



HashiCorp
Nomad



Nomad 0.10

**Secure Service-to-Service Traffic with
Consul Connect Integration.**

Secure Service-to-Service Traffic with Consul Connect Integration.

Erik Veld

Developer Advocate at HashiCorp





Agenda.

Secure Service-to-Service Traffic with Consul Connect Integration.

1. New features in 0.10
2. Network namespaces
3. Consul Connect integration
4. Questions?



—

New features in Nomad 0.10.



—

Host Volumes.



Host Volumes.

Define a host volume in the Nomad client configuration.

/etc/nomad.d/client.hcl

```
host_volume "mysql" {  
    path = "/opt/mysql/data"  
    read_only = false  
}
```



Host Volumes.

Then use that volume in
your Nomad jobs.

jobs/mysql.hcl

```
job "mysql-server" {
    ...
    group "mysql-server" {
        ...
        volume "mysql" {
            type = "host"
            source = "mysql"
            read_only = false
        }
        task "mysql-server" {
            ...
            volume_mount {
                volume = "mysql"
                destination = "/var/lib/mysql"
                read_only = false
            }
        }
    }
}
```



—

Allocation File Explorer.



Allocation File Explorer.

Explore code and log files
from within the Nomad UI.

The screenshot shows the Nomad UI interface. At the top, there is a navigation bar with a search bar and links for "Documentation" and "ACL Tokens". Below the navigation bar, the main title is "Jobs / fs-example / fs-example / 5a21a6e9 / fs-example". On the left, there is a sidebar with sections for "WORKLOAD" (Jobs), "CLUSTER" (Clients, Servers), and a "Logs" section. The "Logs" section is currently active, indicated by a blue underline. The main content area displays the file "example.go" under the path "fs-example / local / code". The code content is as follows:

```
package raft

import (
    "fmt"
    "io"
    "time"

    "github.com/armon/go-metrics"
)

// FSM provides an interface that can be implemented by
// clients to make use of the replicated log.
type FSM interface {
    // Apply log is invoked once a log entry is committed.
    // It returns a value which will be made available in the
    // ApplyFuture returned by Raft.Apply method if that
    // method was called on the same Raft node as the FSM.
    Apply(*Log) interface{}

    // Snapshot is used to support log compaction. This call should
    // return an FMSnapshot which can be used to save a point-in-time
    // snapshot of the FSM. Apply and Snapshot are not called in multiple
    // threads, but Apply will be called concurrently with Persist. This means
    // the FSM should be implemented in a fashion that allows for concurrent
    // updates while a snapshot is happening.
    Snapshot() (FMSnapshot, error)

    // Restore is used to restore an FSM from a snapshot. It is not called
    // concurrently with any other command. The FSM must discard all previous
    // state.
    Restore(io.ReadCloser) error
}

// FMSnapshot is returned by an FSM in response to a Snapshot
// it must be safe to invoke FMSnapshot methods with concurrent
// calls to Apply.
type FMSnapshot interface {
    // Persist should dump all necessary state to the WriteCloser 'sink',
    // and call sink.Close() when finished or call sink.Cancel() on error.
    Persist(sink SnapshotSink) error

    // Release is invoked when we are finished with the snapshot.
    Release()
}

// runFSM is a long running goroutine responsible for applying logs
runFSM()
```

At the bottom right of the code editor, there is a "View Raw File" button.



Allocation File Explorer.

Browse rich media files from
within the Nomad UI.

The screenshot shows the Nomad UI interface. At the top, there's a navigation bar with a refresh icon, 'Documentation' and 'ACL Tokens' links. Below it, a green header bar displays the path: 'Jobs / fs-example / fs-example / 5a21a6e9 / fs-example'. On the left, a sidebar has sections for 'WORKLOAD' (Jobs), 'CLUSTER' (Clients, Servers), and a search bar. The main content area has tabs for 'Overview', 'Logs', and 'Files', with 'Files' being the active tab. Under 'Files', the path 'fs-example / local / images / docker.webp' is shown. A large image of the Docker logo (a blue whale carrying shipping containers) is displayed. Below the image, the file name 'docker.webp' and its dimensions ('792px x 613px, 12 KiB') are visible. In the bottom right corner of the slide, there's a small 'View Raw File' link.



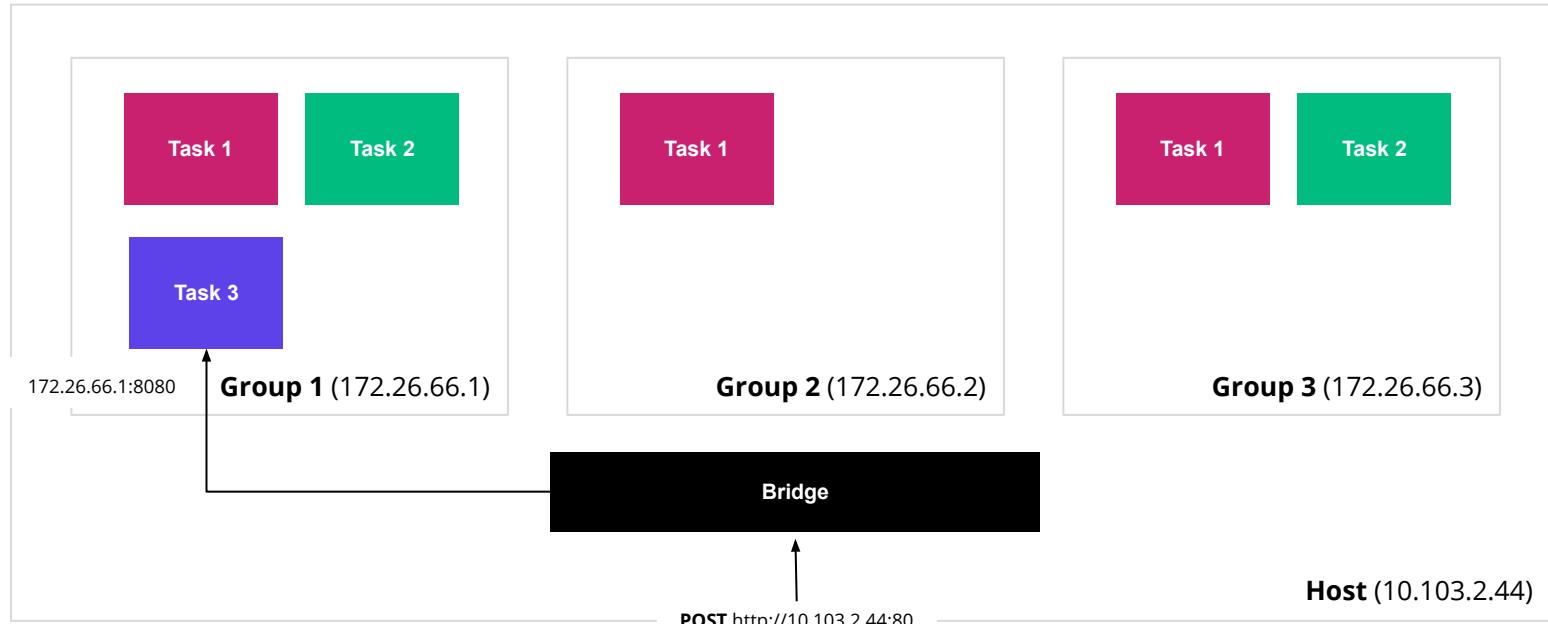
—

Network namespaces.



