
NOVATEC
Java Microservices – Lieber mit Frameworks, Service Meshes oder beidem?
Matthias Häußler, Novatec
JAVA FORUM stuttgart
1997-2022
25 Jubiläum

#whoami

matthiashaeussler

@maeddes

NOVATEC

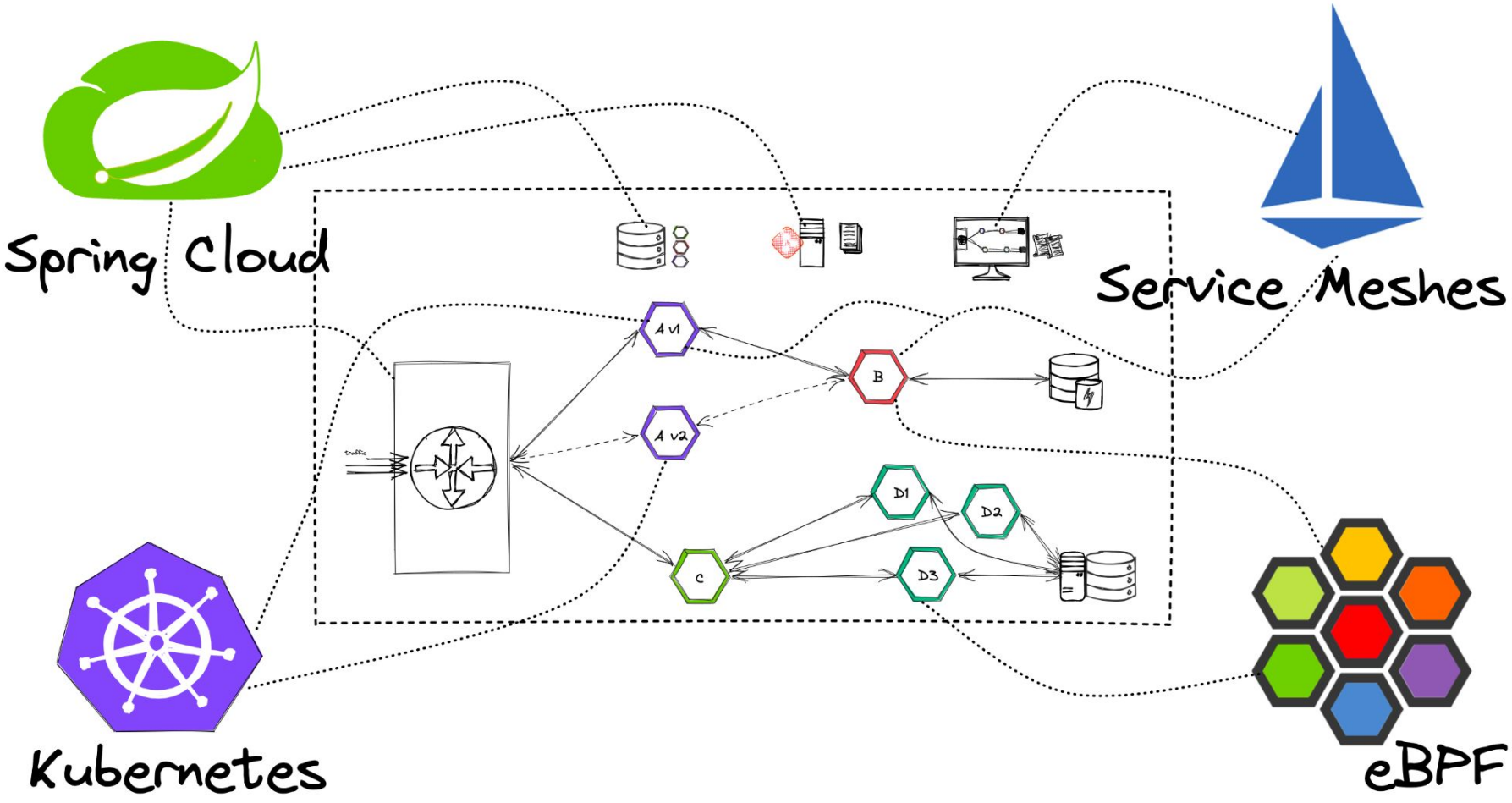
Chief Technologist

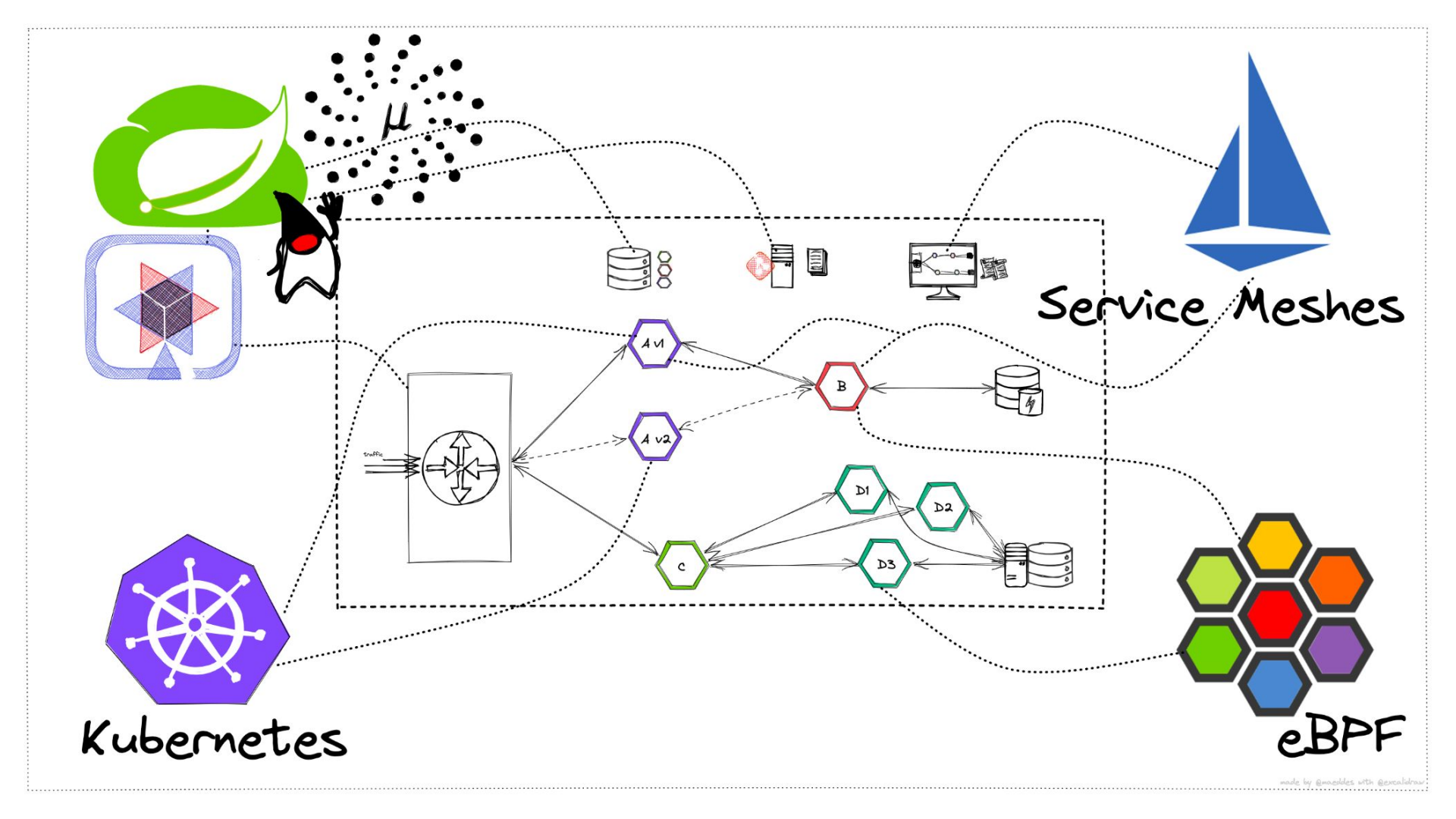
Hochschule für Technik Stuttgart

ESSLINGEN UNIVERSITY

For people and technology.

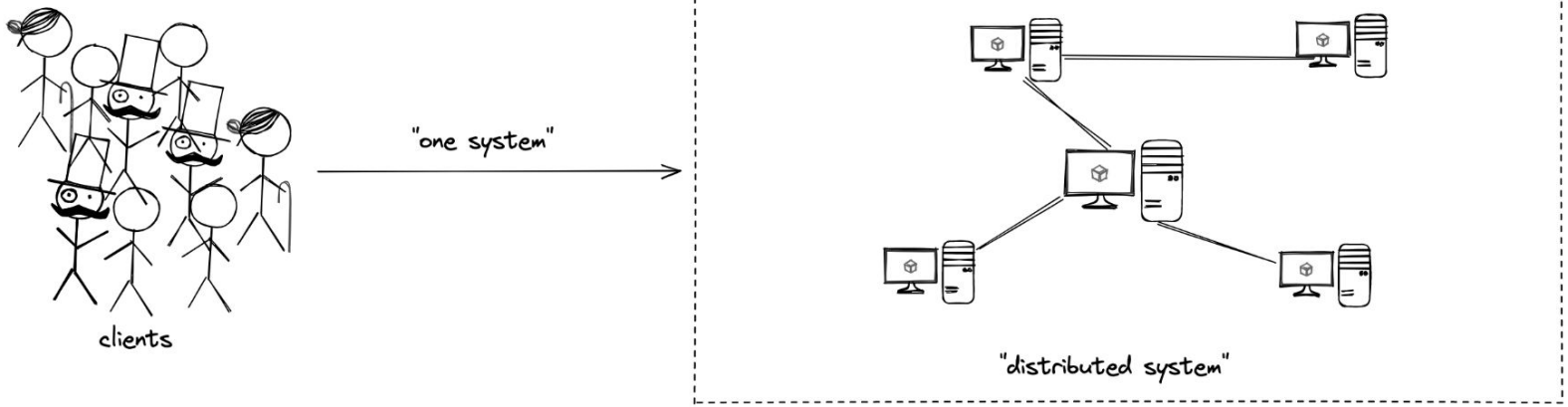
Distributed Systems

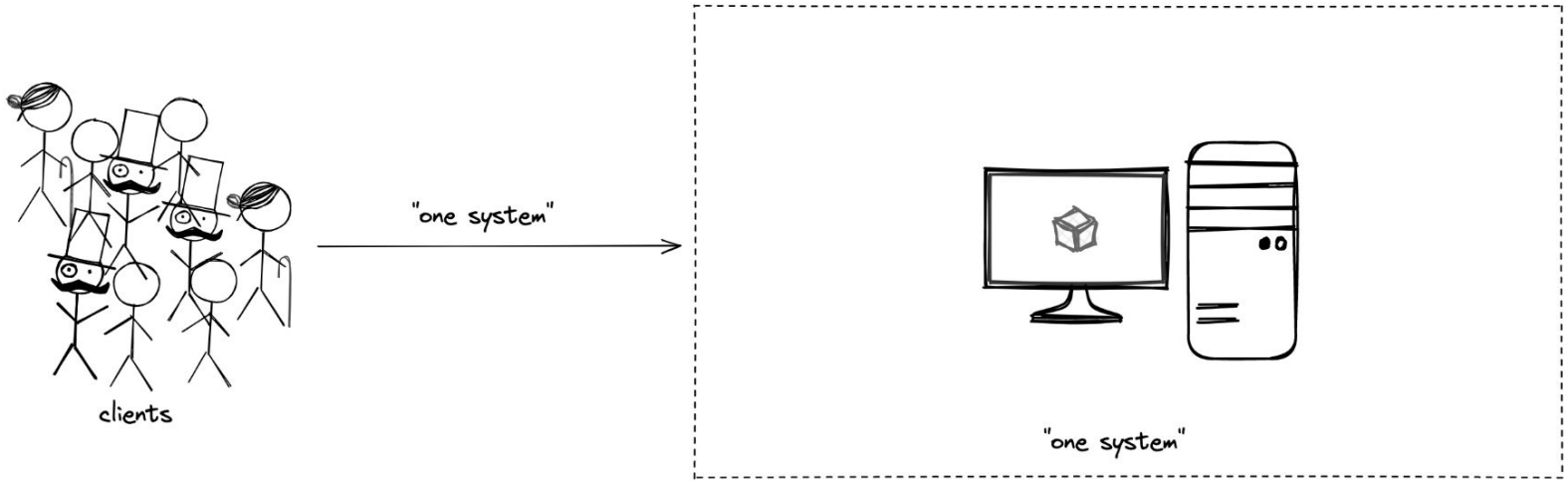

Spring Cloud
Service Meshes
A v1
A v2
B
C
D1
D2
D3
traffic
Kubernetes
eBPF


Service Meshes
A v1
A v2
B
C
D1
D2
D3
traffic
Kubernetes
eBPF

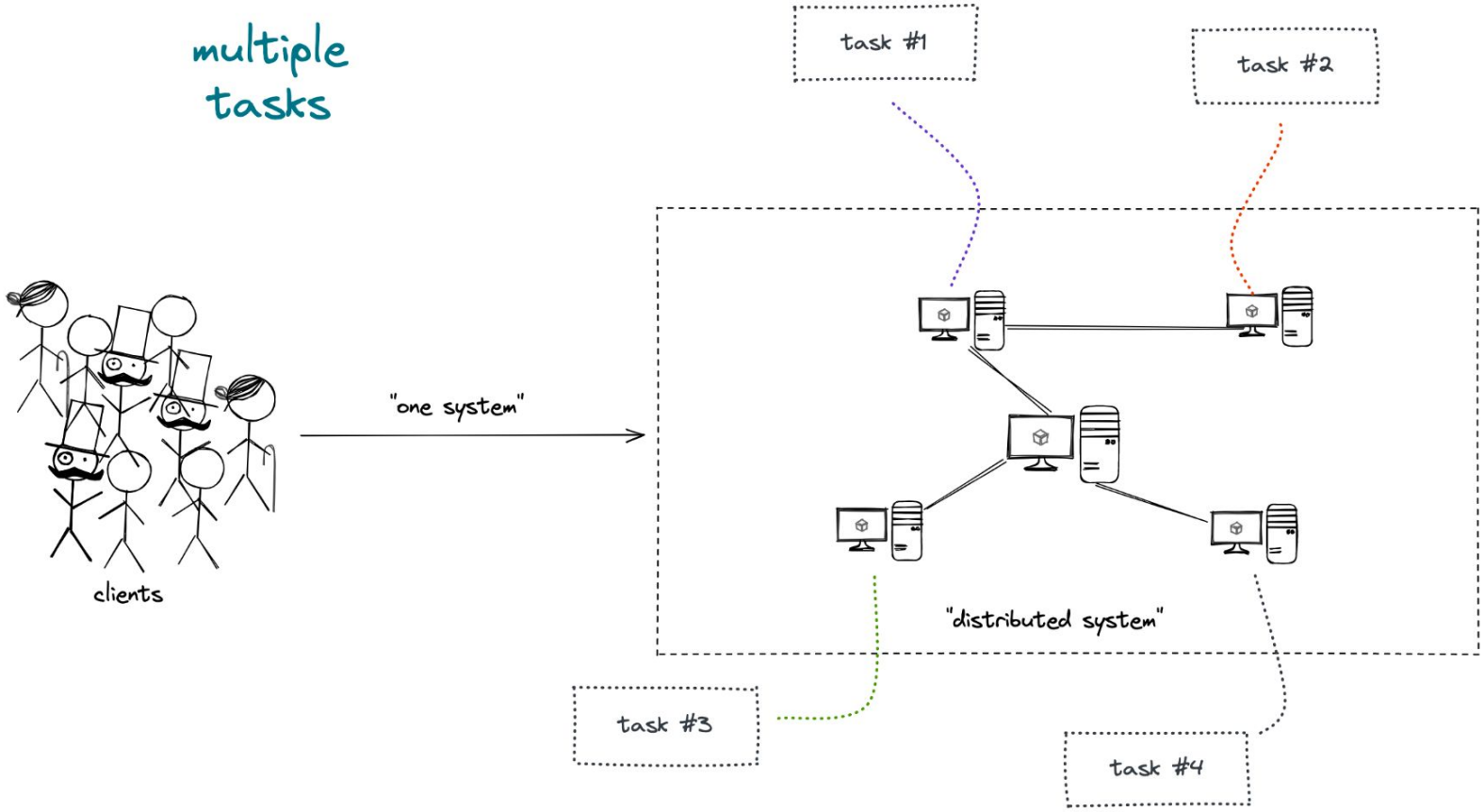
- why?
- how?
- what?
    - frameworks stand-alone
    - Kubernetes with and without frameworks
    - Service Meshes
    - eBPF
- conclusion
- live demo?

Why distributed architecture?

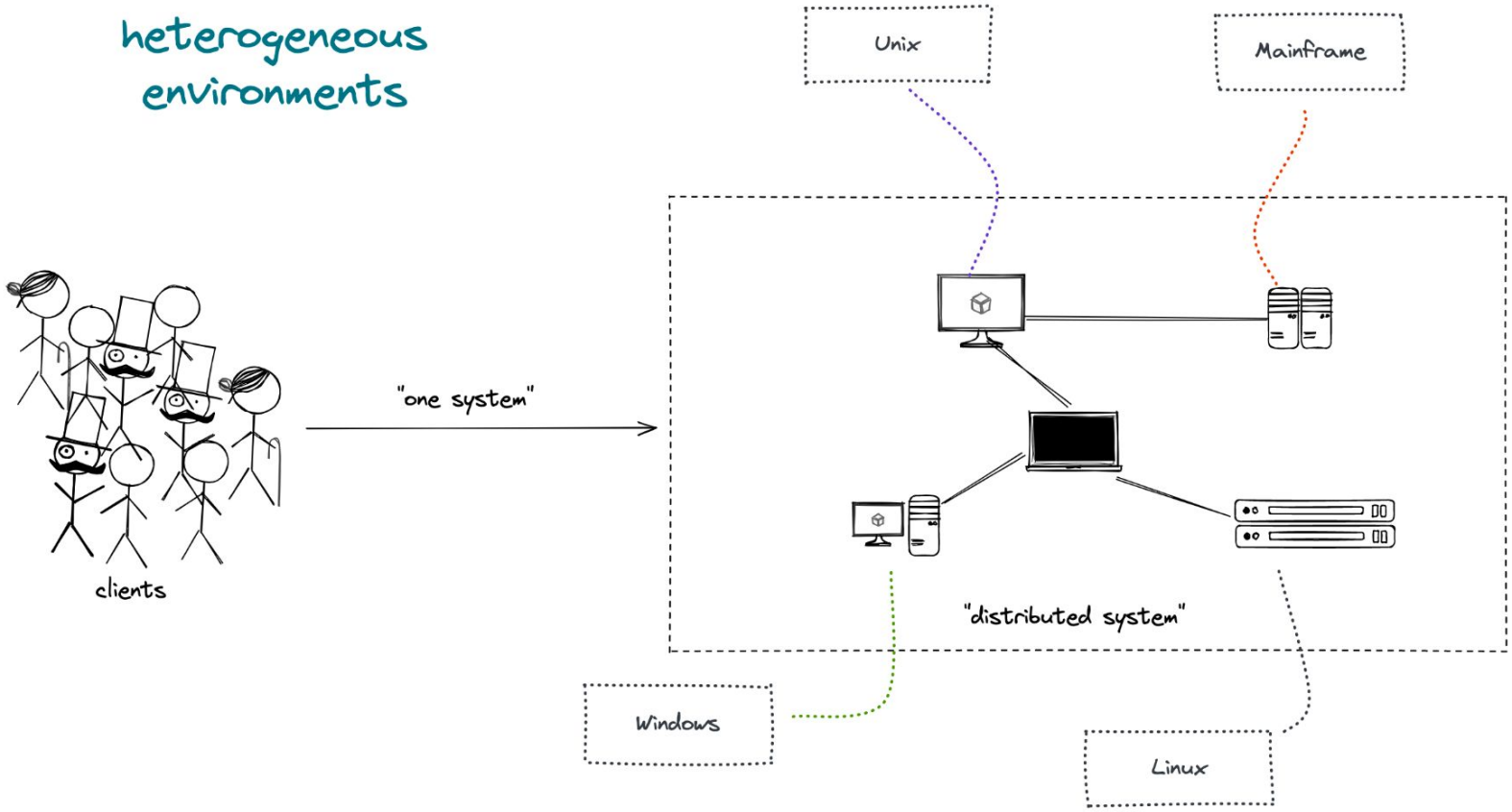




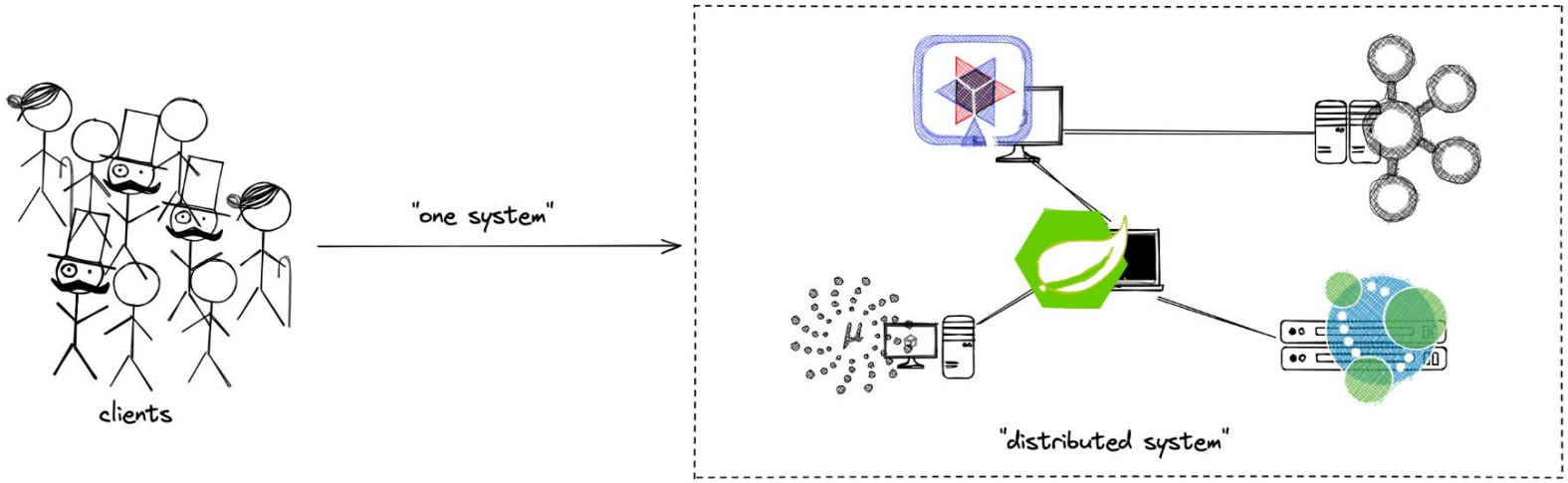
multiple tasks



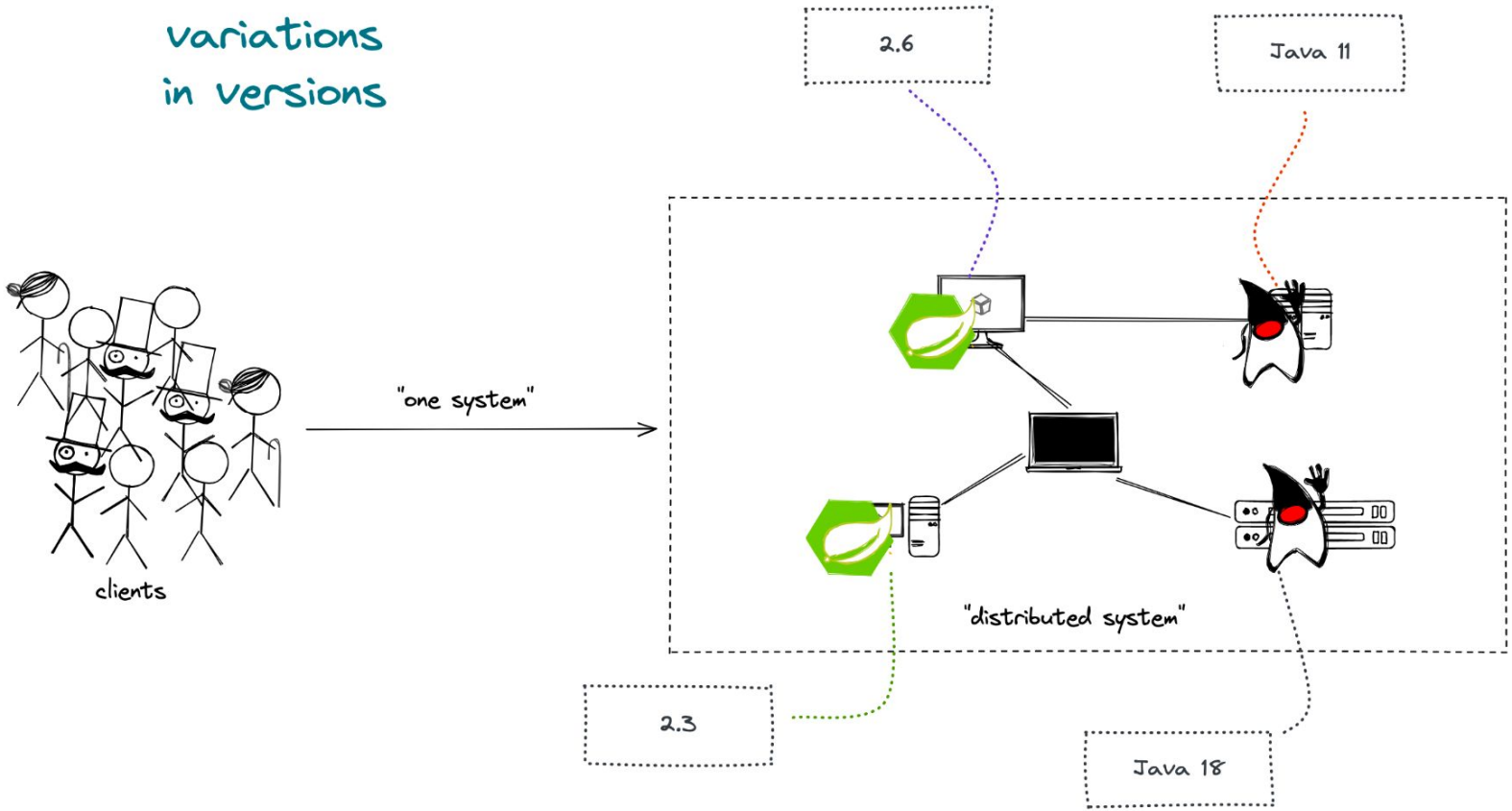
heterogeneous environments



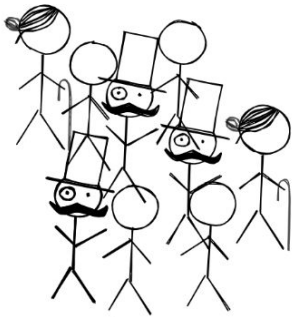
heterogeneous components / frameworks



variations in versions

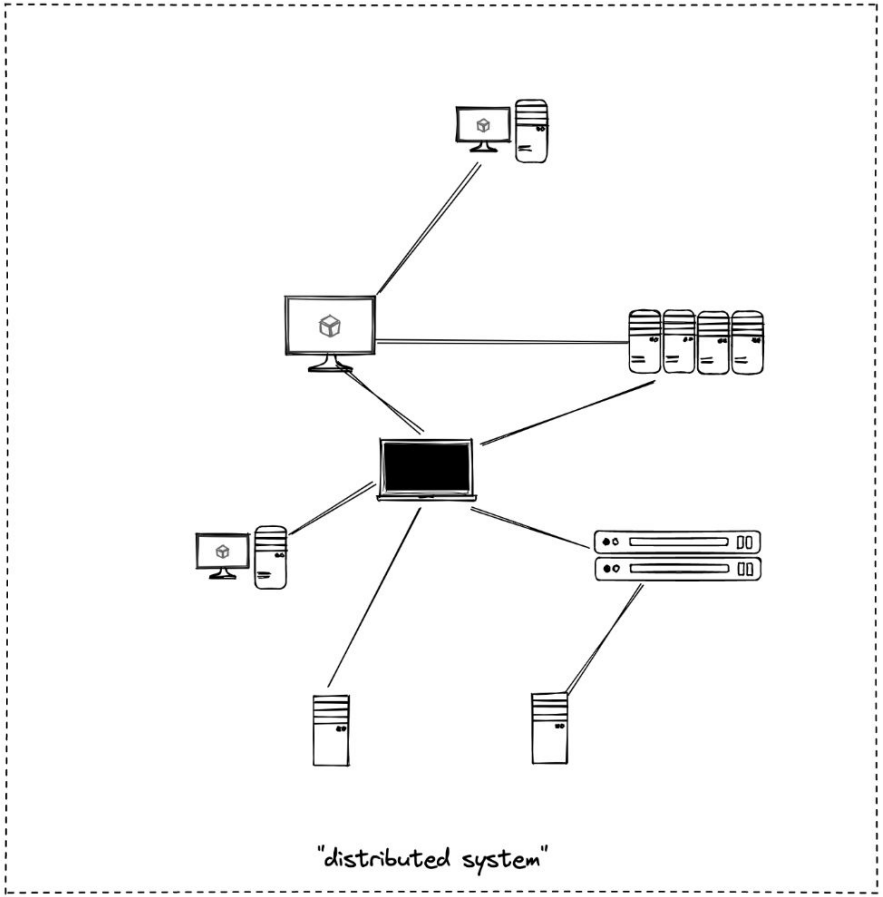


extensibility



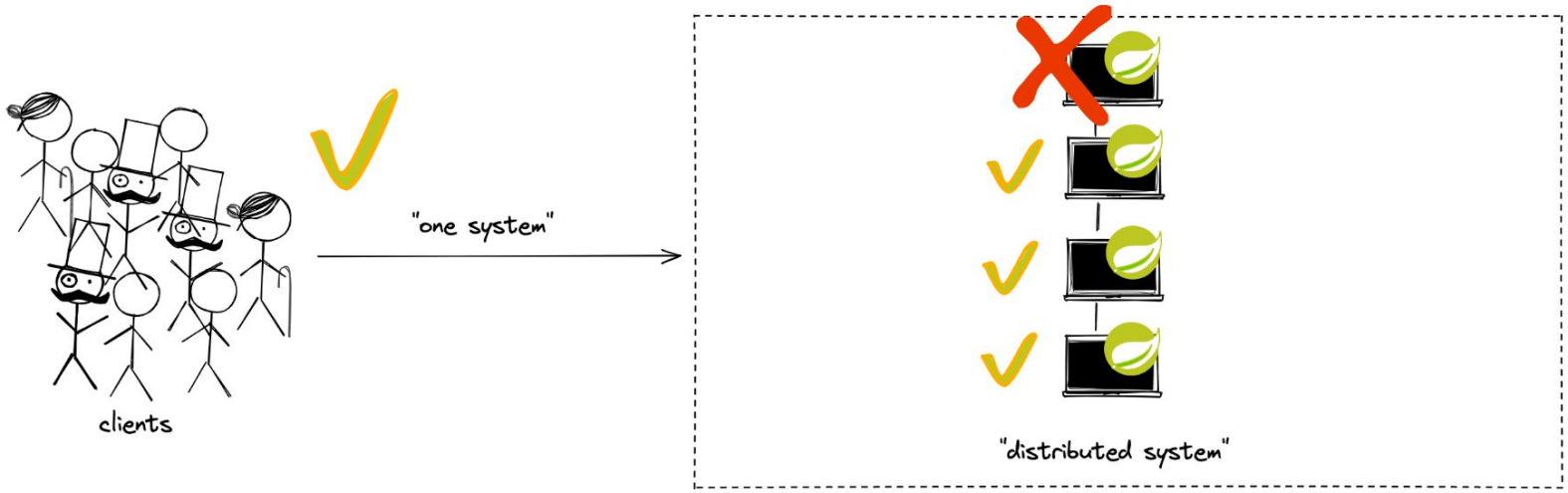
clients

"one system"

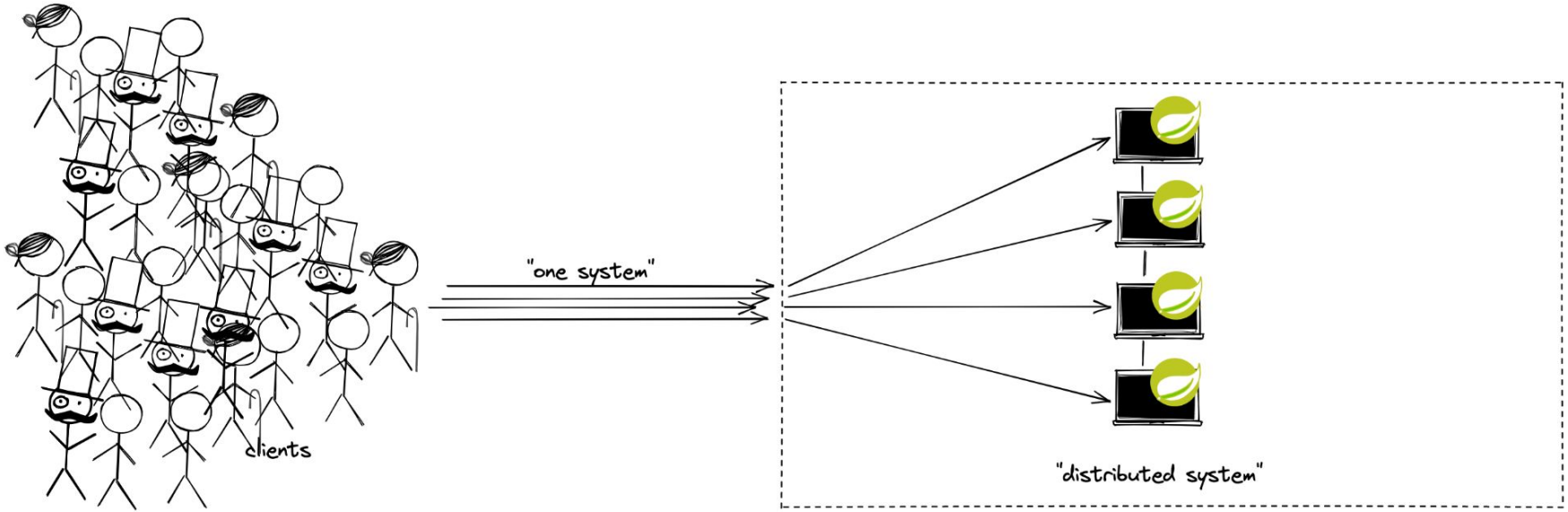


"distributed system"

