



Scalable Security and DevSecOps with Calico's Network Policy Model for Kubernetes

enables organizations to deliver security at the speed of business by facilitating collaboration, delegation, defence-in-depth, scale, and operational simplicity.

Why Implement Microsegmentation

Assume breach mindset:

Microsegmentation helps contain threats to infected workloads and prevents lateral movement across the network.

Limits blast radius:

Even if a breach occurs, microsegmentation limits its impact, ensuring threats don't spread beyond targeted workloads.

Supports zero trust architecture:

Trust is never assumed; communication between workloads requires explicit permissions, ensuring tighter security.

Provides defense in depth:

Microsegmentation adds a granular layer of defense at the workload level, complementing perimeter-based security measures.

Reduces attack surface:

Isolating workloads into smaller segments minimizes exposed systems, lowering the risk of infiltration.

Ensures regulatory compliance:

Microsegmentation helps meet industry-specific compliance requirements (e.g., PCI, healthcare) by isolating sensitive workloads.

Improves threat detection:

Microsegmentation detects abnormal behavior, such as unauthorized communication between workloads, improving security monitoring.

Enables faster threat response:

Predefined controls allow quick actions like quarantining compromised workloads, without needing to create responses on the fly.

Kubernetes-specific benefits:

It mitigates risks in Kubernetes' flat, dynamic, and ephemeral network by enforcing strict communication policies.

Facilitates collaboration and governance:

Microsegmentation fosters security governance across teams and ensures adherence to security policies in Kubernetes environments.

Video Link: [Why Implement Microsegmentation](#)

Defining the Policy Model

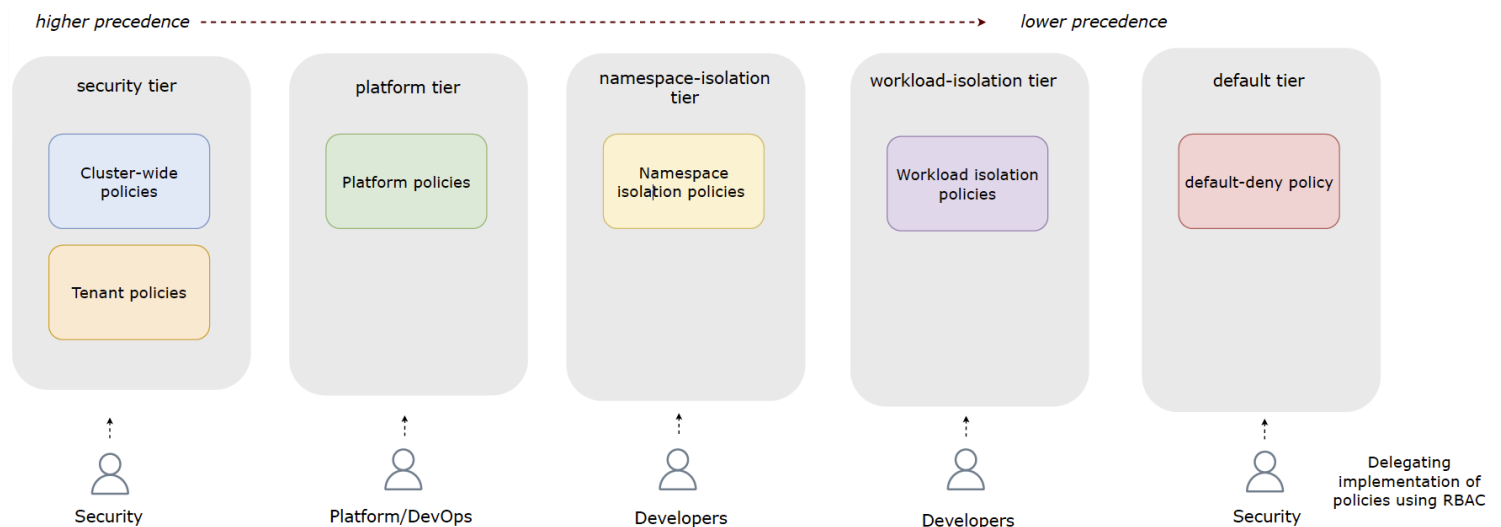
Step 1 – Identify Kubernetes Security Domains

A security domain refers to a defined scope or boundary within a network where a specific set of security policies, controls, and privileges are applied and enforced.