Network namespaces.

Shared networking among tasks of an allocation.





Network Modes.

bridge

- Shared network stack
- Virtual interface to host bridge
- Egress NAT rules
- Ingress port mapping

host

- Host network stack
- No port mapping

none

- Shared network stack
- No egress/ingress
- Loopback only



Network Stanza.

Specify the network stanza at the group level and set the mode to bridge.

```
job "api" {  
  ...  
  group "api" {  
    ...  
    network {  
      mode = "bridge"  
    }  
  }  
}
```

jobs/api.hcl



Network Stanza.

Ports can be exposed on the host machine by specifying a static local port and a destination inside the namespace.

```
job "api" {
  ...
  group "api" {
    ...
    network {
      mode = "bridge"

      port "http" {
        static = "9090"
        to = "9090"
      }
    }
  }
}
```

jobs/api.hcl



Network Stanza.

Each task will join the host network namespace and a shared network namespace is not created.

```
job "api" {
  ...
  group "api" {
    ...
    network {
      mode = "host"
    }
  }
}
```

jobs/api.hcl



Network Stanza.

The network stack can be shared across multiple drivers that support it.

```
jobs/api.hcl
```

```
job "api" {
  ...
  group "api" {
    ...
    network { ... }

    task "api" {
      driver = "exec"
      ...
    }

    task "worker" {
      driver = "docker"
      ...
    }
  }
}
```



Demo.



—

Consul Connect integration.



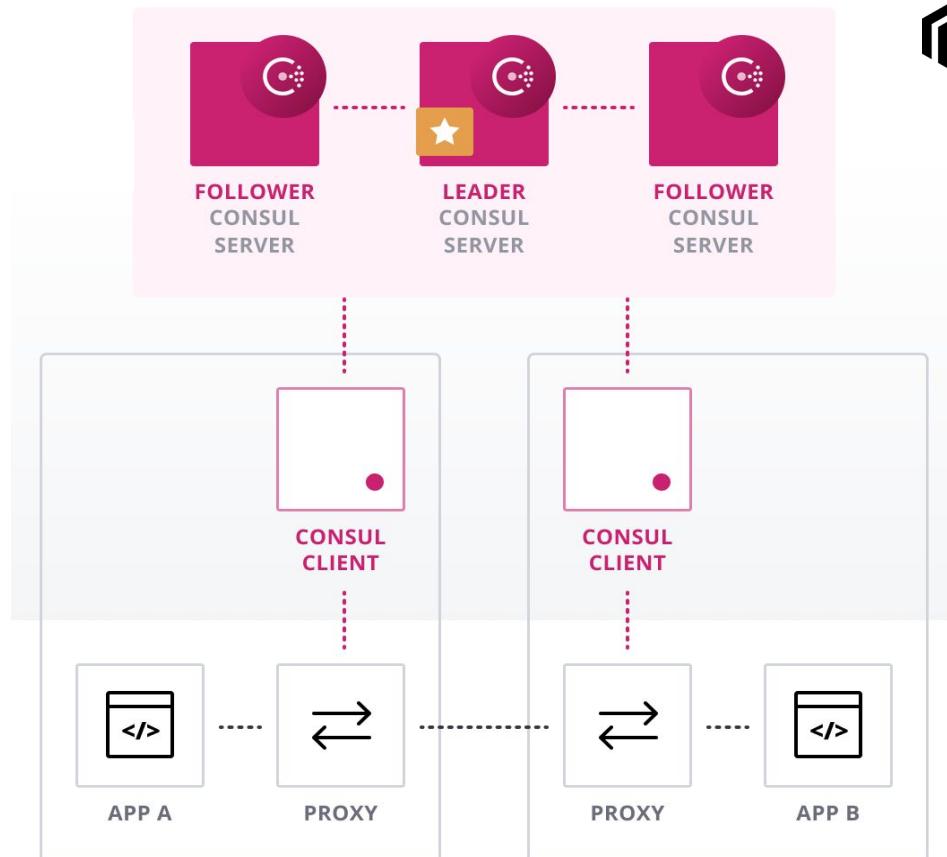
—

Connect.

Service Mesh

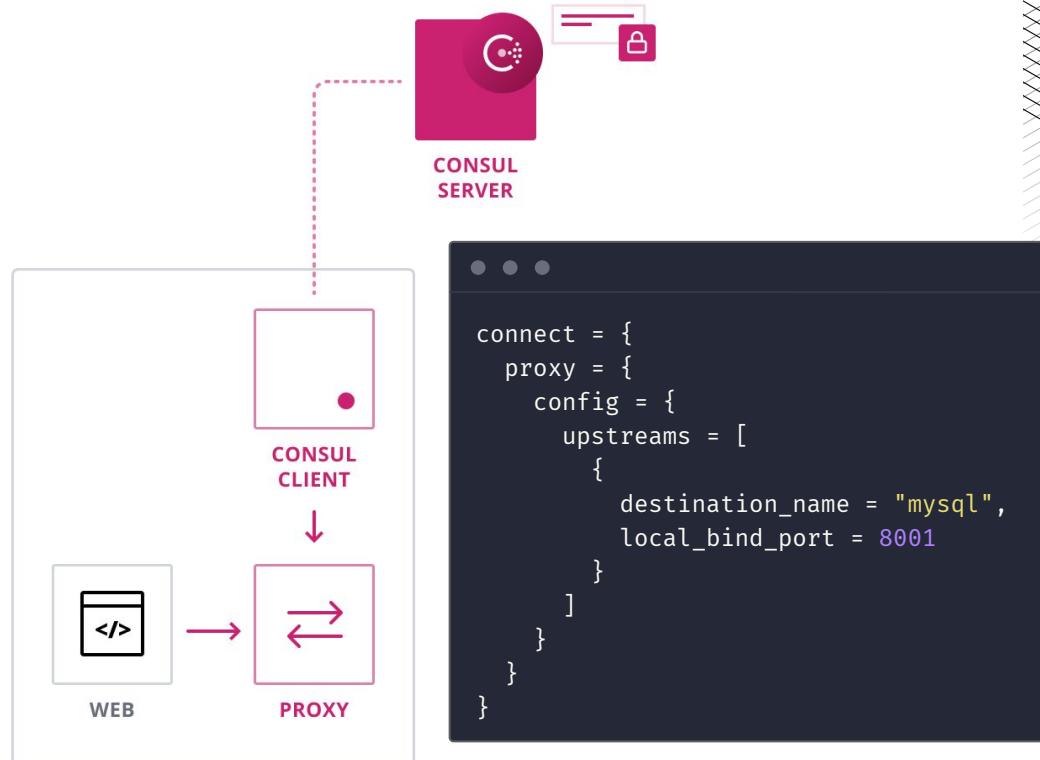
The **Control Plane** is responsible for Service Discovery, Authorization and configuring the proxies of the Data Plane.

