THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

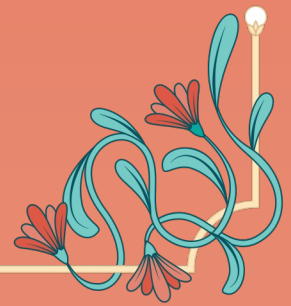
Embedded Linux Conference

Embracing Service-Oriented Connectivity Across OpenStack and Kubernetes

Rei Shimizu & Mitsuhiro Tanino

LY Corporation

THE LINUX FOUNDATION



Speaker



Rei Shimizu

Senior Software Engineer, LY Corporation

- Working for Private Cloud at scale, mainly working for network products like L4LB, L7 Proxy, Service Connect (Today we will talk about it!)
- Previously contributed to Envoy, Istio and their ecosystems for several years

Speaker



Mitsuhiro Tanino

Senior Software Engineer, LY Corporation

- Operates Flava, LY's OpenStack-based private cloud, at scale
- Contributed to OpenStack Cinder and Kubernetes sig-storage for several years
- Previous talks: KubeCon Europe 2026, KubeCon Japan 2025, KubeCon North America 2017

Agenda

1. Our platform and the connectivity problem
2. Our defence models and the issue in “connectivity”
3. The Paradigm Shift
4. How It Works
5. Wiring it into OpenStack
6. What we’d tell you



Flava: Our private cloud platform

One cloud, born from two. Heterogeneity is the starting condition, not a phase.

Verda (LINE)

Hypervisors	**~15K**
Virtual machines	**~120K**
Baremetal servers	**~46K**

YNW (Yahoo! JAPAN)

Hypervisors	**~27K**
Virtual machines	**~220K**

As of 2024

Migration ongoing
since 2025

Flava

Hypervisors	**~1.4K**
Virtual machines	**~41K**
Baremetal servers	**~3.4K**
Availability zones	**(3 AZ per region)**

As of 2026/08

IaaS on OpenStack

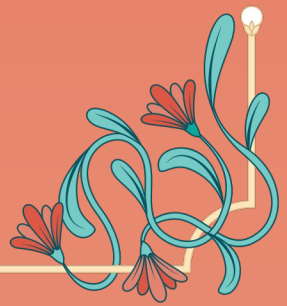
- Server
- VPC
- Load balancer

Managed and serverless on Kubernetes

- Kubernetes engine
- App runner
- Function

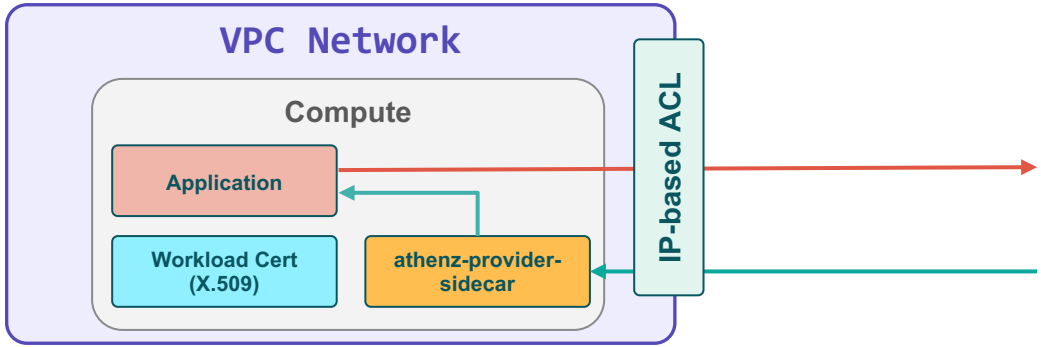
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE
Embedded Linux Conference

Our defense models and the issue in ”connectivity”



Our defense models: IP-based ACL & Athenz AuthN/Z

Defense in depth. Multiple enforcement points exist for service protection



IP-based ACL Policy

- Even in Private Cloud, using VPC is mandatory like Public Cloud
- Deny by default. Minimal default permit rules
- Need to open explicitly to communicate somewhere

App-level Auth N/Z

- Every application are forced to apply RBAC
- Identity is installed to each compute resource
- Based on the identity, this sidecar-proxy applies policy to each RPC call - policy only, no traffic control.

What is Athenz?

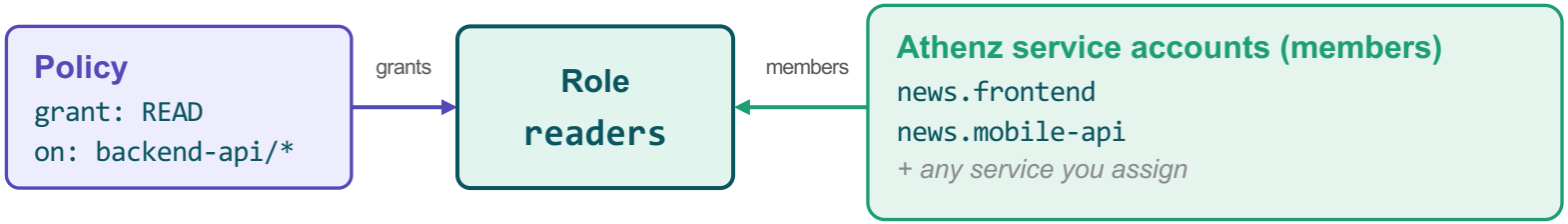
Open-source X.509 identity and fine-grained access control — AuthN + AuthZ.

Role-Based Access Control

- Policies attach to roles, not IP addresses
- Permissions follow the identity, not the network location

Proven at Scale

- Built at Yahoo, open-sourced in 2016
- Covers both provisioning and runtime authorization



Roles Decouple Policy from Identity

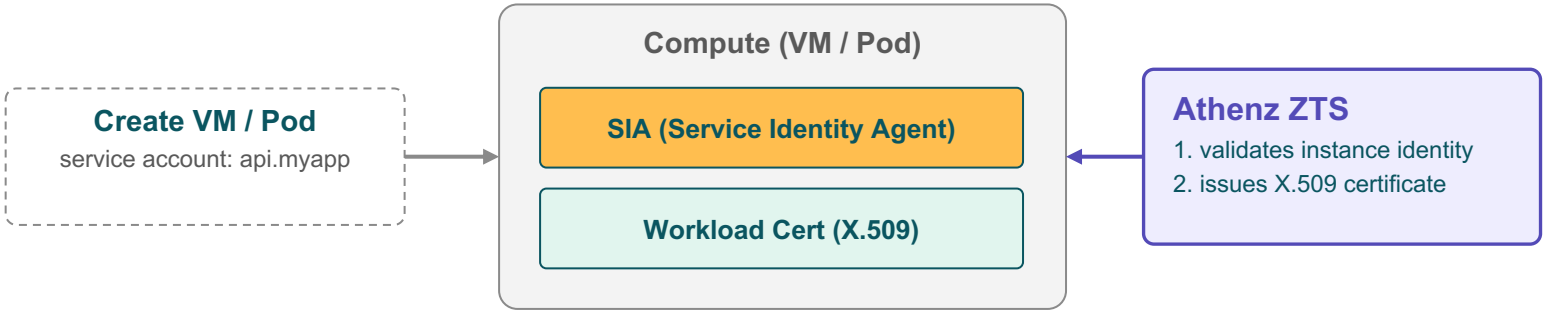
- Policies grant actions on resources to a role, never to a service directly
- Many service accounts can share one role's permissions

Assignment Is the Only Moving Part

- Add a service account to the role to grant access
- Remove it to revoke access instantly — no policy rewrite

CopperArgos - Workload Identity with X.509

Every VM or Pod is minted a short-lived X.509 certificate the moment it boots.



