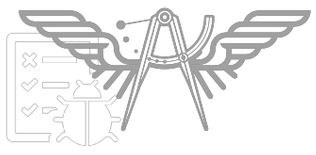
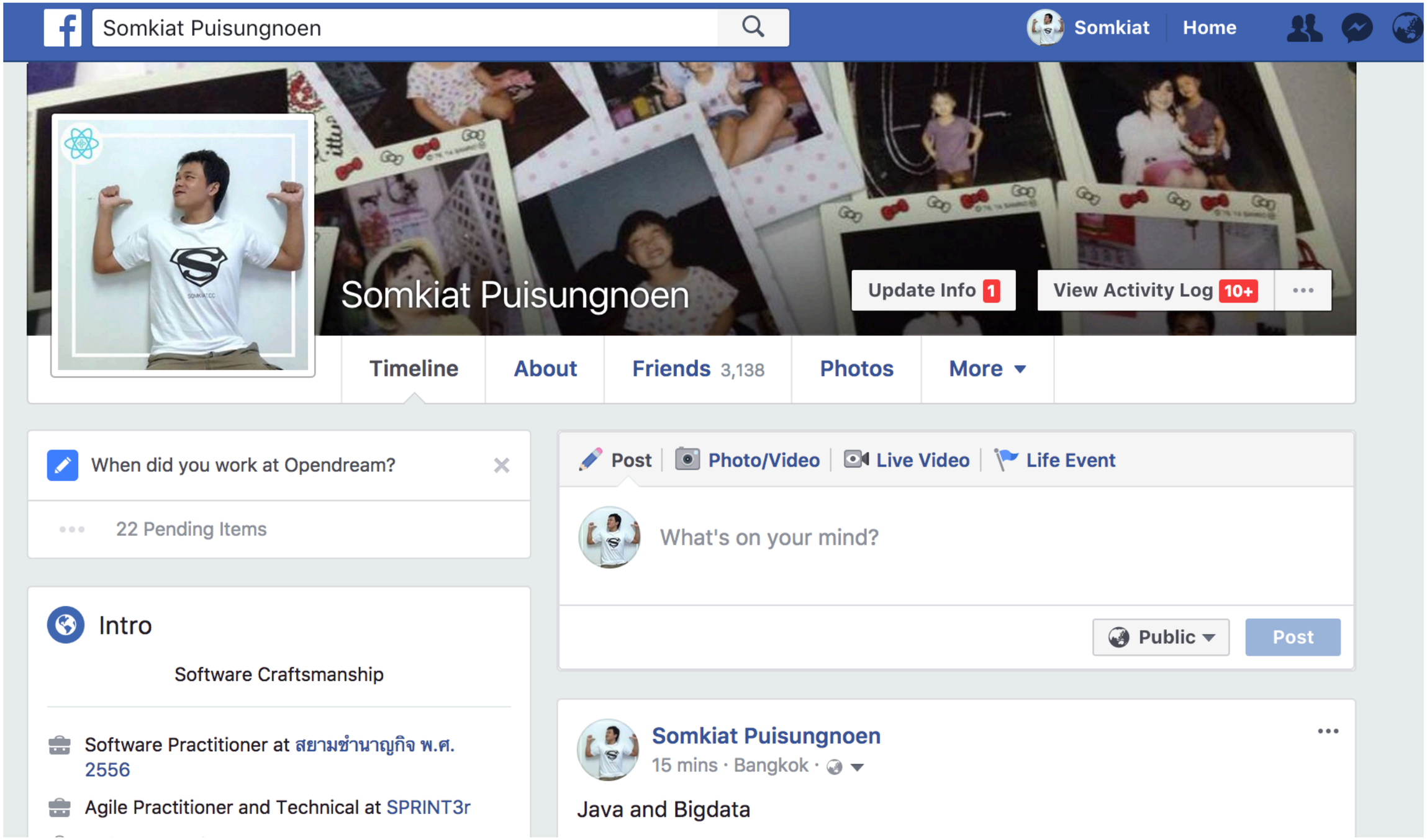