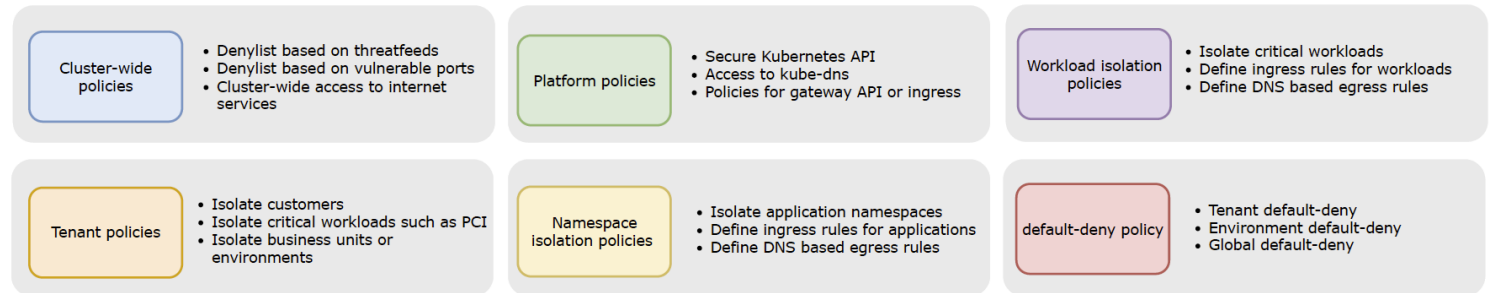
Step 2 – Define Calico Security Policy Tiers

A hierarchical security policy model with Calico policy tiers enables organizations to deliver security at the speed of business by facilitating collaboration, delegation, defence-in-depth, scale, and operational simplicity.



Step 3 – Define Security Policy Standards

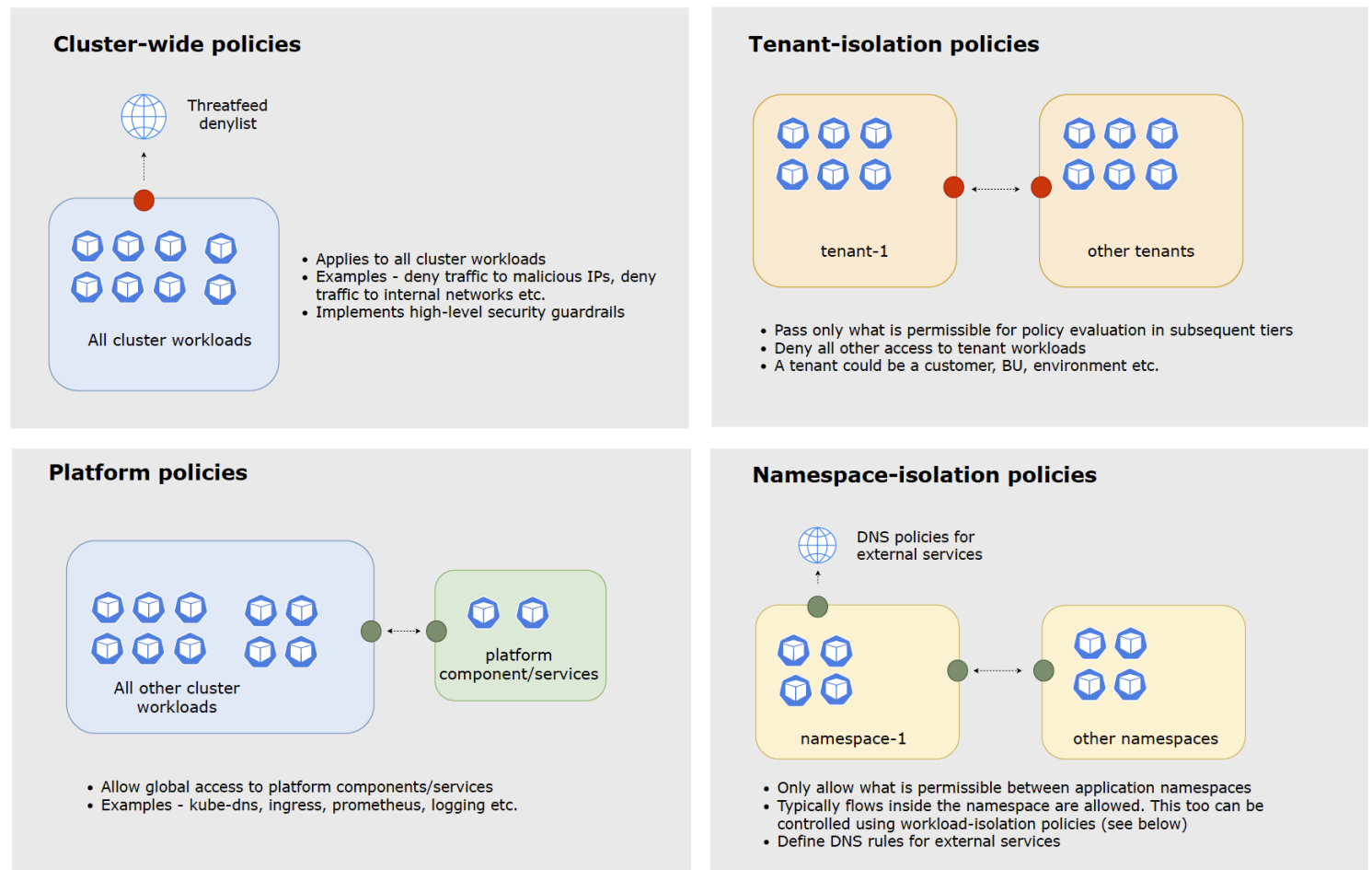
Defining a policy standard for each tier ensures consistency, reduces complexity and provides for a robust, easy to scale security posture. You can have a varying degree of granularity for each security domain or even each policy depending on the security requirements.



