Adopting your infrastructure to go Multi-*

A migration story.

Kelsey Hightower ✓
@kelseyhightower

The whole industry can use a bit more pragmatism. Until the new stuff supports the old stuff then people will continue to use tools that do.

Telling people to rewrite everything (again) for the current platform of the month is how we got into this mess in the first place.

> **Simon Wardley #EEA** @swardley · Oct 18
>
> X : Do you really think kubernetes is a "waste of space"?
> Me : Yes. It's a future home for legacy and niches with managed kubernetes by Amazon already doninating. The real battle is up the stack and currently AWS is storming that one as well. Stop fighting the pointless battles.
>
> Show this thread

6:26 PM · Oct 18, 2019 · Twitter Web App

A migration story.

Erik Veld

Developer Advocate at HashiCorp

HashiCorp

Terraform

Consul

Packer

Vault

Nomad

Vagrant

HashiCorp

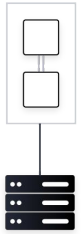
https://github.com/eveld/migrating-applications-with-service-mesh

How did we get ourselves into this mess?

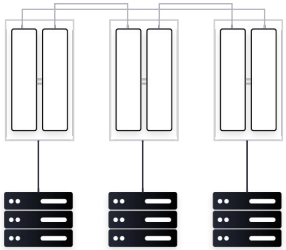
Multi-Service

What?

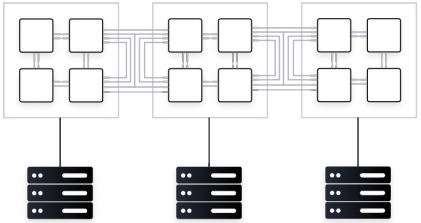
Moving from **monoliths to cloud-native applications.**



Single, Physical Server



Dynamic Virtual Machines



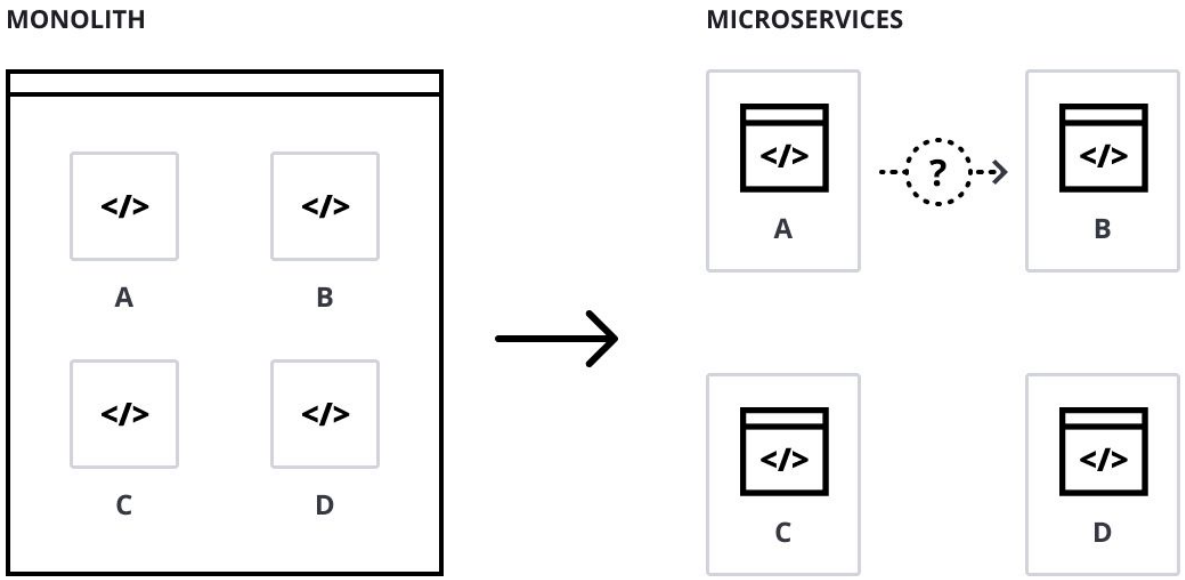
Smaller, Ephemeral Containers

Challenges?

- Handling **stateful** applications.
- **Migrating** between monoliths and microservices.
- **Organizational** challenges (Conway's law).
- **Securing** communication between microservices.

Challenges?

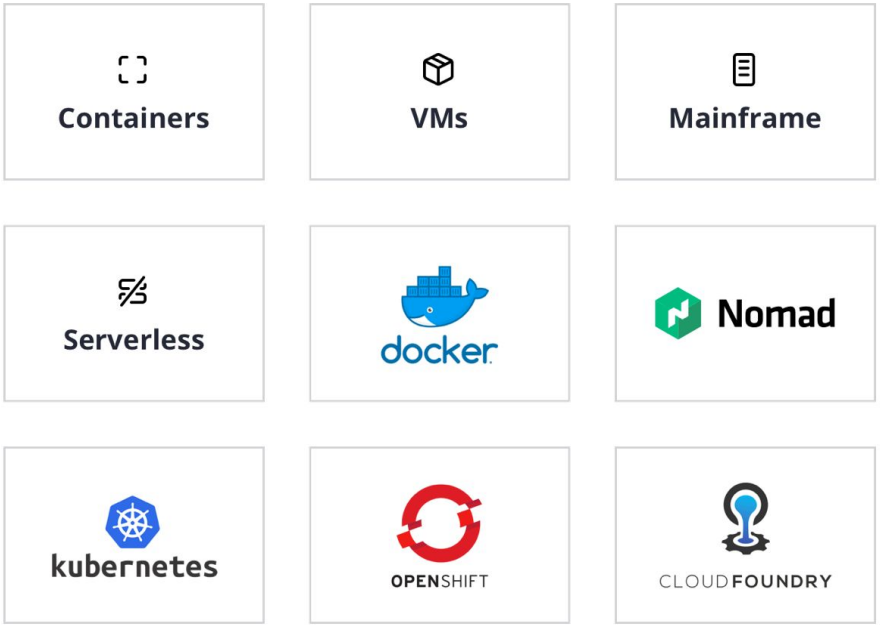
Moving from **function calls to RPC over the network.**



Multi-Platform

What?

Multiple runtime platforms and services from **Kubernetes to VMs, Containers to Serverless functions**.



Challenges?

- Vendor **lock-in**.
- Increased **operational complexity**.
- There is **no** single "**one size fits all**" approach for all of the platforms.
- **Security**.
- **Connecting** all the different platforms together.

Multi-Cloud

Challenges?

- **Connecting** all the different platforms together **x 10**.
- Increased **operational complexity x 10**.
- There is **no** single "**one size fits all**" approach for all of the platforms **x 10**.
- **Security x 10**.

Networking.

Challenges?

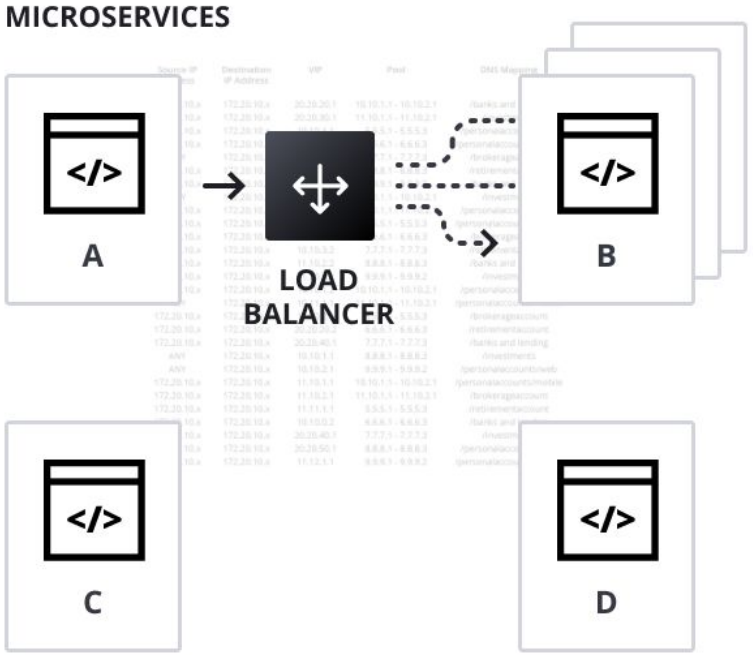
Many common challenges have to do with networking.