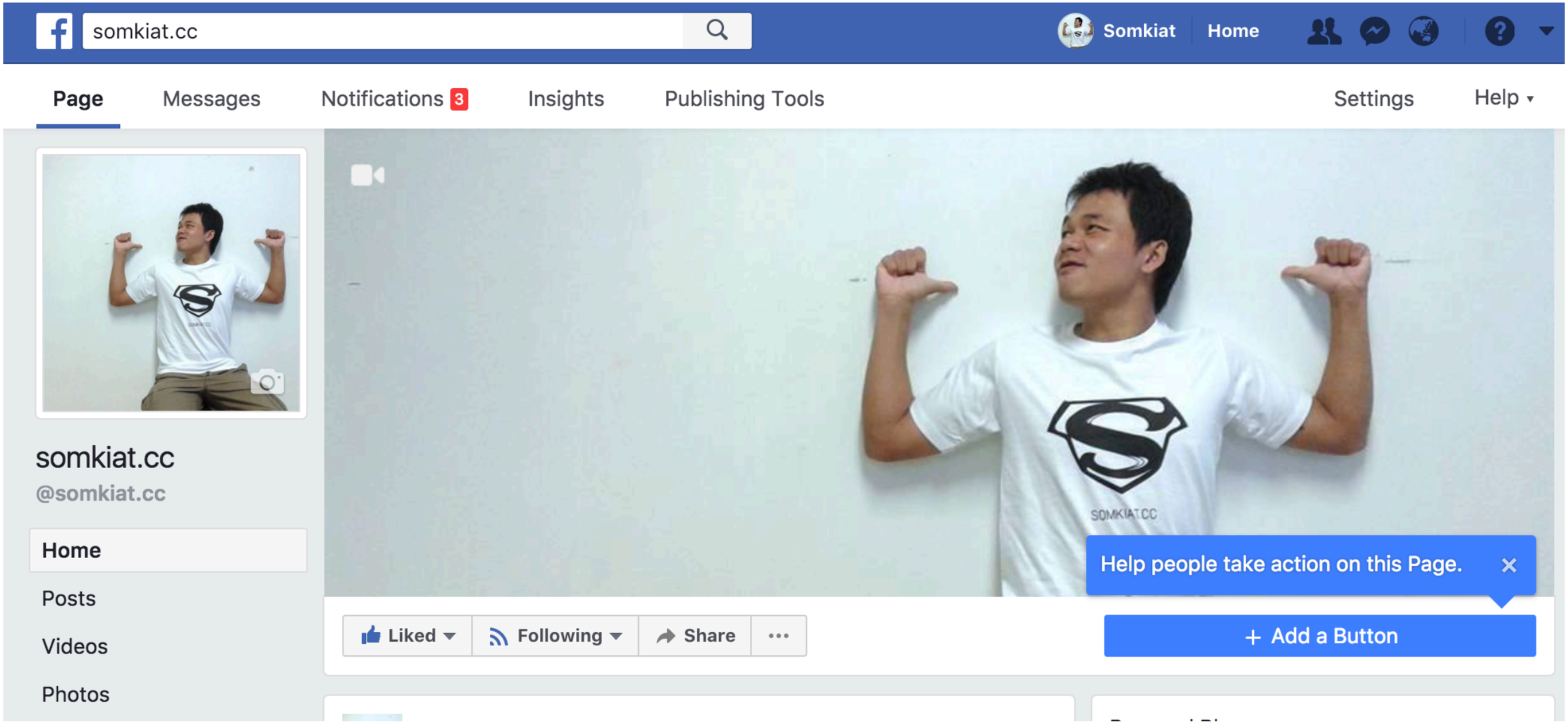
สยามชำนาญกิจ
SIAM CHAMNAN KIT

