Steven Lopez
lopezstevie@gmail.com

Demystifying Kubernetes & Istio

Learn the fundamentals of microservice orchestration, traffic control, and service mesh security.

Steven Lopez | Kubernetes & Istio Architecture

The Twelve Factor - First Published 2011 by Heroku Engineers

A methodology for building SaaS apps that are portable, resilient, and easy to scale.

I. Codebase

One codebase tracked in revision control, many deploys

II. Dependencies

Explicitly declare and isolate dependencies

III. Config

Store config in the environment

IV. Backing services

Treat backing services as attached resources

V. Build, release, run

Strictly separate build and run stages

VI. Processes

Execute the app as one or more stateless processes

VII. Port binding

Export services via port binding

VIII. Concurrency

Scale out via the process model

IX. Disposability

Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity

Keep development, staging, and production as similar as possible

XI. Logs

Treat logs as event streams

XII. Admin processes

Run admin/management tasks as one-off processes

https://12factor.net/

I. Codebase	II. Dependencies	III. Config	IV. Backing services	V. Build, release, run	VI. Processes	VII. Port binding	VIII. Concurrency	IX. Disposability	X. Dev/prod parity	XI. Logs	XII. Admin processes
One codebase tracked in revision control, many deploys	Explicitly declare and isolate dependencies	Store config in the environment	Treat backing services as attached resources	Strictly separate build and run stages	Execute the app as one or more stateless processes	Export services via port binding	Scale out via the process model	Fast startup and graceful shutdown	Keep development, staging, and production as similar as possible	Treat logs as event streams	Run admin/management tasks as one-off processes
GitHub	Apache Maven npm PIP	HashiCorp Vault	mongoDB kafka Amazon S3	GitHub argo Jenkins docker JFrog	pod AWS Lambda redis	spring boot Amazon API Gateway	hpa AWS ECS	 	HashiCorp Terraform minikube	fluentd logstash Grafana DATADOG splunk>	kubectl

Steven Lopez | Kubernetes & Istio Architecture

From Netflix OSS to Kubernetes: The Evolution of Cloud-Native Infrastructure

Netflix OSS

- Zuul
- Eureka
- Ribbon
- Hystrix
- Config Server (Archaius)
- Netflix's Security Components

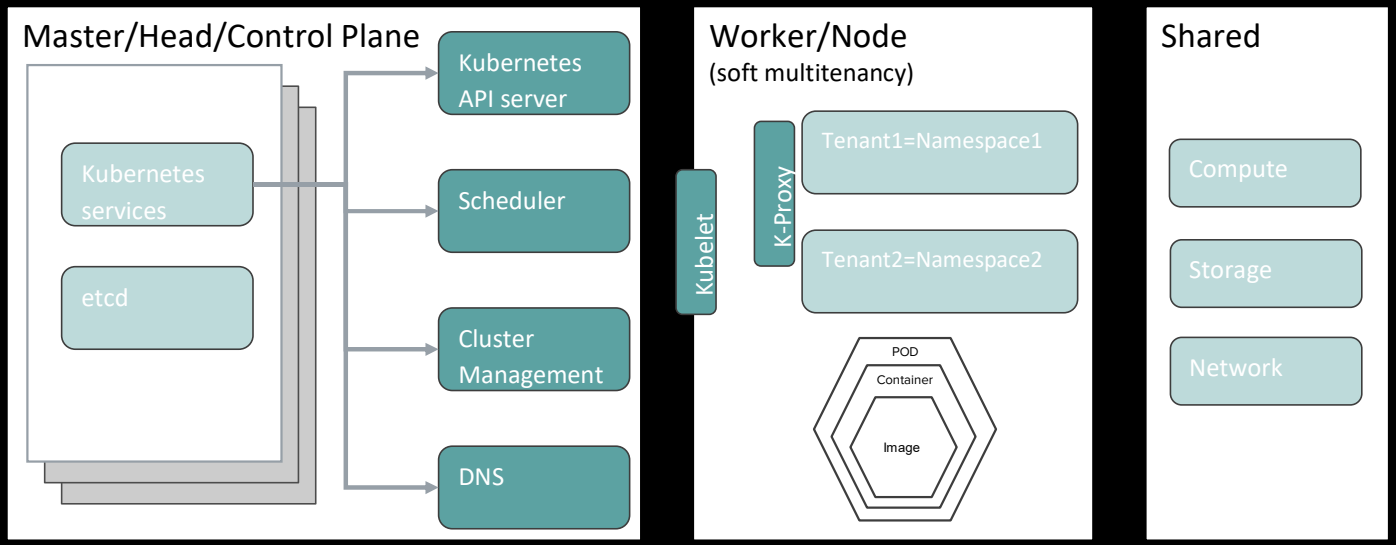
Kubernetes

- Ambassador or Kong (not native)
- Service/DNS
- Kubernetes Services
- Application Level
- ConfigMap and Secret
- RBAC