Certificate Issuance

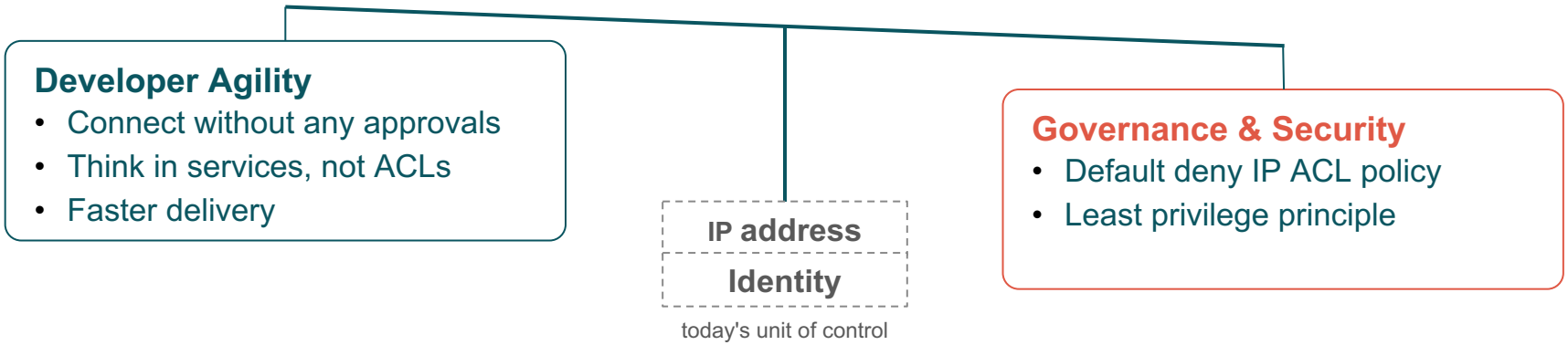
- SIA on the instance generates a key pair and CSR
- ZTS validates the identity document, then mints a 30-day cert

Authentication

- The workload presents this certificate for mTLS
- SIA refreshes the certificate automatically before it expires

The certificate is the identity - issued the moment the instance boots.

The issue: agility versus governance



What we had

VPC + IP ACLs at L3/L4, Identity at L7.
Every hop between them hand-wired.

The gap

A lot of work is required to connect between services. It blocks faster delivery of services due to platform-side restrictions.

Both sides are right. We did not need less security or less agility.

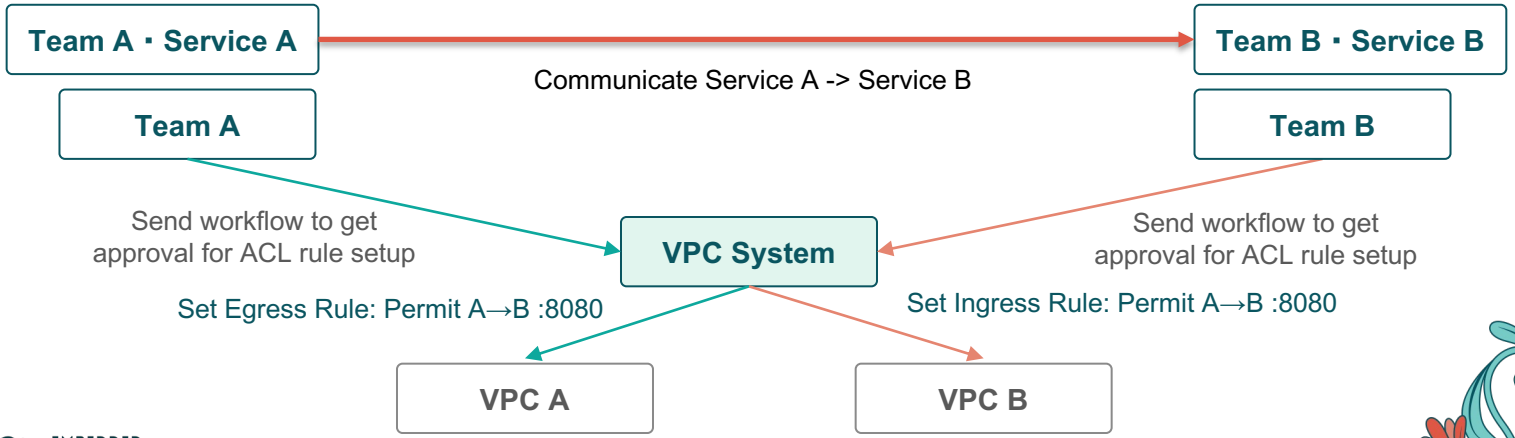
The issue: Where does the IP-based ACL restriction break?

Rules about addresses drift as soon as the addresses move.

Two-sided approvals
An ACL change on both source and destination VPC, per connection.

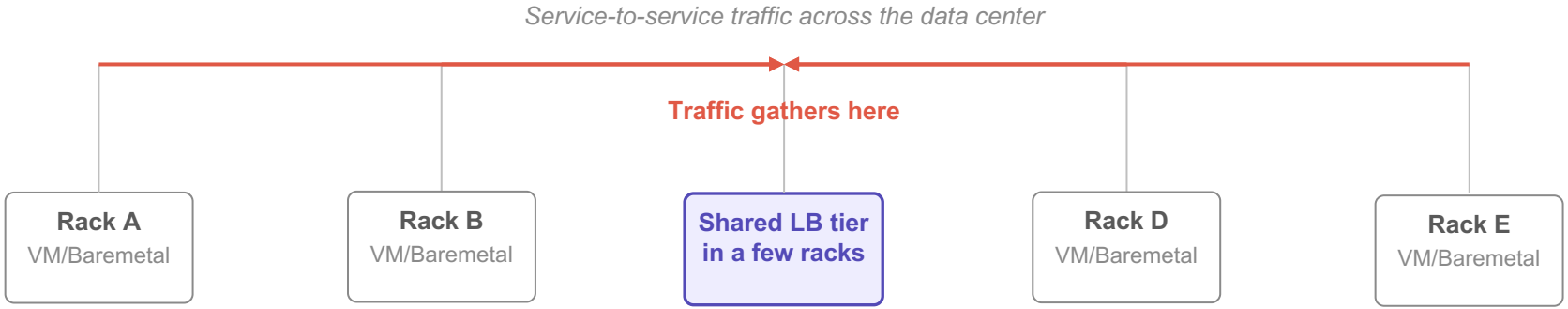
ACL drift
Autoscaling outruns rules. Operators open wide CIDRs to keep up.