Microservices workshop







Agenda

Recap Microservice
Properties of Microservice
Microservice 1.0 - 4.0
How to develop Microservice ?
How to test Microservice ?
12-factors app
Workshop

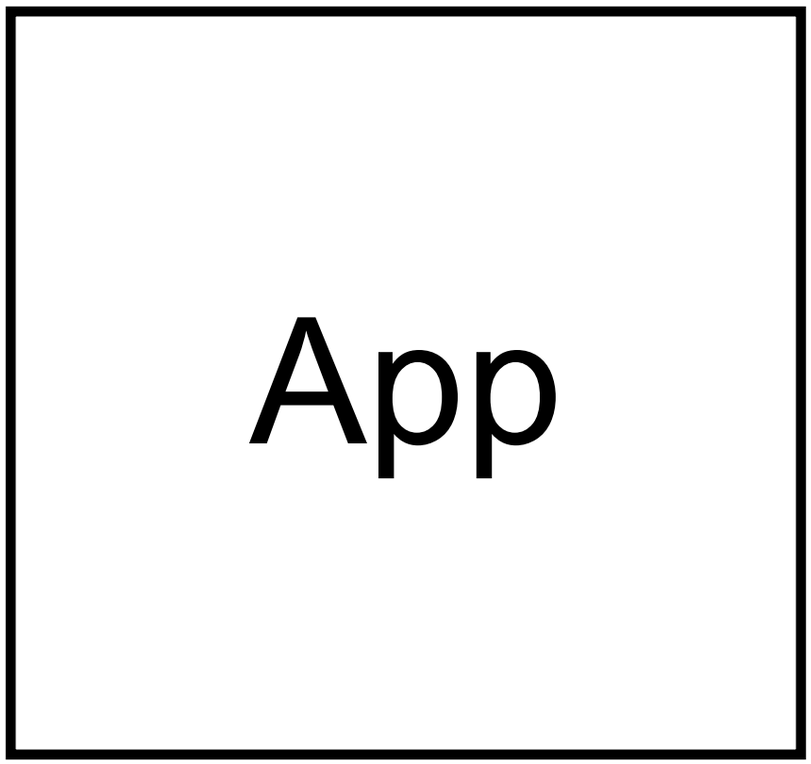
Recap Microservice

Microservices history

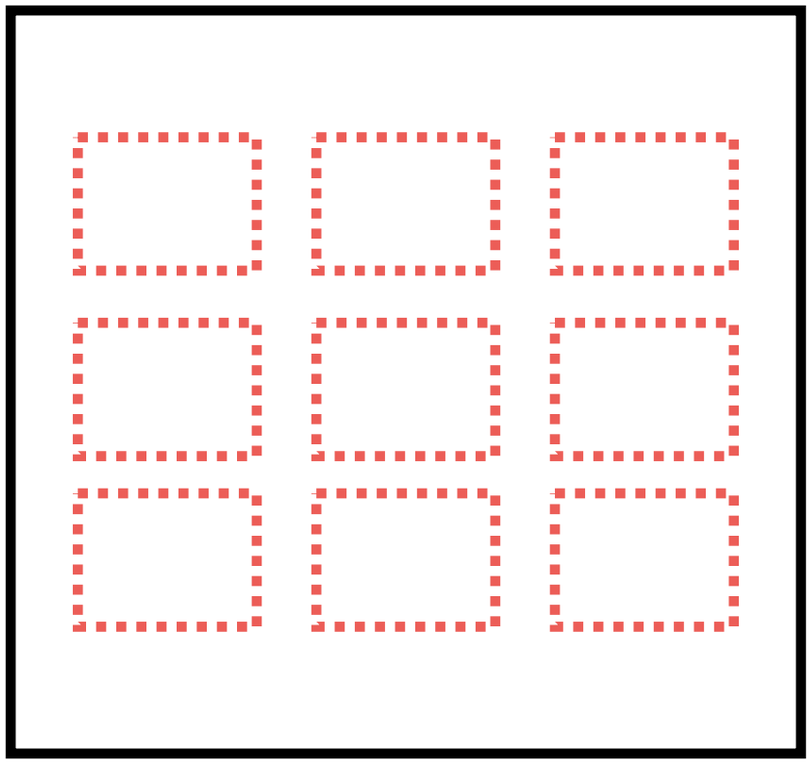
Microservices ?

More/Low splitting
More network interaction
Data storing and sharing
Compatibility issues
Testing
Operation & Monitoring

