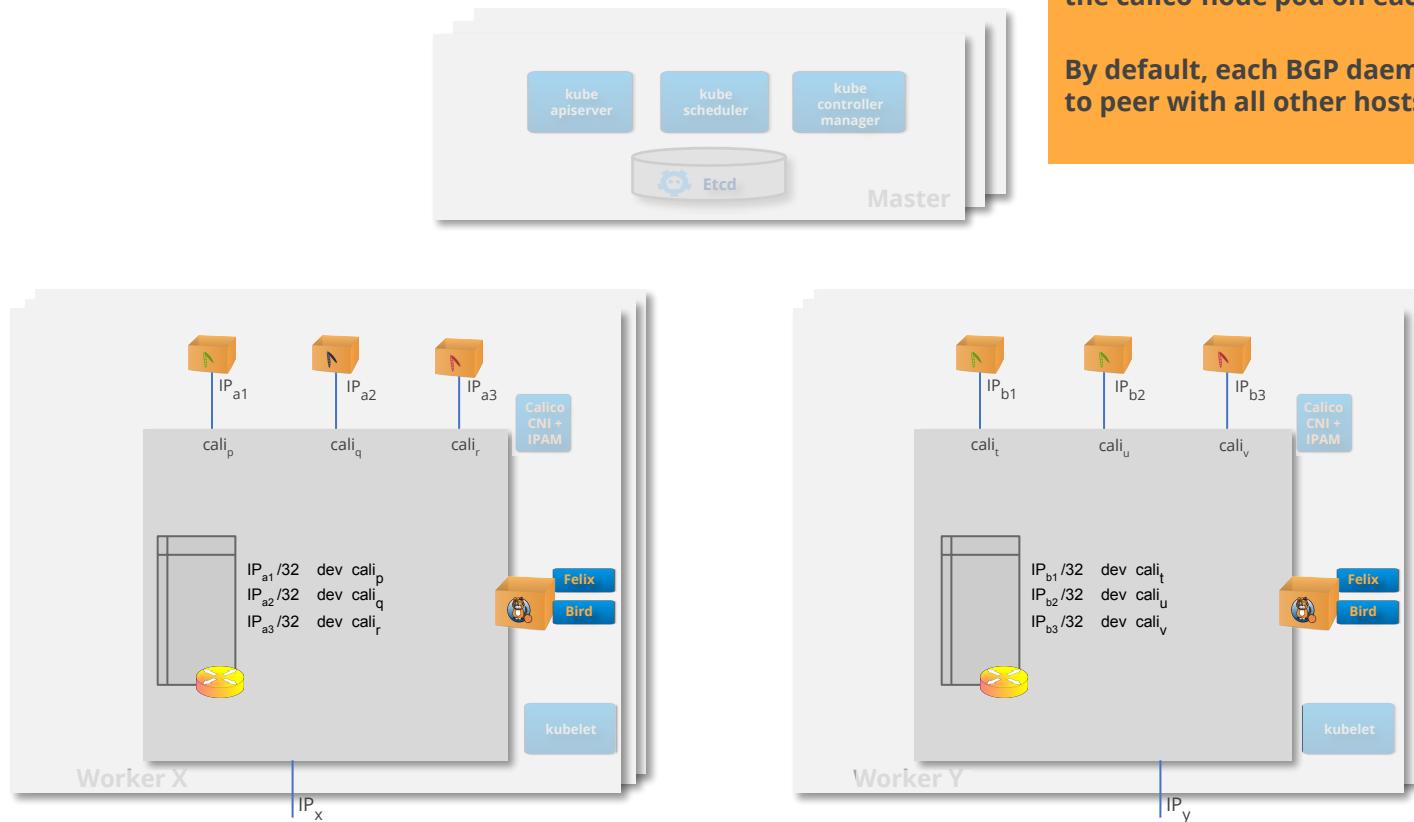


Pod Lifecycle

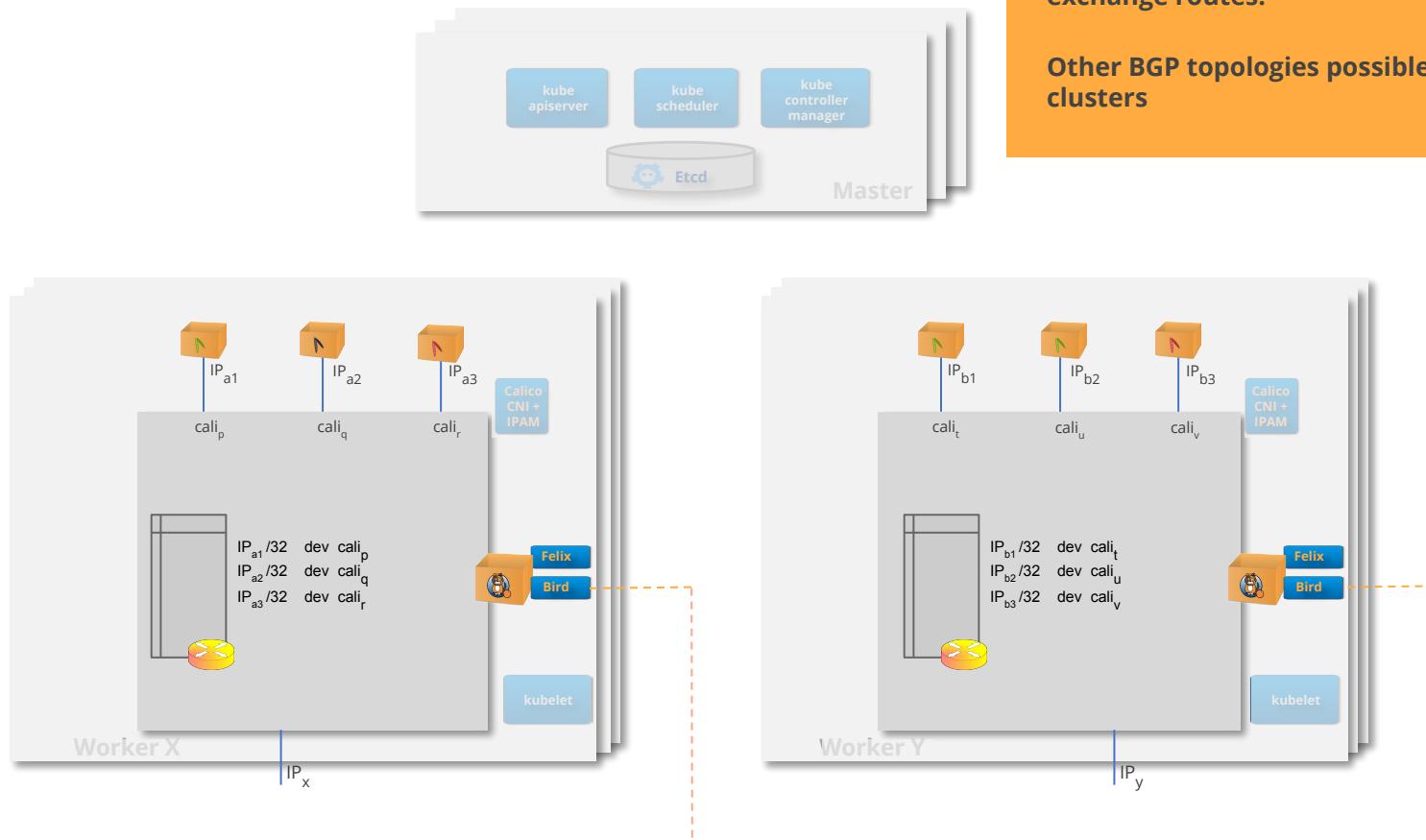
Route added for each pod's IP pointing to corresponding cali*** virtual ethernet



Pod Lifecycle



Pod Lifecycle

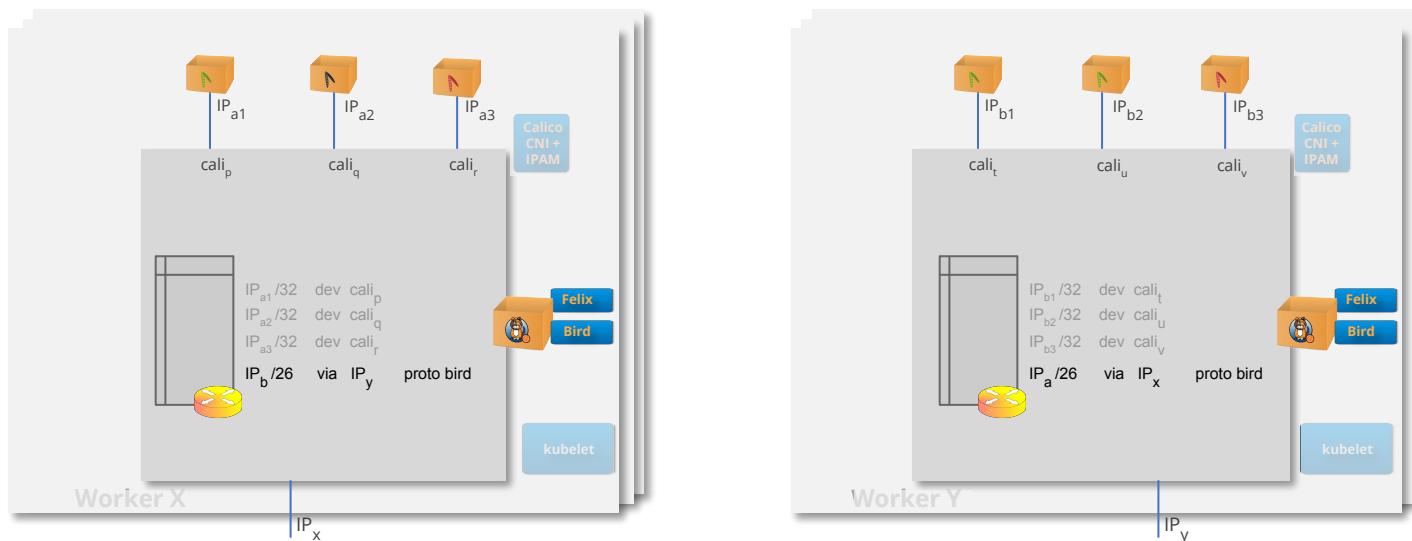


Pod Lifecycle



Routes for Pod IP addresses exchanged among nodes via BGP.

Each Linux node simply routes packets using the default routing table.