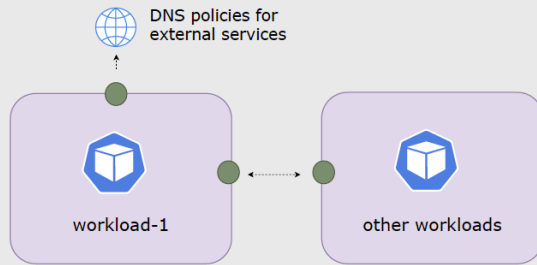
Video Link: [A Hierarchical Policy Model for Microsegmentation in Kubernetes – Part 1](#)

Security Policy Standards – Continued

Ensure each policy maps to a policy standard and YAML definition. Doing so, simplifies the policy model and helps scale and replicate the policies in other clusters seamlessly.

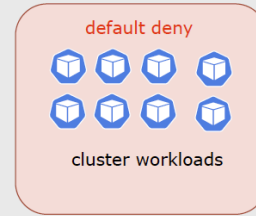


Workload-isolation policies



- Only allow what is permissible between workloads
- This could even apply to workloads in the same namespace
- Define DNS rules for external services

Default-deny policies

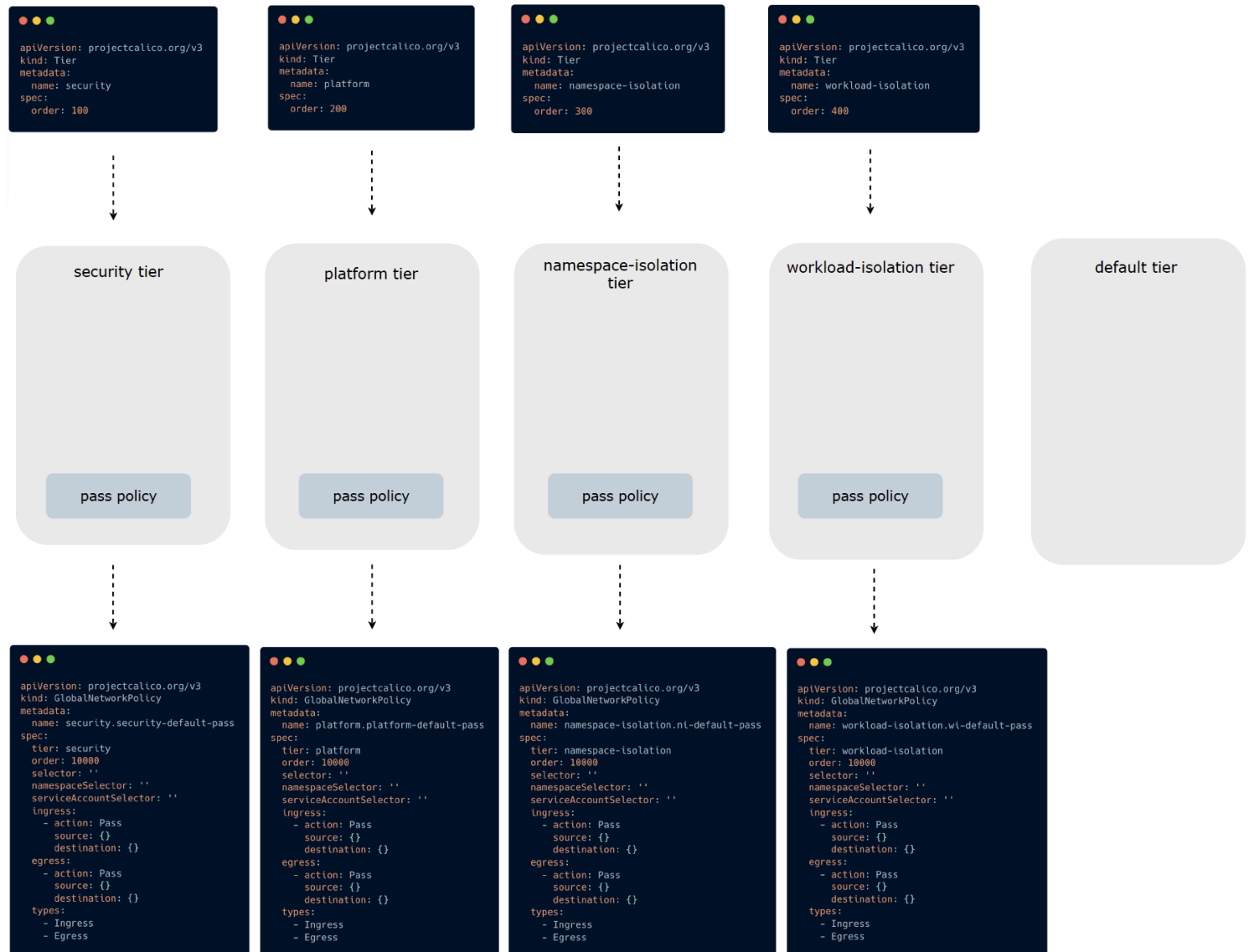


- Deny all flows not explicitly permitted by previous policies
- You can create many stacked deny policies - for example, default-deny for a tenant, namespace, or all cluster workloads