No service context
IP addresses reveal rather neither service names nor context, making impossible to determine the purpose.



The issue: A shared LB tier concentrates east-west traffic

Our cloud LB is shared by many services and deployed in only a few racks.



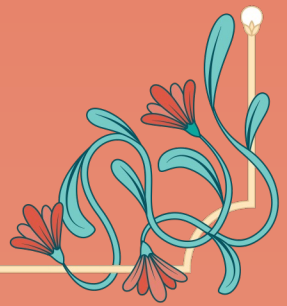
Traffic concentration
East-west traffic from many services gathers at the shared LB tier. Bandwidth use becomes uneven across the data center.

Traffic control far from workloads
Traffic controls happen at the shared tier, away from the clients that send and receive requests.

We need client-side traffic control close to each workload

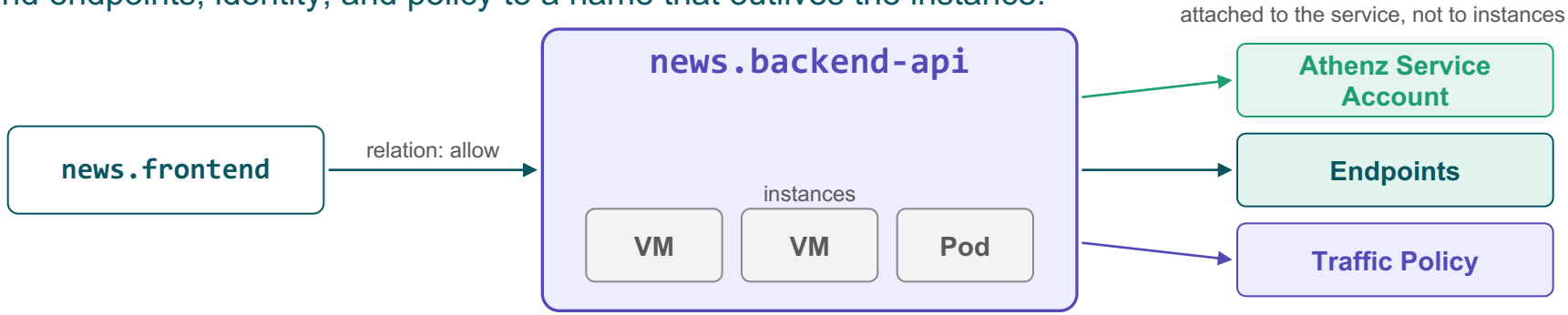
The Paradigm Shift

One abstraction — the service — becomes the unit of control



Paradigm Shift - Make “Service” as the unit of control

Bind endpoints, identity, and policy to a name that outlives the instance.



What the service holds

- A stable name
- Its Athenz service account. Access Control Policy is also related with it.
- Relations to other services
- Traffic Policy connecting to the service (e.g. LB, Retry, Circuit Breaker)

What the platform derives

- Service Discovery: It returns live endpoints. The returned value is used not only LB, but also L3/L4 ACLs.
- Add/Remove the endpoints to the Service, following to its lifecycle

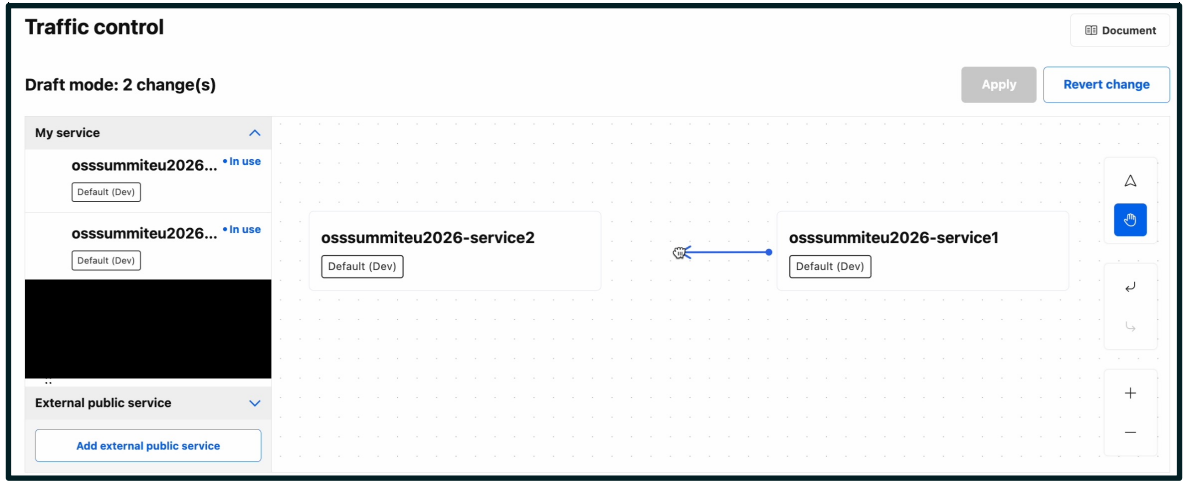
Paradigm Shift – All we need is abstraction of “connection”

The user's intent is just to connect. Today, that alone triggers ACL openings and permission grants.

1 Search & import

Search for the service you want to reach and import it.

2 Connect



Perform the connect action on that service.

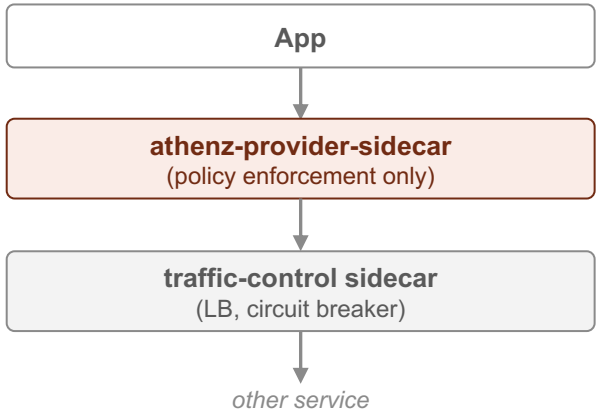
3 Role & service account

Role and service account are assigned as part of connecting.