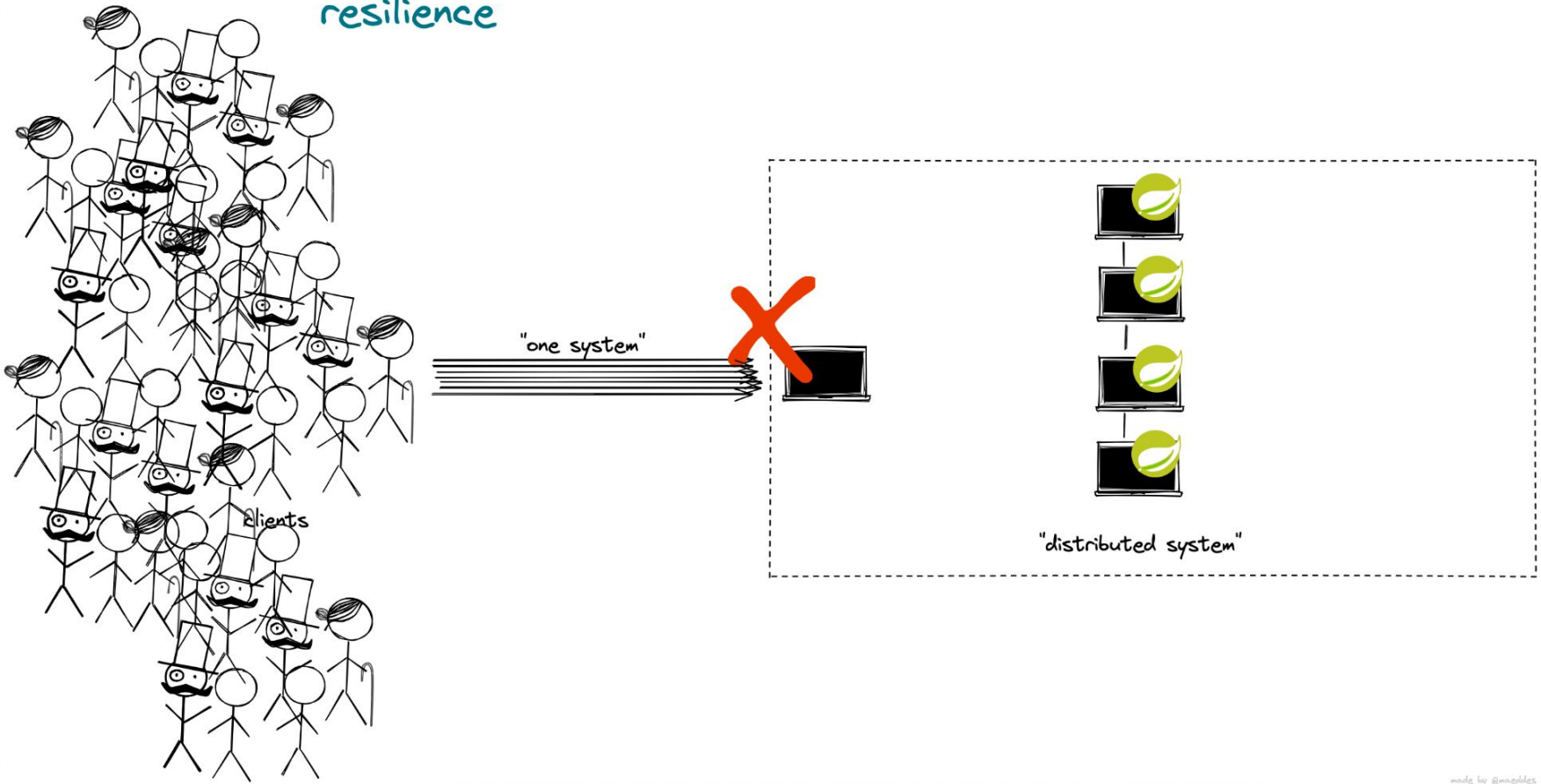
redundancy



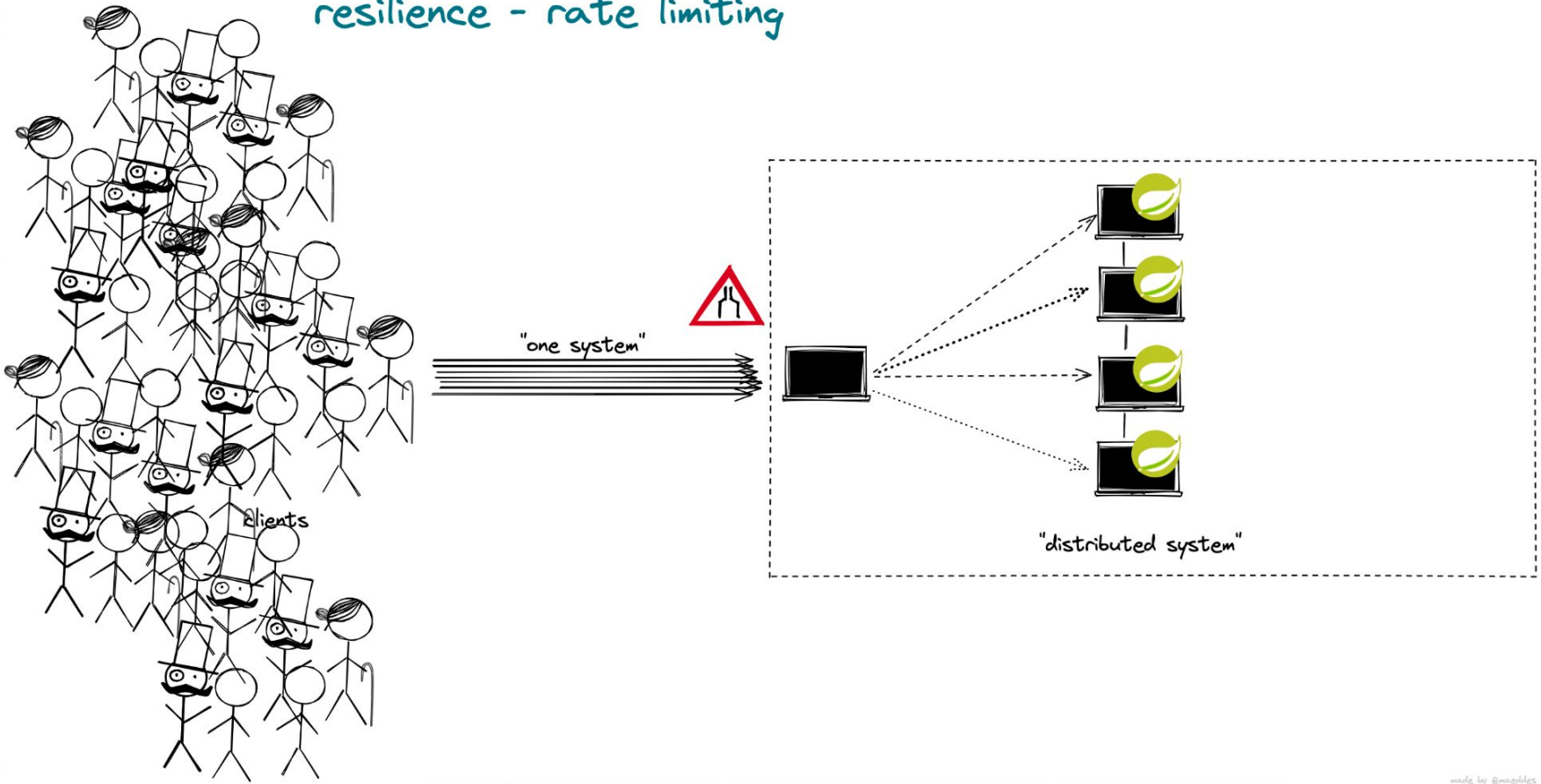
load-balancing

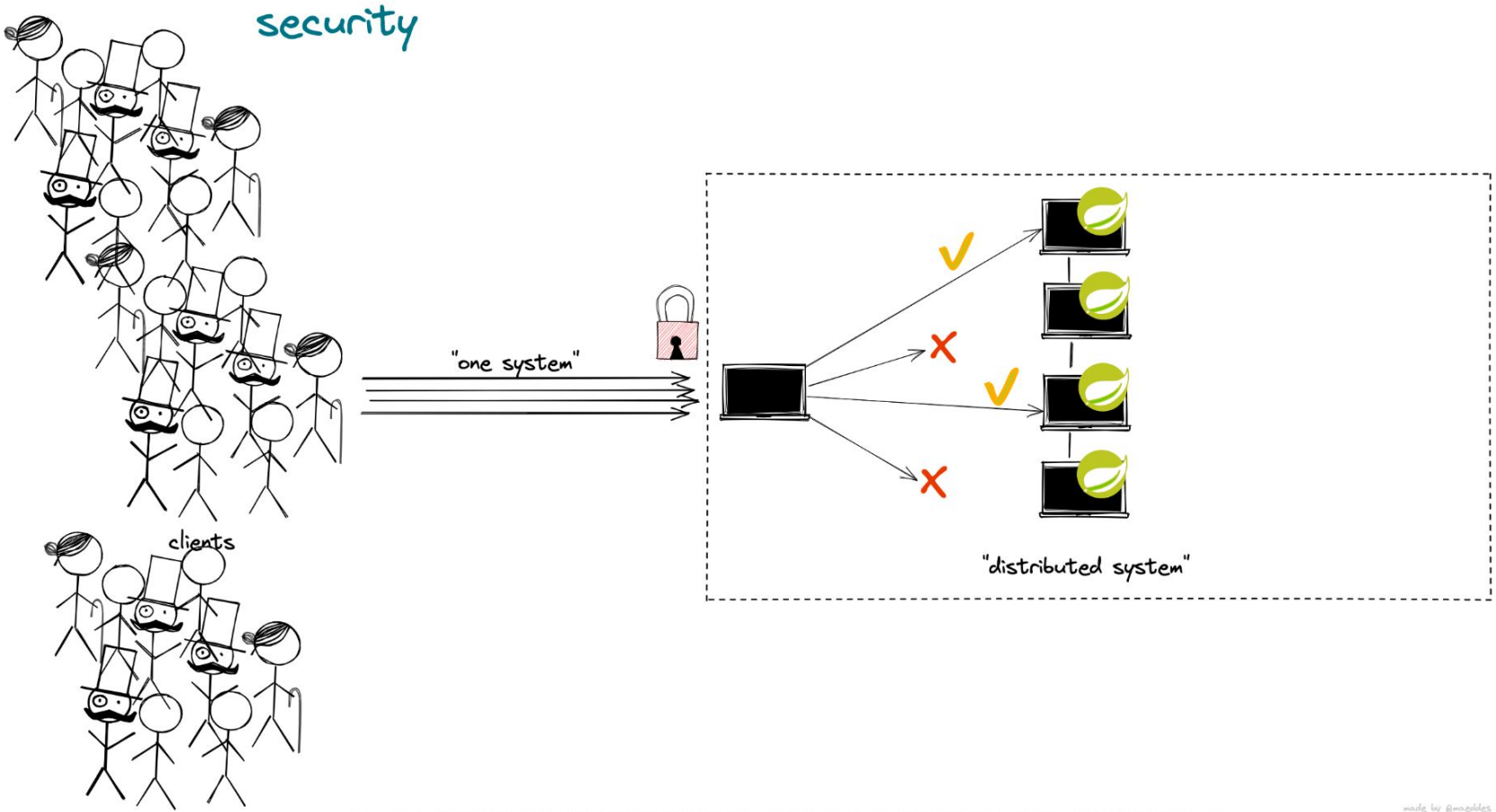


resilience

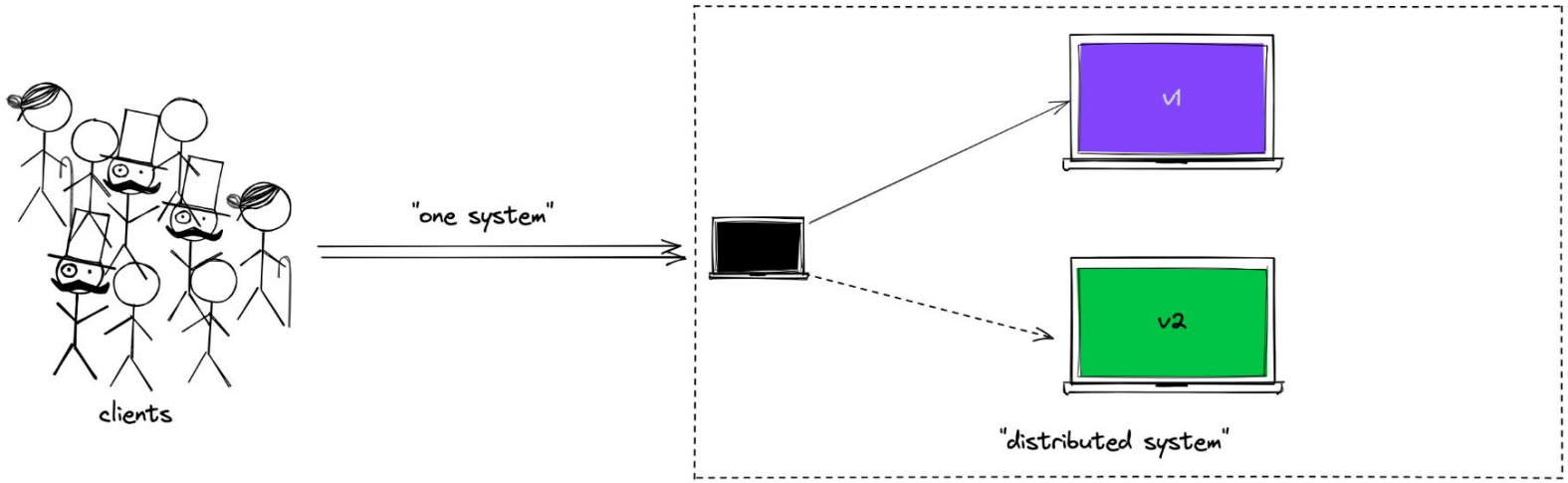


resilience - rate limiting

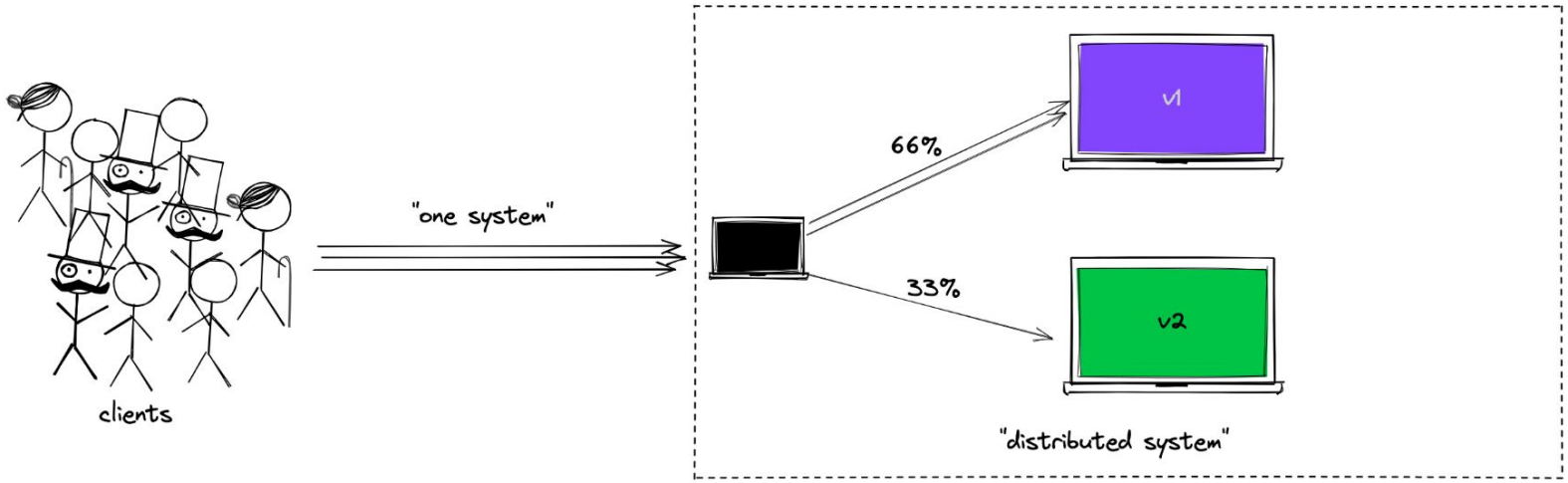



security
clients
"one system"
"distributed system"

dynamic updates - blue / green

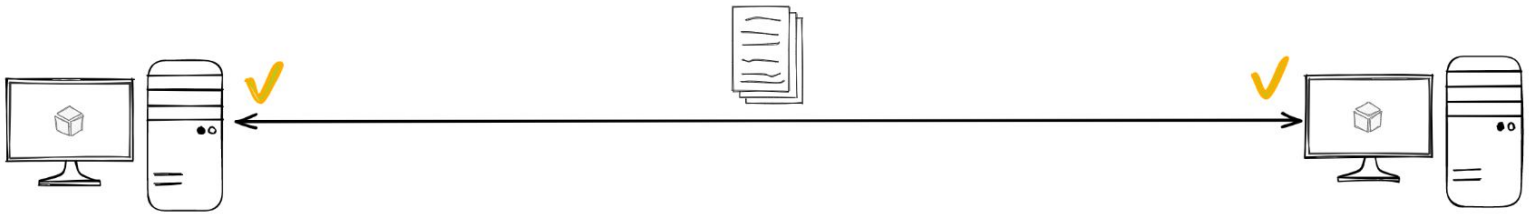


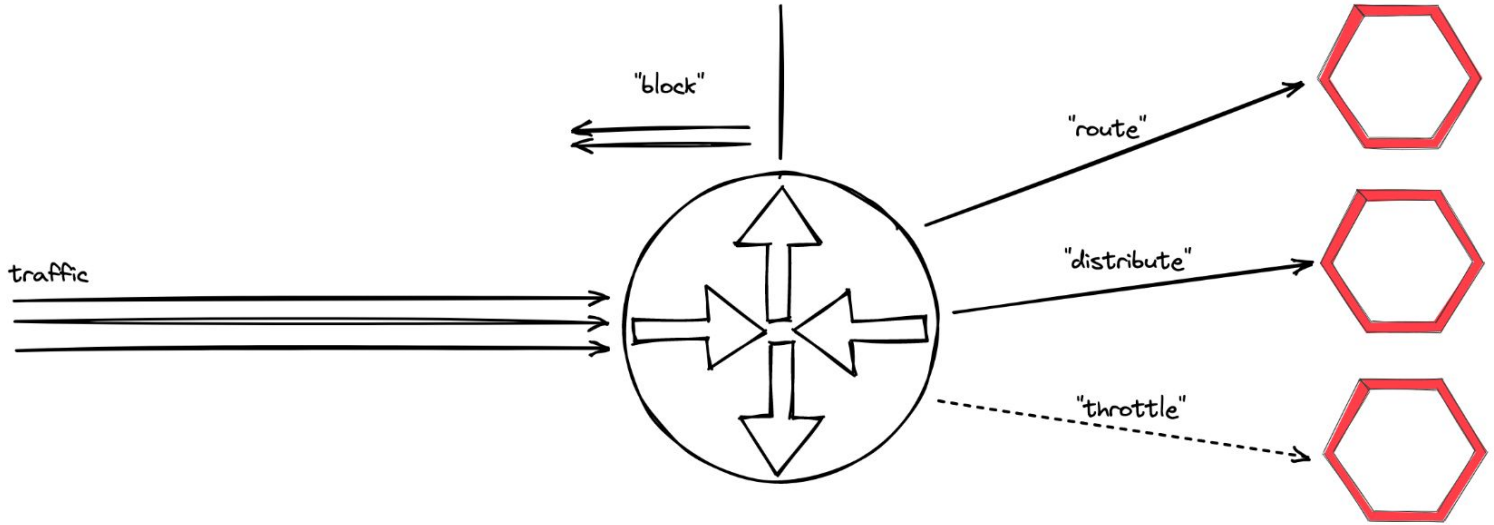
dynamic updates - canary / percentage



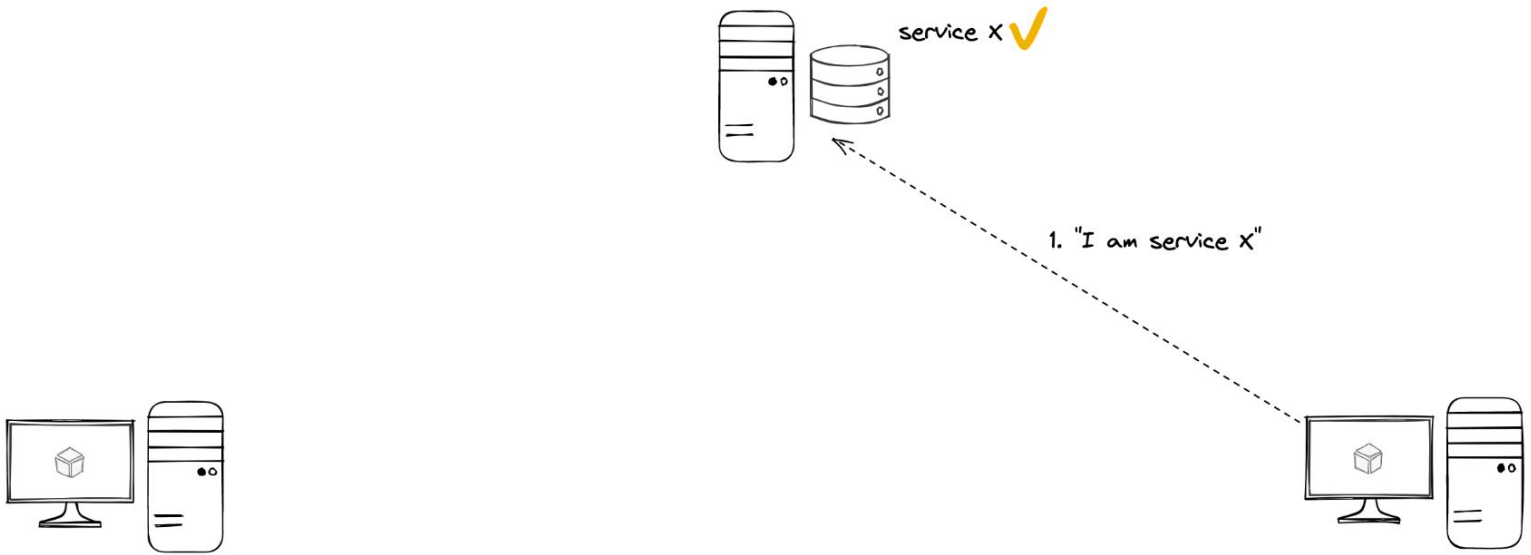
Requirements for implementing distributed systems