- **Naming things** — What do I name the service and how do I find it?
- **Security** — Which services are allowed to talk to the service?
- **Routing** — How do I route to the service?

Naming things and Routing (50%).

Load Balancers

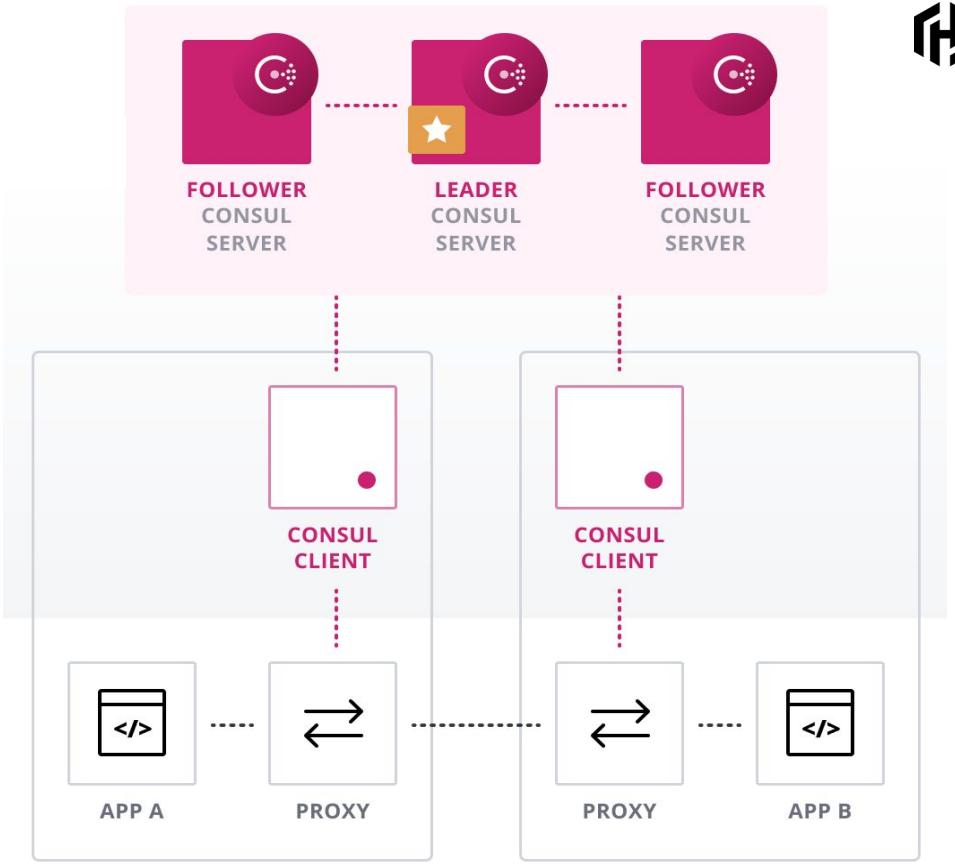
What do I call the service **and how do I find it?**



Service Mesh

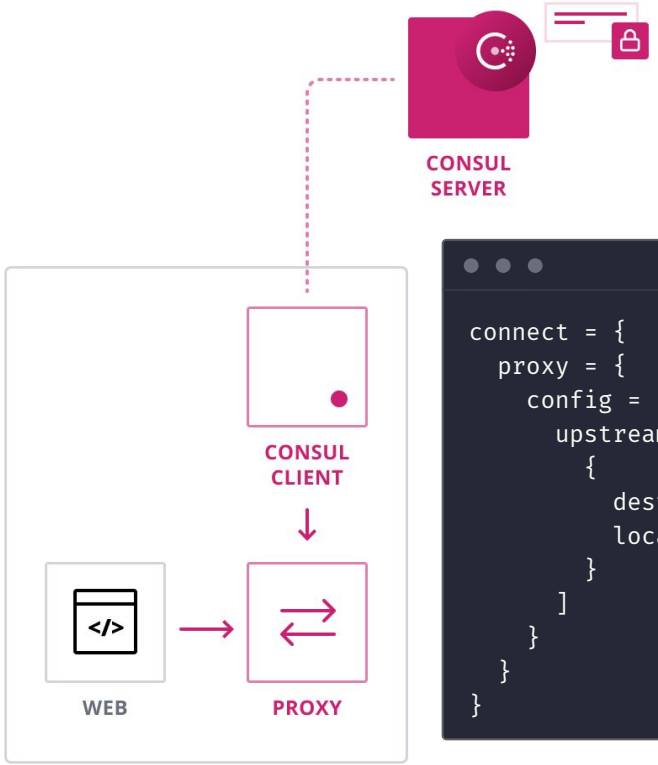
The **Control Plane** is responsible for Service Discovery, Authorization and configuring the proxies of the Data Plane.

The **Data Plane** is responsible for connecting services and routing data between them.



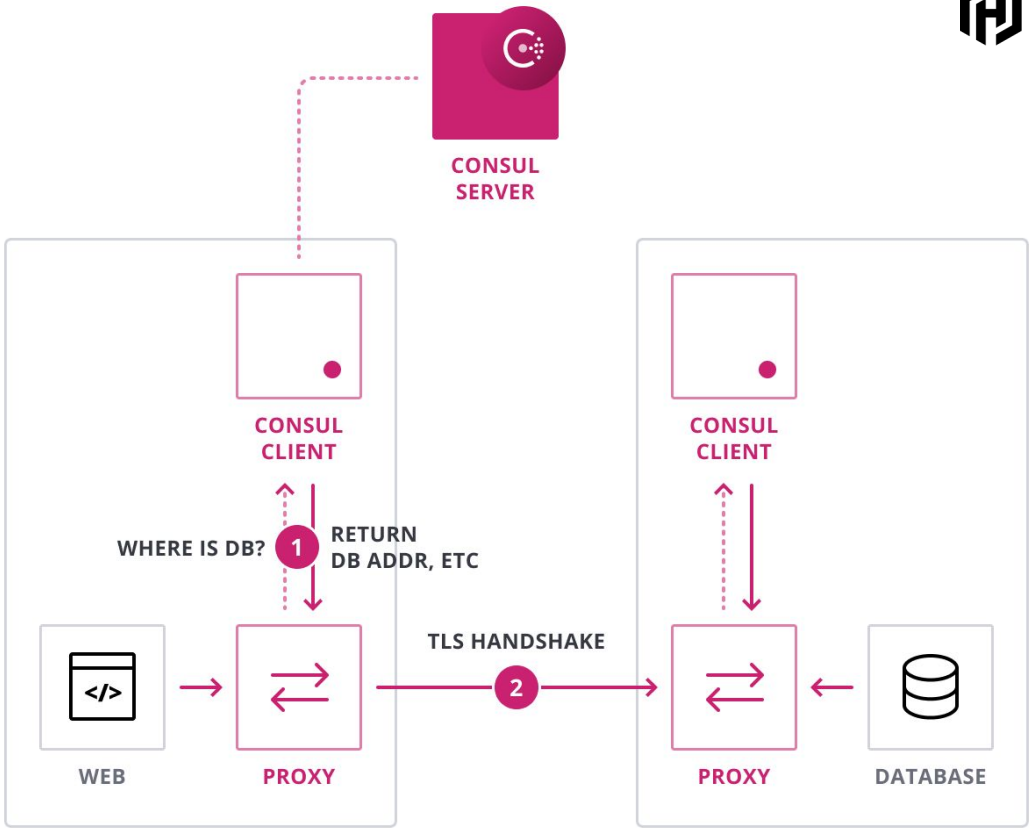
Configuration

- A proxy is **co-located** with the service instance that proxies inbound traffic.
- The Client agent **instantiates** the proxy and **registers** it as a service.
- The proxy is **configured** with a port that is used for the service and ports for any upstream destination that the service wants to connect to.



