solo.io

Multi Cluster Service Mesh Patterns, Operations, and Extensibility with WebAssembly

Idit Levine
Founder and CEO
@idit_levine

Christian Posta
Global Field CTO
@christianposta

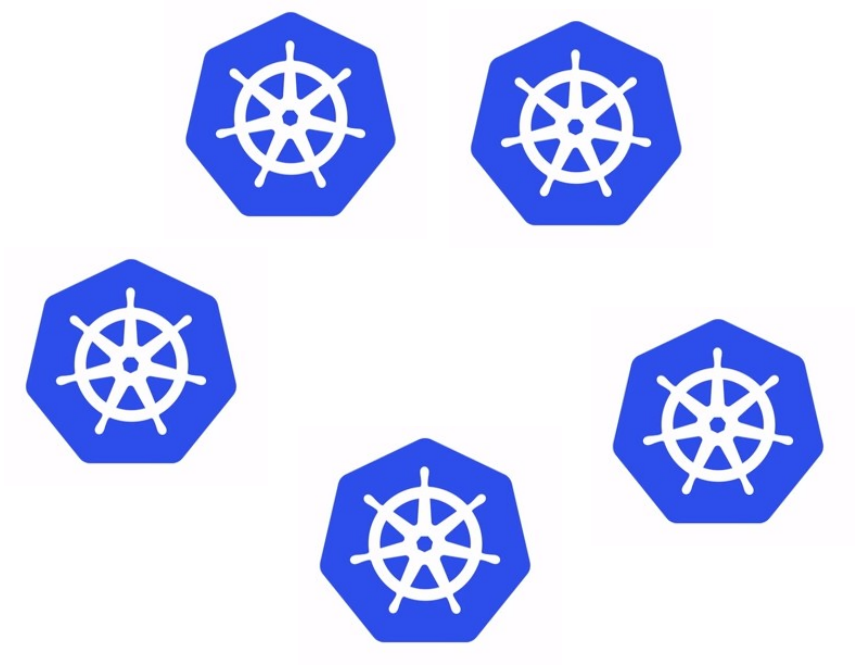
Challenges in Adopting Microservices

- Improve velocity of teams building and delivering code
- Decentralized implementations vs centralized operations
- Connect and include existing systems and investments
- Improve security posture
- Stay within regulations and compliance

solo.io

A Solution is many, smaller clusters

- High availability
- Compliance
- Isolation / Autonomy
- Scale
- Data locality, cost
- Public / DMZ / Private networks



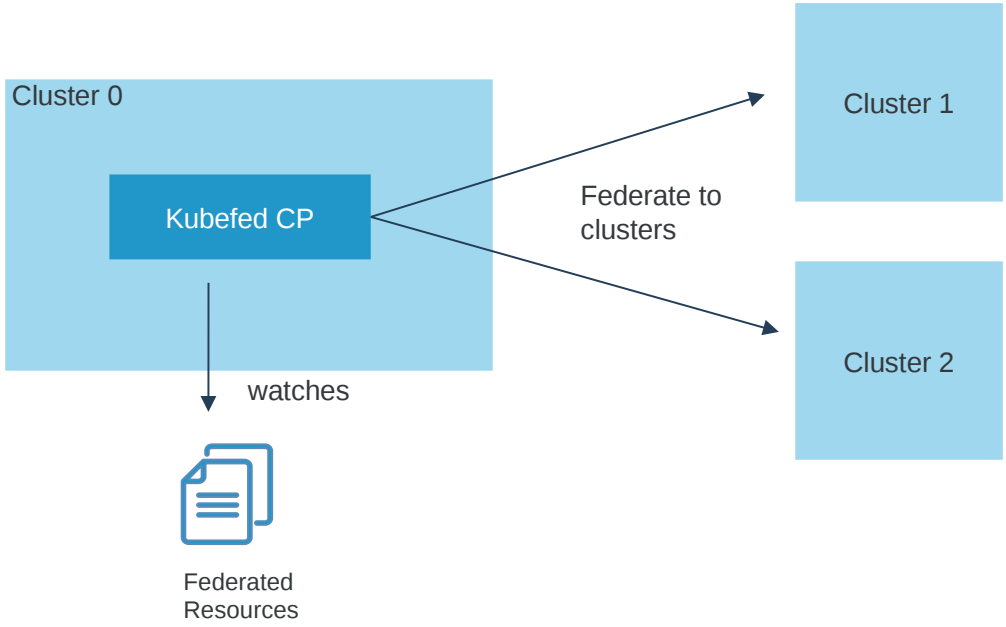
solo.io

Considerations for multiple clusters

- Exact replicas of each other, same fleet?
- Separate, non-uniform deployments?
- Single operational/administrative control
- Segmented by network? Segmented by team?
- Independent administration?
- Cluster federation? Independent clusters, which services are shared?

solo.io

Example: Kubefed




```
apiVersion:
types.kubefed.io/v1beta1
kind: FederatedService
metadata:
  name: echo-server
spec:
  placement:
    clusterSelector:
      matchLabels: {}
  template:
    metadata:
      labels:
        app: echo-server
    spec:
      ports:
      - name: http
        port: 8080
      selector:
        app: echo-server
```