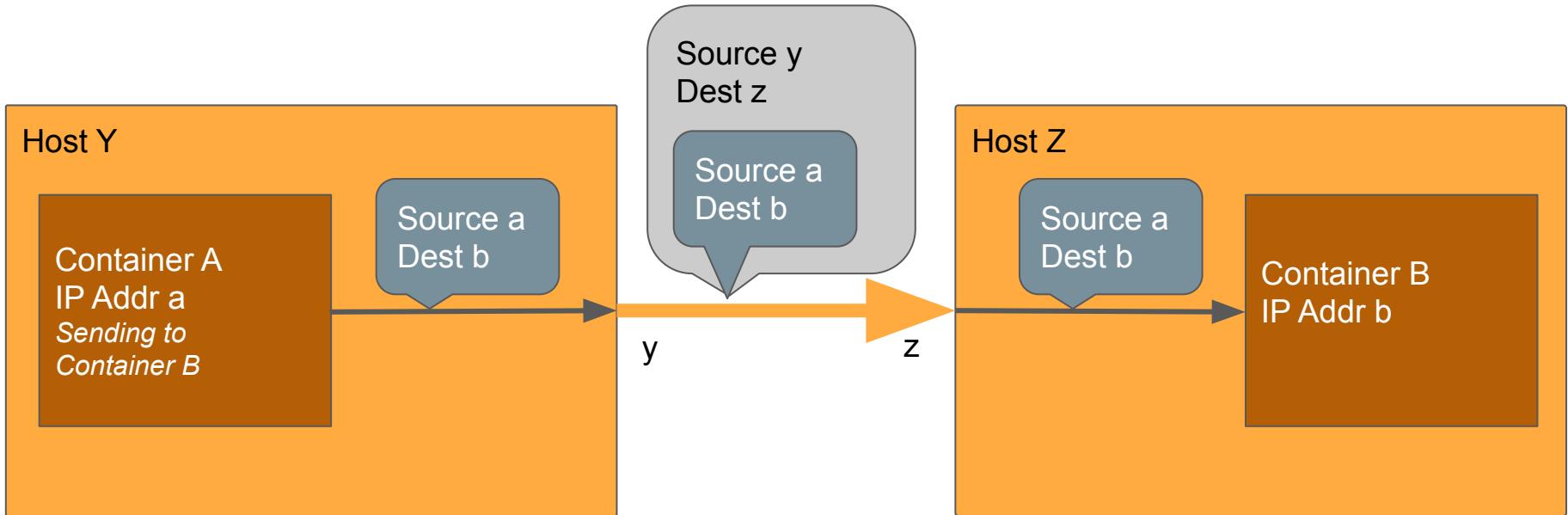
Overlay vs. Flat Network



FLAT NETWORKING VS OVERLAY

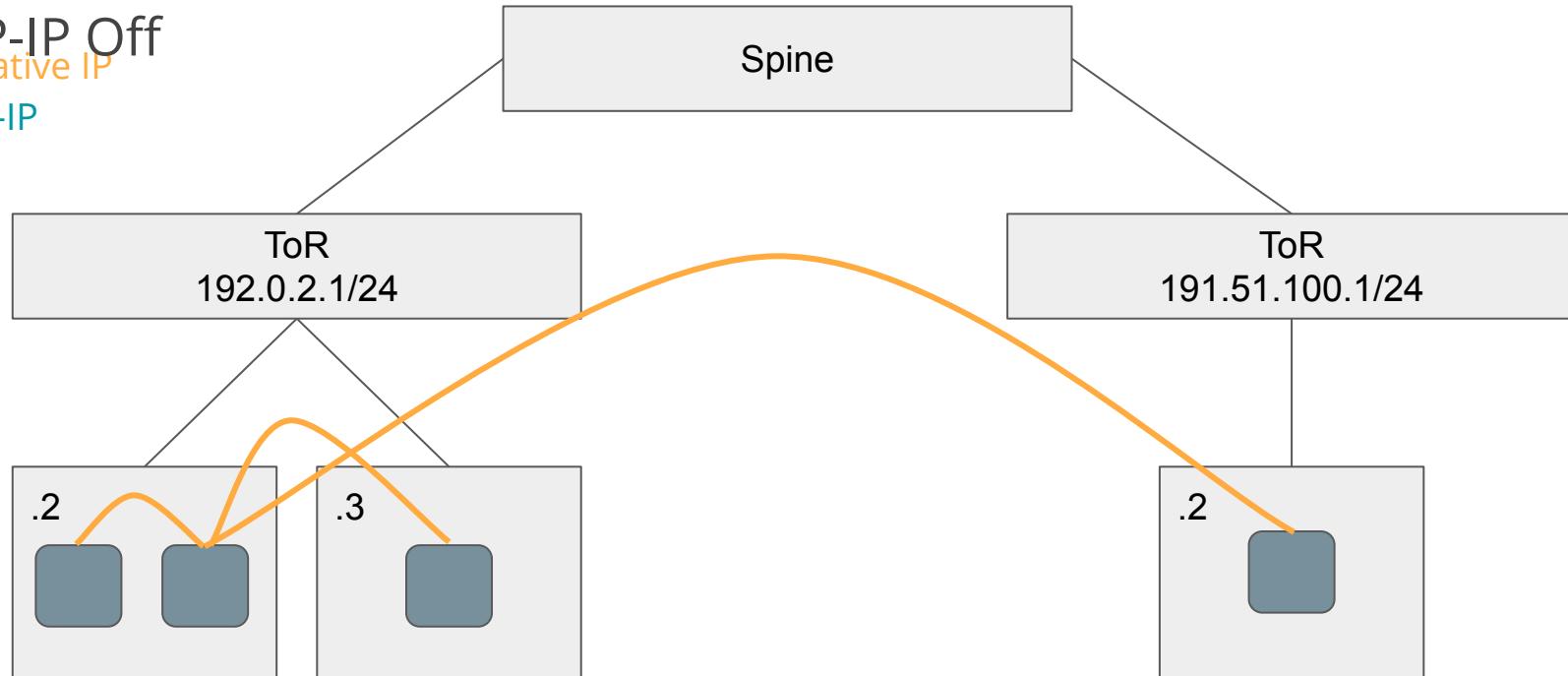
- Docker libnetwork, like most virtual networking, uses overlay technology (VXLAN) to transport packets over the underlying network
 - Advantage: only the host IP address is exposed to the physical network, so only host-to-host IP connectivity is required
 - Disadvantages:
 - Complexity in troubleshooting (have to look inside packet to find actual src/dest addresses)
 - All traffic in/out of the overlay network must be NAT'd and encapsulated/de-encapsulated
 - Processing overhead & bandwidth inefficiency (inc. MTU considerations)
- Calico can operate in two modes: Overlay or Non-Overlay (Flat)
 - Overlay technology used is “IP-in-IP” (or IPIP)
 - No concept of a “virtual network” - endpoint reachability is separate from network isolation
 - Docker EE ships with IPIP **enabled by default**