Service Discovery

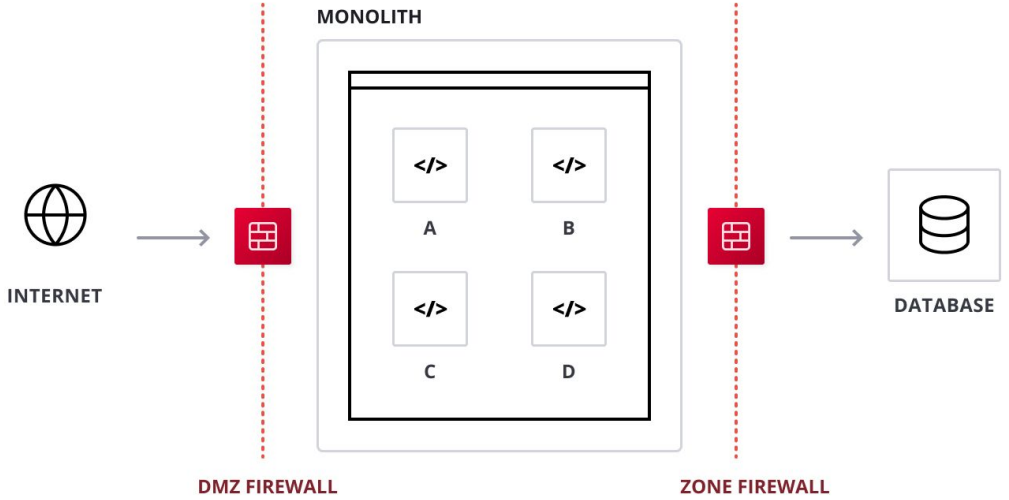
- The proxy of the web service uses service discovery to **request the location** of the DB.
- The local agent **returns the proxy's IP address/port** of a healthy DB instance.



Security.

In the beginning

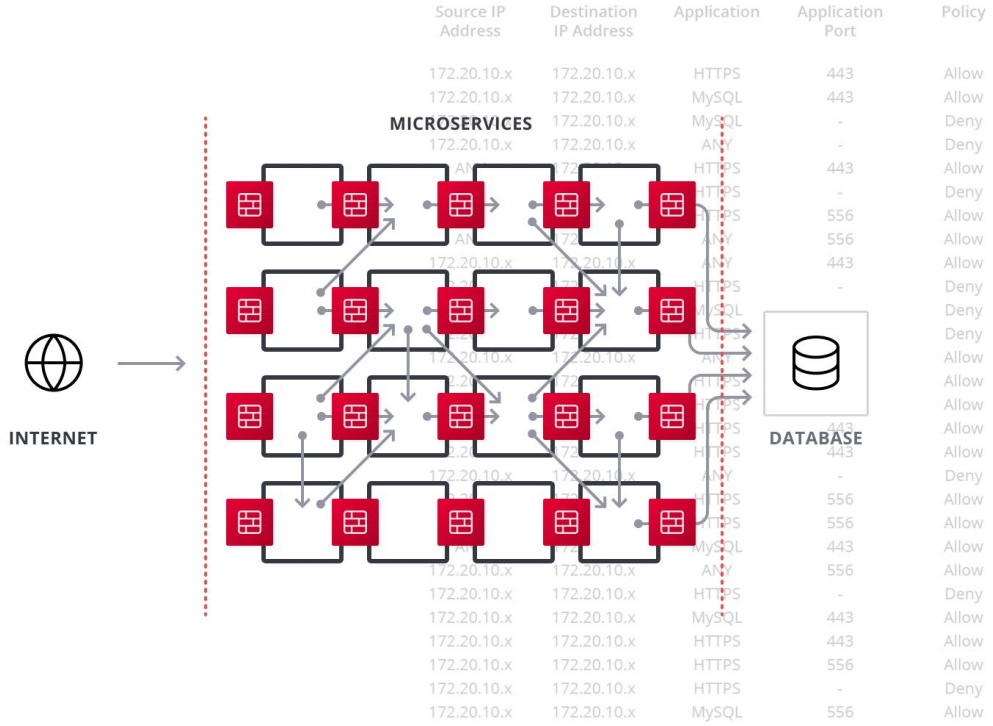
Access is controlled through firewall rules, determining **which IP and port is allowed to connect**.



ZONE	ZONE	SOURCE IP	DESTINATION IP	APPLICATION	APPLICATION PORT	POLICY
Untrust	DMZ	Any	172.20.10.x	HTTPS	443	Allow
DMZ	Trust	172.20.10.x	172.20.20.x	mSQL	443	Allow

Eventually..

With the growing complexity in dynamic environments, **managing these rules becomes impossible**.

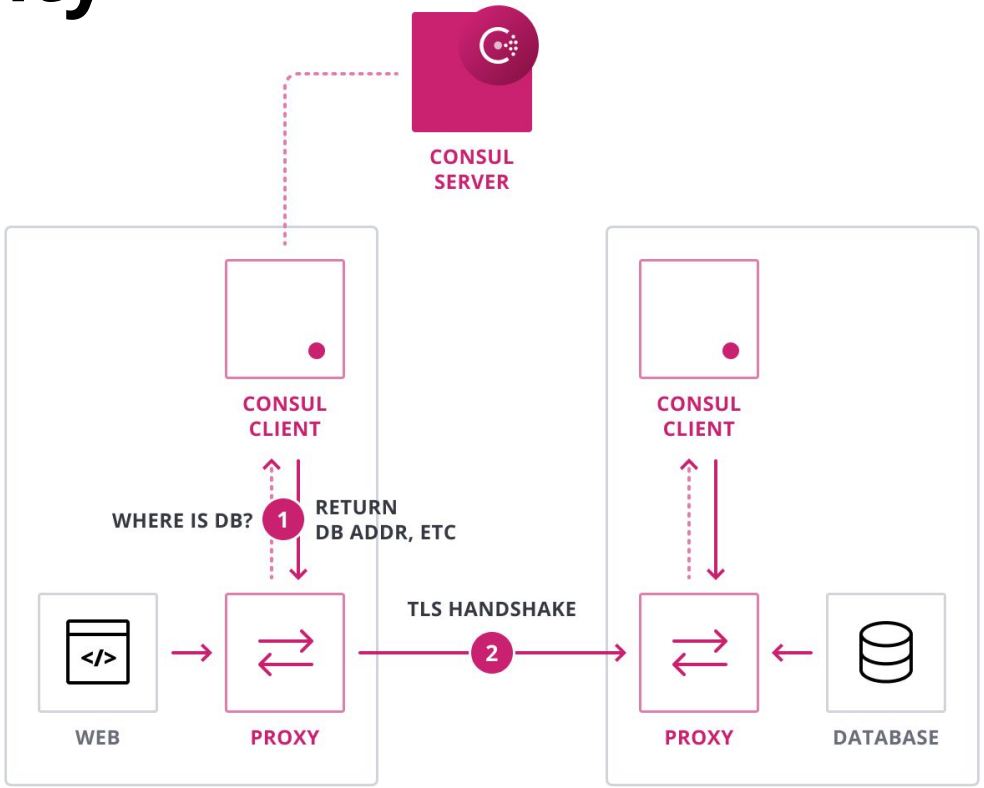


Intentions

The service access graph defines which services are **allowed or denied from communicating through intentions**.

