

Zone Based Security with Calico

Kubernetes in a Zone Based
Architecture World

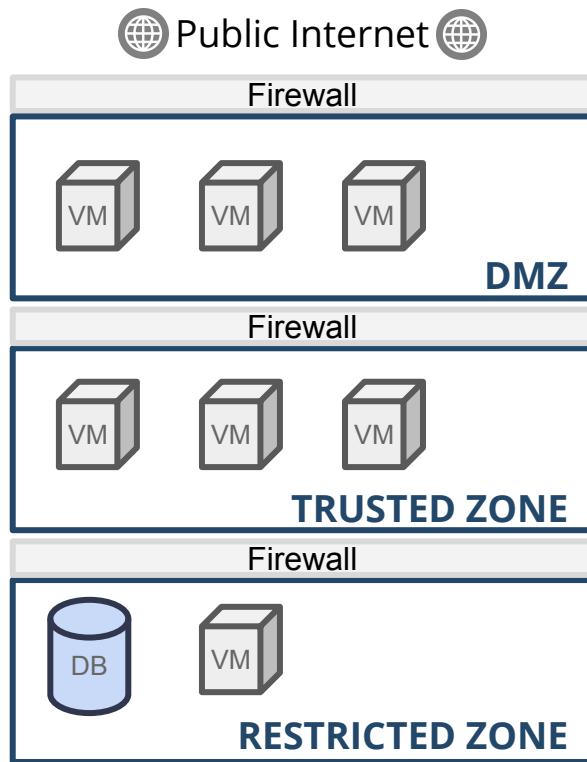


Drew Oetzel - Senior Technical Solutions Engineer



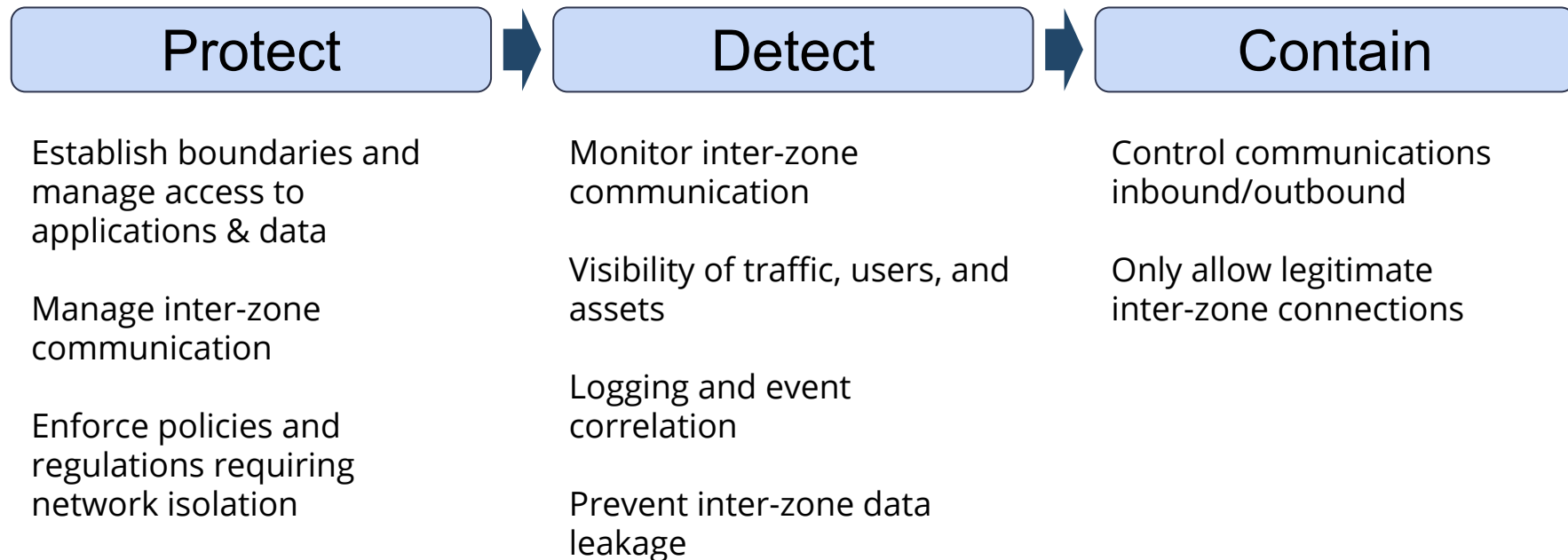
- Working with enterprise software since the late 90s
- 7 years at Splunk, honing his security skills
- 2.5 years at Mesosphere, then Heptio mastering the art of distributed systems, containers, and all that goes along with them
- Outside of tech ask him about history, gardening, or what he's doing to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

Zone based architecture



Zone Based Architecture Overview

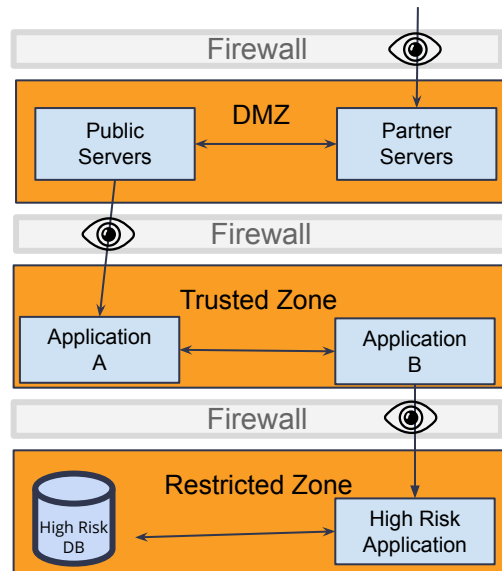
Security Zones are put in place to



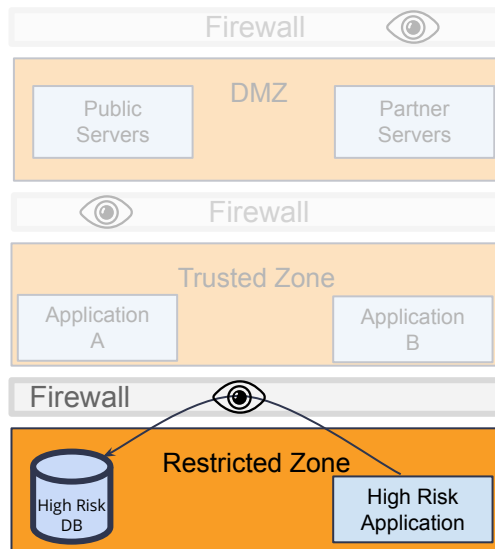
Zone Visibility

Visibility is typically between zones via Firewall.

Within zone is (mostly) trusted



When greater visibility is required, intra-zone traffic is "Hair Pinned" through firewall