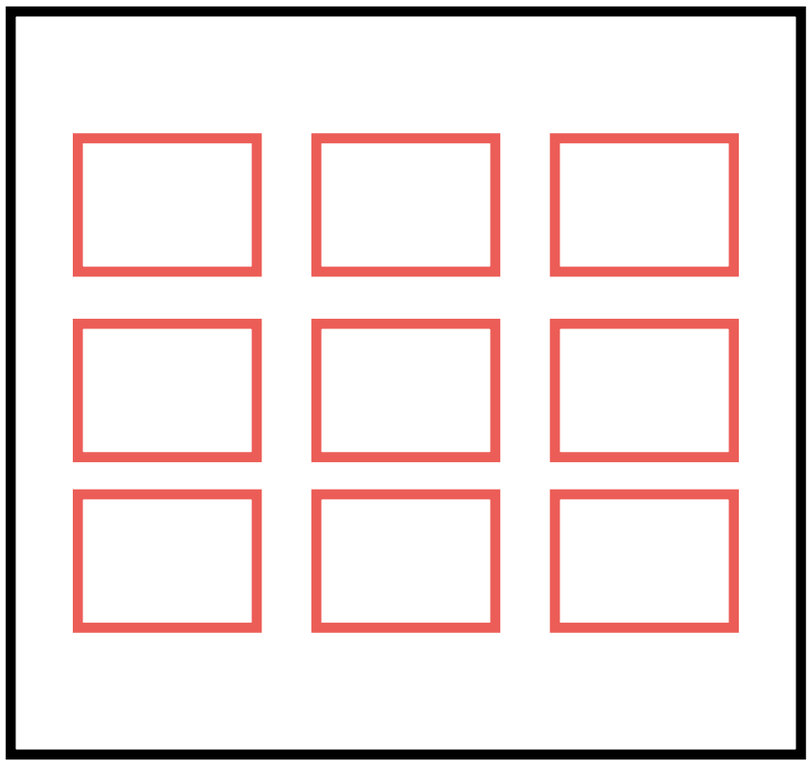
Monolith



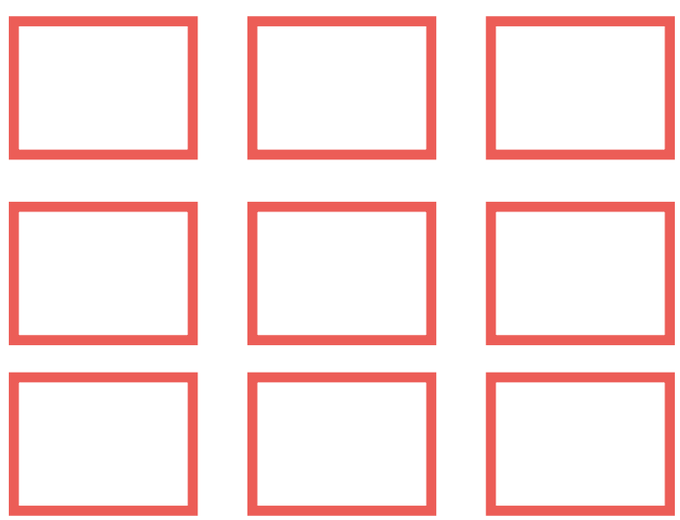
