ADR-010: Zero-Trust Network Policies

Submitters

- Luke Doyle (D00255656)
- Hafsa Moin (D00256764)

Change Log

- approved 2026-02-13

Referenced Use Case(s)

- UVote Network Security Requirements - Internal traffic isolation and lateral movement prevention across the UVote microservices deployment.

Context

The microservices architecture (ADR-008) deploys six services and a PostgreSQL database within a single Kubernetes namespace. Without network policies, any pod can communicate with any other pod. A compromised service could directly access the database, impersonate other services, or exfiltrate data to external hosts. A network security model is needed to restrict communication to only the paths required by the application architecture.

The project research identifies lateral movement as a key threat: once an attacker compromises a single service, they can freely access all other services and the database unless network-level restrictions are in place. Traditional perimeter security that firewalls only the cluster boundary does not protect against compromised internal pods.

The approach is grounded in the NIST Zero Trust Architecture (SP 800-207) principle that no implicit trust is granted based on network location. Applied to Kubernetes, this means no pod-to-pod communication is permitted by default, and every allowed path is explicitly declared.

Proposed Design

Policy architecture:

Twelve NetworkPolicy resources are defined across five YAML files, following a numbered naming convention to make the order and purpose of each file clear.

Policy	File	Type	Purpose
default-deny	`00-default-`	Ingress + Egress	Deny all traffic; zero-trust baseline

	`deny.yaml`		
allow-dns	`01-allow-dns.yaml`	Egress	DNS resolution to kube-system:53
allow-to-database (ingress)	`02-allow-to-database.yaml`	Ingress	6 services to PostgreSQL:5432
allow-database-egress	`02-allow-to-database.yaml`	Egress	Bidirectional for DB connections
allow-from-ingress	`03-allow-from-ingress.yaml`	Ingress	Nginx Ingress to 6 exposed services
allow-to-audit (ingress)	`04-allow-audit.yaml`	Ingress	6 services to audit-service:8005
allow-audit-egress	`04-allow-audit.yaml`	Egress	Bidirectional for audit connections

Label convention:

All policies target pods using the `app: <service-name>` label. Policy targeting is per-service, meaning a policy applied to the voting-service has no effect on the auth-service or any other service.

Configuration:

- Policy API: `networking.k8s.io/v1` (standard Kubernetes, no Calico-specific CRDs)
- Enforcement: Calico CNI (iptables data plane, see ADR-004)

- Namespace: `uvote-dev`

Key implementation note:

In a default-deny architecture, both the sender's egress and the receiver's ingress must be explicitly allowed for a connection to succeed. Egress policies are always paired with their corresponding ingress policies within the same file to prevent silent failures from a missing egress rule. This requirement is documented in full in `NETWORKSECURITY.md`.

Integration points:

- ADR-003 (Kubernetes): policies deployed to the Kind cluster
- ADR-004 (Calico): Calico CNI enforces all policies
- ADR-008 (Microservices): each service targeted by label-based policy selectors
- ADR-014 (DB Users): network policies complement per-service database permissions as separate, independent layers

Considerations

Default allow with explicit deny (not selected): Allowing all traffic by default and adding deny rules for known-bad paths requires no policies initially and allows everything to work immediately. It was not selected because any new pod has full network access by default, unknown threats are not mitigated, and a compromised pod can reach all services and the database. This reactive model is exactly what zero-trust architecture is designed to replace.

Perimeter security only (not selected): Protecting only the cluster boundary while allowing unrestricted internal traffic is a familiar model from traditional infrastructure. It was not selected because it provides no lateral movement prevention. A compromised internal pod has unrestricted access to all other services and the database. This is the castle-and-moat model that NIST SP 800-207 identifies as insufficient for modern distributed systems.

Service mesh with Istio (not selected): A full service mesh would provide automatic mutual TLS between all services, L7-aware traffic policies, circuit breaking, and built-in observability. It was not selected because the resource overhead is prohibitive at this scale: sidecar proxies consume approximately 100MB per pod, meaning 12 sidecars across 6 services and 2 replicas would require approximately 1.2GB of RAM for proxies alone. Istio also has a steeper learning curve than Kubernetes itself and would have consumed a disproportionate share of the Stage 1 timeline. mTLS is noted as a future enhancement for production.

Bidirectional requirement complexity: The most operationally significant aspect of the default-deny model is that both the sender's egress and the receiver's ingress must be explicitly permitted. Missing an egress policy causes silent connection timeouts that are harder to diagnose than an explicit rejection. This is mitigated by always defining egress and

ingress policies together in the same file and documenting the requirement prominently in `NETWORKSECURITY.md`.

New service onboarding: Adding a service to the cluster requires updating two to three policy files. An operational guide covering this process is maintained in `NETWORKSECURITY.md`.

Decision

Zero-trust network policies using Calico NetworkPolicy resources were selected as the network security model for UVote.

The decision follows directly from the threat model. Lateral movement is identified as a key risk: without internal network restrictions, a single compromised service gives an attacker access to the database, other services, and the ability to exfiltrate data externally. The default-deny baseline eliminates this risk by ensuring that no pod has network access unless it is explicitly granted.

The standard `networking.k8s.io/v1` API is used throughout, with no Calico-specific CRDs. This keeps all policies portable to any compliant CNI and ensures they are readable and auditable without knowledge of Calico internals.

Network policies operate as one layer within a broader defence-in-depth model. The full stack is: network isolation (this ADR), per-service database users (ADR-014), parameterised queries, immutability triggers (ADR-002), and hash-chain audit logs (ADR-007). Each layer operates independently, so a failure or bypass at one layer does not compromise the others. In practical terms, the frontend-service has no network path to PostgreSQL, meaning even a fully compromised frontend cannot reach the database directly, regardless of what SQL it might attempt to execute.

The trade-off of 12 policies and associated operational complexity over an open network is accepted. The security benefit, preventing lateral movement, protecting the database, and blocking external exfiltration, is significant relative to the management overhead. All policies are version-controlled YAML with clear labels, comments, and full documentation in `NETWORKSECURITY.md`.

A review is scheduled for the end of Stage 2 (April 2026) to evaluate adding mTLS via a service mesh for production deployments.

Other Related ADRs

- ADR-003: Kubernetes Platform - cluster on which policies are deployed
- ADR-004: Calico CNI - CNI that enforces the policies

- ADR-008: Microservices Architecture - service boundaries that policies map to
- ADR-014: Per-Service Database Users - complementary access control layer

References

- NIST SP 800-207 Zero Trust Architecture
- Kubernetes NetworkPolicy Documentation
- EdgeX Foundry ADR Template