IP-IN-IP VISUALIZED



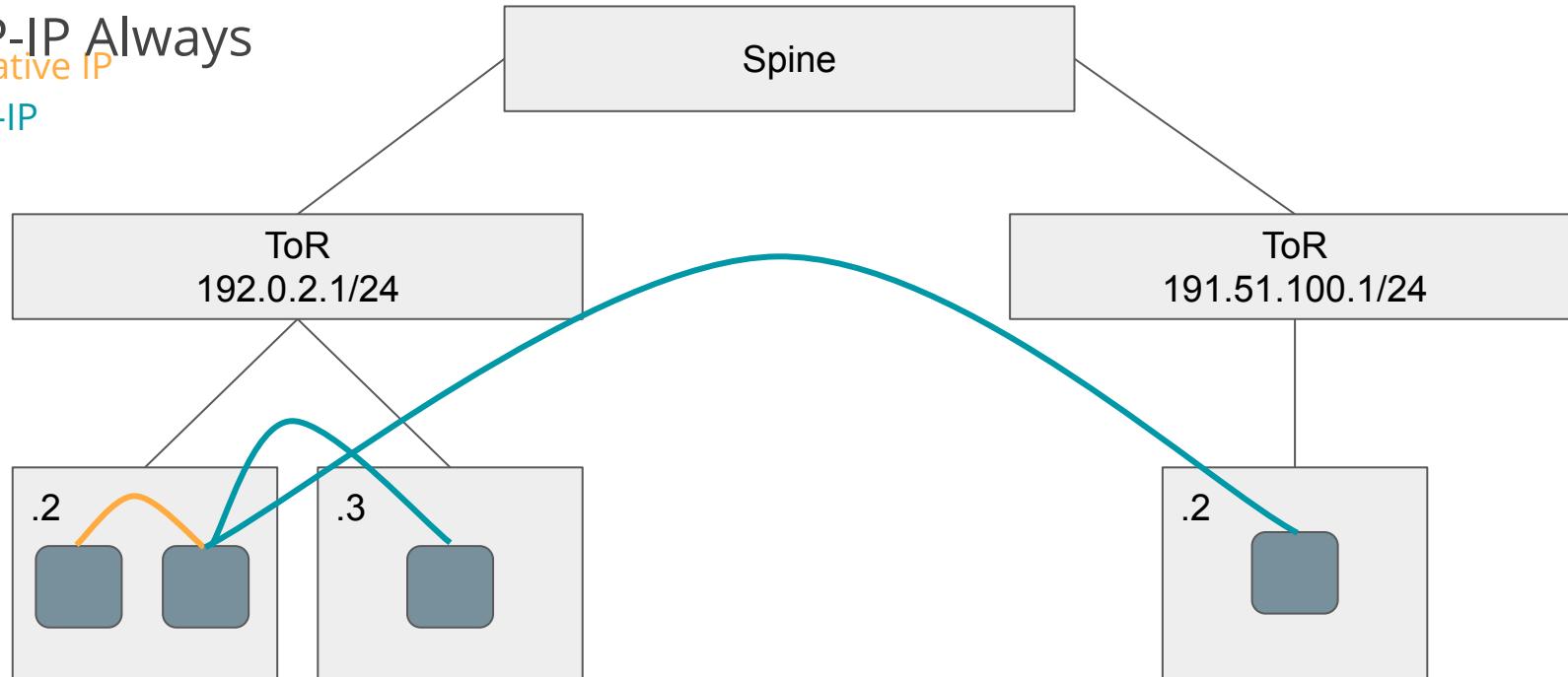
CALICO IP-IN-IP OPTIONS

IP-IP Off
Native IP
IP-IP



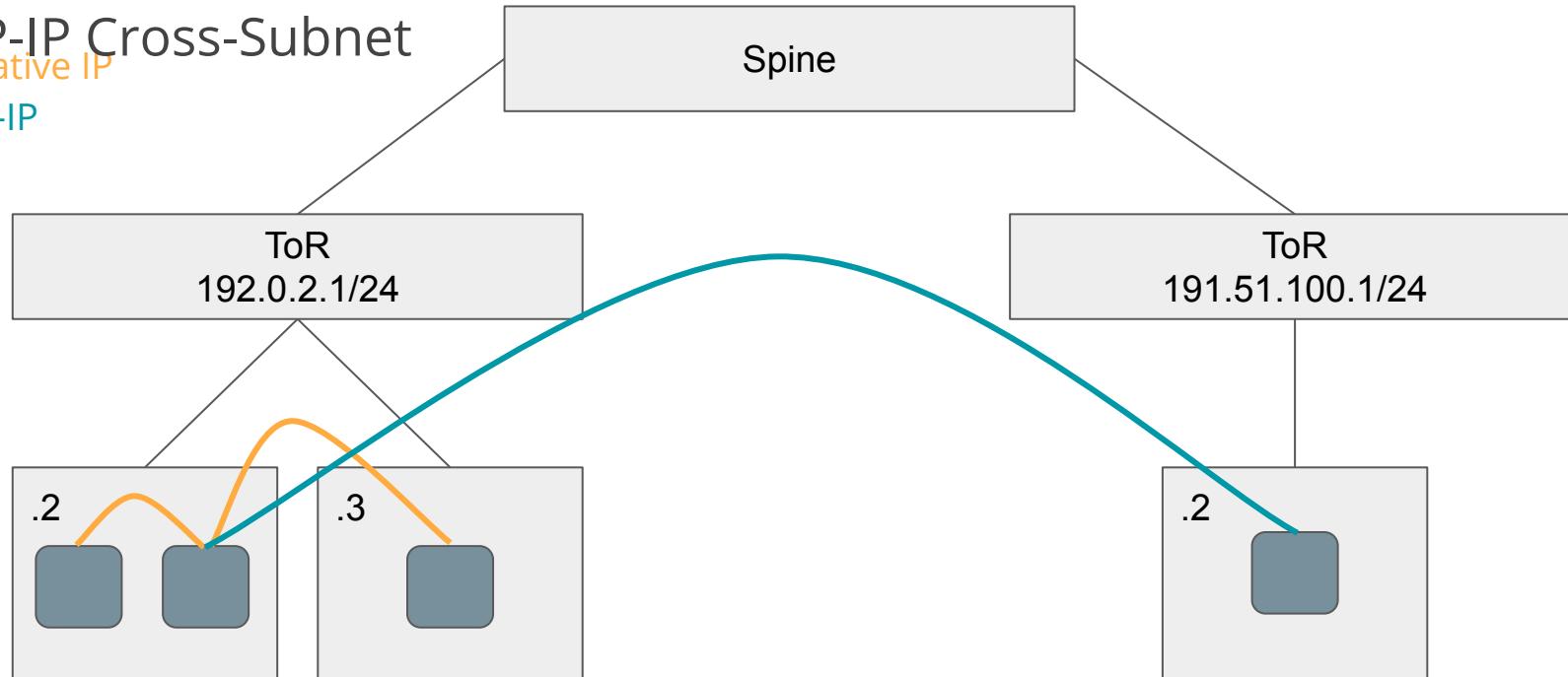
CALICO IP-IN-IP OPTIONS

IP-IP Always
Native IP
IP-IP



CALICO IP-IN-IP OPTIONS

IP-IP Cross-Subnet
Native IP
IP-IP



IP Pools and the Calico CNI



IP Pool Definition

DMZ IP Pool yaml

```
apiVersion: projectcalico.org/v3
kind: IPPool
metadata:
  name: dmz-pool
spec:
  cidr: 10.0.1.0/24
  ipipMode: Always
  natOutgoing: true
  nodeSelector: type: "dmz-node"
```

This will define an IP Pool named dmz-pool.

When running multiple subnets for pods it's usually a good idea to set natOutgoing to true and IPIP mode to always.

nodeSelector is one way to specify IP pool use per topology

Using IP Pools for Namespaces and workloads

Namespace definition yaml

```
apiVersion: v1
kind: Namespace
metadata:
  name: dmz
  labels:
    location: dmz
  annotations:
    "cni.projectcalico.org/ipv4pools":
    "[\"dmz-pool\"]"
```

To assign an IP Pool to a namespace add and annotation to the namespace definition referencing the IP Pools name.

Add this to your namespace or workload yamls
annotations: "cni.projectcalico.org/ipv4pools": "["dmz-pool"]"



Do Labs 1 & 2

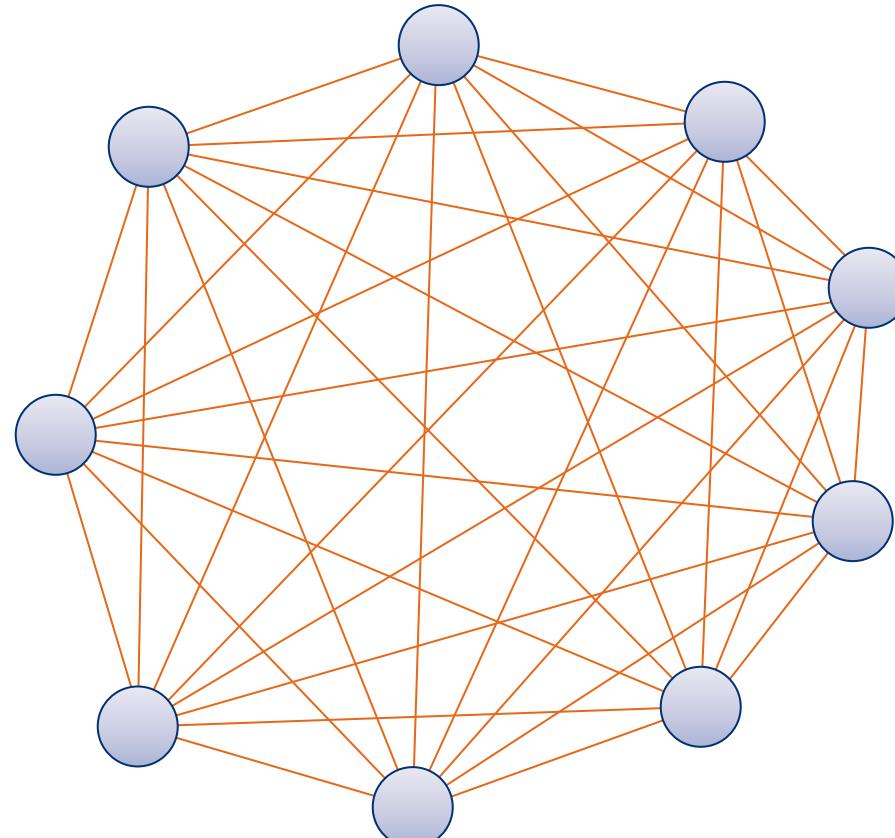


Declarative Network Policy



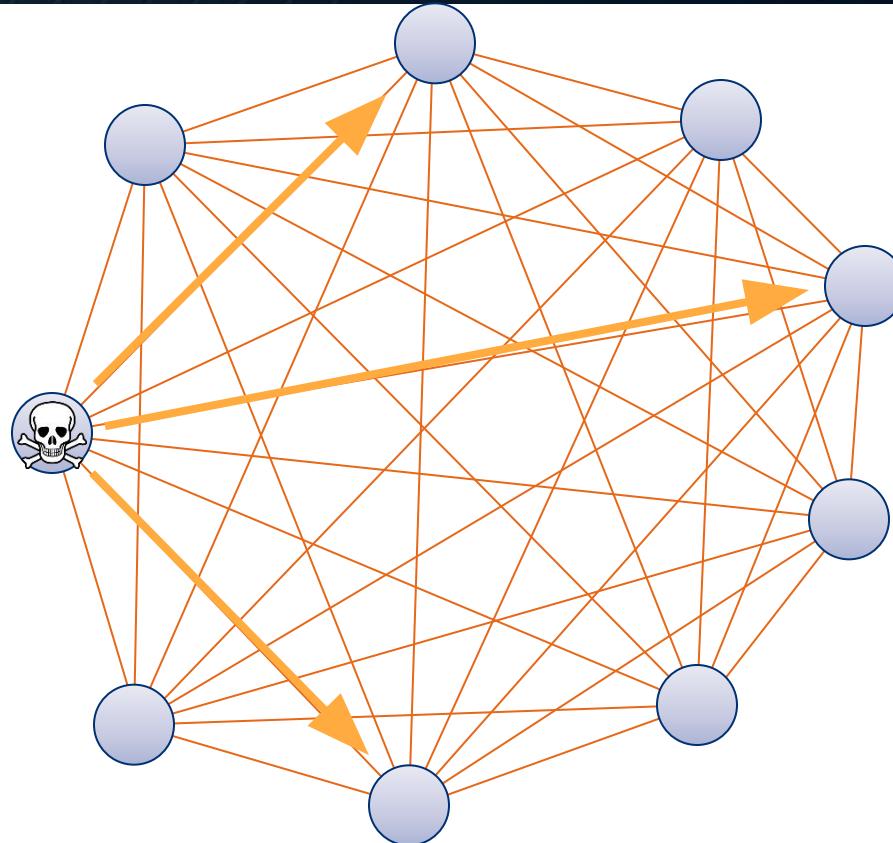
Motivation For Network Policy

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.



Pathways for Attack

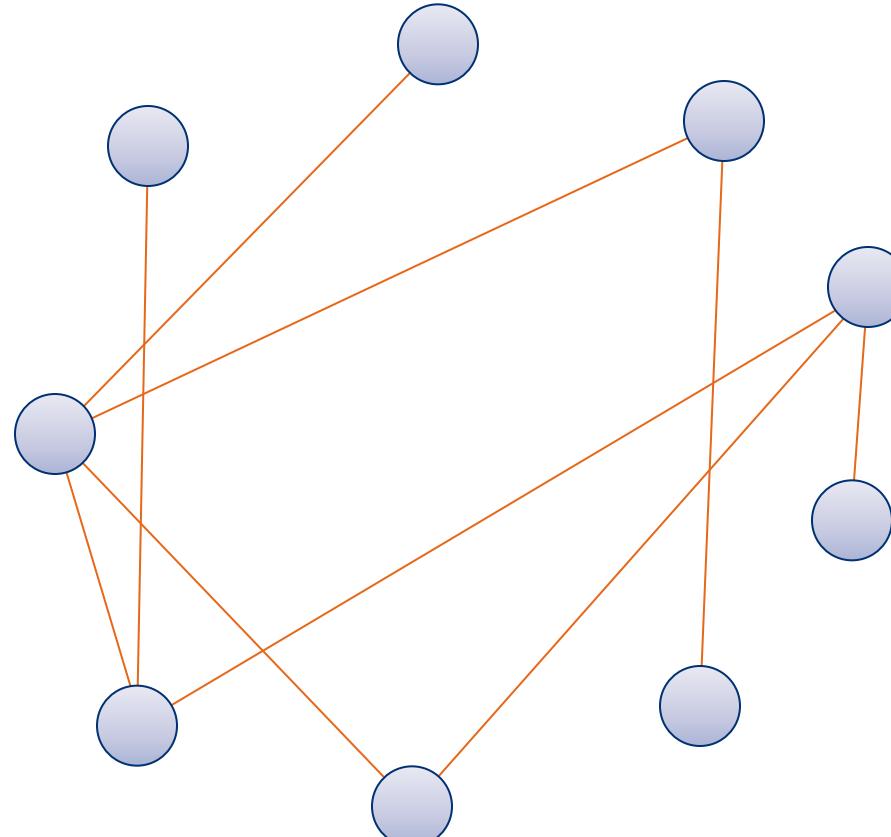
The rest are only useful for an attacker.