Modules



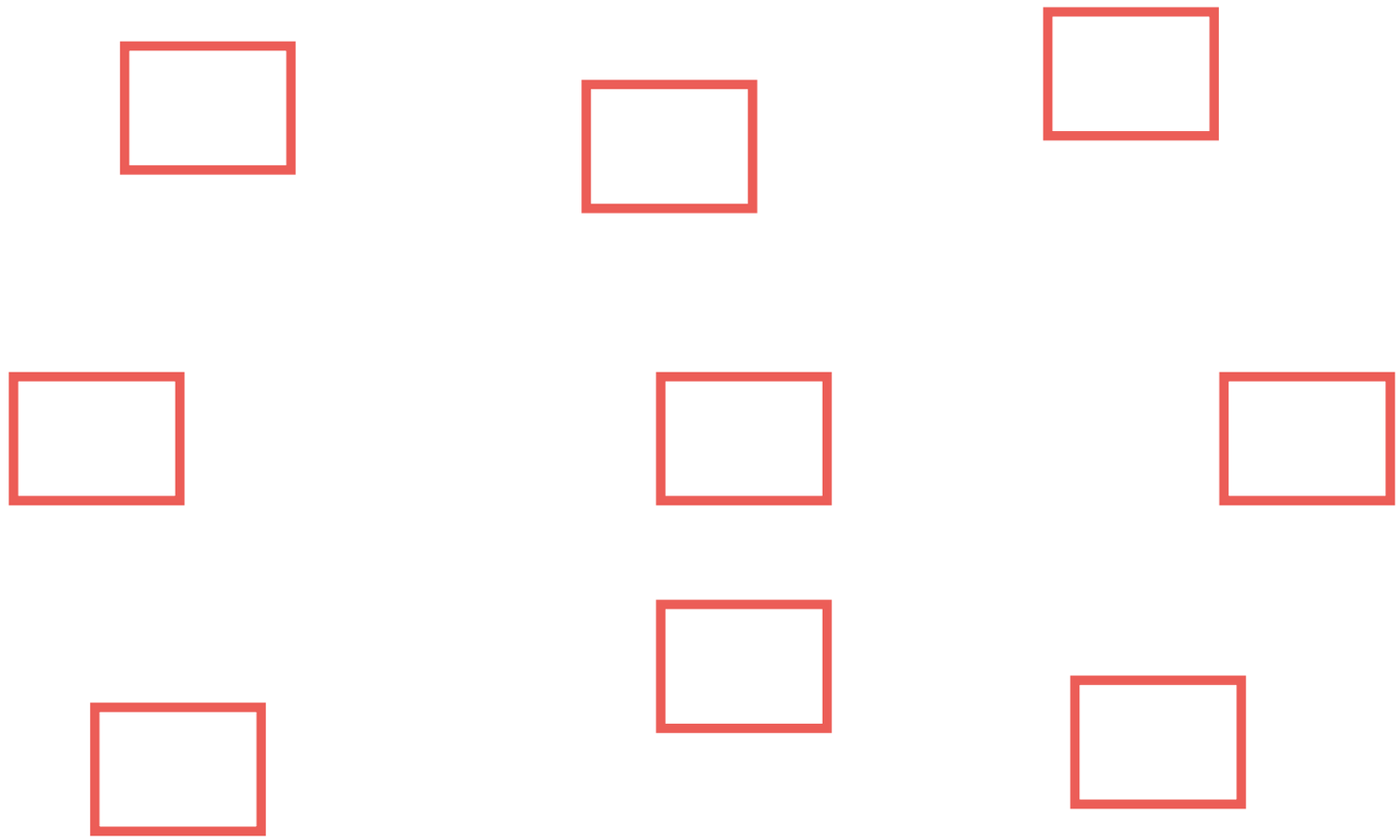
Microservices



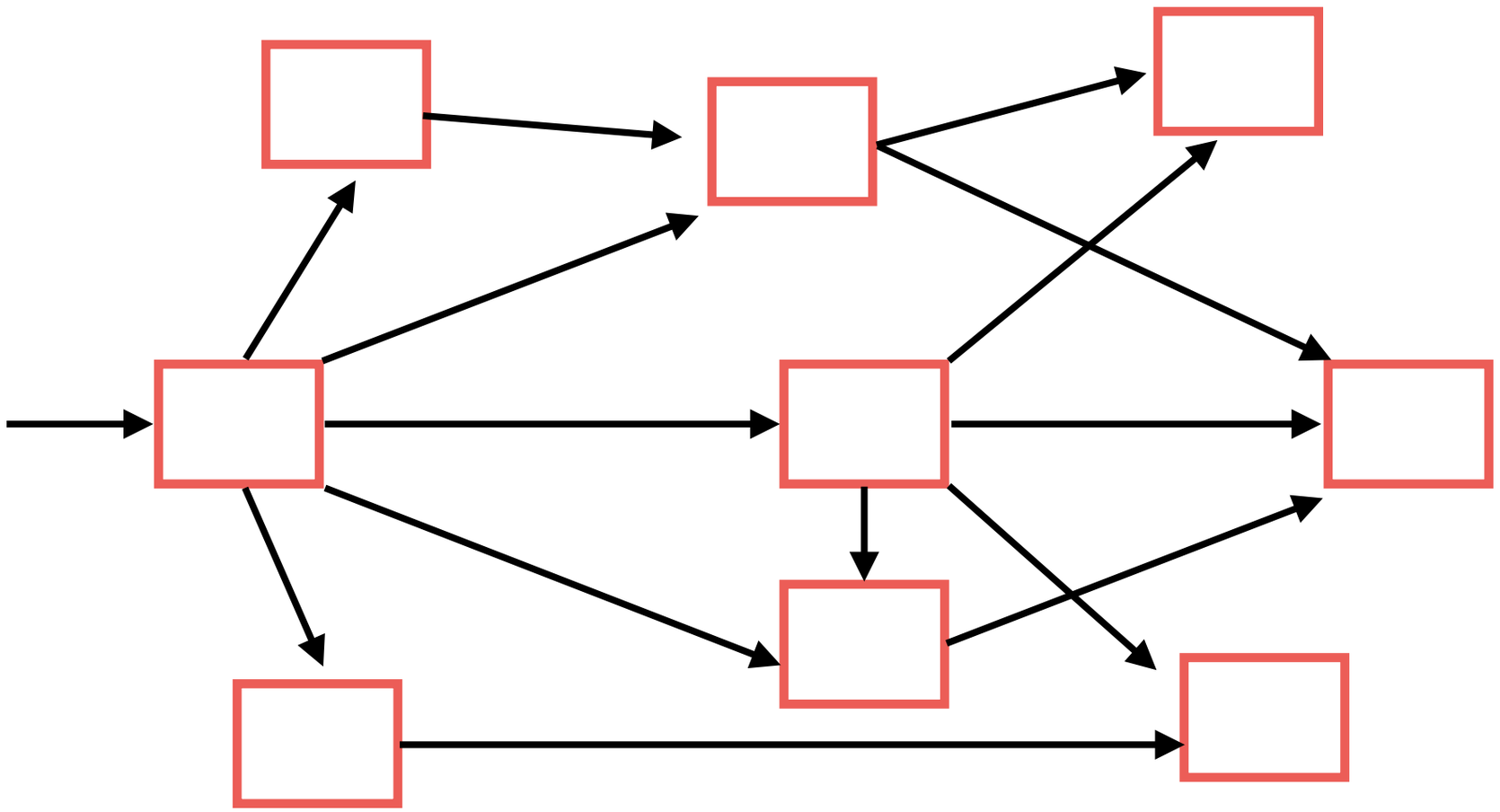