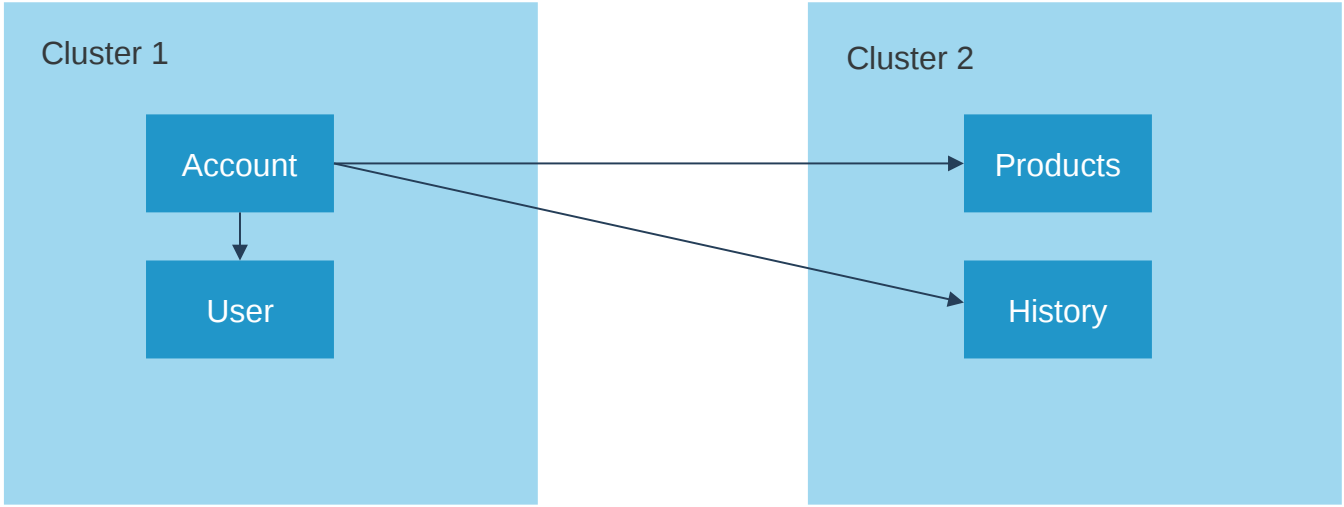
  https://github.com/kubernetes-sigs/kubefed