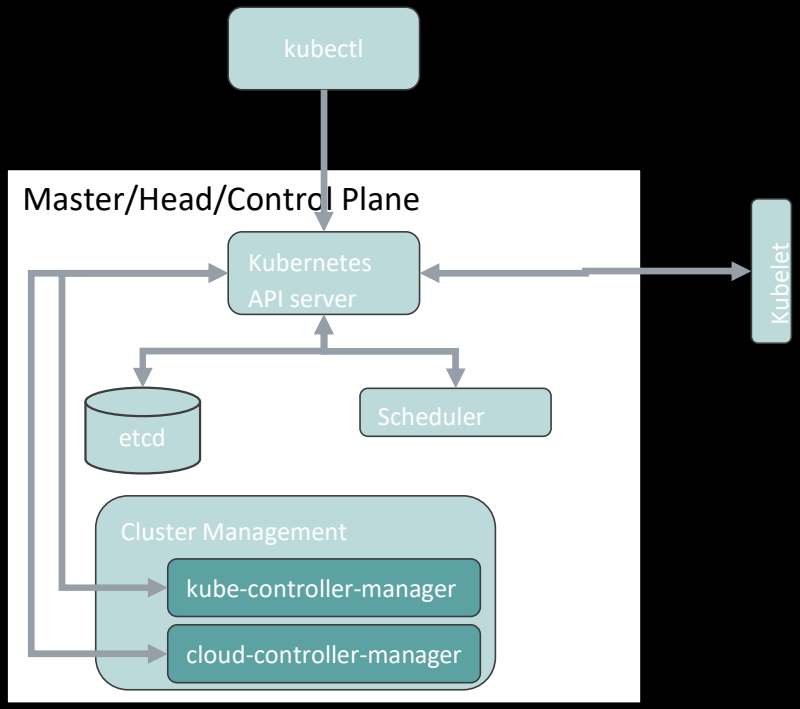
What Kubernetes Solves

1. **Application deployment consistency**
   Kubernetes standardizes how applications are packaged, deployed, scaled, and updated across any environment. It eliminates snowflake deployments and enforces declarative, versioned infrastructure.
2. **Automated scaling and healing**
   ReplicaSets and controllers continuously reconcile desired state. Kubernetes restarts failed workloads, replaces unhealthy nodes, and autos-scales based on real demand.
3. **Infrastructure abstraction**
   Developers build against a uniform API instead of underlying infrastructure. Compute, storage, and network become programmable primitives rather than vendor-specific constructs.
4. **Efficient multi-tenancy and resource governance**
   Namespaces, quotas, and RBAC allow safe team isolation, controlled resource allocation, and strong compliance boundaries.
5. **Portable, cloud-agnostic operations**
   Kubernetes abstracts cloud-specific differences and aligns organizations behind a single operational model, enabling hybrid and multi-cloud strategies.

Kubernetes Architecture: Control Plane vs Worker Nodes



Master/Head/Control Plane



etcd
- Key-value datastore store of desired state
- 3-5 replicas
- Contains RBAC rules, application environment information, etc.

kube-apiserver
- Validates and configures data
- Services REST operations and provides the frontend to the cluster's shared state

kube-scheduler
- Watches the API server and makes assignments to appropriate healthy notes

kube-controller-manager
- Background control loops
- Monitors cluster and responds to events
- Replication controller, endpoints controller, namespace controller, and serviceaccounts controller

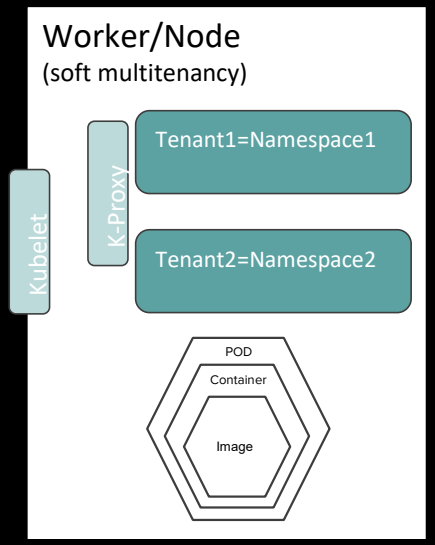
cloud-controller-manager
- Manages integrations with instances, load-balancers and storage

DNS
- Every cluster has an address space based on a DNS domain
- Service register with the DNS including StatefulSets and individual Pods that a StatefulSet manages

Kubelet
- Communication between Master and the Node
- Manages the Pods and the containers

Namespaces: Enabling Soft Multitenancy in Kubernetes

Why namespaces matter (e.g., RBAC isolation, resource quotas).