Firewalls are sized by Gbps throughput

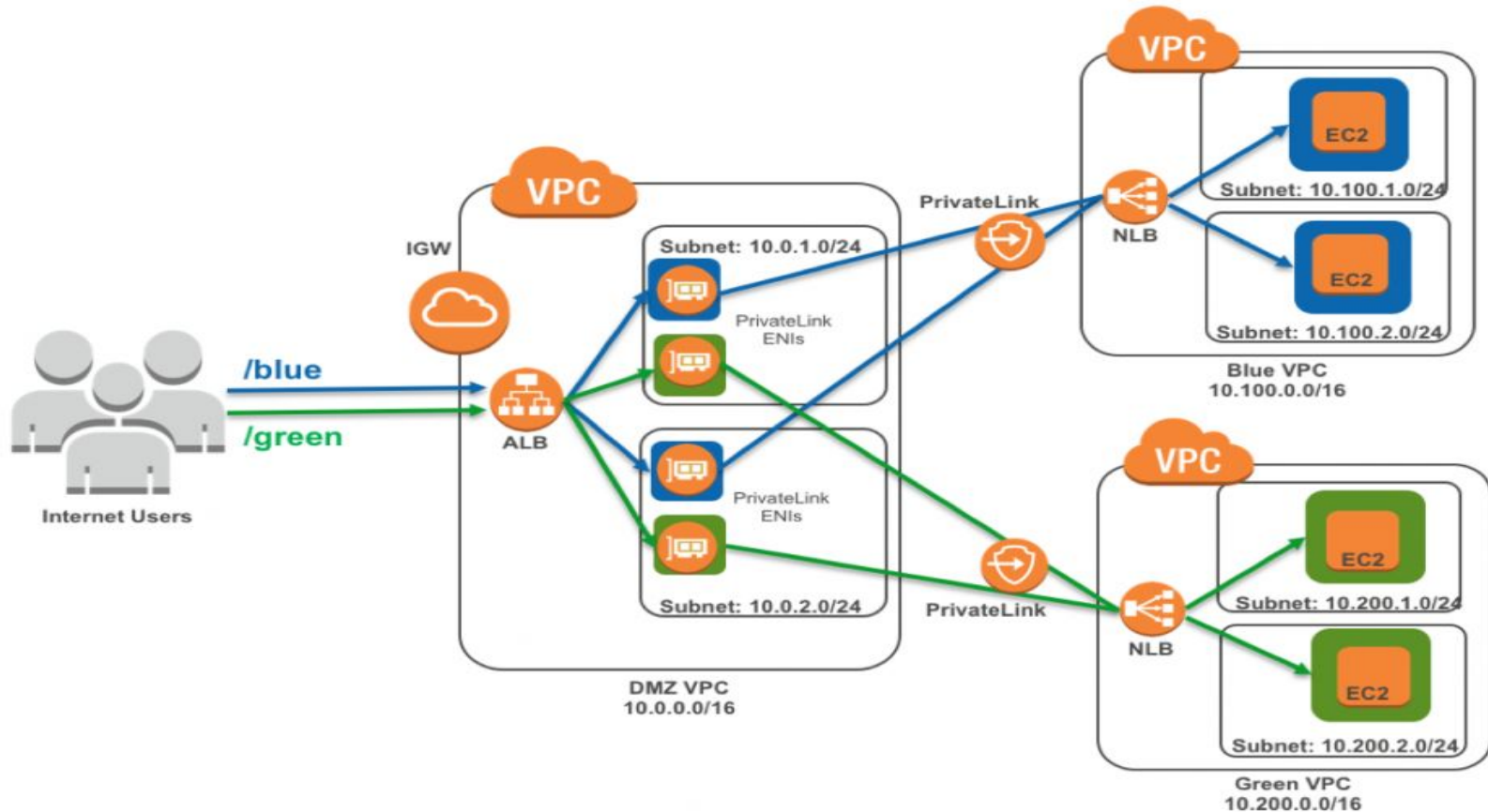
Hair pinning becomes expensive

High throughput firewalls cost millions \$\$\$

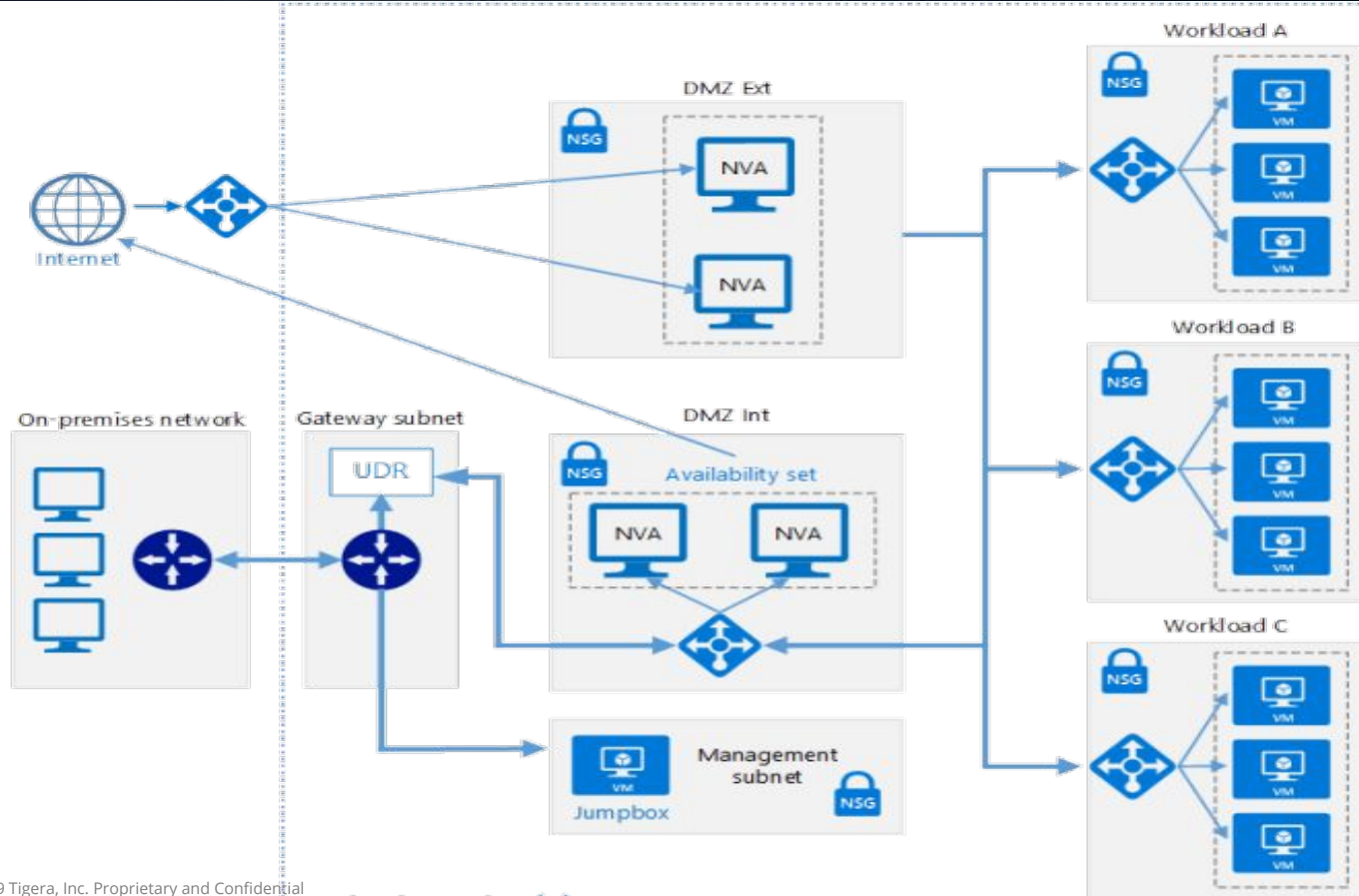
Migration to the cloud



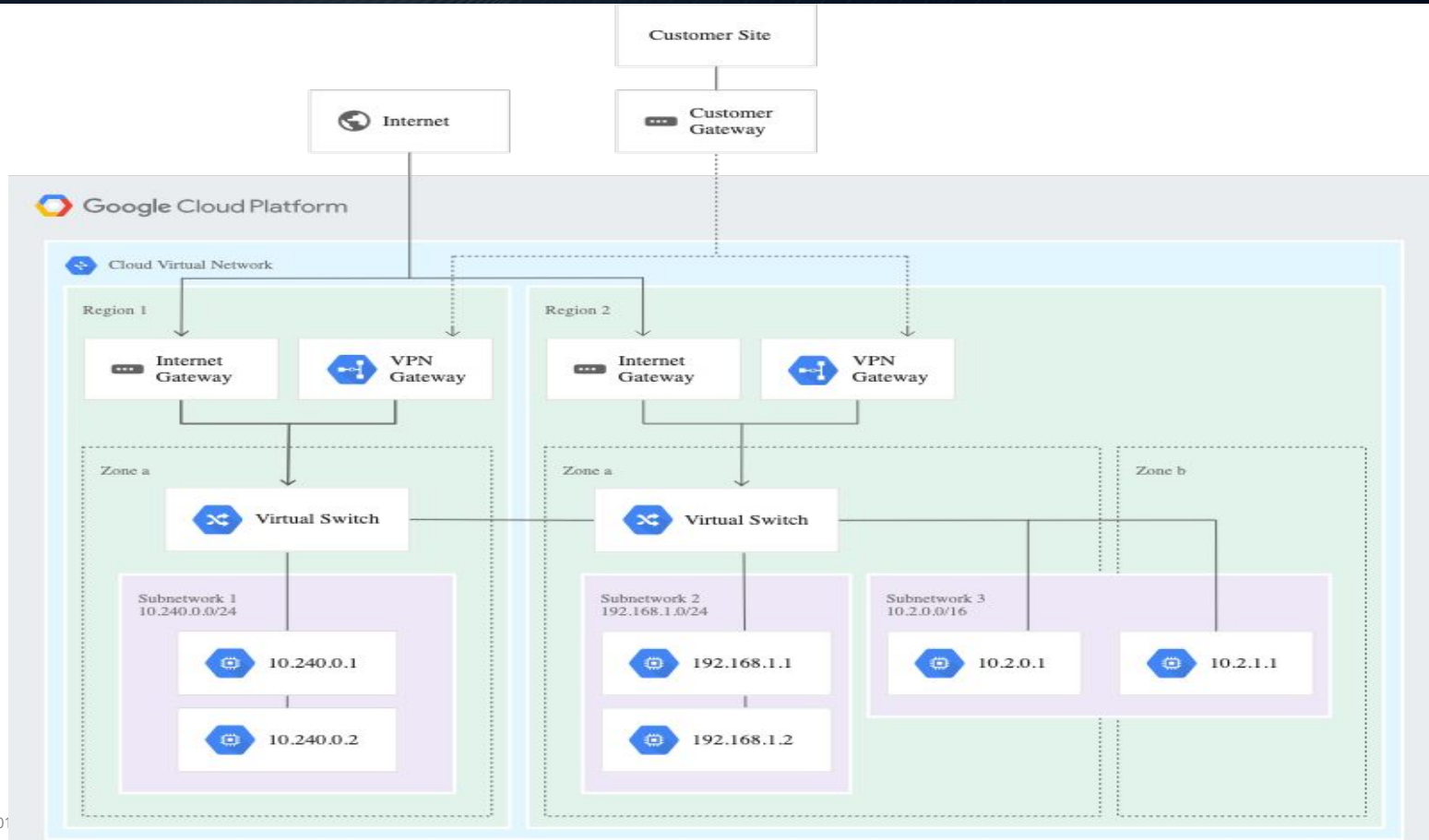
Zone based architecture - in the cloud



Zone based architecture - in the cloud - continued



Zone based architecture - in the cloud - continued

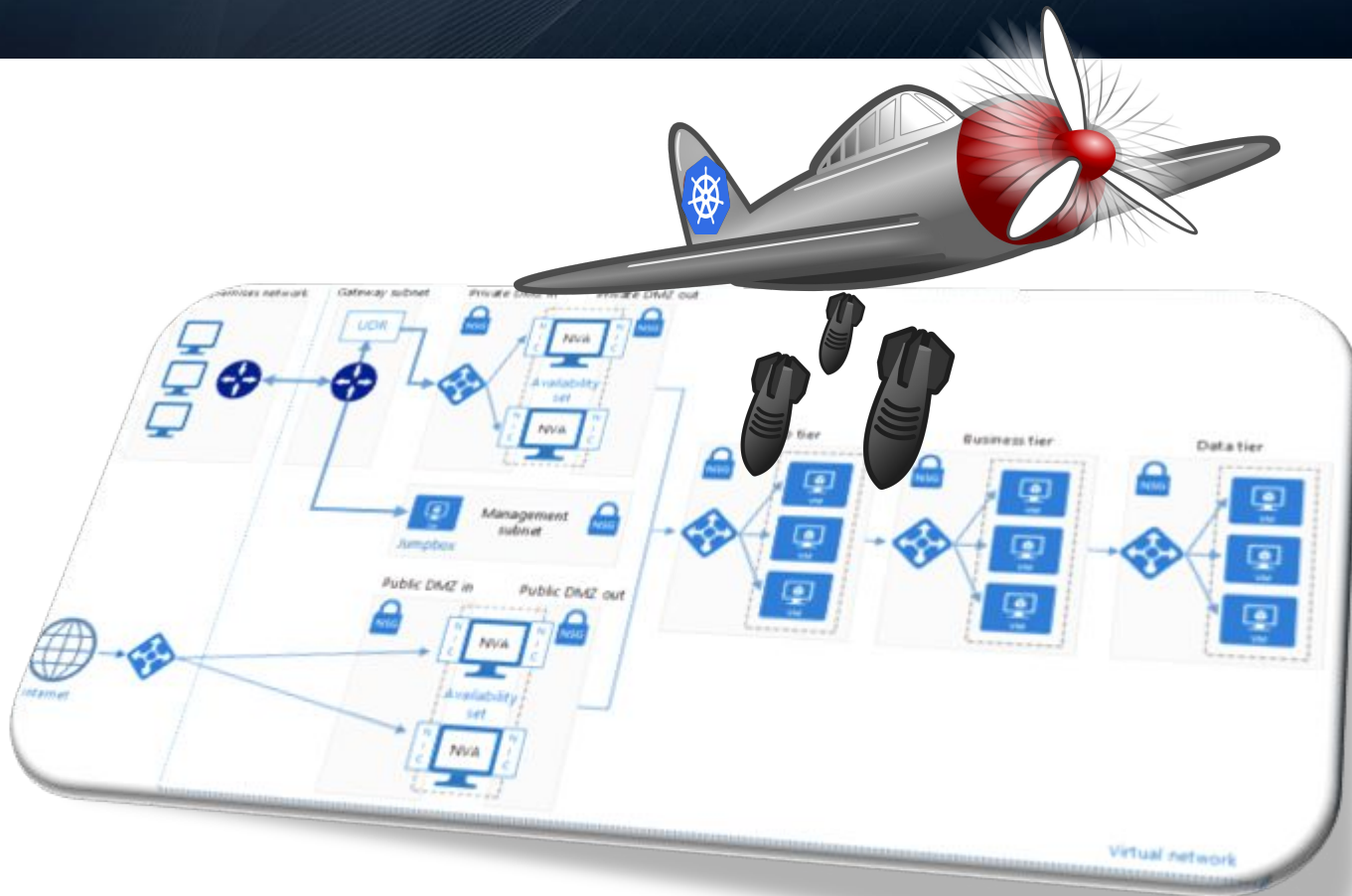


Benefits to Security Team

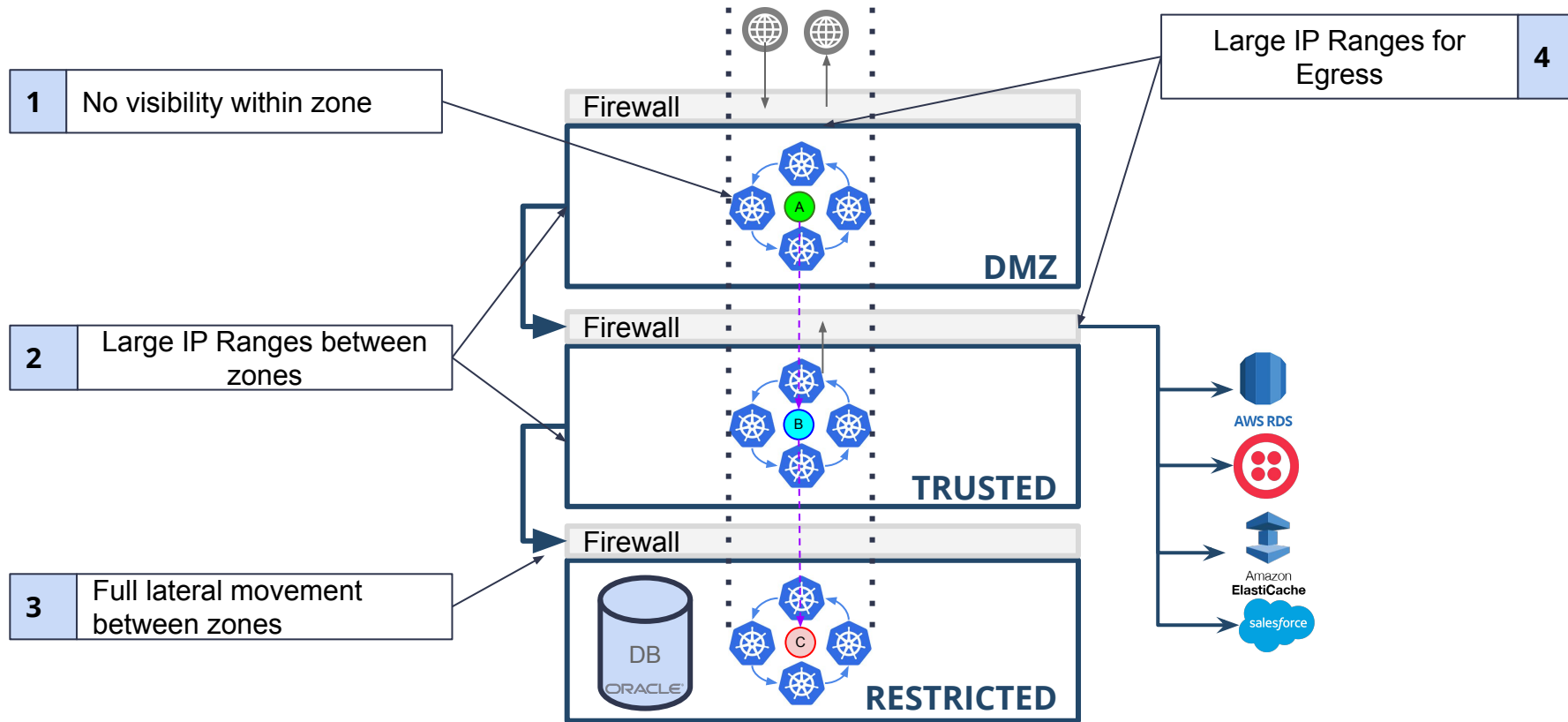
Key Benefits of network segmentation and zones

- Limit exposure & breadth of impact
- Group assets - prioritize focus & spend to high-risk areas
- Visibility and the ability to detect suspicious activity
 - Anomaly detection at each zone firewall
- Containment of an attack to a single zone
- Expedite eradication and recovery - smaller blast radius

Guess What?