Services Need to Communicate with Each Other

Patterns within and outside of Kubernetes

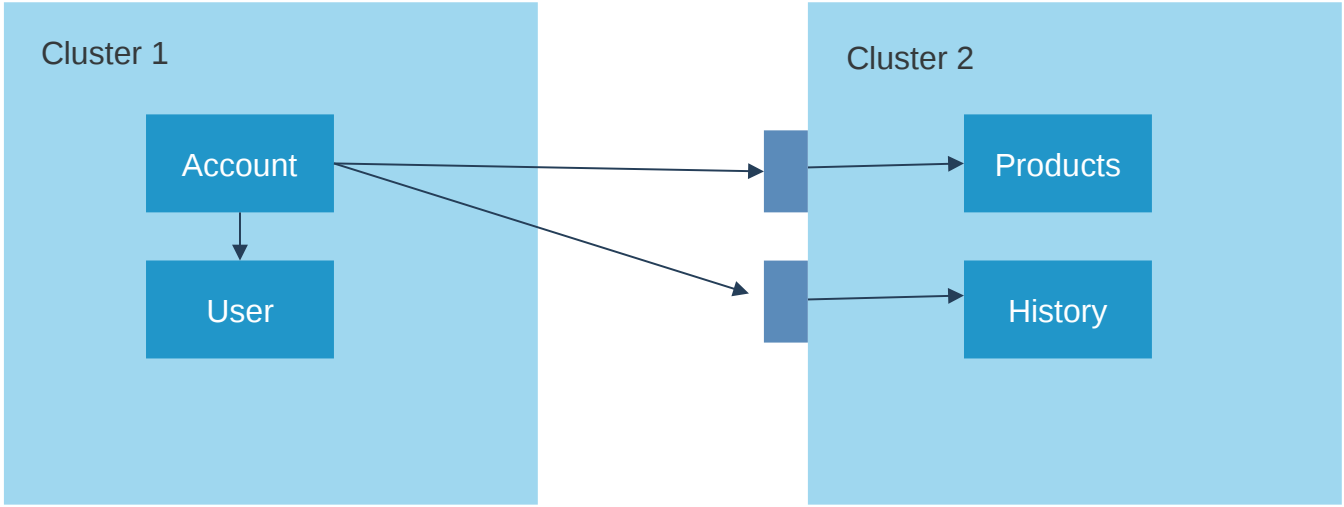
solo.io

Pattern: Flat network across pods



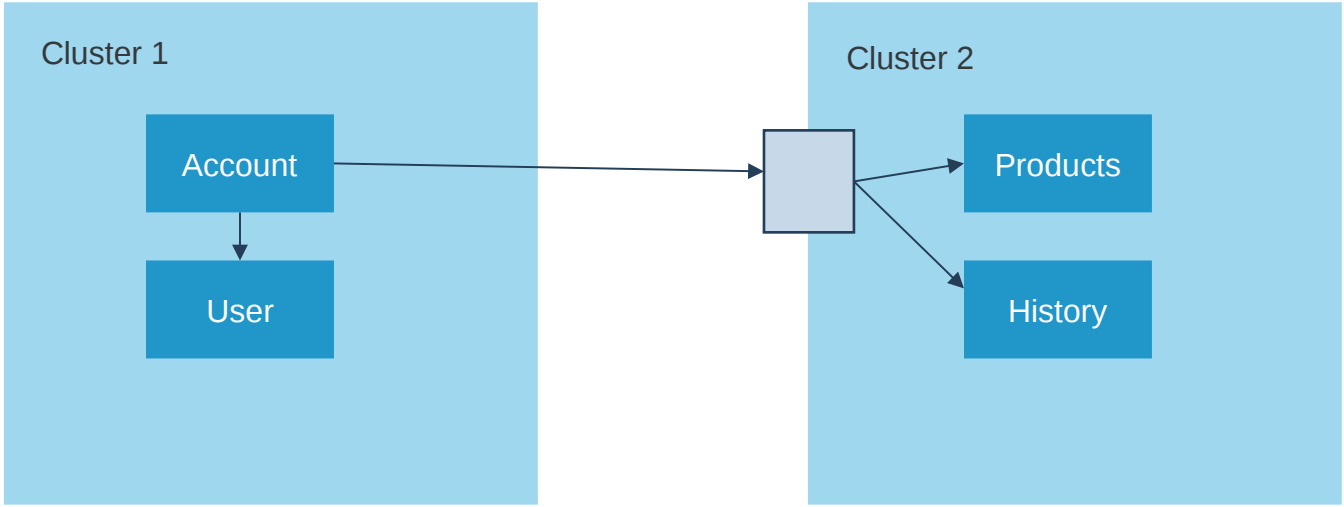
solo.io

Pattern: Different network, expose all services



solo.io

Pattern: Different network, controlled gateway



solo.io

Forces to balance

- Security (AuthN/Z, encryption, identity)
- Service discovery
- Failover / traffic shifting / transparent routing
- Observability
- Separate networks
- Well-defined fault domains
- Building for scale

solo.io

Can you build these patterns *just* using Kubernetes?

solo.io

Service Mesh can help

Envoy Proxy is the magic behind service mesh

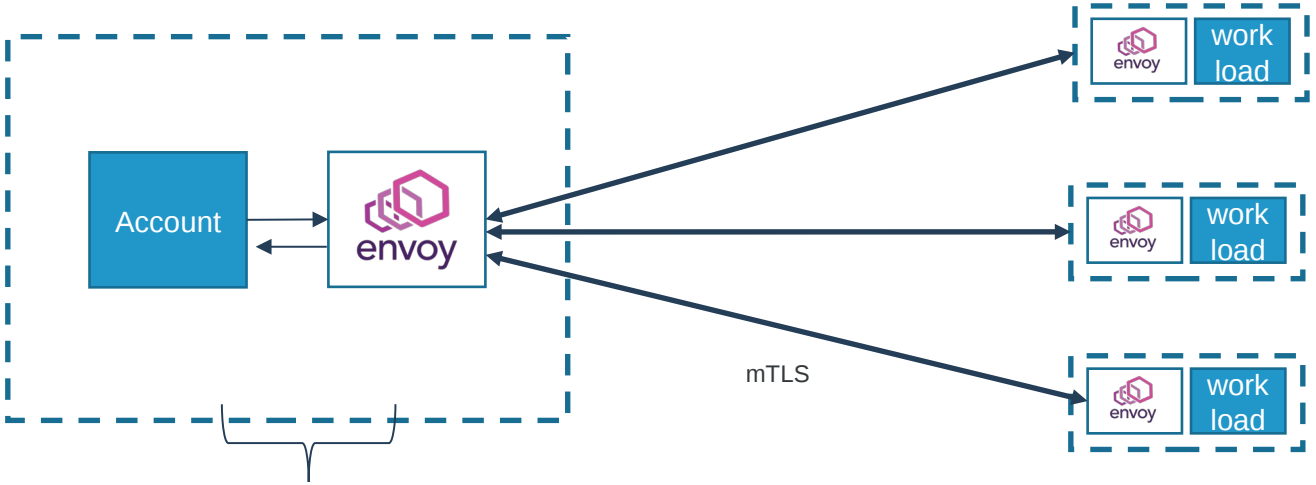
Envoy Implements the following:

- Zone aware, priority/locality load balancing
- Circuit breaking, outlier detection
- Timeouts, retries, retry budgets
- Traffic shadowing
- Request racing
- Rate limiting
- RBAC, TLS origination/termination
- Access logging, statistics collection

http://envoyproxy.io

solo.io

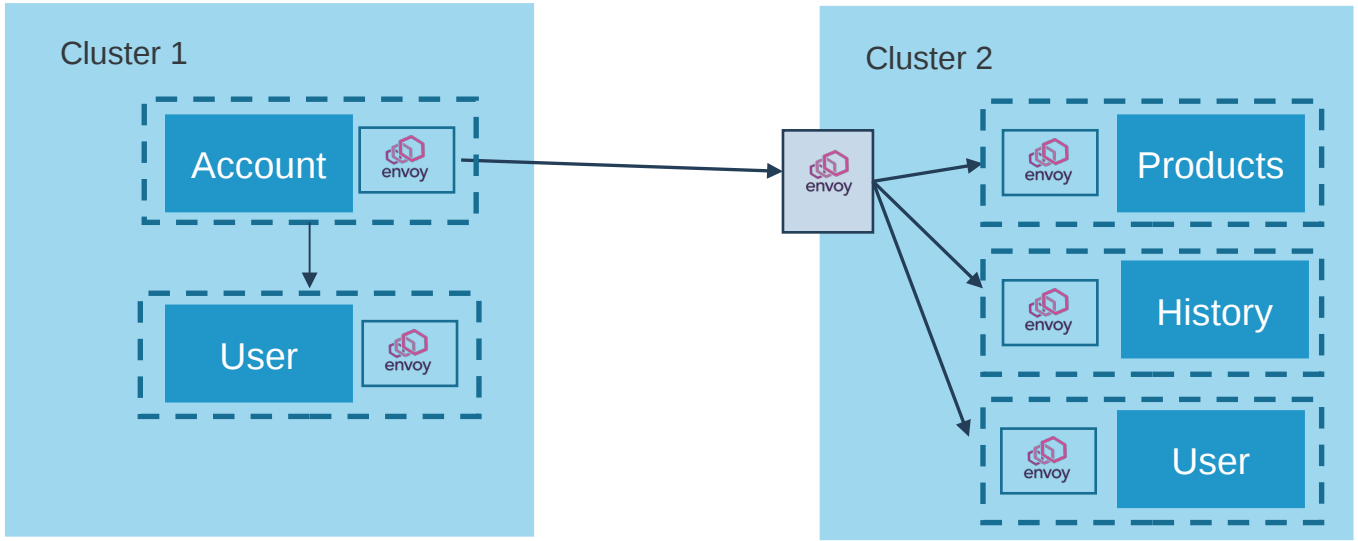
Envoy Proxy to do application networking heavy lifting



- Transparent client-side routing decisions
- TLS orig/termination
- Circuit breaking
- Stats collection

solo.io

Envoy Proxy as backbone for multi-cluster communication federation



solo.io

Other key Envoy Proxy features

- Request hedging
- Retry Budgets
- Load balancing priorities
- Locality weighted load balancing
- Zone aware routing
- Degraded endpoints (fallback)
- Aggregated clusters

solo.io

Multi-Cluster Examples

Service Mesh examples using Envoy Proxy

solo.io

Three Examples and Considerations

1

Shared Service Mesh Control Plane with Flat Network

2

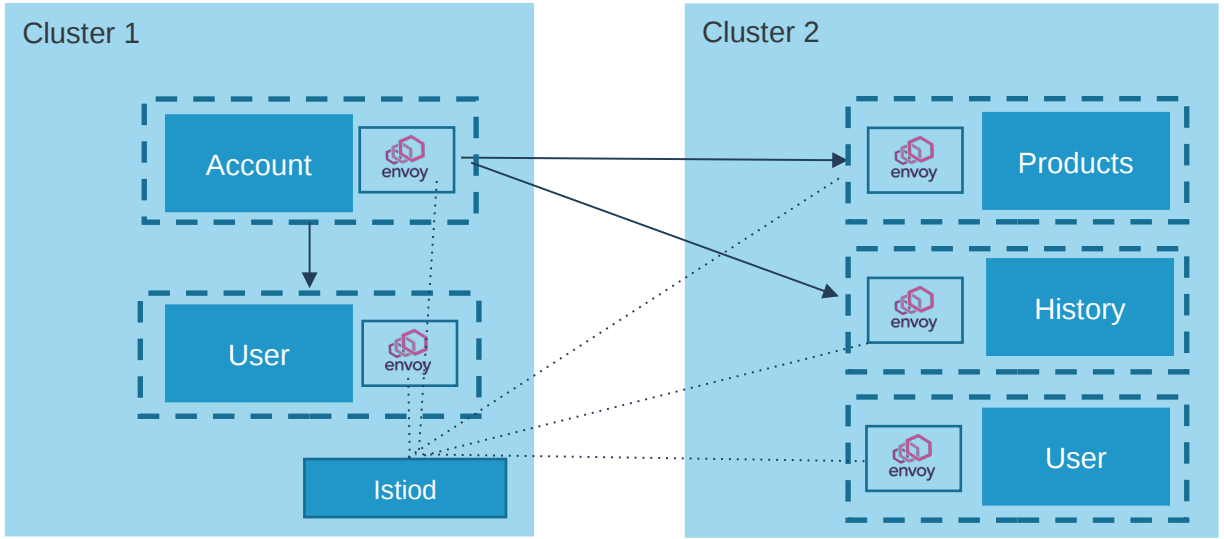
Shared Service Mesh Control Plane with Separate Networks

