What is Envoy?

Created at Lyft and now used inside companies including Google, Apple, Netflix and many more, Envoy is a high-performance open source edge, middle, and service mesh proxy that makes the network transparent to applications. Written in C++ for performance reasons, the Envoy out of process architecture can be used with any application, in any language or runtime; including HTTP/2 gRPC proxying, MongoDB filtering and rate limiting. Envoy is designed to minimize memory and CPU footprint, while providing capabilities such as load balancing and deep observability of network, service, and database activity.

Main features:
- High-performance native code implementation
- Eventually consistent service discovery
- API-driven configuration
- L4 (TCP) proxy with an extensible filter chain mechanism
- Parallel pluggable L7 filter chain
- Transparent HTTP/1 to HTTP/2 proxy in both directions
- Robust and consistent observability within a microservice architecture
- Easy debugging
- Advanced load balancing, including zone awareness, retry, timeouts, circuit breaking, rate limiting, shadowing, and outlier detection
- Best-in-class observability using both statistics, logging, and distributed tracing
- Lyft’s Envoy: From Monolith to Service Mesh

What is a Service Mesh?

Figure 1 illustrates the service mesh concept at its most basic level. There are four service clusters (A-D). Each service instance is co-located with a sidecar network proxy. All network traffic (HTTP, REST, gRPC, Redis, etc.) from an individual service instance flows via its local sidecar proxy to the appropriate destination. Thus, the service instance is not aware of the network at large and only knows about its local proxy. In effect, the distributed system network has been abstracted away from the service programmer.
Why is Envoy needed in the market?

Value Proposition: Envoy helps ease the transition to, and operation of, cloud-native architectures by managing the interactions among microservices in order to ensure application performance.

CIOs: Envoy simplifies management of cloud-native environments by providing a scalable service mesh that handles communications among microservices and components.

The industry is rapidly moving towards a polyglot (many language) microservice architecture. Networking and observability are notoriously some of the hardest things to get right in these architectures. Advanced features such as timeouts, rate limiting, circuit breaking, load balancing, retries, stats, logging, and distributed tracing are required to handle network failures in a fault tolerant and reliable way.

Organizations can solve microservice networking in one of two ways:

1. Develop a sophisticated library for each language in use.
2. Develop a sidecar proxy that can abstract as many of the needed concerns as possible. Thus, complex functionality is implemented once and reused for every application language to the greatest extent possible.

Envoy fills the industry need for an extremely high performance, well designed, robust, and extensible proxy server.

Which companies are using it?

Apple, Booking.com, eBay, F5, Google, IBM, Lyft, Medium, Microsoft, Netflix, Pinterest, Salesforce, Stripe, Tencent, Twilio, Verizon, VSCO, and many more.

What problem does it solve?

Almost every company with a moderately-sized service oriented architecture is having the same problems:

- An architecture composed of a variety of languages, each containing a half-baked RPC library, including partial (or zero) implementations of rate limiting, circuit breaking, timeouts, retries, etc.
- Differing or partial implementations of stats, logging, and tracing across both owned services, as well as infrastructure components such as ELBs.
- A desire to move to SoA for the decompositional scaling benefits, but an on-the-ground reality of chaos as application developers struggle to make sense of an inherently unreliable network substrate.

In summary, an operational and reliability headache. Envoy solves the above problems for the enterprise allowing them to scale.

What is a control plane?

A service mesh control plane provides policy and configuration for all of the running data planes in the mesh. It does not touch any packets/requests in the system. The control plane turns all of the data planes into a distributed system. It is composed of the following pieces:

- **The human**: There is still a (hopefully less grumpy) human in the loop making high-level decisions about the overall system.
- **Control plane UI**: The human interacts with some type of UI to control the system. This might be a web portal, a CLI, or some other interface. Through the UI, the operator has access to global system configuration settings such as deploy control (blue/green and/or traffic shifting), authentication and authorization settings, route table specification (e.g., when service A requests /foo what happens), and load balancer settings (e.g., timeouts, retries, circuit breakers, etc.).
• **Workload scheduler:** Services are run on an infrastructure via some type of scheduling system (e.g., Kubernetes or Nomad). The scheduler is responsible for bootstrapping a service along with its sidecar proxy.