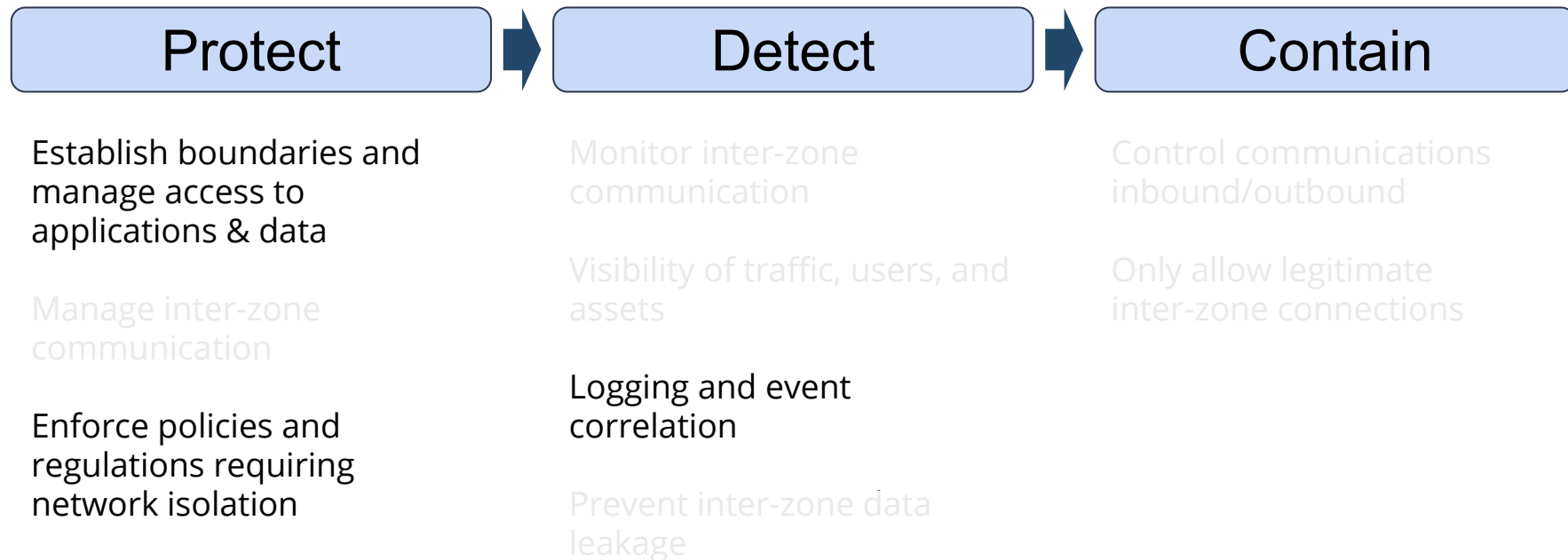


Firewalls for Kubernetes: Key Challenges

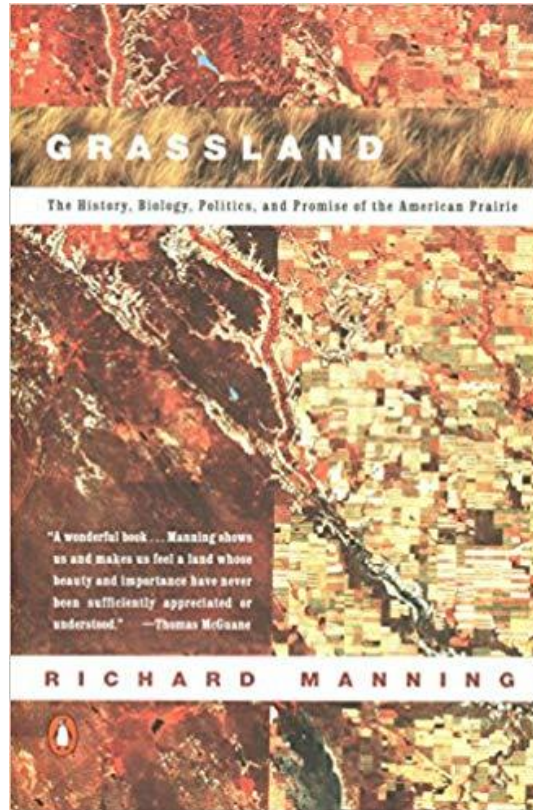


Zone Based Architecture Overview

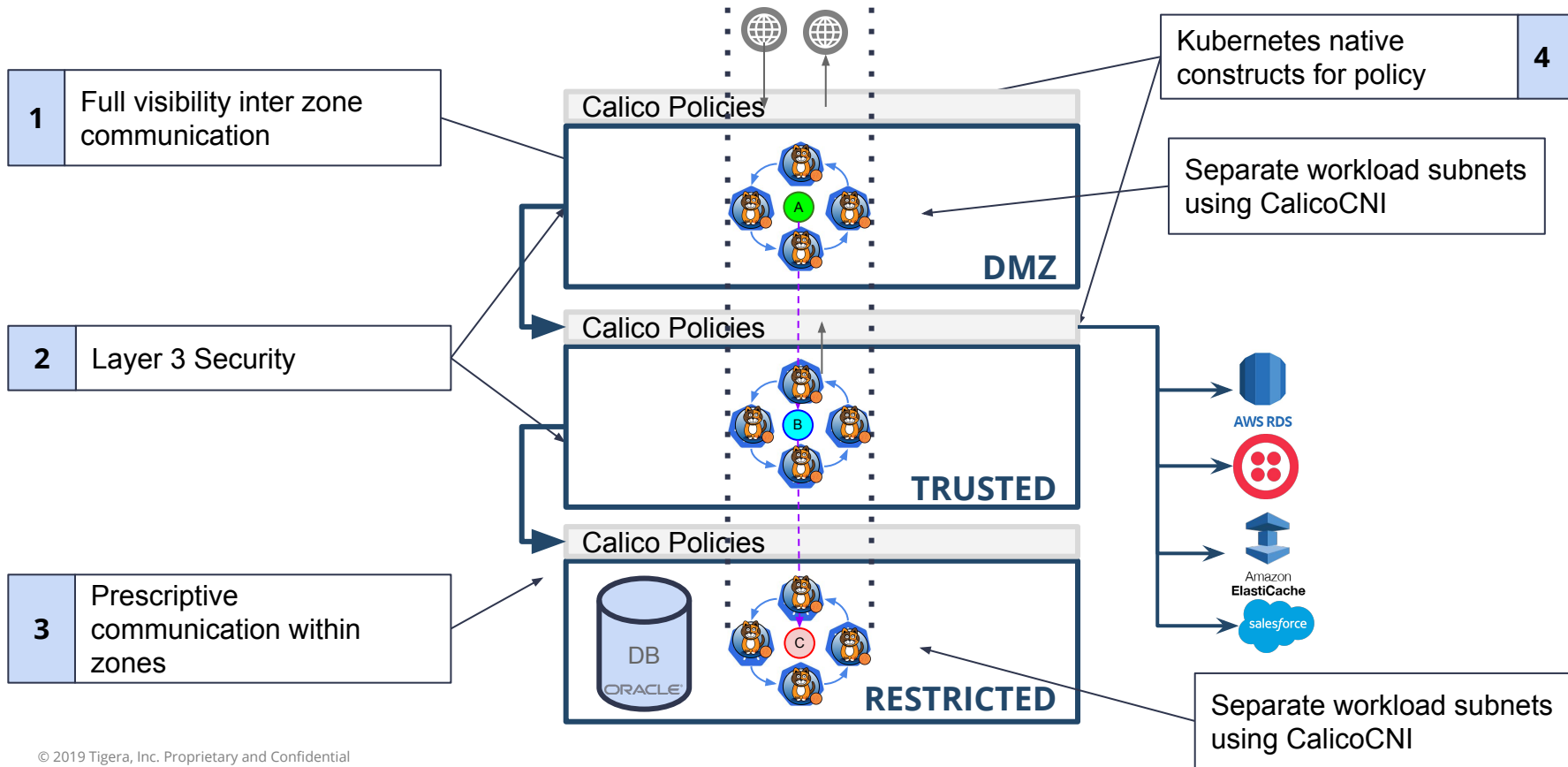
Security Zones are put in place to



A better approach?



Calico to the rescue!



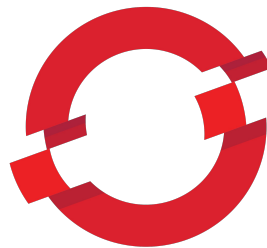
Using the CalicoCNI to Build Zones



Where to run the Calico CNI?



Upstream K8s -
bare metal or
cloud



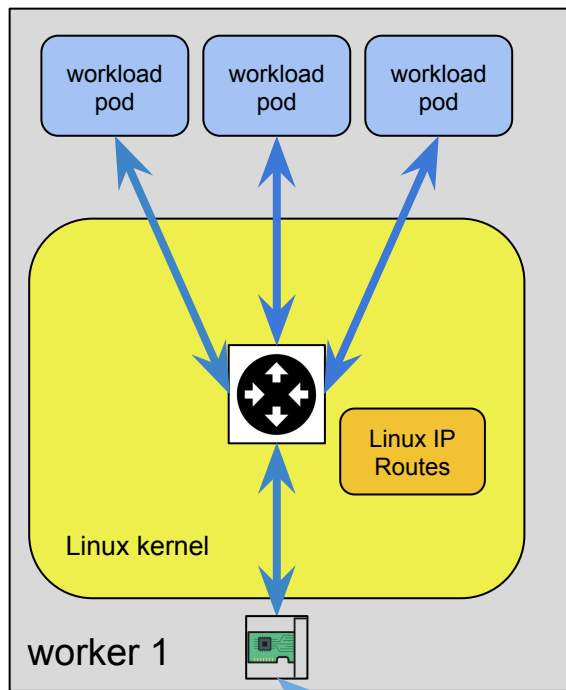
OPENSIFT



The artist formerly
known as
Mesosphere

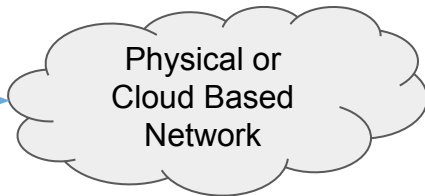
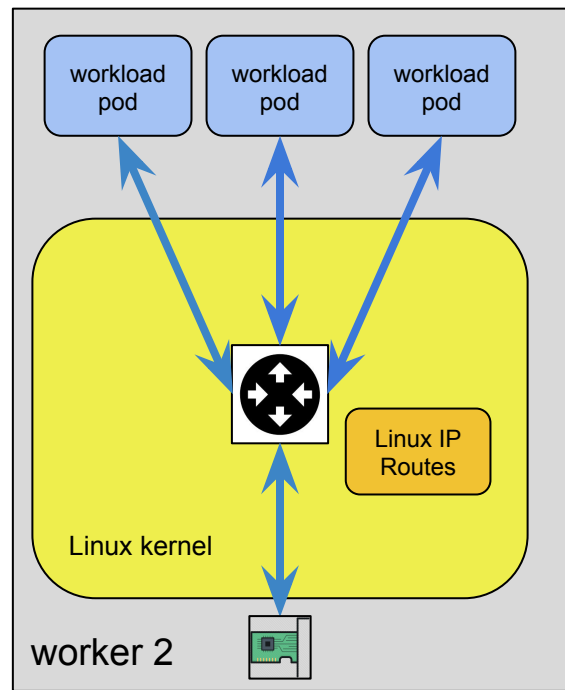


Intra Pod and Intra Node Networking

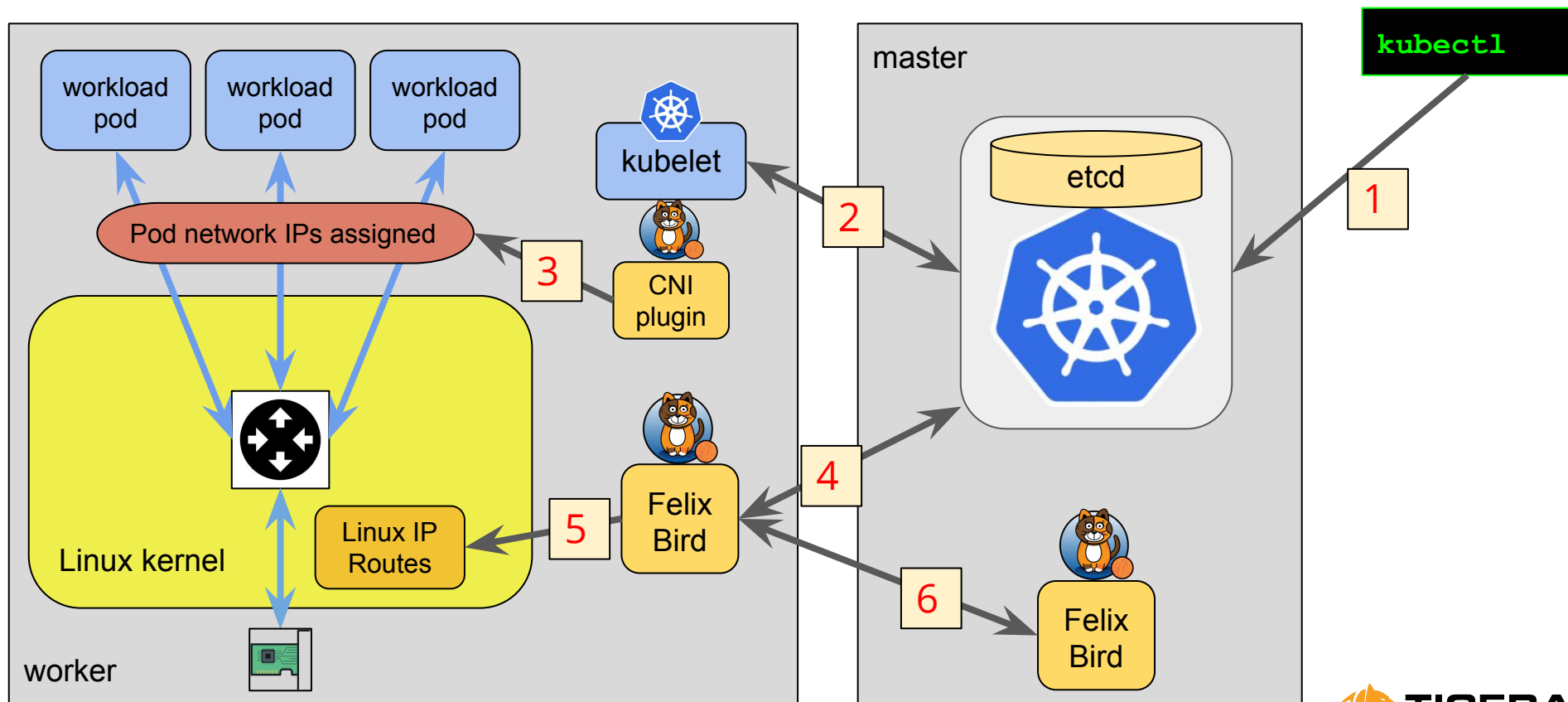


First network hop for any pod / container based workload is the Linux kernel

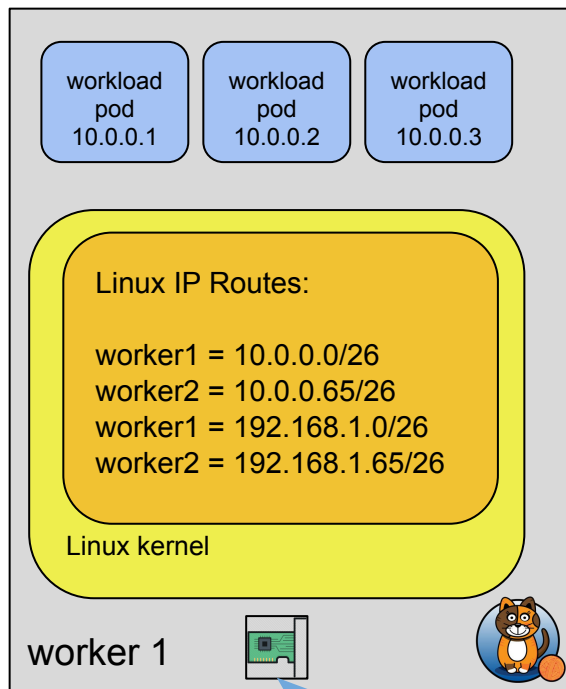
Pod traffic to another pod on the same node never leaves the kernel