Data Plane enforces policies between service-to-service communication

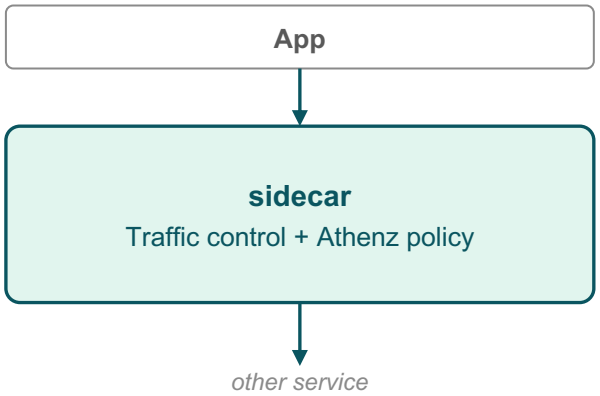
One sidecar per compute — traffic control and Athenz policy enforced in the same hop.

Before: Two Sidecars, Two Hops



Policy and traffic control lived in separate sidecars — every call paid for two hops.

Now: One Sidecar, One Hop



Transparent traffic control and Athenz policy — enforced in the same hop.

Service mesh similarities and differences

The mechanics are familiar, but the scope and platform are different.

Same as a traditional mesh

- Control plane distributes config; a sidecar enforces it
- Traffic controls (e.g., LB, Circuit Breaking, Retry)
- Policy evaluated on every call, not at the edge

What is different

- Scope is the whole private cloud, not one app's microservices
- Baremetal, VM, Pod and PaaS are equal citizens
- It drives L3/L4 VPC ACLs — the network follows the service relation
- Identity is issued at boot by the platform, before any mesh

We didn't set out to build a service mesh. A mesh-like connectivity model emerged, not by design, but as a consequence of the constraints we had to satisfy.

We call this "service-oriented cloud", and built a system named "Service Connect"

Why we built it in-house

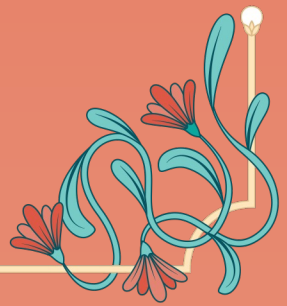
Not because we wanted a mesh. The constraints ruled the existing ones out.

Requirement	Kubernetes-Native Mesh	Service Connect
Baremetal + VM + Pod + PaaS	✕ Pods only	✓ one model for all
Equal citizenship	✕ VMs as second-class	✓ same identity, routing, policy
Drive our VPC default-deny ACLs	✕ no integration	✓ ACLs follow the service relation
Sidecar proxy and sidecarless	✕ **sidecar only**	✓ **both modes supported**

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE
Embedded Linux Conference

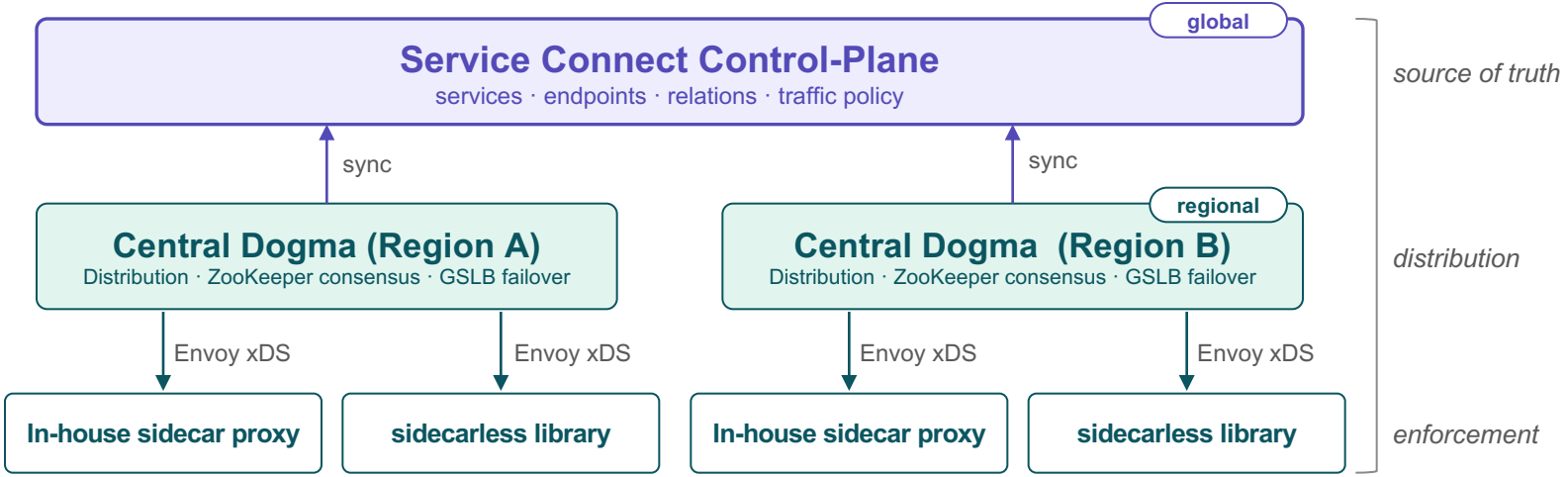
How It Works

Inside the control plane and data plane we built



Control-Plane architecture

One global registry, regional distribution, standard xDS at the edge.



Nothing below Service Connect Control-Plane is proprietary protocol. Data-Plane speak Envoy xDS.

Two data planes, one control plane

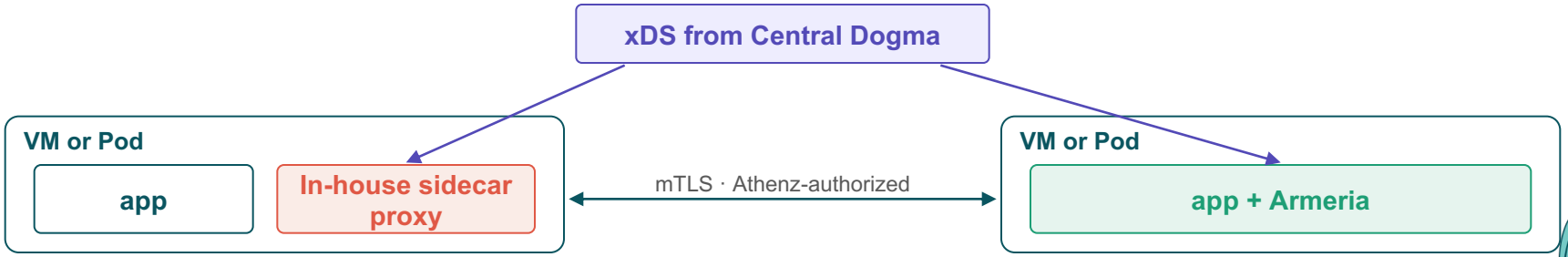
A sidecar for everything, in-process for the JVM fleet.