• **Service discovery:** As the scheduler starts and stops service instances it reports liveness state into a service discovery system.

• **Sidecar proxy configuration APIs:** The sidecar proxies dynamically fetch state from various system components in an eventually consistent way without operator involvement. The entire system composed of all currently running service instances and sidecar proxies eventually converge. Envoy's universal data plane API is one such example of how this works in practice.

**What is a data plane?**

Service mesh data plane, sometimes called the sidecar proxy, touches every packet/request in the system and is responsible for service discovery, health checking, routing, load balancing, authentication/authorization, and observability.

• **Service discovery:** What are all of the upstream/backend service instances that are available?

• **Health checking:** Are the upstream service instances returned by service discovery healthy and ready to accept network traffic? This may include both active (e.g., out-of-band pings to a /healthcheck endpoint) and passive (e.g., using 3 consecutive 5xx as an indication of an unhealthy state) health checking.

• **Routing:** Given a REST request for/from the local service instance, to which upstream service cluster should the request be sent?

• **Load balancing:** Once an upstream service cluster has been selected during routing, to which upstream service instance should the request be sent? With what timeout? With what circuit breaking settings? If the request fails should it be retried? Authentication and authorization: For incoming requests, can the caller be cryptographically attested using mTLS or some other mechanism? If attested, is the caller allowed to invoke the requested endpoint or should an unauthenticated response be returned?

• **Observability:** For each request, detailed statistics, logging, and distributed tracing data should be generated, so that operators can understand distributed traffic flow and debug problems as they occur.

**What projects/technologies compete with Envoy?**

Data planes: Linkerd, NGINX, HAProxy, Traefik, F5, Cloud load balancers

1. NGINX Plus
2. HAProxy
3. Hardware and software load balancers from F5, Juniper, etc.
4. Envoy tends to be complimentary with cloud load balancers such as AWS NLB, but does compete against AWS ALB.

**What projects/technologies are complementary to Envoy?**

Control planes: Istio, Nelson, SmartStack

**Are Istio and Envoy competitors?**

No. Istio was announced in May 2017. The project goals of Istio look very much like the advanced control plane. The default proxy of Istio is Envoy. Thus, Istio is the control plane and Envoy is the data plane. In a short time, Istio has garnered a lot of excitement, and other data planes have begun integrations as a replacement for Envoy (both Linkerd and NGINX have demonstrated Istio integration). The fact that it's possible for a single control plane to use different data planes means that the control plane and data plane are not necessarily tightly coupled. An API such as Envoy's universal data plane API can form a bridge between the two pieces of the system.
Where can I find more information about Envoy?


For educational, technical and case study presentations about Envoy, check out:

- CloudNativeCon North America 2017 Session: Bottoms-Up Adoption of a Microservices Workflow Using Kubernetes & Envoy – Rafael Schloming & Phil Lombardi, Datawire
- Datawire: Microservices Architecture Guide: Getting started with Lyft Envoy for microservices resilience
- Envoy Community Meeting: 11-7-2017
- Envoy Community Meeting: 10-24-2017
- Lyft Engineering blog: Envoy 7 Months Later

What does it mean to be a CNCF Project?

As a CNCF hosted project, Envoy is part of a neutral community aligned with technical interests to help companies move to cloud native deployment models and help developers deliver on the promise of microservices and cloud native applications at scale. As Envoy grows, CNCF is helping build its community, marketing and documentation efforts. For more, read “Envoy joins CNCF.”

What level Project is Envoy?

Envoy is an incubation level project, under the CNCF Graduation Criteria v1.0. The CNCF Graduation Criteria by the TOC provides every CNCF project an associated maturity level of either inception, incubating or graduated, which allows CNCF to review projects at different maturity levels to advance the development of cloud native technology and services. As an incubating project, Envoy must have documentation that it is being used successfully in production by at least three independent end users, have a healthy number of committers, and demonstrate a substantial ongoing flow of commits and merged contributions.

Why does CNCF have competing projects?

The cloud native ecosystem is still nascent and quickly evolving. CNCF recognizes that many promising technologies are emerging, some better suited for certain workloads, use cases or environments. CNCF’s goal is help many rising technologies mature. In this respect, we expect to have overlapping projects in some cases.