Beyond Kubernetes: Service Mesh Strategies for Microservices Deployment & Monitoring with Istio

Hemamalini Nithyanandam

AGENDA

Beyond Kubernetes: Service Mesh Strategies for Microservices Deployment & Monitoring with Istio
01 | Kubernetes powerful but not Enough
02 | Istio – The Traffic Cop
03 | Istio Under the Hood - Components
04 | Deployment Strategies
05 | Resiliency & Scalability with Istio
06 | From Fragile to Fault-Tolerant: The Architecture That Scales
07 | Demo Time

Who Am I ....

Professional

- Software Designer in HP
- 12+ years IT experience
- AWS Solutions Architect
- Data Engineer
  &
- AI Enthusiast

Personal

- Punnagai Foundation
- Certified Yoga Trainer
- Blogger

https://www.linkedin.com/in/hemamalini-nithyanandam/

Life Before the Mesh

"Let me take you back to a time when our micro services empire was growing faster than we could manage..."

Struggle – Resilience ,Complexity at Scale

- As our services grew, so did the chaos.
- By breaking monoliths into micro services, we gain agility, but we also inherit new forms of complexity:
- Service Explosion – many services with interdependencies
- Distributed Failures: Partial outages, cascading failures
- Networking – Dynamic IP, Service Discovery, DNS flakiness
- Debugging is Distributed: Tracing a request across 12 services is no joke
- Security Scattering: TLS, RBAC, authN/z scattered across teams & codebases

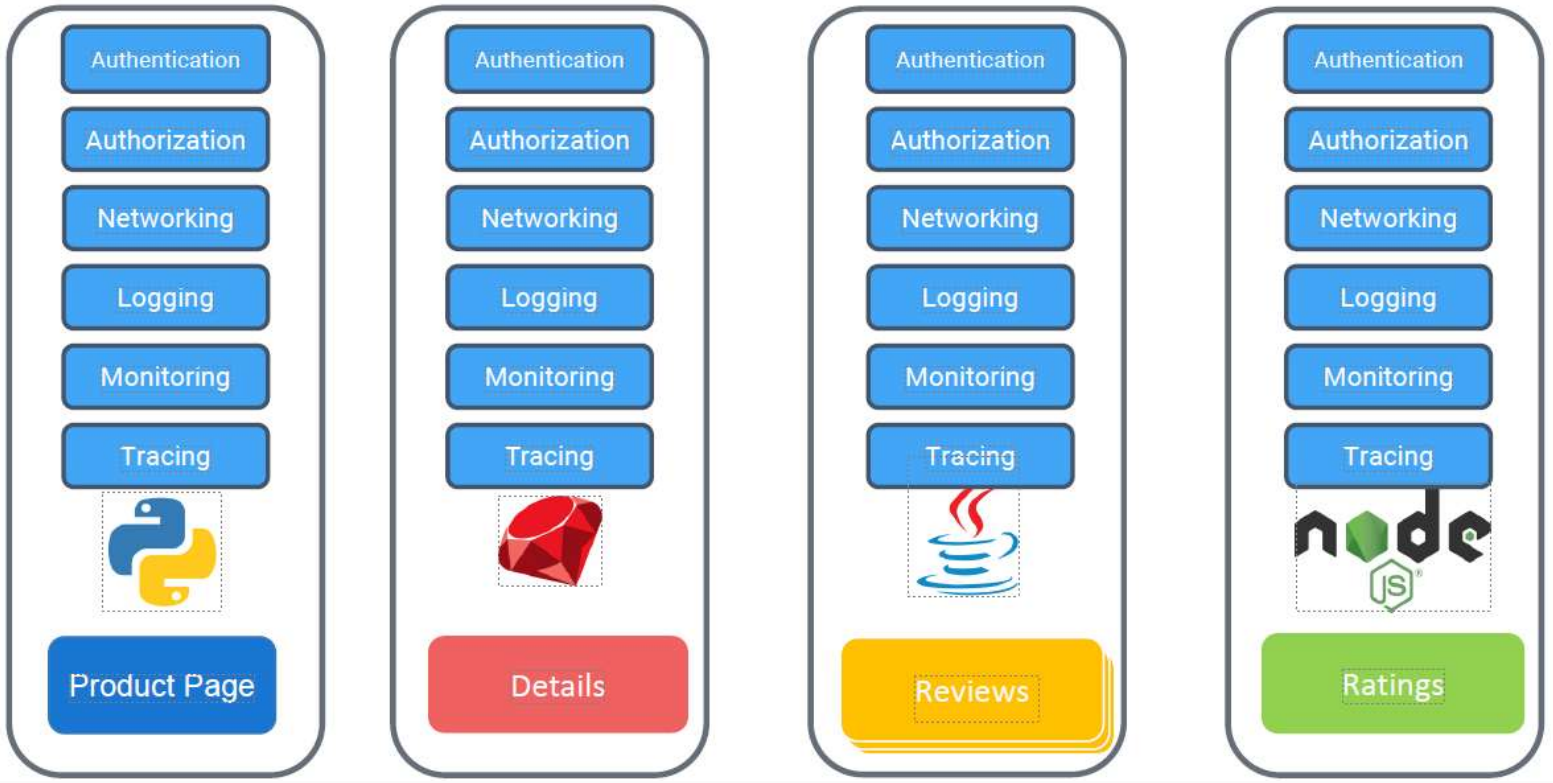
Kubernetes: Powerful But Not Enough

- Infrastructure automation – container orchestration, Scheduling, Auto scaling(HPA), Networking (service,ingress), Self-healing (probes,restarts)

Where Kubernetes doesn't cover well

- Fine-grained **traffic management** (e.g., canary, routing by headers)
- **Secure service-to-service communication** (mTLS, identity-based)
- Retries, timeouts, circuit breakers
- Rich **telemetry** (distributed tracing, granular metrics)
- **Policy enforcement** at runtime (e.g., denylist/blocklist, rate limiting)

A Problem: Fat Microservices



Hsource ttps://kodekloud.com

Istio - The traffic cop

"Kubernetes gave us the canvas. But to paint a resilient architecture, we needed a brush — that's where Istio comes in."