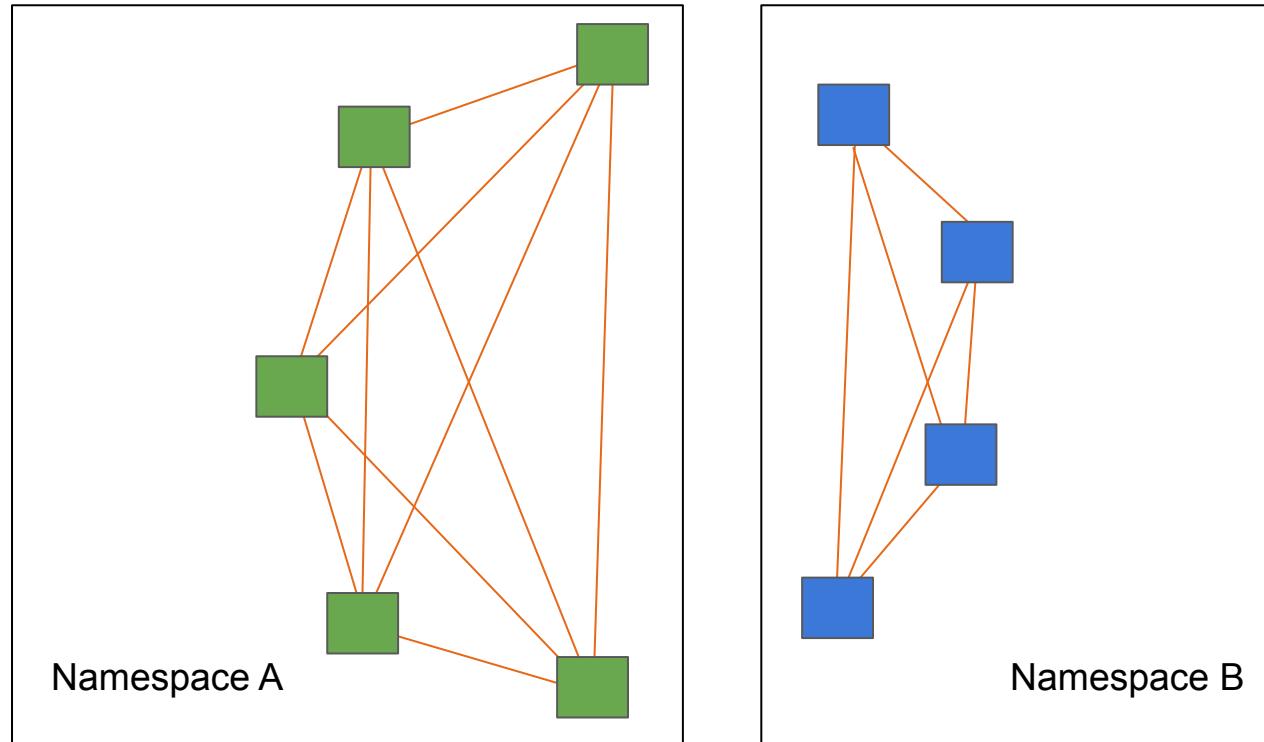
Identifying Unused Connections

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.

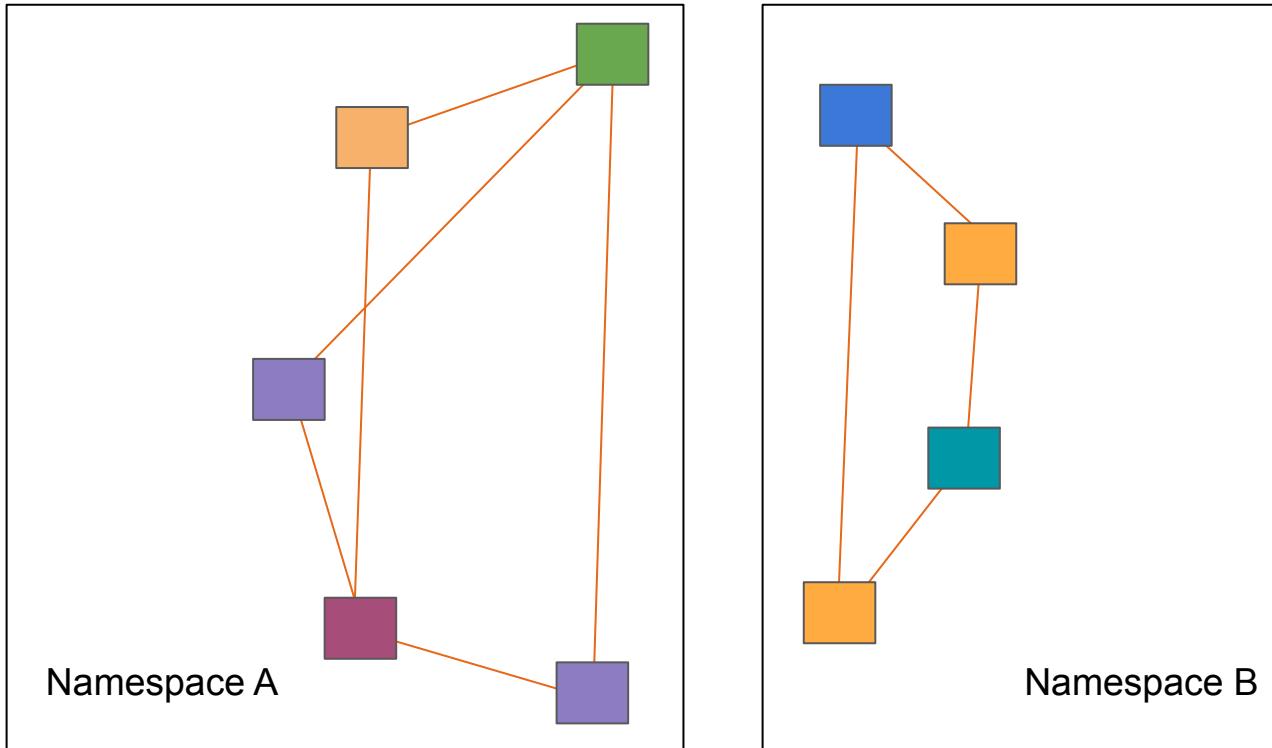
Each connection path that we can eliminate will reduce complexity and improve security.



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

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

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

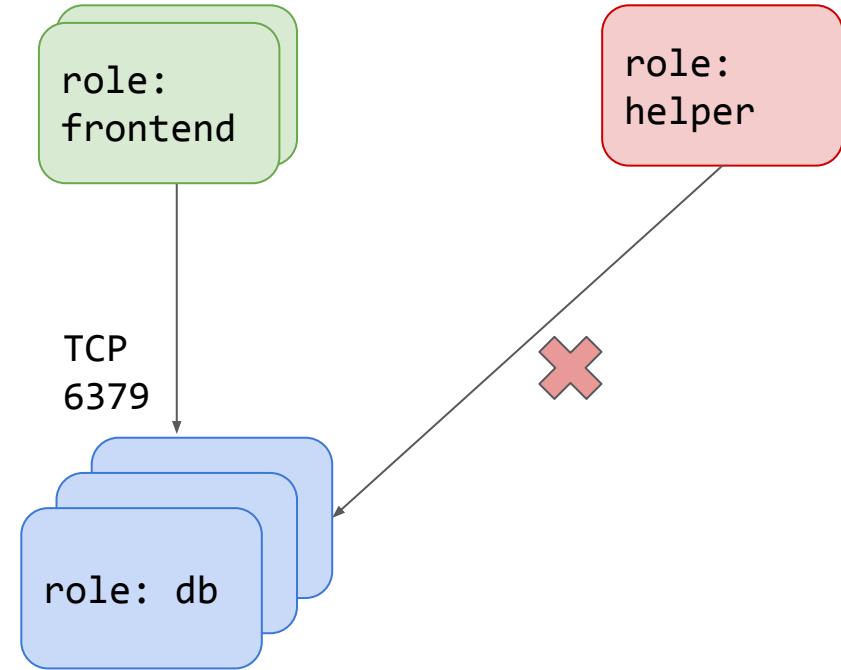
Kubernetes NetworkPolicy Resource

- 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/>

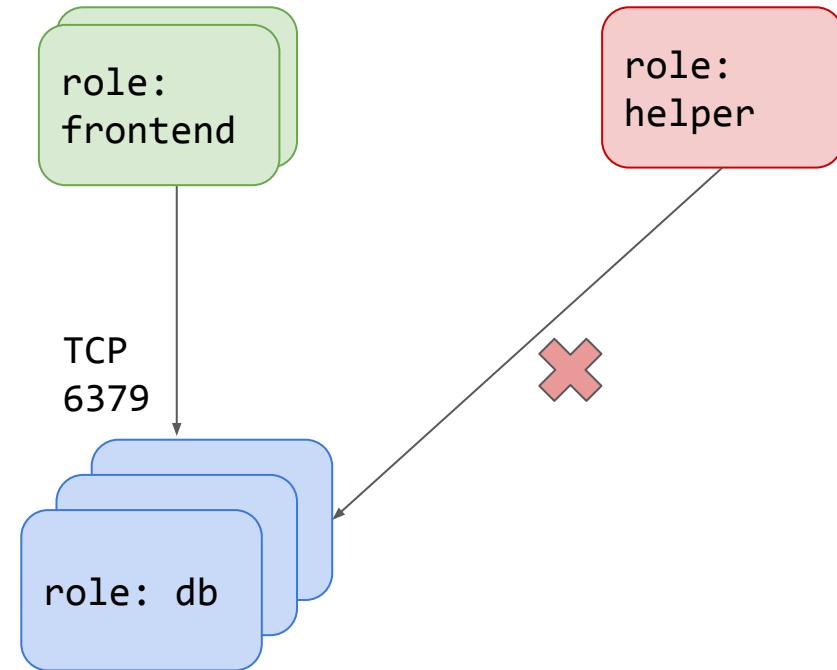
Kubernetes Network Policy Example

```
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
```