3

Separate Service Mesh Control Planes with Separate Networks

solo.io

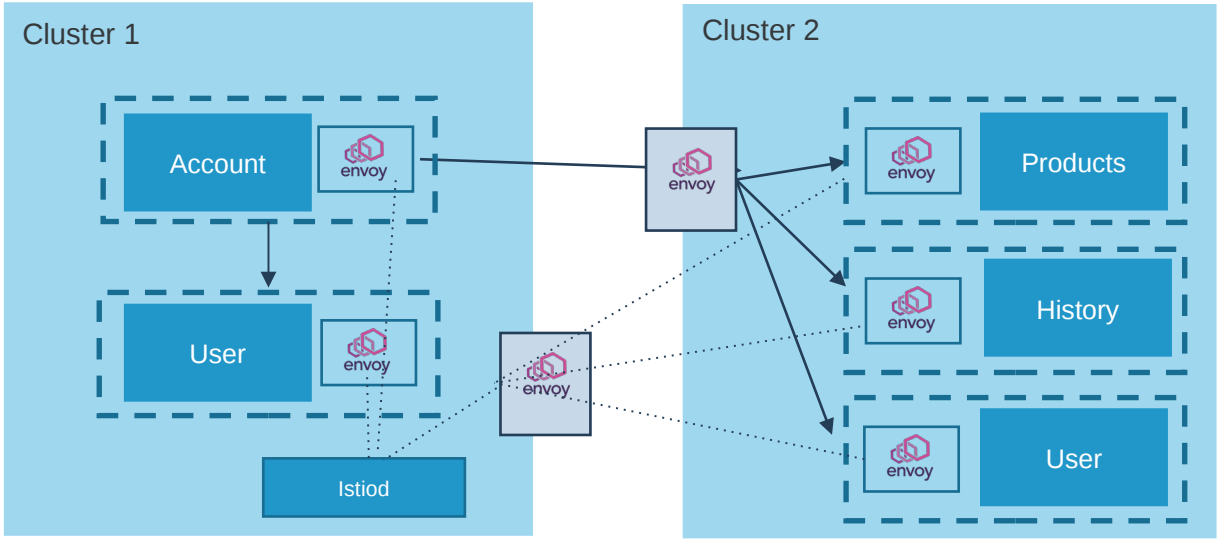
1 - Istio **shared** control plane, **flat** network



- Simplest setup for multi-cluster
- No special Envoy routing (though may use zone-aware)
- Shared control plane increases the failure domain to multiple clusters
- Use simpler, flat networking if possible
- No special considerations for identity (identity domain is shared)
- Still need to federate telemetry collection