Service Mesh: Taming the Microservices Jungle

- Dedicated infrastructure layer for managing service-to-service communication in a micro services architecture — abstracting away concerns like traffic control, observability, and security without changing application code.
- Think of a service mesh as a smart *network layer* that travels with your services, ensuring they talk to each other *securely, reliably, and* transparently.
- Without service mesh , we need to handle retry logics, circuit breakers,timeouts,TLS encryption,Logging/Tracing headers, Rate limiting,access control policies in application

Istio – Companion to Kubernetes

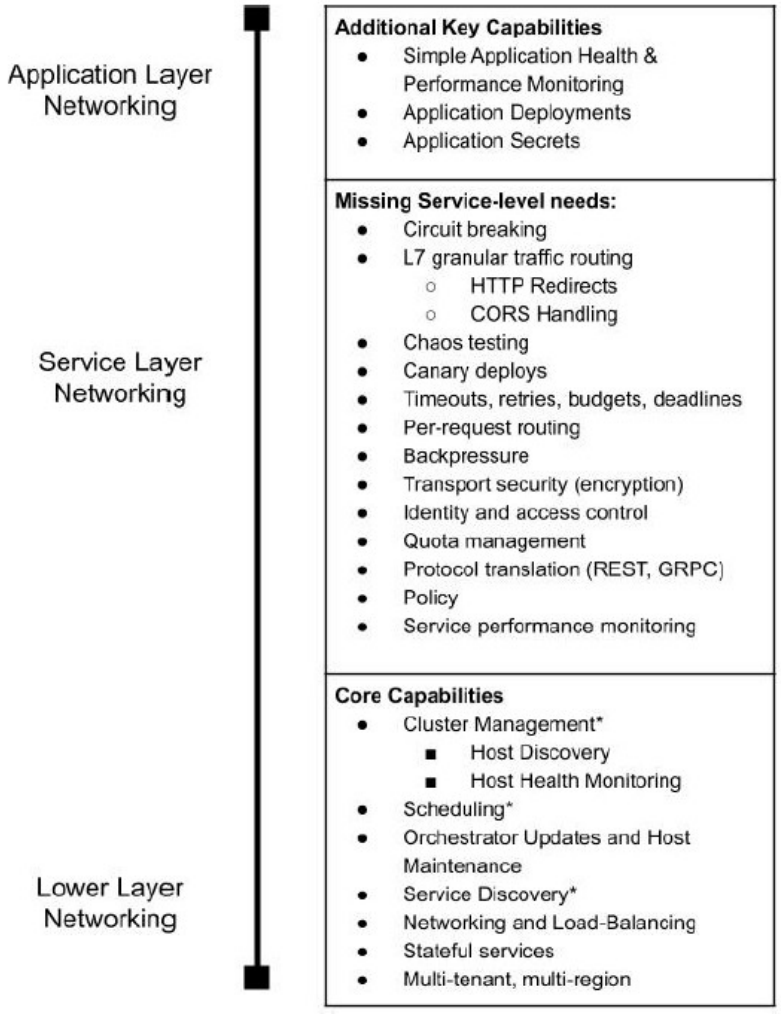
- Istio – open source service mesh
- Kubernetes provides the "where"(infrastructure) and Istio provides the "how" (traffic,policy,security)

Pain points	How Istio helps
Inconsistent mTLS & security	Automatic encryption between services with mutual TLS
Manual traffic shaping	Canary, A/B testing, blue-green via simple config
Debugging distributed systems	Built-in distributed tracing (Jaeger), metrics (Prometheus), logs
Fault tolerance logic in code	Declarative config for retries, timeouts, circuit breakers
Policy enforcement gaps	Fine-grained RBAC and rate limiting via policies

OSI Network Model

Layer		Protocol data unit (PDU)	Function	Examples
7	Application	Data	High-level protocols such as for resource sharing or remote file access	HTTP, FTP, POP3, SMTP, WebSocket
6	Presentation		Data provisioning and encryption	ASII, EBCDIC, JPEG, MIDI
5	Session		Managing communication sessions	RPC, PAP, L2TP, gRPC
4	Transport	Segment Datagram	Reliable transmission of data segments between points on a network	TCP, UDP, SCTP, Порты
3	Network	Packet	Structuring and managing a multi-node network	IPv4, IPv6, IPsec, AppleTalk, ICMP
2	Data link	Frame	Transmission of data frames between two nodes connected by a physical layer	PPP, IEEE, 802.22, Ethernet, DSL, ARP, network card
1	Physical	Bit Symbol	Transmission and reception of raw bit streams over a physical medium	USB, RJ (twisted copper pair, coaxial, fiber optic), radio channel