High Level Calico CNI Architecture



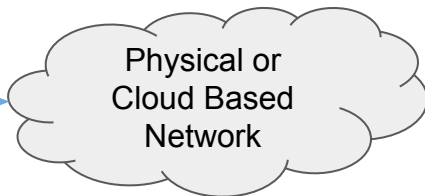
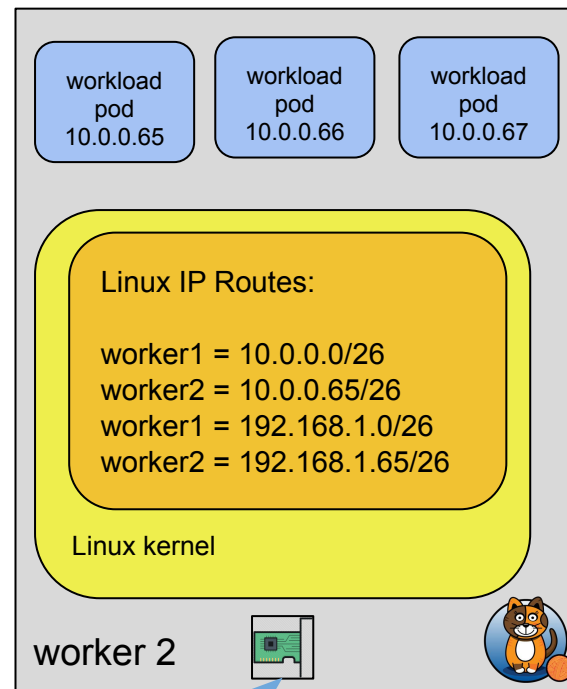
Calico Gives Each Node a Subnet



By default Calico gives each node a /26 subnet for up to 64 pods

64 contiguous IPs means simple route tables - one entry per node

Additional subnets can be allocated if a node gets more than 64 pods scheduled



IP Pools

- > **IP Pools** can be associated with namespaces, workloads, or topology
- > **Topology**: designate certain nodes with a node label in Kubernetes, then reference that label in the pool definition
- > **Namespace**: add a Calico annotation to the namespace definition
- > **Workload**: same as namespace – though any annotation set at the namespace level will take priority over a workspace annotation
 - Example annotation added to namespace or workload yaml:
`annotations: "cni.projectcalico.org/ipv4pools": "[\"dmz-pool\"]"`
 - Note all the “\” escapes are necessary



IP Pool Definition - Topology

```
dmz_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



IP Pools – Namespaces and workloads