Sidecar: in-house sidecar proxy

- polyglot apps and PaaS workloads
- built with Rust to achieve runtime safety, and small resource consumption rather than exising solutions

Proxyless: Armeria

- open-source JVM RPC framework
- native xDS client, in-process mTLS
- zero proxy hop, nothing to operate



Same identity, same policy. Different footprint.

The control plane does not know which one it is talking to.

What's Central Dogma & Armeria?

Both open-sourced by LY Corporation (LINE) — building blocks under our control and data planes.



Central Dogma

- Open-source Git-based, version-controlled configuration repository from LY Corporation
- Multi-master replication via embedded ZooKeeper
- Provides Envoy xDS-compatible feature

https://line.github.io/centraldogma/



Armeria

- Open-source JVM microservice framework from LY Corporation, built on Netty
- Serves gRPC, Thrift, and REST on a single port
- Our proxyless data plane

https://armeria.dev/

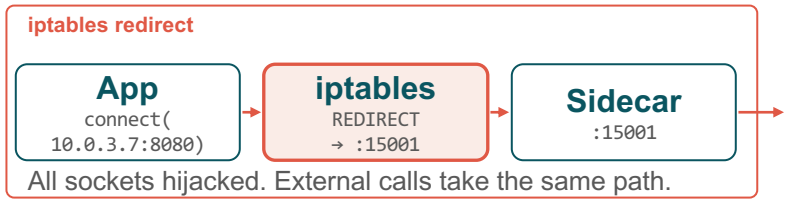
Two LY Corporation projects — our config store and proxyless data plane.

Non-invasive routing via local name resolution

No iptables, no privileges. The app talks to a name, and the name is always local.

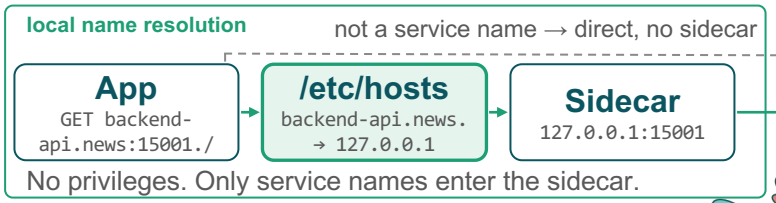
What iptables interception costs

- Istio-style meshes redirect every socket through the sidecar with iptables / tproxy.
- Needs container privileges (`CAP_NET_ADMIN`) or CNI plugin integration.
- Every egress call pays the hop, including third-party and external traffic.

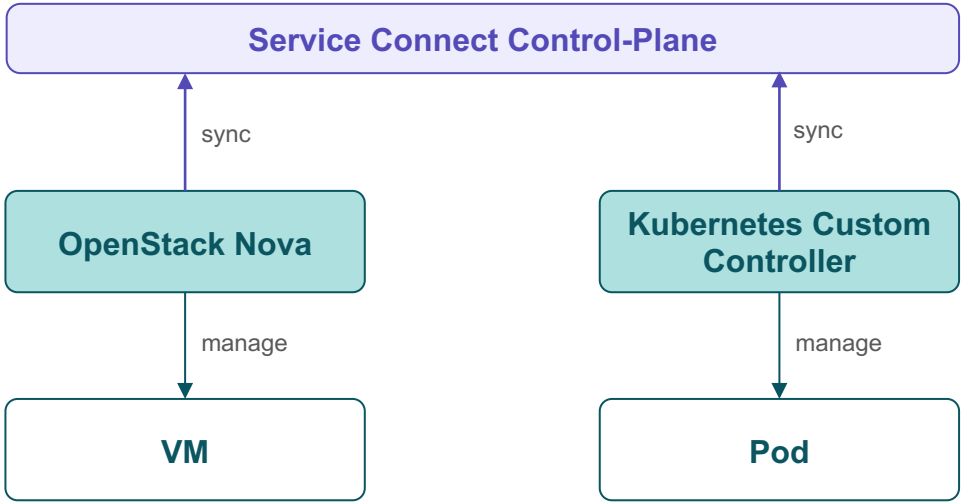


What we do instead: /etc/hosts

- Service names resolve locally. The trailing dot always maps to `127.0.0.1`.
- The app sends plain HTTP/gRPC to the service name; the sidecar listens on a loopback port.
- Anything that is not a service name bypasses the sidecar. Zero impact on unmanaged traffic.



Resource state synchronization



One synchronization model for VMs and Pods

- The Service Connect Control-Plane must track the current state of every managed resource, including deletions.
- Nova and Kubernetes components drive the same lifecycle synchronization for VMs and Pods.
- We extended OpenStack Nova and Kubernetes to implement this synchronization.

Realizing it on Kubernetes as a Service in our cloud

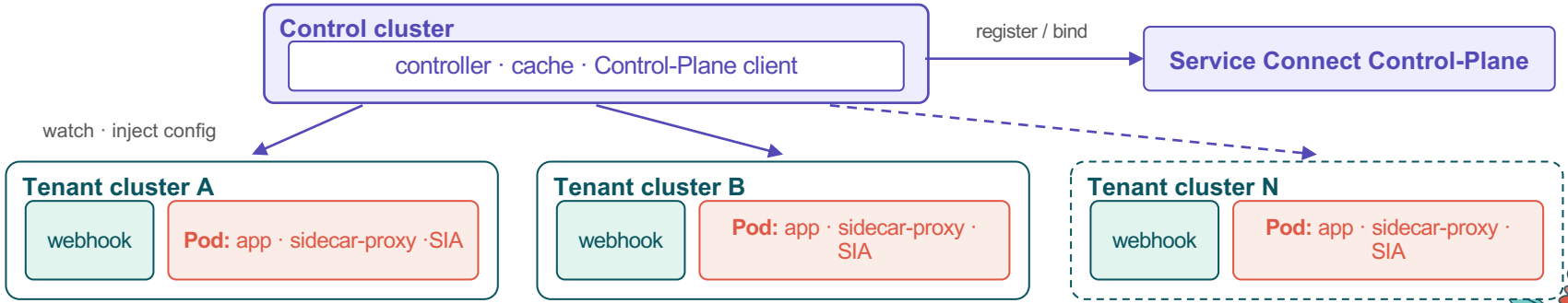
Kubernetes as a Service (KaaS): one tenant cluster per user. One control cluster runs them.