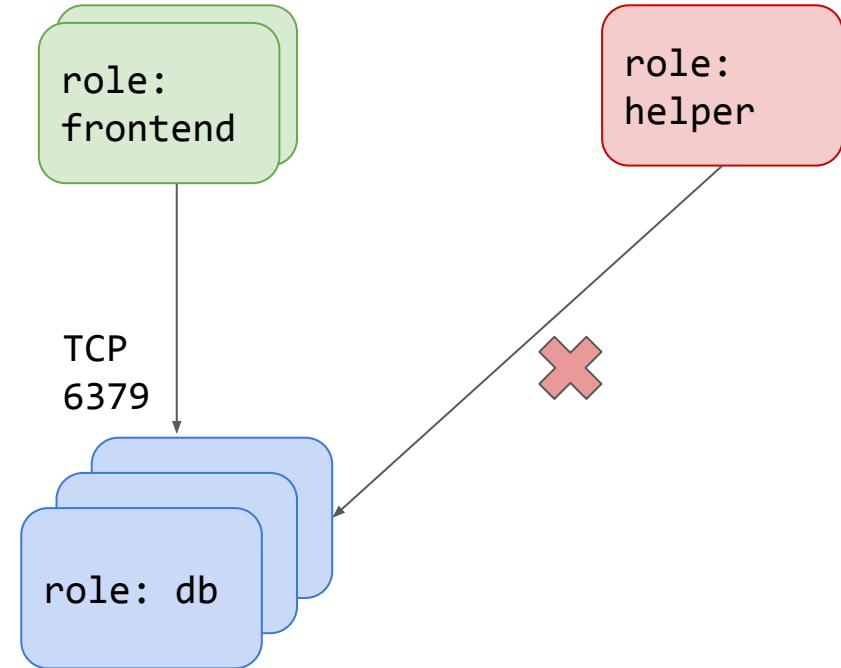
Kubernetes Network Policy Example

```
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
```



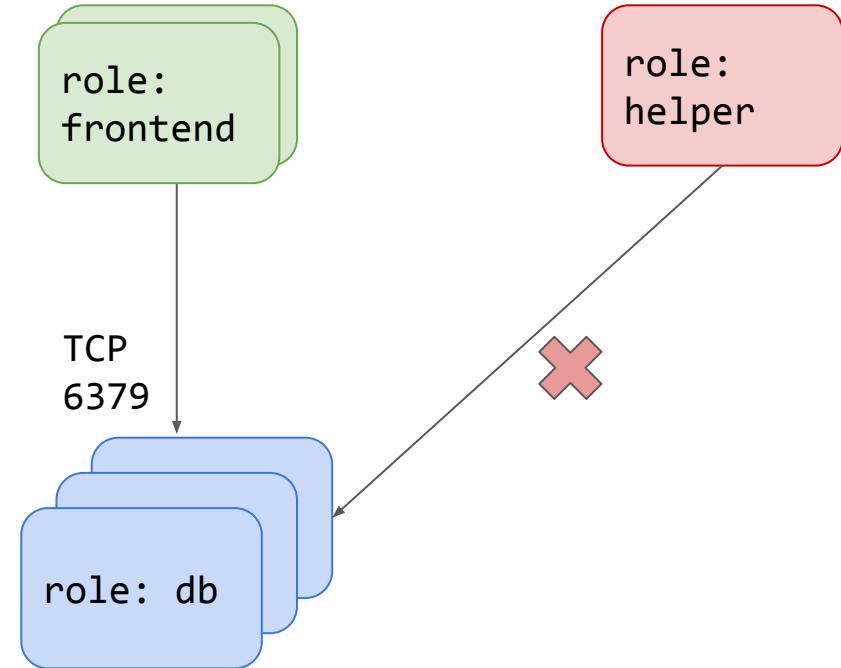
Kubernetes Network Policy Example

```
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
```



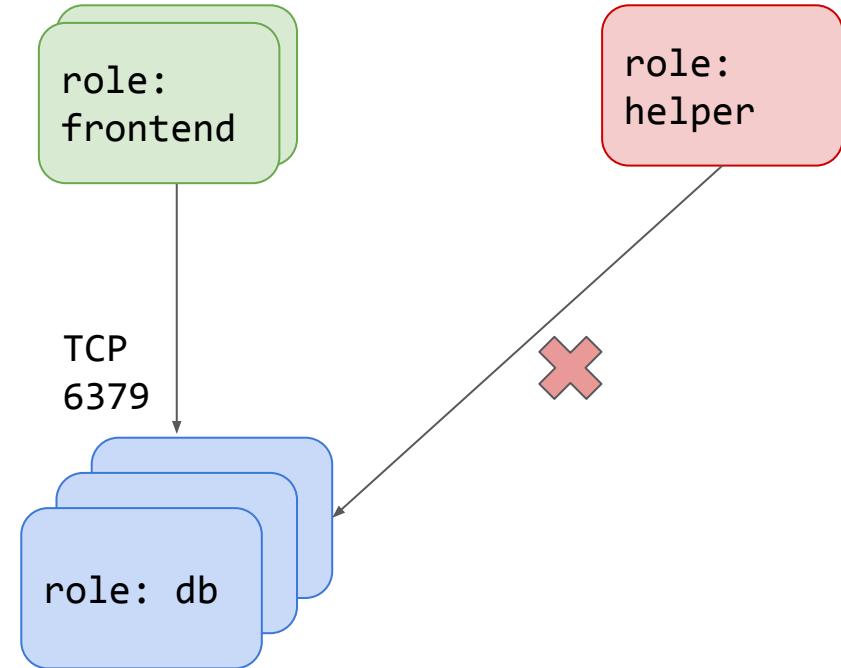
Kubernetes Network Policy Example

```
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
```

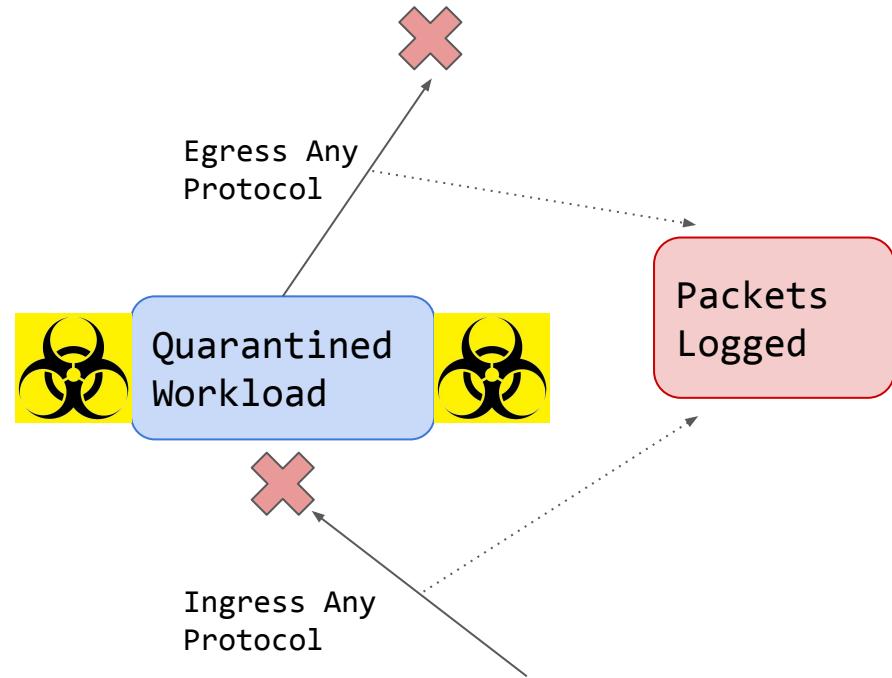


Advanced use cases beyond Kubernetes Network Policy

- **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>

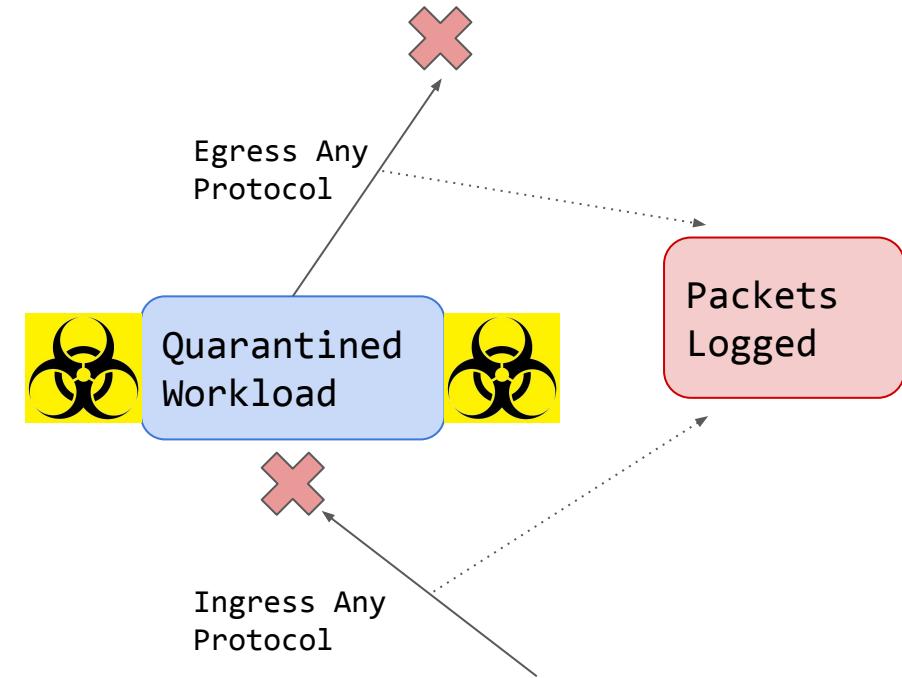
Calico Network Policy Example

```
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
```



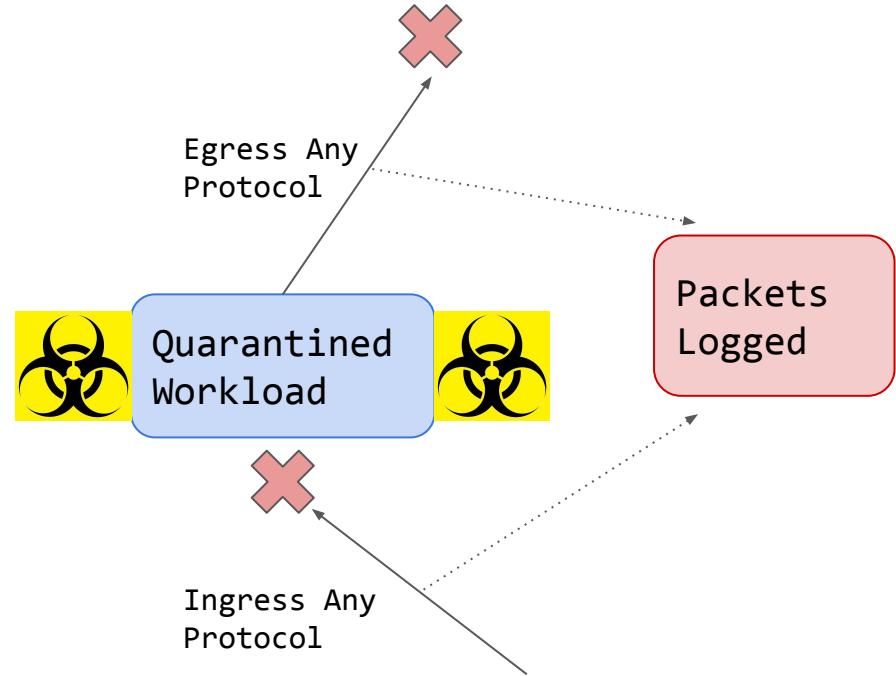
Calico Network Policy Example

```
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
```