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



Configuration

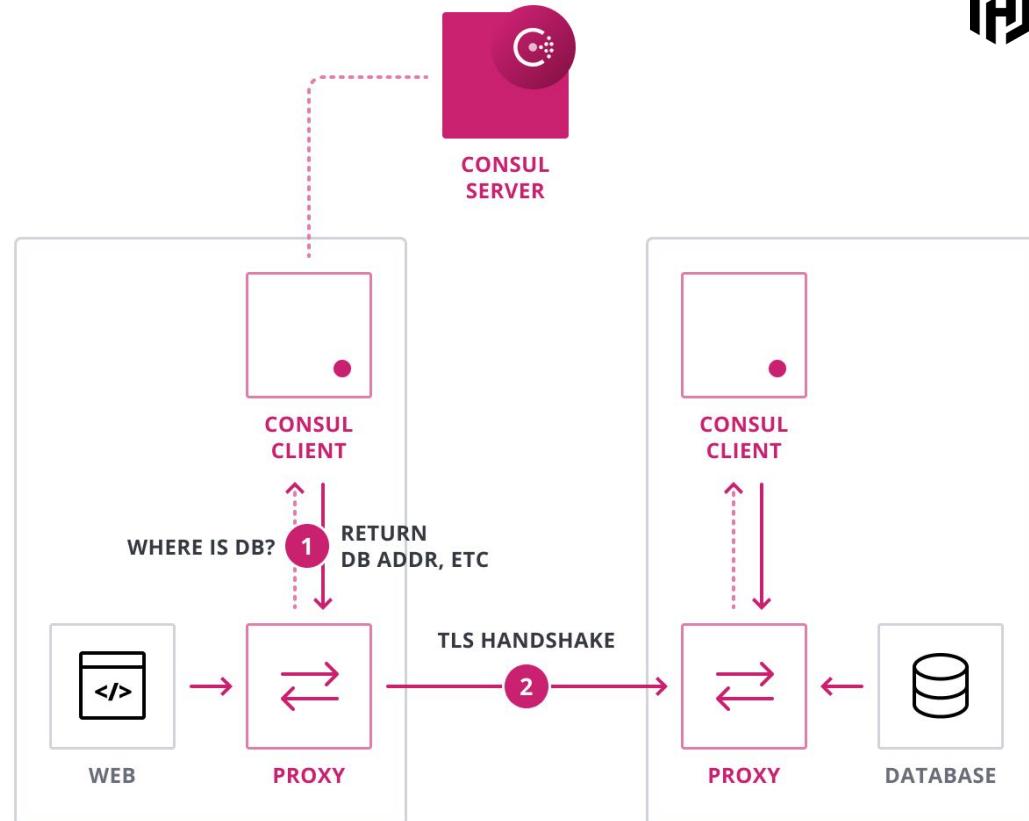
- A proxy is **co-located** with the service instance that proxies inbound traffic.
- The Client agent **instantiates** the proxy and **registers** it as a service.
- The proxy is **configured** with a port that is used for the service and ports for any upstream destination that the service wants to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





—

Service based security.

Intentions

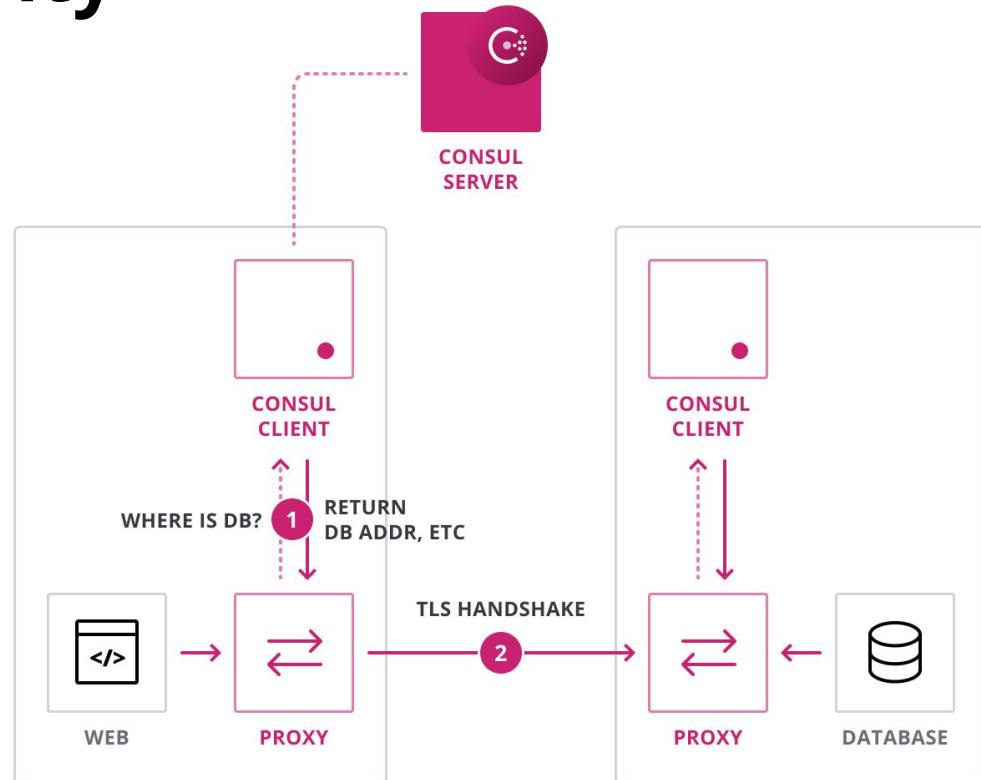
The service access graph defines which services are **allowed or denied from communicating through intentions.**



```
$ consul intention create -deny '*' '*'  
Created: * => * (deny)  
  
$ consul intention create -allow web app  
Created: web => app (allow)  
  
$ consul intention create -allow web db  
Created: web => db (allow)
```

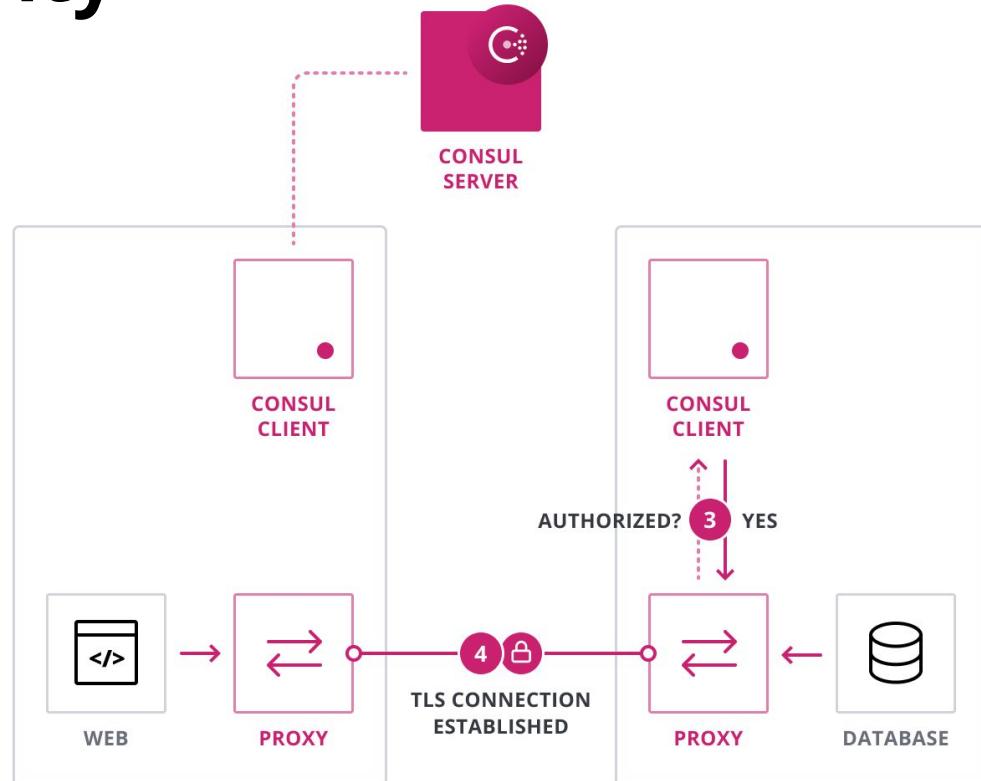
Service based security

- 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.





—

Service Stanza.



Service Stanza.

The service stanza can now be defined on a task group.

Use the **connect** and **sidecar_service** stanza to inject a sidecar proxy into your task group.

```
● ● ● jobs/api.hcl

job "api" {
  ...
  group "api" {
    ...
    service {
      name = "api"
      port = 9090

      connect {
        sidecar_service {}
      }
    }
  }
}
```



Service Stanza.

Upstreams can be defined on the sidecar service to allow communication between connect enabled services.

```
jobs/api.hcl
```

```
job "api" {
  ...
  group "api" {
    ...
    service {
      name = "api"
      port = 9090

      connect {
        sidecar_service {
          proxy {
            upstreams {
              destination_name = "payments"
              local_bind_port = 9091
            }
          }
        }
      }
    }
  }
}
```



Service Stanza.

Extra **proxy configuration** can be defined that's opaque to Nomad and is passed directly to Consul.

```
jobs/api.hcl
```

```
job "api" {
  ...
  group "api" {
    ...
    service {
      name = "api"
      port = 9090

      connect {
        sidecar_service {
          proxy {
            upstreams {
              destination_name = "payments"
              local_bind_port = 9091
            }
          }
        }
      }
    }
  }
}
```



—

**Migrate from monolith
to microservices.**

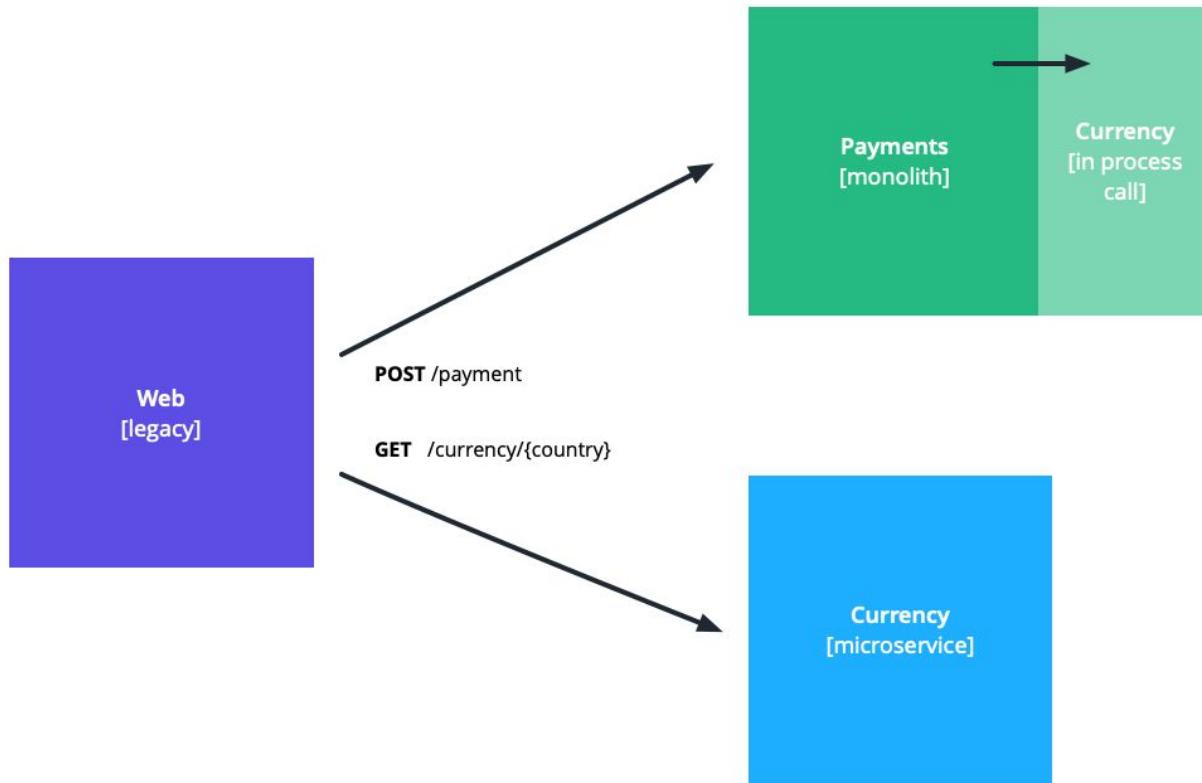


Current state



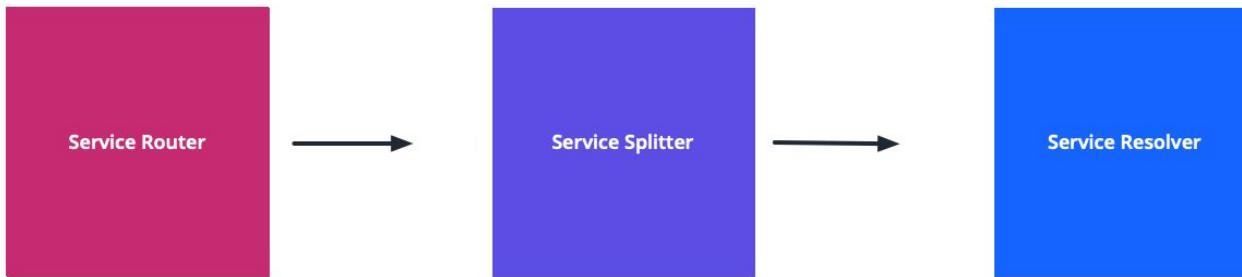


Desired state





Discovery Chain



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

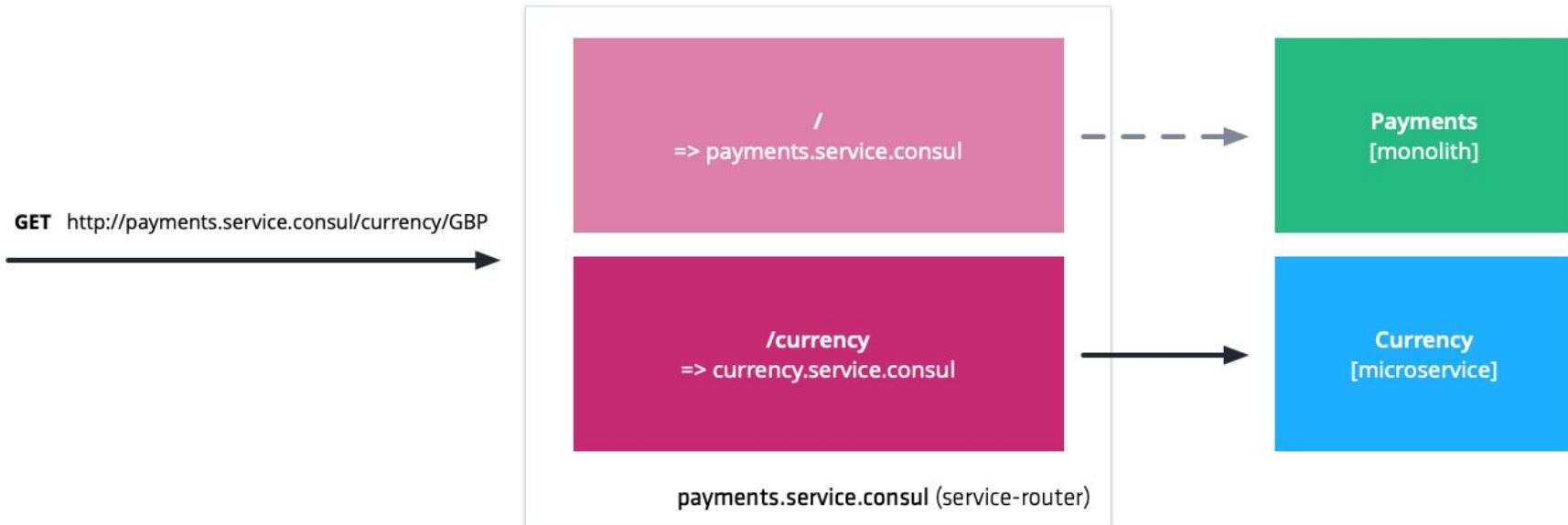
Split incoming requests across **different subsets** of a single service, or across **different services**.

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



Traffic Routing

Service Router





—

Demo.

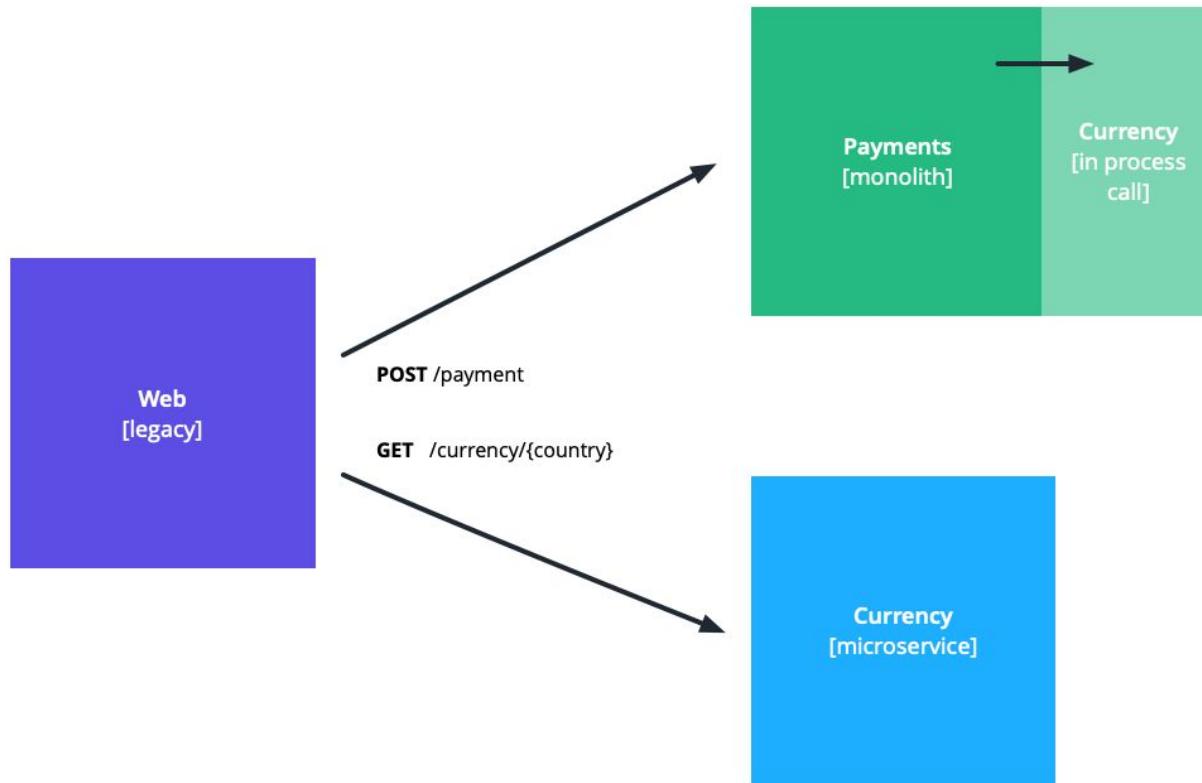


-

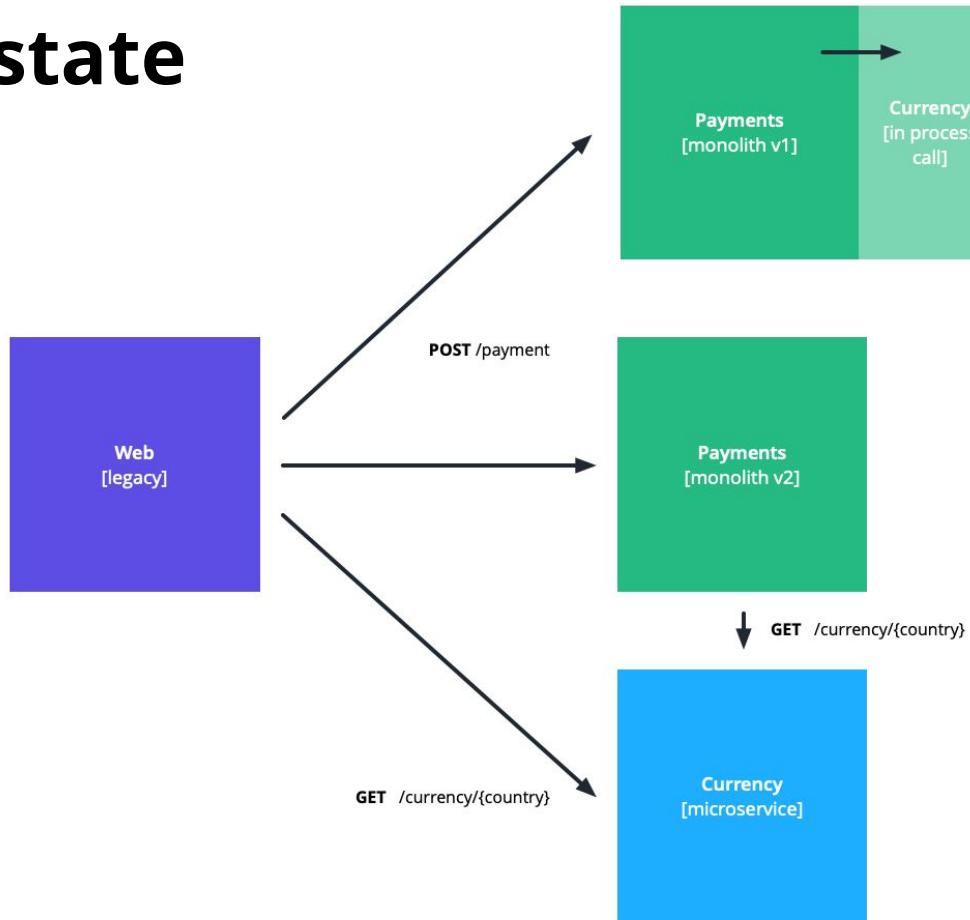
A/B testing.



Current state



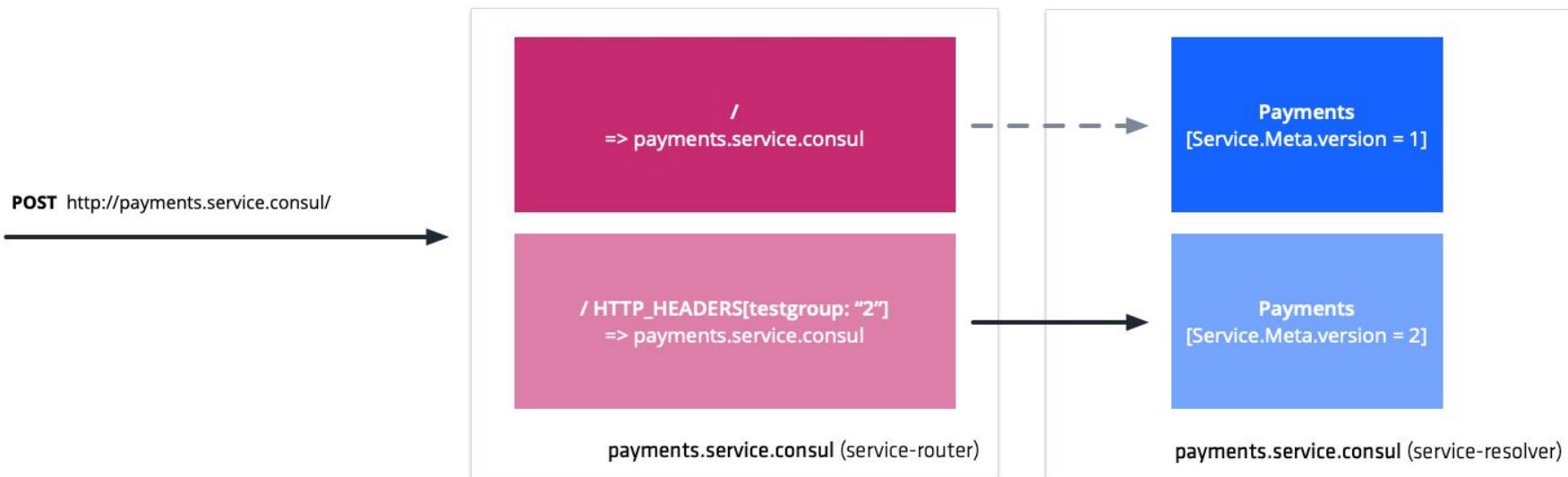
Desired state





Traffic Resolver

Service Router





—

Demo.

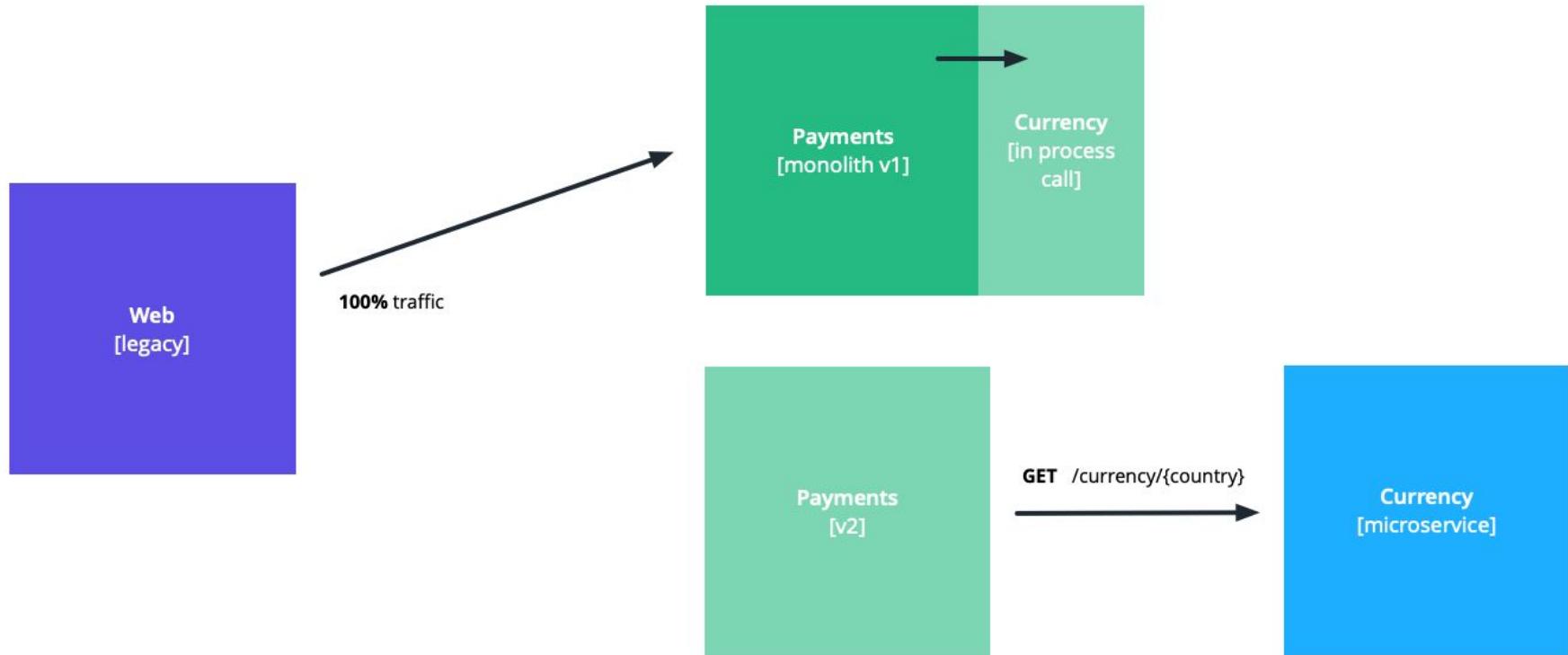


-

Canary release.