communication

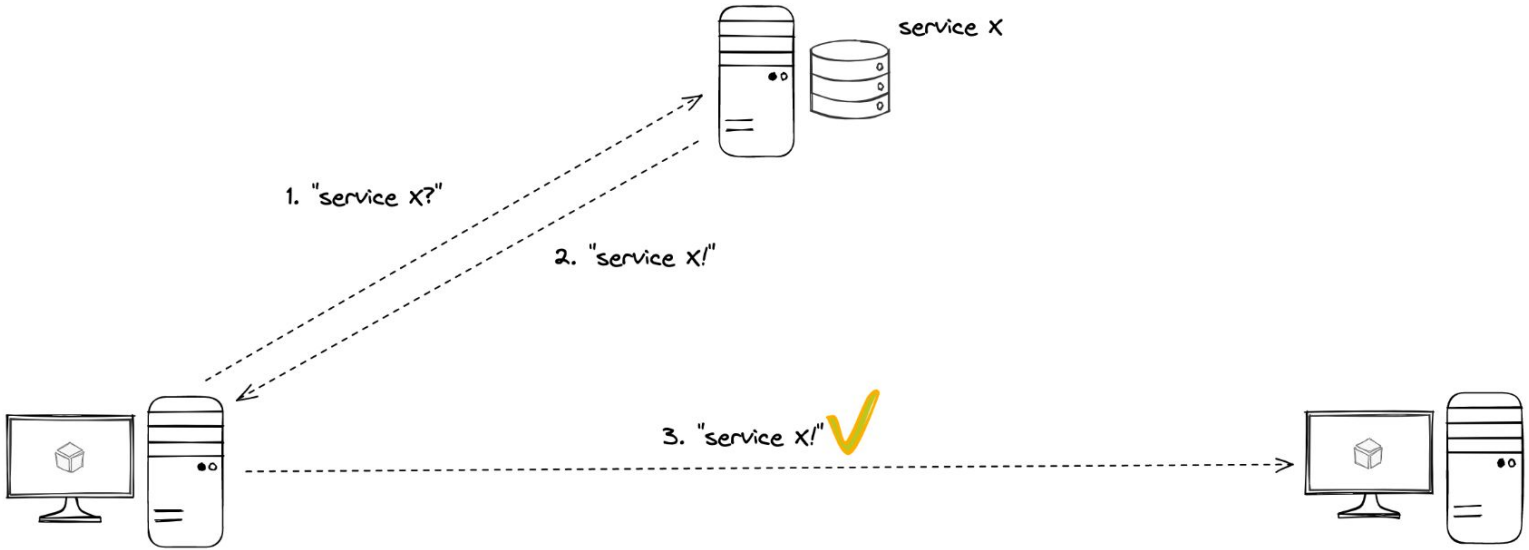




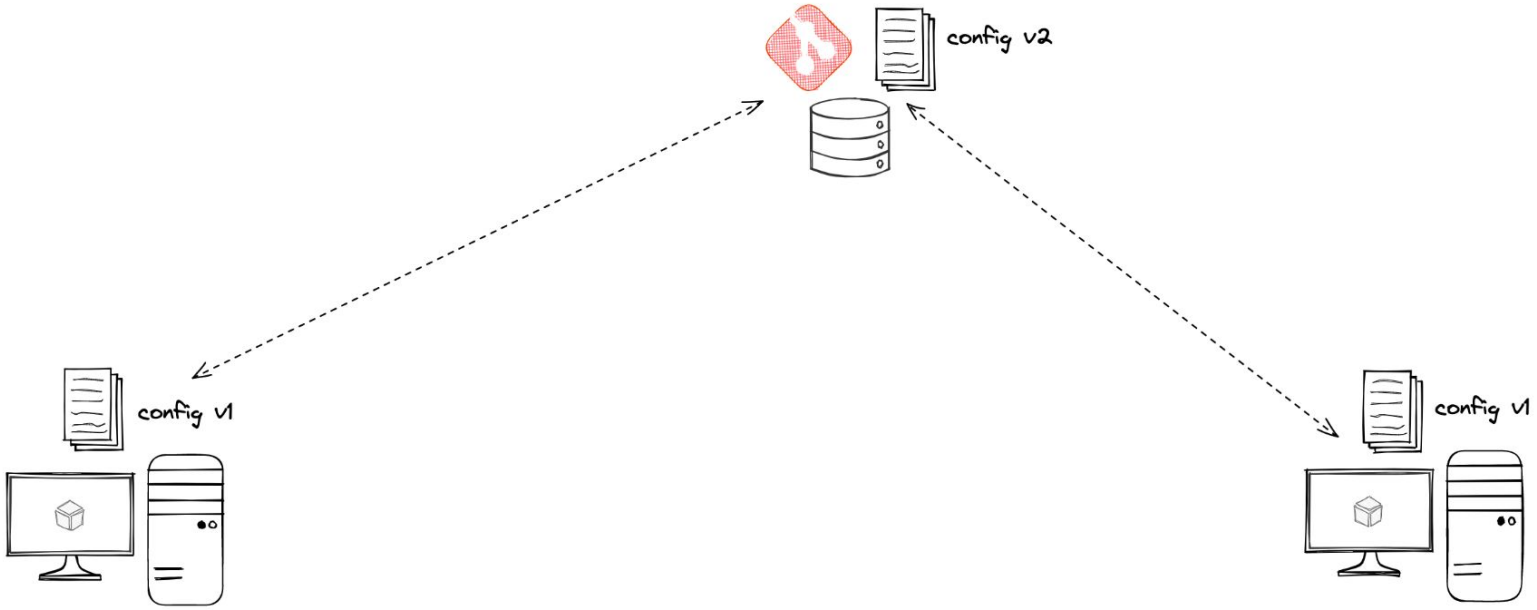
routing / traffic shaping



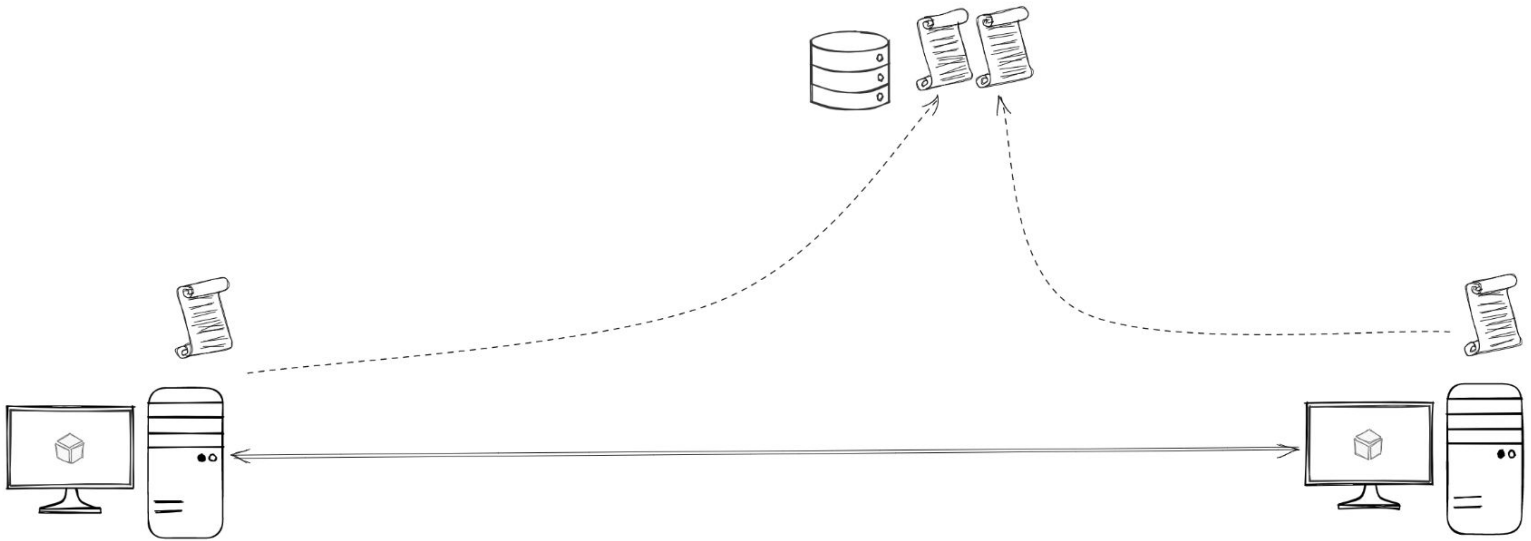
service registration



service discovery

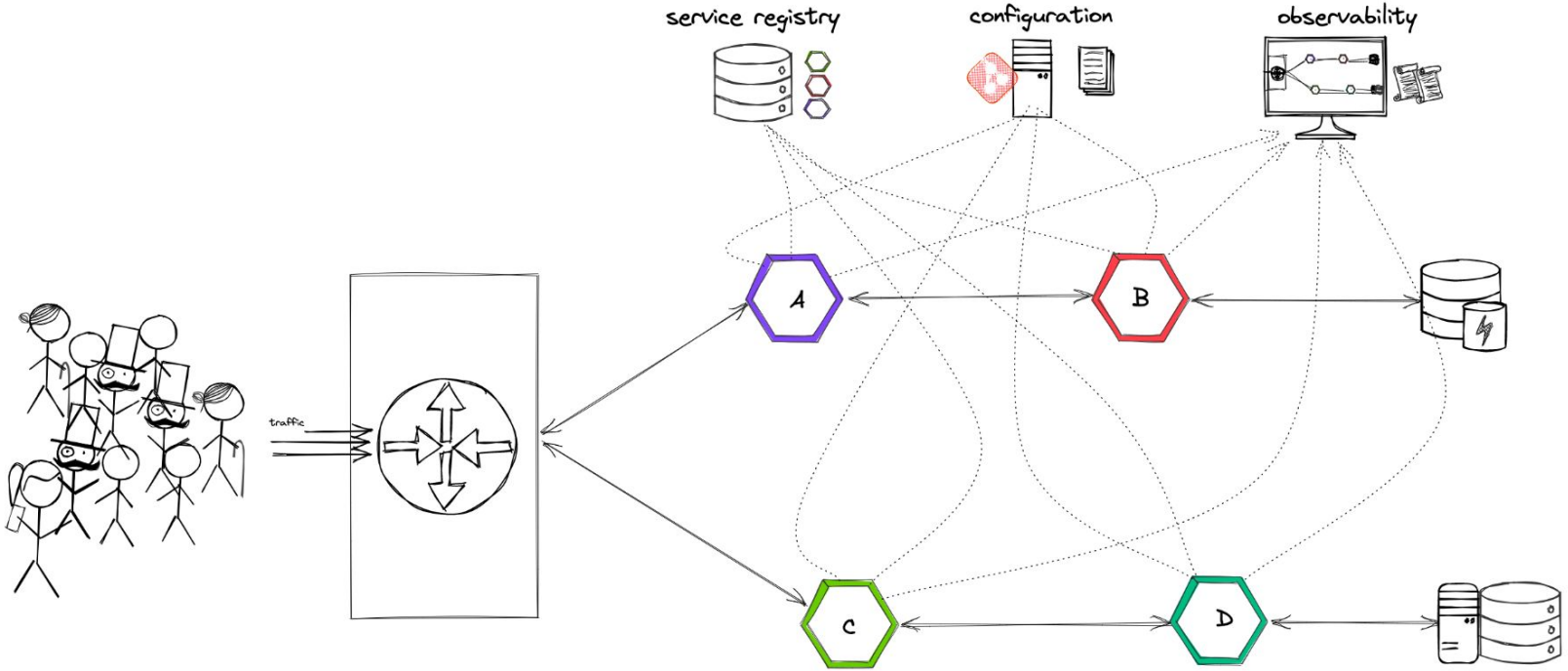


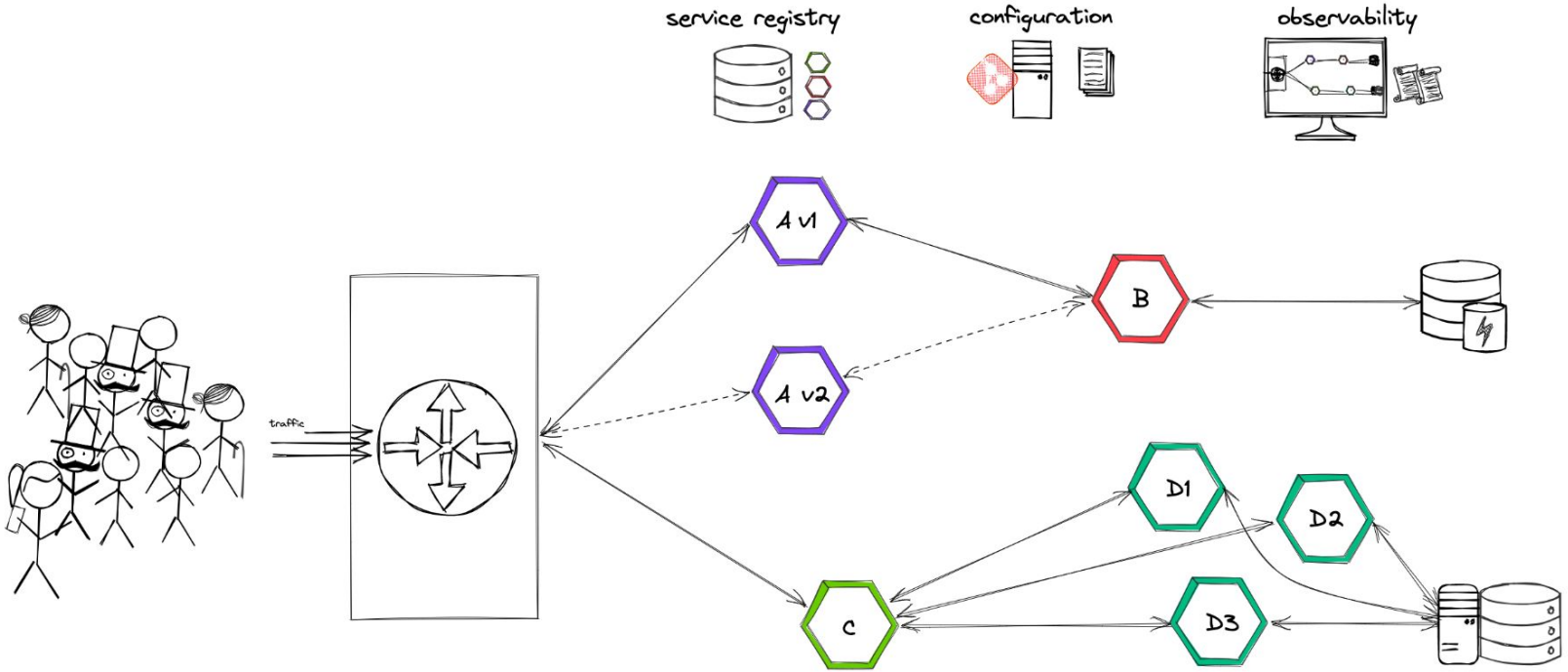
distributed configuration



distributed logging and tracing

Distributed Architecture Sample



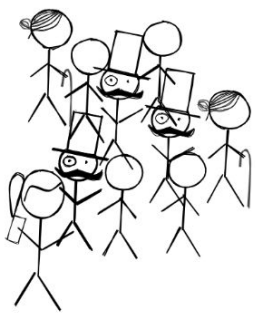


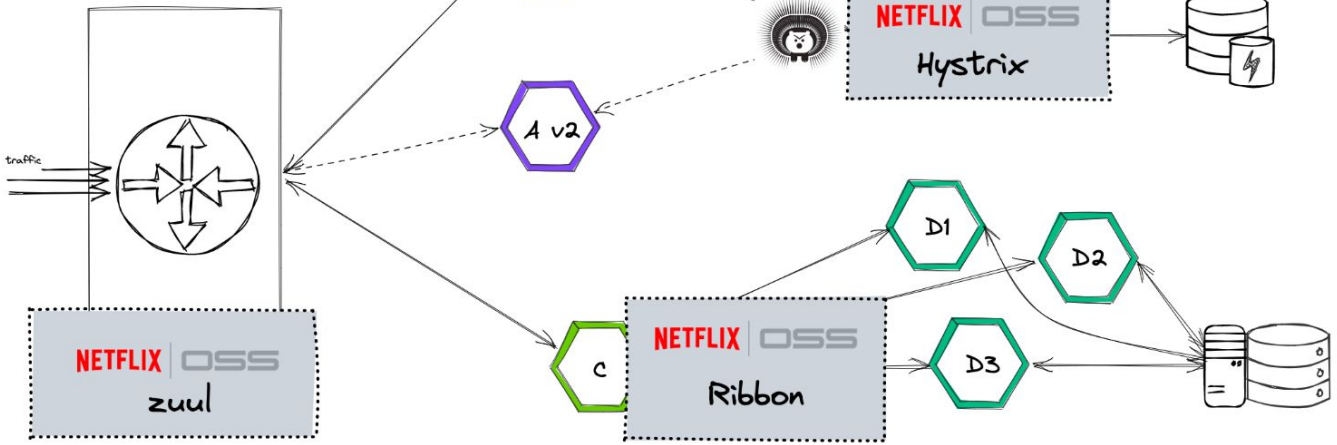
the

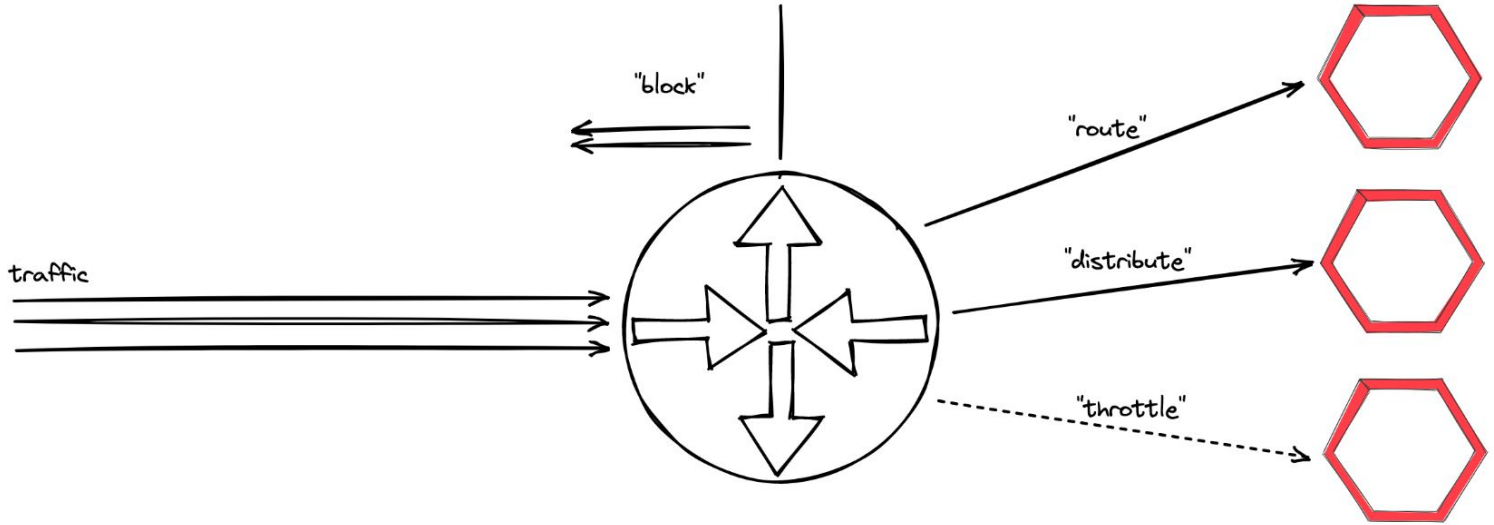
NETFLIX

approach

Netflix OSS 2015

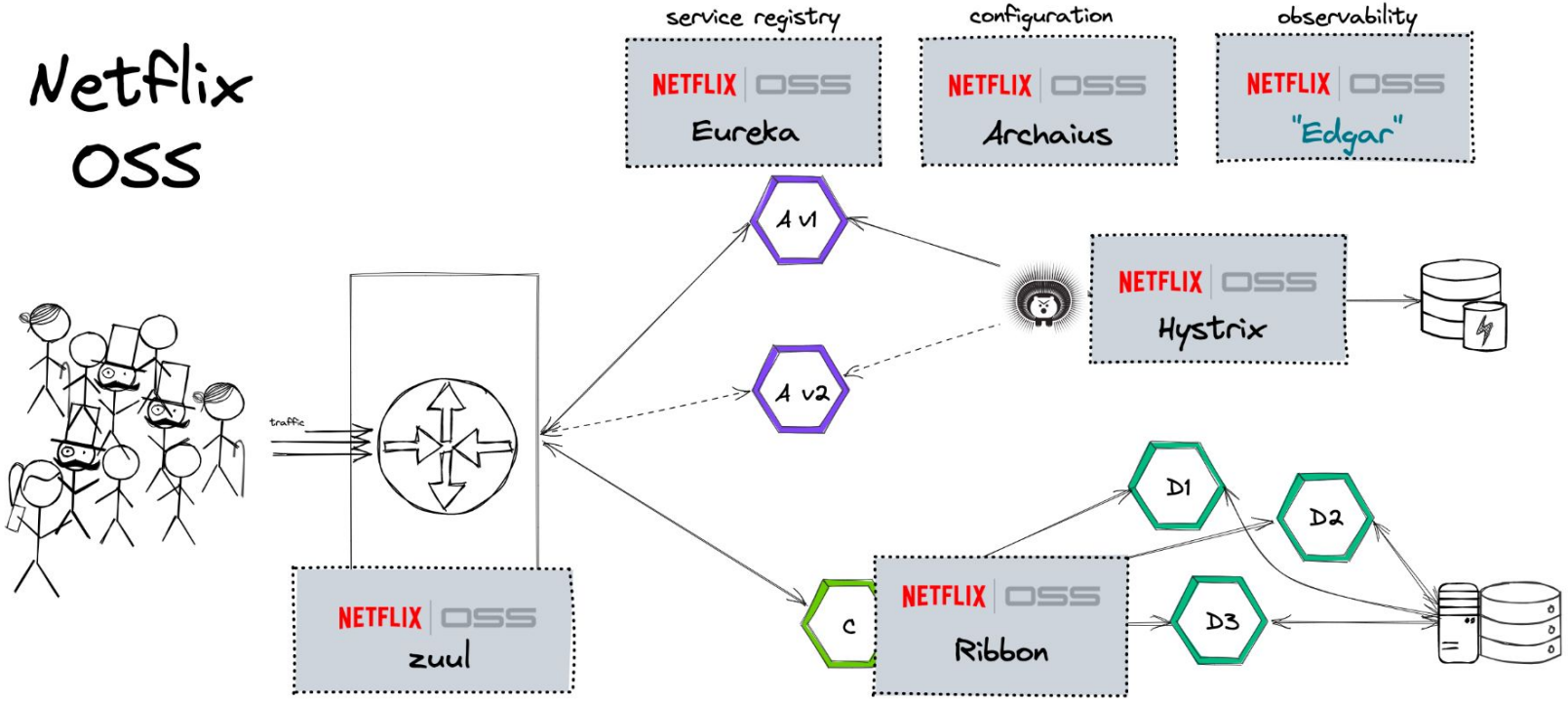




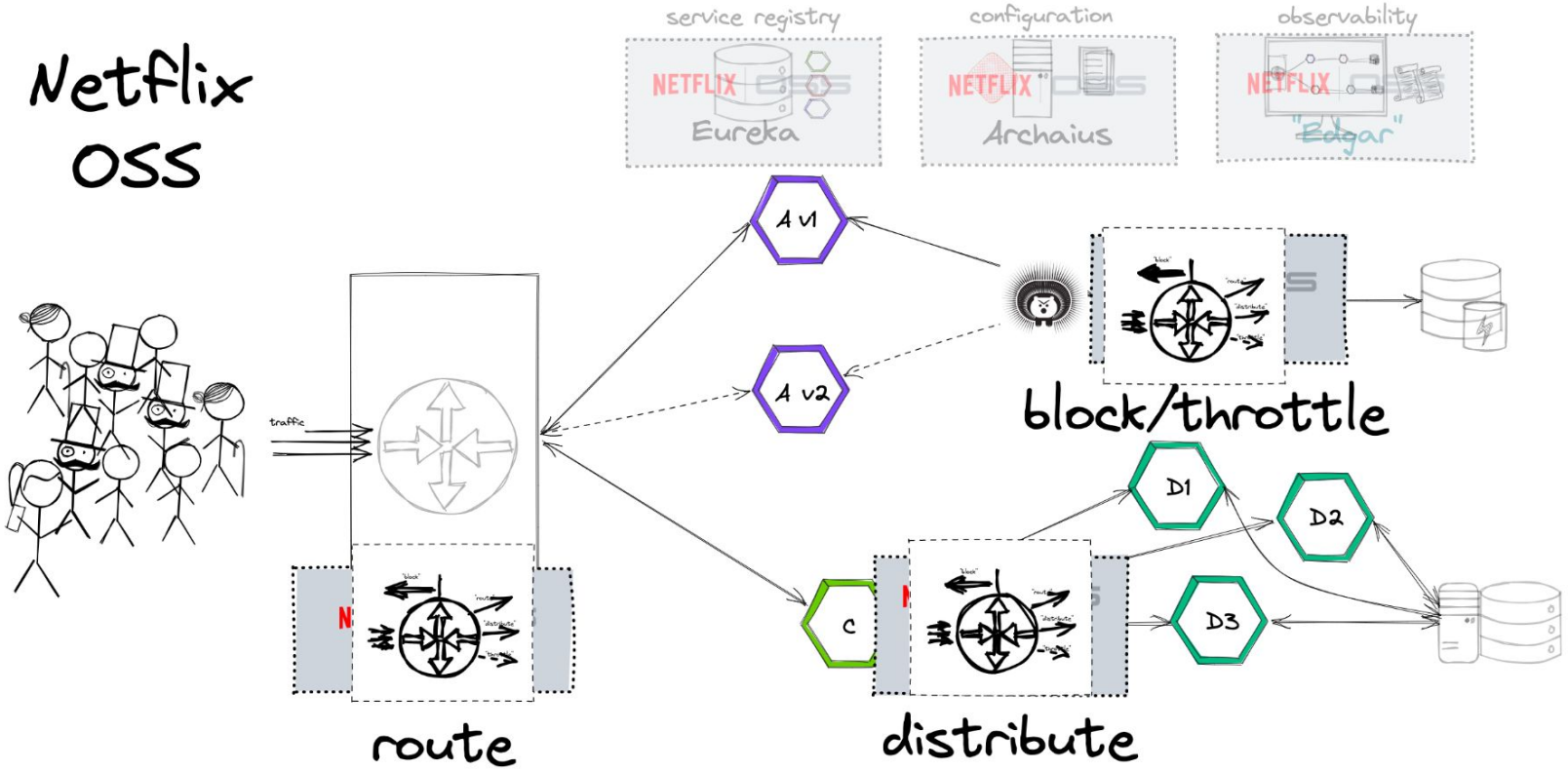


routing / traffic shaping

Netflix OSS



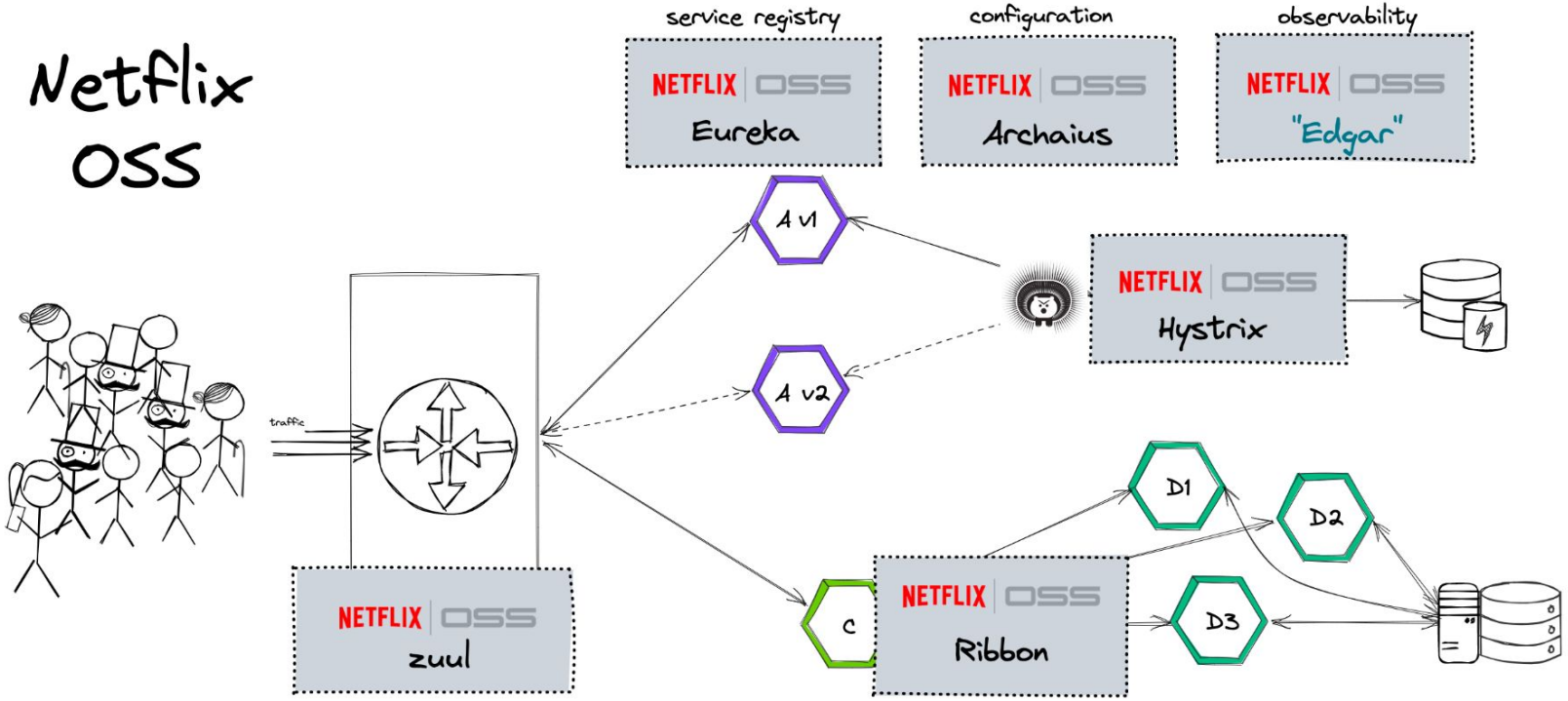
Netflix OSS



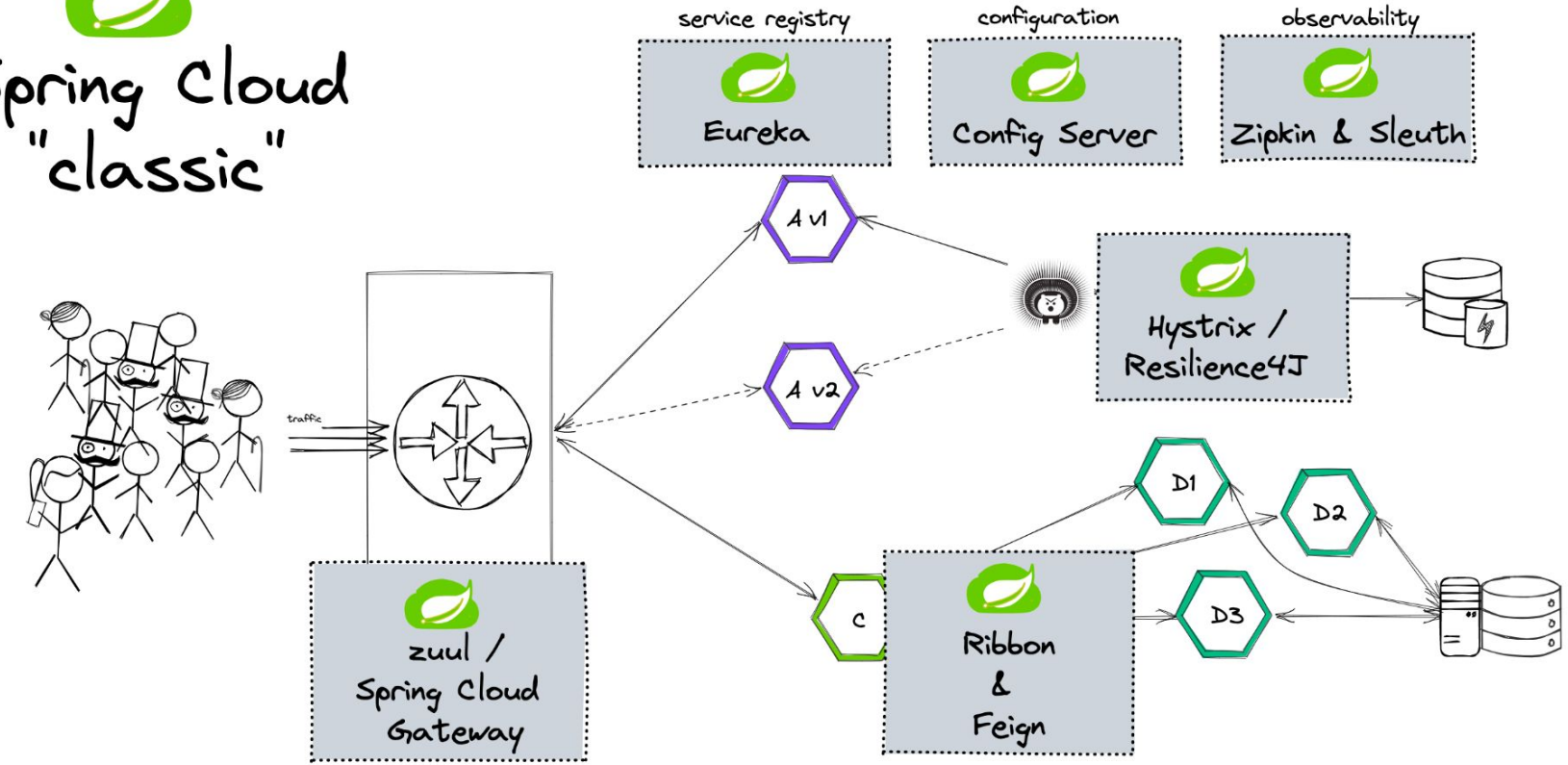
the

approach

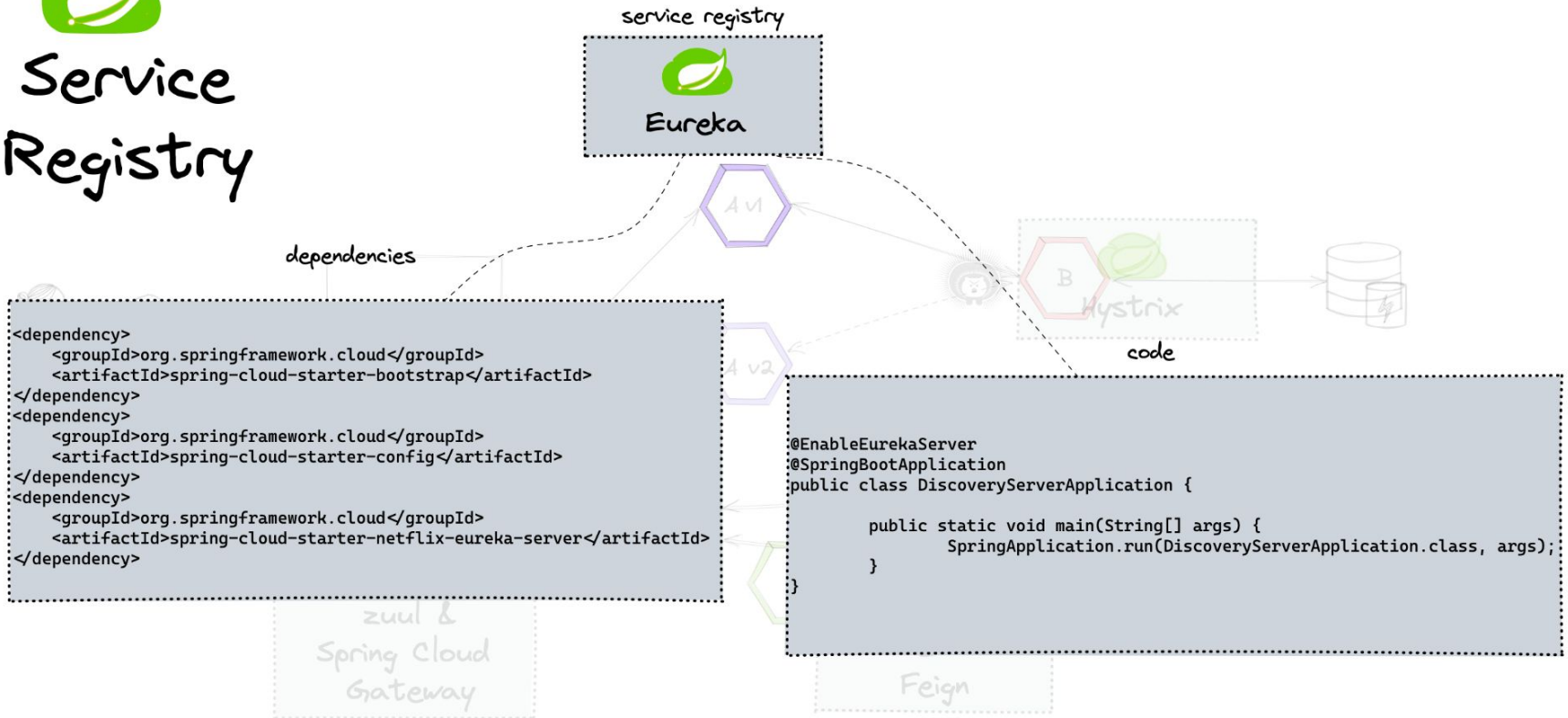
Netflix OSS



Spring Cloud "classic"



Service Registry

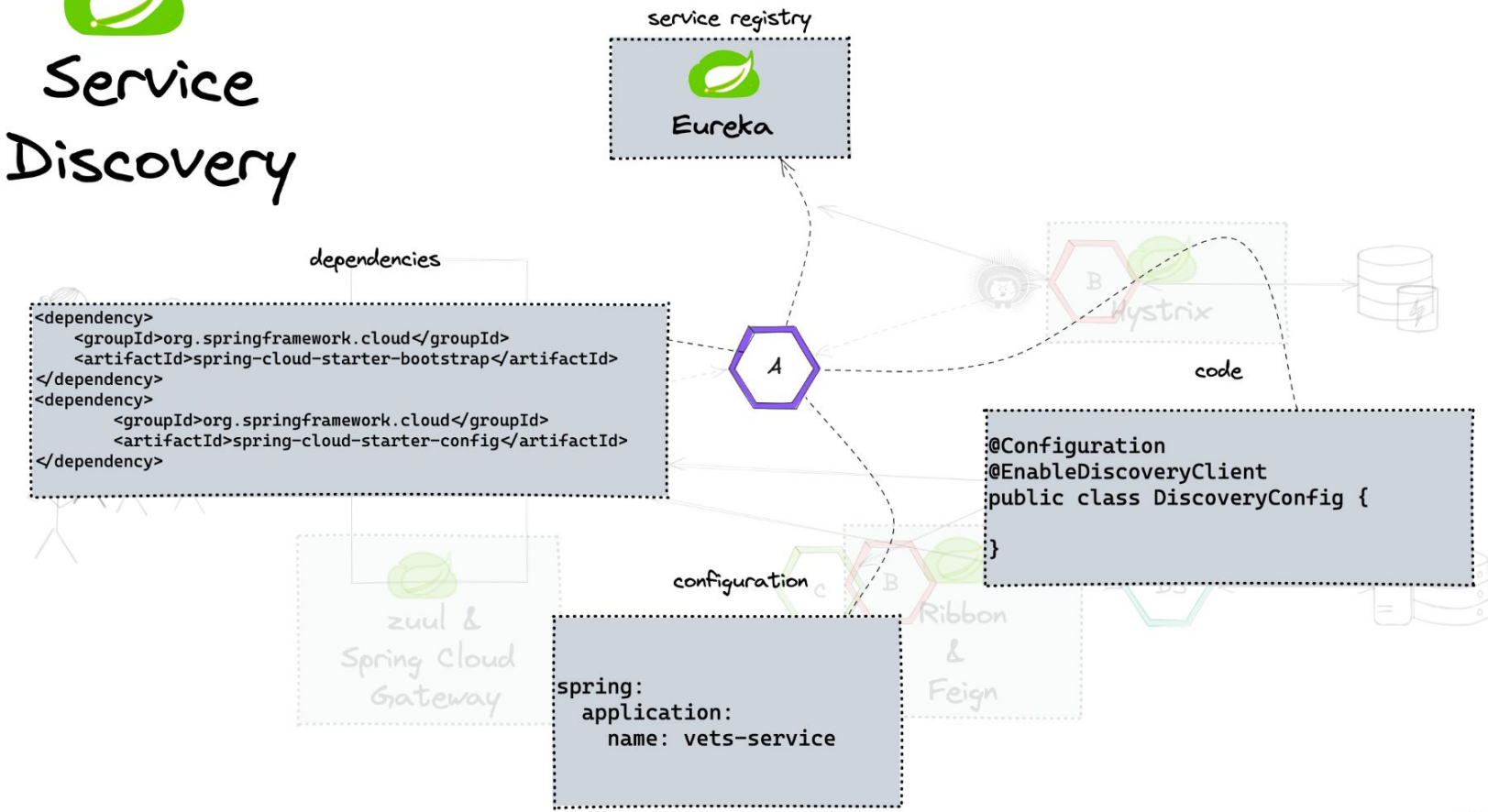



```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-bootstrap</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-config</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-netflix-eureka-server</artifactId>
</dependency>
```

```
@EnableEurekaServer
@SpringBootApplication
public class DiscoveryServerApplication {

        public static void main(String[] args) {
                SpringApplication.run(DiscoveryServerApplication.class, args);
        }
}
```

Service Discovery



Spring Cloud Gateway

service registry configuration observability

```
<dependency>
        <groupId>org.springframework.cloud</groupId>
        <artifactId>spring-cloud-starter-gateway</artifactId>
</dependency>
```

```
spring:
  cloud:
    gateway:
      routes:
      - id: novatec_route
        uri: http://www.nvtc.io
        predicates:
        - Path=/
```