Calicectl: The Calico Command Line

- Calicectl is the command line utility for advanced Calico configuration
- Download: <https://github.com/projectcalico/calicectl/releases>
- Reference: <https://docs.projectcalico.org/latest/reference/calicectl/>
- Already installed on your training servers



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

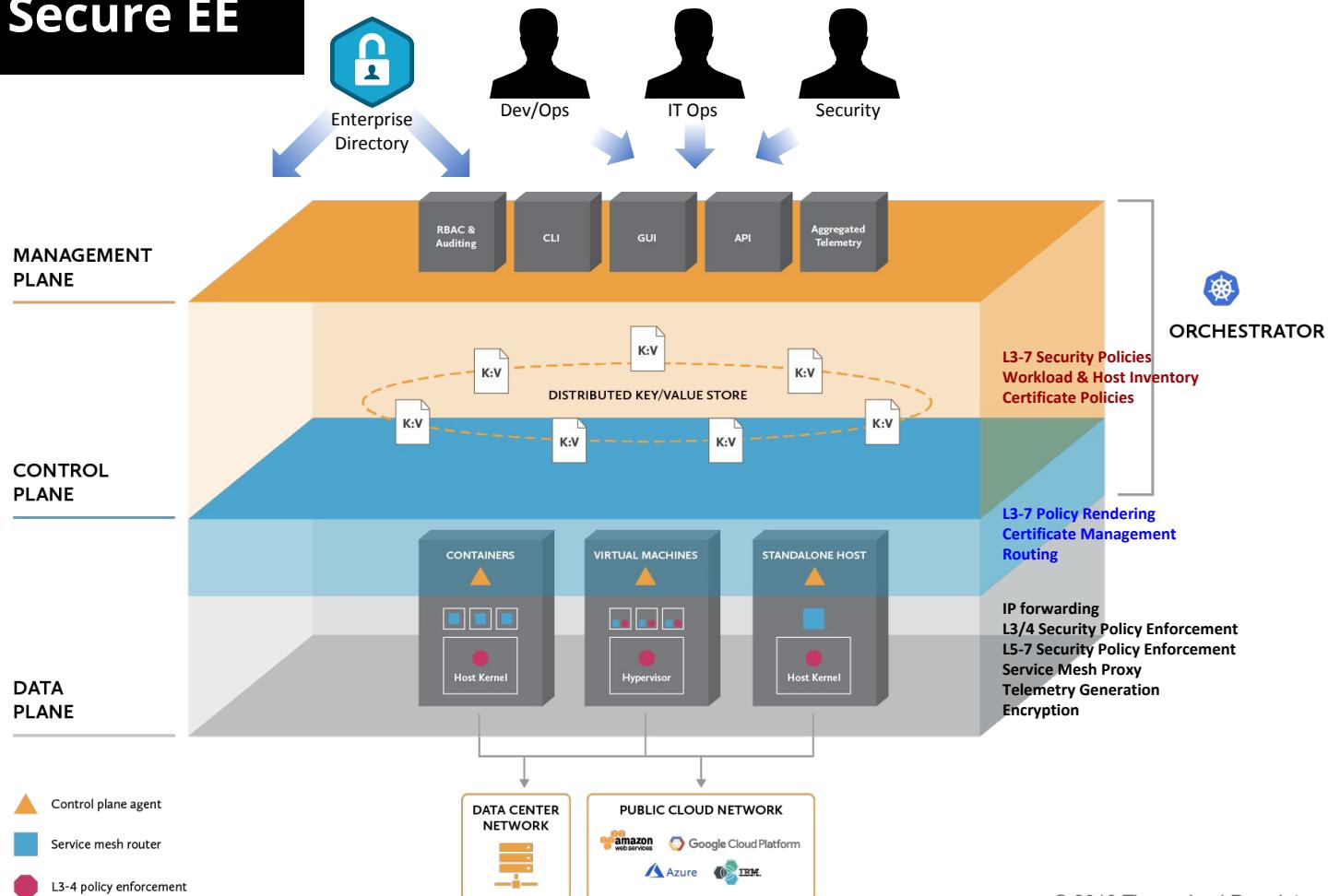
Do Lab 3



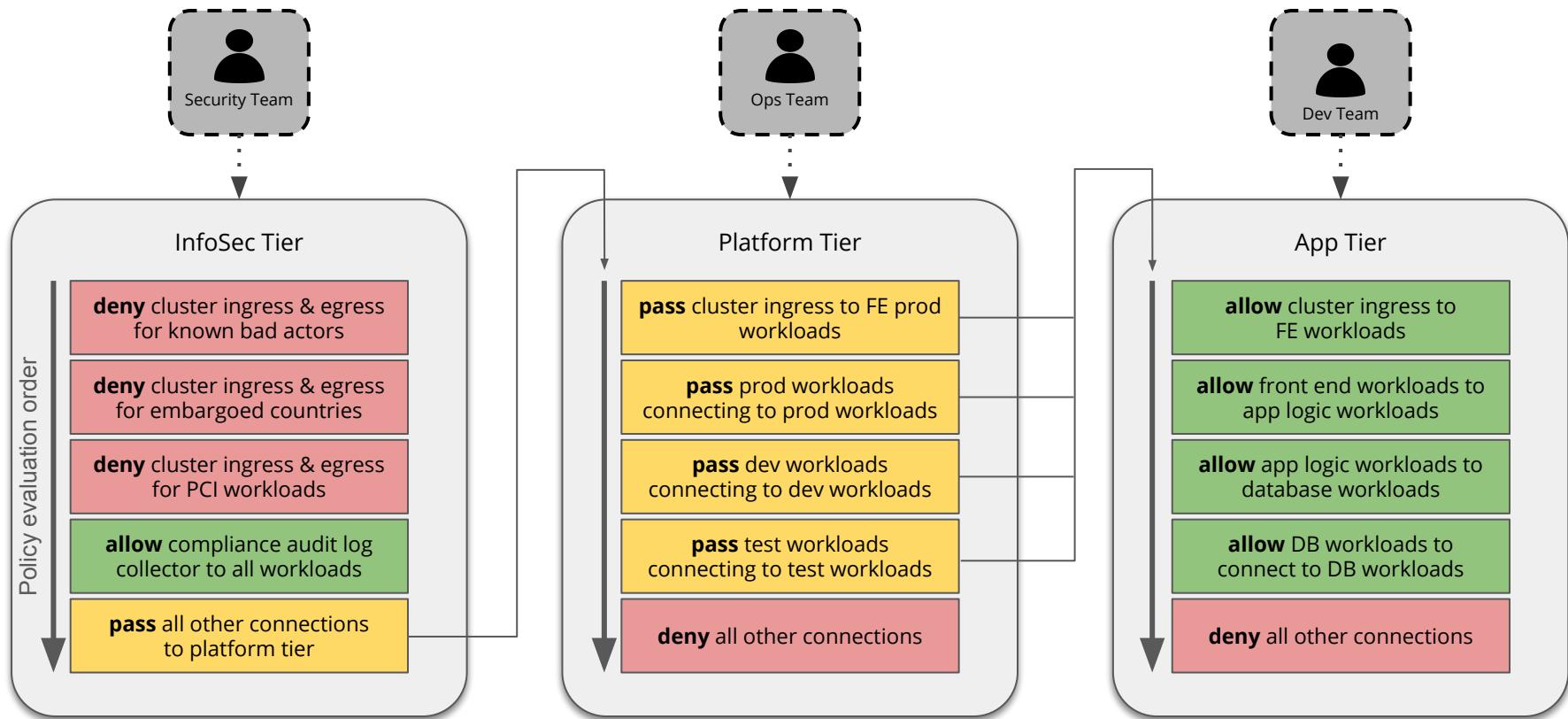
Anatomy of TSEE



Tigera Secure EE



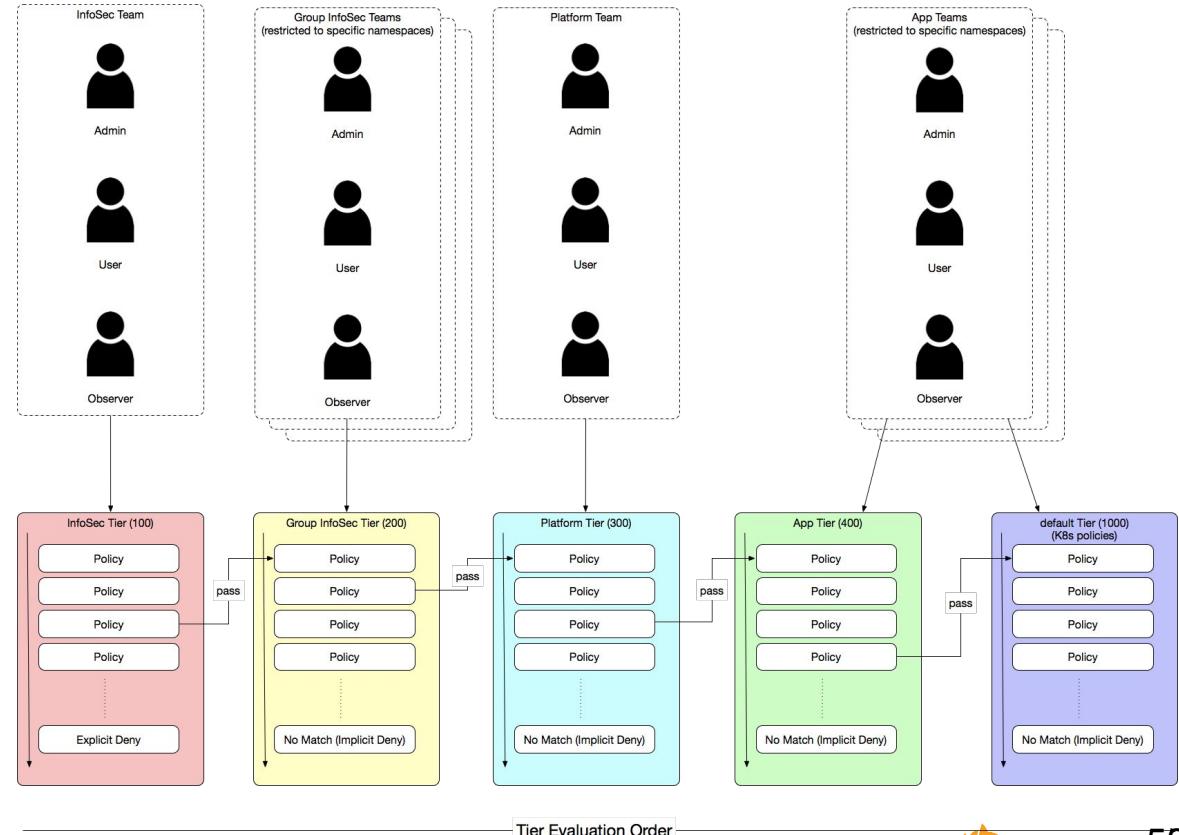
CONTINUOUS COMPLIANCE HIERARCHICAL POLICIES AND TIERS