Node
- Worker machine
- Can be virtual or physical machine
- Data plane where application service run
- Can container multiple Pods

k-proxy
- Guarantees each node gets its own unique IP address
- Implements local IPTABLES or IPVS to handle routing and load-balancing

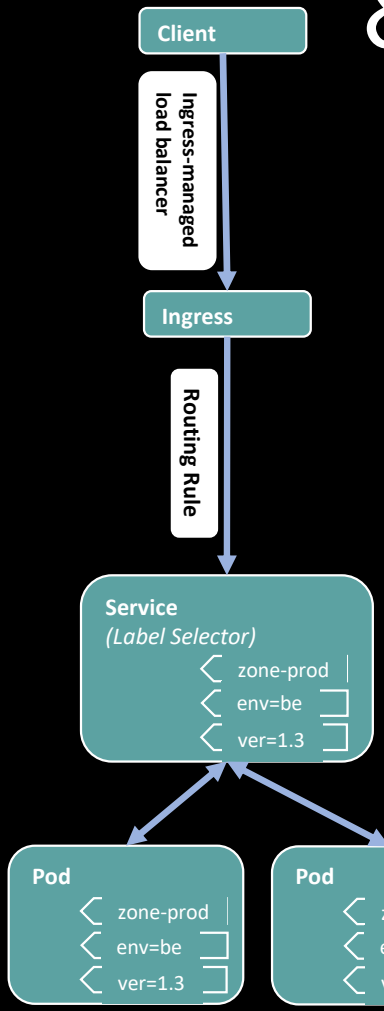
Namespace
- Partitions the address space below the cluster domain
- Can place Services and other objects in

Pod
- Host OS, network stack, IPC namespace, kernel namespace, memory, volumes, IP addresses, port range, routing table, etc.

Container
- Decouple applications from underlying host infrastructure
- Immutable
- Managed by cgroups, a Linux technology, that prevent starving another container in the Pod of CPU and memory

Image
- Ready-to-run software package
- Contains the application and its software dependencies

Kubernetes Services, Deployments & Ingress: Managing Access & Load



Ingress

- Instead of using and External Service, you can use Ingress as a more production-worthy approach
- Forward request to the internal service
- Load balance traffic
- Terminate SSL/TLS
- Offers name-based virtual hosting

Service

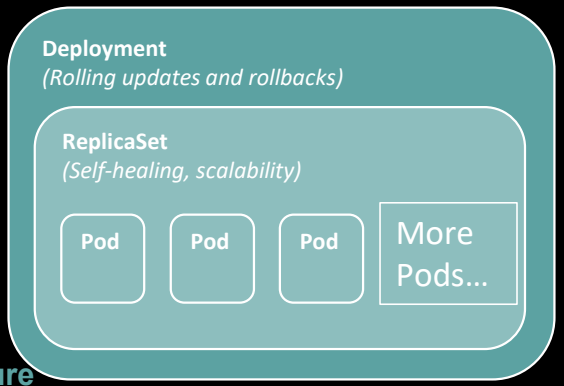
- Have a front-end that consists of a stable DNS name, IP address, and port
- Back-end can load-balance across a dynamic set of Pods
- Service automatically updates itself to reflect Pod as they come and go
- Provides stable networking endpoint

Deployment

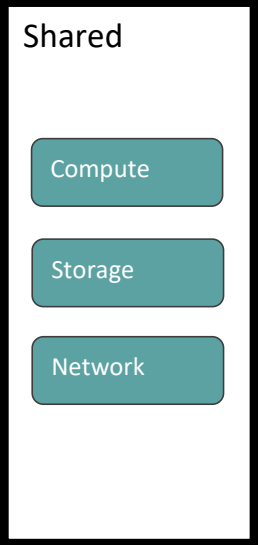
- Manges ReplicaSets

ReplicaSets

- Manages Pods



Shared



Compute, Storage, and Networking Extensions
- Used to enhance the nodes in your cluster