Source: https://vasexperts.com/resources/glossary/osi-network-model/



Source :https://www.oreilly.com/library/view/istio-up-and/9781492043775

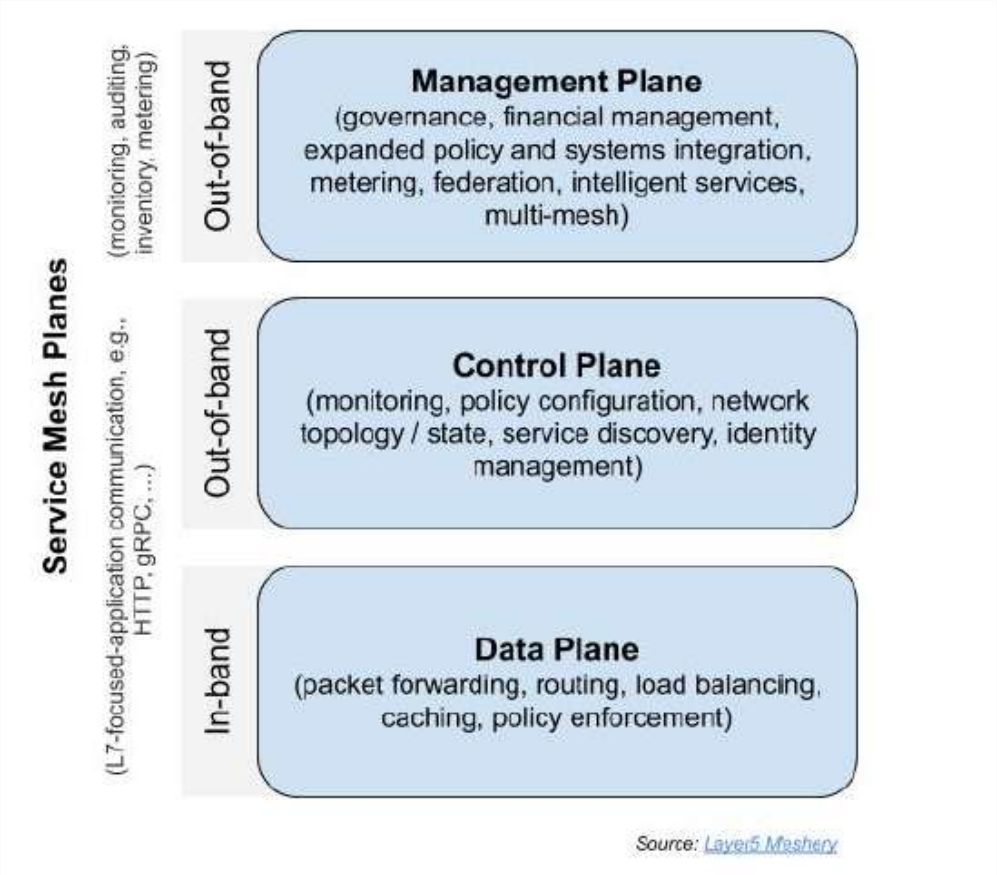
Istio – Companion to Kubernetes

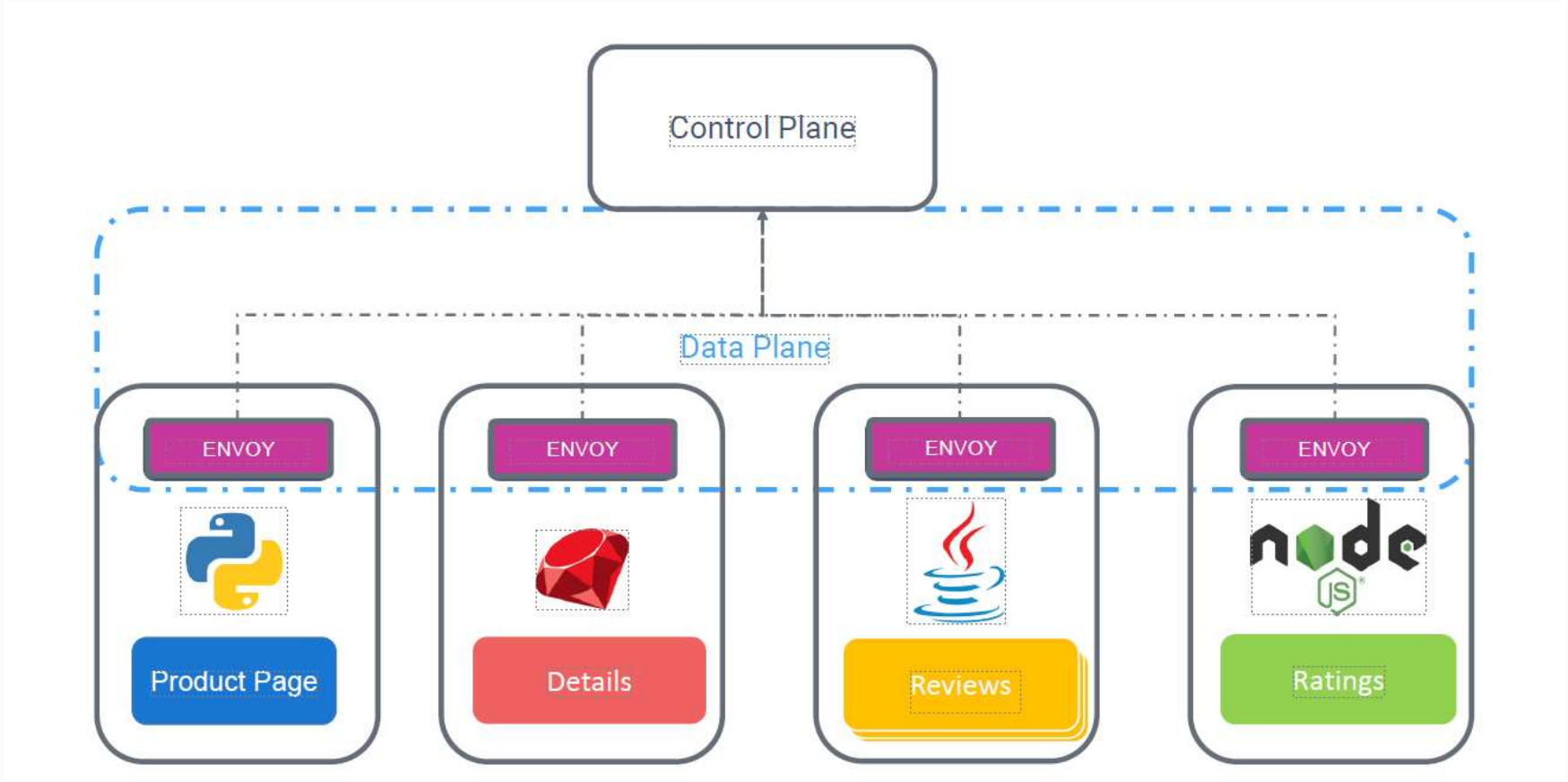
- Istio intercepts and manages traffic without app code
- Leverages side car proxies (envoy) injected into pods
- Traffic between services is routed through the mesh
- Istio can encrypt,observe and control it
- Developers can focus on features ,Istio handles resilience ,security and observability behind the scenes.
- Kubernetes schedules containers. Istio ensures they play nicely with each other.
- Sidecar - coscheduling utility containers with application containers grouped in the same logical unit of scheduling.