Video Link: [A Hierarchical Policy Model for Microsegmentation in Kubernetes – Part 2](#)

Implementing the Policy Model – Build the Tiers

- The tiers are ordered according to the required precedence
- Lower the order, higher the precedence



- A “pass” policy is required to ensure the policy evaluation is deferred to subsequent tiers for endpoints/pods matched in policies in the current tier.
- If a “pass” action is not implemented for a given endpoint, an implicit “deny” is enforced after all policies are evaluated for that endpoint in the respective tier.

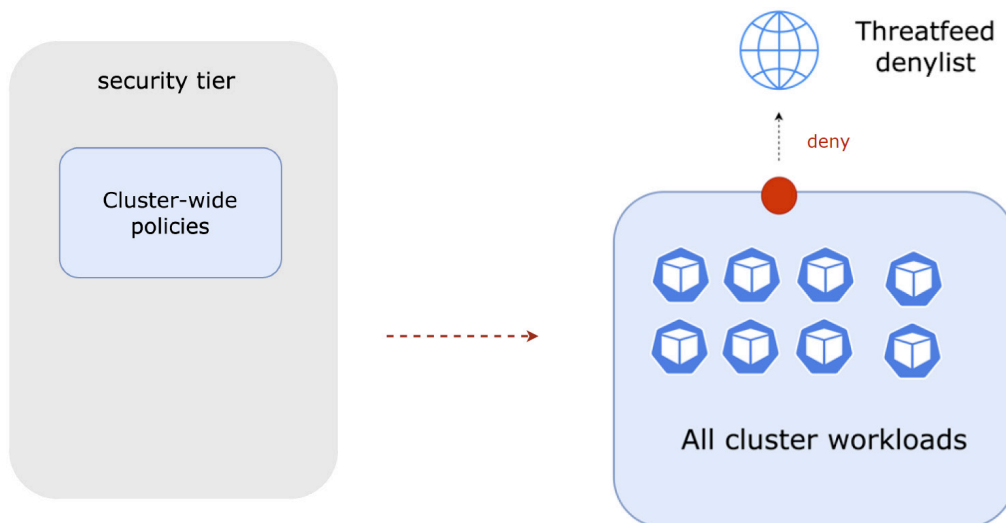
Video Link: [Building the Tiers](#)

The policy examples in the following sections can be found in this git repo – [Microsegmentation Git Repo](#)

An overview of the demo environment can be found here – [Deploying Applications](#)

Cluster-wide Policy Example

- The following example cluster-wide policy denies traffic to malicious IPs based on a threat-feed
- The threat-feed which is a globalnetworkset is selected using an endpoint selector – `feed == “otx-ipthreatfeed”`
- The cluster-wide policy is a high-level guardrail that applies to all workloads in the cluster and is deployed in the security tier



```

apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: security.block-alienvault-ipthreatfeed
spec:
  tier: security
  order: 1
  selector: all()
  namespaceSelector: ''
  serviceAccountSelector: ''
  egress:
    - action: Deny
      source: {}
      destination:
        selector: feed == "otx-ipthreatfeed"
  types:
    - Egress

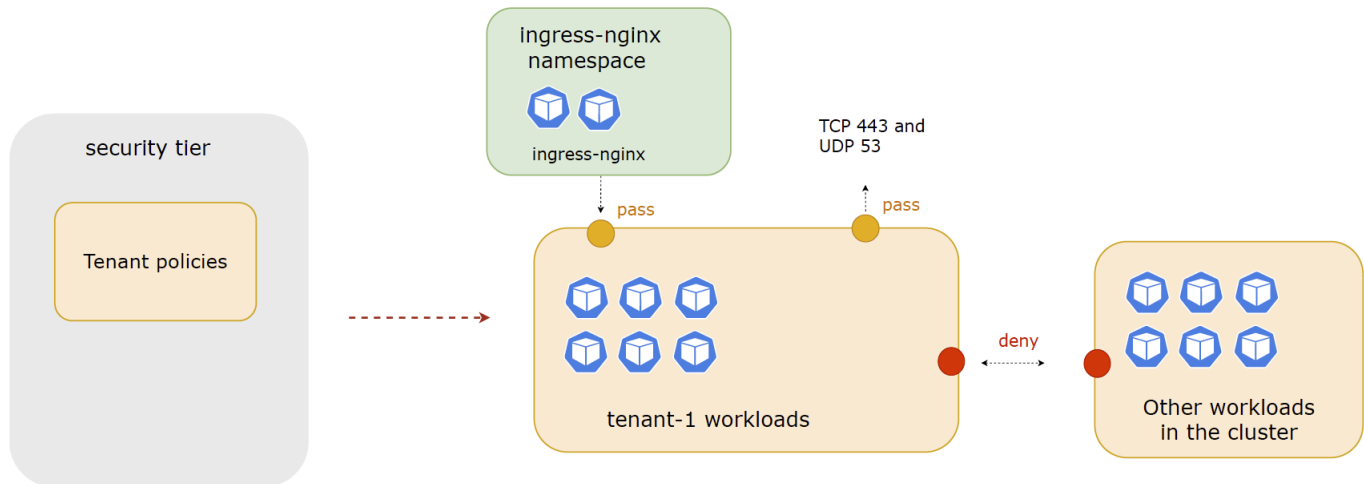
```