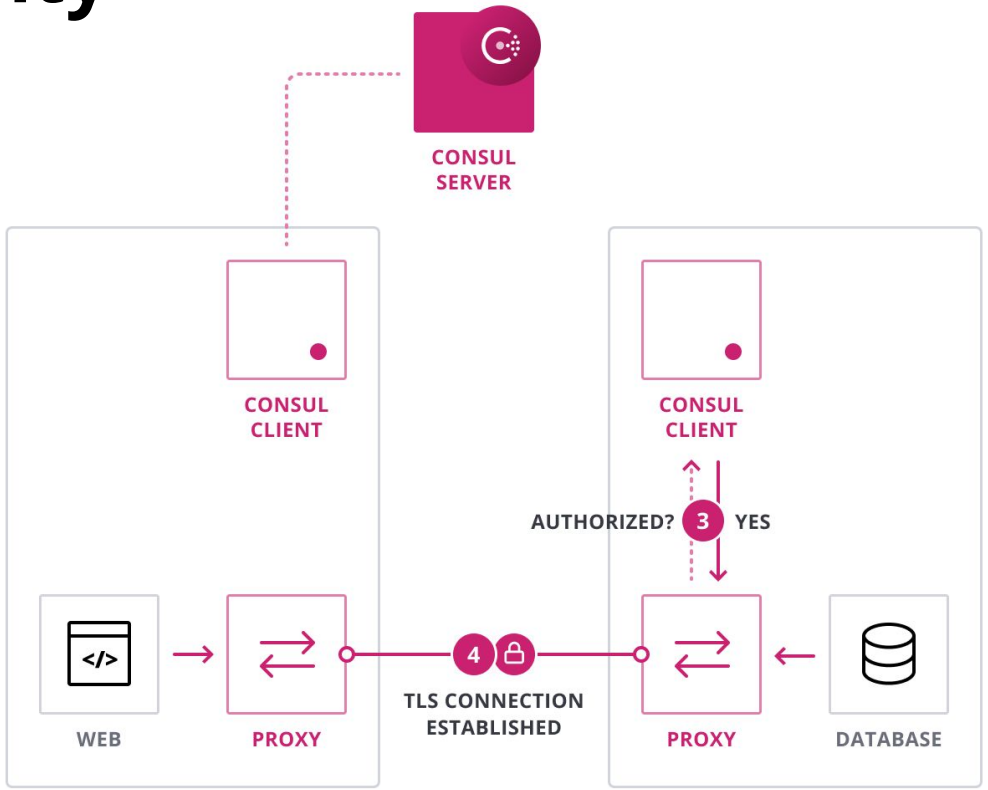
Service based security

- The local agent also returns the URI for the **expected identity** of the service it is connected to.
- Proxies between web and database start TLS handshake to **authenticate the identity**.



Service based security

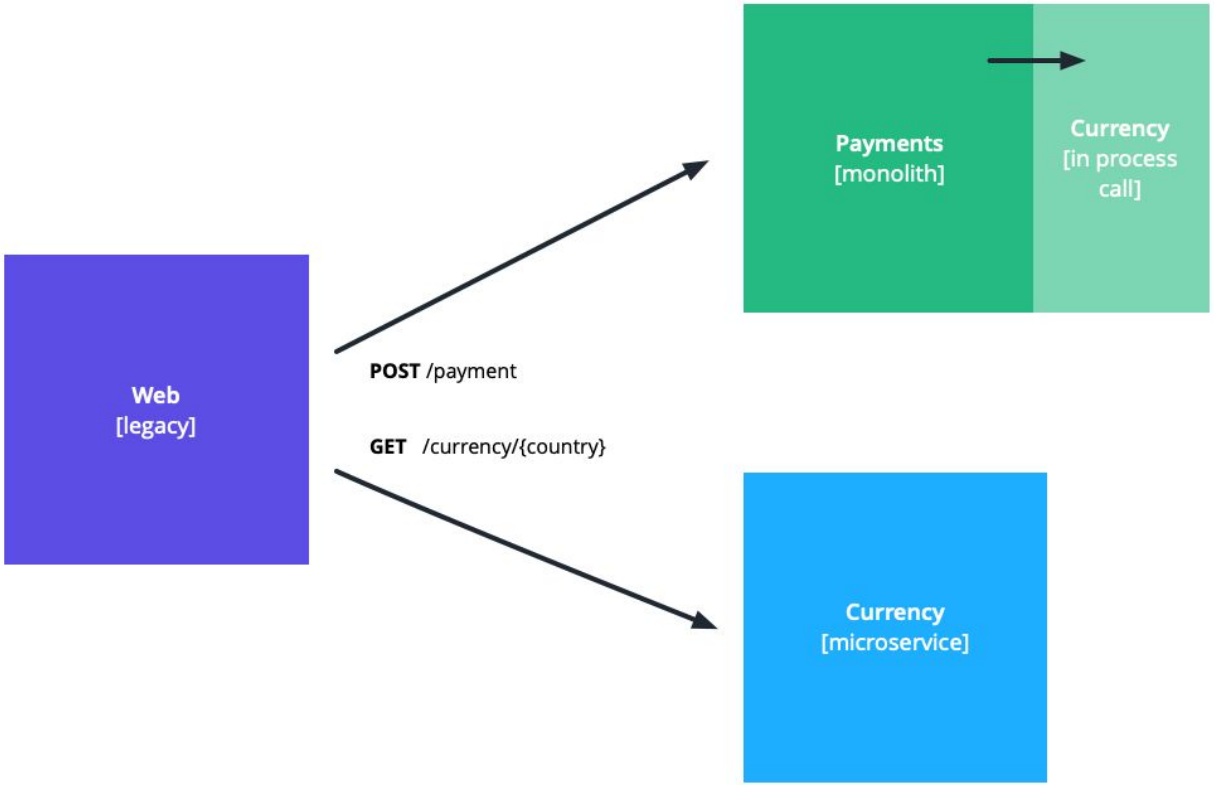
- The DB proxy sends the **authorization request** to its local agent.
- The local agent **authorizes the connection** based on locally cached intention.
- **Mutual TLS** is established.



Migrating from monolith to microservices.