In each tenant cluster

- one MutatingAdmissionWebhook
- injects sidecar-proxy, hostAliases
- no reconcile loop locally

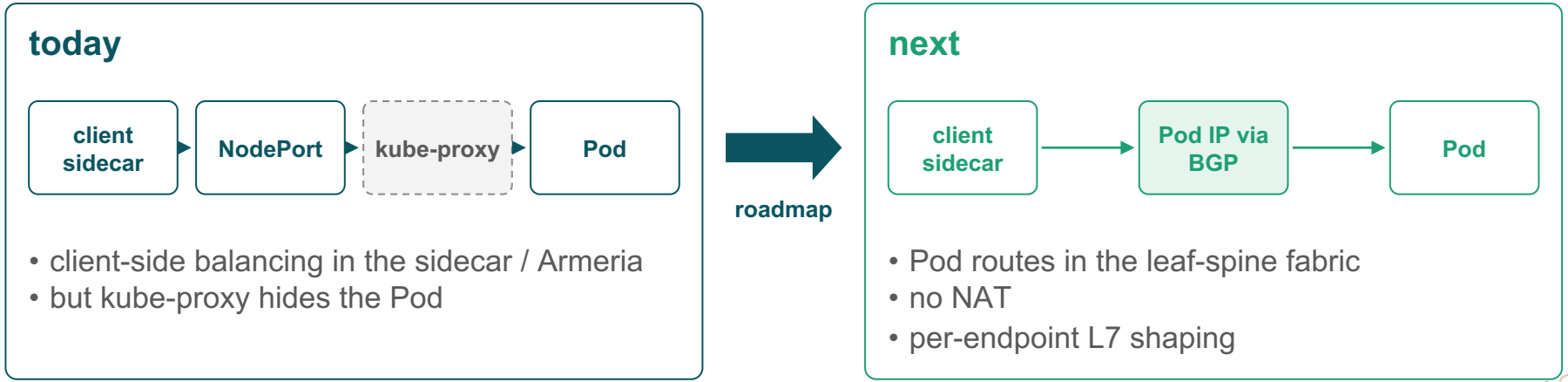
In the control cluster

- watches NodePorts across tenants
- registers and deregisters endpoints
- caches topology, shields backends



East-West routing in Kubernetes: today and future

No central LB on the service-to-service path — east-west traffic stops concentrating.



Same direction as our VM networking: routed L3, per-instance host routes. The OpenStack side runs on it today.

THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

Embedded Linux Conference

Wiring it into OpenStack

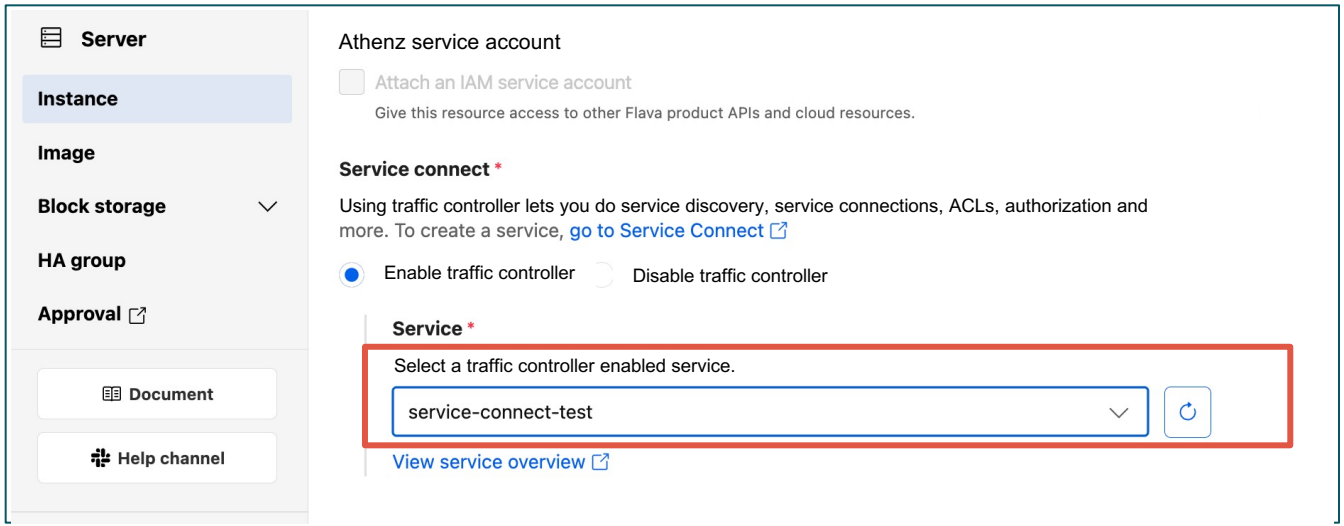
Registration, identity, and policy for VMs, without turning Nova into a platform



What the user sees: one field in

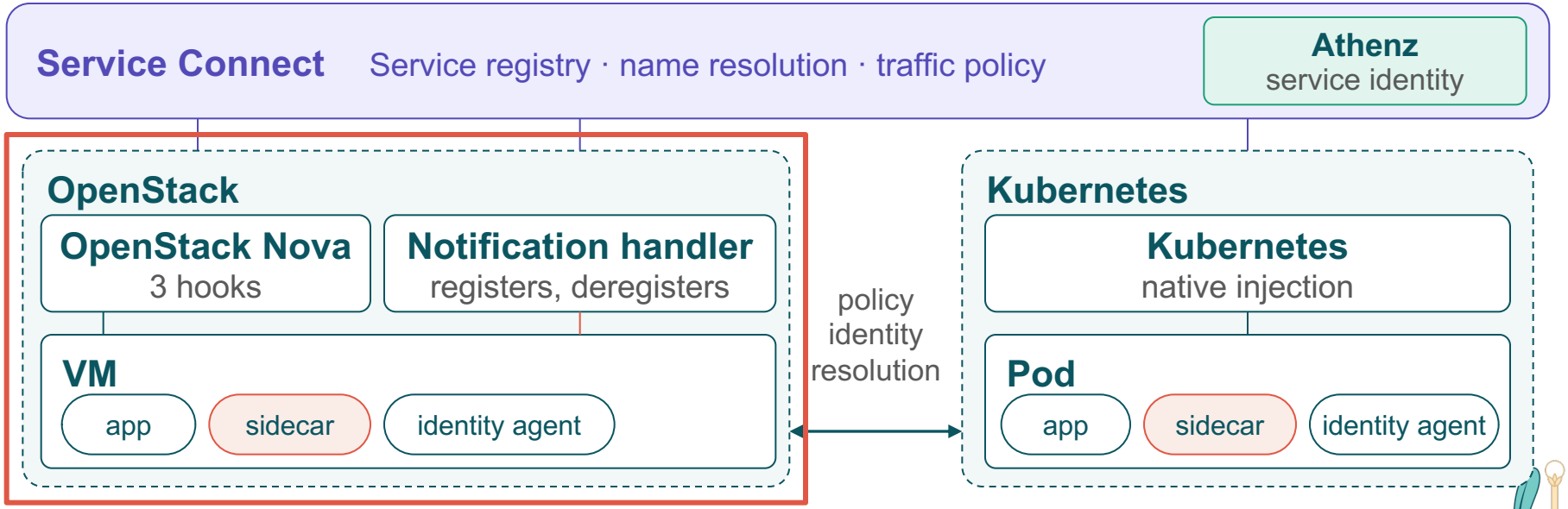
One field at an instance creation time. Everything else appears by itself.