```
dmz_ns.yaml
```

```
apiVersion: v1
```

```
kind: Namespace
```

```
metadata:
```

```
  name: dmz
```

```
  labels:
```

```
    location: dmz
```

```
  annotations:
```

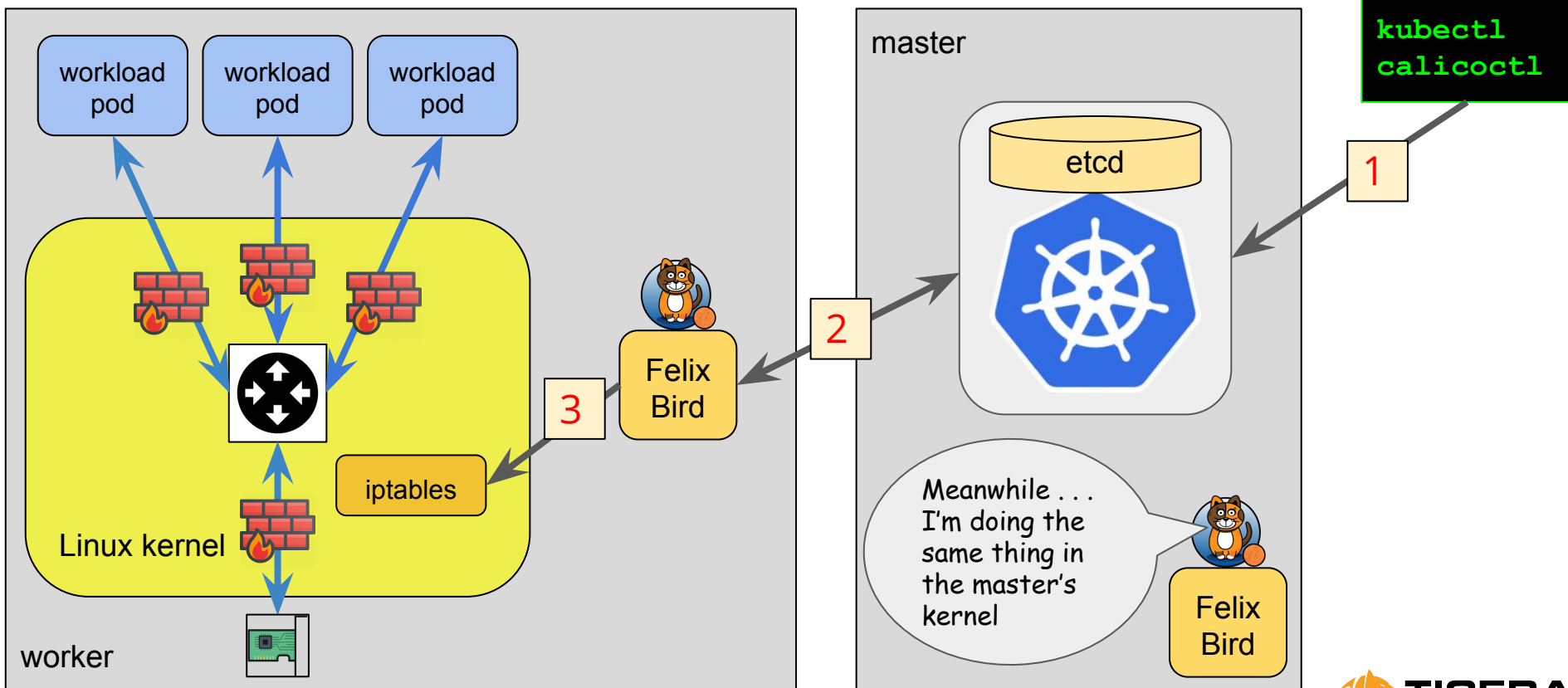
```
    "cni.projectcalico.org/ipv4pools": "[\"dmz-pool\"]"
```

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



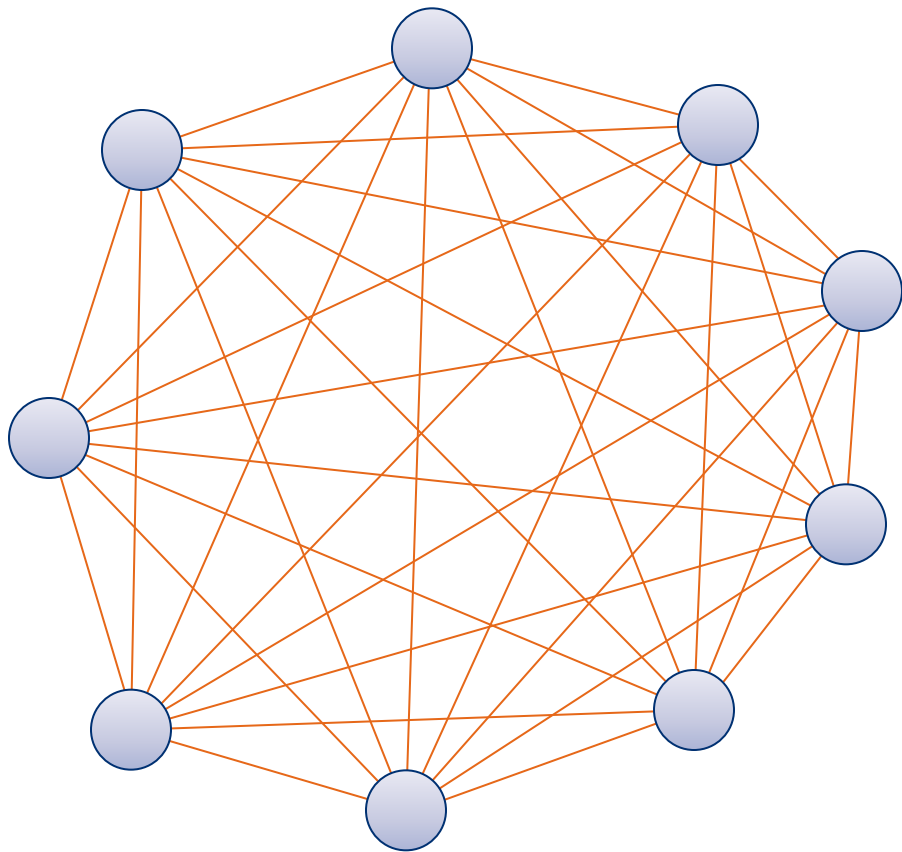
Calico Network Policy

Calico High Level Policy Architecture

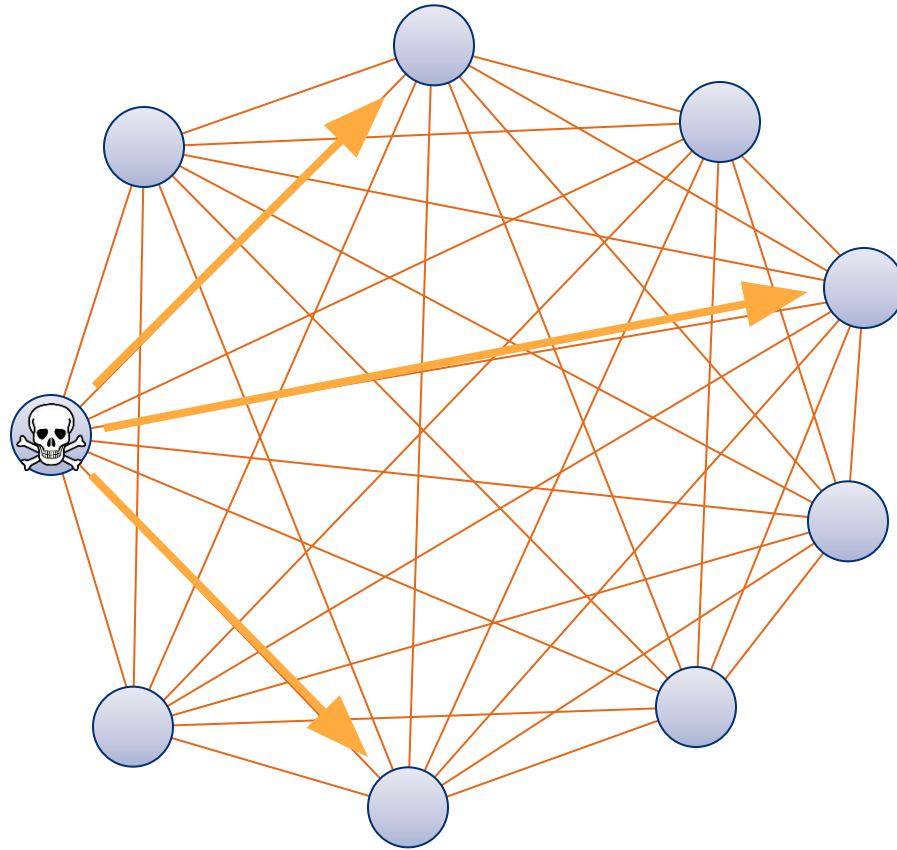


MOTIVATION FOR NETWORK POLICY

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.