Istio – Under the Hood

- Envoy Proxy (Data Plane) - Sidecar injected into each pod, Handles traffic in/out of the service, Performs routing, TLS, retries, tracing, etc.
- Istiod (Control Plane) - brain of the mesh, Pushes configuration to Envoy sidecars
- Control Plane vs. Data Plane
- Data Plane: Handles real traffic (Envoy sidecars)
- Control Plane: Configures and manages the mesh (Istiod)
- Envoy is like a personal bodyguard for each service. Istiod is mission control.

Istio – Under the Hood





Source :https://kodekloud.com

Key Terms

- Gateways - Define entry and exit points for the mesh. Unlike Kubernetes Ingress, Istio Gateways work at the L4-L7 level and configure ports, TLS, and routing.

Use case: Expose your services to external clients securely.

- Virtual Services - Abstracts routing logic for services. Allows fine-grained traffic control (e.g., routing based on headers, paths, weights).

Use case: Route 90% of traffic to v1 and 10% to v2 for canary deployment.

- Destination Rules - Define policies for traffic after routing — such as load balancing, connection pool settings, or circuit breaking.

Use case: Apply round-robin load balancing to a backend service.

circuit breaker pattern - Resilience pattern that protects services from being overwhelmed by automatically stopping requests to a failing service when it detects repeated failures or high load.

Key Terms

- Subsets - Specify named groups of service versions, usually based on labels (e.g., version: v1, v2).

Use case: Enable routing rules to target specific versions of a service.

- Timeouts - Configure maximum duration a client should wait for a response before timing out.
- Retries - Specify how and when Istio should retry failed requests to services.
- Request Routing - Control how requests are routed to different services or versions based on conditions.

Istio – Data Plane

- Intercepts and handles all service traffic within the mesh.
- Sidecar proxies (Envoy) intercept every request
- Transparent traffic redirection
- Handles:
  - Health checking
  - Service discovery
  - Load balancing
  - Routing
  - Authentication & Authorization
  - Observability (metrics, logs, tracing)

Istio – Control Plane (Brain)

- Manages and configures the data plane without touching traffic.
- Combines stateless sidecars into a cohesive service mesh
- Supports real-time updates as services scale/move
- Provides centralized APIs for:
  - Traffic routing & resilience policies
  - Security (mTLS, cert rotation, workload identity)
  - Telemetry and observability config
  - Quotas and usage restrictions
  - Authentication & Authorization
  - Observability (metrics, logs, tracing)

mTLS (Mutual Transport Layer Security) is a security protocol where both the client and server authenticate each other using digital certificates before any data is exchanged.