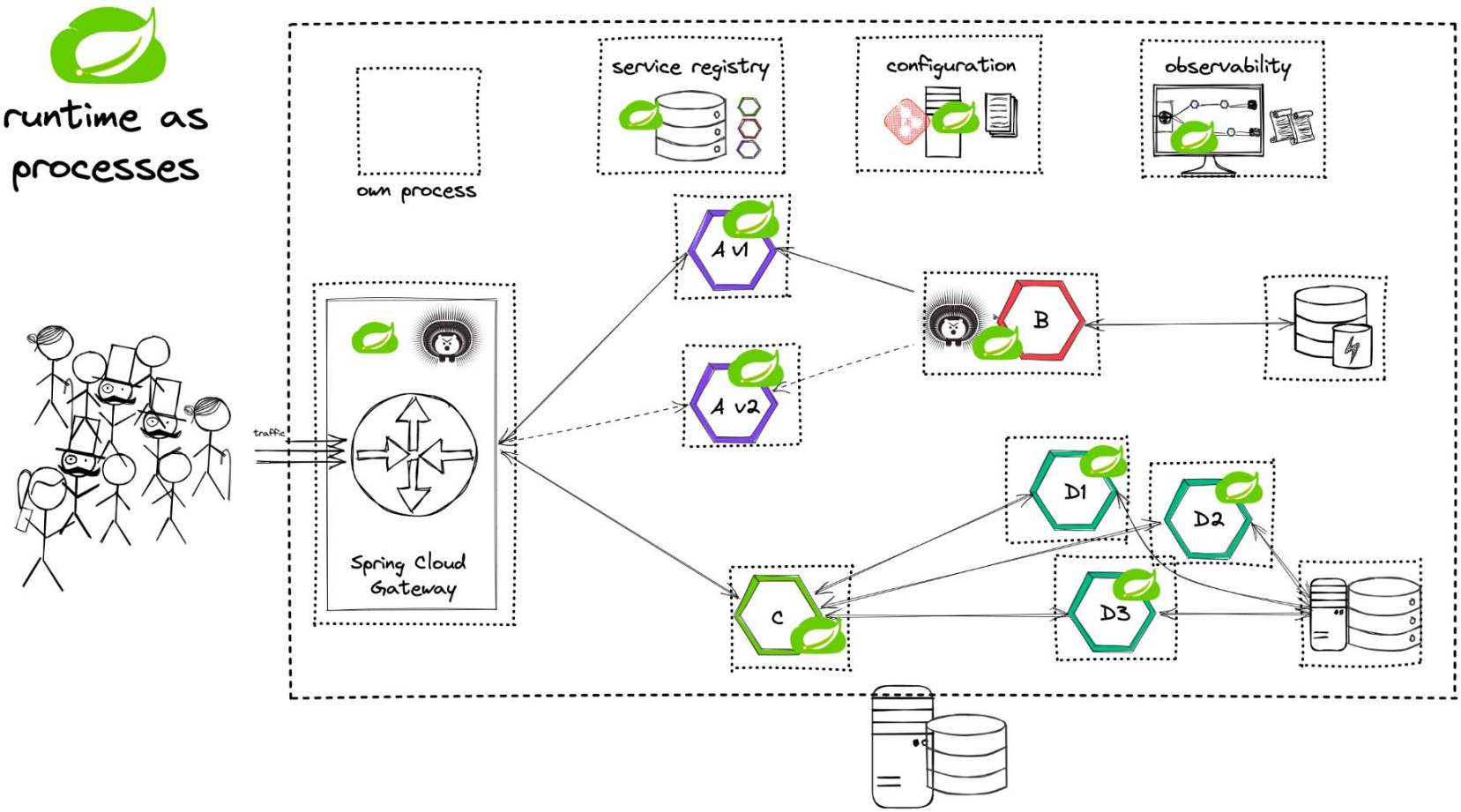
B Hystrix

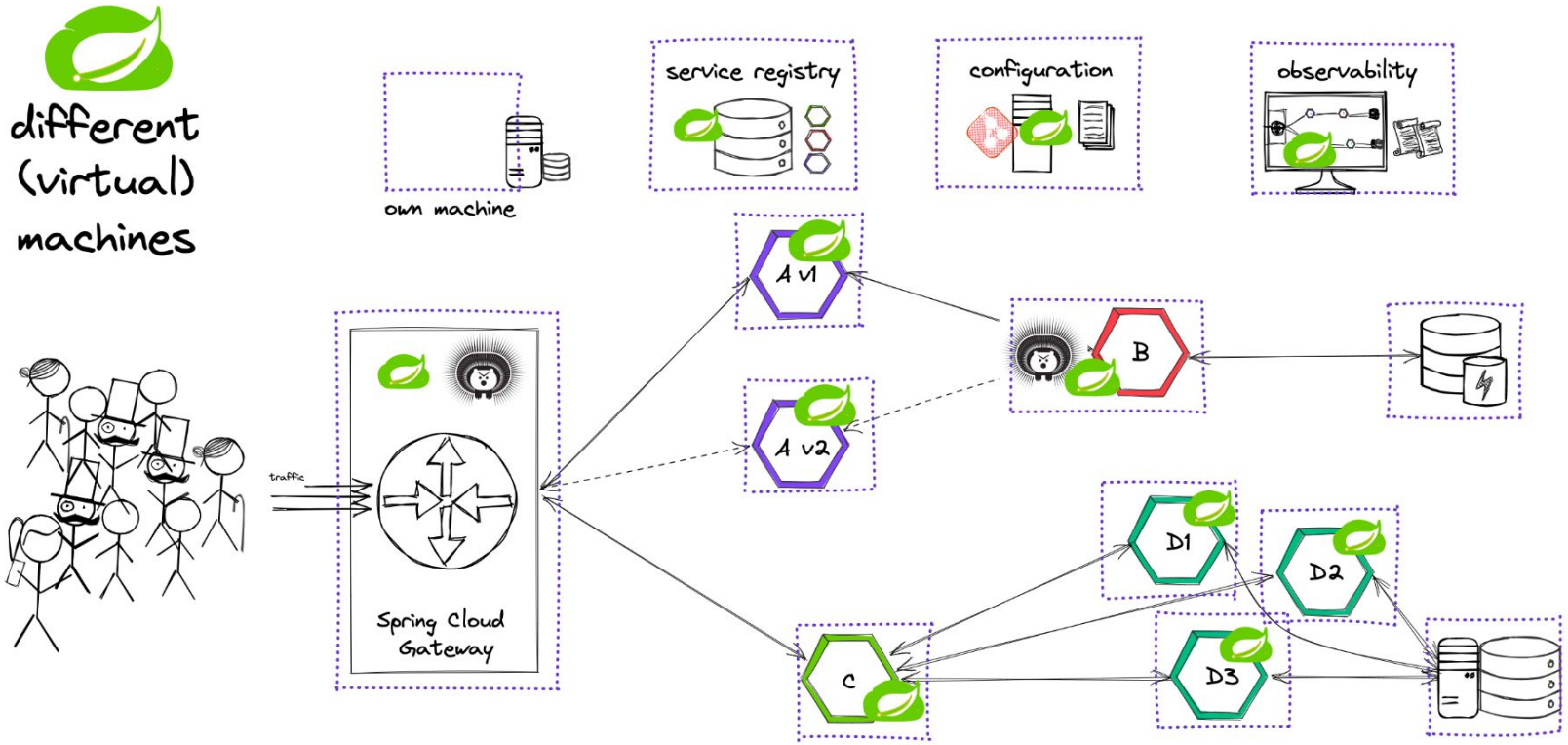
D2

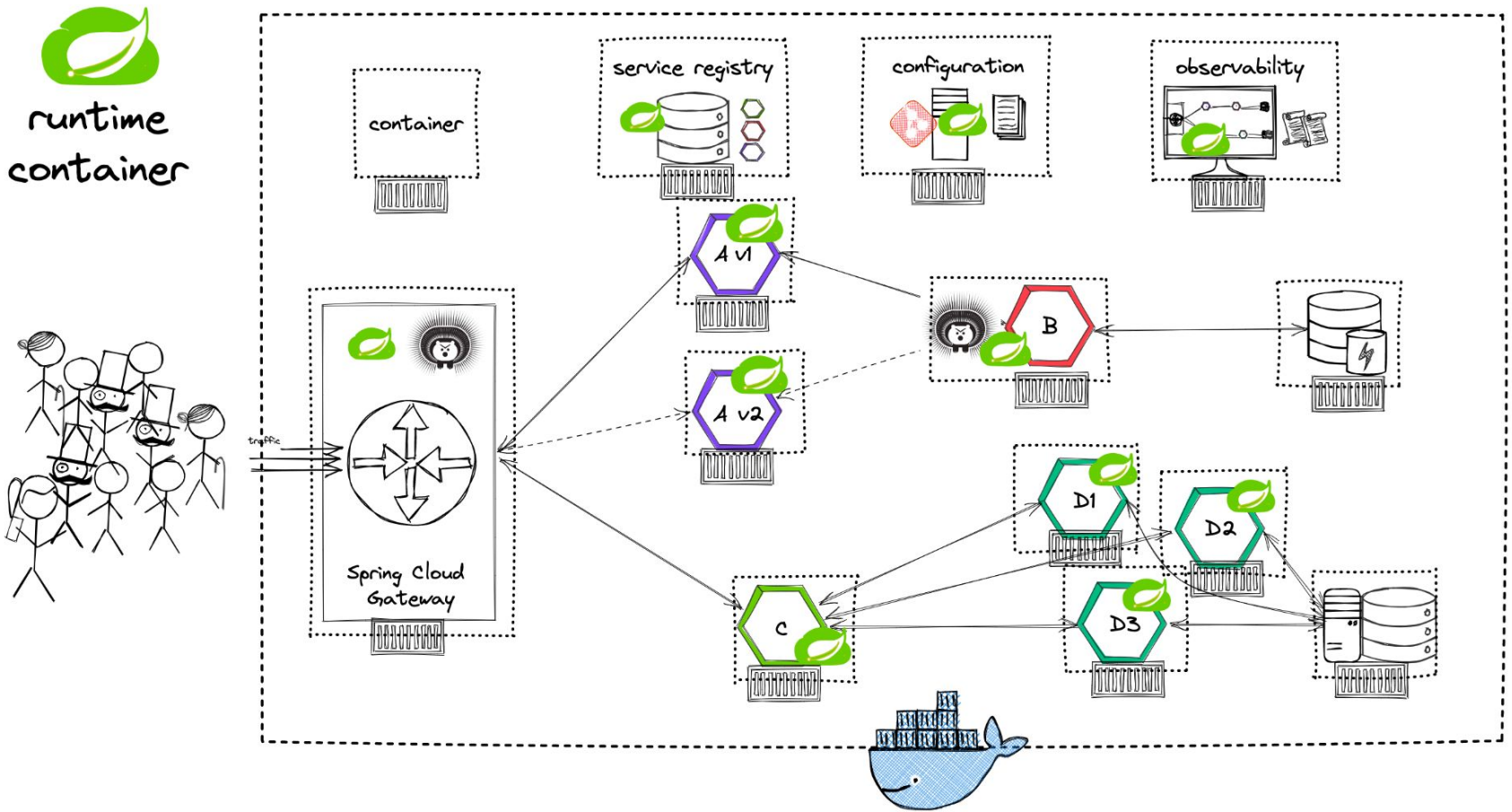
C B Ribbon & Feign

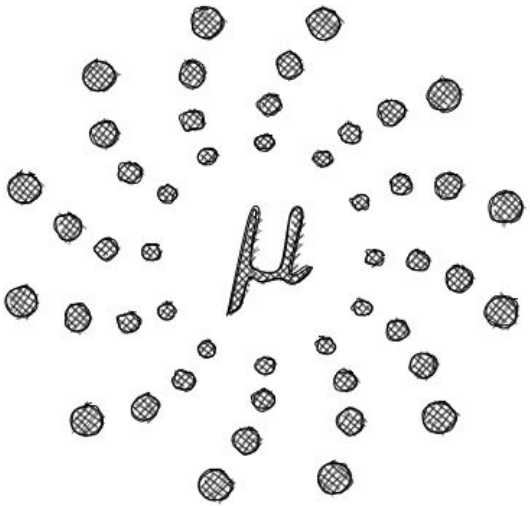
D3

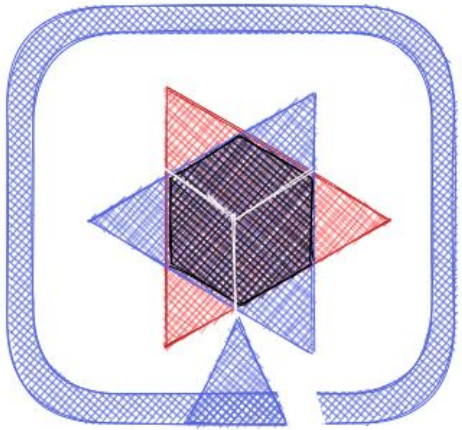
runtime as processes

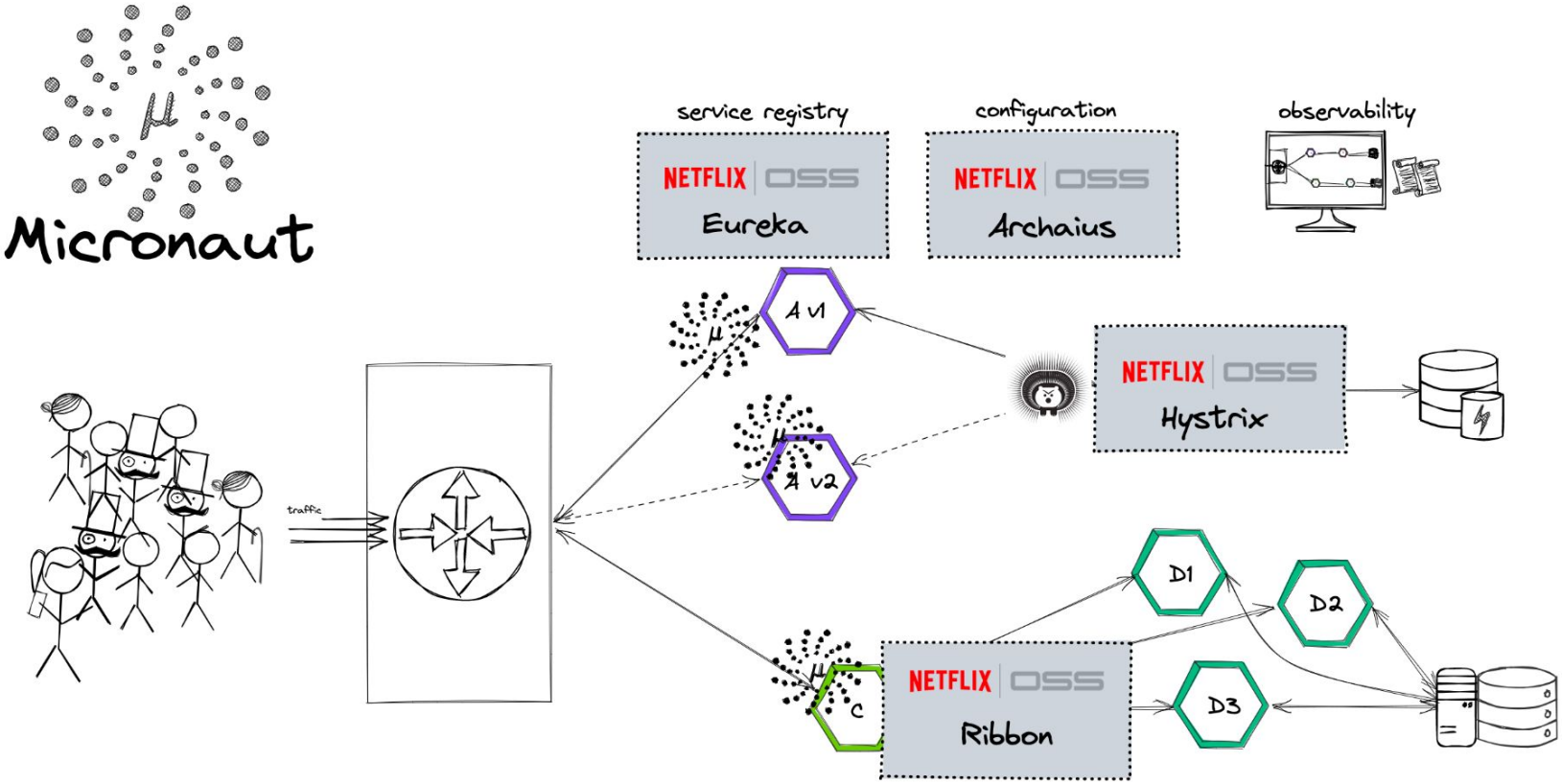


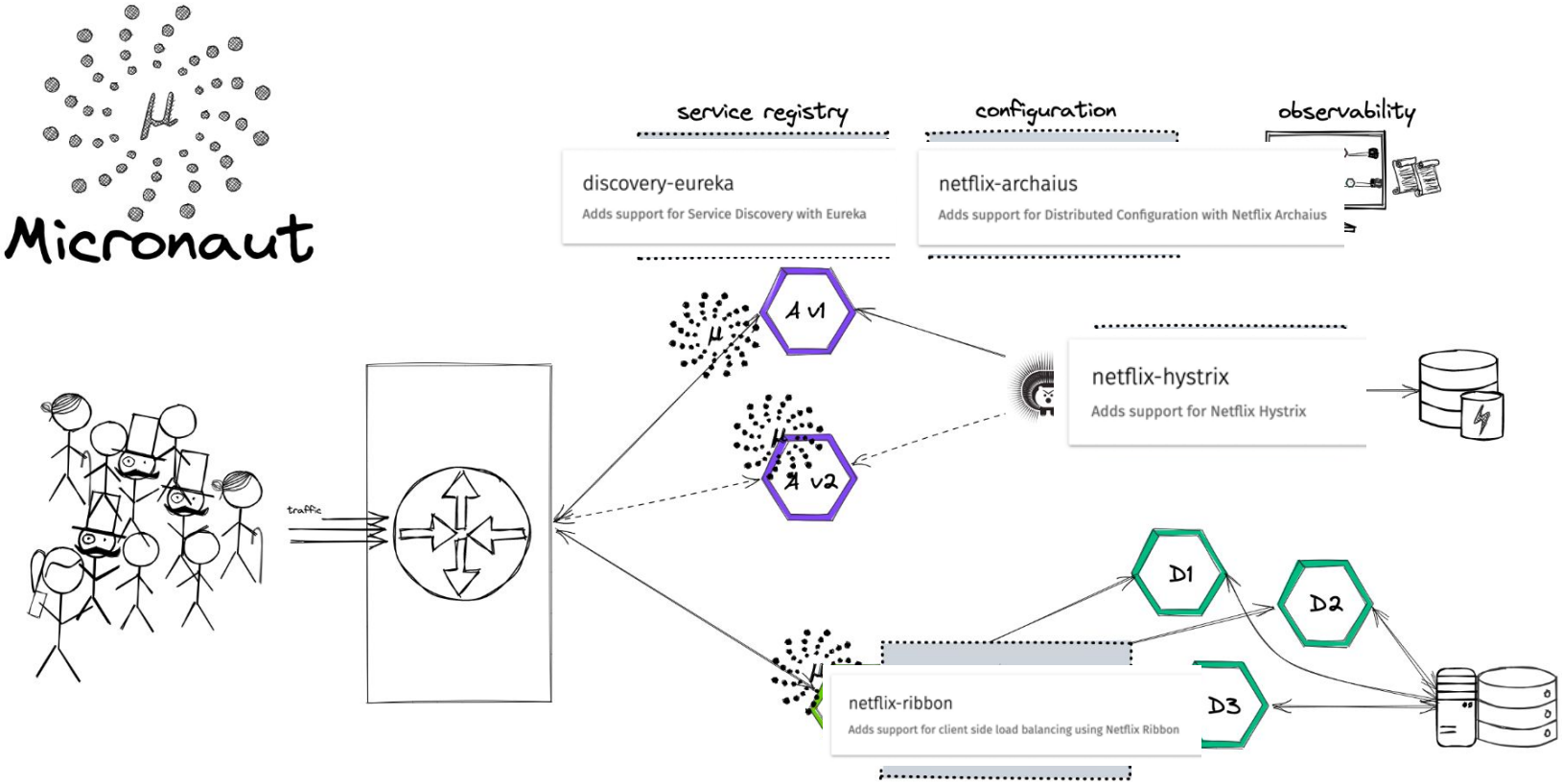

different (virtual) machines
own machine
service registry
configuration
observability
A v1
A v2
B
traffic
Spring Cloud Gateway
C
D1
D2
D3
made by @maebbles with @excalidraw

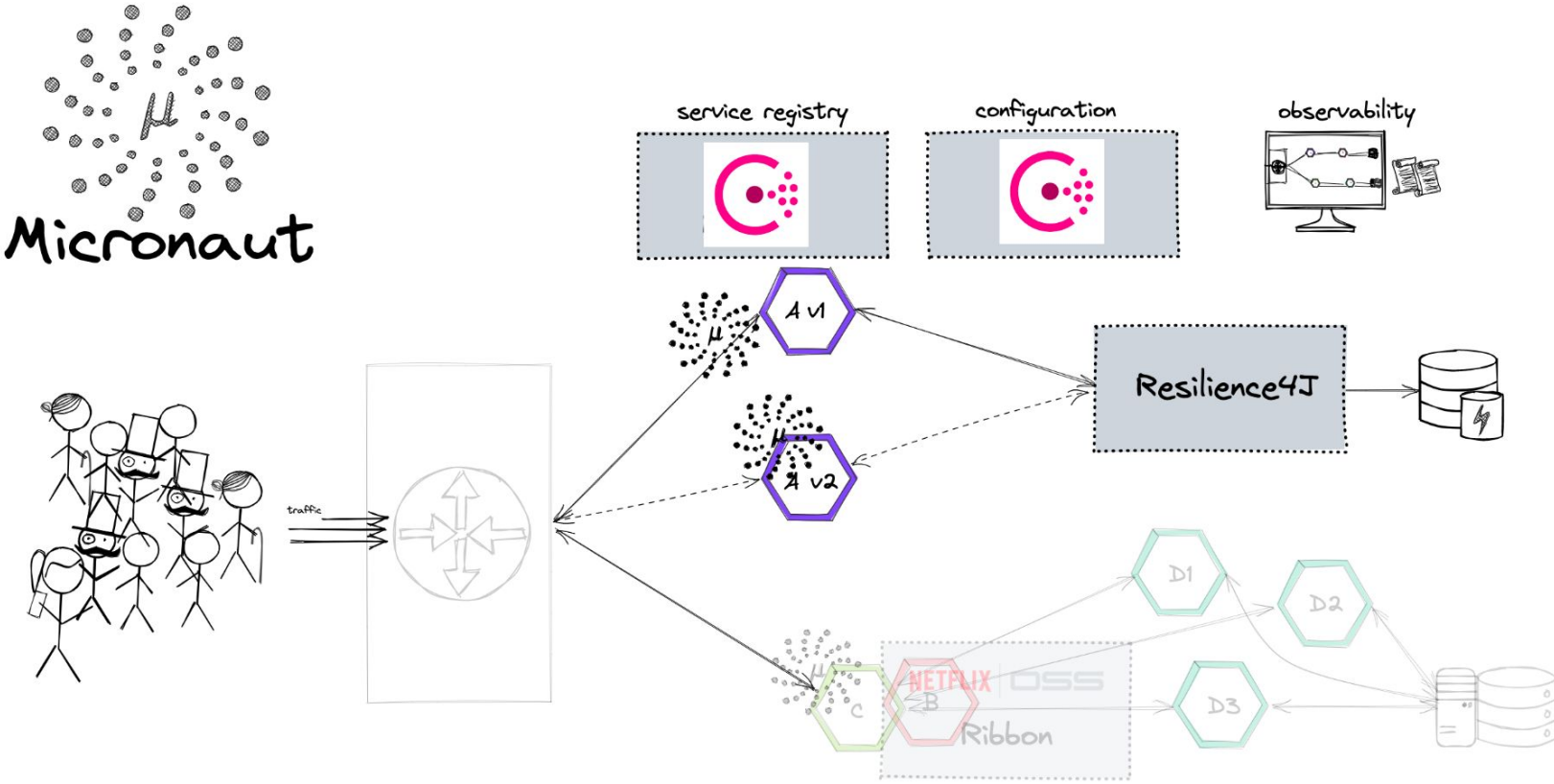

runtime container
container
service registry
configuration
observability
A v1
A v2
B
traffic
Spring Cloud Gateway
C
D1
D2
D3

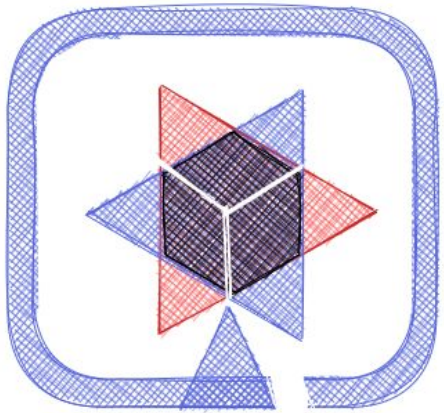






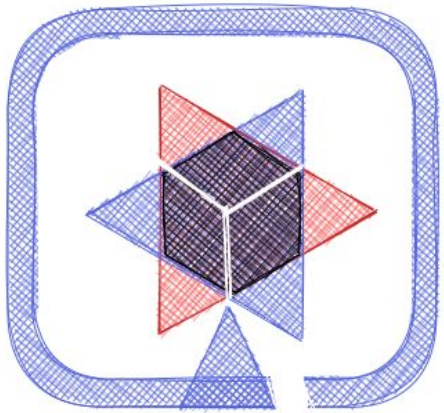






&

SMALLRYE



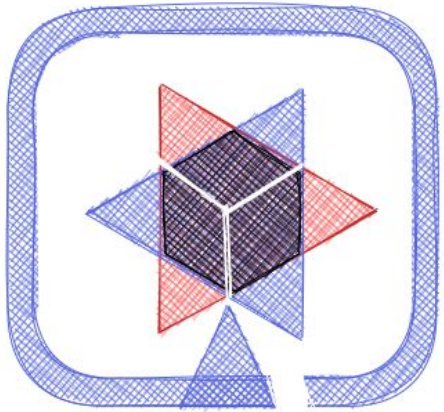
WildFly

Open Liberty

&

SMALLRYE

MICROPROFILE™



Health

Config

Service Discovery

& SMALLRYE

Fault Tolerance

Load Balancing

Metrics

Characteristics:

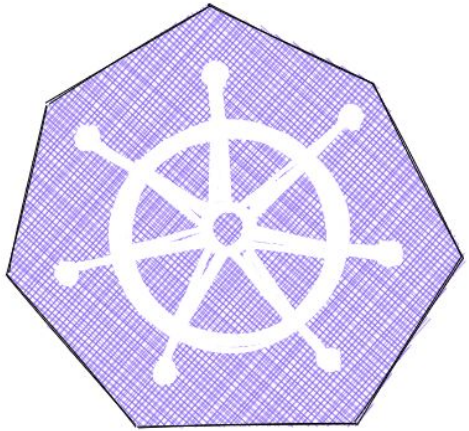
- on application level

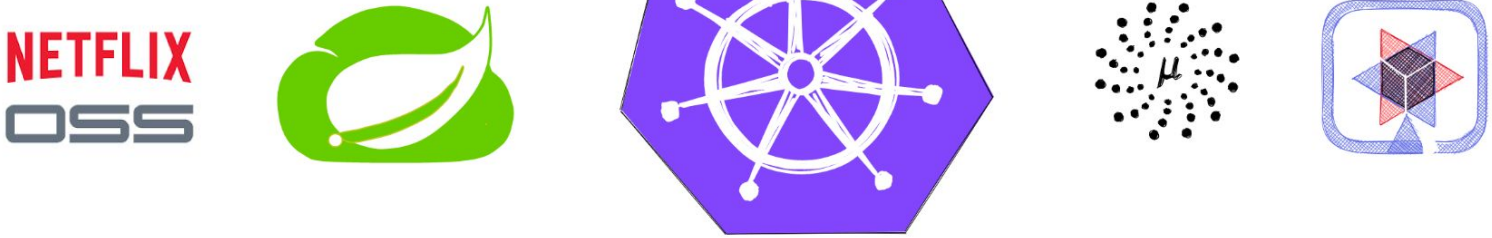
- mostly combination of dependency,
  code & configuration property

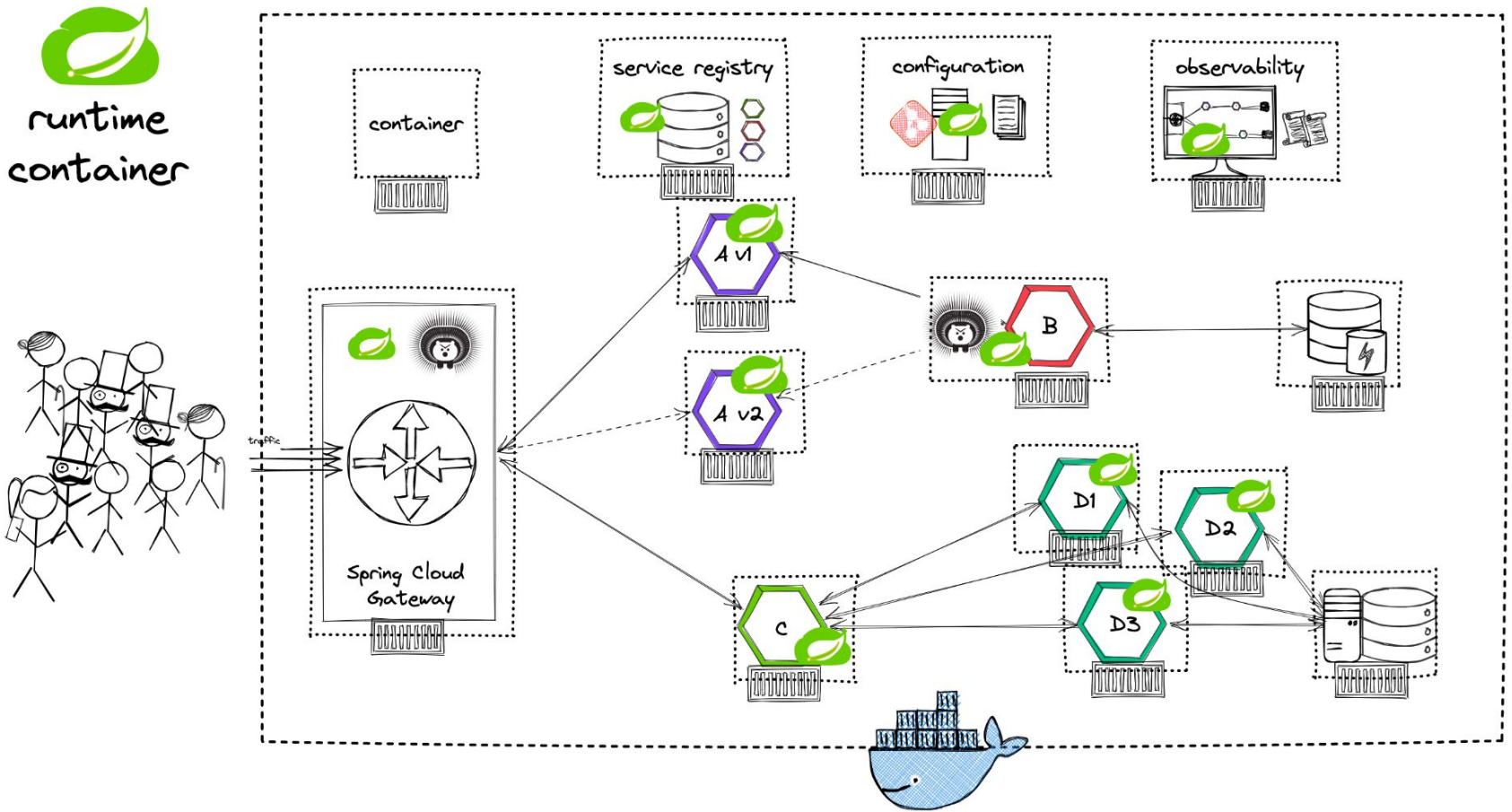
- independent of underlying framework

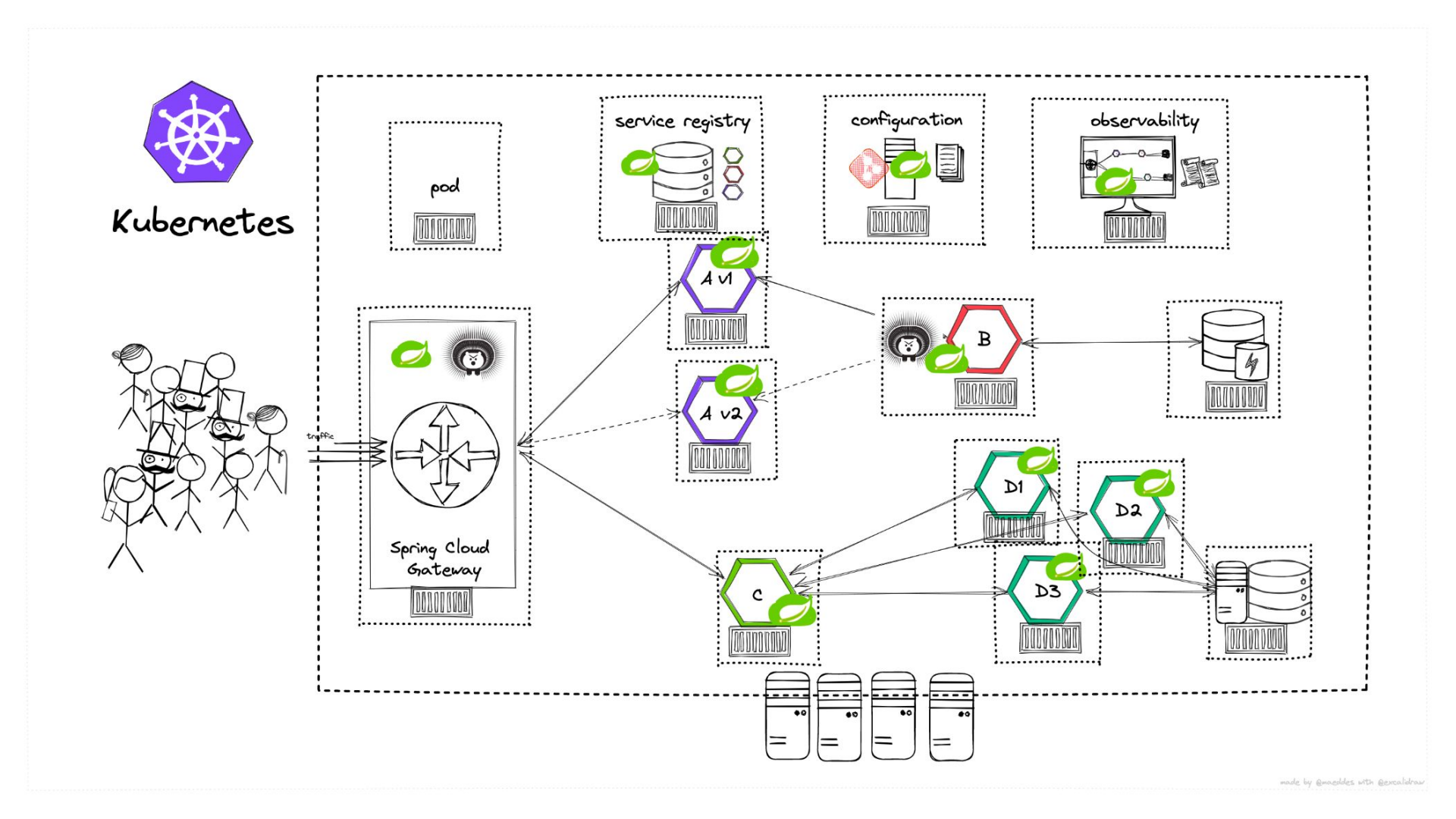
- lightweight, only the required functionalities to be bound
  -> right tool for the right job

Kubernetes

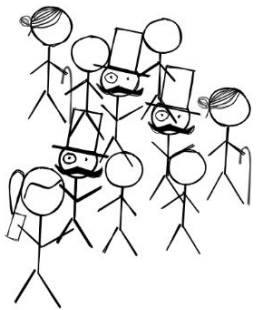


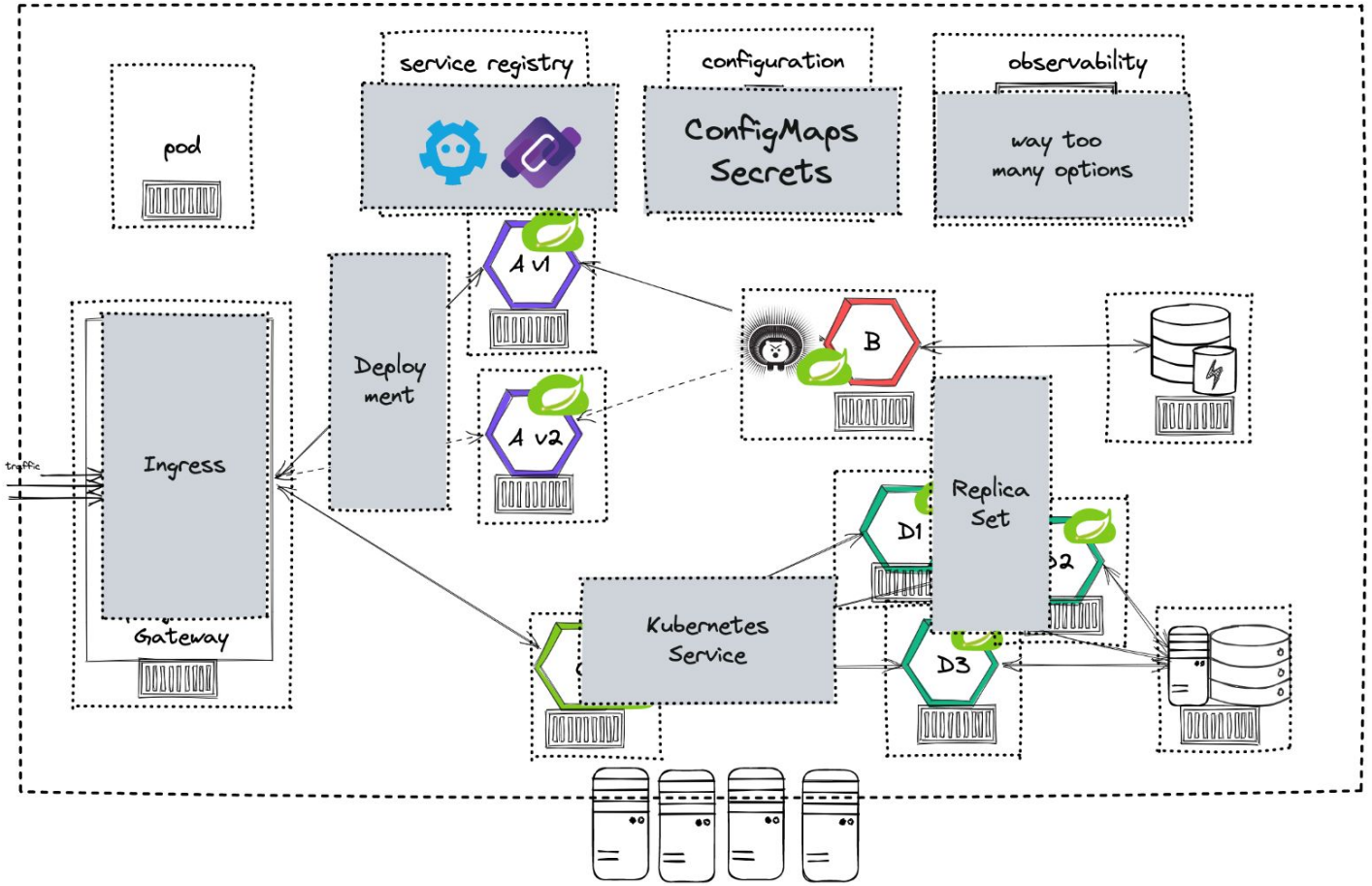


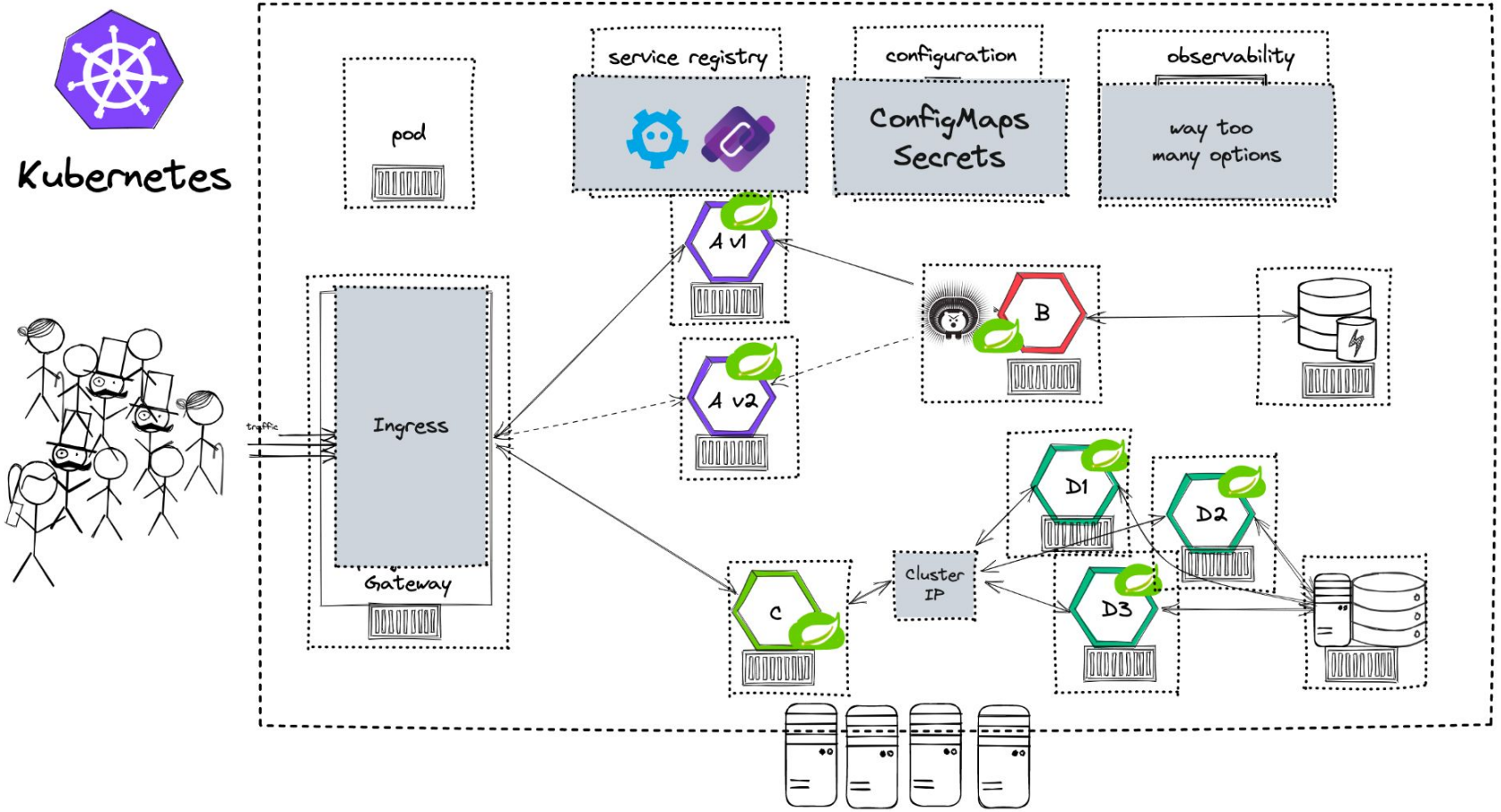

runtime container
container
service registry
configuration
observability
A v1
A v2
B
traffic
Spring Cloud Gateway
C
D1
D2
D3