Microservices



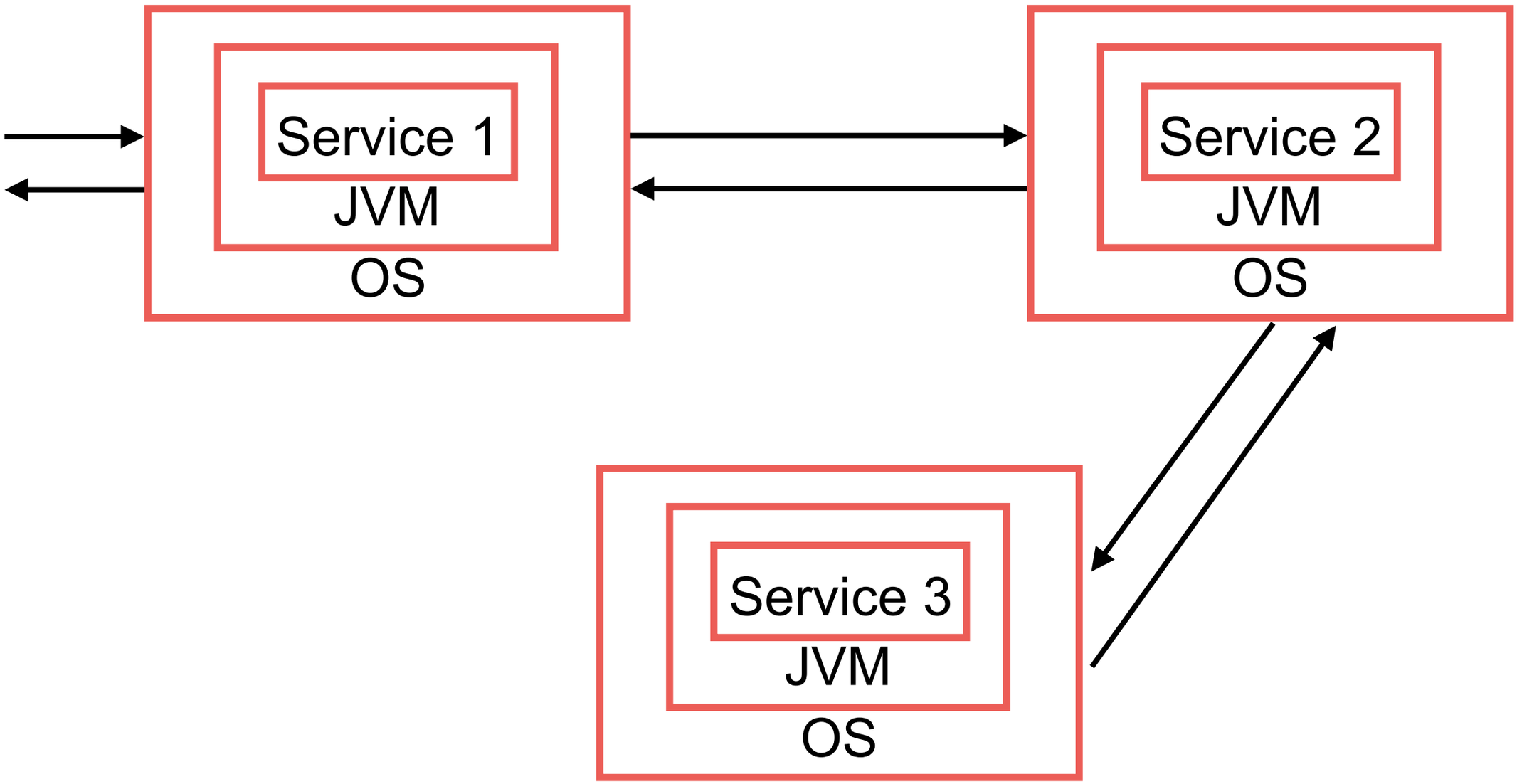
Microservices