Objective: Organizational Control in Policy Management

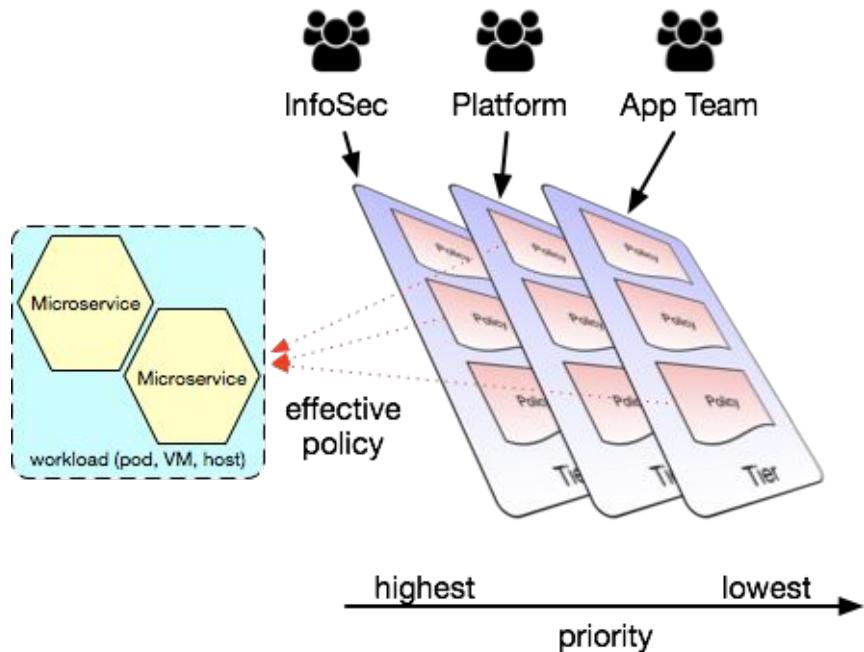
Policy precedence is a requirement.

Namespaces constrain access to tenant specific policies.



TSEE Policy Tiers

TSEE enables policies to be specified and evaluated in tiers.



Each tier is a sequence of policies that are executed in order.

RBAC can be used to control access to tiers and tenants (by using namespaces and controlling access via resource type and namespace).

TSEE Policy Tier Deployment Example

The screenshot displays the Tigera Policies Board interface, which is a web-based tool for managing network policies across various Kubernetes clusters. The interface is organized into several sections, each representing a different policy tier or category:

- peira**: Contains policies for `mock-allow-proberesult`, `mockservice-allow-dns`, `mockservice-egress`, `peira-control-plane-egress`, and `peira-default-pass`. These policies define allowed and denied traffic flows between global endpoints.
- allow-cnx**: Contains policies for `cni-apiserver-access` and `cni-manager-access`. These policies allow traffic from specific IP ranges to the kube-system namespace.
- security-operations**: Contains policies for `outbound-remediation` and `inbound-logging`. These policies are used for monitoring and remediation purposes.
- operations**: Contains policies for `inbound-logging` and `outbound-logging`. These policies log network traffic to help with troubleshooting and audit.
- infosec**: Contains policies for `eu-none-block`, `us-nous-block`, and `infosec-implicit-pass`. These policies implement security rules like blocking traffic from specific regions or allowing implicit traffic between pods.
- default**: Contains policies for `backend-from-frontend`, `onprem-db-from-onprem-be`, `cloud-db-from-cloud-be`, `eu-cloud-db-from-onprem-vm-be`, and `calico-node-alertmanager`, `calico-monitoring`. These policies provide a baseline for network access control.

Each policy entry includes fields for `packets/sec` and `bytes/sec`, indicating the rate of traffic being processed by the policy. The interface also features a sidebar with navigation links and a bottom bar with a "ADD NEW TIER" button.

Building Policies



Key Design Considerations

- Zero trust = “default deny” = all (ingress / egress) traffic is whitelisted
- Calico policy (recommended) vs K8s network policy
- L3-L7 rules
- One policy for each service
- Policy workflow automation (more on this in CI/CD section)

Label, Label, Label

- Semantic attributes of your endpoint, namespace and serviceaccount
- Collaborate and converge to a standard
- Audit (and/or enforce) your deployments for alignment with the standard
- We recommend: *application, service (producer/consumer, or similar), version, stage, release, owner (TBD add example)*
- Additional Attributes for consideration
 - BU, organization, cost center, security level, compliance status
 - Review the CMDB (if you have one)

Namespace vs Global

Scope	Namespace	GlobalNetworkPolicy
Matches on	Endpoints within a specific namespace	Endpoints across all namespaces
When to use	Individual microservices	<ul style="list-style-type: none">- Common services (logging, build systems, DNS, SMTP relay, SSH etc)- Security and Compliance controls (PCI rules, export compliance, bad actor blacklist)- Incident response workflows (quarantine, forensics)
Tier	Typically in lower tiers.	Higher Tiers for priority evaluation
Label selector for rules	<ul style="list-style-type: none">- Endpoints (within the namespace)- Namespace and SvcAccounts (cross namespace)	<ul style="list-style-type: none">- Namespace & ServiceAccount (be cautious with endpoint selector)- Explore Admission controller for governance

Global network policy (PCI Example)

Applies To

Match all

[+ ADD LABEL SELECTOR](#)

Options

Use Kubernetes network policy

Type

Ingress Egress

Ingress Policy Rules (3)

Deny Any Protocol **From:** Service account: Selectors [PCI != true] **To:** Service account: Selectors [PCI = true]



Deny Any Protocol **From:** Service account: Selectors [PCI !Exist] **To:** Service account: Selectors [PCI = true]



Pass Any Protocol



[+ ADD INGRESS RULE](#)

Egress Policy Rules (3)

Deny Any Protocol **From:** Service account: Selectors [PCI = true] **To:** Service account: Selectors [PCI != true]



Deny Any Protocol **From:** Service account: Selectors [PCI = true] **To:** Service account: Selectors [PCI !Exist]



Pass Any Protocol



[+ ADD EGRESS RULE](#)

Deny all ingress to
PCI workloads from
non-PCI workloads

Deny all egress
from PCI workloads
to non-PCI
workloads

Namespace scope policy

Applies To

Policy Label selector: [app = backend]

Options

Use Kubernetes network policy

Type

Ingress Egress

Ingress Policy Rules (3)

Allow Protocol is TCP **From:** Endpoints [app = microservice1] **To:** Ports [Port is 8080]

Endpoints	1↓	0↑	Connections/sec	0↓	0↑
Allowed	packets/sec: 1.6↓	0.8↑	bytes/sec: 86.4 b↓	44.8 b↑	Denied
Passed	packets/sec: 0↓	0↑	bytes/sec: 0 b↓	0 b↑	

Allow Protocol is TCP **From:** Endpoints [app = microservice2] **To:** Ports [Port is 8080]

Endpoints	1↓	0↑	Connections/sec	0↓	0↑
Allowed	packets/sec: 1.6↓	0.8↑	bytes/sec: 86.4 b↓	44.8 b↑	Denied
Passed	packets/sec: 0↓	0↑	bytes/sec: 0 b↓	0 b↑	

Deny Any Protocol

Endpoints	0↓	0↑	Connections/sec	0↓	0↑
Allowed	packets/sec: 0↓	0↑	bytes/sec: 0 b↓	0 b↑	Denied
Passed	packets/sec: 0↓	0↑	bytes/sec: 0 b↓	0 b↑	

Egress Policy Rules (1)

Allow Protocol is TCP **To:** Endpoints [app = logging] Ports [Port is 8080]

Endpoints	0↓	1↑	Connections/sec	0↓	0↑
Allowed	packets/sec: 0.4↓	0.8↑	bytes/sec: 22.4 b↓	43.2 b↑	Denied
Passed	packets/sec: 0↓	0↑	bytes/sec: 0 b↓	0 b↑	

Key Takeaways

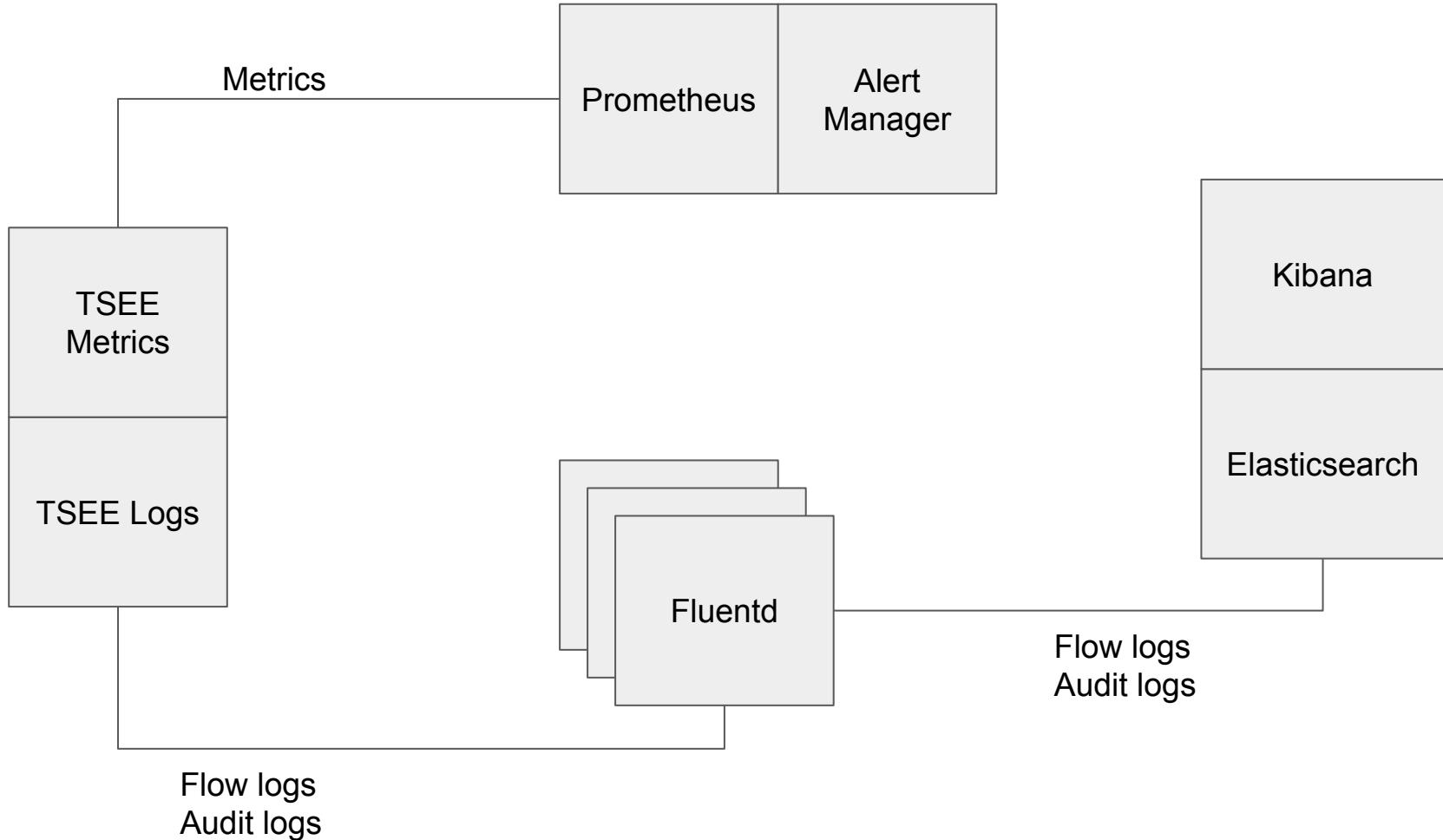
- Zero trust networking and security
- Label design before policy design
- 1 policy per service
- For x-namespace traffic use namespace and serviceaccount selectors