Kubernetes
pod
service registry
configuration
observability
A v1
A v2
B
traffic
Spring Cloud Gateway
C
D1
D2
D3

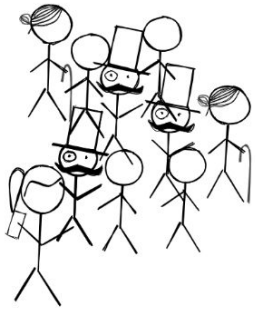
Kubernetes

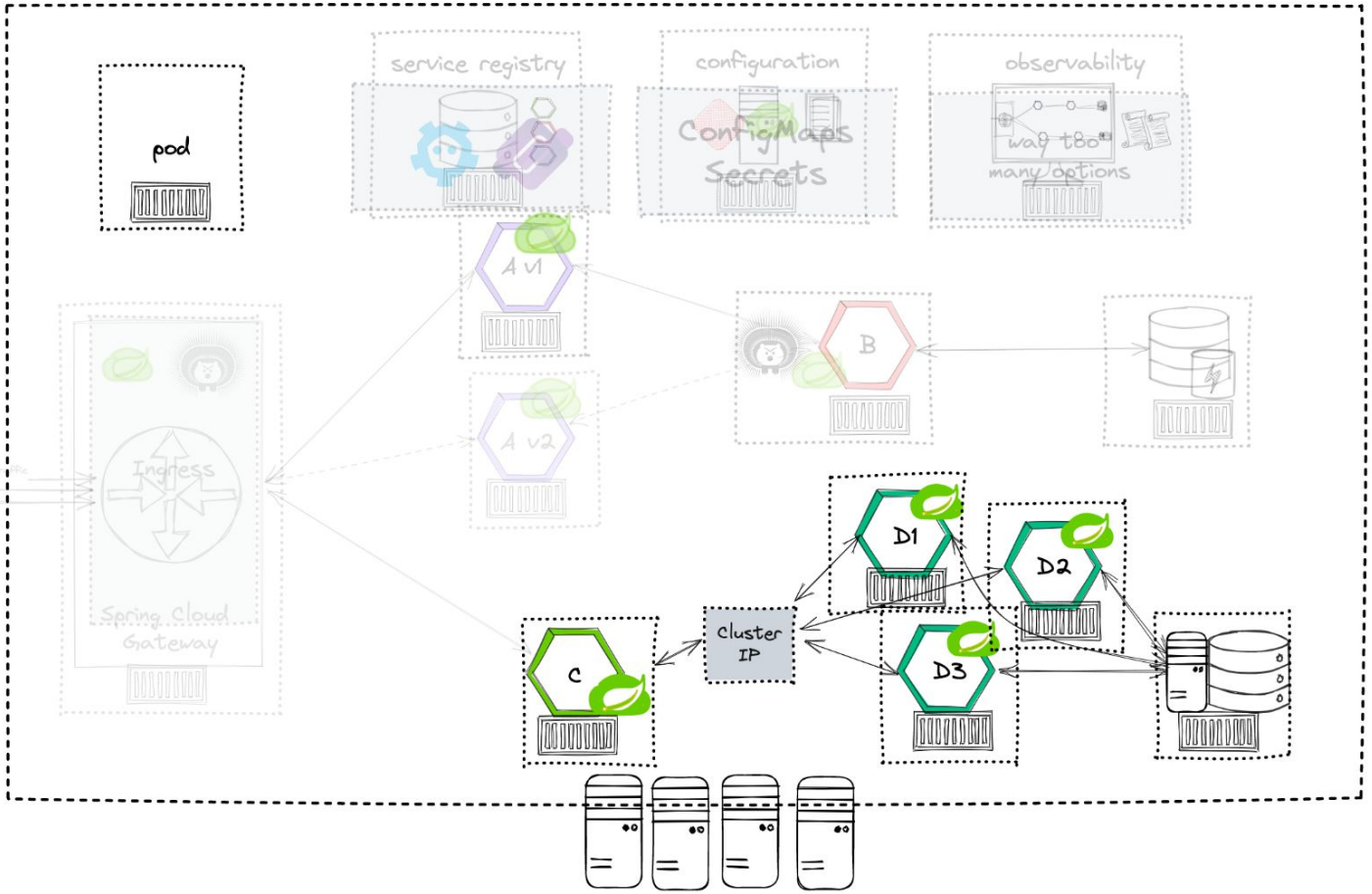




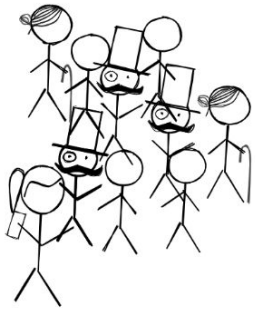

Kubernetes
pod
service registry
configuration
ConfigMaps
Secrets
observability
way too
many options
A v1
A v2
B
Ingress
Gateway
traffic
C
Cluster
IP
D1
D2
D3

Kubernetes Service





Kubernetes Service



pod

service registry

configuration

ConfigMaps
Secrets

observability

way too
many options

A v1

A v2

B

```
apiVersion: v1
kind: Service
metadata:
  labels:
    app: visits-service
  name: visits-service
spec:
  ports:
  - name: "http"
    port: 8080
    targetPort: 8080
  selector:
    app: visits-service
```

```
replicas: 3
..
labels:
    app: visits-service
```

D1

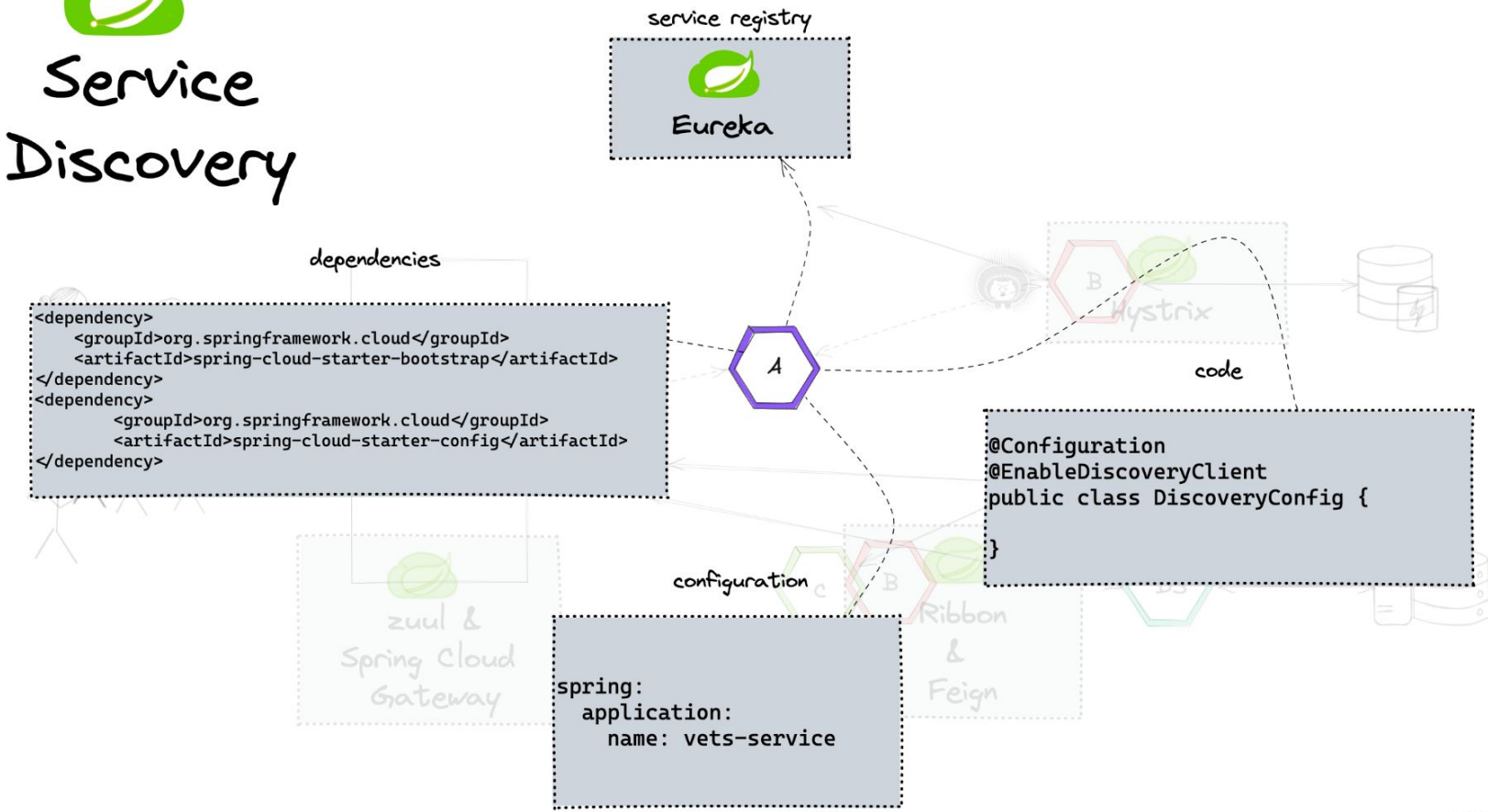
D2

D3

C

Cluster
IP

Service Discovery



Service Discovery

service registry

Eureka

service registry

dependencies

```
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-bootstrap</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.cloud</groupId>
    <artifactId>spring-cloud-starter-kubernetes</artifactId>
</dependency>
```

A

code

```
@Configuration
@EnableDiscoveryClient
public class DiscoveryConfig {

}
```

configuration

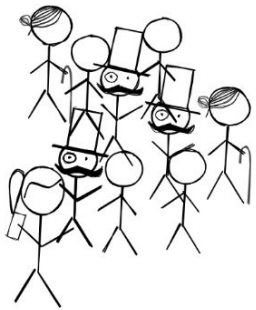
```
spring:
  cloud:
    service:
      discovery:
        kubernetes: true
```

zuul & Spring Cloud Gateway

Ribbon & Feign

Hystrix

Kubernetes Config



pod

service registry

configuration

ConfigMaps
Secrets

observability

way too
many options

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: client-service
data:
  application.properties: |-
    bean.message=Hello
```

```
@Configuration
@ConfigurationProperties(prefix = "bean")
public class ClientConfig {