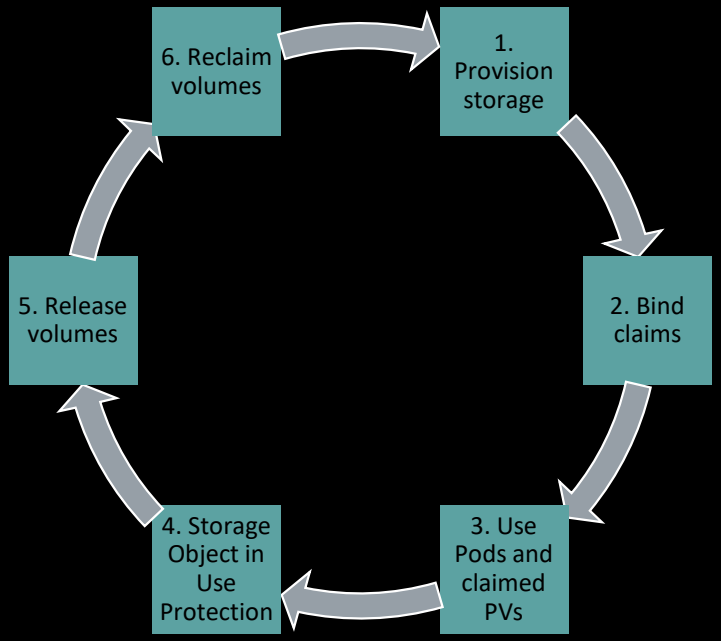
CSI and FlexVolume storage plugins
- Extend support for new kinds of volumes
- Durable storage
- Ephemeral storage
- Read-only filesystem paradigm

Device plugin
- Allows node to discover node facilities
- Built-in Node resources such as cpu and memory
- Provides these node-local facilities to Pods that request them

Network plugin
- Allows Kubernetes to work with different networking topologies and technologies
- Provides working Pod network and support other aspects of the Kubernetes network model

Persistent Storage in Kubernetes: PVs, PVCs, and Storage Classes

Pods request storage via PVCs, which bind to PVs.