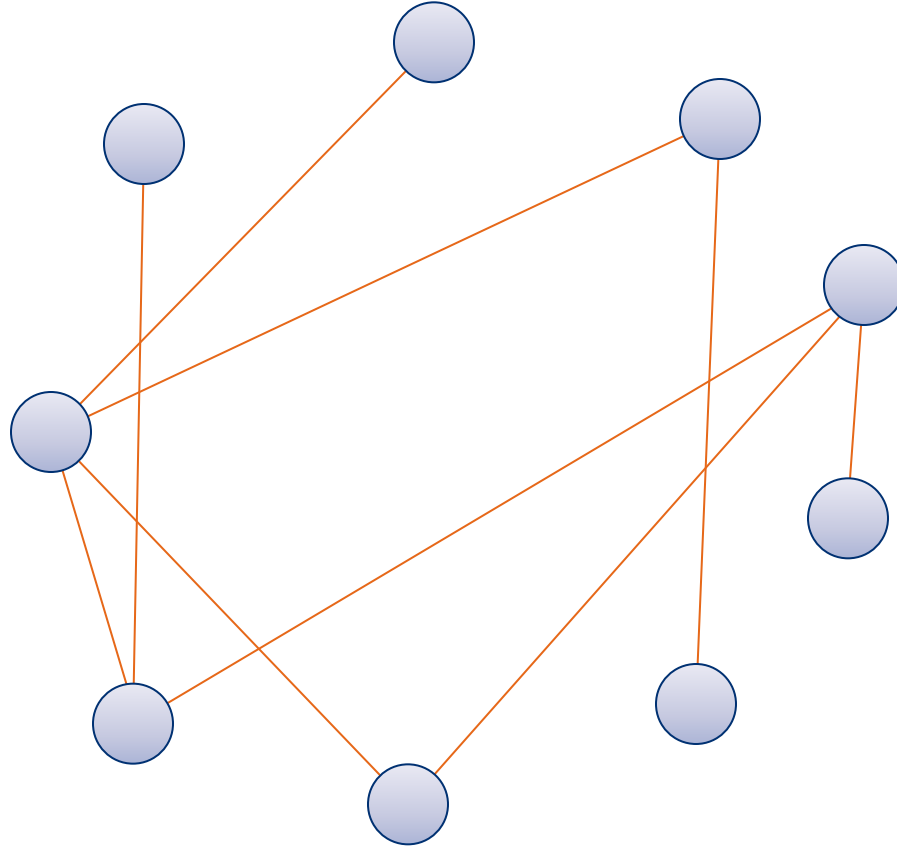


The rest are only
useful for an
attacker.

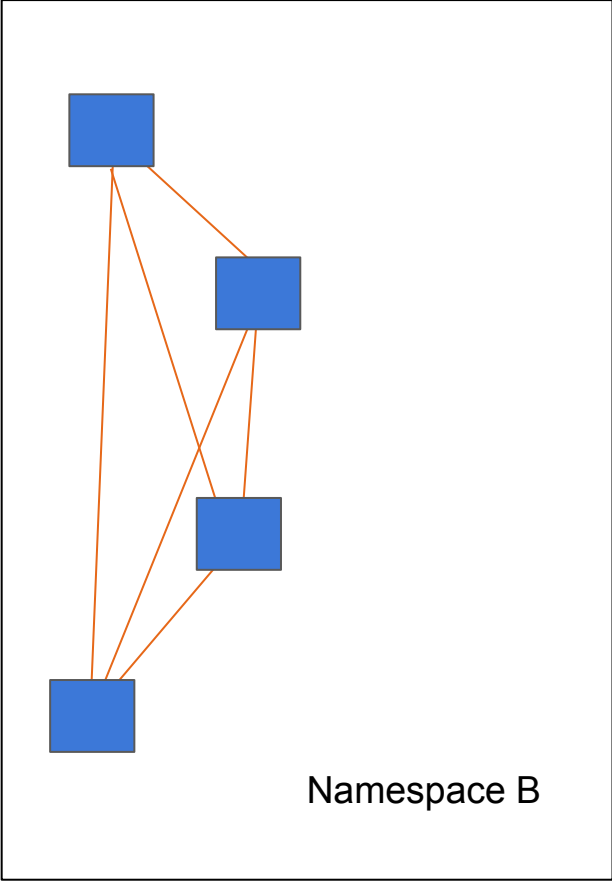
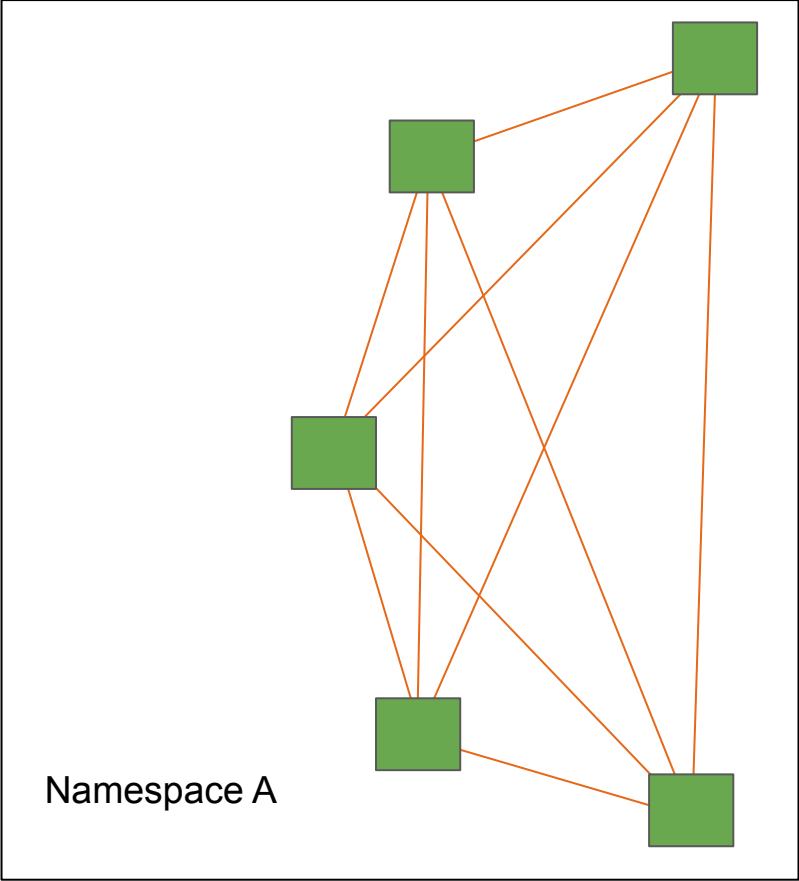


We know that our frontend load balancers don't connect directly to the backend database. dev containers do not connect to prod, and so on.

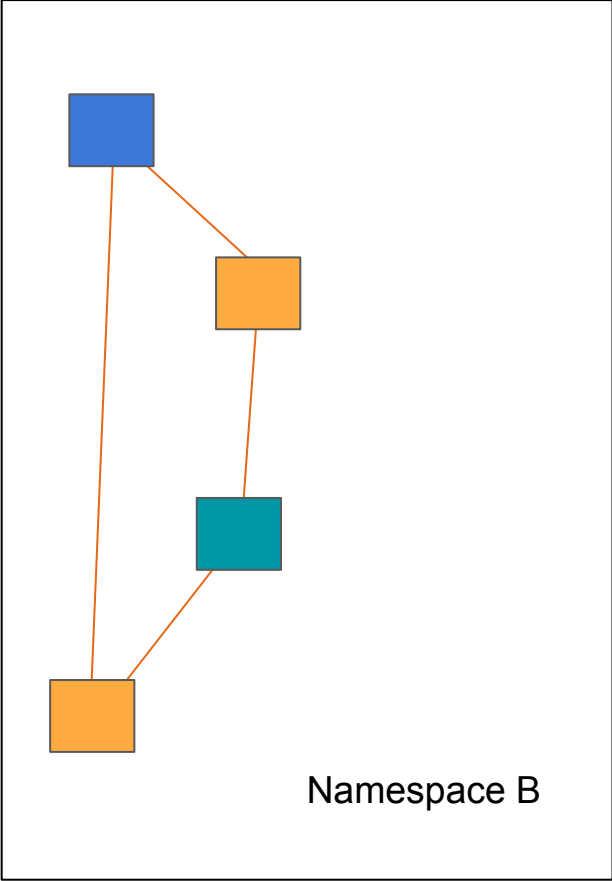
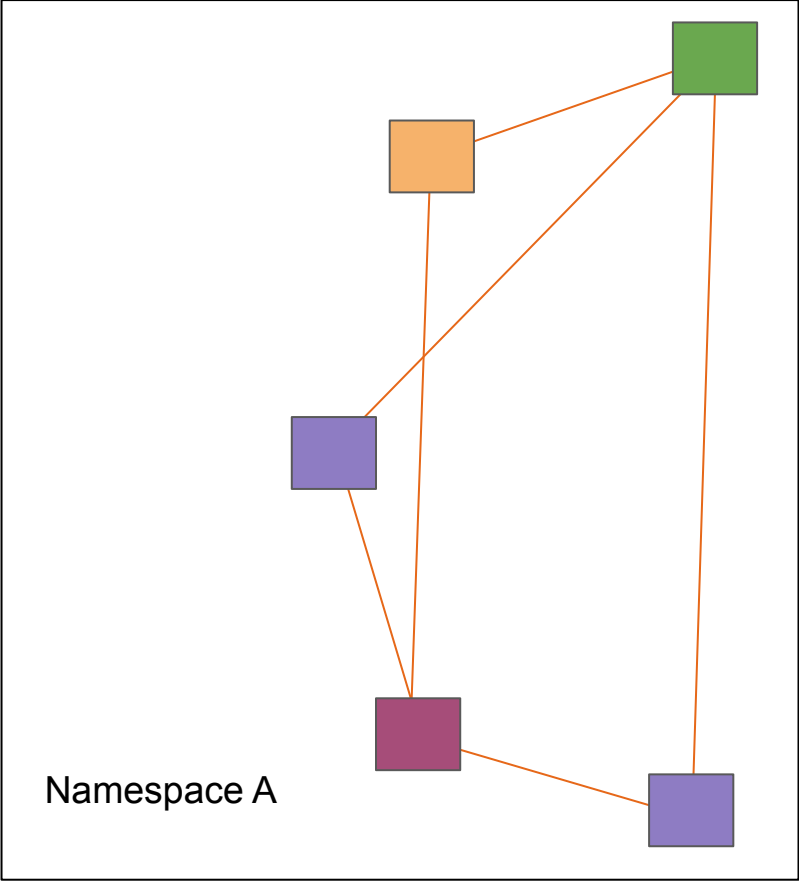
Each connection path that we can eliminate will reduce complexity and improve security.



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
```