Istio – Control Plane Components(Behaviour)

Component	Description
istiod	Unified control plane component; replaces Pilot, Citadel, Galley, and Mixer
Istio CA	Issues and rotates mTLS certificates (built into `istiod`)
xDS APIs	Protocols used by istiod to push config to Envoy proxies
Envoy Filters	Used to customize proxy behavior (e.g., WASM-based telemetry, auth filters)
Webhooks	Injects Envoy sidecars into pods automatically

Istio – Data Plane Components(Traffic manager)

Component	Description
Envoy Proxy	Sidecar proxy deployed with each service; enforces traffic rules
Service Proxy (istio-proxy)	Envoy with Istio config to route traffic, apply policies, collect telemetry
Ingress Gateway	Handles external inbound traffic into the mesh
Egress Gateway	Manages traffic leaving the mesh to external services

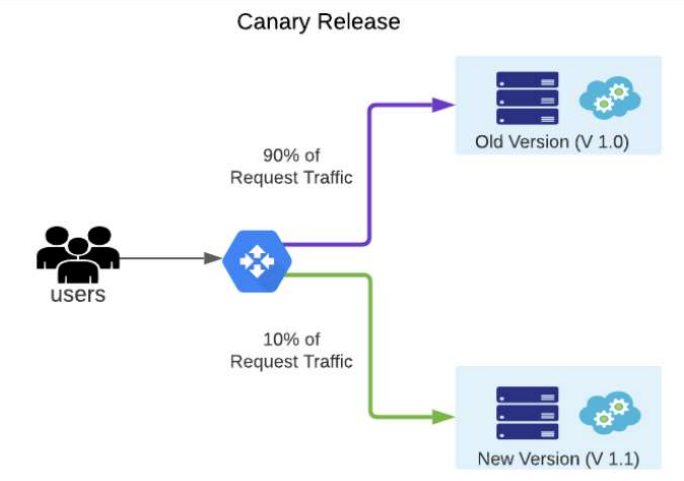
Istio – Add-ons

Component	Description
Kiali	Visualizes mesh topology and traffic
Prometheus	Collects metrics from Envoy proxies
Grafana	Visualizes Prometheus metrics
Jaeger/Zipkin	Traces distributed requests across services
Cert-Manager	(Optional) Issues TLS certificates for Istio

Service Deployment Patterns

1. Canary releases –

- Gradually roll out new version to a small percentage of users.
- Example: 10% traffic to v2, 90% to v1
- Allows safe validation before full rollout
- Easy rollback if issues arise

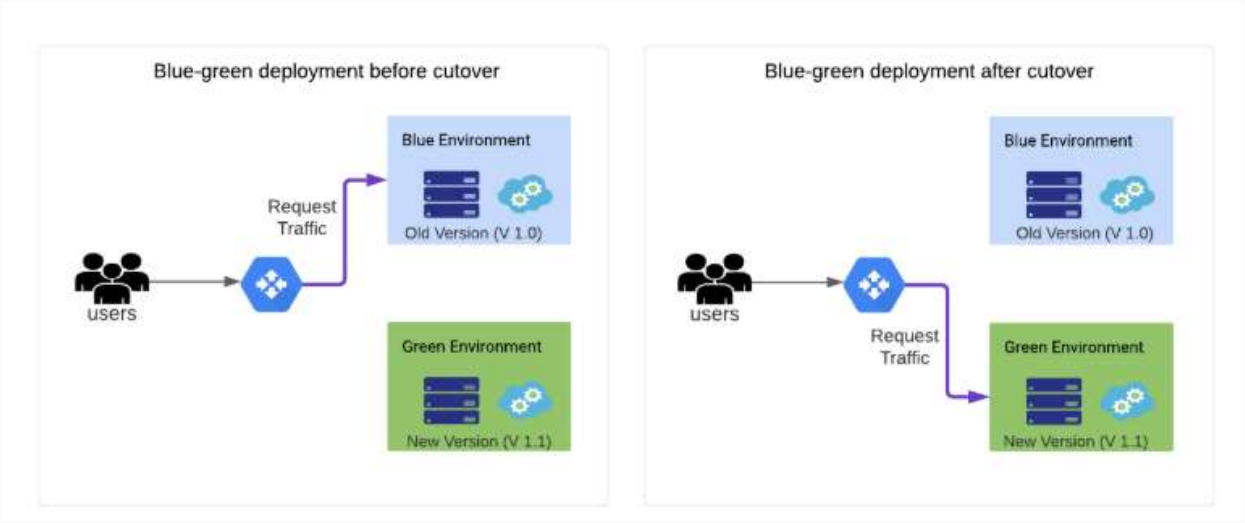


https://www.w6d.io/blog/canary-release-vs-blue-green-deployments/

Service Deployment Patterns

2. Blue-Green Deployment –

- Run two parallel environments (blue = current, green = new).
- Shift traffic from blue to green instantly
- Zero downtime & easy rollback



https://www.w6d.io/blog/canary-release-vs-blue-green-deployments/

Service Deployment Patterns

3. A/B Testing–

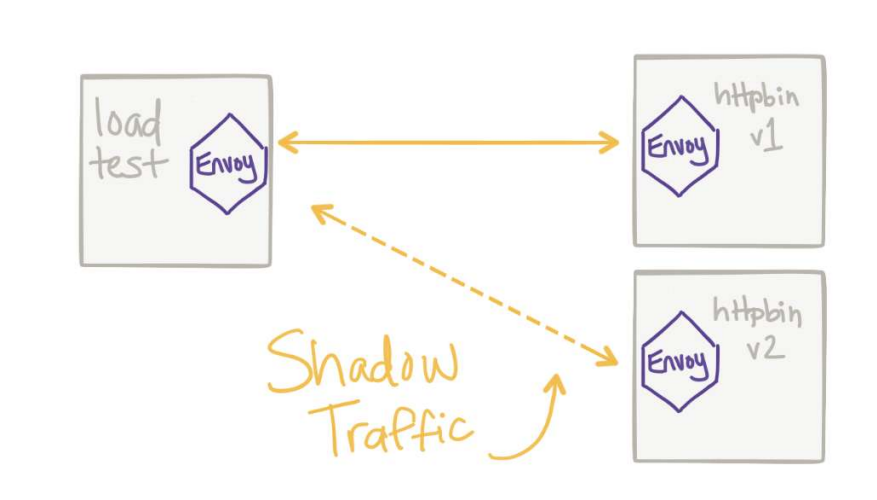
- Simultaneous Versions: Direct different subsets of users to separate versions.
- Measure differences in user engagement or performance.
- Key Use: UX experiments and feature comparisons.