egress rule 1: deny traffic to IPs in the threat-feed

Video Link: [Cluster-wide Policy](#)

Tenant-isolation Policy Example

- The following example tenant-isolation policy passes policy evaluation to subsequent tiers for flows that are deemed acceptable and **denies** all other flows. For tenant-1 endpoints, it passes ingress flows from other tenant-1 endpoints and the `ingress-nginx` namespace. All other ingress flows are denied. Similarly it passes egress flows to other tenant-1 endpoints, TCP port 443 (https) and UDP port 53 (dns). All other egress flows are denied.
- The tenant-isolation policy is a high-level guardrail and is deployed in the security tier.



```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: security.tenant-01-isolate
spec:
  tier: security
  order: 2
  selector: tenant == "tenant-1"
  namespaceSelector: ''
  serviceAccountSelector: ''
  ingress:
    - action: Pass
      source:
        selector: tenant == "tenant-1"
        destination: {}
    - action: Pass
      source:
        namespaceSelector: projectcalico.org/name == "ingress-nginx"
        destination: {}
    - action: Deny
      source: {}
      destination: {}
  egress:
    - action: Pass
      source: {}
      destination:
        selector: tenant == "tenant-1"
    - action: Pass
      protocol: TCP
      source: {}
      destination:
        ports:
          - '443'
    - action: Pass
      protocol: UDP
      source: {}
      destination:
        ports:
          - '53'
    - action: Deny
      source: {}
      destination: {}
  types:
    - Ingress
    - Egress
```

ingress rule 1: pass ingress traffic from other tenant-1 workloads

ingress rule 2: pass ingress traffic from ingress-nginx namespace

ingress rule 3: deny all ingress traffic

egress rule 1: pass egress traffic to other tenant-1 workloads

egress rule 2: pass egress traffic to port 443

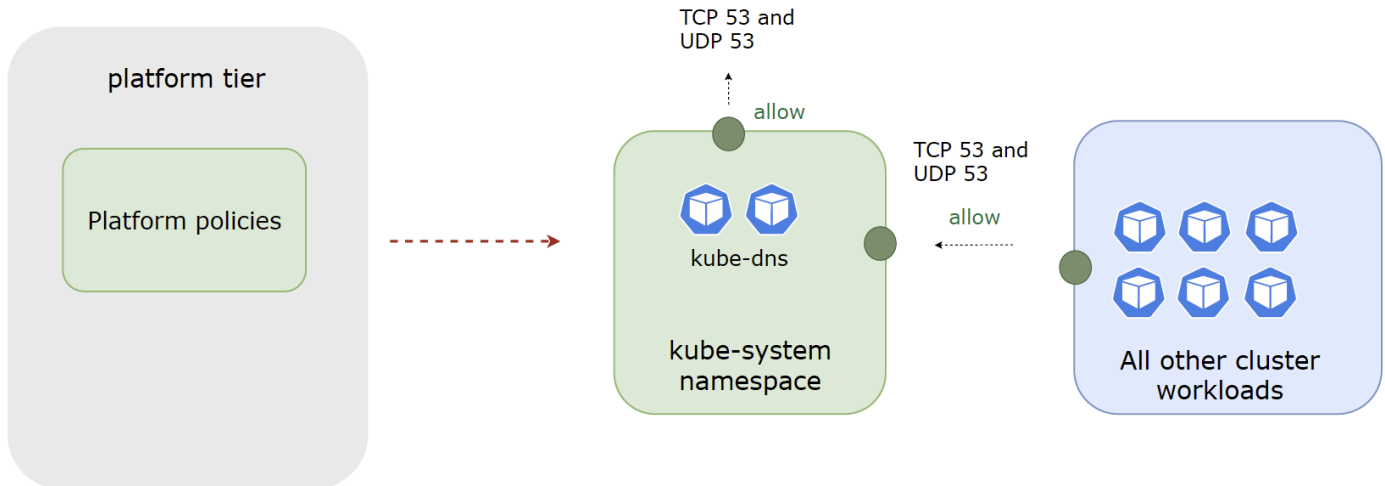
egress rule 3: pass egress traffic to udp port 53

egress rule 4: deny all egress traffic

Video Link: [Tenant-isolation Policy](#)