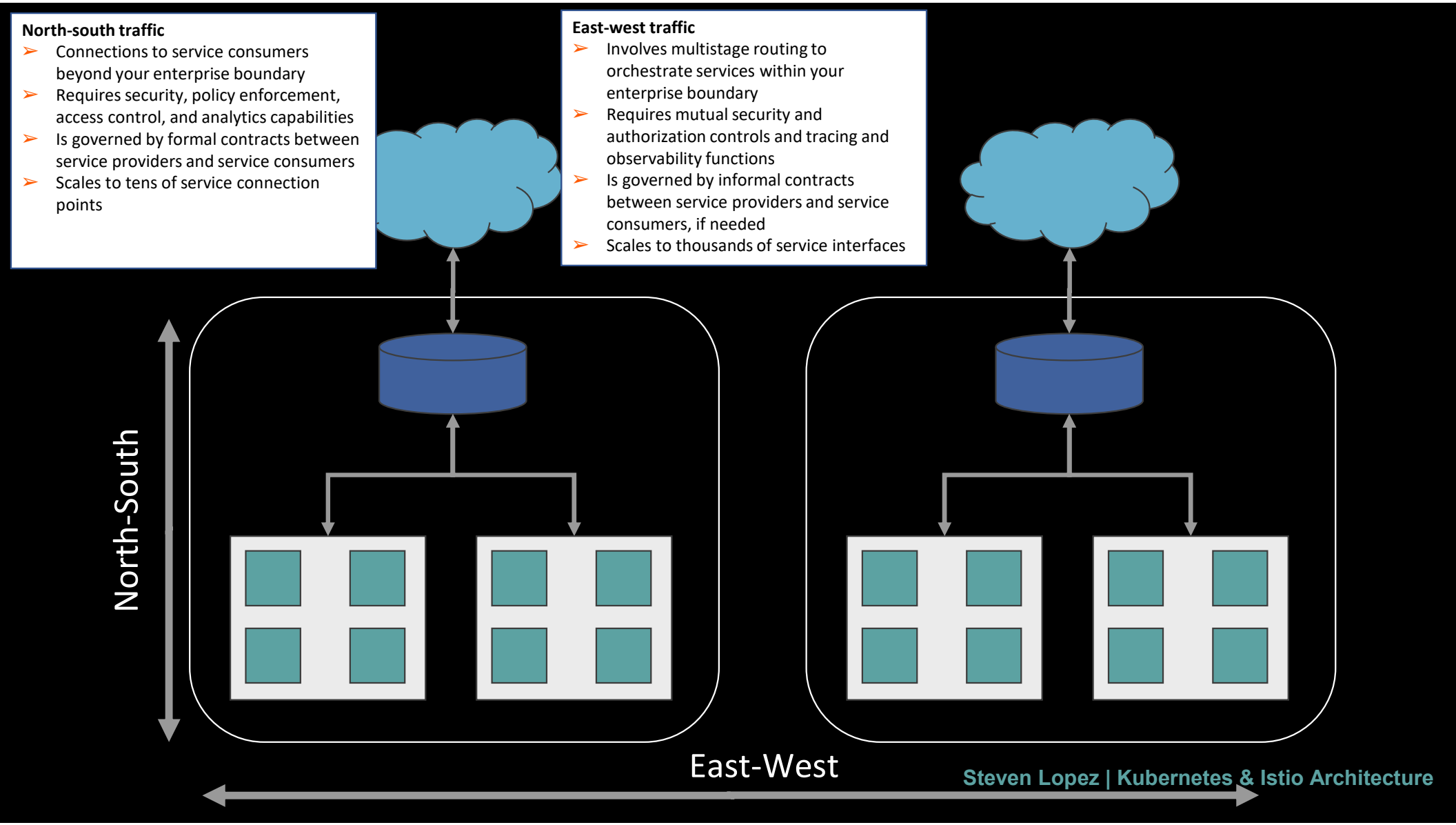
Lifecycle of a volume and claim

PV

- Defined by PersistentVolume API object
- Provisioned storage in the cluster
- Can be dynamically provisioned using StorageClass object
- Are volume plug-ins like Volumes but have a lifecycle that is independent of any individual Pod that uses the PV
- Not scoped to a single project
- Can be shared across the cluster from any project
- NFS, iSCSI, or cloud-provider-specific storage system
- HA relies on the underlying storage provider

PVC

- Developers can use PVC to request PV resources without knowledge of the underlying storage infrastructure
- Specific to a project
- Can request storage capacity and access modes


North-south traffic
Connections to service consumers beyond your enterprise boundary
Requires security, policy enforcement, access control, and analytics capabilities
Is governed by formal contracts between service providers and service consumers
Scales to tens of service connection points
East-west traffic
Involves multistage routing to orchestrate services within your enterprise boundary
Requires mutual security and authorization controls and tracing and observability functions
Is governed by informal contracts between service providers and service consumers, if needed
Scales to thousands of service interfaces
North-South
East-West
Steven Lopez | Kubernetes & Istio Architecture

Helm Charts vs Operators:

Automating Kubernetes Workloads

	Helm Chart	Operator
Packaging	✓	✓
App Installation	✓	✓
App Update (Kubernetes manifests)	✓	✓
App Upgrade (data migration, adaption, etc)	-	✓
Backup & Recovery	-	✓
Workload & Log Analysis	-	✓
Intelligent Scaling	-	✓
Auto tuning	-	✓

Helm is great for templated installs;
Operators add lifecycle automation.

CRD – Custom Resources Definition

CRD – allows us to EXTEND the Kubernetes API

- Modify the API without recompiling
- Create our very own API resource/object
- Resource/object exists but nothing acts on its presence, and this is where controllers come in

Customer Controllers

- Especially effective when combined with customer resources
- Operator pattern combines customer resources with custom controllers
- Encode domain knowledge

Operator pattern

- Software extensions that make use of customer resources to manage applications and their components
- Repeatability of installation and upgrade
- Constant health checks of every system component

Examples

- Deploy applications on demand
- Restoring backups and application state
- Performing upgrades of the application code alongside of related changes such as database schemas or extra configuration settings
- Publish a service to applications that don't support Kubernetes API to discover them
- Etc.