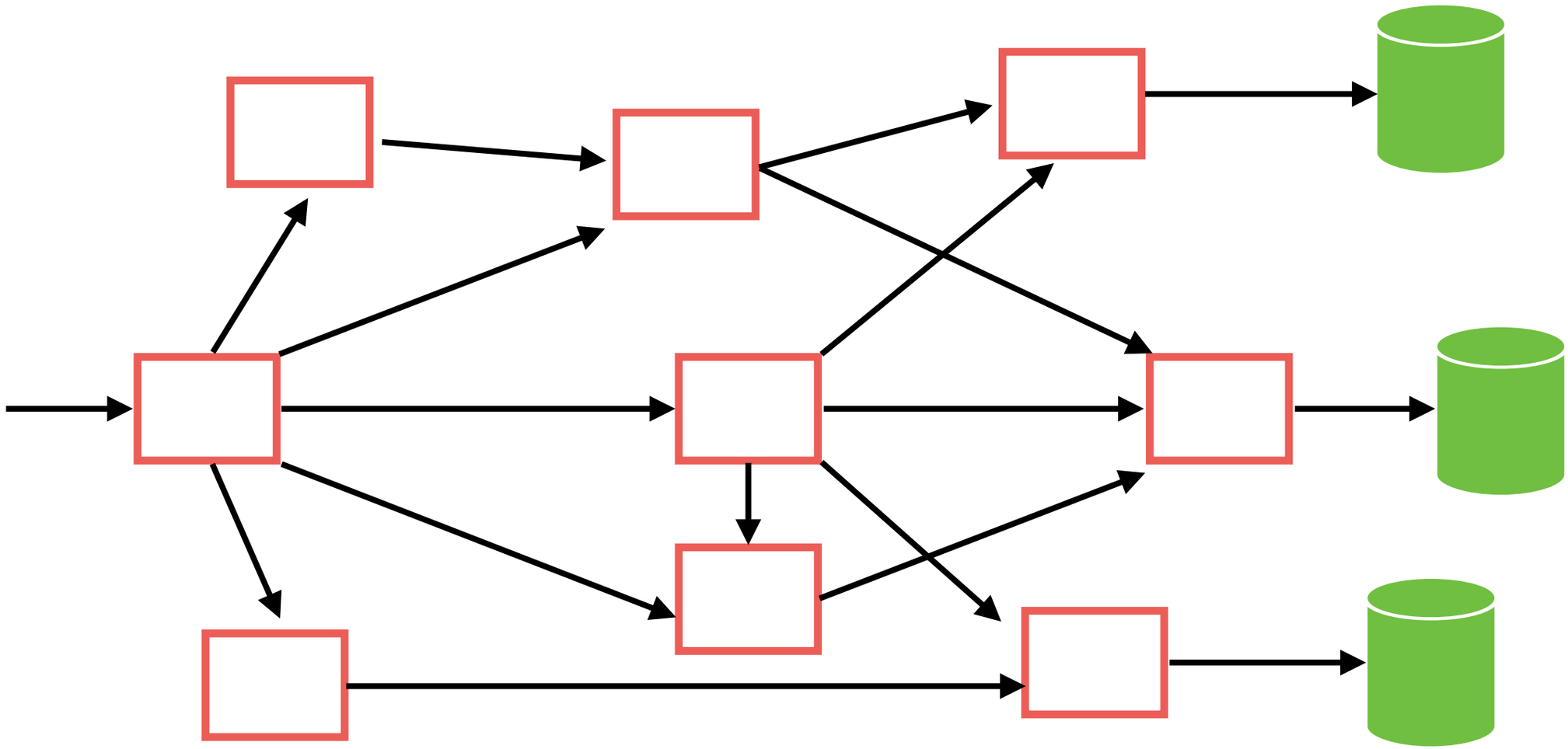
Microservices == Distributed



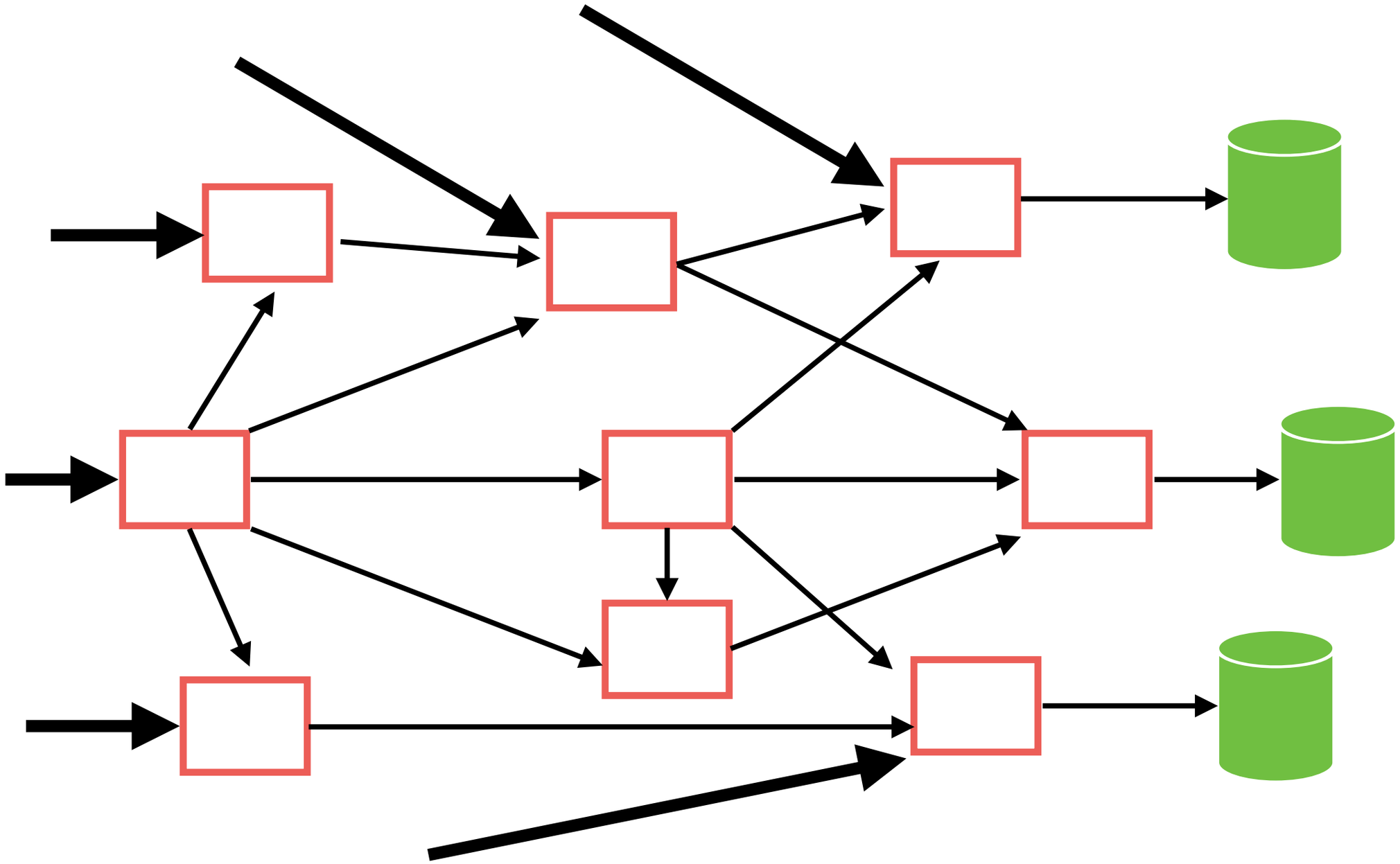