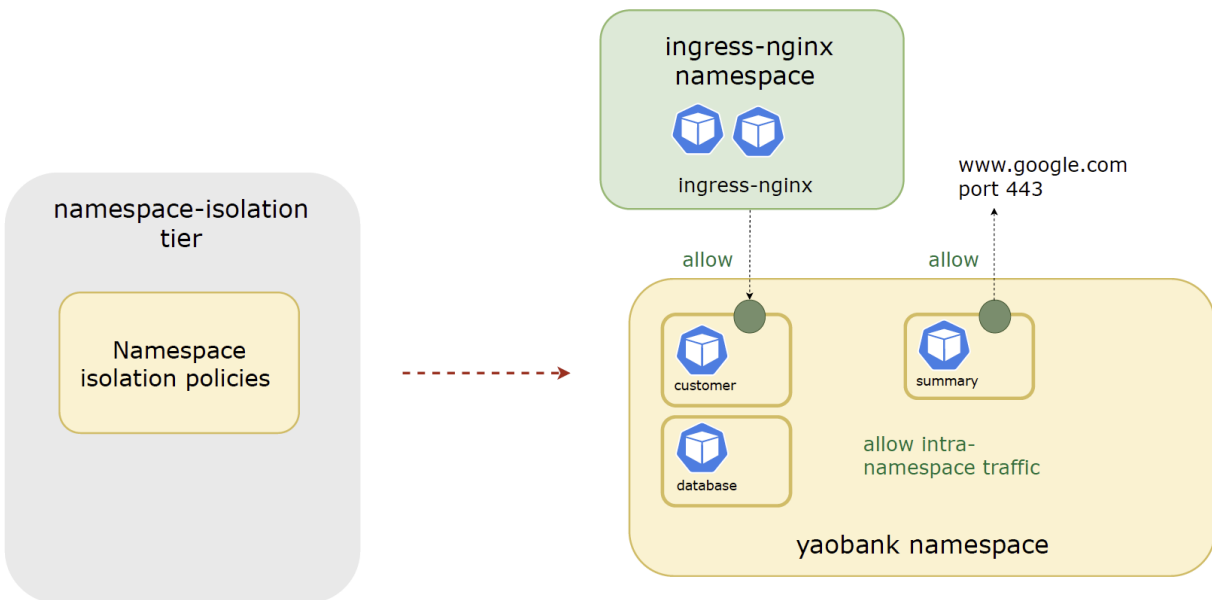
Platform Policy Example

- The following example platform policy allows traffic from all workloads to the “kube-dns” pods over TCP and UDP port 53. Similarly, “kube-dns” is permitted to receive these ingress flows on the same port. Defining such a policy in the platform tier, globally, eliminates the need for creating rules for individual workloads.
- The platform-tier policies ensure that applications can communicate with the required platform or shared infrastructure components to function in a Kubernetes cluster.



Namespace-isolation Policy Example

- The following example namespace-isolation policy permits intra-namespace flows for all workloads in the "yaobank" namespace. It also allows ingress flows from the "ingress-nginx" endpoints in the "ingress-nginx" namespace to the "customer" endpoints and egress flows to www.google.com on port 443 from the "summary" endpoints.
- The namespace-isolation policy is a low-level control that is implemented to protect the endpoints in the namespace.



```
apiVersion: projectcalico.org/v3
kind: NetworkPolicy
metadata:
  name: namespace-isolation.tenant-1-yaobank
  namespace: yaobank
  labels:
    projectcalico.org/tier: namespace-isolation
spec:
  tier: namespace-isolation
  order: 100
  selector: ''
  serviceAccountSelector: ''
  ingress:
    - action: Allow
      source:
        selector: all()
        destination: {}
    - action: Allow
      protocol: TCP
      source:
        selector: app.kubernetes.io/name == "ingress-nginx"
        namespaceSelector: projectcalico.org/name == "ingress-
nginx"
      destination:
        selector: app == "customer"
      ports:
        - '80'
  egress:
    - action: Allow
      source: {}
      destination:
        selector: all()
    - action: Allow
      protocol: TCP
      source:
        selector: app == "summary"
      destination:
        ports:
          - '443'
        domains:
          - www.google.com
  types:
    - Ingress
    - Egress
```

ingress rule 1: allow ingress traffic from other endpoints in the "yaobank" namespace

ingress rule 2: allow ingress traffic from the "ingress-nginx" endpoints to the "customer" endpoints on TCP port 80