https://blog.christianposta.com/deploy/blue-green-deployments-a-b-testing-and-canary-releases/

Service Deployment Patterns

4. Shadow (Mirroring) Deployments–

- Traffic Mirroring: Duplicate live traffic to the new service version without impacting production.
- Observability Focus: Validate behavior under production-like conditions.
- Key Use: Testing new features without exposing real users..



https://blog.christianposta.com/microservices/traffic-shadowing-with-istio-reduce-the-risk-of-code-release/

Service Deployment Patterns

5. Progressive Delivery–

- Automated Rollouts: Combines canary strategies with continuous monitoring and automated scaling.
- Policy Driven: Uses real-time metrics to adjust deployment stages.
- Key Use: Safe rollouts with automatic adjustment based on performance metrics

Traffic Routing: Istio VirtualServices and DestinationRules enable dynamic traffic splitting and shifting.

Observability & Resilience: Use Istio's telemetry, retries, and circuit breaking to monitor and secure deployments.

Envoy

- High-performance, open-source L7 proxy and communication bus designed for modern cloud-native applications.
- It is the data plane used by Istio, serving as the sidecar proxy injected into each service pod.
- L4/L7 Proxy - Supports HTTP, HTTP/2, gRPC, TCP, and TLS
- Receives config from Istio's control plane (istiod) in real-time
- Advanced Traffic Management - Load balancing, routing, retries, timeouts, circuit breaking
- Observability - Emits metrics, logs, and traces (to Prometheus, Jaeger, etc.)
- A smart service mesh proxy that handles all networking for microservice — securely, reliably, and observably — without changing app code.

Scalability with Istio

- Horizontal Scaling: Add more pods to handle load (Be Present)
- Vertical Scaling: Increase pod resource limits (CPU/RAM)
- Istio config (like circuit breakers) can help prevent resource overloading

```yaml
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: llm-app-hpa
  namespace: default
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: llm-app
  minReplicas: 2
  maxReplicas: 8
  metrics:
    - type: Resource
      resource:
        name: cpu
        target:
          type: Utilization
          averageUtilization: 70
```

Istio DestinationRule with Circuit Breaking

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: llm-app
  namespace: default
spec:
  replicas: 2
  selector:
    matchLabels:
      app: llm-app
  template:
    metadata:
      labels:
        app: llm-app
    spec:
      containers:
        - name: llm-app
          image: llm-app-image:latest
          resources:
            requests:
              cpu: "500m"
              memory: "1Gi"
            limits:
              cpu: "1000m"
              memory: "2Gi"
```

```yaml
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: llm-app-dr
  namespace: default
spec:
  host: llm-app.default.svc.cluster.local
  trafficPolicy:
    connectionPool:
      http:
        http1MaxPendingRequests: 20
        maxRequestsPerConnection: 10
        maxRetries: 3
    outlierDetection:
      consecutive5xxErrors: 3
      interval: 5s
      baseEjectionTime: 30s
      maxEjectionPercent: 50
```

Load Balancing Strategies with Istio

- Istio Sidecar Proxies - Every pod has an Envoy sidecar that intercepts traffic and applies load balancing policies.
- Intelligent L7 Load Balancing , Connection Pooling & Retries , Fine-grained control over load balancing strategies
- Ingress Gateway - Entry point for external traffic into the mesh.
- TLS Termination - Handle HTTPS securely at the edge
- Traffic Control - Route traffic based on paths, headers,etc
- Egress Gateway- Manages traffic leaving the mesh to external services.
- Policy Enforcement- Define what services can access the internet
- Secure Communication -Apply mTLS or mutual TLS to outbound traffic
- Apply mTLS or mutual TLS to outbound traffic - Log external calls for compliance and security

Istio Config

```yaml
# 1. DestinationRule for fine-grained load balancing
apiVersion: networking.istio.io/v1beta1
kind: DestinationRule
metadata:
  name: llm-app-dr
spec:
  host: llm-app.default.svc.cluster.local
  trafficPolicy:
    loadBalancer:
      simple: LEAST_CONN  # RoundRobin | RANDOM | LEAST_CONN
    connectionPool:
      tcp:
        maxConnections: 100
      http:
        http1MaxPendingRequests: 100
        maxRequestsPerConnection: 10
    outlierDetection:
      consecutiveErrors: 5
      interval: 10s
      baseEjectionTime: 30s
    tls:
      mode: ISTIO_MUTUAL
```

Istio Config