    private String message;
    // getters and setters
}
```

Spring Cloud
Gateway

B

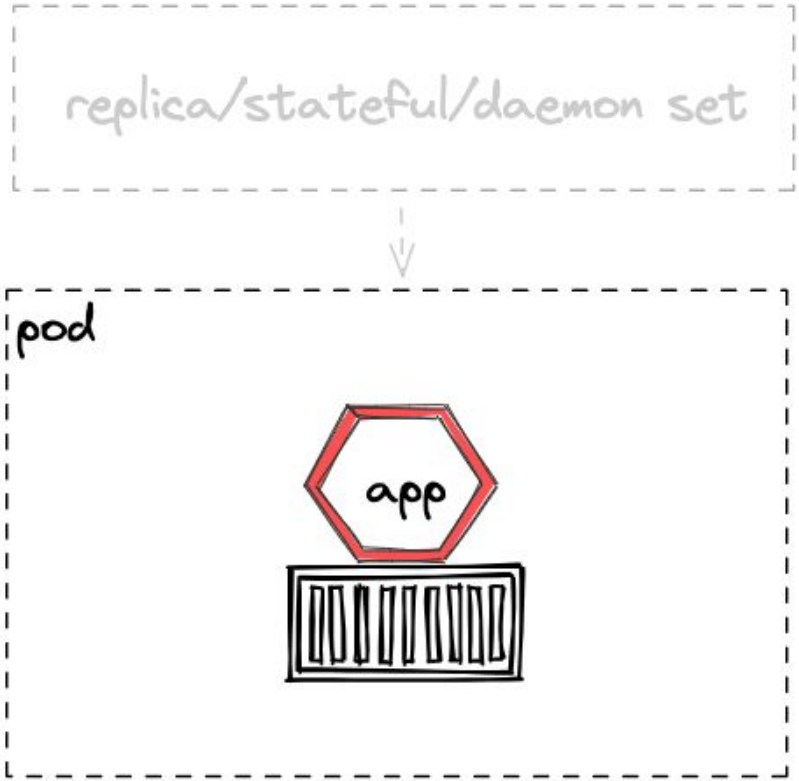
C

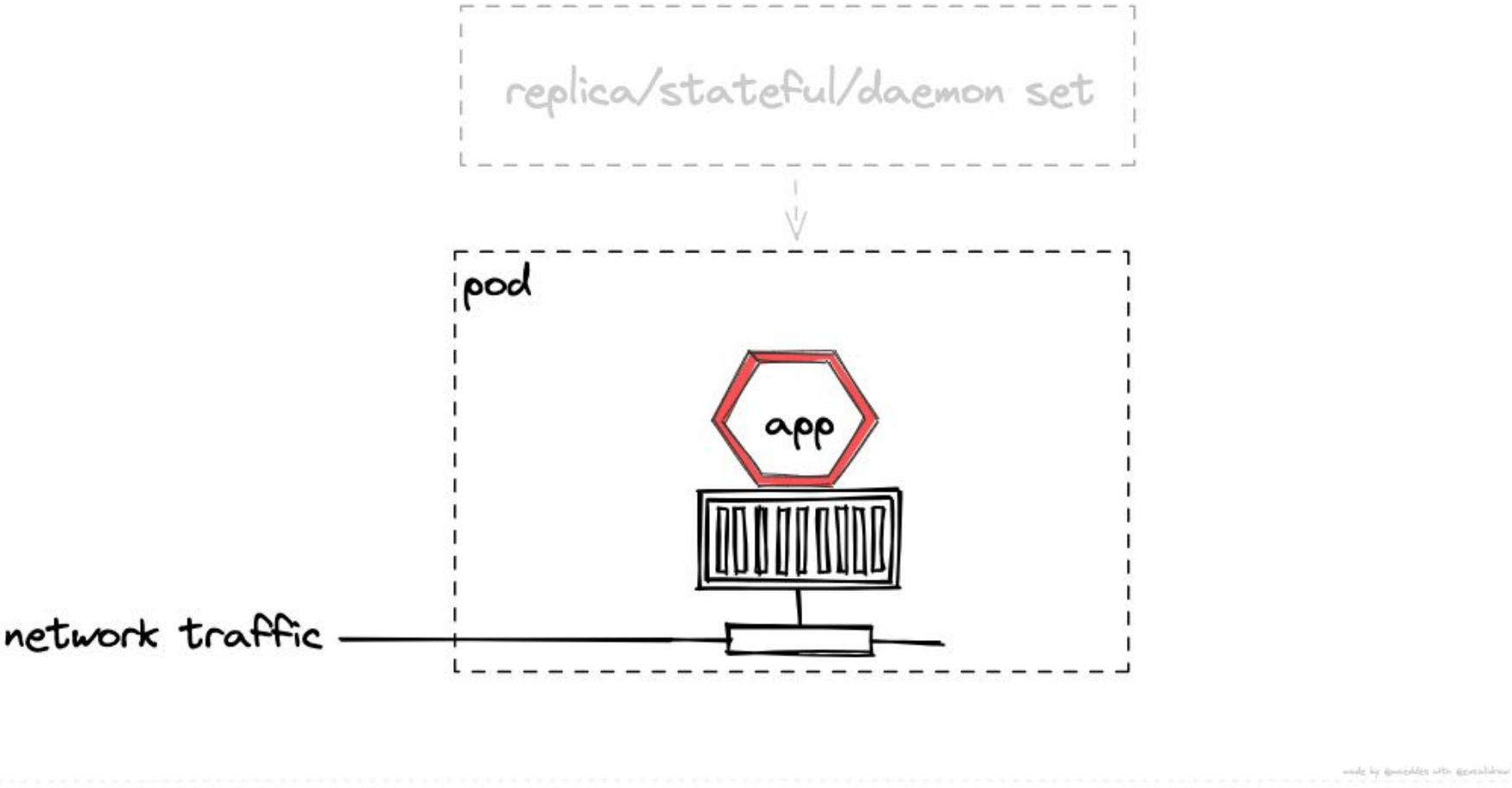
Characteristics:

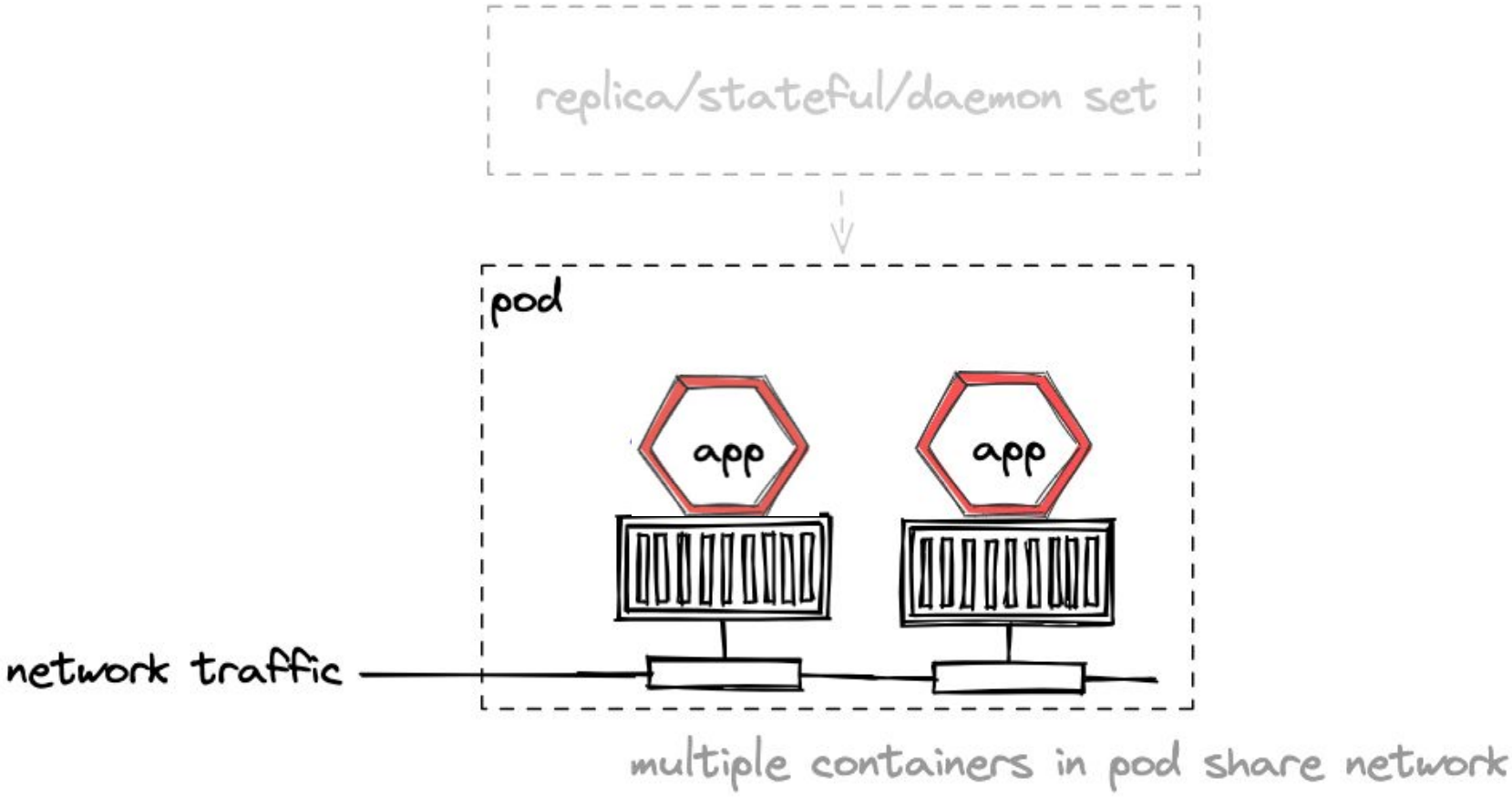
- Kubernetes provides OOTB functionality for
  scaling, load balancing, service registry,
  external configuration (and more) independent of framework
- Not equivalent in resilience patterns
  (configurable routing, network traffic awareness, circuit breaking ..)

- Spring Cloud adapts for usage with Kubernetes without code changes
  - Eureka-like behaviour with Kubernetes Service Registry
  - Config Server-like behaviour with ConfigMaps/Secrets

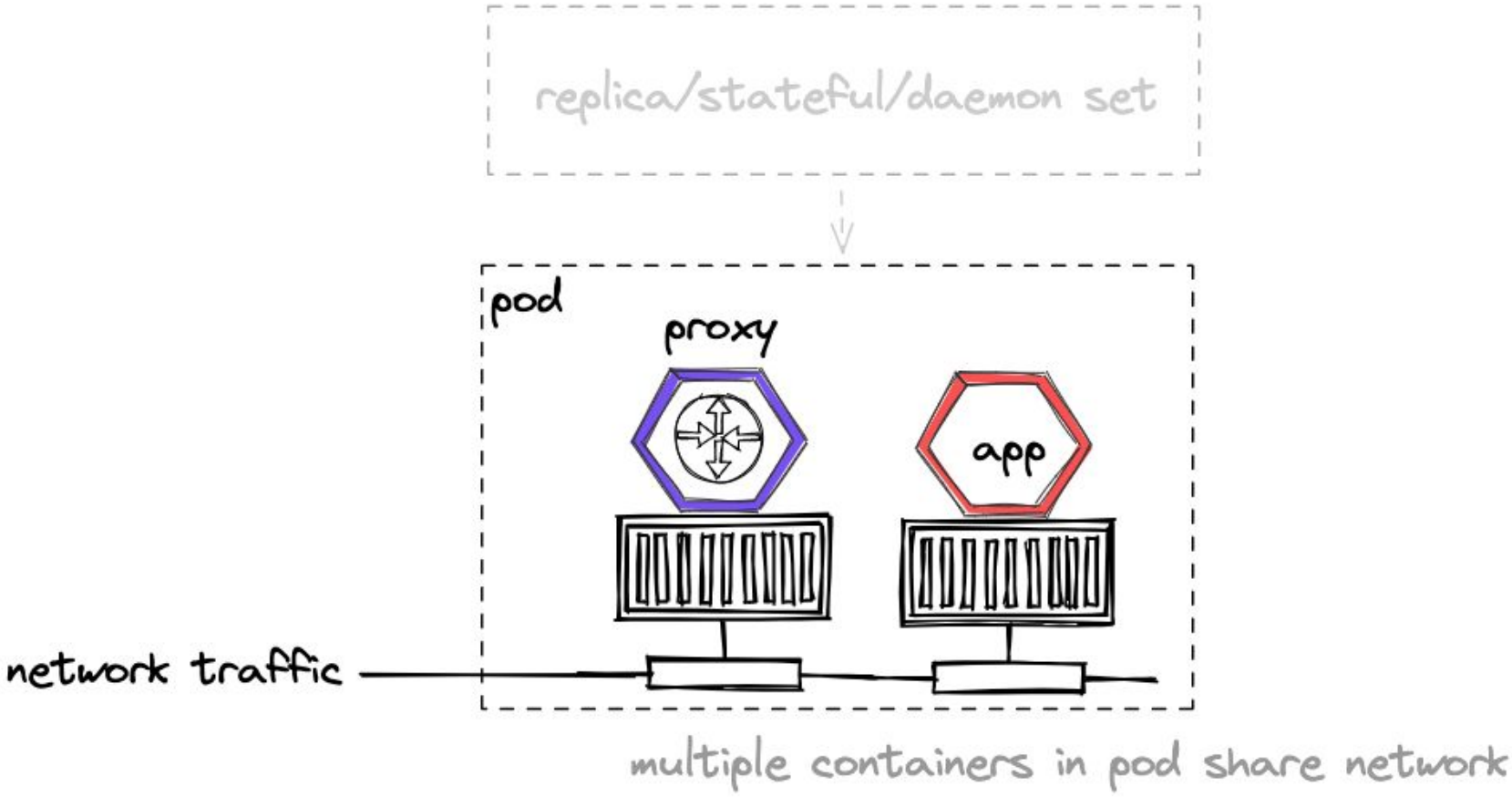
Service Mesh


replica/stateful/daemon set
pod
app

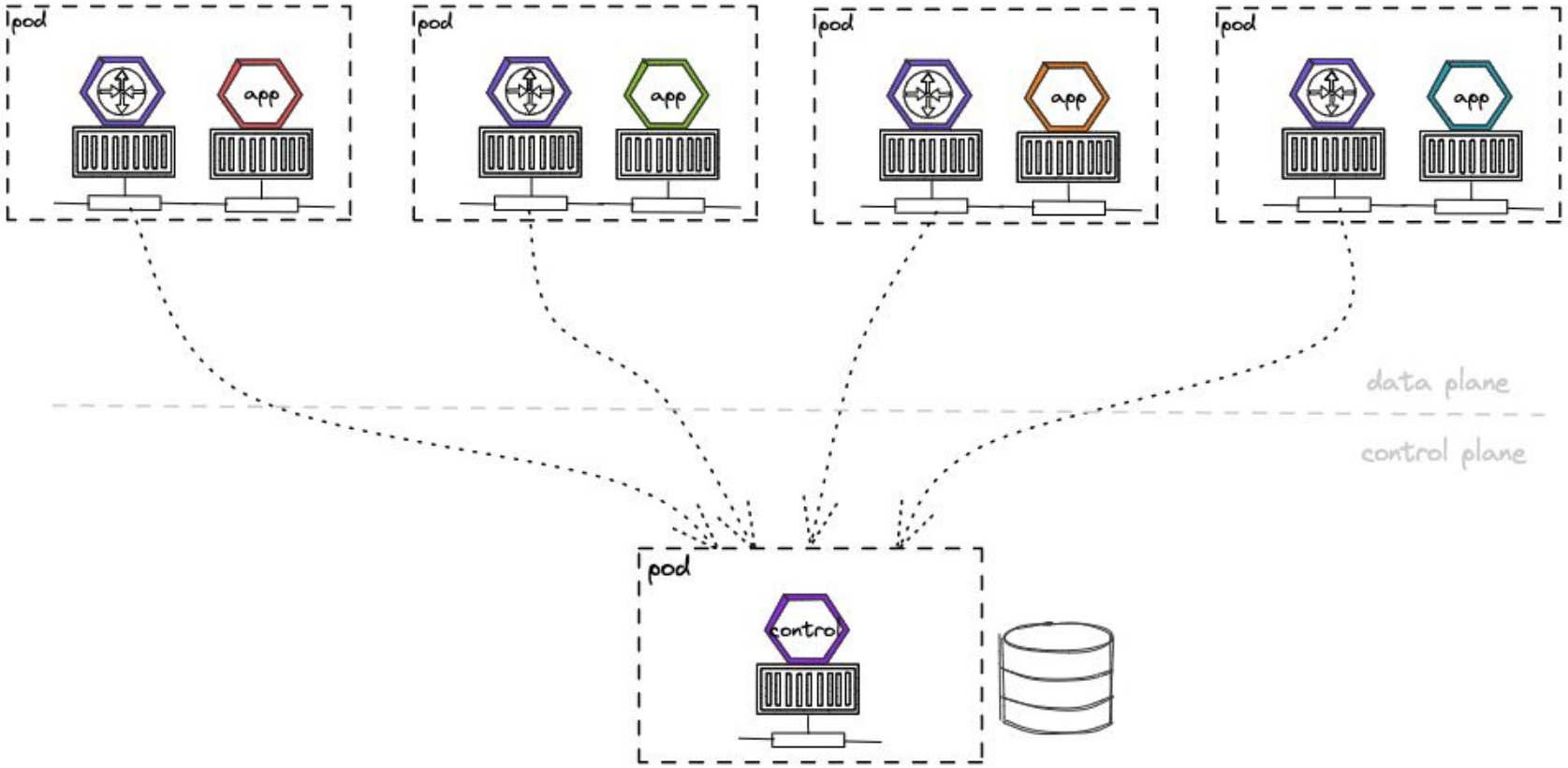

replica/stateful/daemon set
pod
app
network traffic

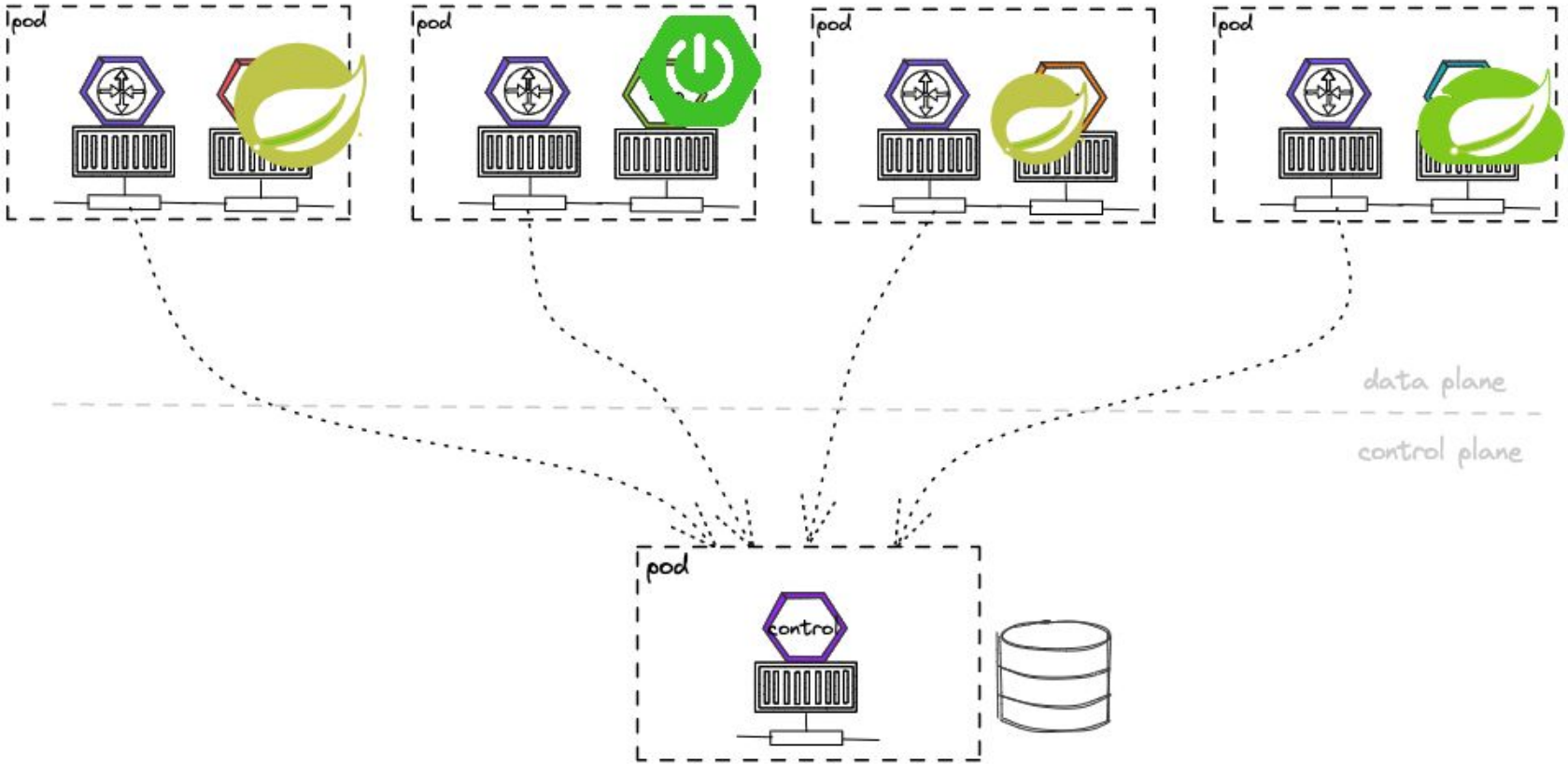


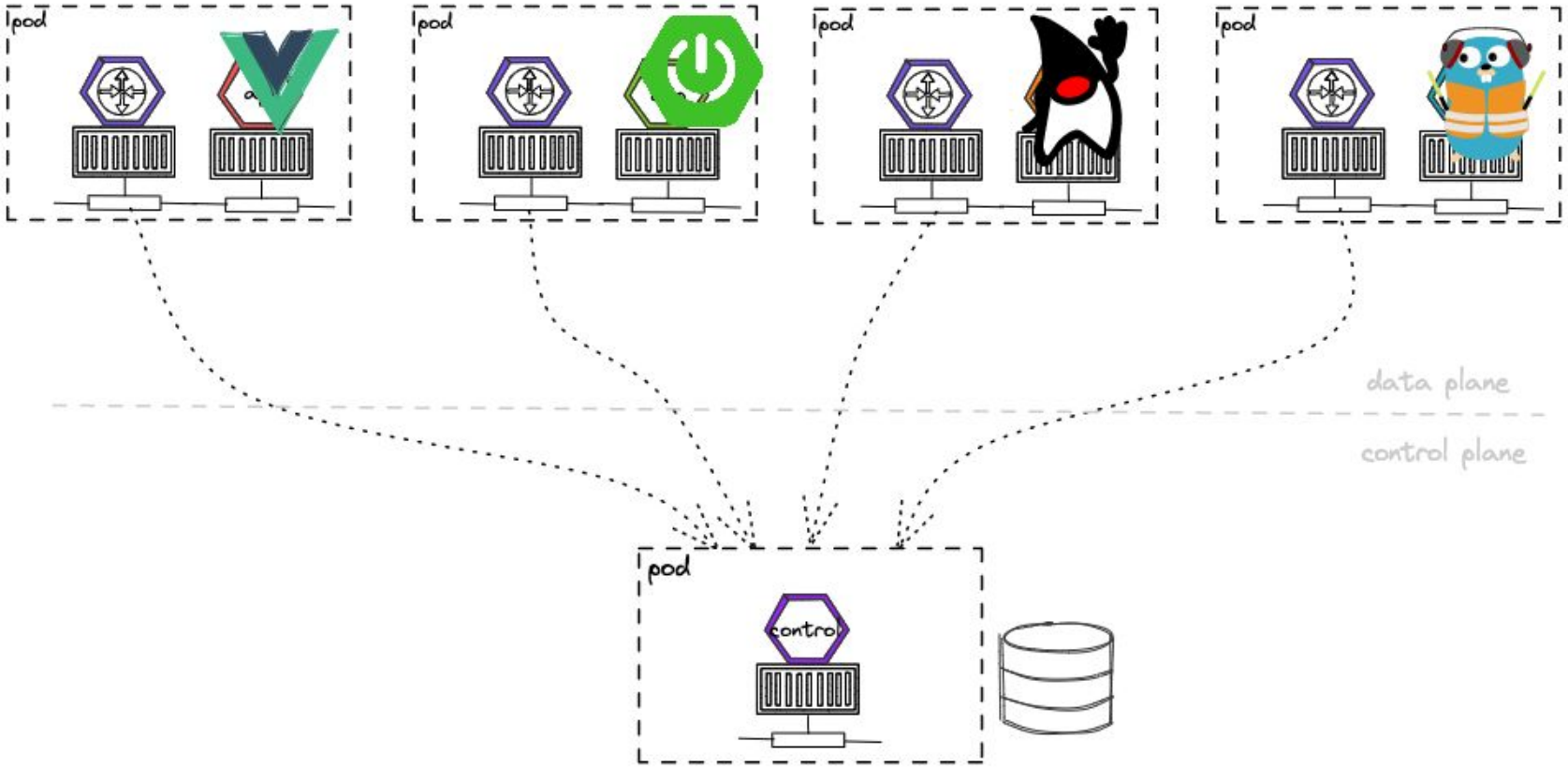
multiple containers in pod share network

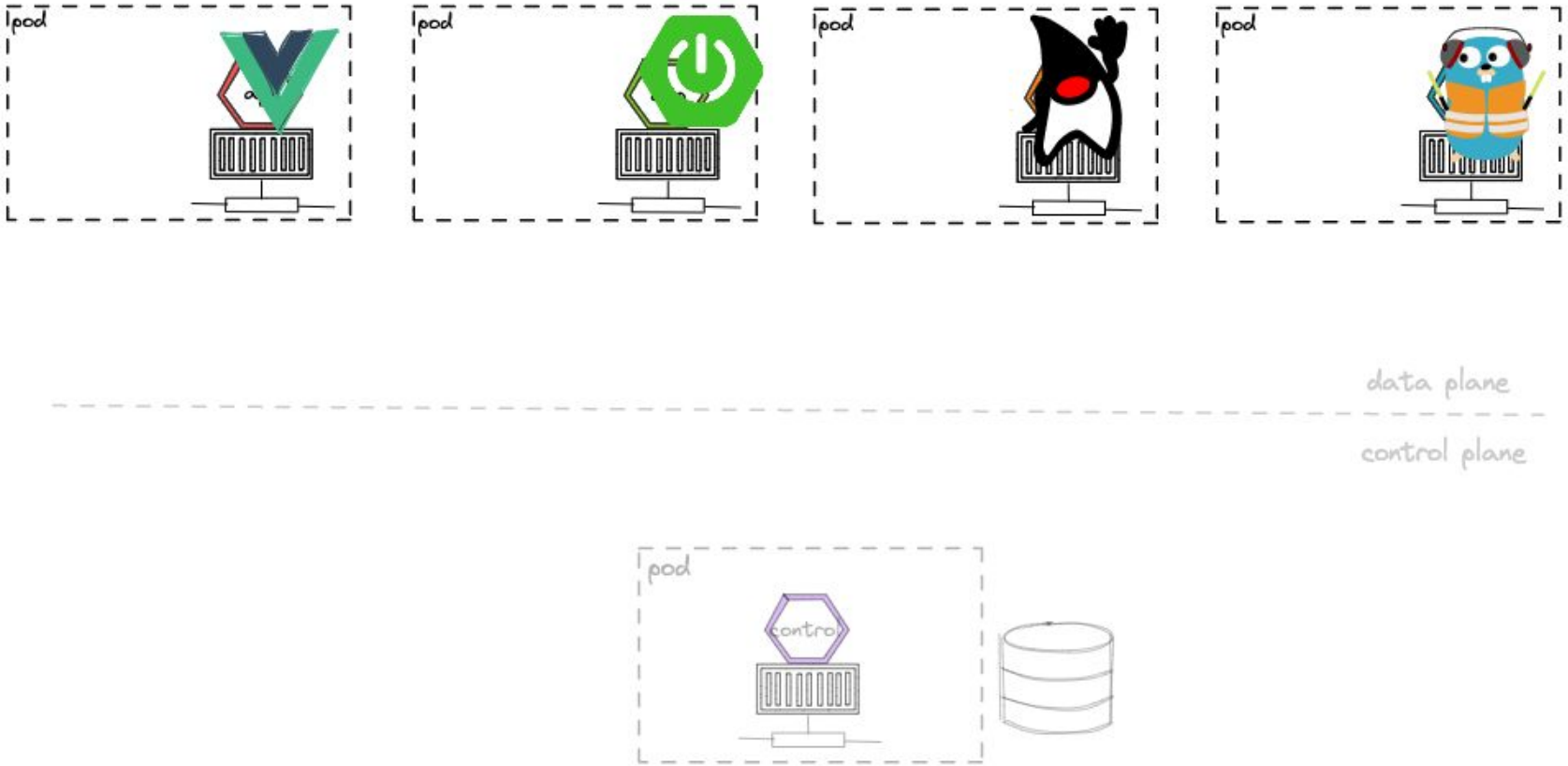


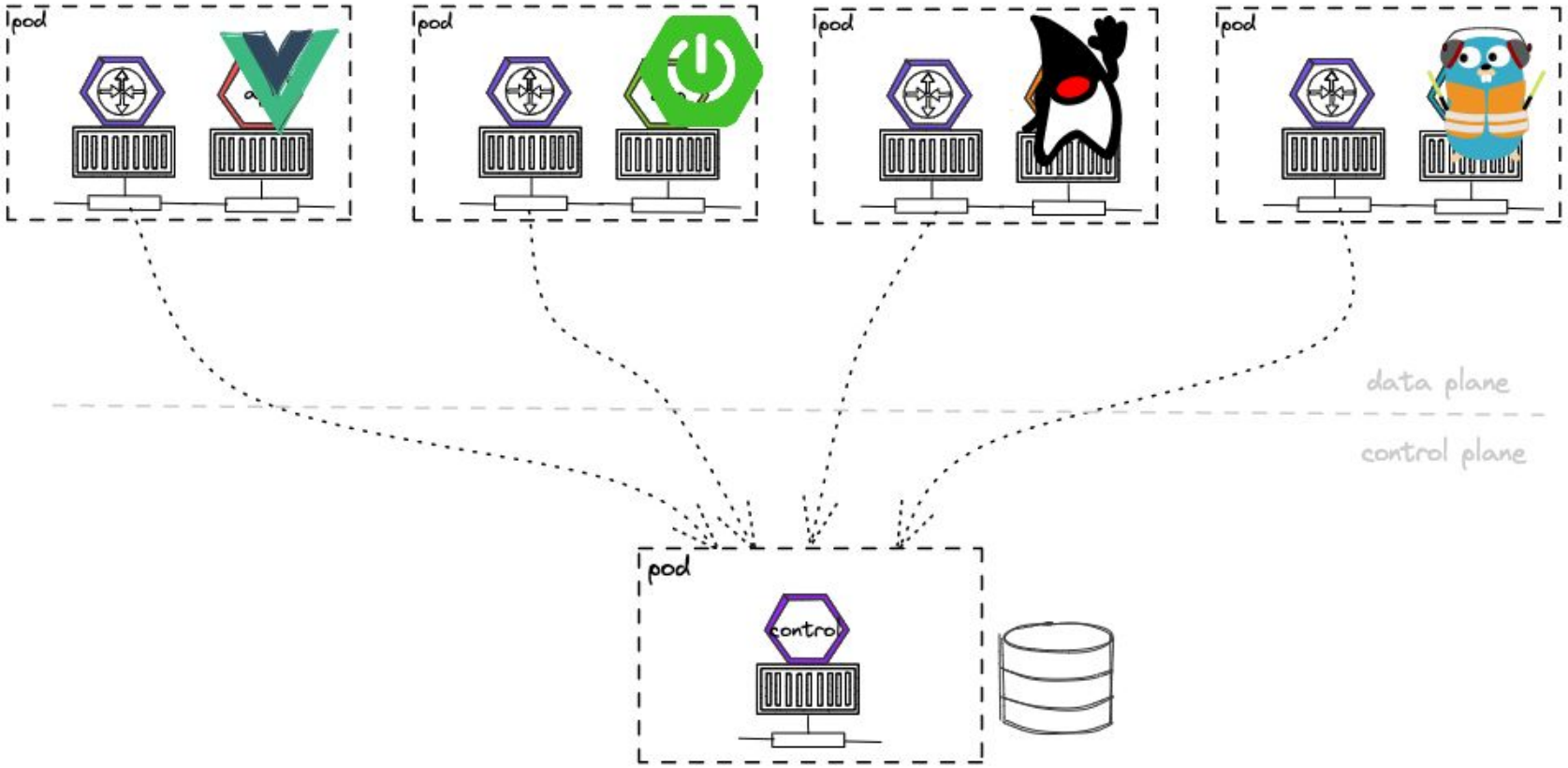
multiple containers in pod share network

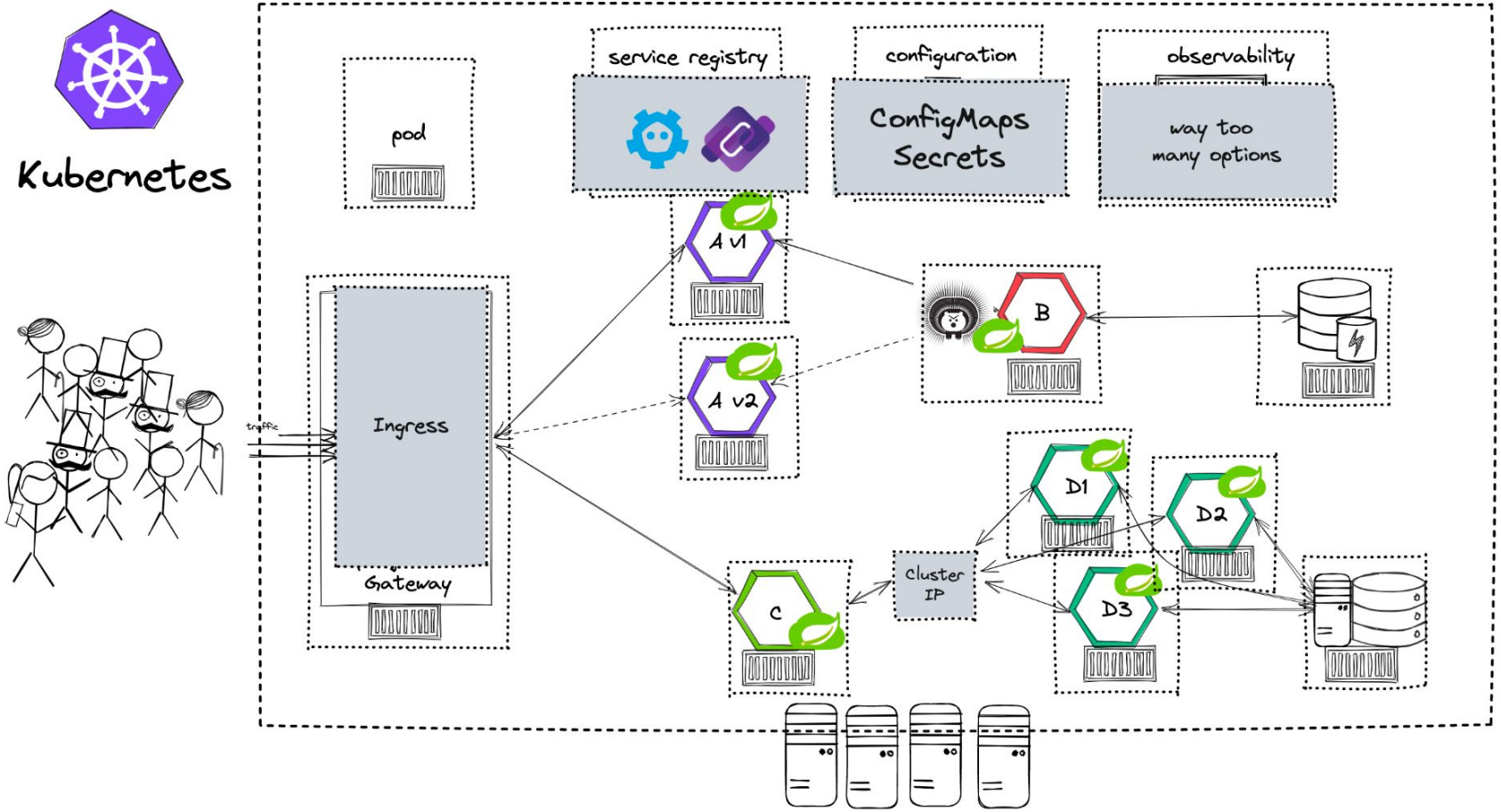

pod
app
pod
app
pod
app
pod
app
data plane
control plane
pod
control

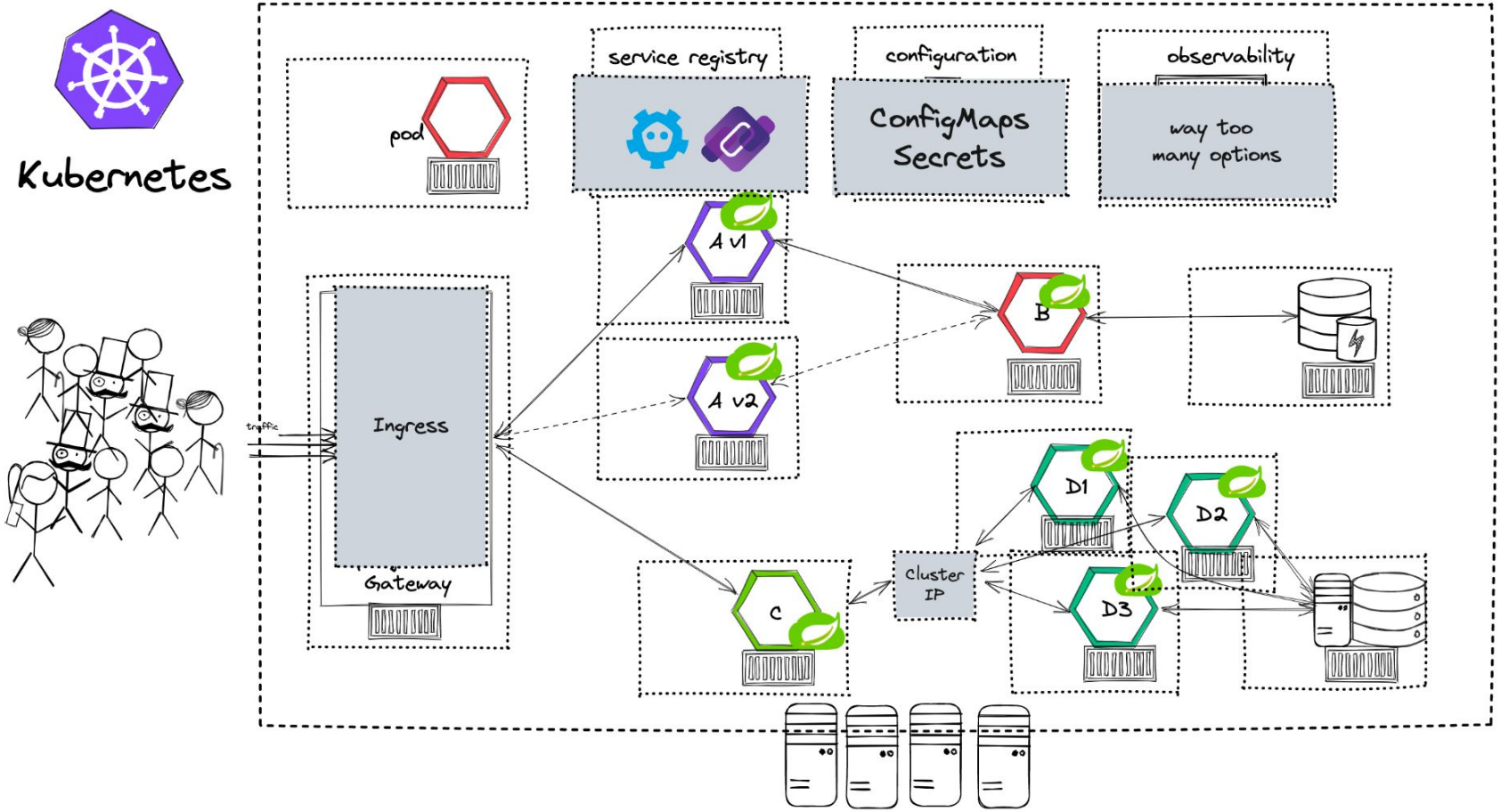

pod
pod
pod
pod
data plane
control plane
pod
control


pod
pod
pod
pod
data plane
control plane
pod
control

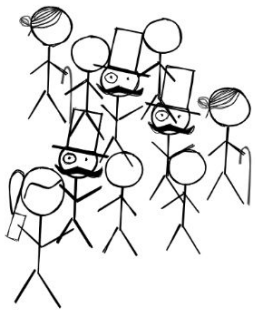

pod
pod
pod
pod
data plane
control plane
pod
contro

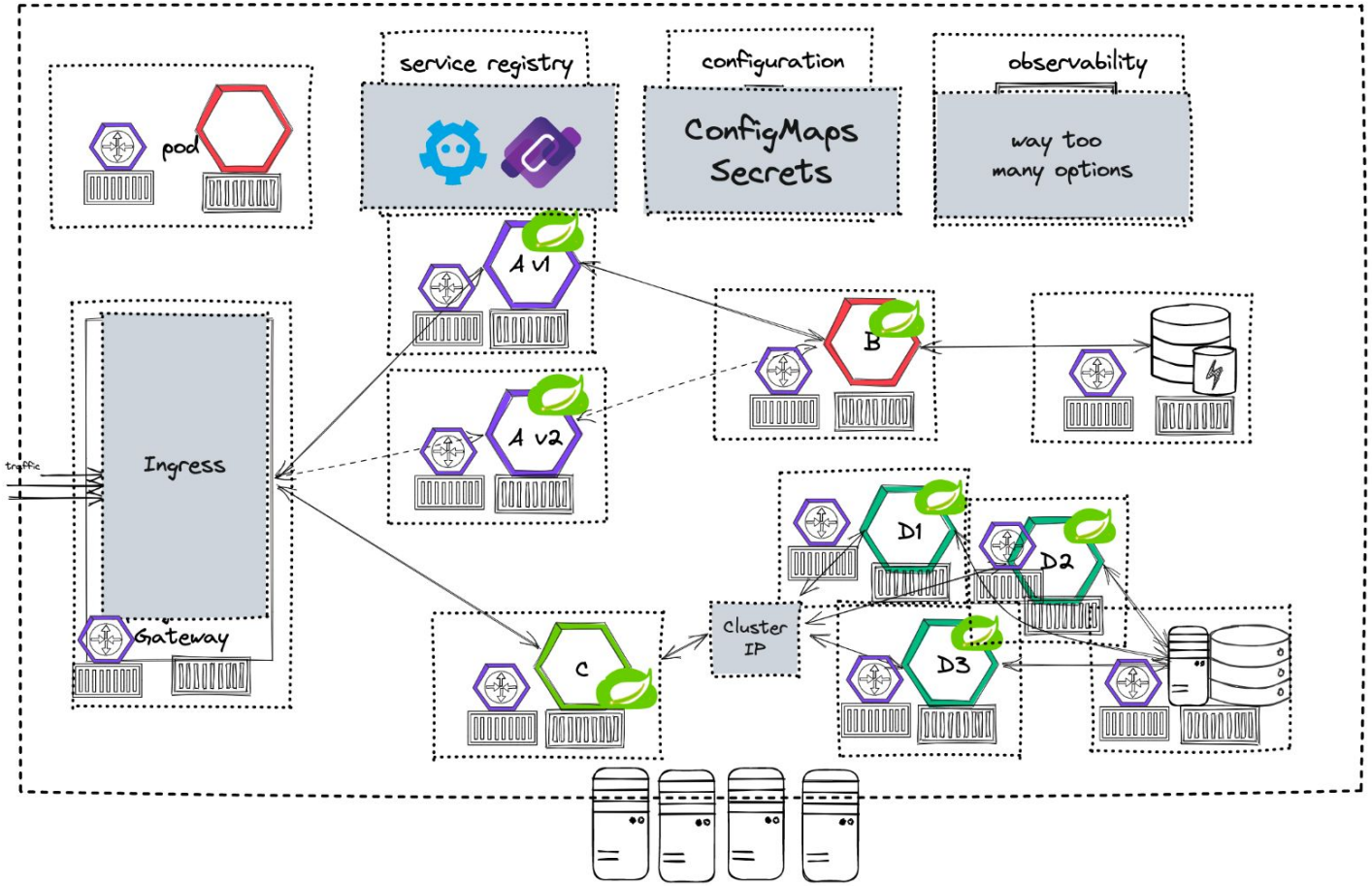

pod
pod
pod
pod
data plane
control plane
pod
control


Kubernetes
pod
service registry
configuration
ConfigMaps
Secrets
observability
way too
many options
A v1
A v2
B
Ingress
Gateway
traffic
C
Cluster
IP
D1
D2
D3

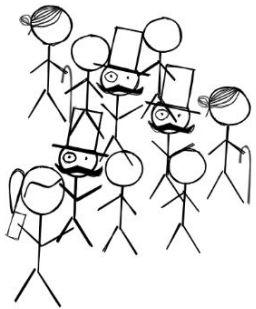

Kubernetes
pod
service registry
configuration
ConfigMaps
Secrets
observability
way too
many options
A v1
A v2
B
Ingress
traffic
Gateway
C
cluster
IP
D1
D2
D3
made by @maxddes with @excalidraw

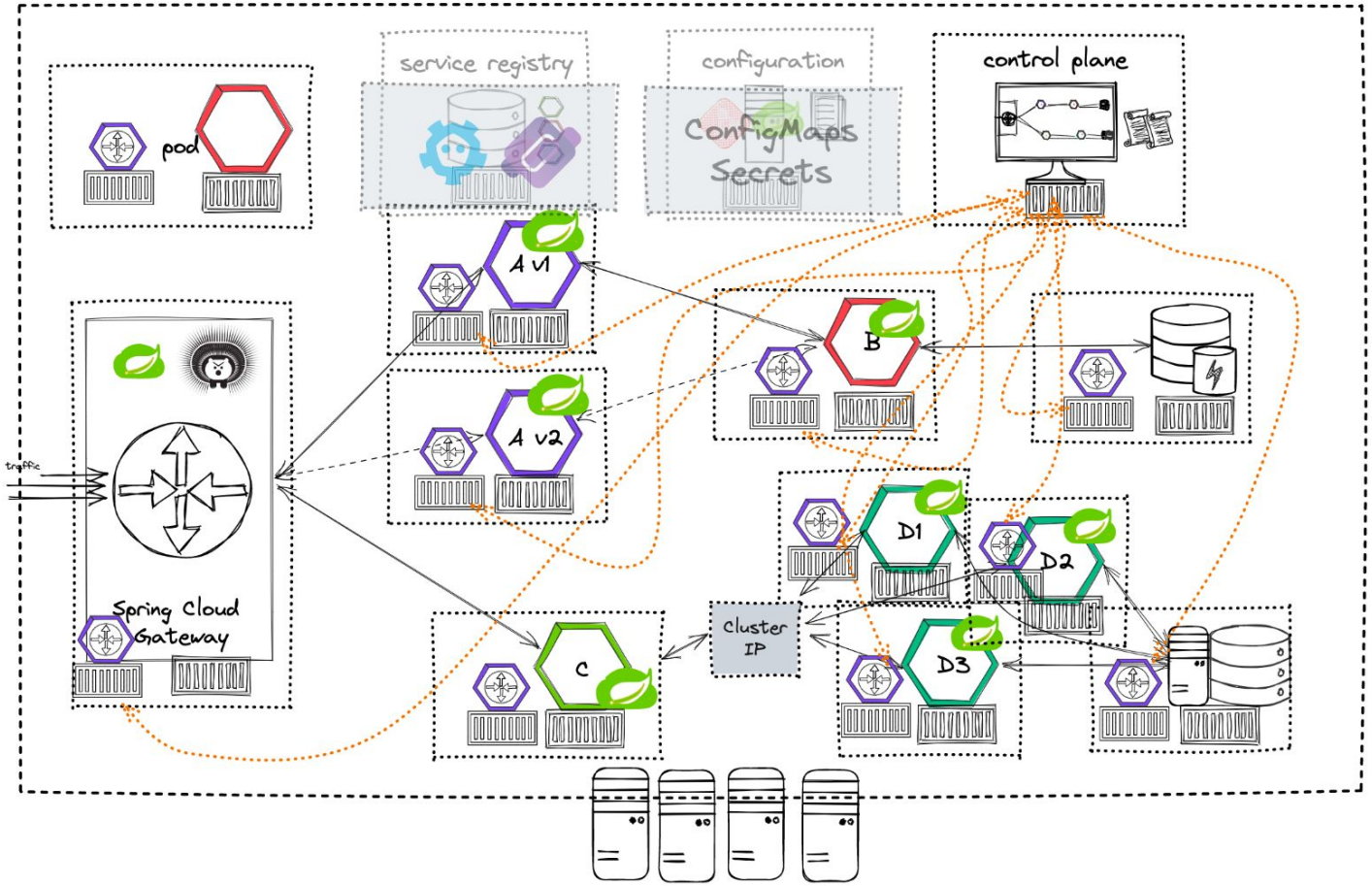
Kubernetes with sidecars



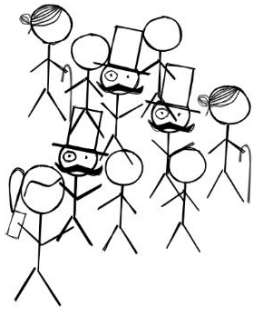


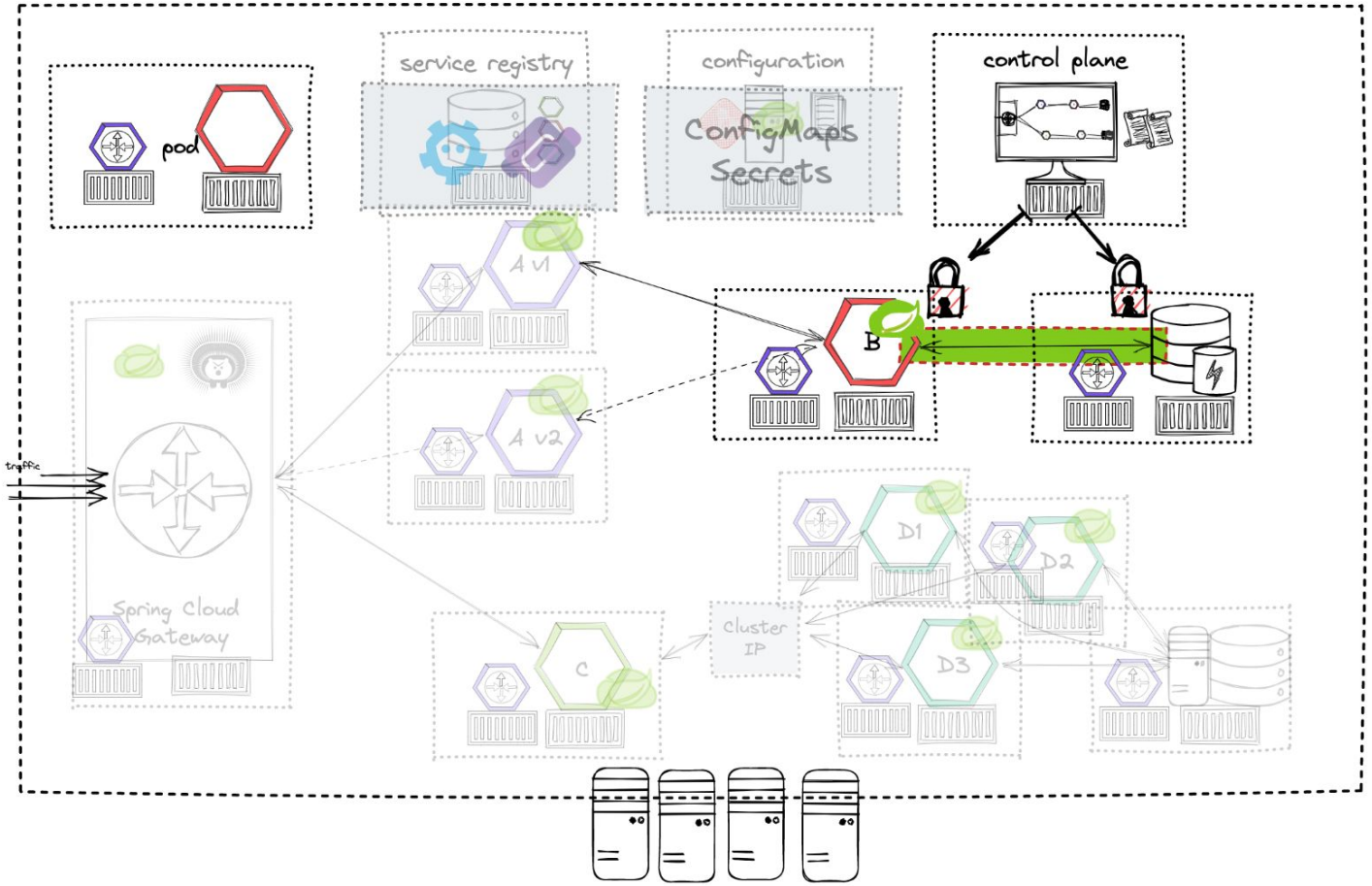
Kubernetes with sidecars



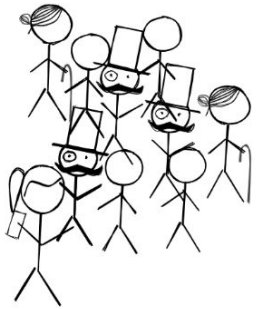


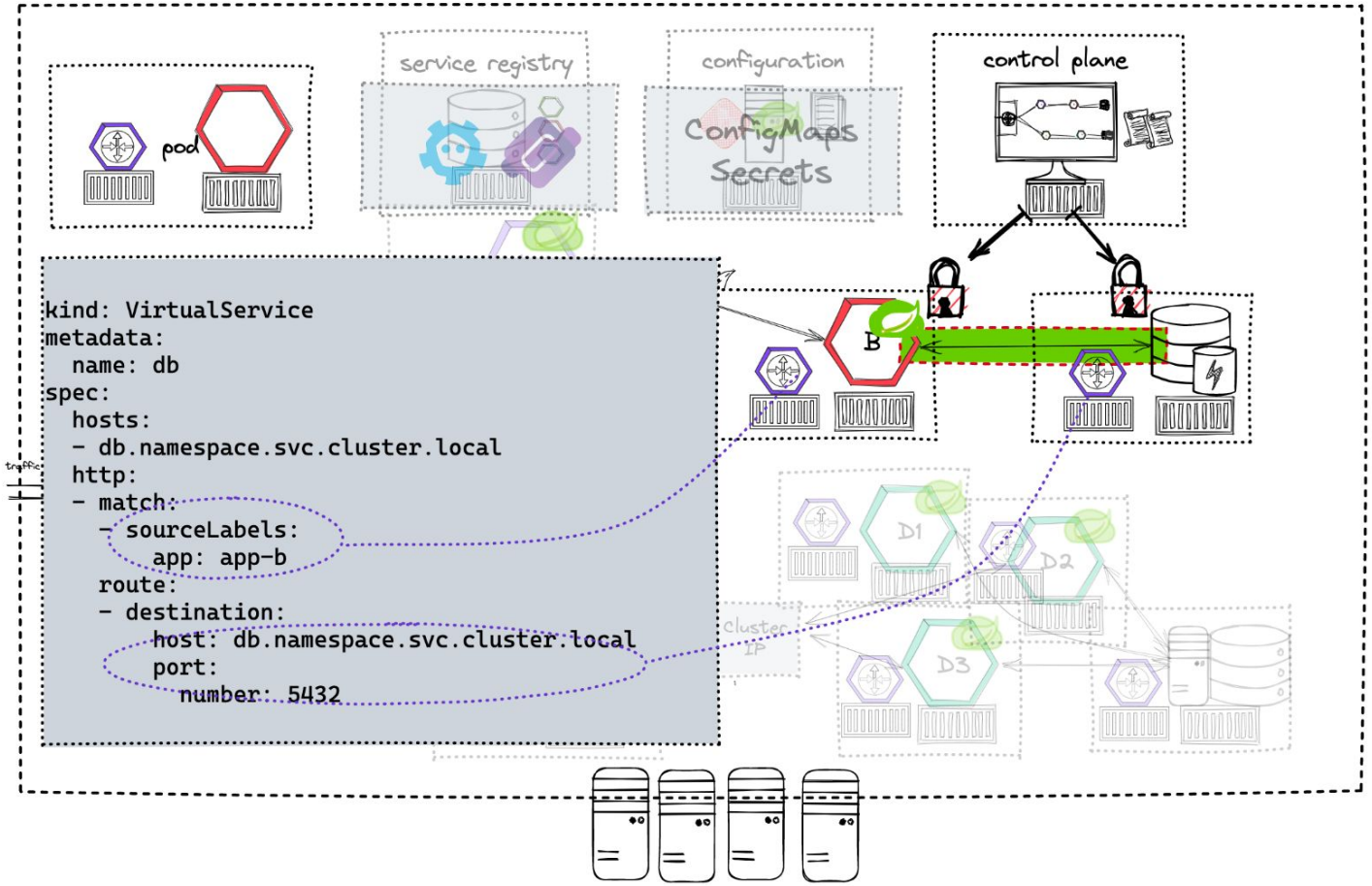
Kubernetes with sidecars

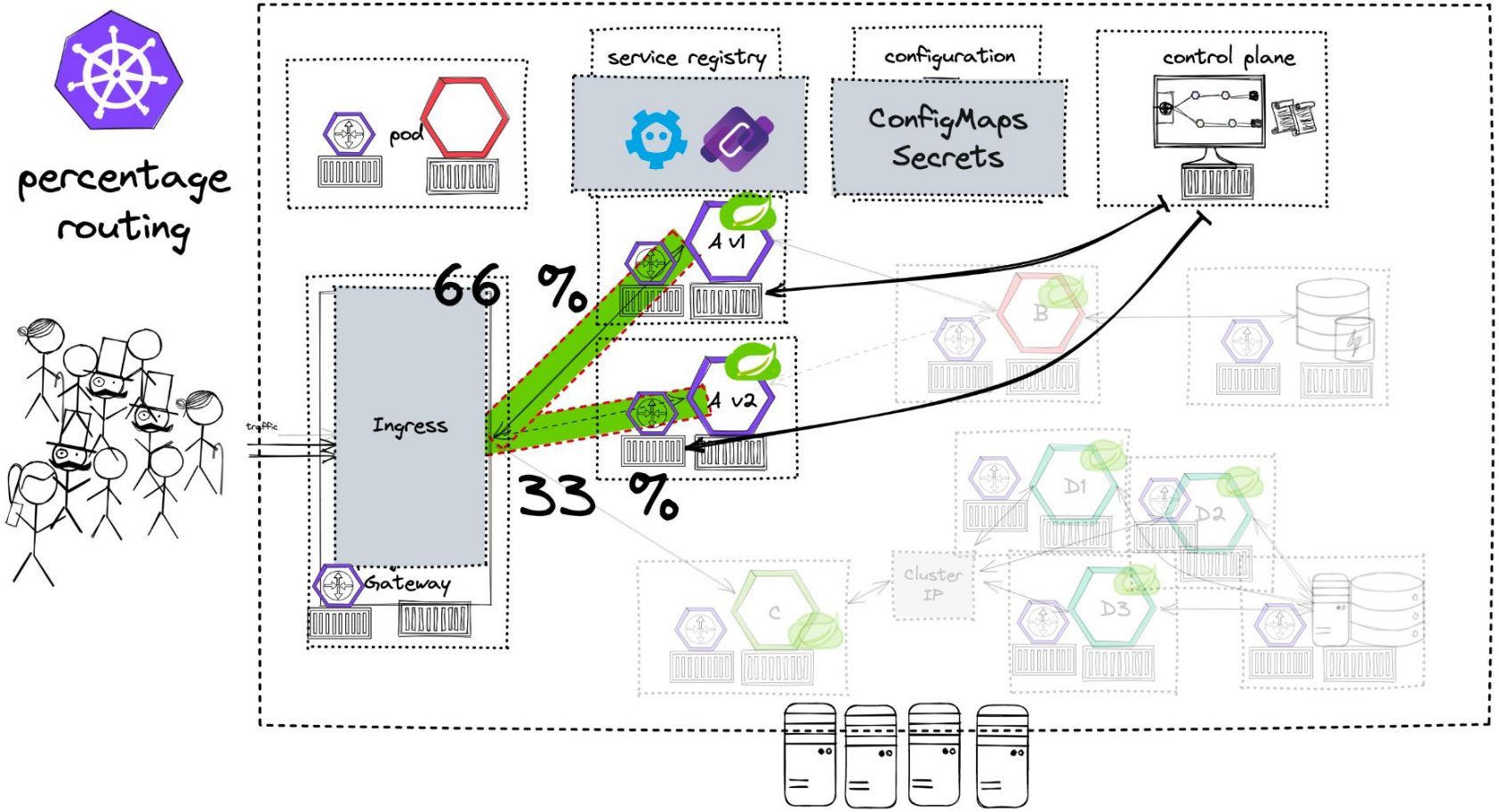




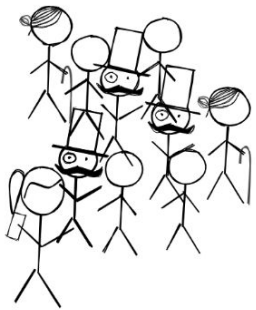
Kubernetes with sidecars

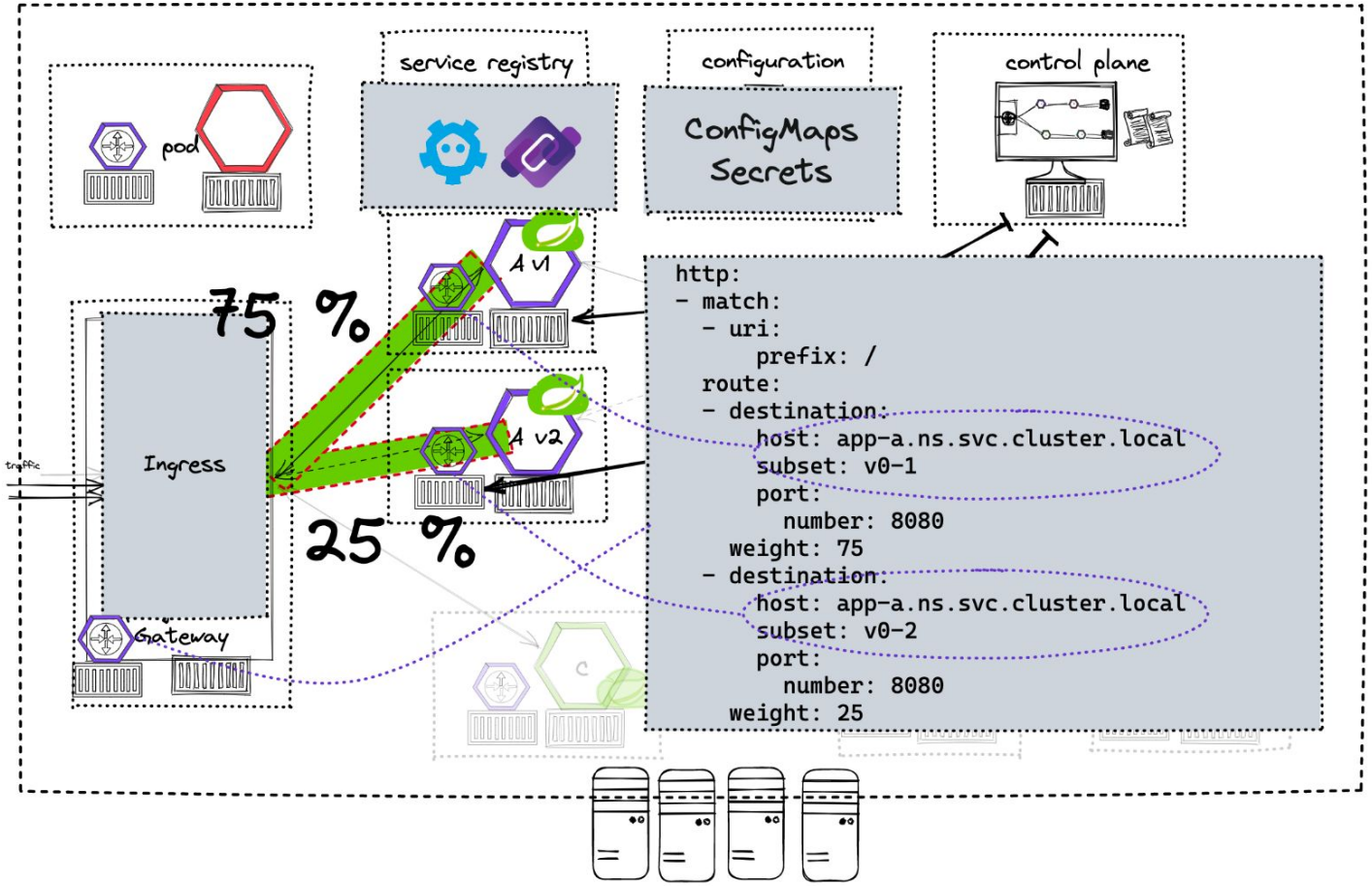




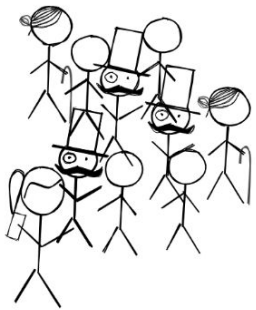

percentage routing
pod
service registry
configuration
ConfigMaps
Secrets
control plane
A v1
A v2
66 %
33 %
traffic
Ingress
Gateway
B
C
D1
D2
D3
cluster IP