Create a server



Pick the service the VM belongs to. In the CLI, this is one metadata key: service_id.

Where the pieces go



Two entry points, one registry. From here, we focus on the left side.

Integrating Service Connect with OpenStack

Everything a Kubernetes Pod gets at creation, a VM must also get at creation. In OpenStack, creation is Nova's job.

What a VM needs to join Service Connect

1. Its endpoint registered in the Control-Plane
2. A service identity it can present (Athenz)
3. A sidecar listening on the registered port

Where a VM is created: Nova



For a VM, creation is Nova's job.

So the hooks go into Nova.

Registration and identity are guaranteed at boot. The sidecar is installed by the user.

Nova is a hook, not a brain

But Nova is upstream open source

Every line we add is a line we carry across upgrades. Business rules of our platform do not belong inside it.

Our rule: Nova is a hook, not a brain

Nova asks, waits, and protects. Deciding and registering happen outside, in the Notification handler.

Three hooks go into Nova, one at each step of VM creation:

Hook 1 · **ask**

Hook 2 · **wait**

Hook 3 · **protect**

Hook 1 [ask]: boot-time identity injection

The user names the service. The registry decides who the VM is.

```
# One extra metadata key at create time. That is all the user sees.
$ openstack server create --property service_id=news.backend-api \
    --image ubuntu --network vpc-net my-vm
```

service_id
from the user

→

Service Connect Control-Plane
returns the service's Athenz service account

→

Athenz service account
set by Nova, not the user

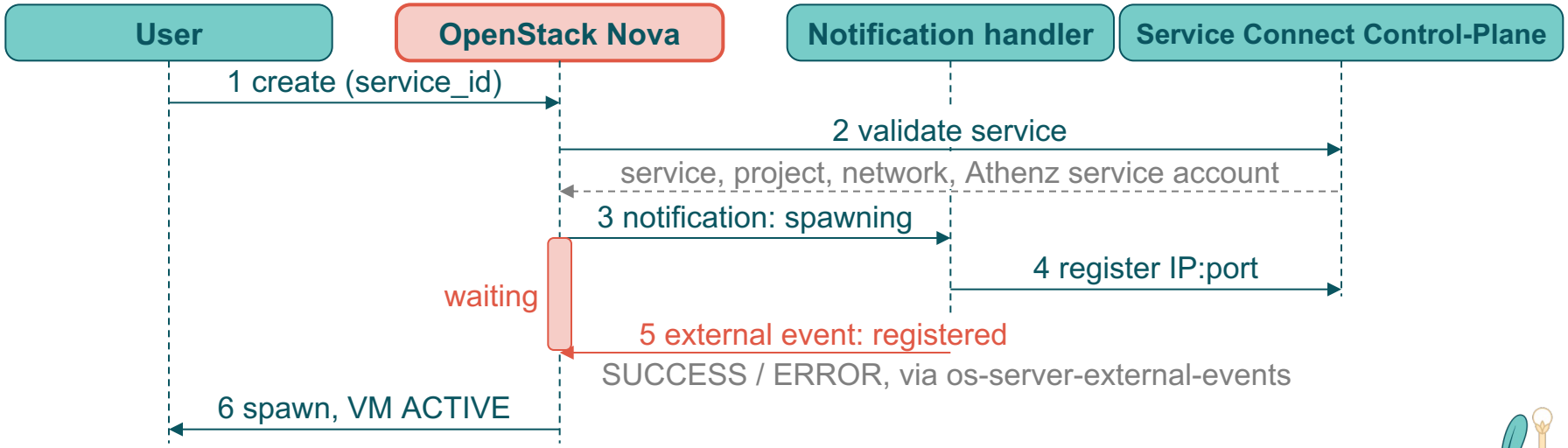
Nova checks with the Control-Plane before the VM exists:

- The service exists in the registry
- It belongs to your OpenStack project (= your Athenz domain)
- It already has an Athenz service account

Identity is decided by the registry, not typed by the user.

Hook 2 [wait]: the registration handshake

Nova waits, but doesn't know why. The Notification handler knows why, but never touches Nova.



Same wait path Neutron uses for network-vif-plugged. Nova only asks and waits; it never registers anything. The port is a platform default, not per-VM. Tenants correct it in the Service Connect console.

Design decision: fail fast, don't reconcile

A half-working VM is more dangerous than a dead one.

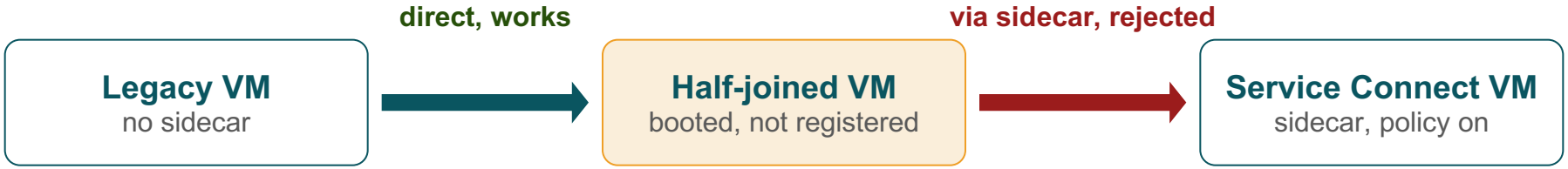
Registration fails at boot. What now?

A. Boot anyway, fix later
Higher boot success rate. But identity is injected only at boot, so there is nothing to fix later.

Selected

B. Fail the VM
Lower boot success rate. Nothing partially registered ever reaches production.

Why A is worse than it looks

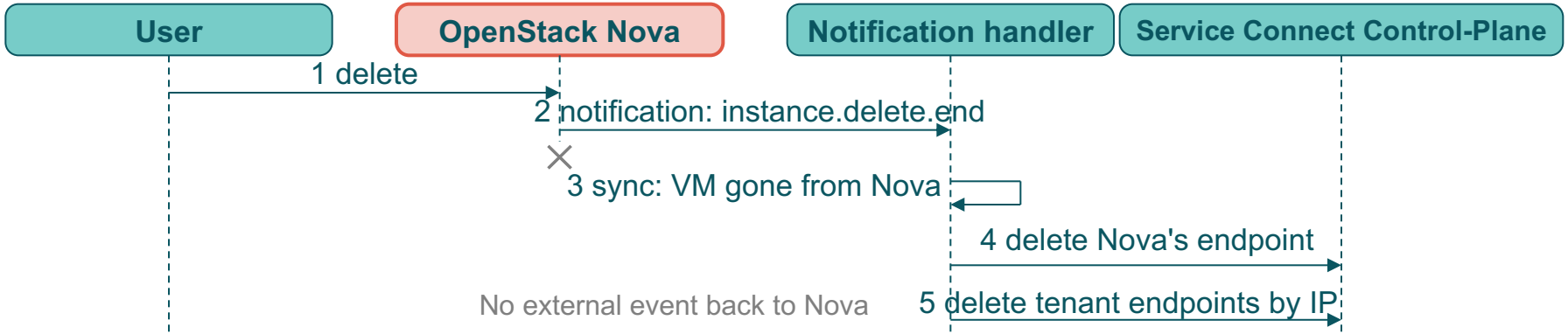


Registry down at validation → 503, no VM created. Registration failure or 600 s timeout → rescheduled; ERROR once retries run out. Never ACTIVE half-registered.