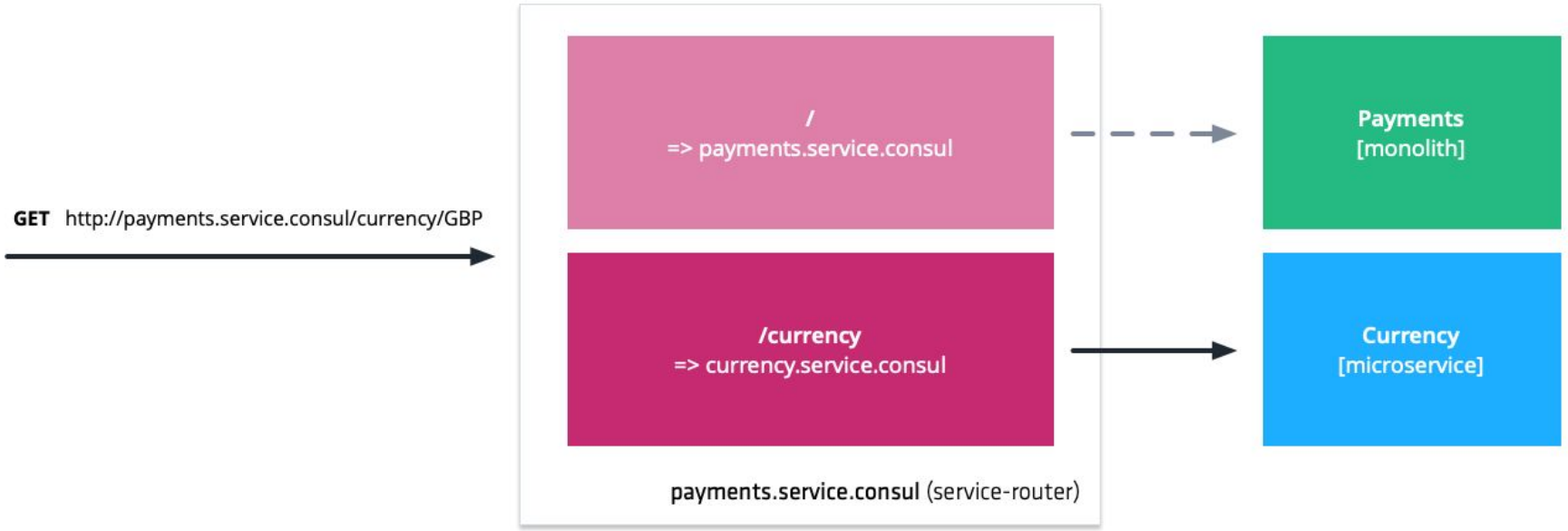
Current state

Desired state



Traffic Routing

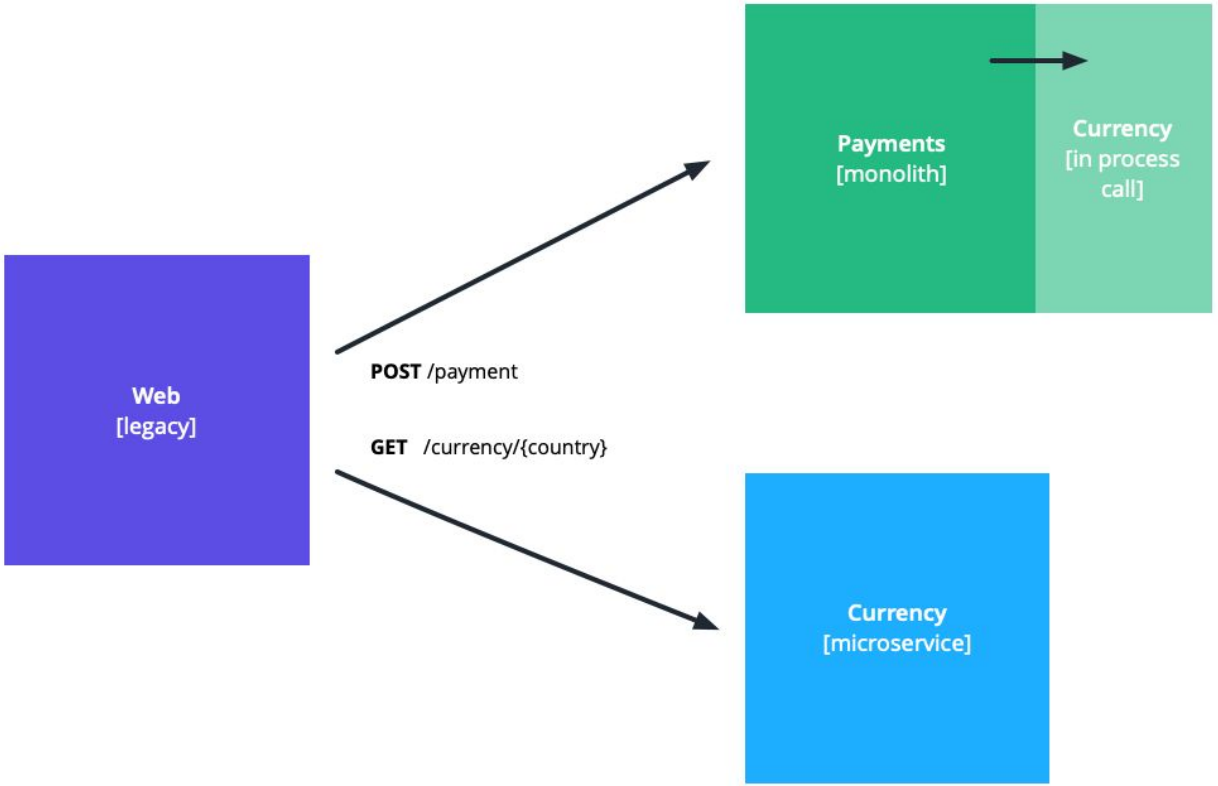
Service Router



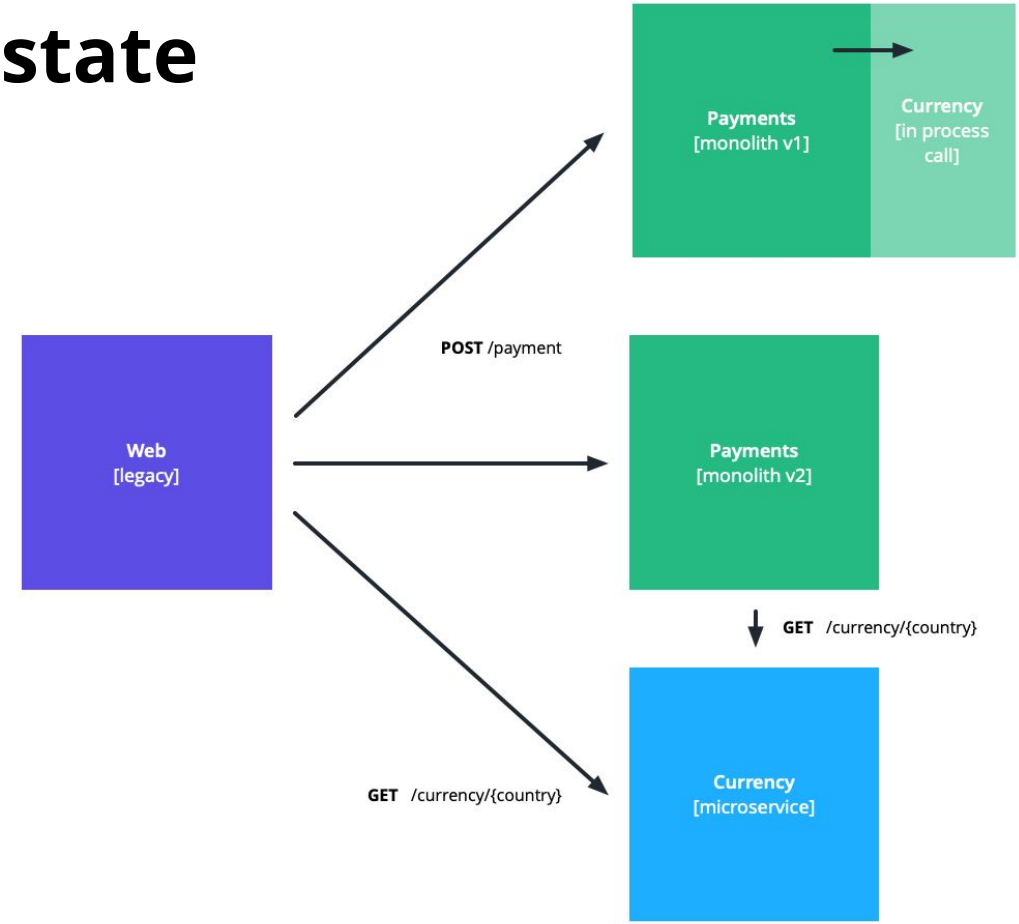
Demo

Splitting off the currency microservice

Current state

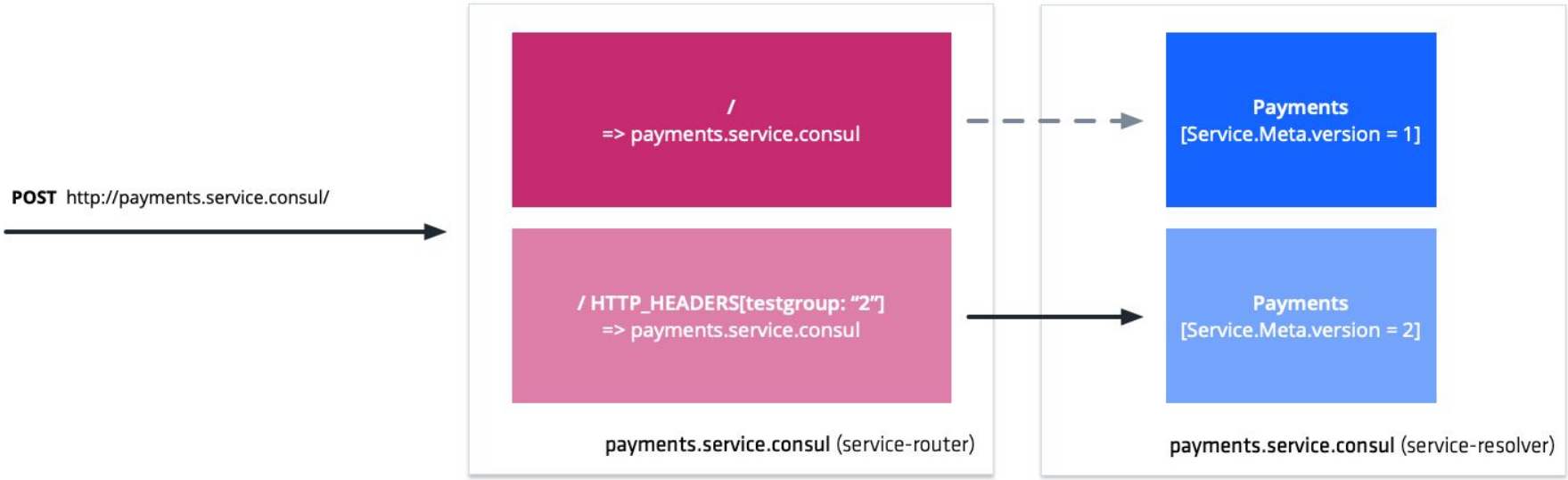


Desired state



Traffic Resolver

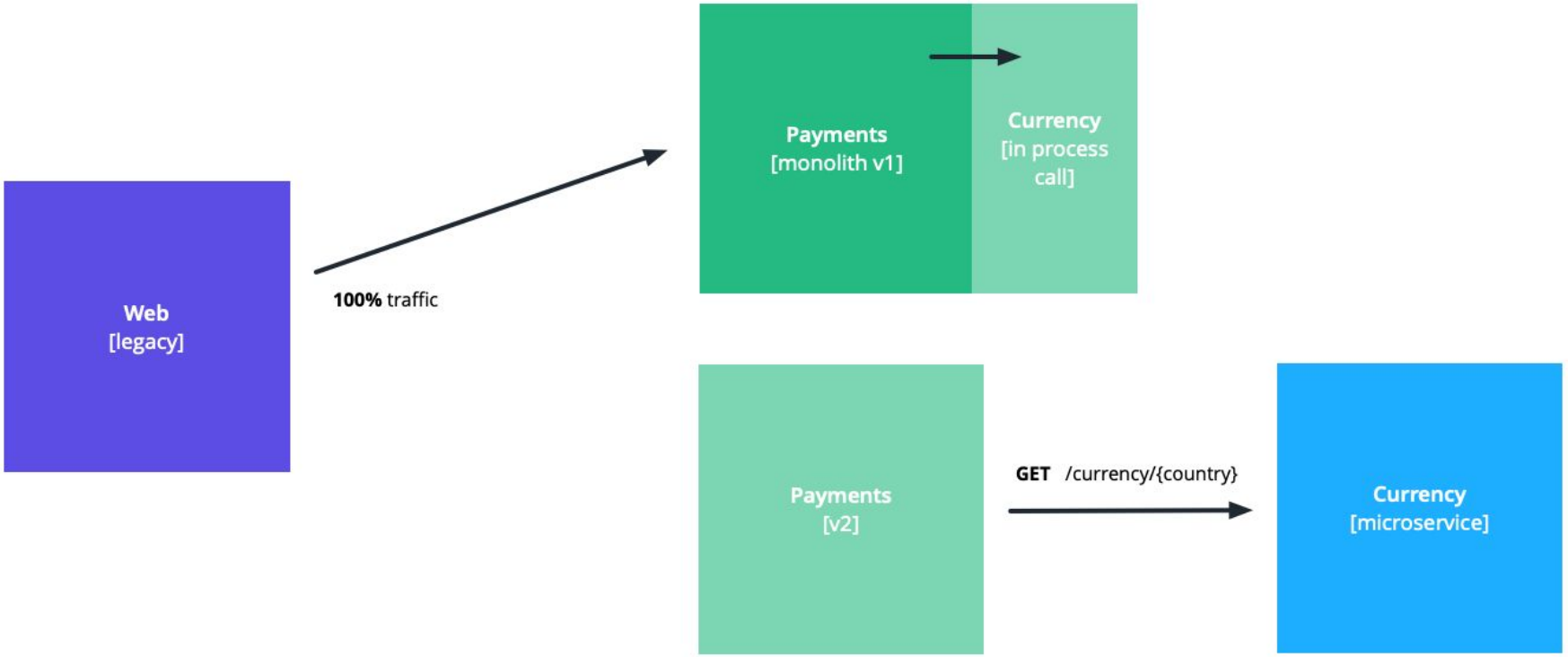
Service Router



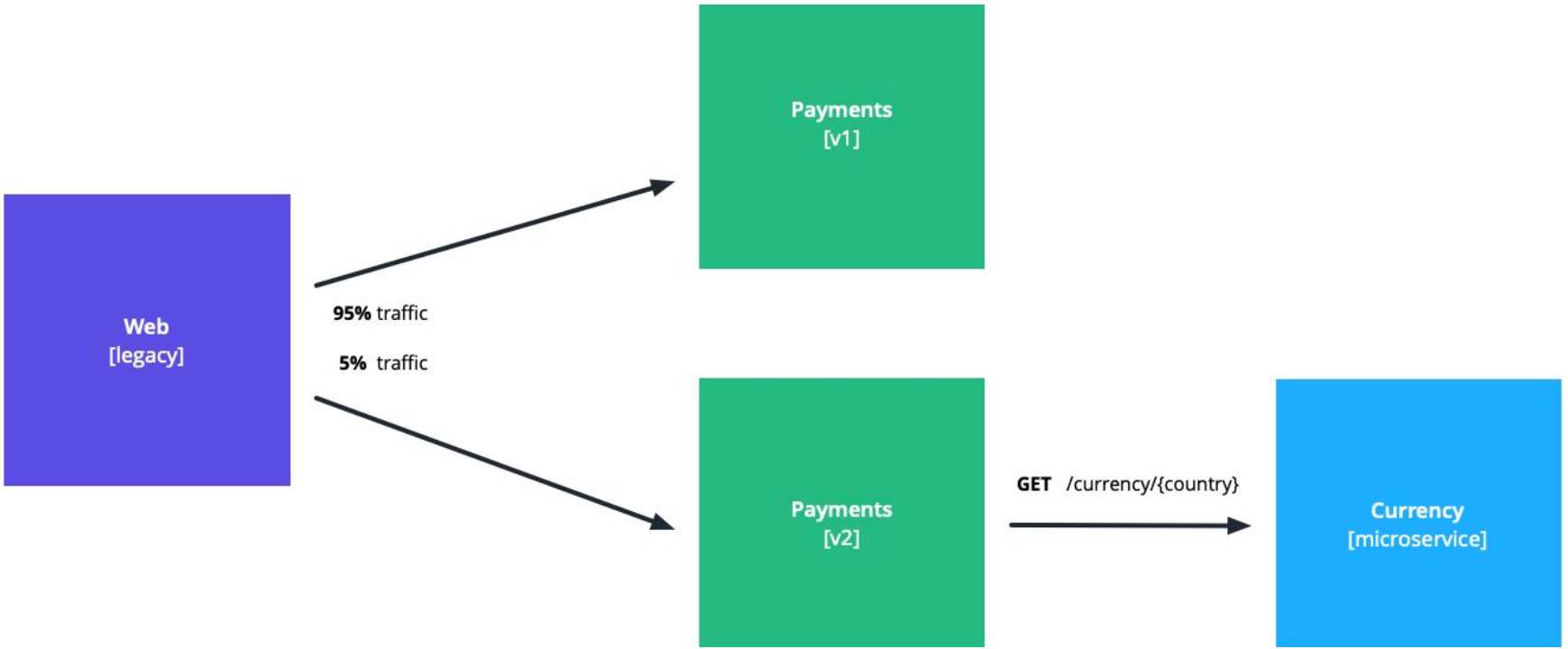
Demo

A/B testing v2 of the payments service

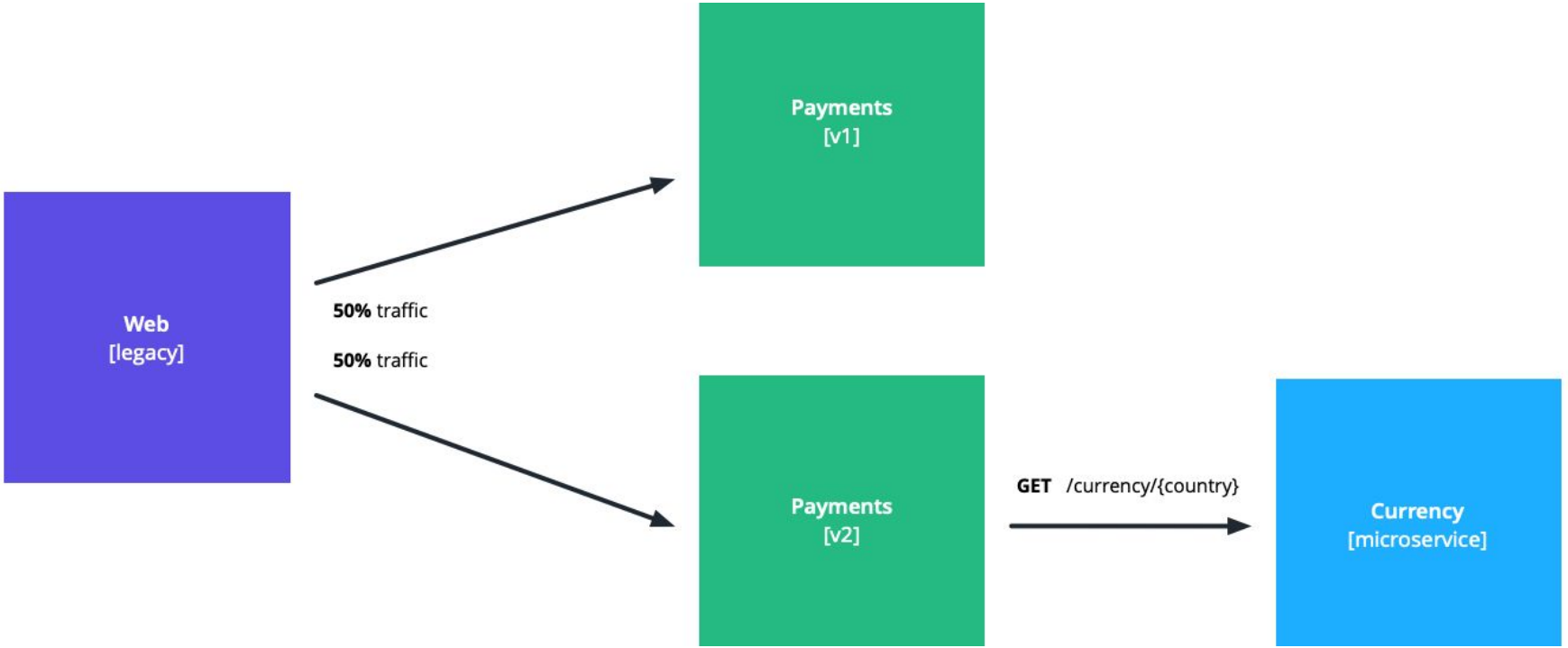
Current state



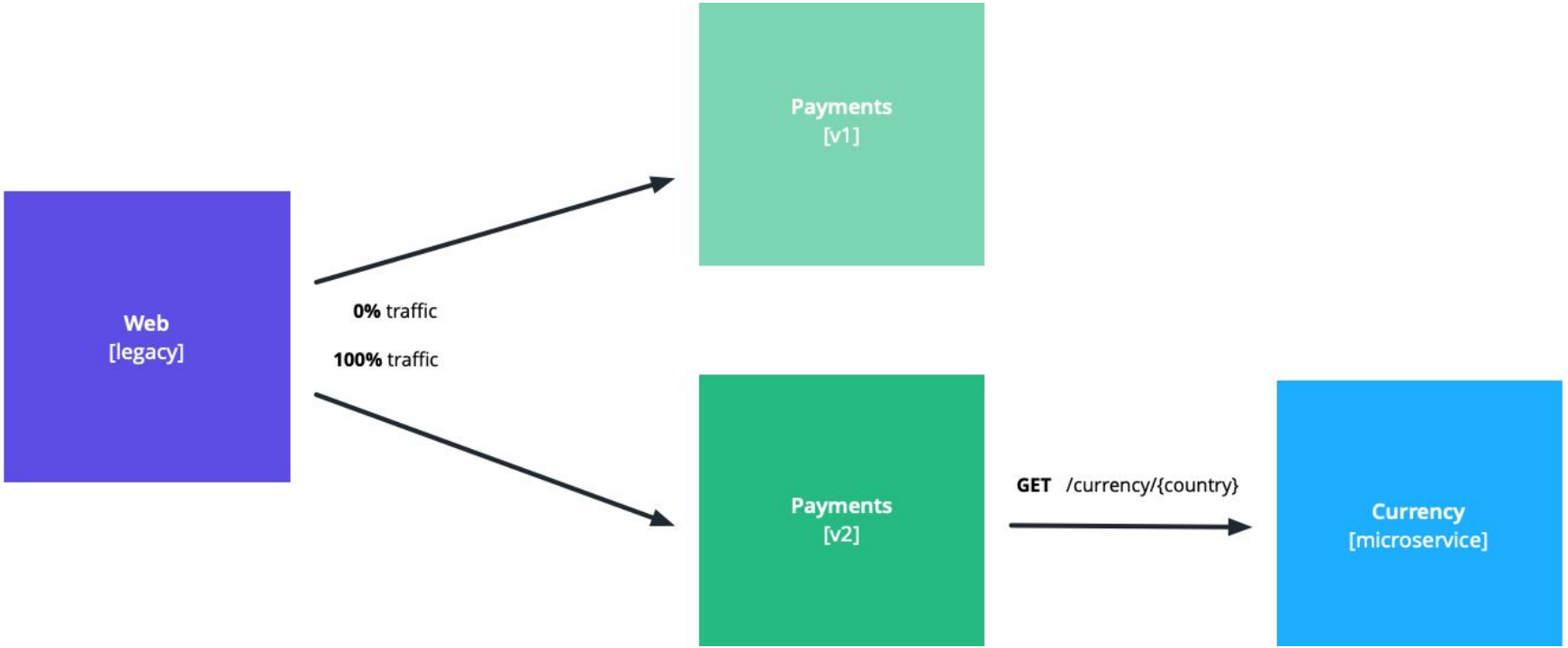
Desired state



Desired state

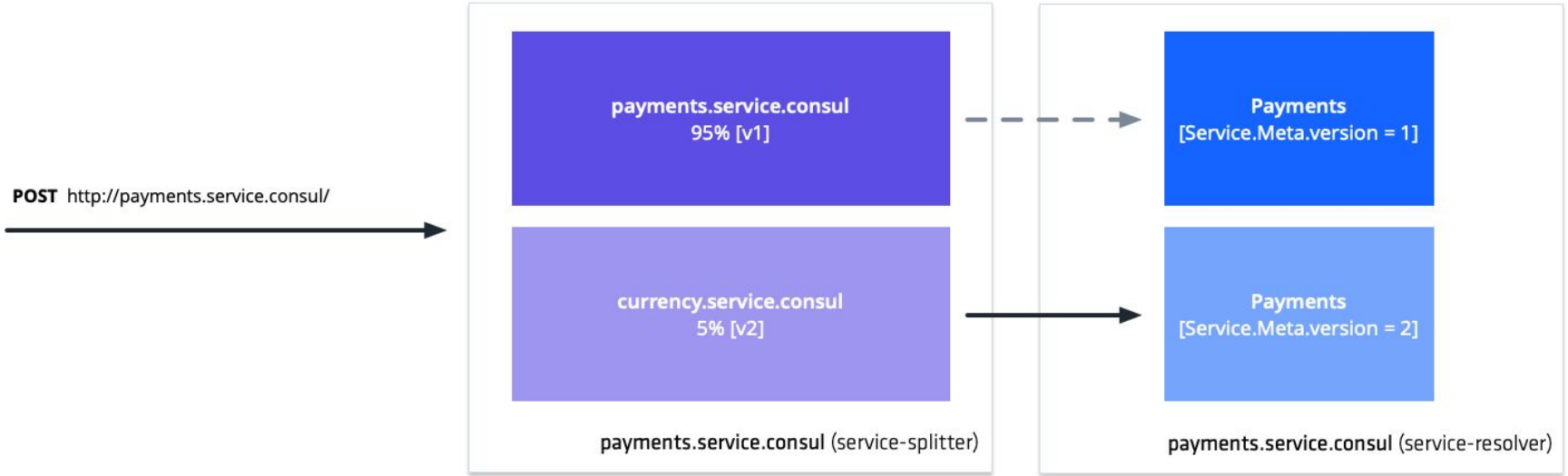


Desired state



Traffic Splitting

Service Splitter



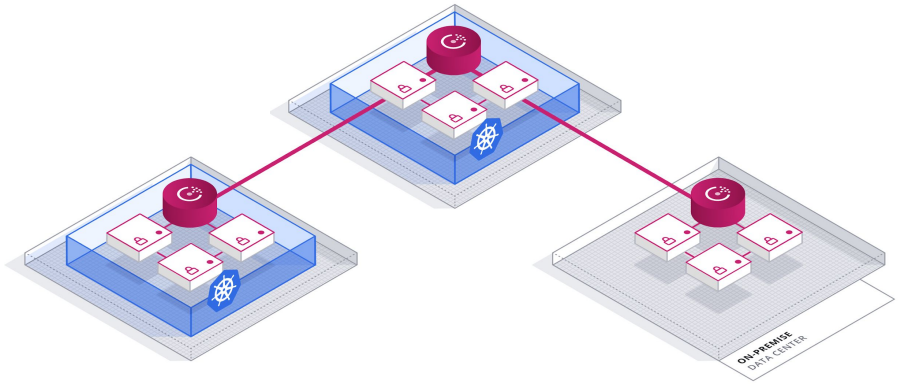
Demo

Canary release v2 of the payments service

Multi-Cluster Networking.