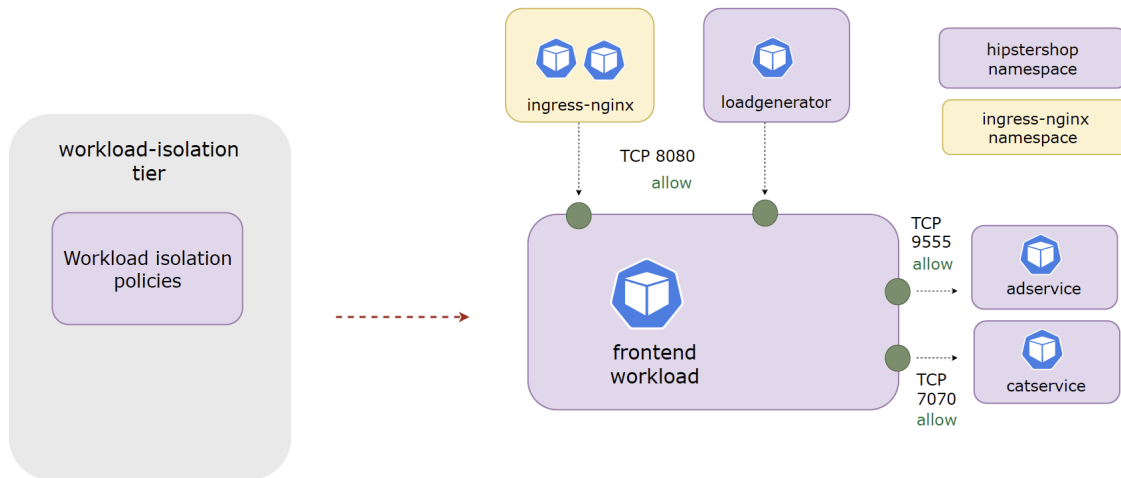
egress rule 1: allow egress traffic from other endpoints in the "yaobank" namespace

egress rule 2: allow egress traffic to "www.google.com" on port 443 from the "summary" endpoint

Video Link: [Namespace-isolation Policy](#)

Workload-isolation Policy Example

- The following example workload-isolation policy applies to the “frontend” endpoint in the “hipstershop” namespace. It allows ingress traffic to the endpoint from the “loadgenerator” pod in the same namespace and from the “ingress-nginx” pod in the “ingress-nginx” namespace on TCP port 8080. The policy also allows egress traffic to other endpoints/services in the same namespace which have been omitted for brevity.
- The workload-isolation policy is a low-level guardrail implemented to protect sensitive workloads even from other workloads in the same namespace.



```
apiVersion: projectcalico.org/v3
kind: NetworkPolicy
metadata:
  name: workload-isolation.frontend
  namespace: hipstershop
  labels:
    projectcalico.org/tier: workload-isolation
spec:
  tier: workload-isolation
  order: 100
  selector: app == "frontend"
  serviceAccountSelector: ''
  ingress:
    - action: Allow
      protocol: TCP
      source:
        selector: app == "loadgenerator"
      destination:
        ports:
          - '8080'
    - action: Allow
      protocol: TCP
      source:
        selector: app.kubernetes.io/name == "ingress-nginx"
        namespaceSelector: projectcalico.org/name == "ingress-nginx"
      destination:
        ports:
          - '8080'
  egress:
    - action: Allow
      protocol: TCP
      source: {}
      destination:
        selector: app == "adservice"
        ports:
          - '9555'
    - action: Allow
      protocol: TCP
      source: {}
      destination:
        selector: app == "catservice"
        ports:
          - '7070'
  ## rest of the rules were omitted for brevity
  types:
    - Ingress
    - Egress
```

ingress rule 1: allow ingress traffic from the "loadgenerator" endpoint in the same namespace on port 8080

ingress rule 2: allow ingress traffic from the "ingress-nginx" app in the "ingress-nginx" namespace on port 8080