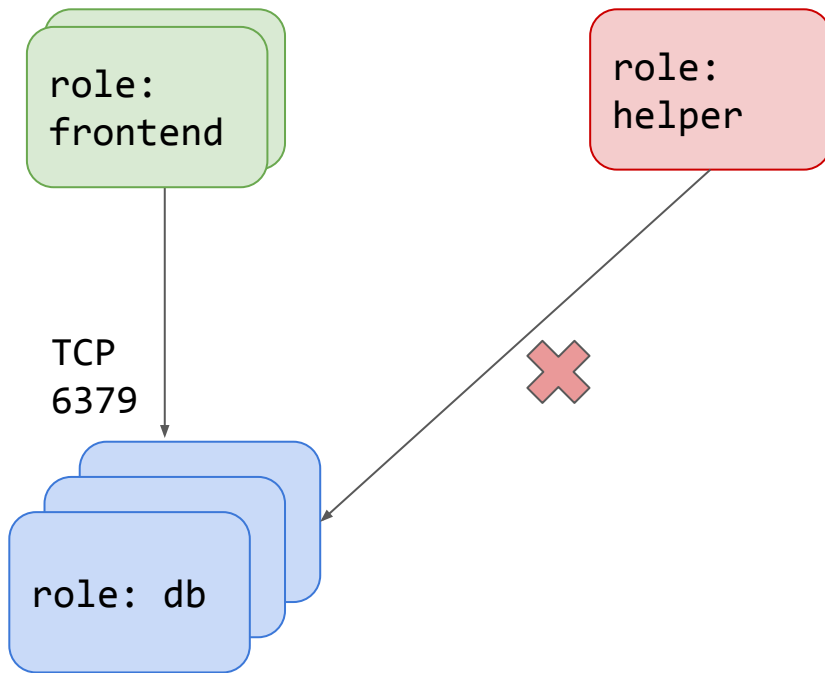
KUBERNETES NetworkPolicy RESOURCE

- > 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:
```

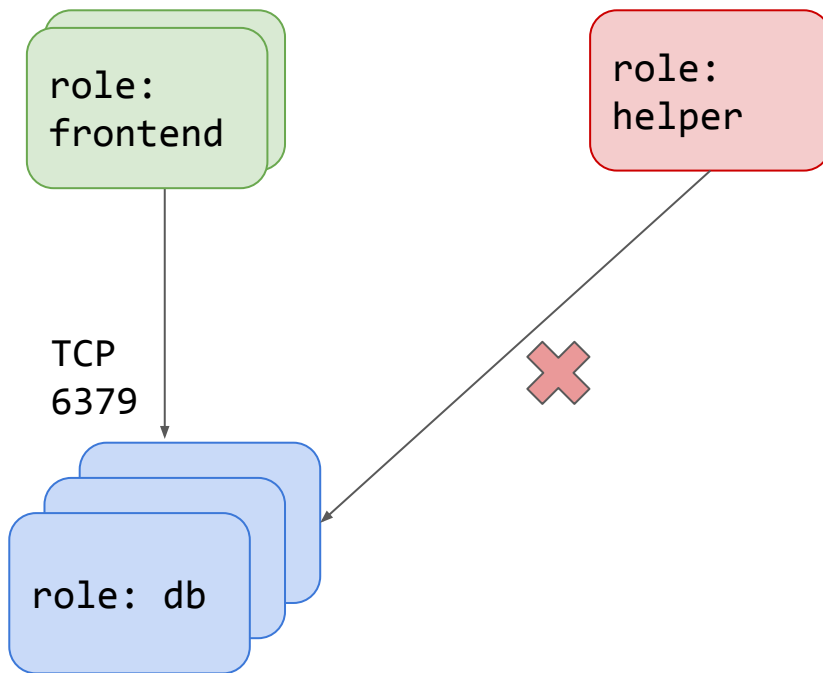
```
        matchLabels:
```

```
          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:
```

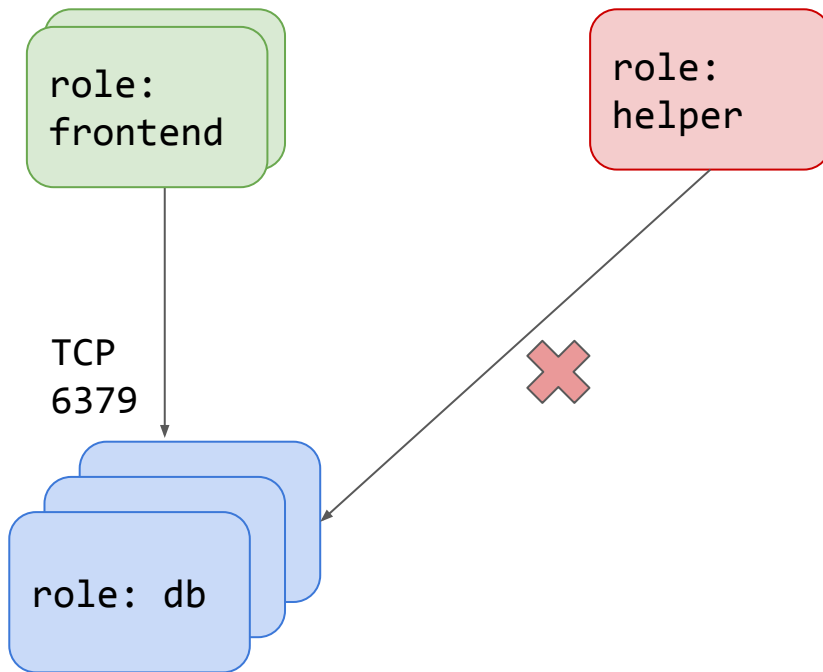
```
      matchLabels:
```

```
        role: frontend
```

```
  ports:
```

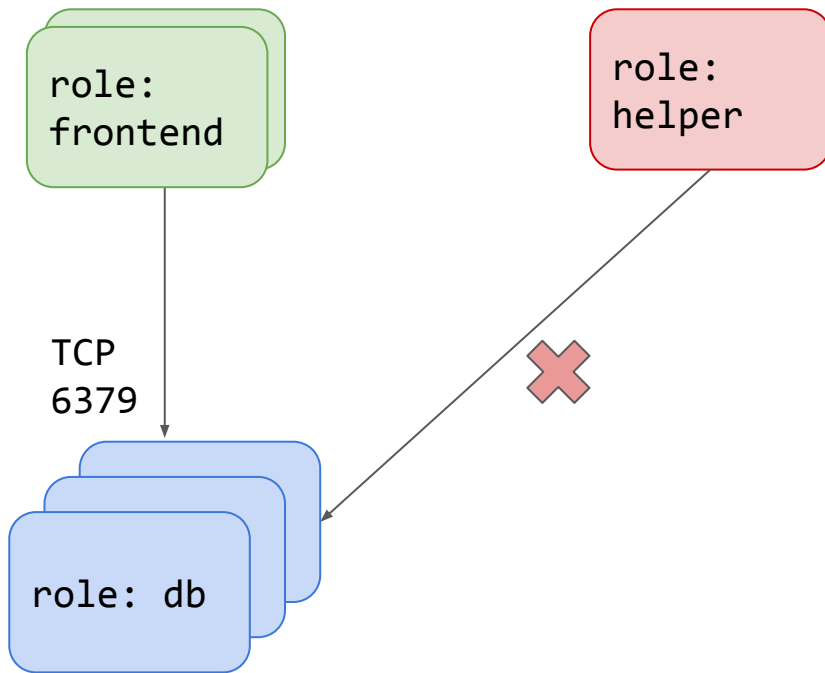
```
  - protocol: TCP
```