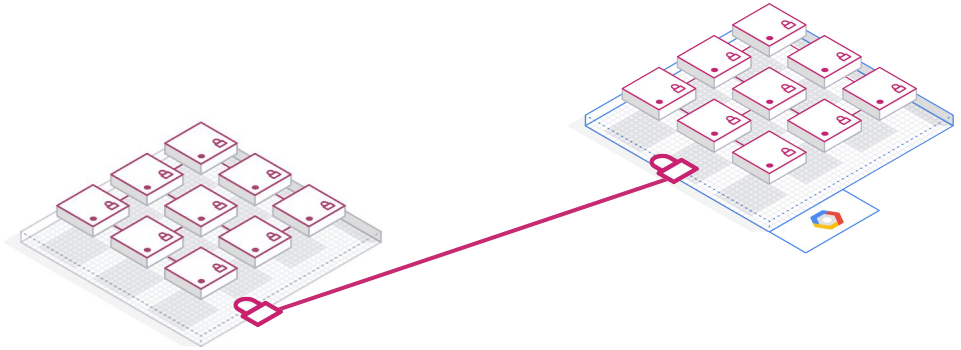
Multi-Cluster networking is hard

- Handle **overlapping IP ranges** in multiple clusters.
- Make routing work **without opening up** the cluster to all traffic.
- Integrating with **non Kubernetes** resources.



Multi-Cloud networking is harder

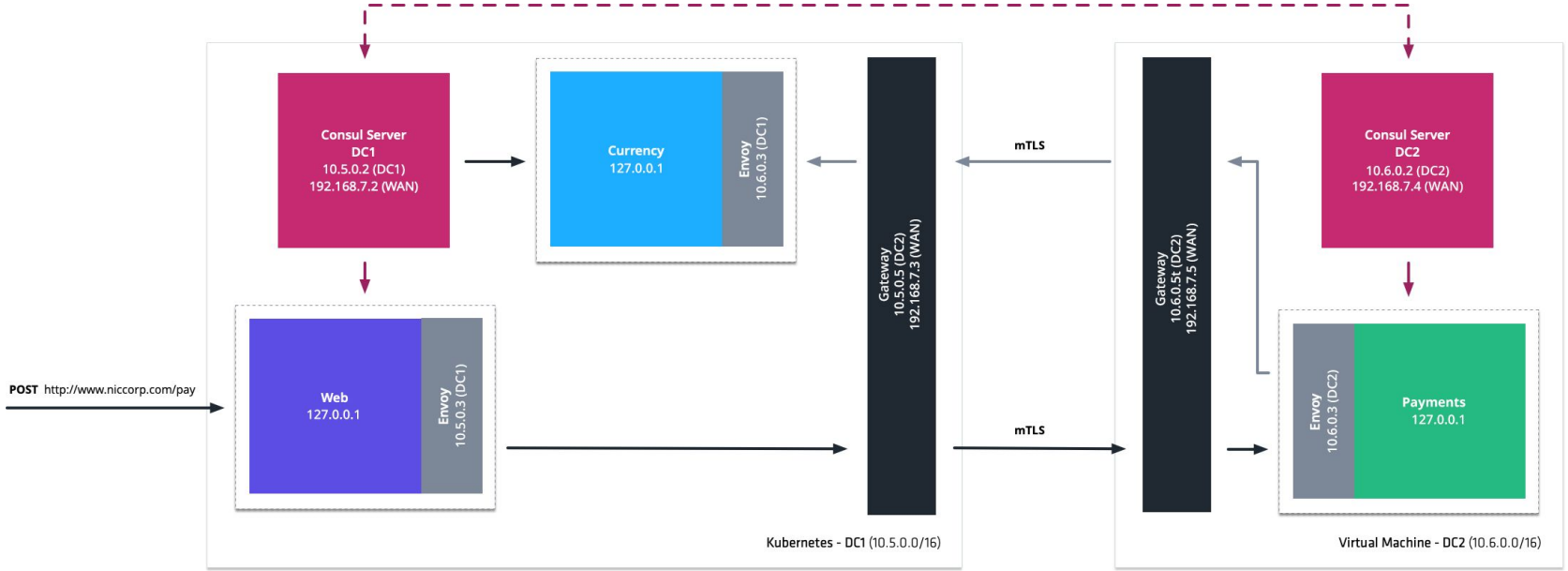
- Handle **overlapping IP ranges** in multiple clouds.
- Make routing work **without opening up** the cloud to all traffic.
- Integrating with **non cloud** resources.



Migrating from VMs to Kubernetes.

Multi-Cluster Gateways

Cross cluster / cross cloud service routing



Demo

Moving v2 of the currency service to Kubernetes

Demo

Pilot a new service on multiple Kubernetes clusters

Key takeaways

Adopting Multi-* does not have to be scary

- You don't have to do a big bang migration