```
# 2. VirtualService for routing (for Ingress or internal routing)
apiVersion: networking.istio.io/v1beta1
kind: VirtualService
metadata:
  name: llm-app-vs
spec:
  hosts:
    - llm-app.example.com
  gateways:
    - istio-ingressgateway
  http:
    - match:
        - uri:
            prefix: /chat
      route:
        - destination:
            host: llm-app.default.svc.cluster.local
            port:
              number: 8080
          weight: 100
      retries:
        attempts: 3
        perTryTimeout: 2s
        retryOn: gateway-error,connect-failure,refused-stream
      timeout: 5s
```

Istio Config

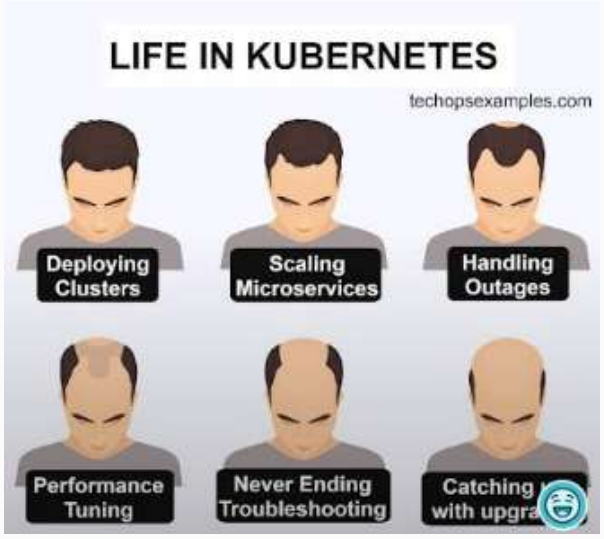
```
# 3. Gateway for LLM App ingress
apiVersion: networking.istio.io/v1beta1
kind: Gateway
metadata:
  name: llm-app-gateway
spec:
  selector:
    istio: ingressgateway
  servers:
    - port:
        number: 443
        name: https
        protocol: HTTPS
      hosts:
        - llm-app.example.com
      tls:
        mode: SIMPLE
        credentialName: llm-app-cert  # Kubernetes secret
        minProtocolVersion: TLSV1_2
```

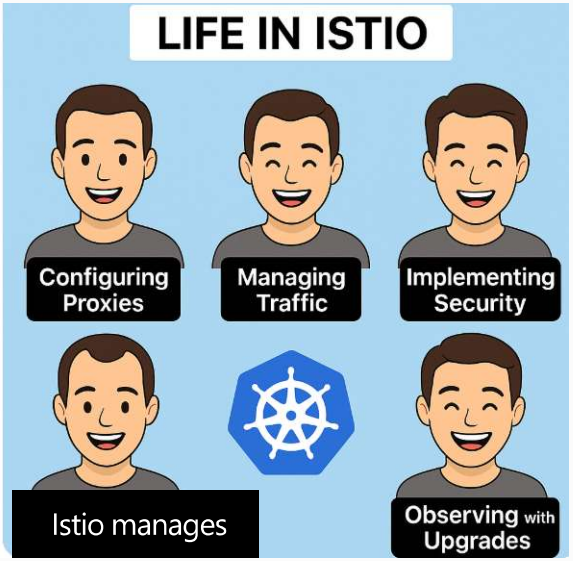
Istio Config

```yaml
# 4. ServiceEntry for external APIs (Egress example)
apiVersion: networking.istio.io/v1beta1
kind: ServiceEntry
metadata:
  name: openai-api-egress
spec:
  hosts:
    - api.openai.com
  ports:
    - number: 443
      name: https
      protocol: HTTPS
  location: MESH_EXTERNAL
  resolution: DNS
```

Istio Config