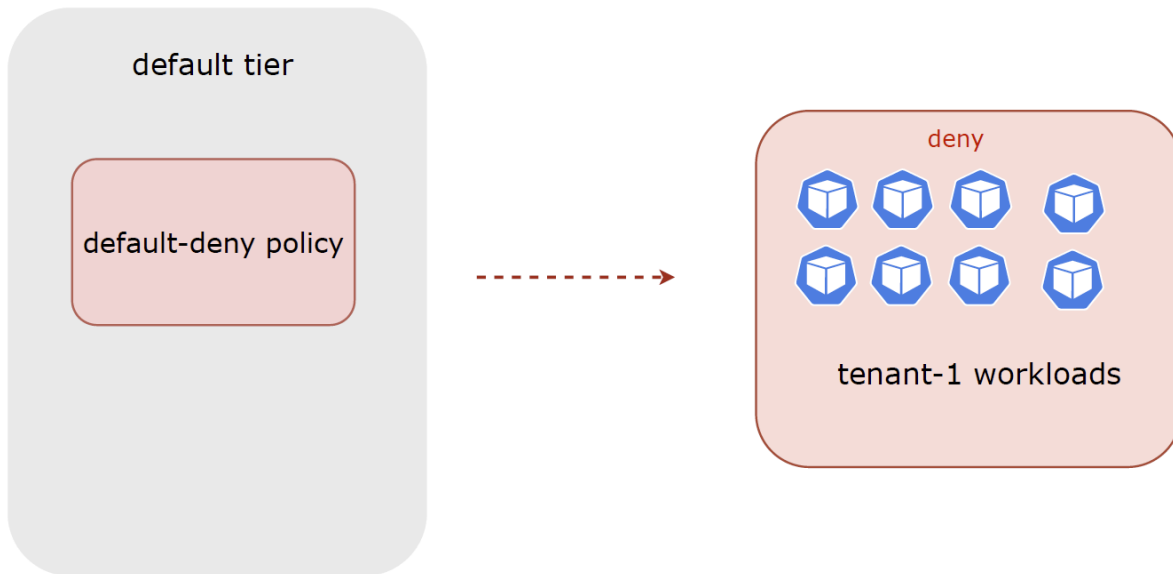
egress rule 1: allow egress traffic to the "adservice" endpoint in the same namespace on port 9555

egress rule 2: allow egress traffic to the "catservice" endpoint in the same namespace on port 7070

Video Link: [Workload-isolation Policy](#)

Deny Policy Example

- The following 'deny' policy blocks all flows for the specified tenant-1 endpoints. Any flows that need to be allowed for tenant-1 workloads should be permitted in the preceding tiers.
- The "deny" policy enforces a zero-trust network security posture for the scoped endpoints.



```
apiVersion: projectcalico.org/v3
kind: GlobalNetworkPolicy
metadata:
  name: default.tenant-1-deny
spec:
  tier: default
  order: 1
  selector: tenant == "tenant-1"
  namespaceSelector: ''
  serviceAccountSelector: ''
  ingress:
    - action: Deny
      source: {}
      destination: {}
  egress:
    - action: Deny
      source: {}
      destination: {}
  types:
    - Ingress
    - Egress
```

ingress rule 1: deny all ingress

egress rule 1: deny all egress

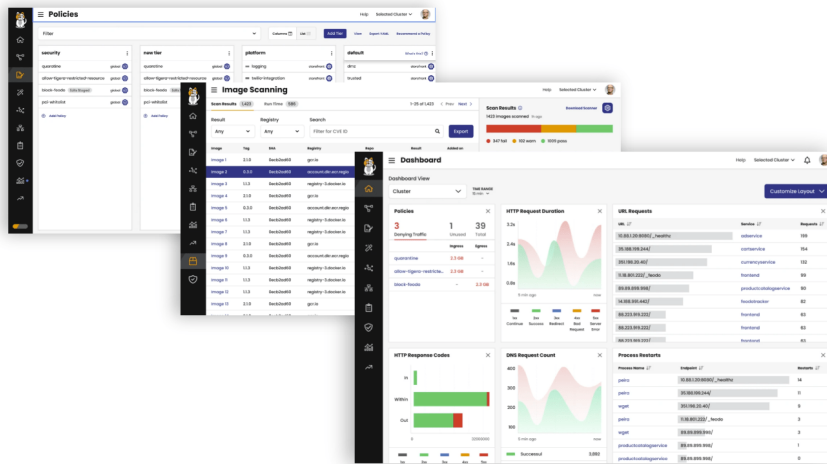
Video Link: [Deny Policy](#)

Summary

- **Policy models meet compliance and organizational requirements:** A hierarchical policy model ensures that security policies for micro-segmentation align with compliance regulations and organizational objectives.
- **Breaking down complex tasks:** The model helps divide the challenging task of microsegmentation into manageable domains, allowing for a phased approach to implementation.
- **Alignment with governance structures:** The model stipulates roles and responsibilities, ensuring teams are aligned on who implements what policies at different stages, fostering effective collaboration.
- **Consistency and fault tolerance:** It ensures consistent application of security policies across environments, preventing misconfigurations from impacting the entire system by limiting the scope of potential errors.
- **Scalability and demonstration of value:** The policy model supports GitOps-based workflows and DevSecOps practices, enabling scaling while also helping to showcase operational effectiveness and compliance to management.

Video Link: [Learning Track Summary](#)

Connect to Staged Demo Environment



To get access to staged demo environment, visit our Interactive Training page:

<https://www.tigera.io/interactive-training/>



Tigera, Inc.

2890 Zanker Rd, Suite 205

San Jose, CA 95134

www.tigera.io

"Tigera", the Tigera logo, Calico and Calico Cloud are trademarks of Tigera, Inc. All rights reserved. Other trademarks are the property of their respective owners. For more information, email us at contact@tigera.io.

Tigera. San Jose, CA | San Francisco, CA | Cork, Ireland | Vancouver, Canada | London, UK

Copyright © 2024 Tigera, Inc. All rights reserved