- Service Mesh gives you simple and secure service segmentation
- Mesh Gateways provide effortless multi-cluster/cloud networking

Thank You

eveld@hashicorp.com
www.hashicorp.com

https://github.com/eveld/migrating-applications-with-service-mesh

Discovery Chain

Use L7 criteria such as **path prefixes** or **http headers**, and send traffic to a different service or service subset.

Split incoming requests across **different subsets** of a single service, or across **different services**.

Define which instances of a service should **satisfy discovery requests** for the provided name.

Why?

- **Quickly build and deliver** applications in response to customer needs.
- Automate operations to **eliminate the risk of failure** due to human error.
- **Architect for resilience** and focus on automated improvements to replace routine.
- Build applications that **run anywhere** without modification.

Why?

- Increase **resiliency** by leveraging schedulers and cloud services.
- Easily **scale** up and down by using containers or serverless functions.
- Choose the **best of breed** when selecting services.
- Create resources **on demand** and only **pay for** what is **used**.

Why?

- Try to **prevent** vendor **lock-in**.
- Choose the **best of breed** services from multiple cloud providers.
- Companies that operate on a **global scale**.
- **Failover** to another cloud in case of an outage.

Step 2

Migrating from monolith to microservices.

Step 3

Migrating from monolith to microservices.

Step 5

Multiple Kubernetes clusters.

Step 6

Multi-Cloud.

Demo

Connect everything up together