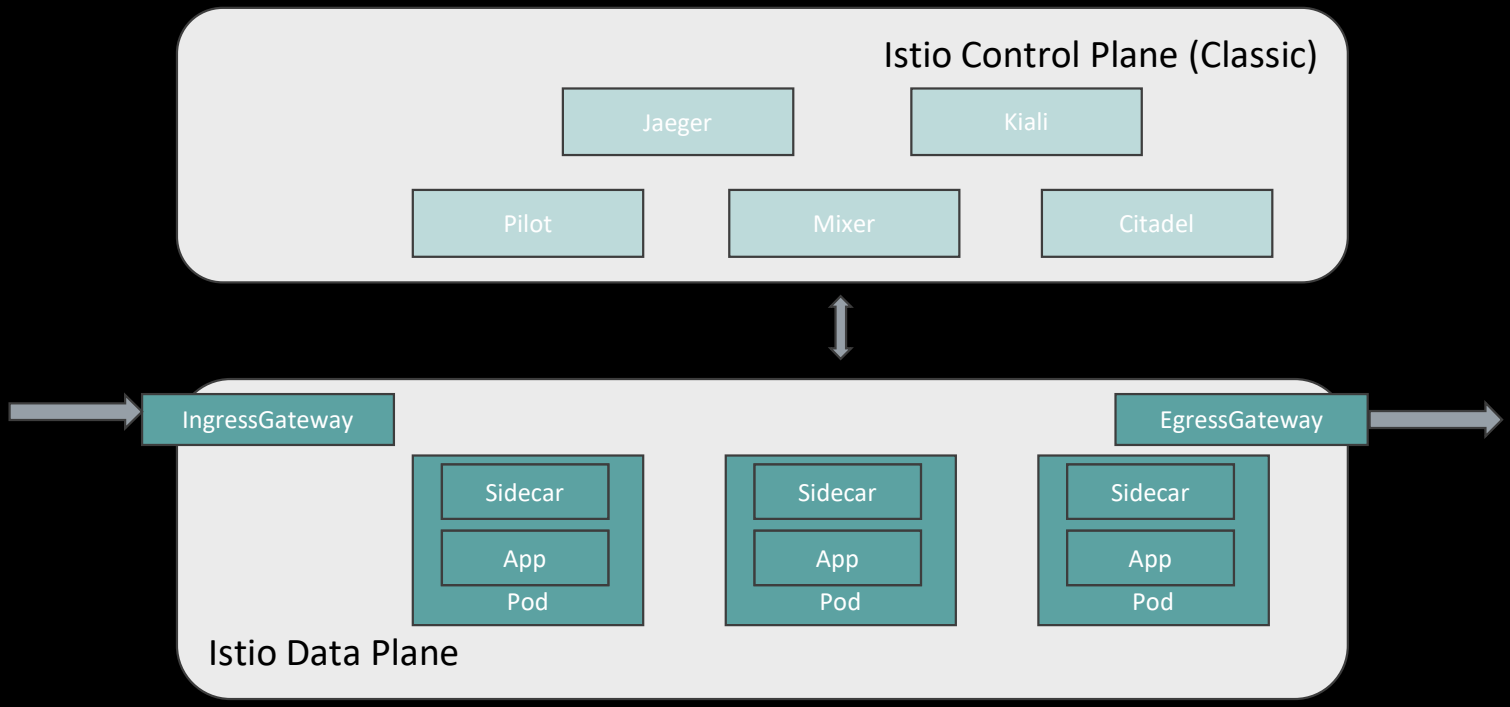
Istio

What Istio Solves

1. **Fine-grained traffic management**
   Istio introduces intelligent routing (canary releases, A/B tests, retry budgets, timeouts, circuit breaking) that are impossible or brittle when implemented inside application code.
2. **Zero-trust service-to-service security**
   Istio enforces mutual TLS (mTLS), identity-based authentication, authorization policies, and encrypted in-cluster traffic without requiring code changes.
3. **Unified observability across microservices**
   Envoy sidecars generate consistent metrics, logs, and distributed traces across all workloads, providing deep insight into east-west traffic.
4. **Policy enforcement decoupled from application logic**
   Rate limits, quotas, access rules, and mesh-level behaviors are configured centrally. This keeps services simple and reduces security drift.
5. **Service mesh extensibility**
   Istio provides a programmable data plane (Envoy) and a declarative control plane, enabling enterprises to standardize on a uniform mesh behavior across languages, teams, and clusters.