```
    port: 6379
```



Kubernetes Network Policy Example

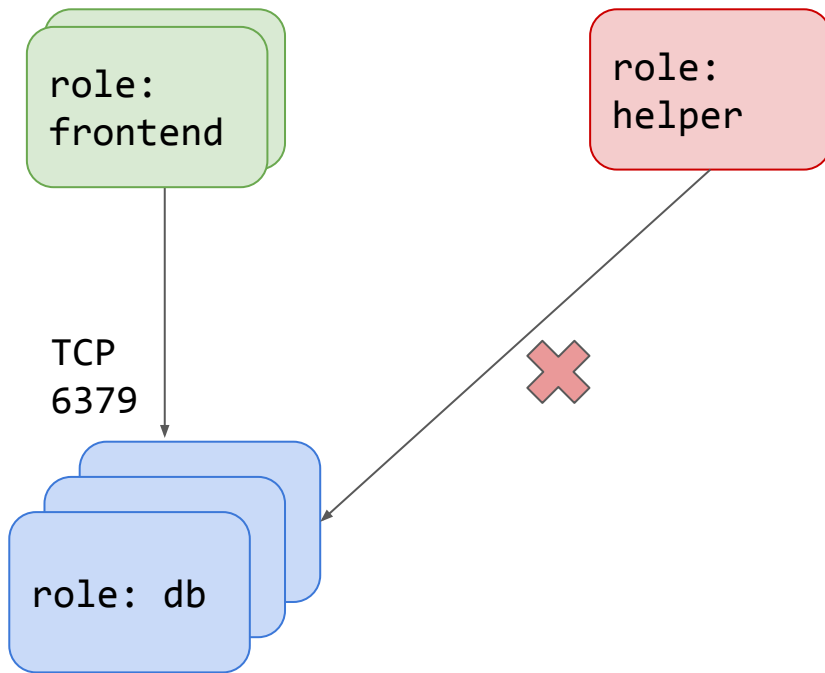
```
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: my-network-policy
  namespace: my-namespace
spec:
  podSelector:
    matchLabels:
      role: db
  ingress:
    - from:
        - podSelector:
            matchLabels:
              role: frontend
      ports:
        - protocol: 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** (alpha in 3.0, GA in 3.1)
 - **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

Calico Network Policy Example

```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
```

```
metadata:
```

```
  name: security-operations.quarantine
```

```
spec:
```

```
  order: 200
```

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

```
  ingress:
```

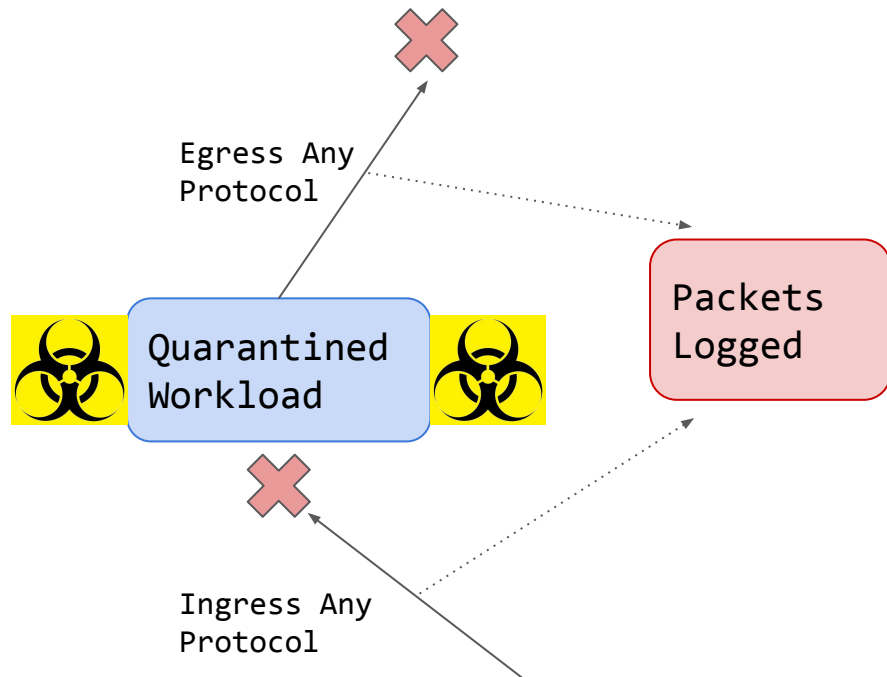
- action: Log
 source: {}
 destination: {}
- action: Deny
 source: {}
 destination: {}

```
  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:

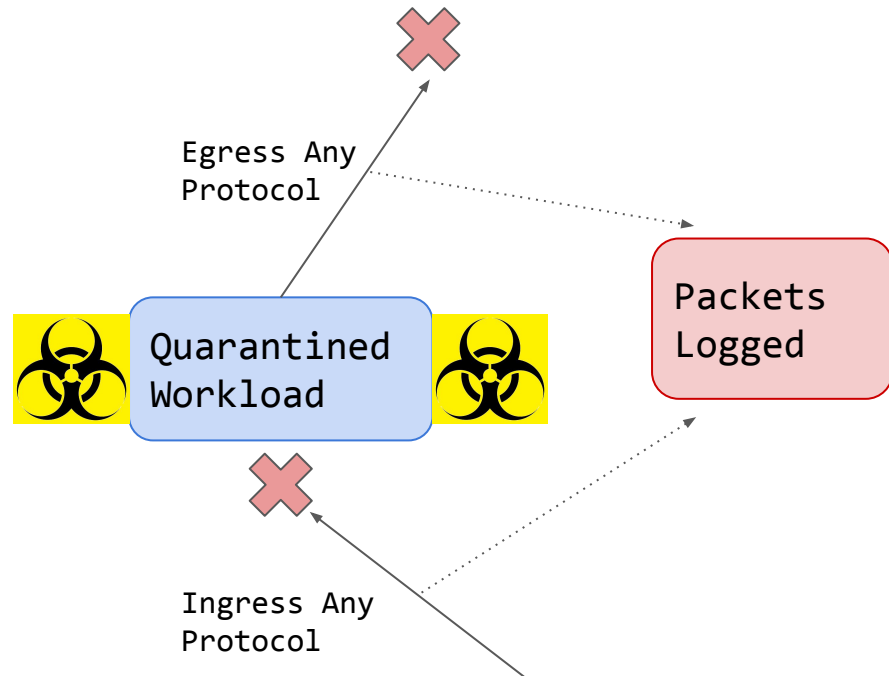
- action: Log
source: {}
destination: {}
- action: Deny
source: {}
destination: {}

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:

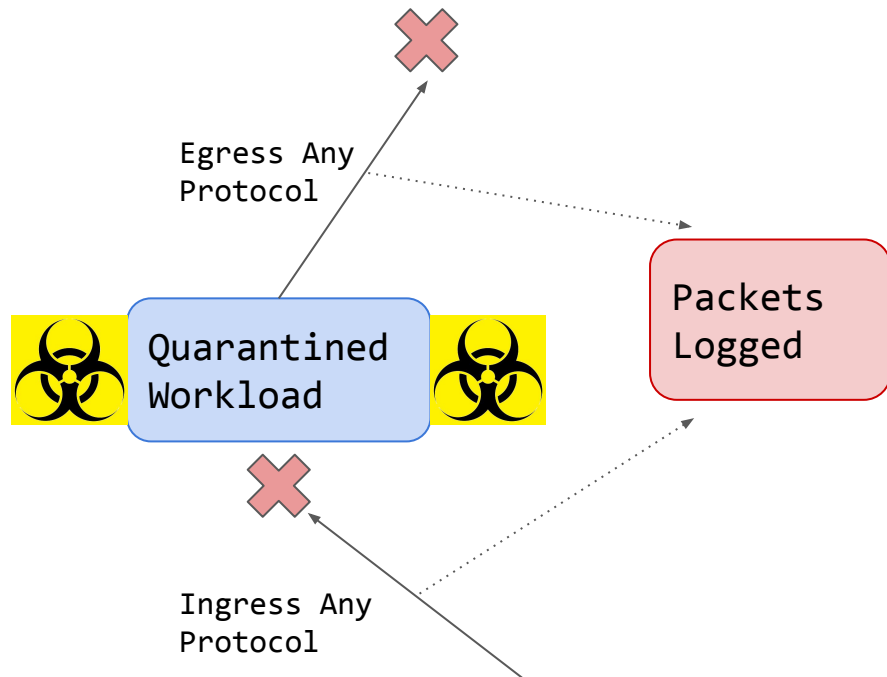
- action: Log
source: {}
destination: {}
- action: Deny
source: {}
destination: {}

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/>



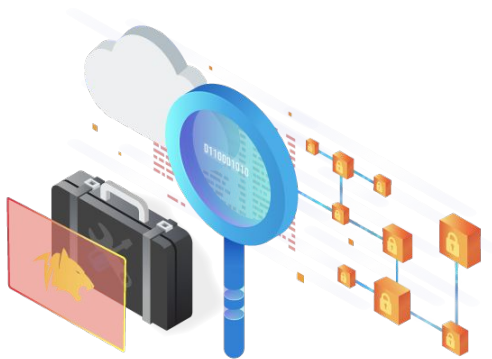
FELIX TIP: Calicoctl is available for Linux, MacOS and Windows. Download it for your favorite desktop operating system!

Tigera Secure Enables Key Security Capabilities



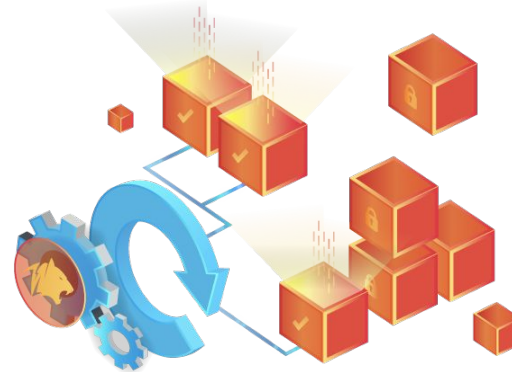
Zero Trust Network Security

Extend your security controls
to Kubernetes



Visibility & Threat Detection

Monitor traffic, detect and
prevent threats



Continuous Compliance

Continuous reporting, alert on
non-compliance

Thank You - Please like us at
github.com/projectcalico/calico



TIGERA

Follow us on:

