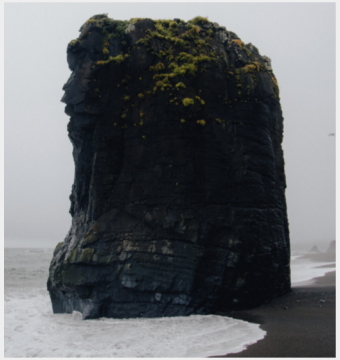
Hashi Service Mesh

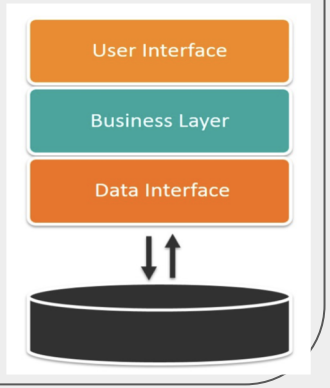
Using Service Mesh to
build communication
between containers

Vlad Silverman
Insight DevOps Fellow

Motivation

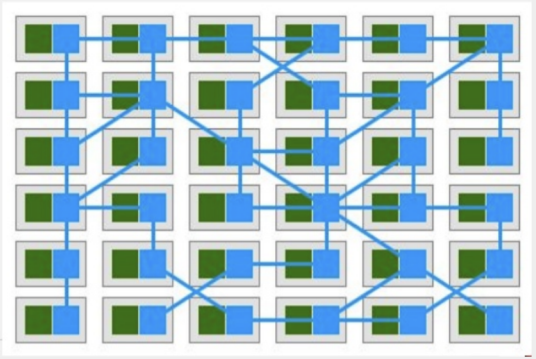
Monolithic Architecture





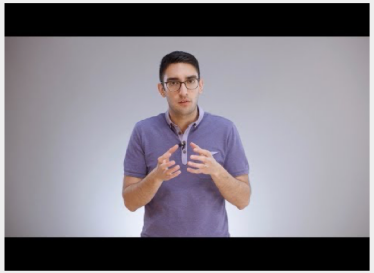
Trend

Microservices Architecture



Problems

Continuous Deployment
Upgrade to new technologies
Reliability



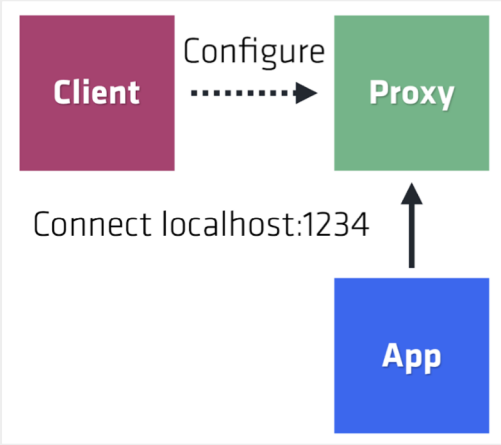
HashiCorp

Consul CTO
Armon Dadgar

Project Goal

Create a DevOps pipeline that will:

- Use principles of service mesh architecture
- Enable microservices communication
- Put logic of communications into proxies

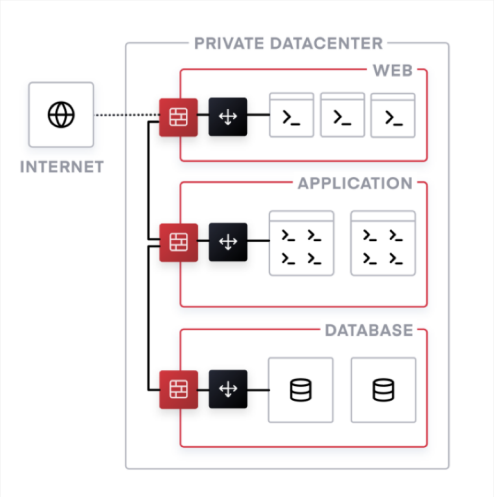


Service Mesh with Consul

Advantage: Functionality is shifted from network middleware to the endpoints

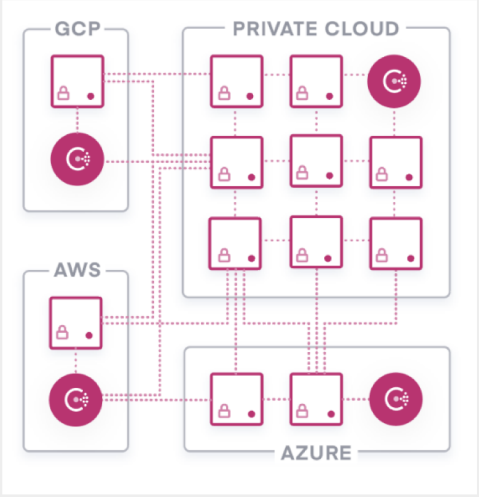
Result: Network is simplified

Before



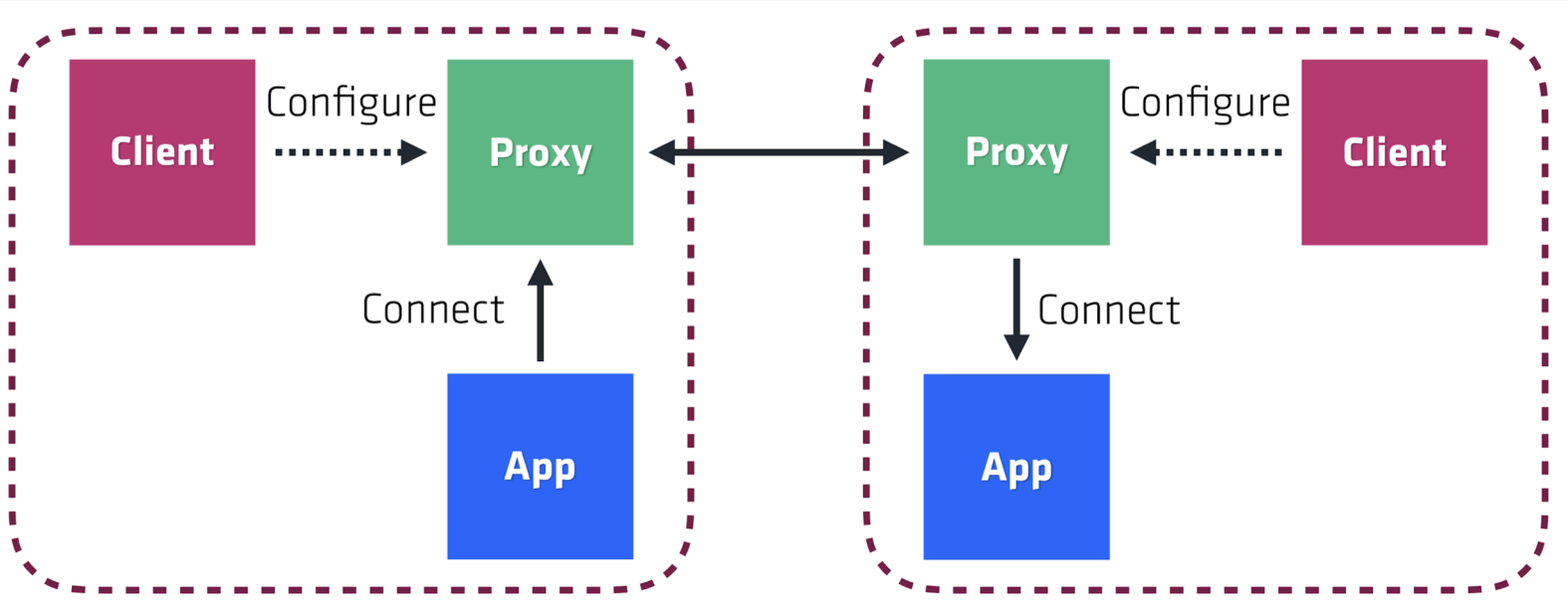
Transition

After

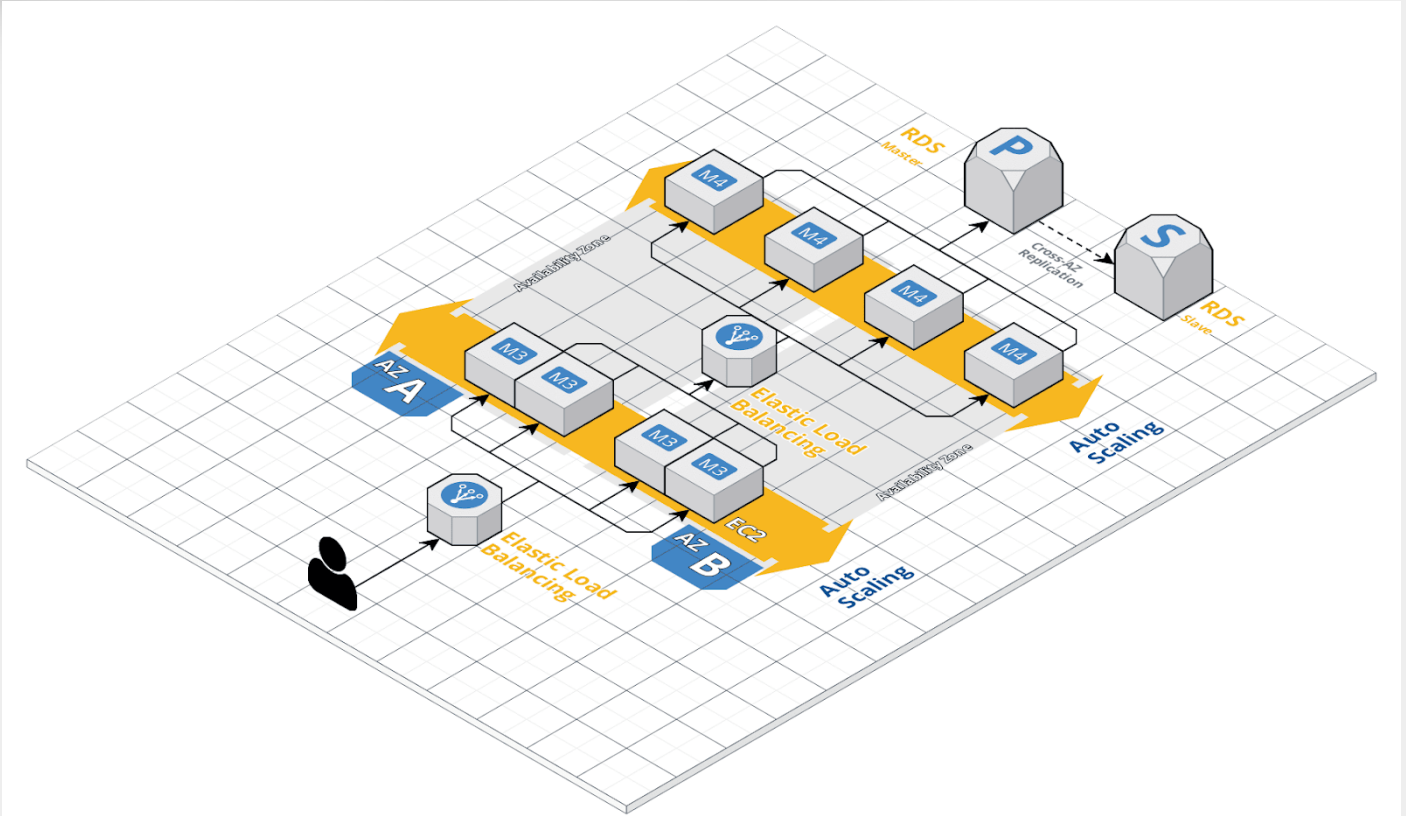


How it works

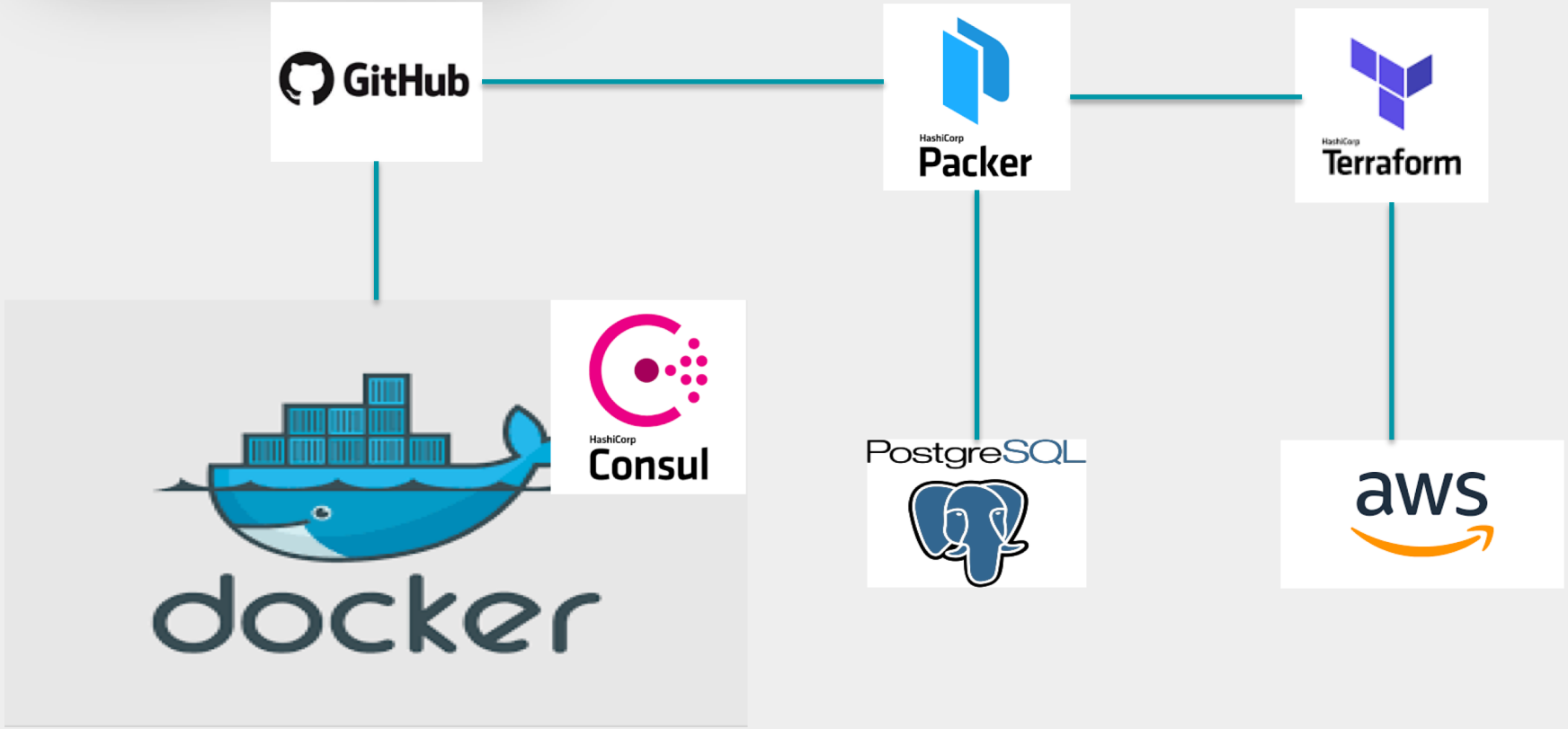
- Service code unchanged
- Service configuration modified



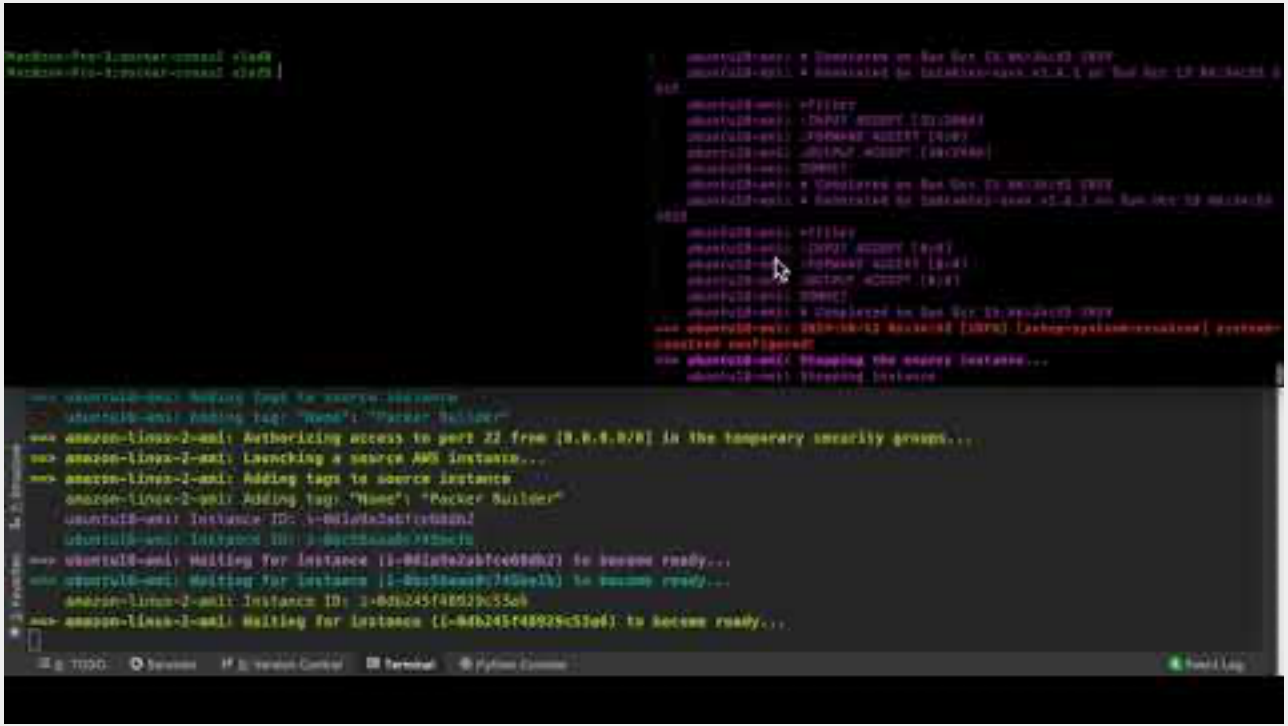
Project infrastructure



Tech Stack


GitHub
HashiCorp
Packer
HashiCorp
Terraform
HashiCorp
Consul
docker
PostgreSQL
aws

Demo


==> amazon-linux-2-ami: Authorizing access to port 22 from [0.0.0.0/0] in the temporary security groups...
==> amazon-linux-2-ami: Launching a source AWS instance...
==> amazon-linux-2-ami: Adding tags to source instance
amazon-linux-2-ami: Adding tag: "Name": "Packer Builder"
==> ubuntu18-ami: Waiting for instance (i-06b245f40929c51a6) to become ready...

Vlad Silverman | DevOps Fellow

https://github.com/vsilverman

Backups

Engineering Challenge

Network appliances (load balancers or firewalls) don't scale in dynamic settings.

- **Increased cost** from expensive network appliances and overheads for maintenance
- **Increased complexity** from maintaining topologies that constrain traffic through centralized middleware

Solution

Service mesh as automated and distributed approach to networking and security

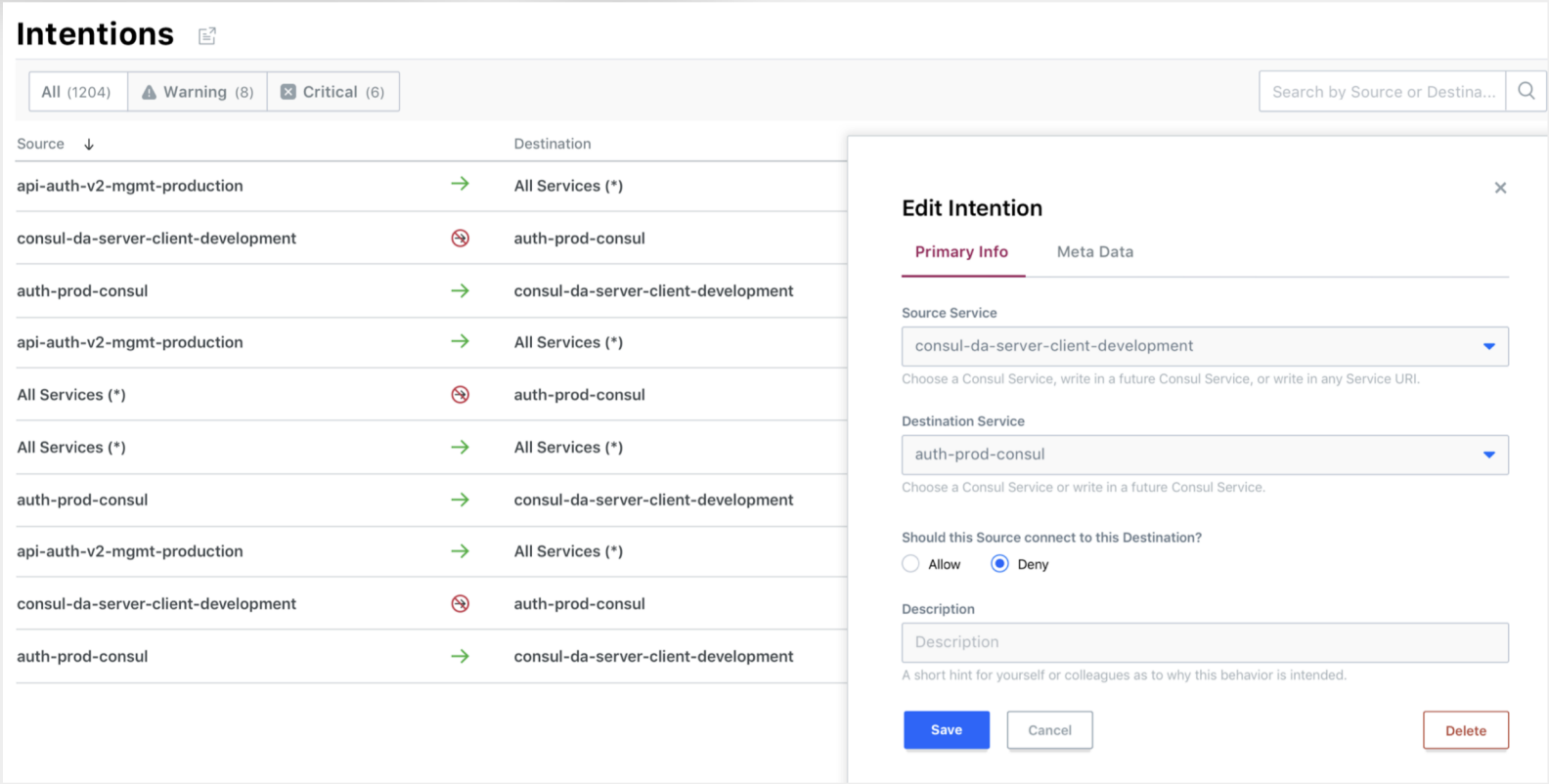
- **Reduce cost** by avoiding a proliferation of network appliances
- **Simplify Networks** by pushing authorization and traffic management to the endpoints, avoiding complex topologies

Intentions - CLI

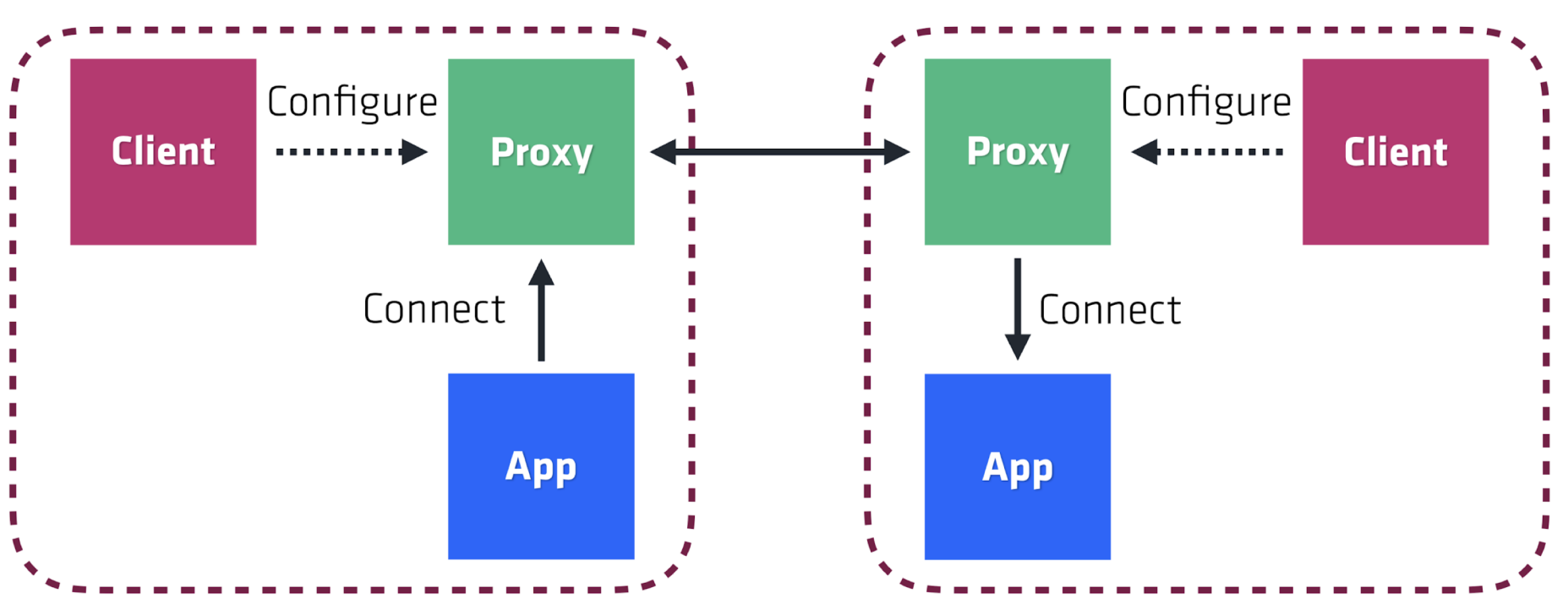
```
$ consul intention create -deny web '*'
Created: web => * (deny)

$ consul intention create -allow web db
Created: web => db (allow)
```

Intentions - GUI



Sidecar proxies



Sidecar proxies

```
{
  "service": "web",
  "connect": {
    "proxy": {
      "config": {
        "upstreams": [{
            "destination_name": "redis",
            "local_bind_port": 1234
        }]
      }
    }
  }
}
```

Connecting to DB through proxy

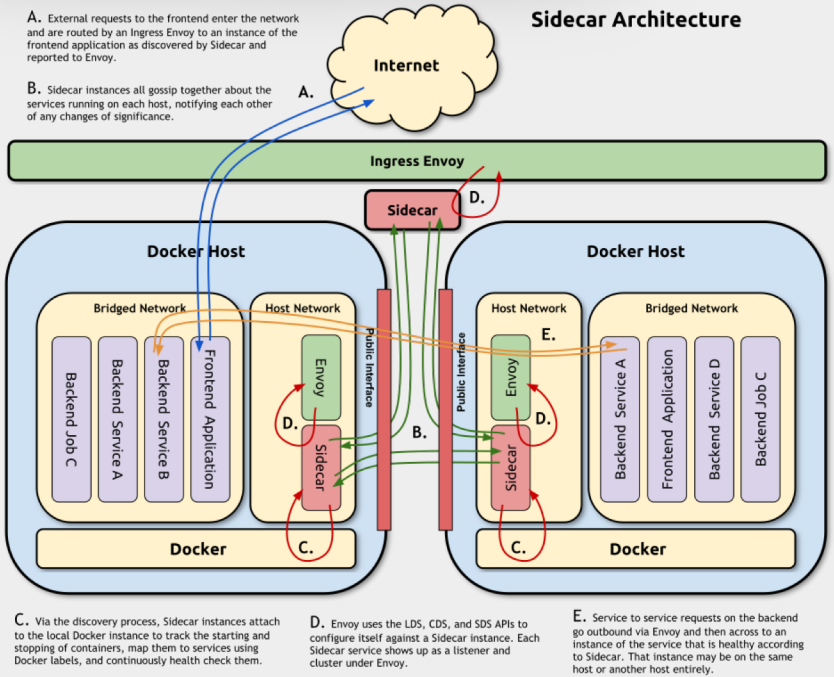
```
$ consul connect proxy \
  -service web \
  -upstream postgresql:8181

$ psql -h 127.0.0.1 -p 8181 -U mitchellh mydb
>
```

Keep alive

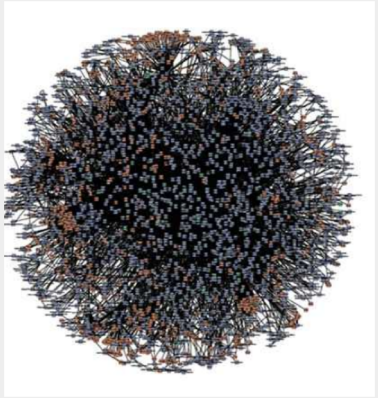
```
$ while true;do curl -i http://localhost:5050/service/counting/;sleep 0.5;done
```

Sidecar Architecture



Problem

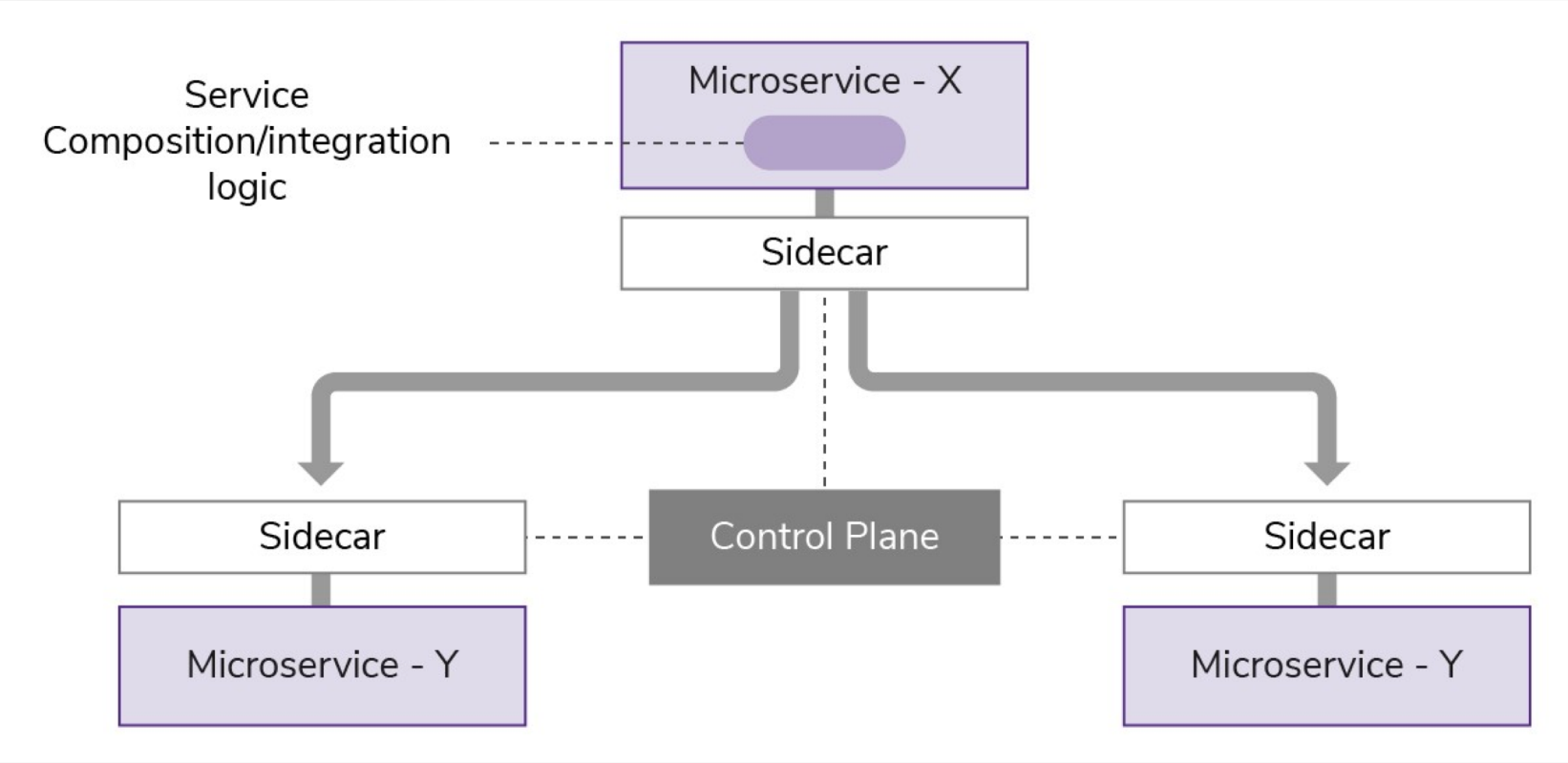
- How do we organize services discovery?
- How do we manage rules for microservices communication?



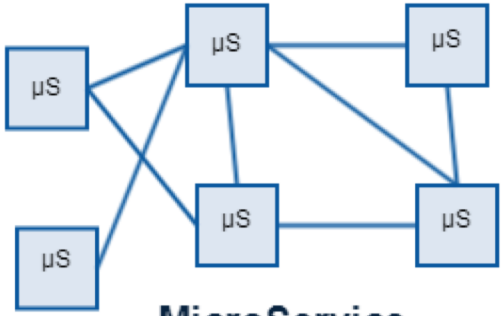
Solution

- Use service-mesh architecture
- Create registry of running services
- Discovery of available services
- Maintain rules of communication

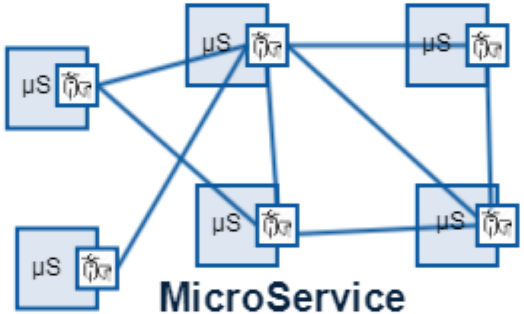
Sidecars


Service
Composition/integration
logic
Microservice - X
Sidecar
Sidecar
Control Plane
Sidecar
Microservice - Y
Microservice - Y

Sidecars

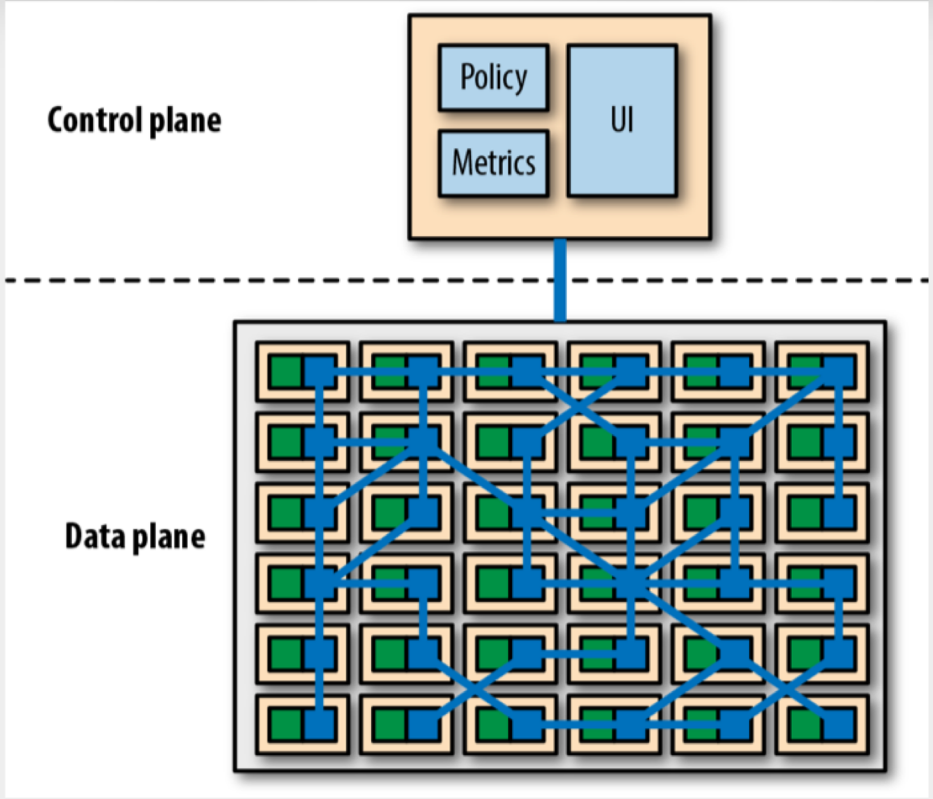


MicroService architecture

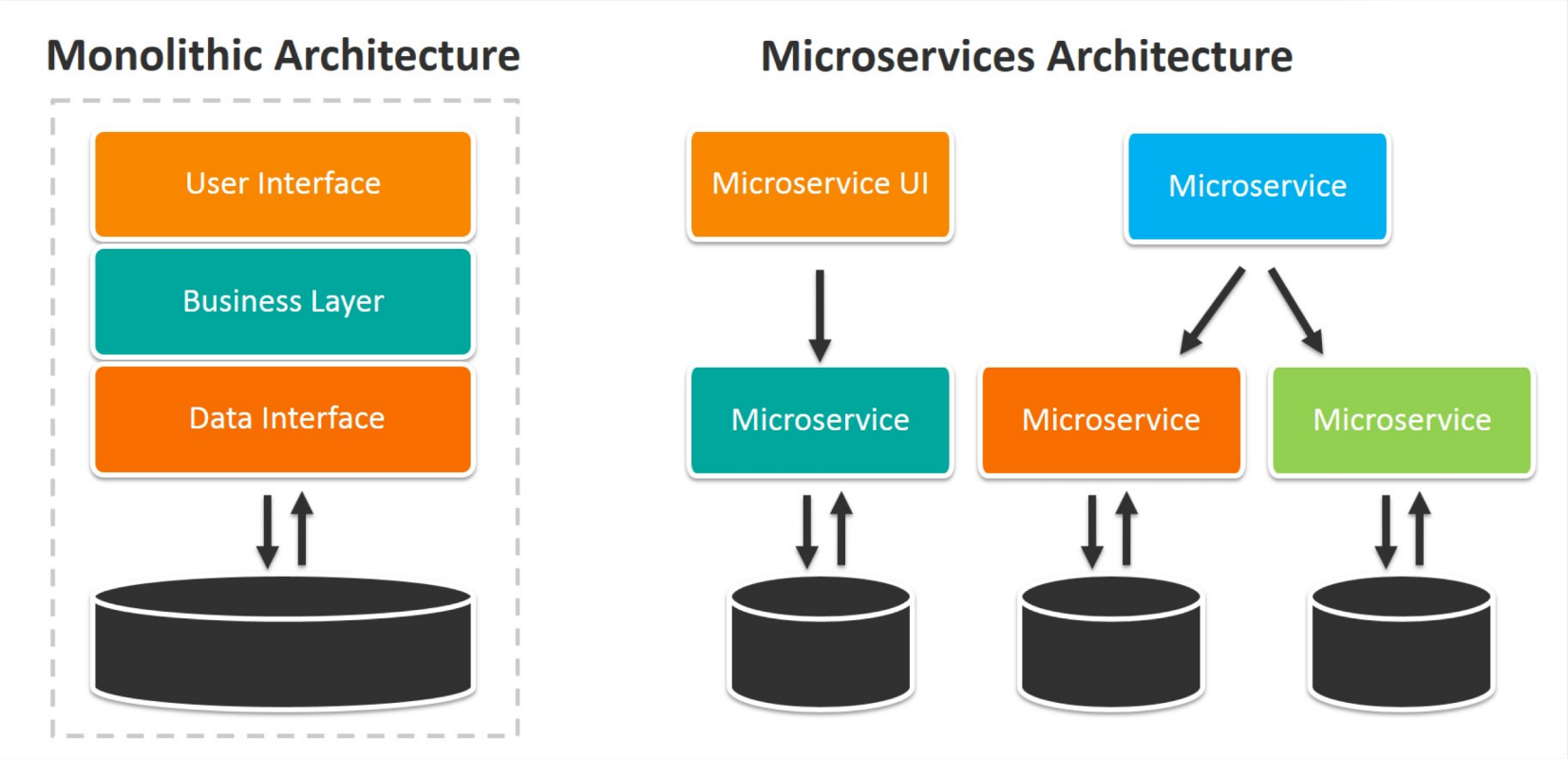


MicroService Mesh with sidecar

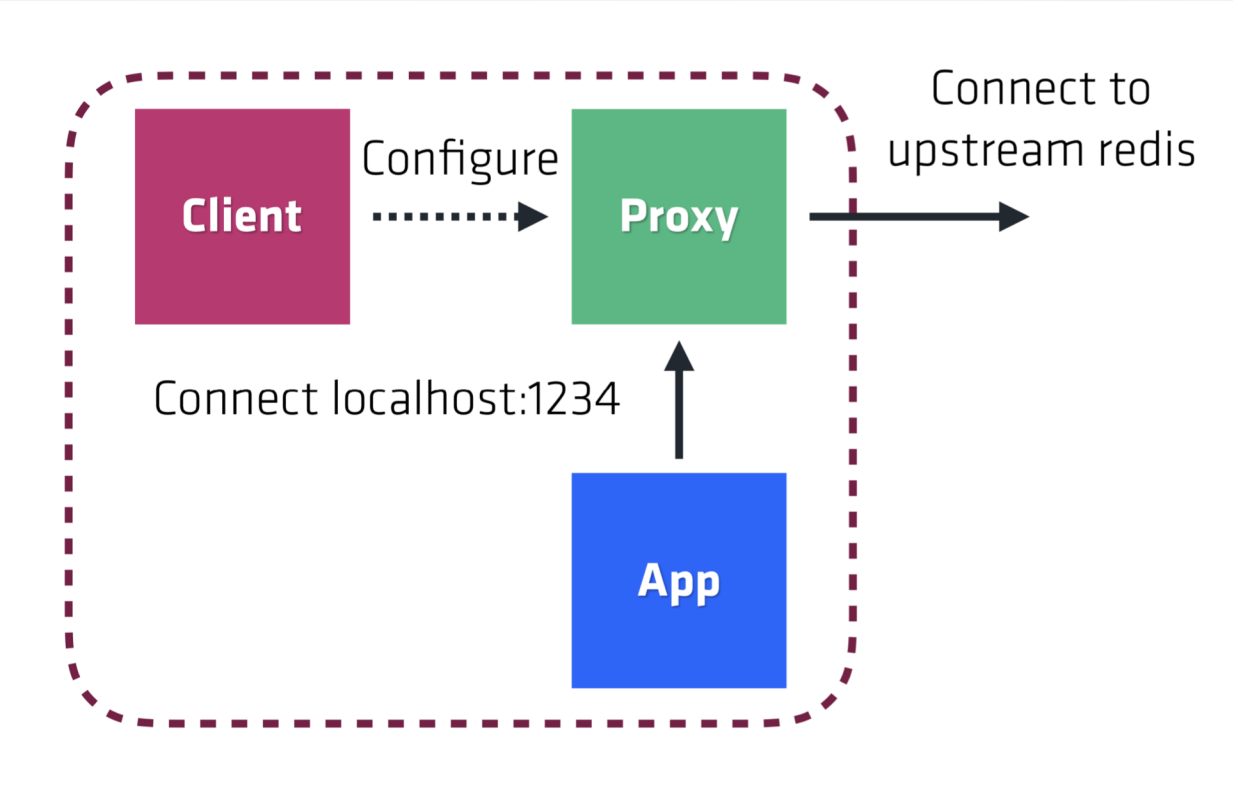
Control Plane vs Data Plane



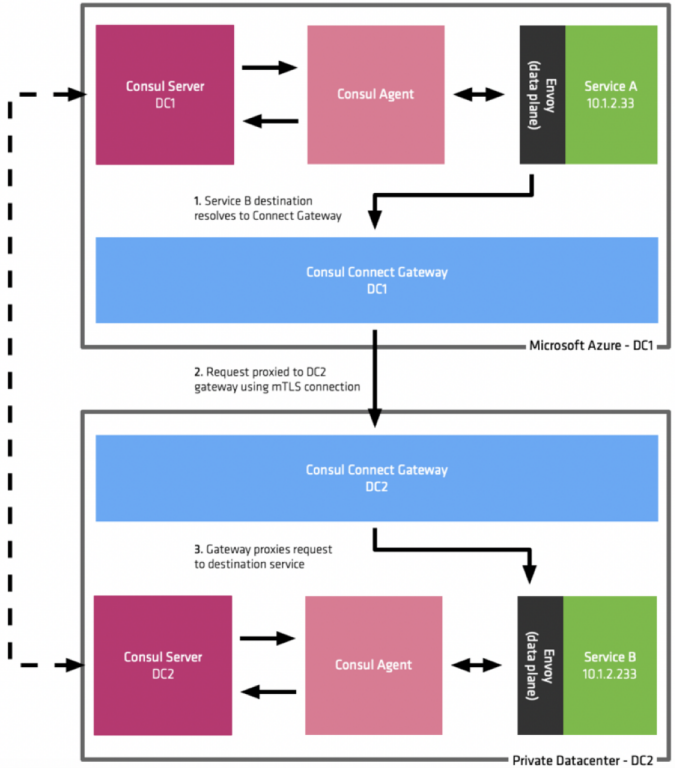
Monolithic vs Microservices



Connect to service through proxy



Mesh Gateways



Enable routing of Connect traffic between different Consul datacenters

Datacenters can reside in different clouds or runtime environments

Appropriate destination is based on the server name requested

Infrastructure Diagram

Mesh Gateways

Connect between different cloud regions,
VPCs without complex network tunnel