Istio High-Level Overview

** Pilot/Mixer/Citadel components*
Replaced with Consolidated Control Plane



Jaeger
- Tracing system

Kiali
- Observability console

Pilot
- Route configuration

Mixer
- Telemetry + policy

Citadel
- Service credential management

IngressGateway
- Route incoming connections

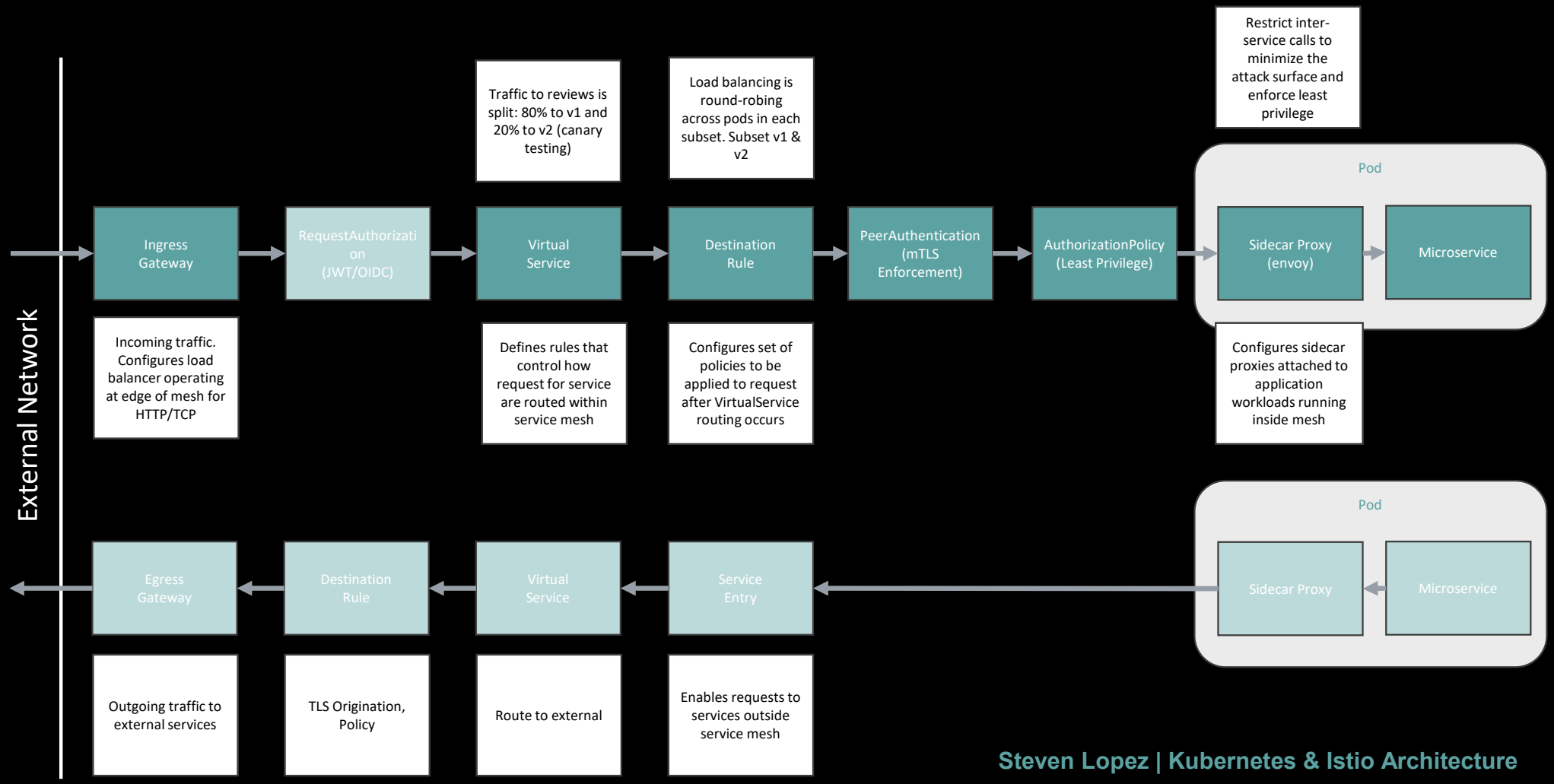
EgressGateway
- Control outgoing connections

Sidecar
- mediates inbound and outbound communication

App
- Ready-to-run software package
- Contains the application and its software dependencies

Pod
- Host OS, network stack, IPC namespace, kernel namespace, memory, volumes, IP addresses, port range, routing table, etc.