made by @maeddes with @excalidraw

percentage routing

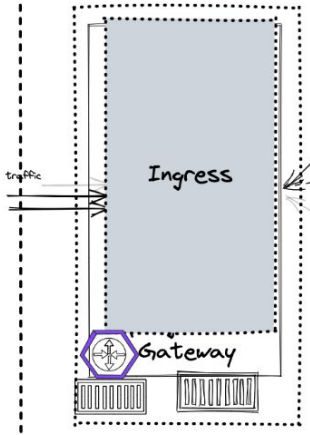




features



traffic



- "opt-in"
- fault injection
- circuit breaker
- retries

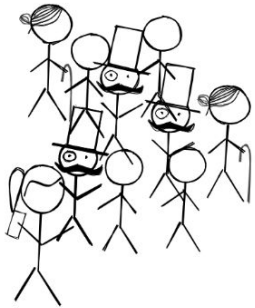
security

- certificates
- mutual TLS

policy enforcement

- rate limiting

Spring Cloud Gateway



service registry

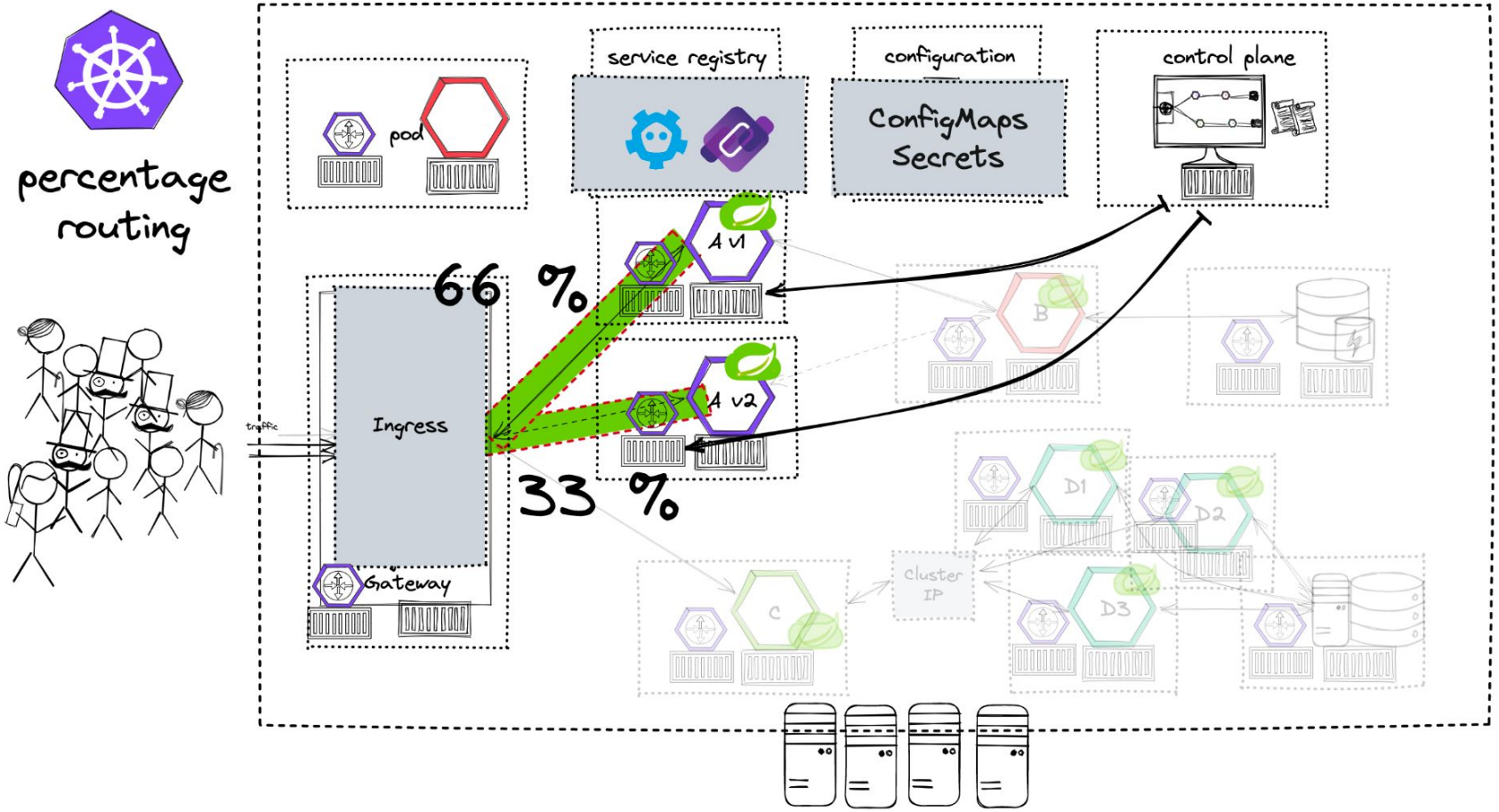
configuration

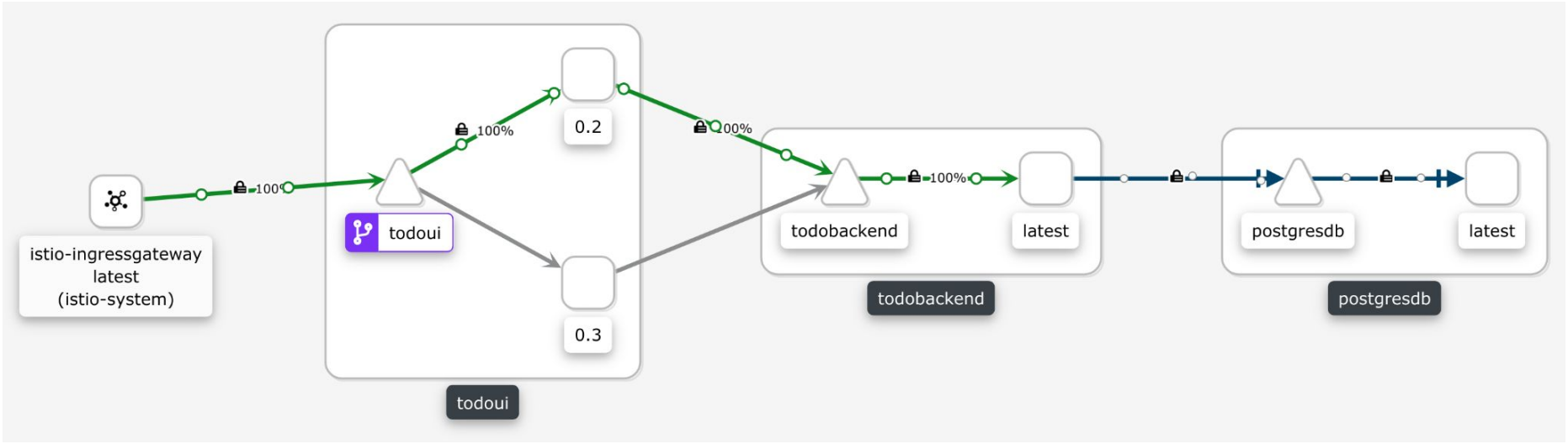
observability

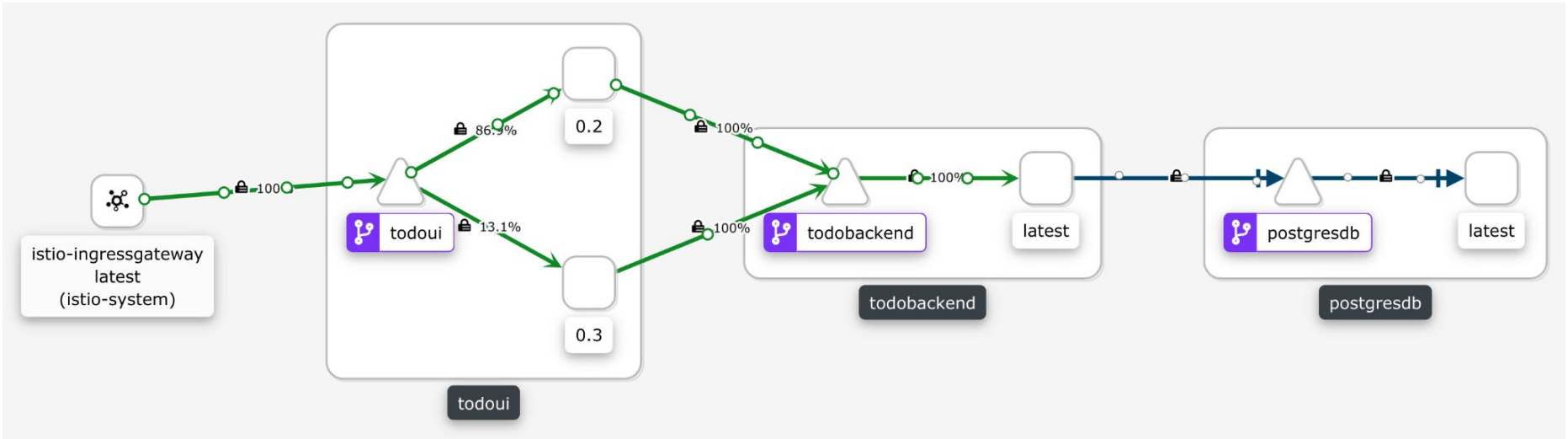
"predicates"
- timestamps (before, after ..)
- request (path, method, host, cookies ..)
- query
- weighted routing

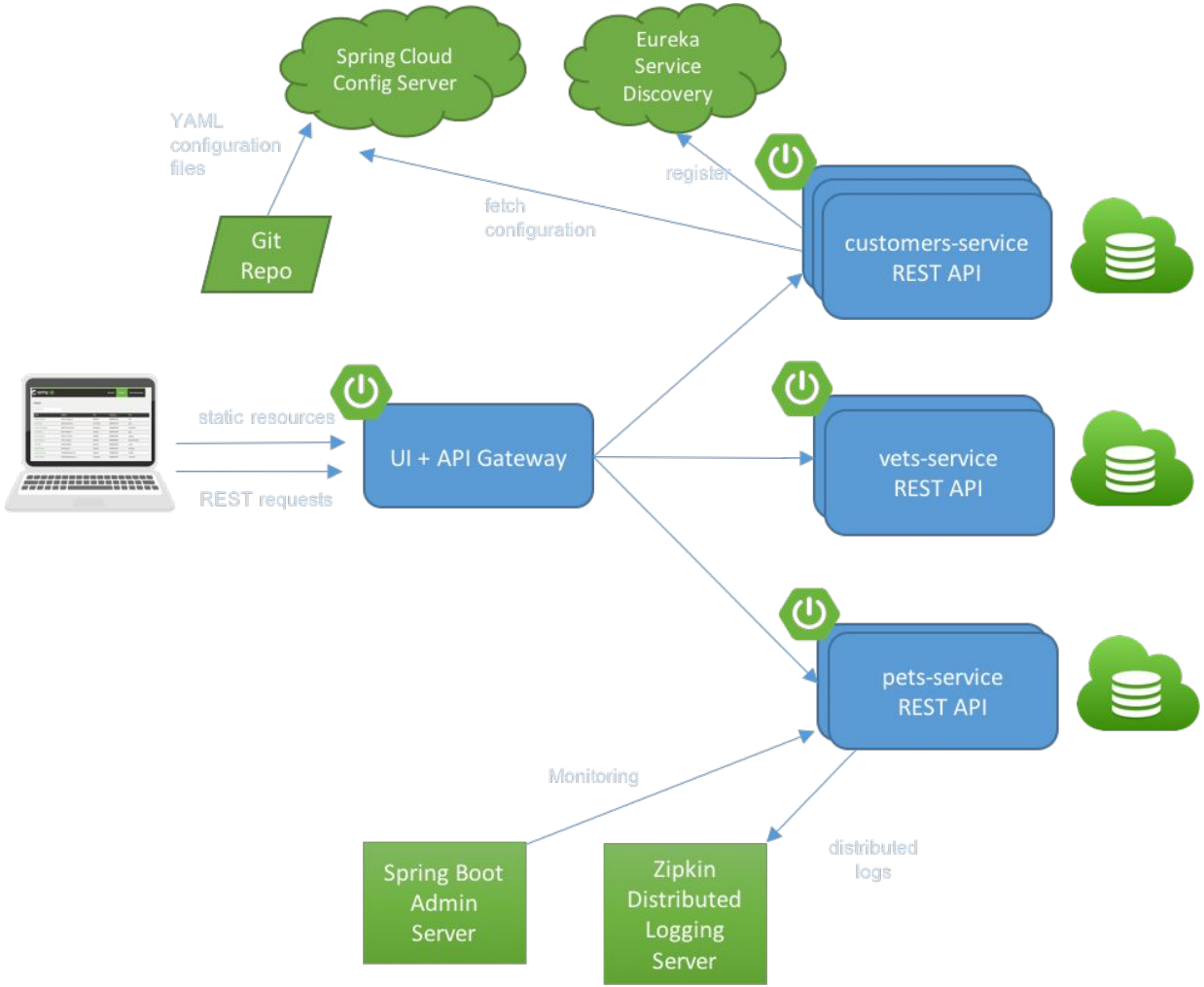
Security
- TLS & SSL

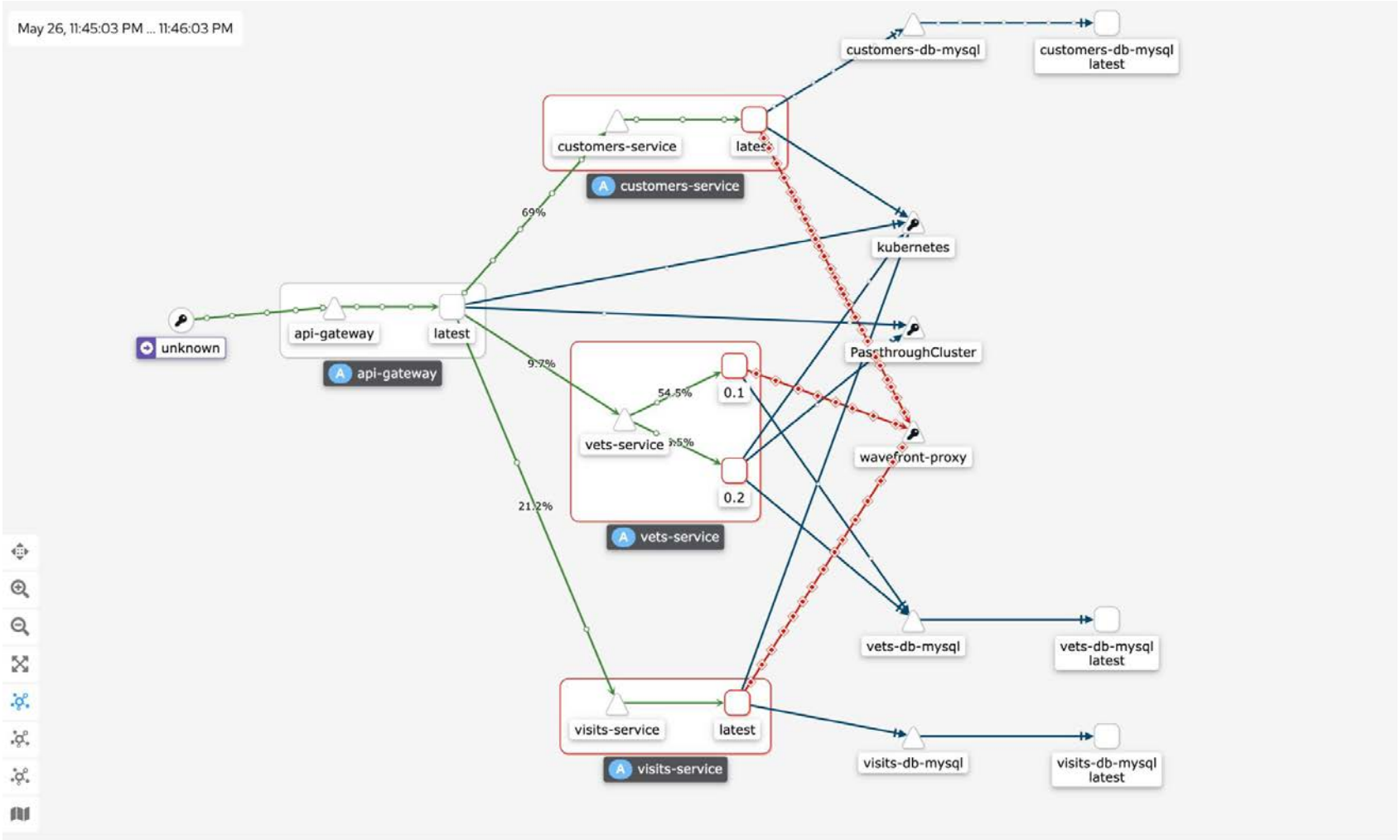
Resilience
- Timeout
- Circuit Breaker integration

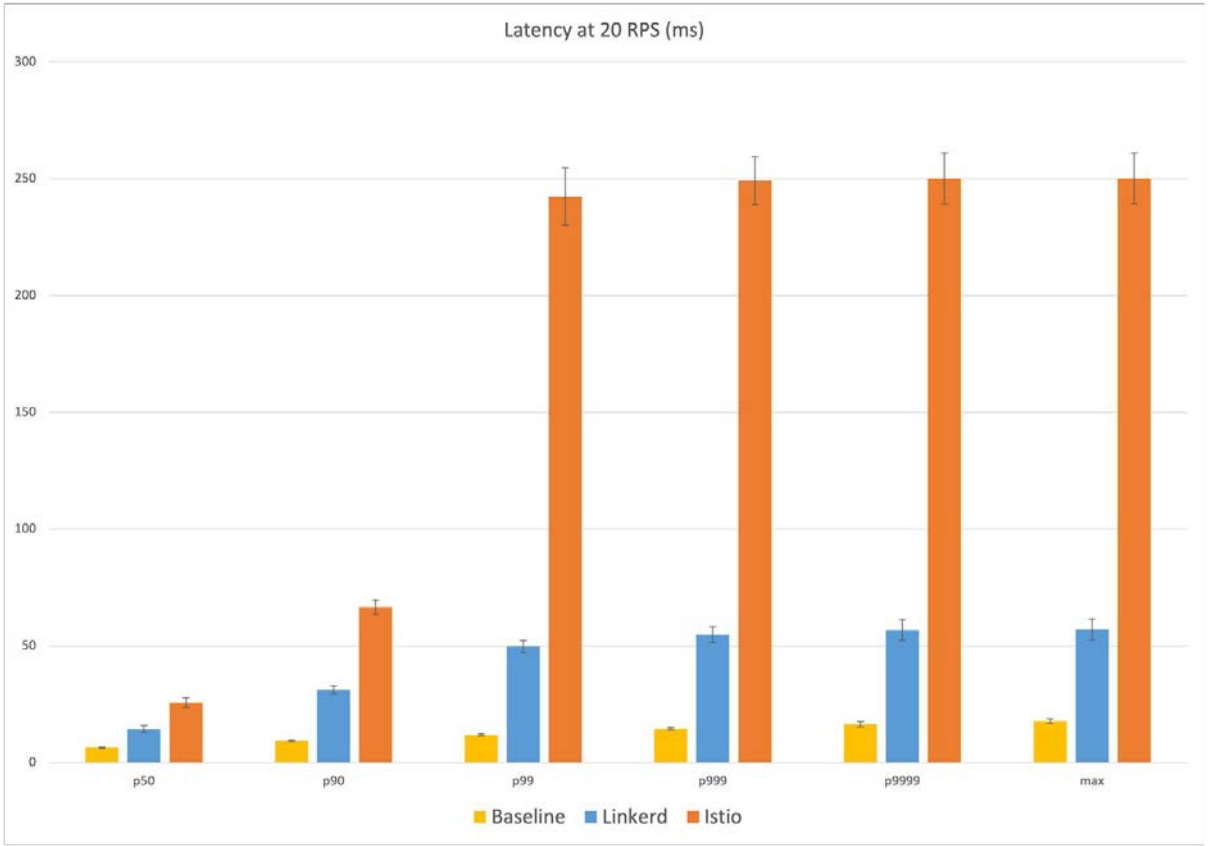

percentage routing
pod
service registry
configuration
ConfigMaps
Secrets
control plane
A v1
A v2
66 %
33 %
traffic
Ingress
Gateway
B
C
D1
D2
D3
cluster IP
made by @maeddes with @excalidraw


istio-ingressgateway
latest
(istio-system)
100%
todoui
100%
0.2
0.3
todoui
100%
todobackend
latest
100%
todobackend
postgresdb
latest
postgresdb


istio-ingressgateway
latest
(istio-system)
100
todoui
86.9%
13.1%
0.2
0.3
todoui
100%
100%
todobackend
100%
latest
todobackend
postgresdb
latest
postgresdb


Spring Cloud Config Server
Eureka Service Discovery
YAML configuration files
Git Repo
fetch configuration
register
customers-service REST API
static resources
REST requests
UI + API Gateway
vets-service REST API
pets-service REST API
Monitoring
Spring Boot Admin Server
Zipkin Distributed Logging Server
distributed logs


May 26, 11:45:03 PM ... 11:46:03 PM
customers-db-mysql
customers-db-mysql latest
customers-service
latest
customers-service
69%
kubernetes
unknown
api-gateway
latest
api-gateway
PassthroughCluster
9.7%
54.5%
0.1
vets-service
45.5%
wavefront-proxy
0.2
21.2%
vets-service
vets-db-mysql
vets-db-mysql latest
visits-service
latest
visits-db-mysql
visits-db-mysql latest
visits-service



https://www.cncf.io/blog/2021/12/17/benchmarking-linkerd-and-istio-2021-redux/

Characteristics:

- extends Kubernetes for limitations in traffic awareness and shaping capabilities

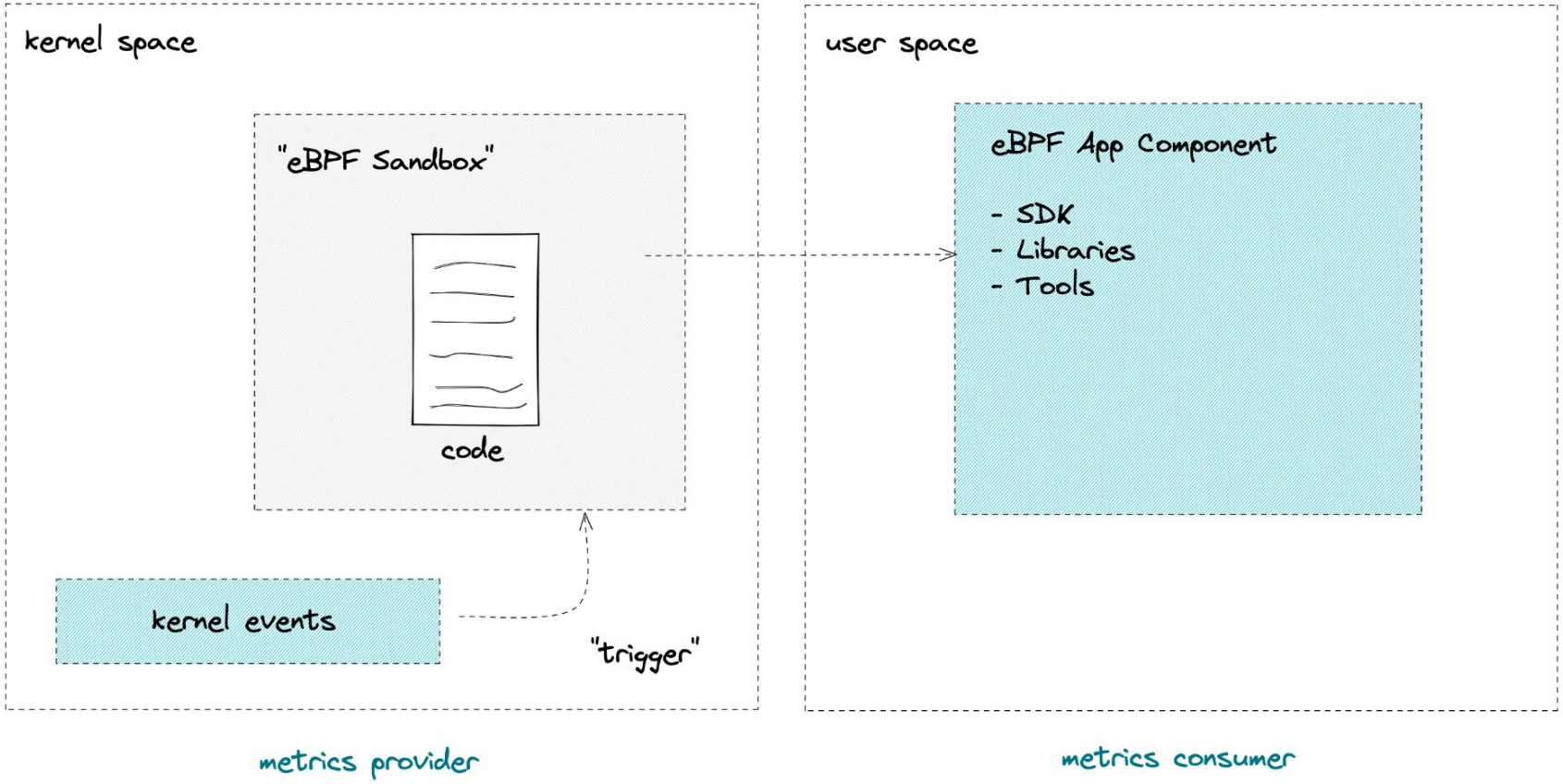
- concept of injecting sidecar-proxy into each pod to have control over entire network flow

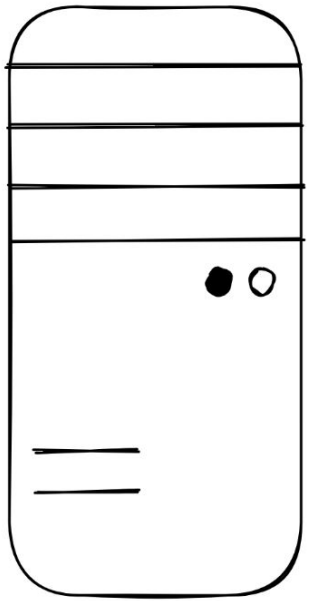
- no application level metrics

- configuration changes at runtime

eBPF

extended Berkeley Packet Filter


kernel space
"eBPF Sandbox"
code
kernel events
"trigger"
user space
eBPF App Component
- SDK
- Libraries
- Tools
metrics provider