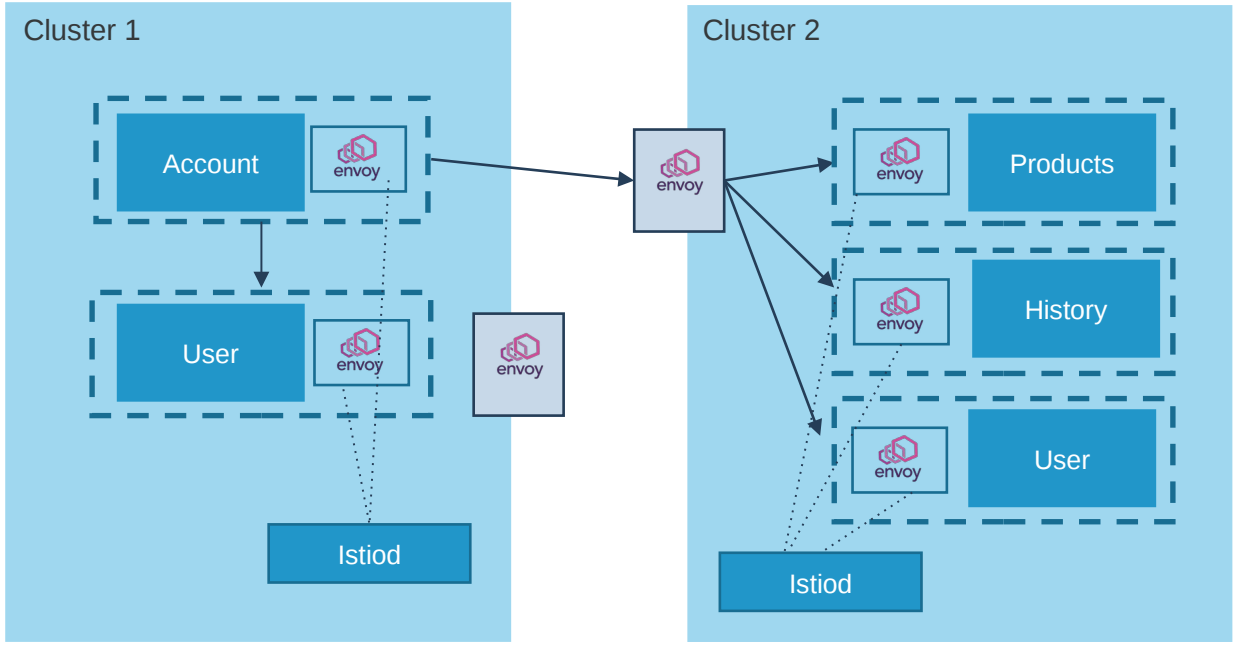
solo.io

2 - Istio **shared** control plane, **separate** networks



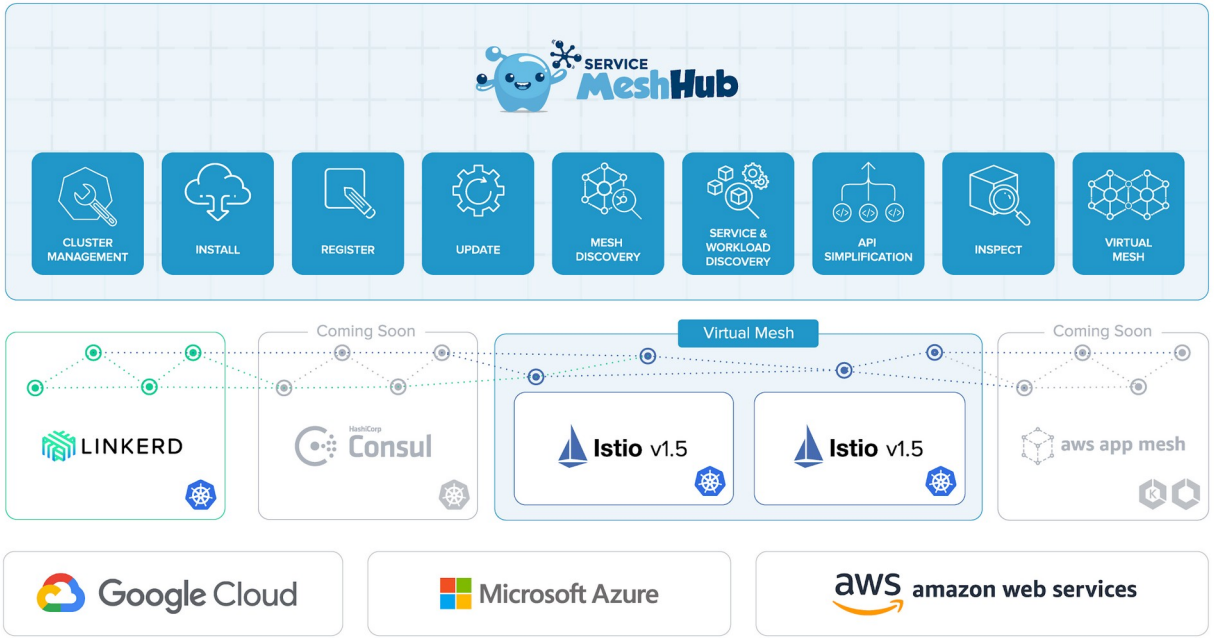
- Gateway allows communication between networks
- Envoy Locality Weighted LB (for the gateway endpoints). Istio calls this “split horizon EDS”.
- Shares same failure domain across all clusters
- Gateways facilitate communication AND control plane
- Slight increase in burden on operator to label networks and gateway endpoints correctly so Istio has that information

3 - Istio **separate** control plane, **separate** networks



- Gateway allows communication between networks
- Uses Istio ServiceEntry to enable cross-network discovery
- Independent control planes
- Separate, independent failure domains
- Doesn’t solve where trust domains MUST be separate (with federation at the boundaries)
- Increased operator burden to maintain service discovery, identity federation, and configuration across meshes