```
---
# 5. PeerAuthentication for mTLS within the mesh
apiVersion: security.istio.io/v1beta1
kind: PeerAuthentication
metadata:
  name: default
  namespace: default
spec:
  mtls:
    mode: STRICT
```





Event-Driven Architecture with Istio

- Decouple producers and consumers for scalability
- Secure communication between microservices and Kafka brokers
- Trace Kafka events and Pub/Sub flows using Prometheus, Grafana, Jaeger
- Use AuthorizationPolicy and PeerAuthentication to manage event flows
- Async Event Ingestion: Services publish events, Istio manages secure delivery
- Externalize broker access with ServiceEntry + DestinationRule
- Leverage Envoy filters for advanced Kafka protocol handling
- https://www.youtube.com/watch?v=qMkV5qeOnfg

KEDA (Event-Driven Autoscaler)

- Triggers scale-out based on queue depth (e.g., Kafka topics, Redis, RabbitMQ)
- Triggers scale-out based on queue depth (e.g., Kafka topics, Redis, RabbitMQ)
- Ensure GPU-bound LLM inference pods have guaranteed access
- In Kubernetes (or any shared compute environment), the Noisy Neighbor Problem occurs when one pod or container consumes excessive CPU, memory, or I/O, negatively affecting the performance of other pods on the same node.
- While Kubernetes enforces resource limits, Istio adds an application-layer safety net - observes, controls, and enforces limits on traffic before it becomes a resource issue.

Observability - See Everything, Miss Nothing

- Envoy Sidecars collect metrics, logs, and traces automatically
- Istio captures request latency, error rates, throughput
- Pre-configured Prometheus scraping via Istio control plane to visualize Pod-to-pod traffic, CPU/GPU utilization,
- Distributed Tracing with Jaeger / Zipkin
- Export data to Datadog, New Relic, Azure Monitor using OpenTelemetry

From Fragile to Fault-Tolerant: The Architecture That Scales

- **Design for failure** – Expect failures and use timeouts, retries ,circuit breakers. Validate behaviour with chaos testing
- **Loose Coupling & High Cohesion –** Isolate service boundaries with clear API, minimize shared dependencies, enable parallel deployment
- **Scalability First, Performs follows –** HPA,VPA, Use stateless services,load balancing
- **Caching for speed & reliability**
- **Idempotency & Safe retries -** Ensure retrying a failed request won't break the system,
- **Observability by default –** Metrics ,Dashboard ,real time alerts
- **Security –** Enforce RBAC, least privilege access
- **Async & Event driven**

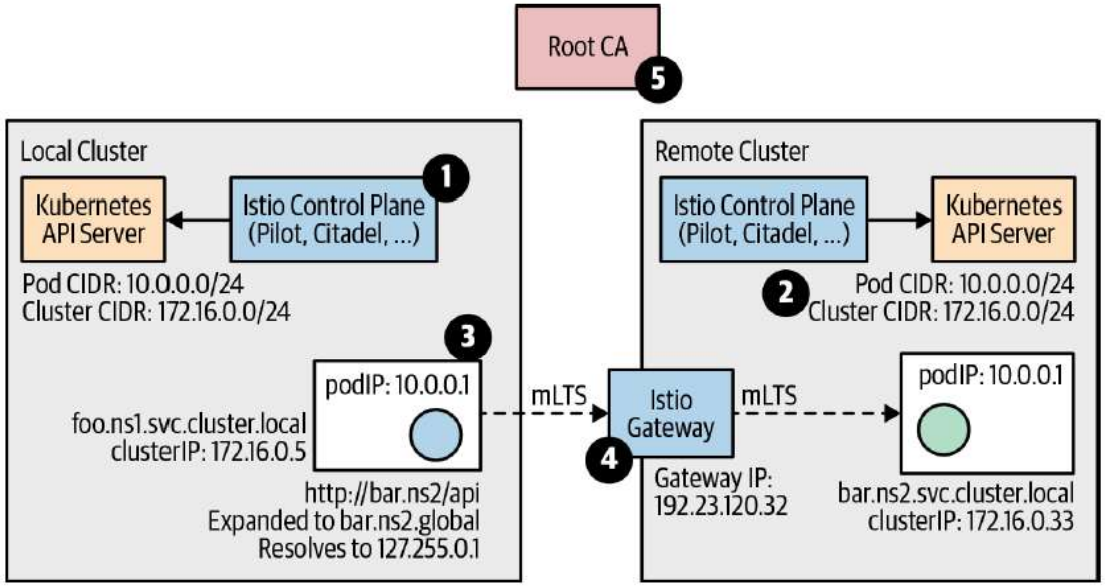
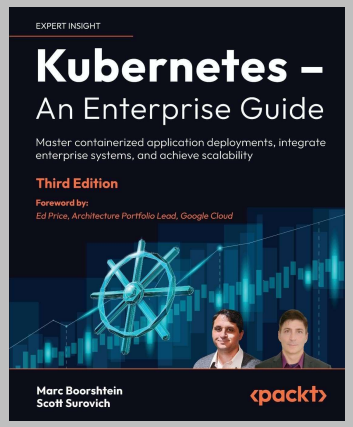


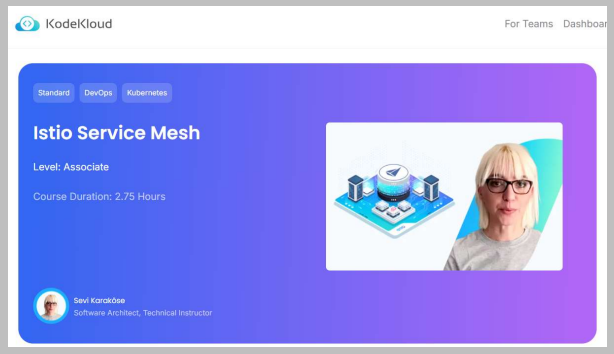
Figure 13-1. The Istio multicluster approach: a single-control-plane Istio deployment with direct connection (flat networking) across clusters

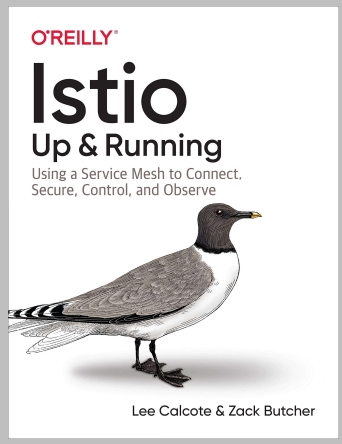
Source :https://www.oreilly.com/library/view/istio-up-and/9781492043775

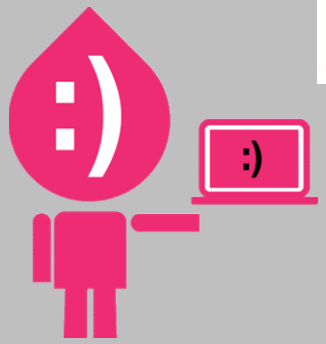
From Fragility to Fortitude: Wrapping Up the Mesh Story

- Great systems don’t just work — they heal, adapt, and thrive under pressure like us ☺
- Kubernetes provides the foundation — but resilience demands more.
- Service Mesh (Istio) empowers us with security, observability, and control — without touching code.
- Architecture patterns like caching, idempotency, retries, and async design make systems predictable under pressure.
- Linkerd, AWS App Mesh, Consul Connect









Q&A...

https://github.com/hemsush/TechXConfTalk2025