Do Lab 4



Logs, Logging, and Metrics





Fluentd: Tigera-es-config Config-map

```
bikram@MBP-staged:~$ kubectl get configmap tigera-es-config -o yaml
```

```
apiVersion: v1
data:
  tigera.elasticsearch.access-mode: insecure
  tigera.elasticsearch.audit-retention: "367"
  tigera.elasticsearch.cluster-name: cluster
  tigera.elasticsearch.compliance-report-retention: "7"
  tigera.elasticsearch.flow-filtering: "true"
  tigera.elasticsearch.flow-filters.conf: |-  
    <filter flows>  
      @type grep  
      <exclude>  
        key source_namespace  
        pattern (^istio|^kube|^calico)  
      </exclude>  
      <exclude>  
        key dest_namespace  
        pattern (^istio|^kube|^calico)  
      </exclude>  
    </filter>
  tigera.elasticsearch.flow-retention: "8"
  tigera.elasticsearch.flows-index-shards: "5"
  tigera.elasticsearch.host: elasticsearch-tigera-elasticsearch.calico-monitoring.svc.cluster.local
  tigera.elasticsearch.port: "9200"
  tigera.elasticsearch.scheme: http
  tigera.elasticsearch.snapshot-retention: "367"
kind: ConfigMap
```

Excluding flows from certain namespaces

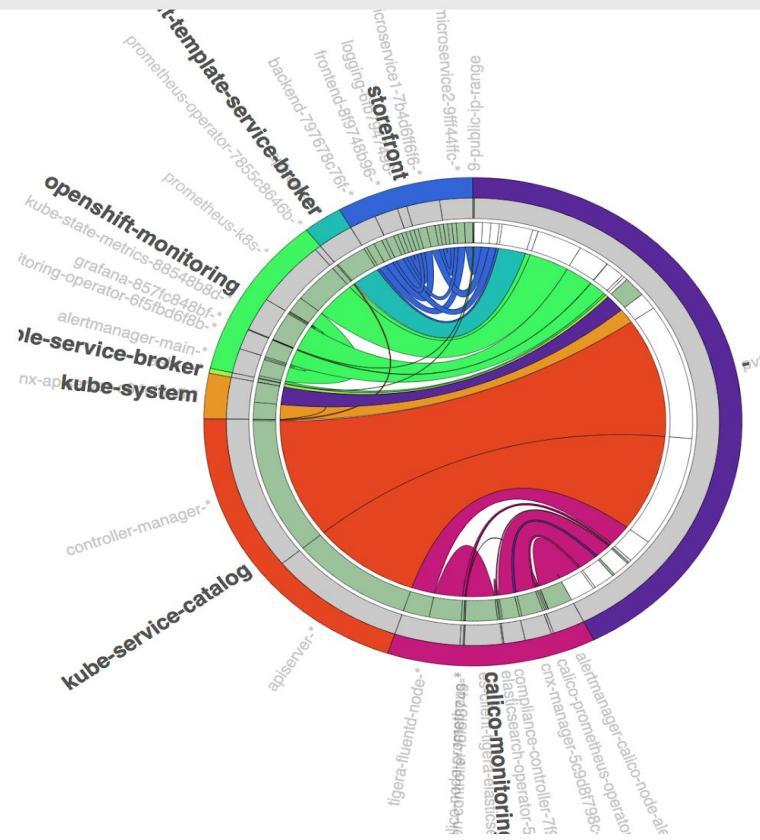
Flow Logs Aggregation and Push Interval

```
birkams-MBP:demo223 birkams$ cat aggregation.yaml
apiVersion: projectcalico.org/v3
kind: FelixConfiguration
metadata:
  name: default
spec:
  flowLogsFileAggregationKindForAllowed: 1
  FlowLogsFlushInterval: 10s
  CloudWatchLogsFlushInterval: 10s
```

Ref: <https://docs.tigera.io/v2.4/reference/calicoctl/resources/felixconfig#spec>

Flow Logs

TIGERA



Namespaces	Names	Status
Any status		x ▾
All Namespaces		x ▾
All Names		x ▾
All Flows		x ▾
Allowed Traffic		
● -	0.036	7.01
● calico-monitoring	0	0.708
● kube-system	0.036	6.30
● calico-monitoring	6.55	37.8
● -	6.12	27.6
● calico-monitoring	0.434	10.2
● kube-service-catalog	2.19	78.0
● -	2.19	77.6
● openshift-ansible-service-broker	0	0.053
● openshift-template-service-broker	0	0.383
● kube-system	0	5.33
● -	0	5.33
● openshift-ansible-service-	0.037	1.20
Denied Traffic		
● -	2.28k	2.28k
● calico-monitoring	208	208
● kube-system	2.07k	2.07k
● calico-monitoring	21.0k	21.0k
● -	7.47k	7.47k
● calico-monitoring	13.6k	13.6k
● kube-service-catalog	72.2k	72.2k
● -	66.4k	66.4k
● openshift-ansible-service-broker	117	117
● openshift-template-service-broker	5.66k	5.66k
● kube-system	946	946
● -	946	946
● openshift-ansible-service-	758	758

Legend: Allowed Denied Denied by remote

Key Elements

- 1.** Metrics & Alerts
- 2.** Logging
- 3.** Dashboards
- 4.** Incident Response Workflows

Metrics

- Denied bytes and packet count (Instance, policy)
- Policy rule bytes, connection, count
 - Action, Traffic Direction, Policy, Rule Direction, Rule Index, Instance, Tier, Namespace

Recommended Alerts

- 1.** Denied packet count
- 2.** Denied packet rate
- 3.** PCI violation (Denied packet count on PCI policy)
- 4.** Export compliance violation (Denied packet count on embargo policy)

Alerts (Example)

DeniedPacketsCount (0 active)

```
alert: DeniedPacketsCount
expr: count_over_time(calico_denied_packets{policy!="security|quarantine|1|deny"}[5s])
  > 0
labels:
  severity: critical
annotations:
  description: '$labels.instance is denying packets by policy $labels.policy.'
  summary: Policy $labels.policy denying packets
```

DeniedPacketsRate (0 active)

```
alert: DeniedPacketsRate
expr: rate(calico_denied_packets[10s])
  > 50
labels:
  severity: critical
annotations:
  description: '$labels.instance with calico-node pod $labels.pod has been
    denying packets at a fast rate $labels.sourceIp by policy $labels.policy.'
  summary: Instance $labels.instance - Large rate of packets denied
```

Alert Manager (Slack, PagerDuty or your favorite alerting tool)

 AlertManager APP 9:45 AM

Source IP: 192.168.235.178

- Policy default|dev/microservice-3|0|deny denying packets @channel
- worker0 is denying packets by policy default|dev/microservice-3|0|deny.

 pagerduty tsee2.pagerduty.com

Hello Nagi Vumma, you have one open incident assigned to you:

INCIDENT #7

[[FIRING:3](#)] [calico-node-metrics \(DeniedPacketsRate calico-metrics-port kube-system default|demo1/frontendl-1|deny calico-node-metrics critical\)](#)

[View Incident](#)

DETAILS

firing: Labels:

- alertname = DeniedPacketsRate
- endpoint = calico-metrics-port
- instance = node0
- job = calico-node-metrics
- namespace = kube-system
- pod = calico-node-2dj97
- policy = default|demo1/frontendl-1|deny
- service = calico-node-metrics
- severity = critical
- srclP = 192.168.135.145

Annotations:

- description = node0 with calico-node pod calico-node-2dj97 has been denying packets at a fast rate by policy default|demo1/frontendl-1|deny.
- summary = Instance node0...

STATUS

Triggered

URGENCY

↑ High

Audit Logs

Timeline

Filter: none

22 May, 2019

Network set threatfeed.feodo-tracker updated
02:14:15 **system:serviceaccount:calico-monitoring:intrusion-detection-controller**

21 May, 2019

Network set threatfeed.feodo-tracker updated
02:14:15 **system:serviceaccount:calico-monitoring:intrusion-detection-controller**

20 May, 2019

Network set threatfeed.feodo-tracker updated
02:14:15 **system:serviceaccount:calico-monitoring:intrusion-detection-controller**

19 May, 2019

Network set threatfeed.feodo-tracker updated
02:14:15 **system:serviceaccount:calico-monitoring:intrusion-detection-controller**

18 May, 2019

Network set threatfeed.feodo-tracker updated
02:14:15 **system:serviceaccount:calico-monitoring:intrusion-detection-controller**

17 May, 2019

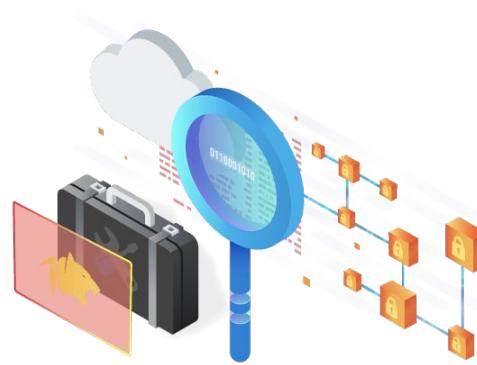
Network set threatfeed.feodo-tracker updated
02:14:15 **system:serviceaccount:calico-monitoring:intrusion-detection-controller**

Tigera Secure Enables Key Security Capabilities



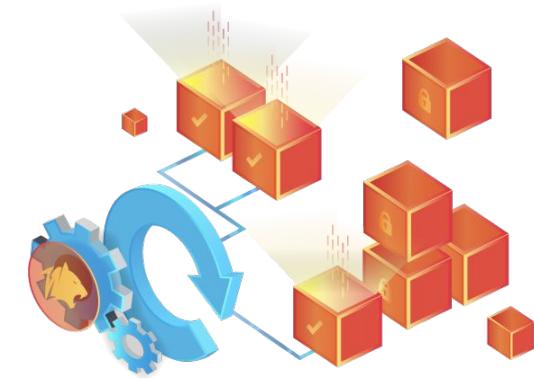
Zero Trust Network Security

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



TIGERA



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

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

Q&A

Schedule a Demo: tigera.io/demo

Follow us on:



TIGERA