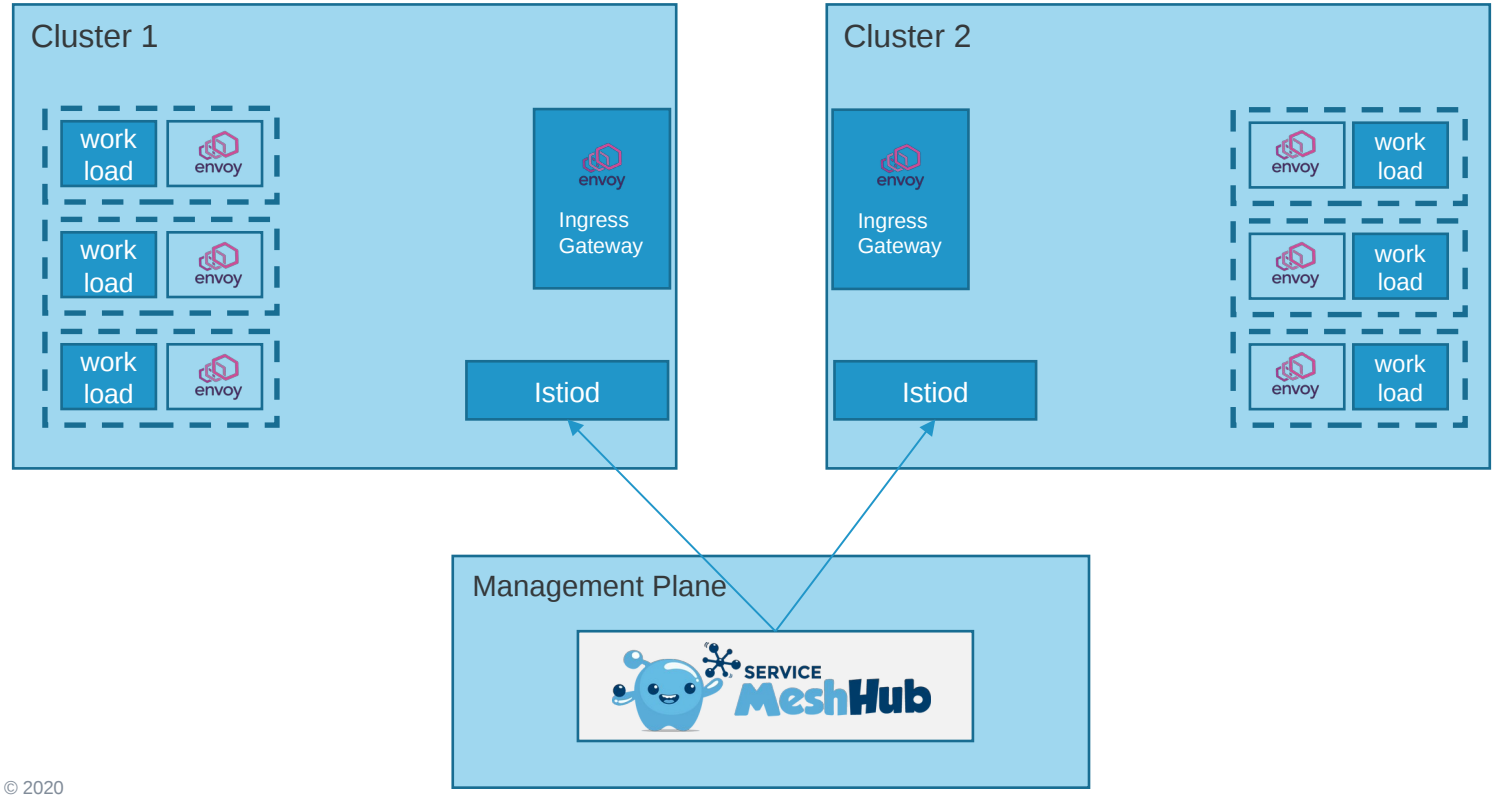
Open Source Service Mesh Hub



- Open Source
- Install & Configure
- Discovery
- Unified API
- Virtual Mesh
- Shared Identity
- Developer friendly tooling

solo.io

Open Source Service Mesh Hub



solo.io

Demo

solo.io

Extending the Service Mesh Data Plane with WebAssembly

solo.io

What is WebAssembly? It’s neither web nor assembly

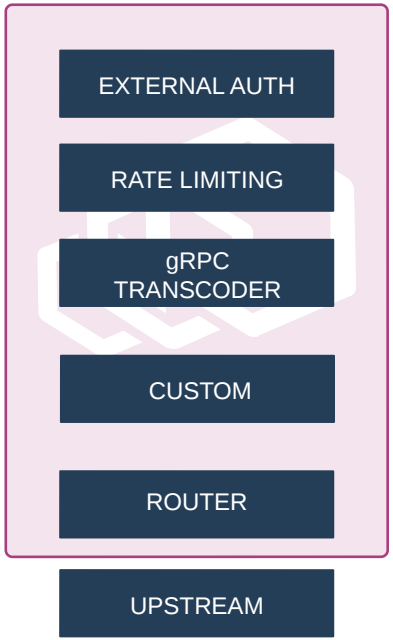
WebAssembly (abbreviated Wasm) is a binary instruction format for a stack-based virtual machine. Wasm is designed as a portable target for compilation of high-level languages like C/C++/Rust, enabling deployment on the web for client and server applications.