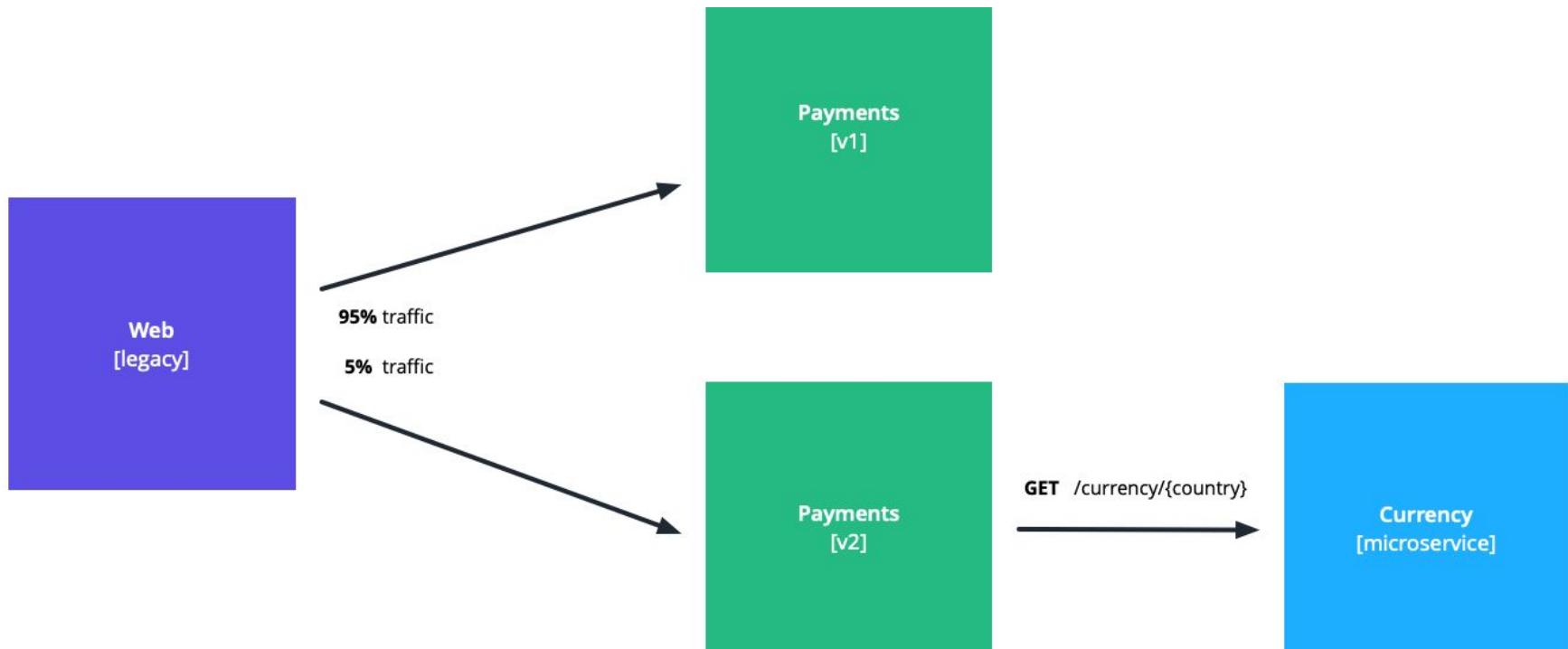


Current state



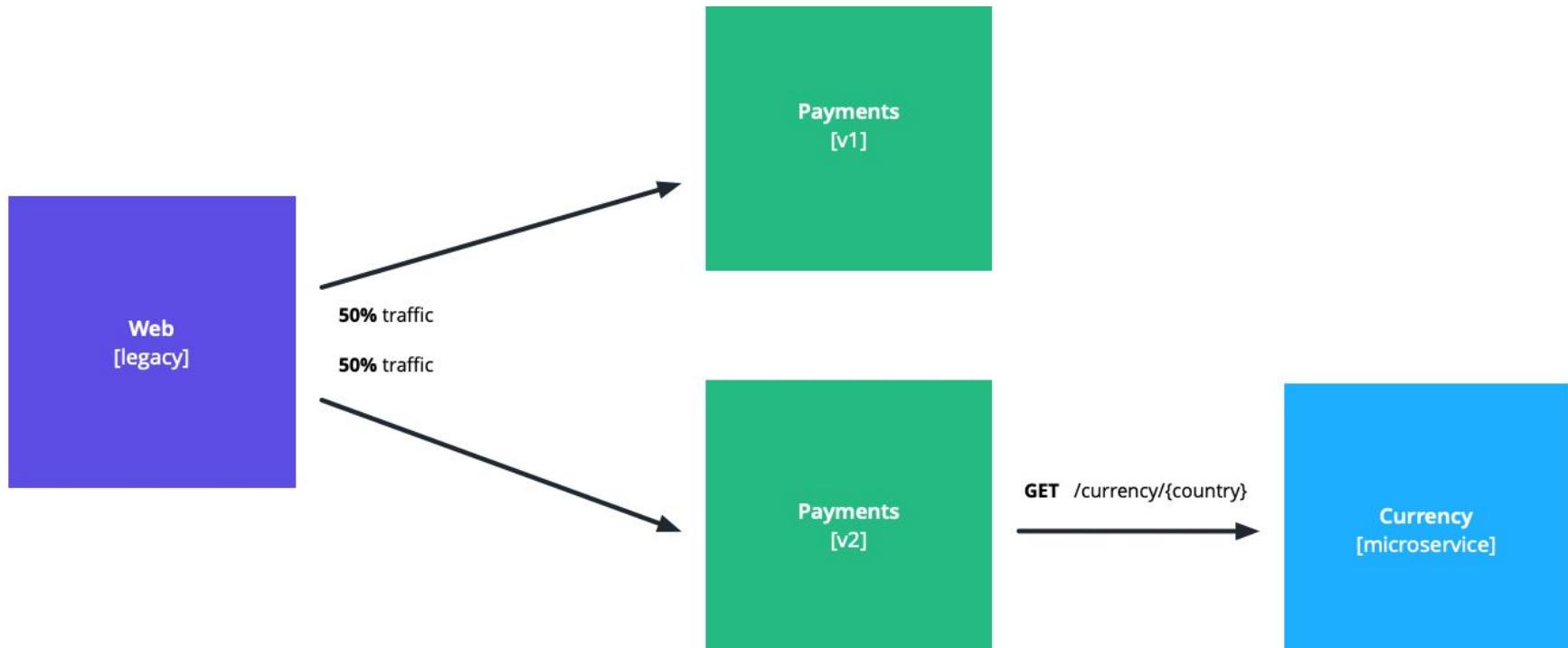


Desired state



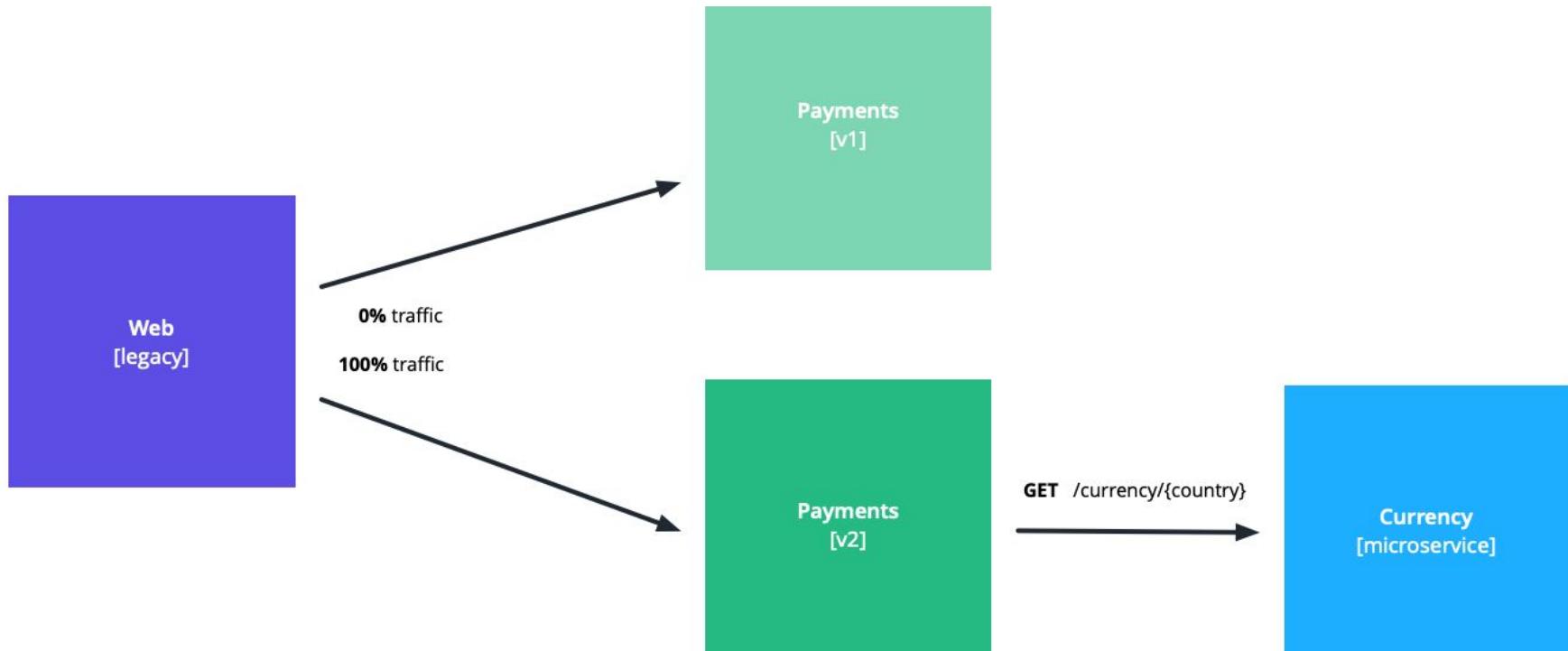


Desired state





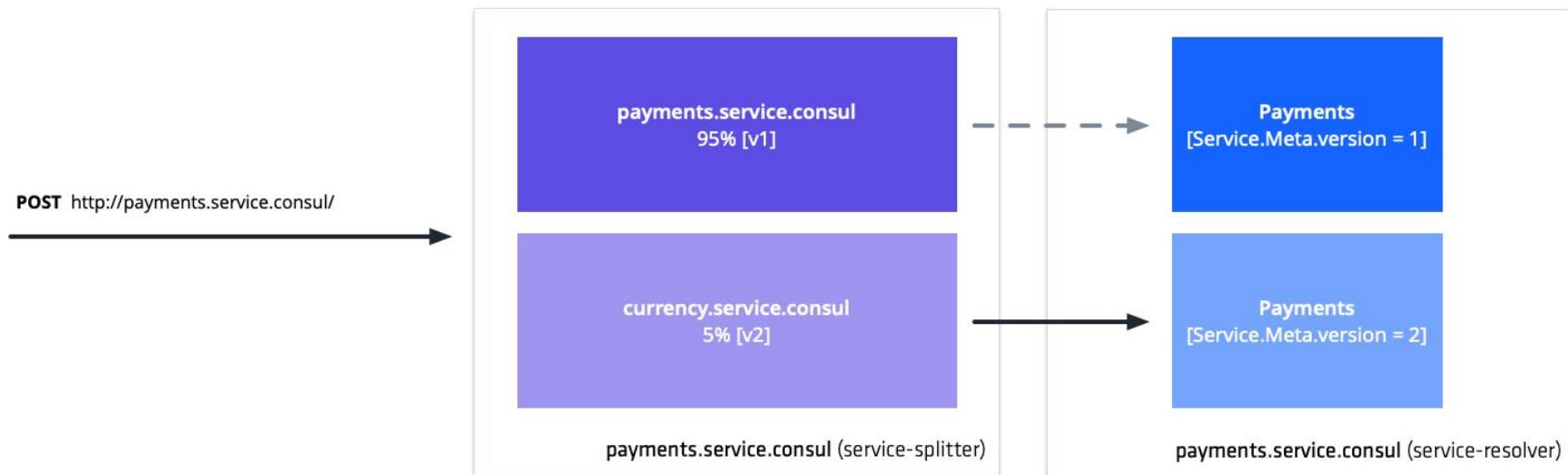
Desired state





Traffic Splitting

Service Splitter





—

Demo.



Thank You

eveld@hashicorp.com
www.hashicorp.com



—

Questions?