Core Istio Resources: How Traffic is Controlled



Steven Lopez | Kubernetes & Istio Architecture

Istio Security: Locking Down Ingress, Egress, and Service Access

- RBAC
- mTLS
- namespace isolation

Deny All

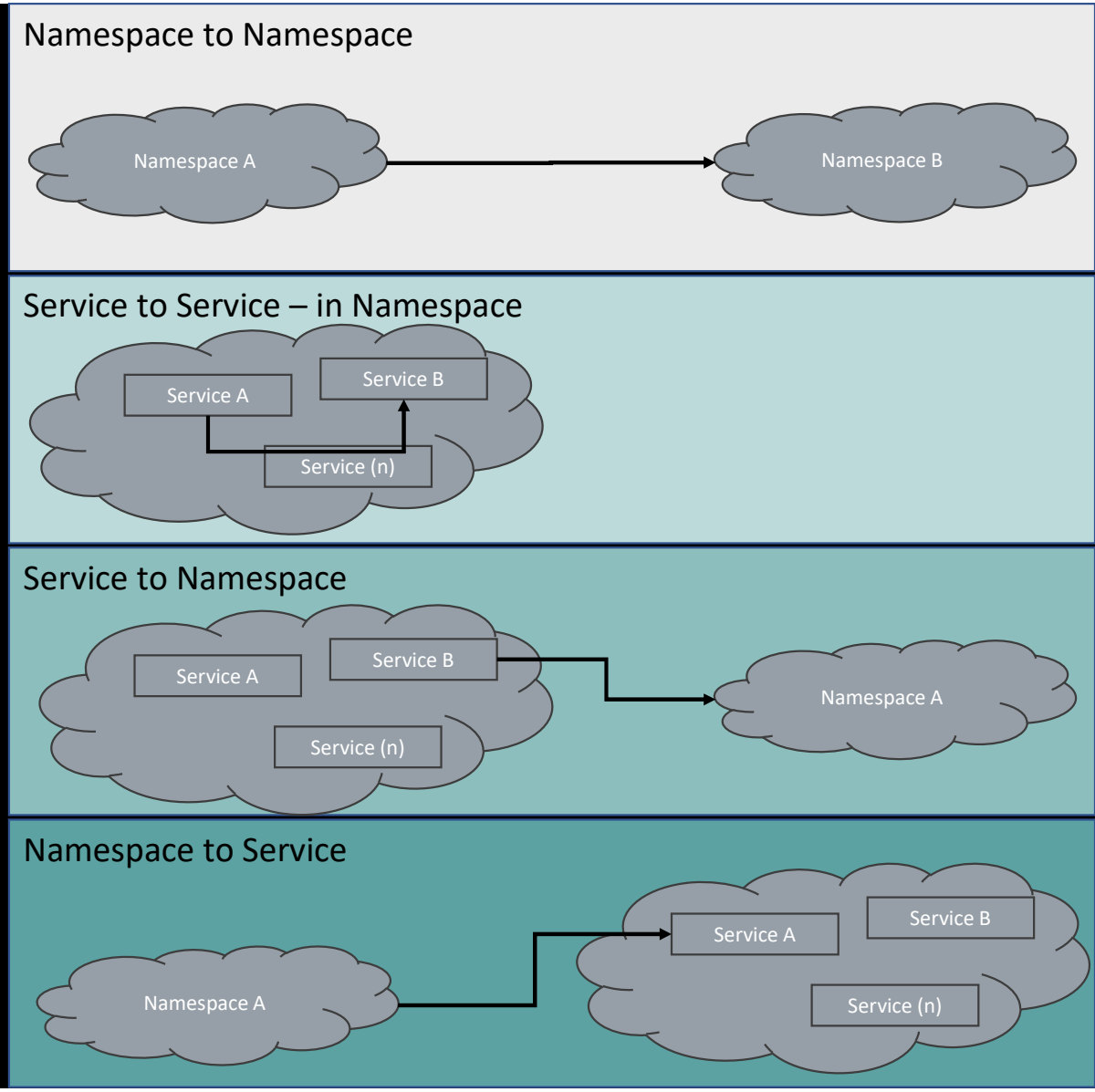
Deny All Ingress

Deny All Egress

Service to Service

Namespace to Namespace

Service to Namespace



How Enterprises Combine Them to Achieve Resiliency, Security, and Scale

1. **Kubernetes becomes the compute substrate; Istio becomes the traffic substrate**
   Kubernetes handles orchestration, scheduling, and lifecycle. Istio overlays secure, intelligent communication across every pod.
2. **Microservices gain reliability through mesh-enabled traffic shaping**
   Kubernetes keeps services alive and scaled. Istio ensures calls between services degrade gracefully with retries, circuit breaking, and canary routing.
3. **Zero-trust architecture is enforced end-to-end**
   Kubernetes isolates workloads via namespaces and RBAC. Istio enforces mTLS, identity-aware access, and namespace/service-level policies to prevent lateral movement.
4. **Standardized observability pipelines across the enterprise**
   Kubernetes logs events, deployments, and node health. Istio enriches telemetry with traces, latency metrics, and real request-level visibility across all services.
5. **Platform teams gain centralized governance**
   Kubernetes gives declarative control over compute resources; Istio gives declarative control over traffic, trust boundaries, and communication behaviors. Together they create a unified “platform API” for the enterprise.
6. **Multi-cluster and hybrid-cloud architectures become manageable**
   Kubernetes handles cluster provisioning and scaling across cloud and on-prem. Istio provides a consistent mesh identity and traffic layer across clusters, enabling true hybrid resiliency.

📘 Kubernetes:
- https://kubernetes.io/docs/home/
- https://kubernetes.io/docs/concepts/extend-kubernetes/operator/

🔧 Helm & Operators:
- https://helm.sh/
- https://artifacthub.io/
- https://operatorhub.io/

🔐 Istio & Service Mesh:
- https://istio.io/
- https://maistra.io/
- https://cloud.redhat.com/blog/design-considerations-at-the-edge-of-the-servicemesh


Questions?