-> No Assembly -> Binary Instruction Format

-> No Web -> WASI

PORTABLE | **SECURE** | **FAST** | **ANY LANGUAGE**

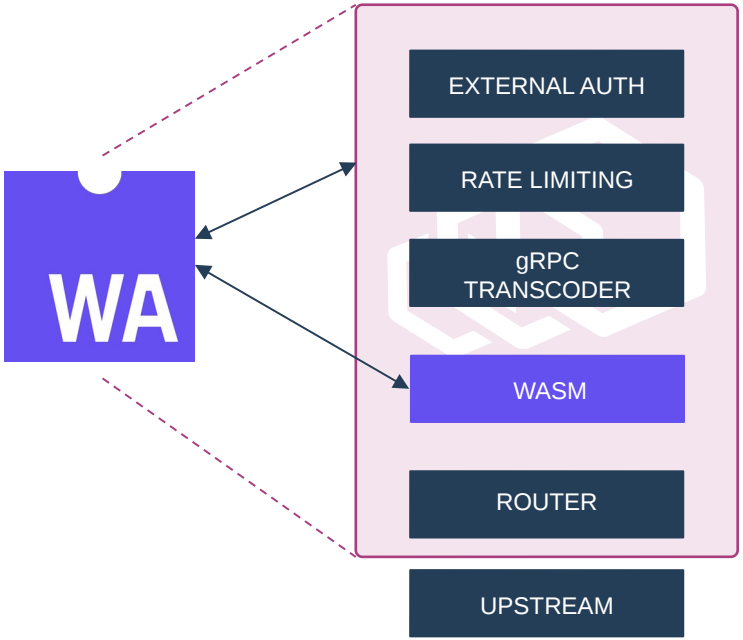
Challenges in customizing Envoy Proxy



- Must maintain efficient performance for the data plane
- Writing custom filters is difficult and limited to C++
- Changes built into Envoy, maintain separate distro
- Hard dependencies, cascading failures
- Must stop and recompile

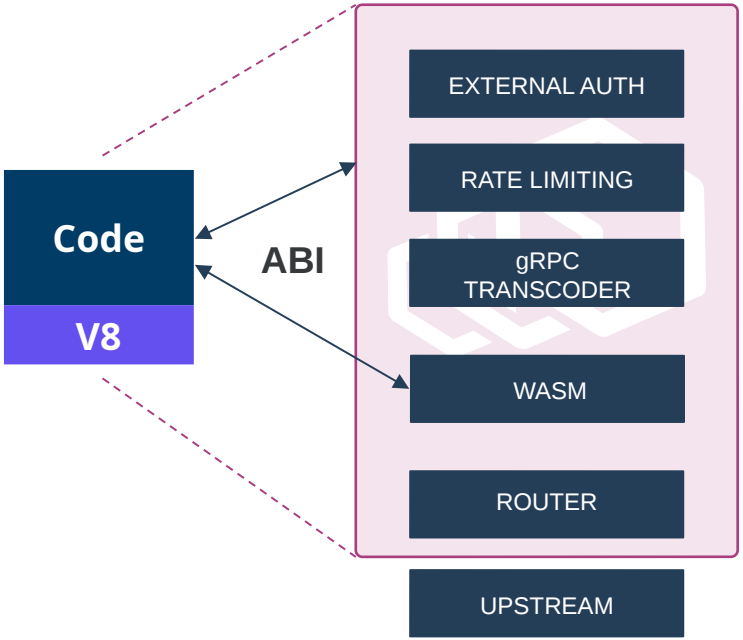
solo.io

WebAssembly in Envoy Proxy



solo.io

WebAssembly in Envoy Proxy



ABI: Application Binary Interface

HTTP (L7) extensions

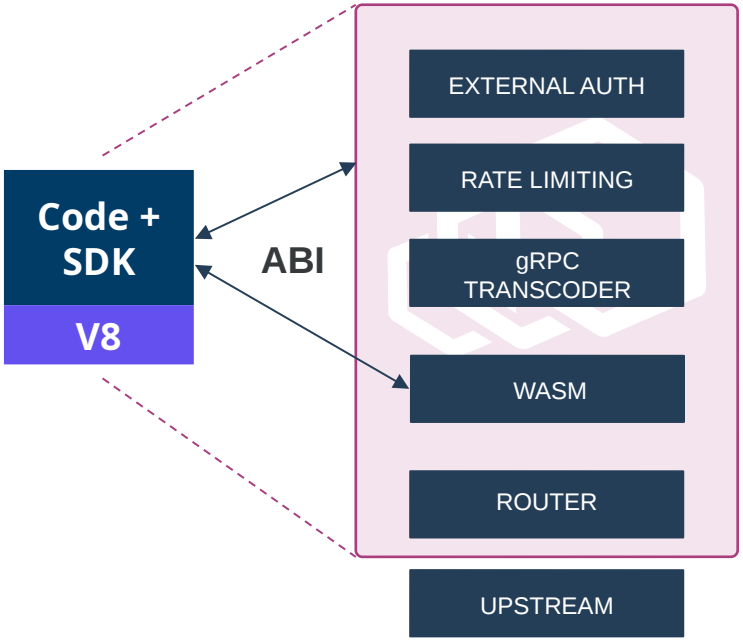
`proxy_on_http_request_headers`

- params:
  - `i32 (uint32_t) context_id`
  - `i32 (size_t) num_headers`
  - `i32 (bool) end_of_stream`
- returns:
  - `i32 (proxy_action_t) next_action`

Called when HTTP request headers are received from the client. Headers can be retrieved using `proxy_get_map` and/or `proxy_get_map_value`.

solo.io

WebAssembly in Envoy Proxy



SDK Includes the following with more coming soon.

solo.io

WebAssembly Hub

- Developer and operator workflow to build, share, and deploy
- Team and user management
- Operator uses declarative CRD
- Modules are OCI images
  https://github.com/solo-io/wasm-image-spec

solo.io

Demo

solo.io

About Us - We're Hiring!
solo.io

WebAssembly Hub
webassemblyhub.io

Service Mesh Hub - Community meetings starting June 17th
github.com/solo-io/service-mesh-hub

Join the Community
slack.solo.io