metrics consumer



Kubernetes Worker Node

Container

Container

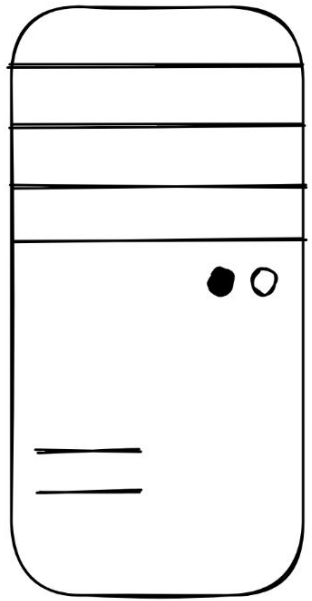
Container

kubelet

kube-proxy

Container Daemon

Operating System



Kubernetes Worker Node

Container

Container

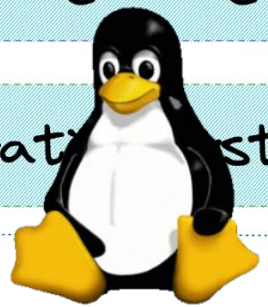
Container

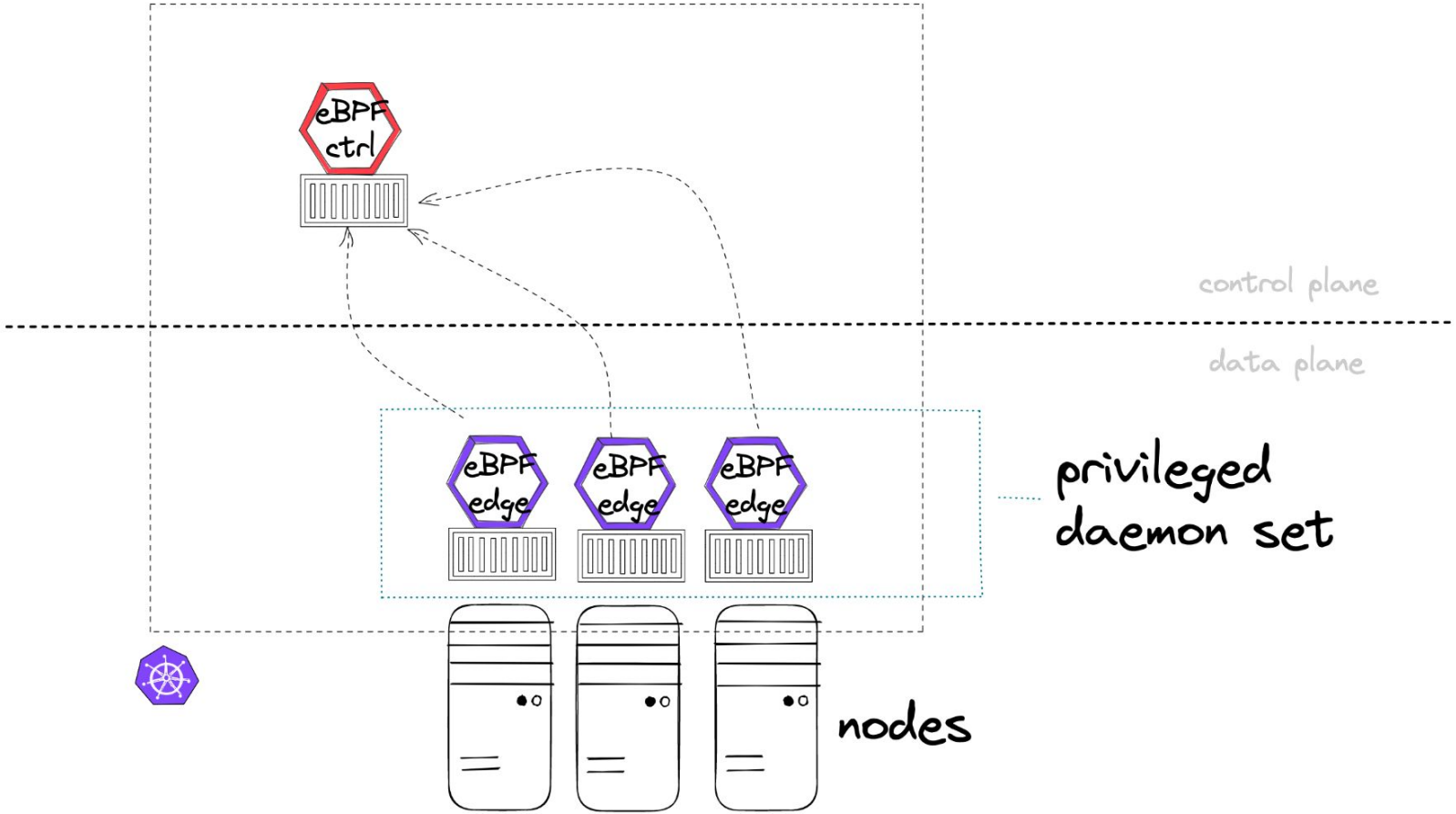
kubelet

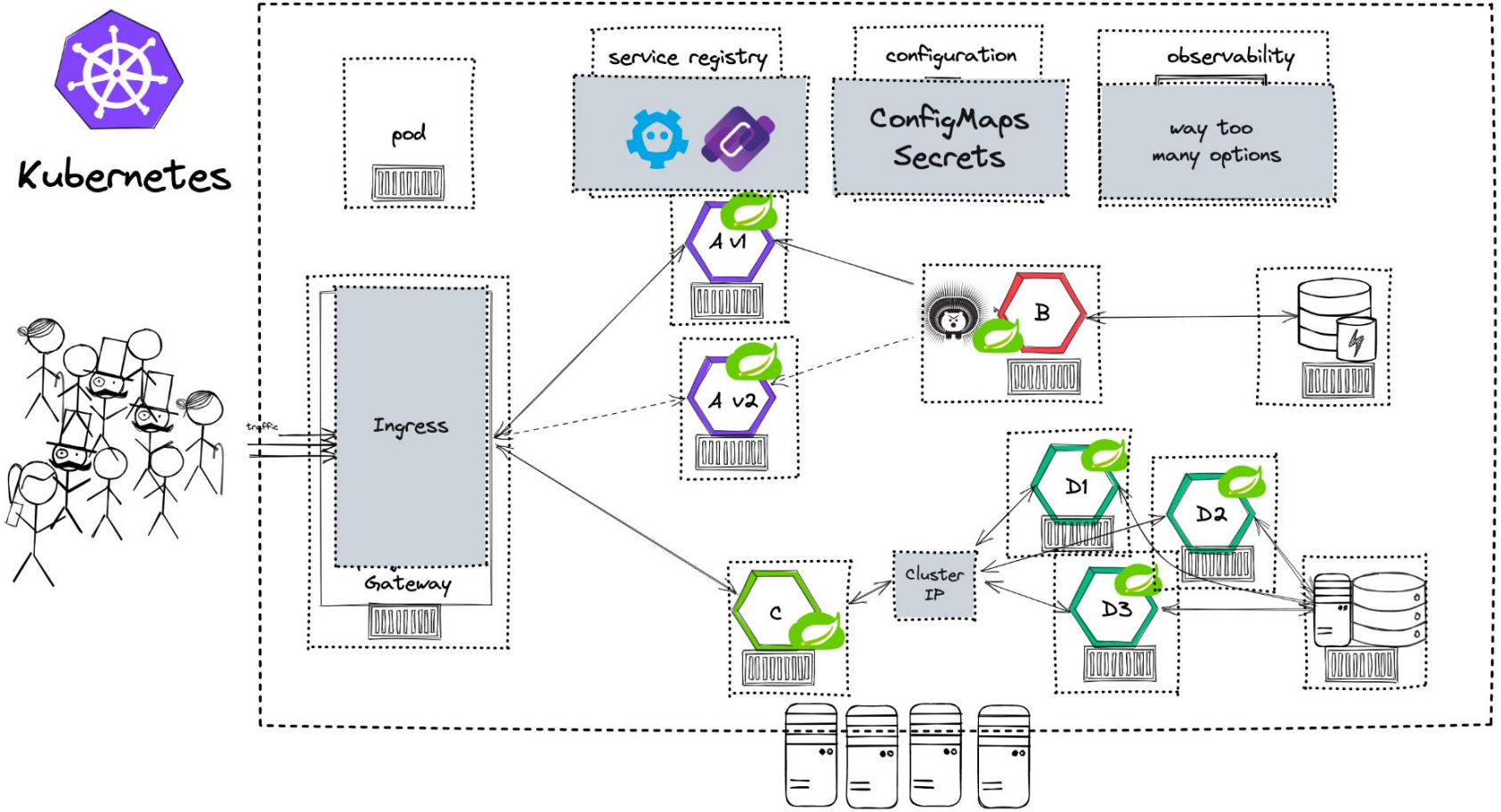
kube-proxy

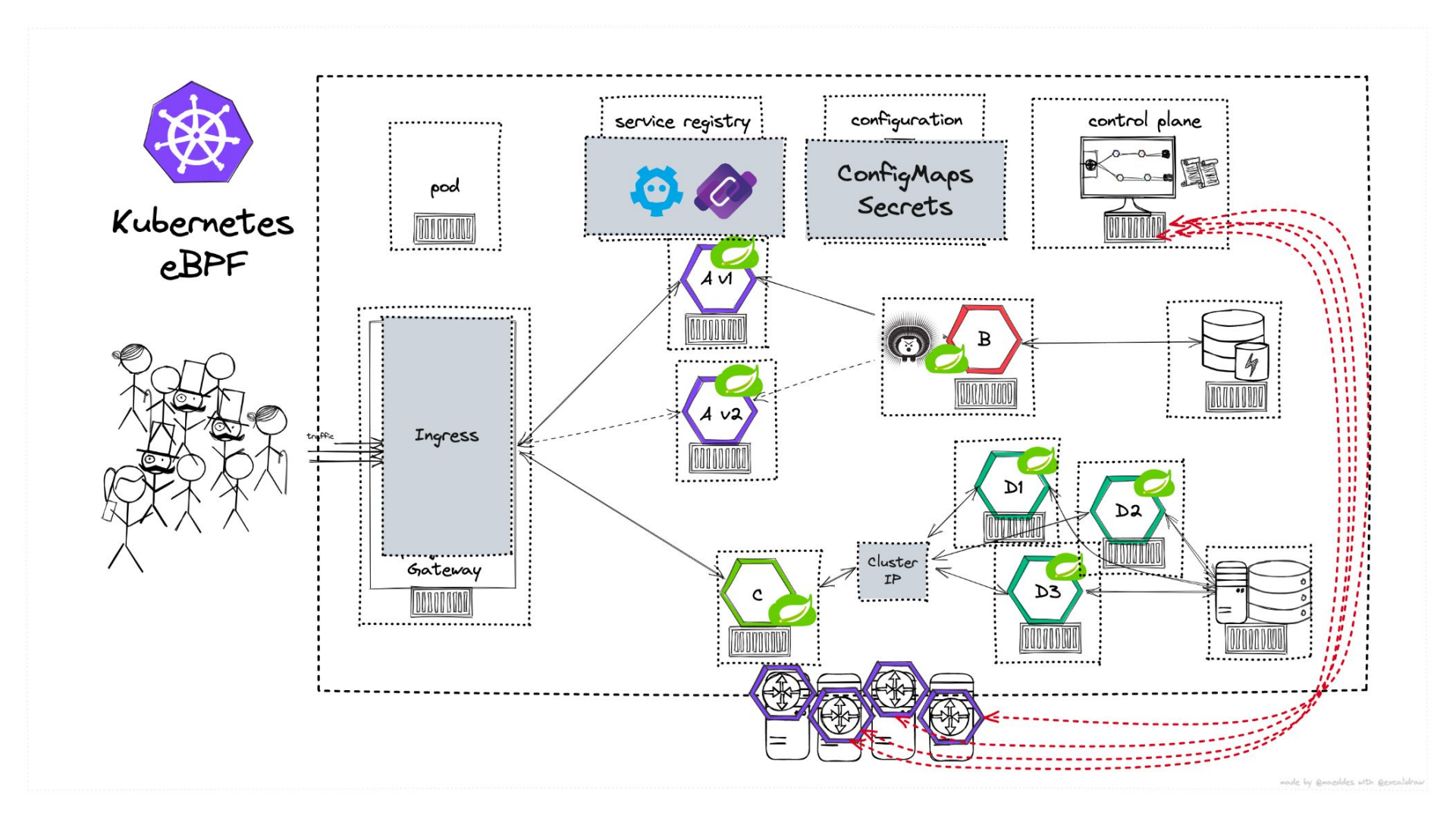
Container Daemon

Operat stem

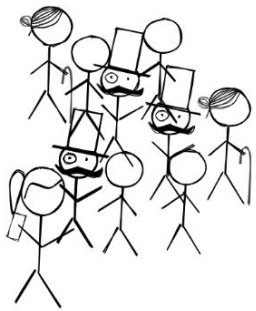


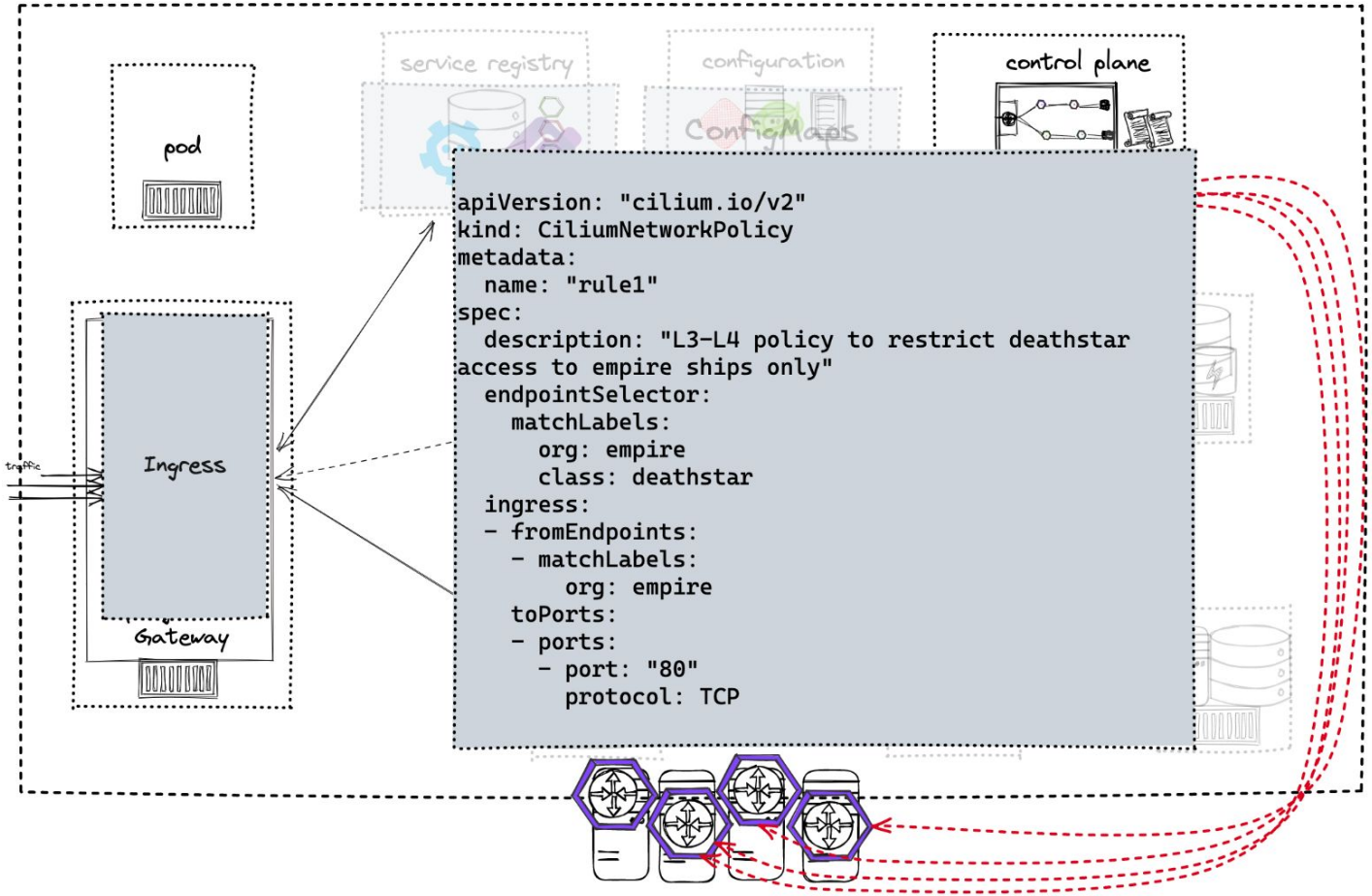


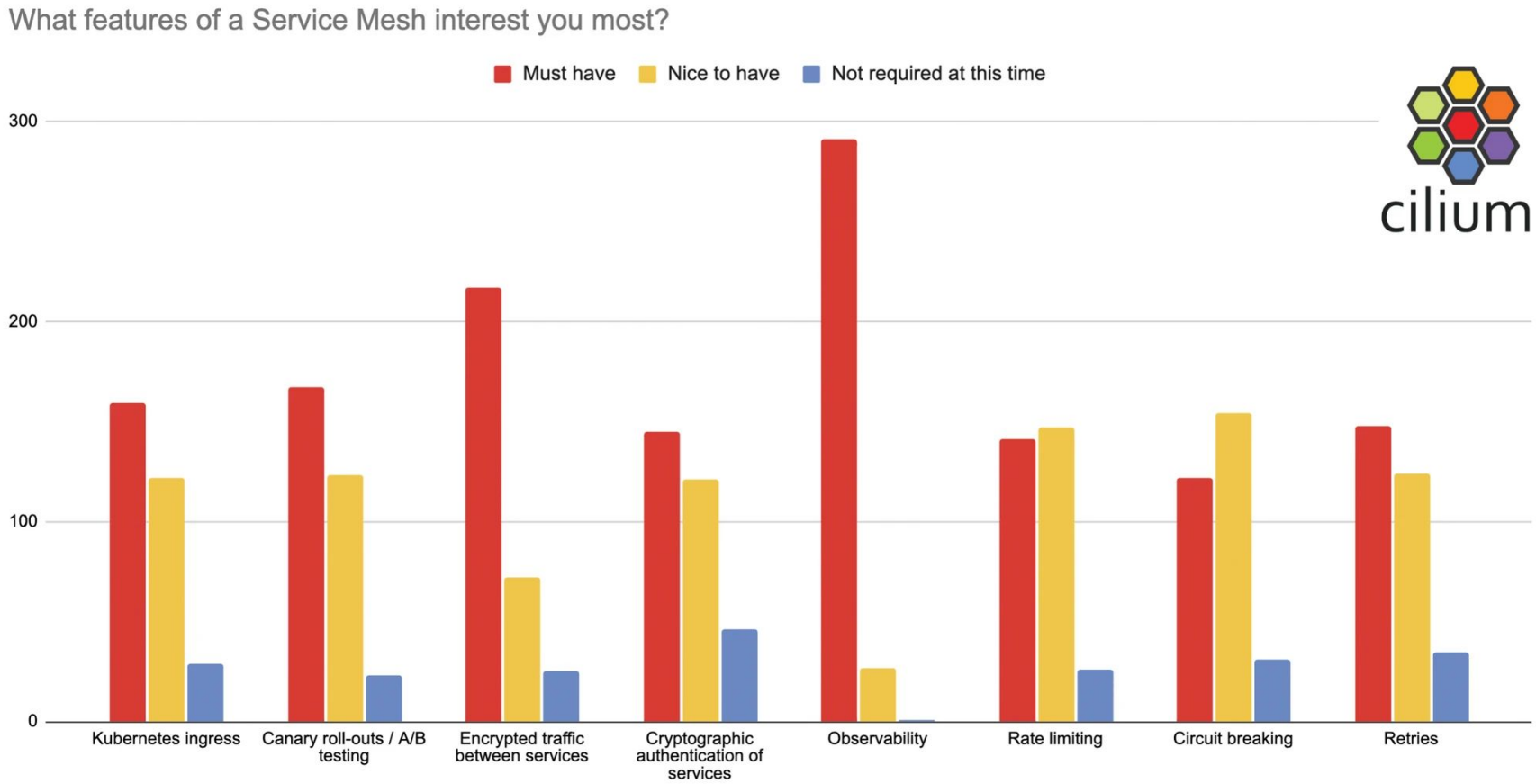

Kubernetes
pod
service registry
configuration
ConfigMaps
Secrets
observability
way too
many options
A v1
A v2
B
Ingress
Gateway
traffic
C
Cluster
IP
D1
D2
D3

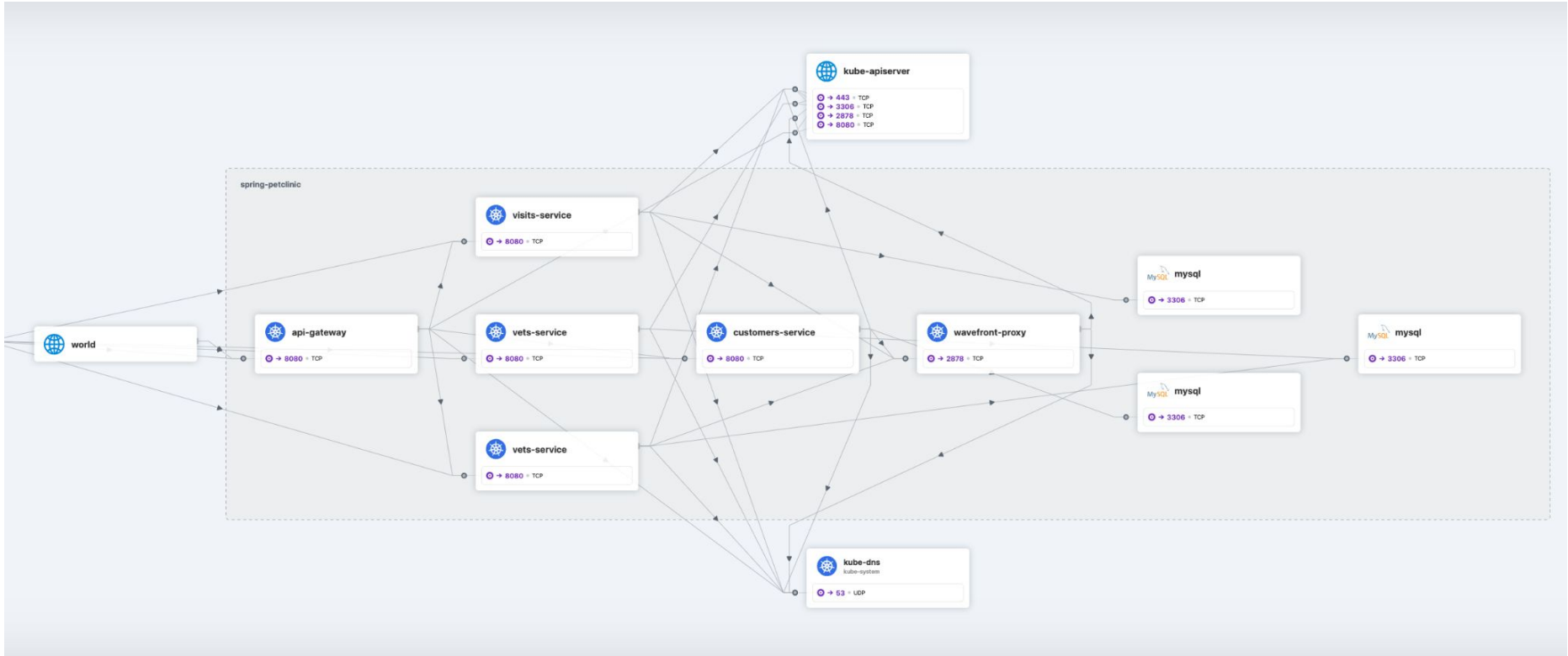

Kubernetes
eBPF
pod
service registry
configuration
ConfigMaps
Secrets
control plane
A v1
A v2
B
Ingress
Gateway
traffic
C
cluster
IP
D1
D2
D3
made by @maeddes with @excalidraw

Kubernetes eBPF





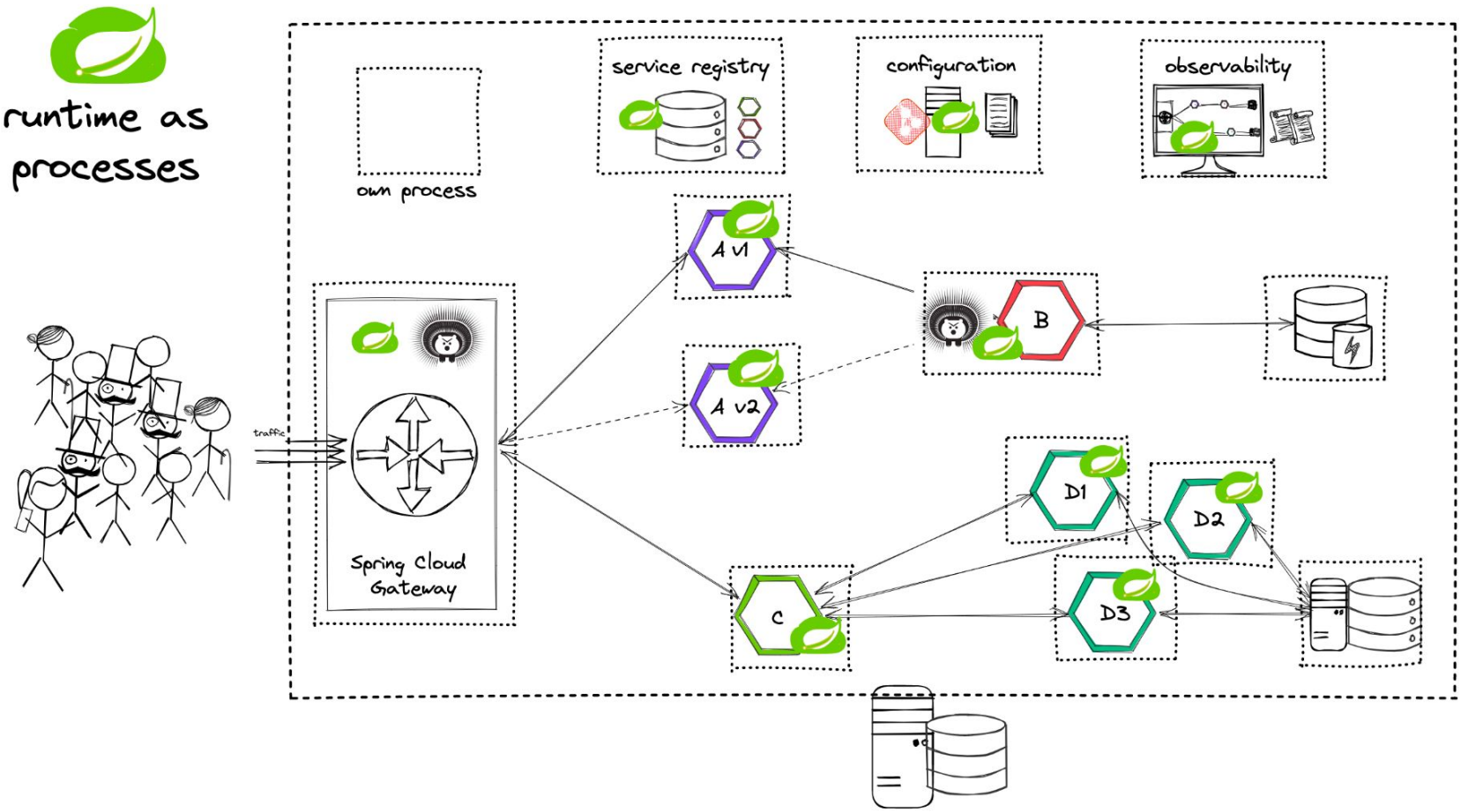

What features of a Service Mesh interest you most?
Must have
Nice to have
Not required at this time
300
200
100
0
Kubernetes ingress
Canary roll-outs / A/B testing
Encrypted traffic between services
Cryptographic authentication of services
Observability
Rate limiting
Circuit breaking
Retries
cilium


kube-apiserver
443 TCP
3306 TCP
2878 TCP
8080 TCP
spring-petclinic
visits-service
8080 TCP
mysql
3306 TCP
world
api-gateway
8080 TCP
vets-service
8080 TCP
customers-service
8080 TCP
wavefront-proxy
2878 TCP
mysql
3306 TCP
mysql
3306 TCP
vets-service
8080 TCP
kube-dns
kube-system
53 UDP

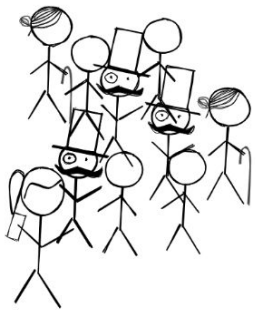
Characteristics:

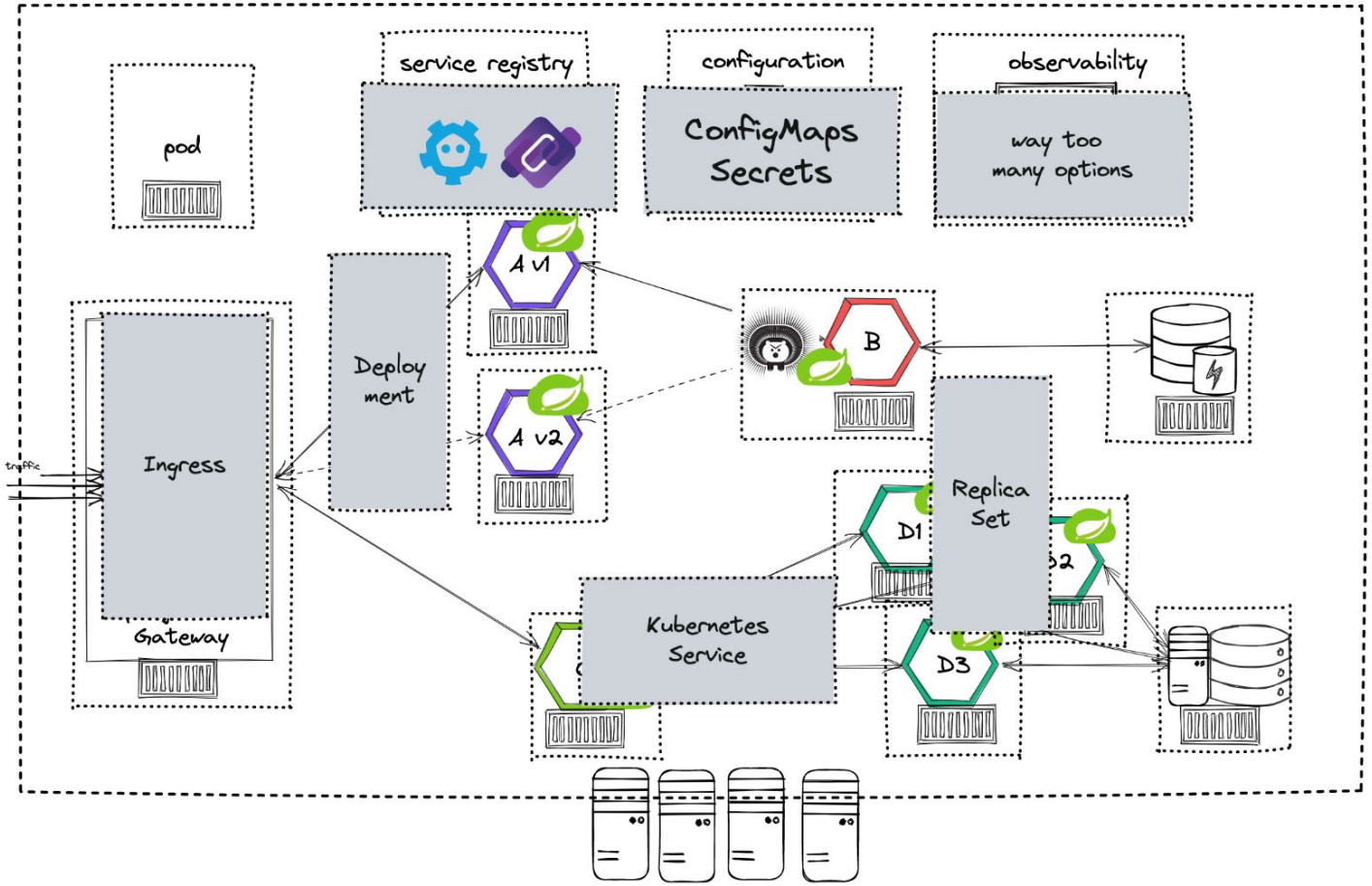
- injection of proxy component on node instead of pod level
- Linux low-level functionality leveraged for Kubernetes
- early stage of development
- Service Mesh functionality in beta program level by Cilium
- lower latency with less hops as opposed to pod injection

runtime as processes

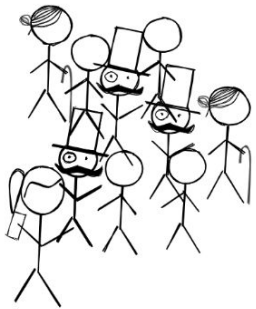


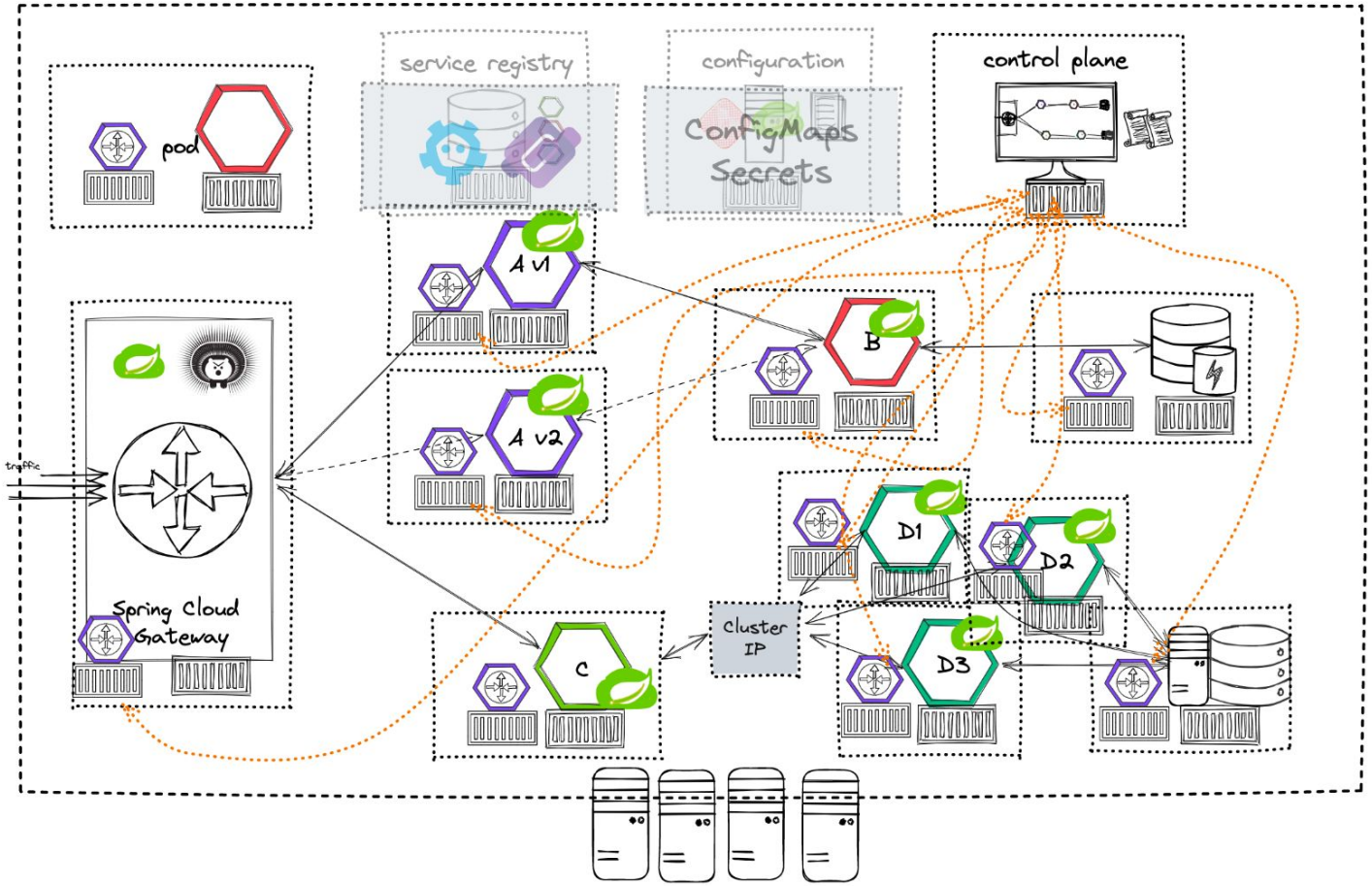
Kubernetes

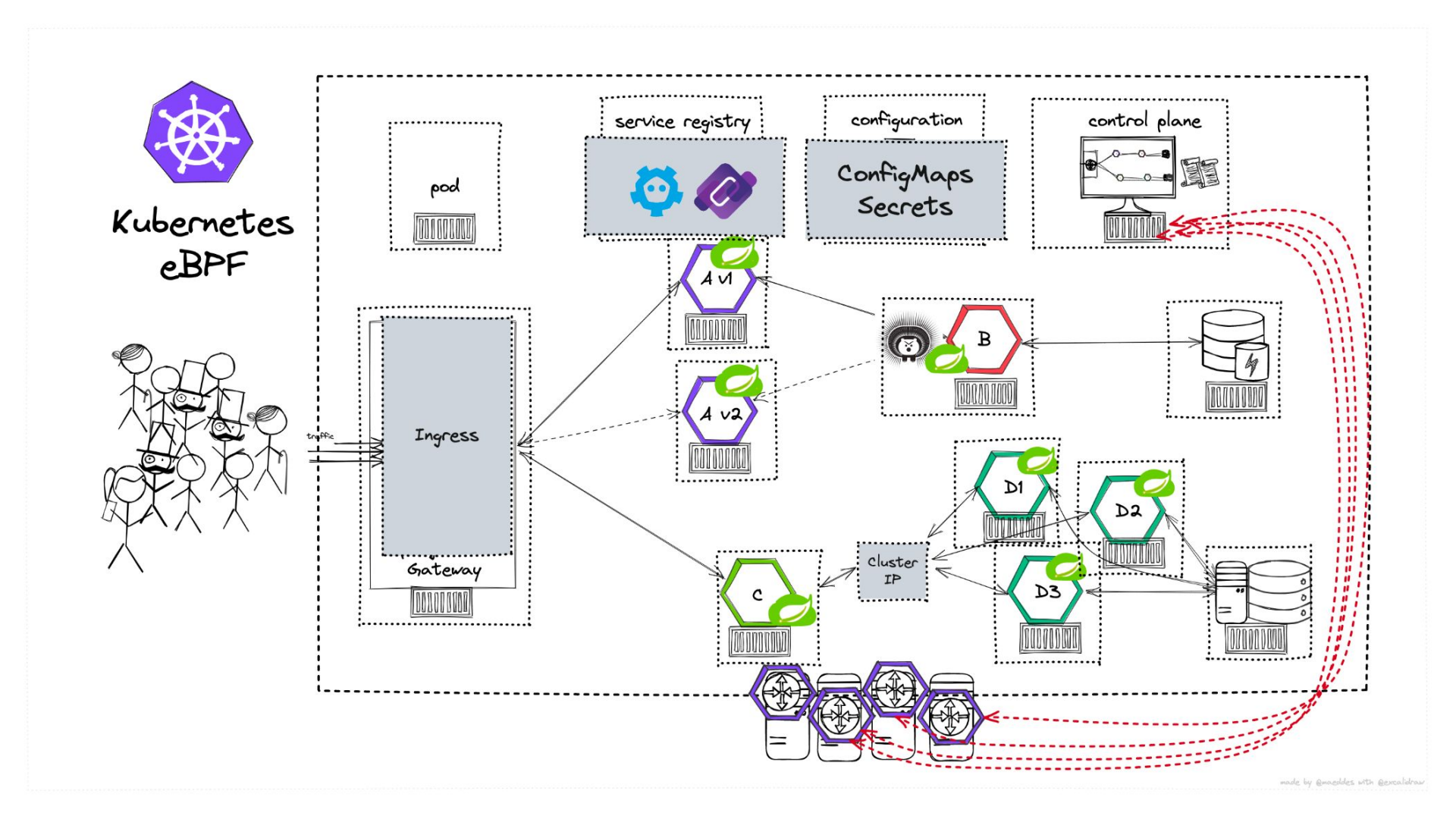




Kubernetes with sidecars






Kubernetes
eBPF
pod
service registry
configuration
ConfigMaps
Secrets
control plane
A v1
A v2
B
Ingress
Gateway
traffic
C
cluster
IP
D1
D2
D3
made by @maeddes with @excalidraw

Thanks!

matthiashaeussler

@maeddes

Matthias Haeussler
Chief Technologist

Twitter: @maeddes
E-Mail: matthias.haeussler@novatec-gmbh.de

Novatec Consulting GmbH
Bertha-Benz-Platz 1
D-70771 Leinfelden-Echterdingen

T. +49 711 22040-700
info@novatec-gmbh.de
www.novatec-gmbh.de