We chose fewer boots, not hidden failures.

And on delete: nobody waits

Create blocks so an unregistered VM never serves. Delete cannot make that mistake.



What Nova added for delete

Nothing. Zero lines. The existing delete notification is enough.

Known limits

Failures are logged, not retried. The next VM on that IP cleans up old entries at boot.

Deletion is fire-and-forget: Nova emits, the handler deregisters, nobody waits.

Hook 3 [protect]: identity keys are write-once

Tenants set the key at boot. Only admins can change it afterwards.

What is protected

service_id and athenz service account are policy targets on every write path.

How

Nova spec in review · nova-specs #977339

One oslo.policy helper. The rules live in policy.yaml, not in code.

Execution order in instance create()

1. policy check on metadata
2. Nova injects athenz service account
3. Control-Plane validation

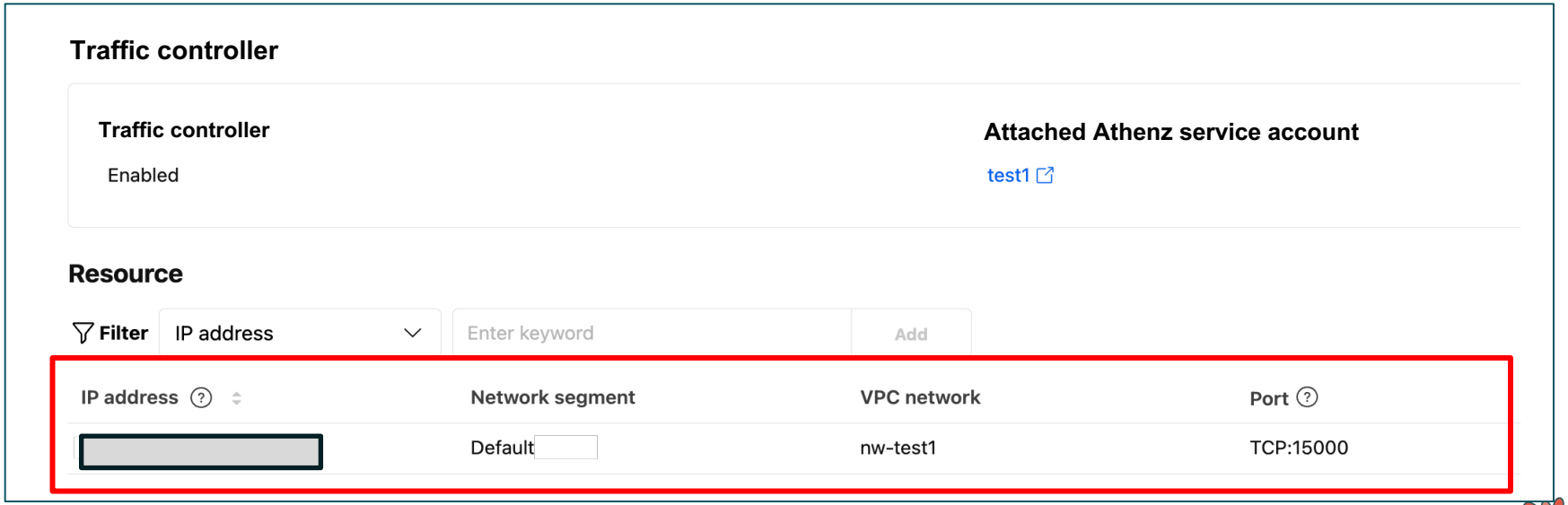
Nova adds the Athenz key after the check, so it never hits the policy.

```
# policy.yaml
server-metadata:create_on_boot:service_id: "@"
server-metadata:update:service_id:  "rule:admin"
server-metadata:delete:service_id:  "rule:admin"
```

Identity is injected once, then locked. Nothing needs reconciling because nothing can be rewritten.

What the user sees: a full endpoint out

Three hooks later, this is what the user gets. They never saw any of them.



The VM is already an endpoint of the service, on the platform default port.

No registry call, no identity setup, no policy configuration by the user.

What it cost, and where we are

Three costs. We expected all of them, so none became a problem.

Sidecar footprint

Ports and resources on each VM with Service Connect, and no managed update path yet.

How we live with it

Manual install first, automation second.

Registry on the boot path

A VM using service_id asks the registry at boot. If the registry is down or slow, that VM boots late.

How we live with it

The registry gets the same uptime target as the IaaS itself, and a VM asks it only once per boot.

Two paths to operate at once

Both the direct identity key and service_id are accepted today.

How we live with it

Both modes sit behind one config flag, and a request that sets both is rejected with a 400.

Where we are Flava Dev since May 2026, now rolling out to production. New services adopt first. Existing IP-based ACLs keep working; both models coexist by design.

What we'd tell you

1 **Make the service the unit of control.**

We stopped writing rules about IP addresses. One name carries the endpoints, the identity and the policy — for a VM or a Pod alike.

2 **Abstract the connection, not the network.**

The user's intent is just to connect. Service Connect turns that single intent into registration, identity and routing — no approvals, no per-address ACL rules.

3 **Keep the IaaS thin.**

Three hooks, seven upstream files, zero new extension mechanisms, no fork to carry. Everything that decides lives outside, in the Notification handler.



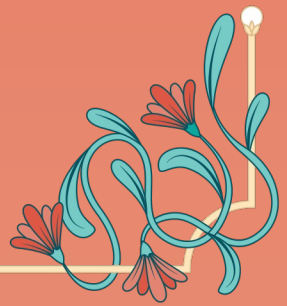
THE LINUX FOUNDATION
OPEN SOURCE SUMMIT
EUROPE

Embedded Linux Conference

Thank you. Questions?

Rei Shimizu and Mitsuhiro Tanino, LY Corporation

Athenz is a trademark of The Linux Foundation. OpenStack is a trademark of the OpenInfra Foundation.
Kubernetes is a trademark of The Linux Foundation.
Armeria and Central Dogma are open-source projects of LY Corporation.
All other trademarks are the property of their respective owners.



THE LINUX FOUNDATION



OPEN SOURCE SUMMIT



EMBEDDED LINUX CONFERENCE

EUROPE