Network of services



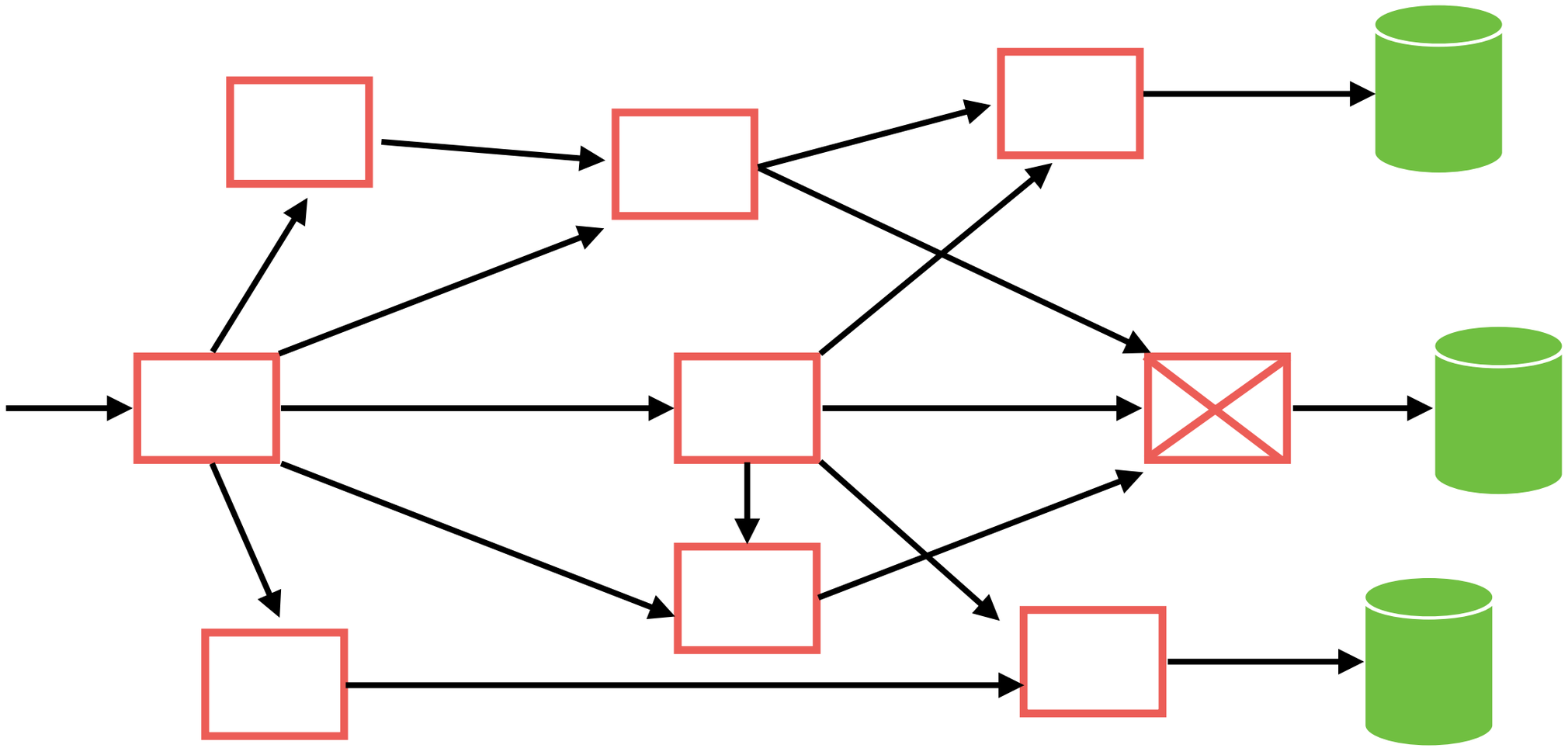
Own data



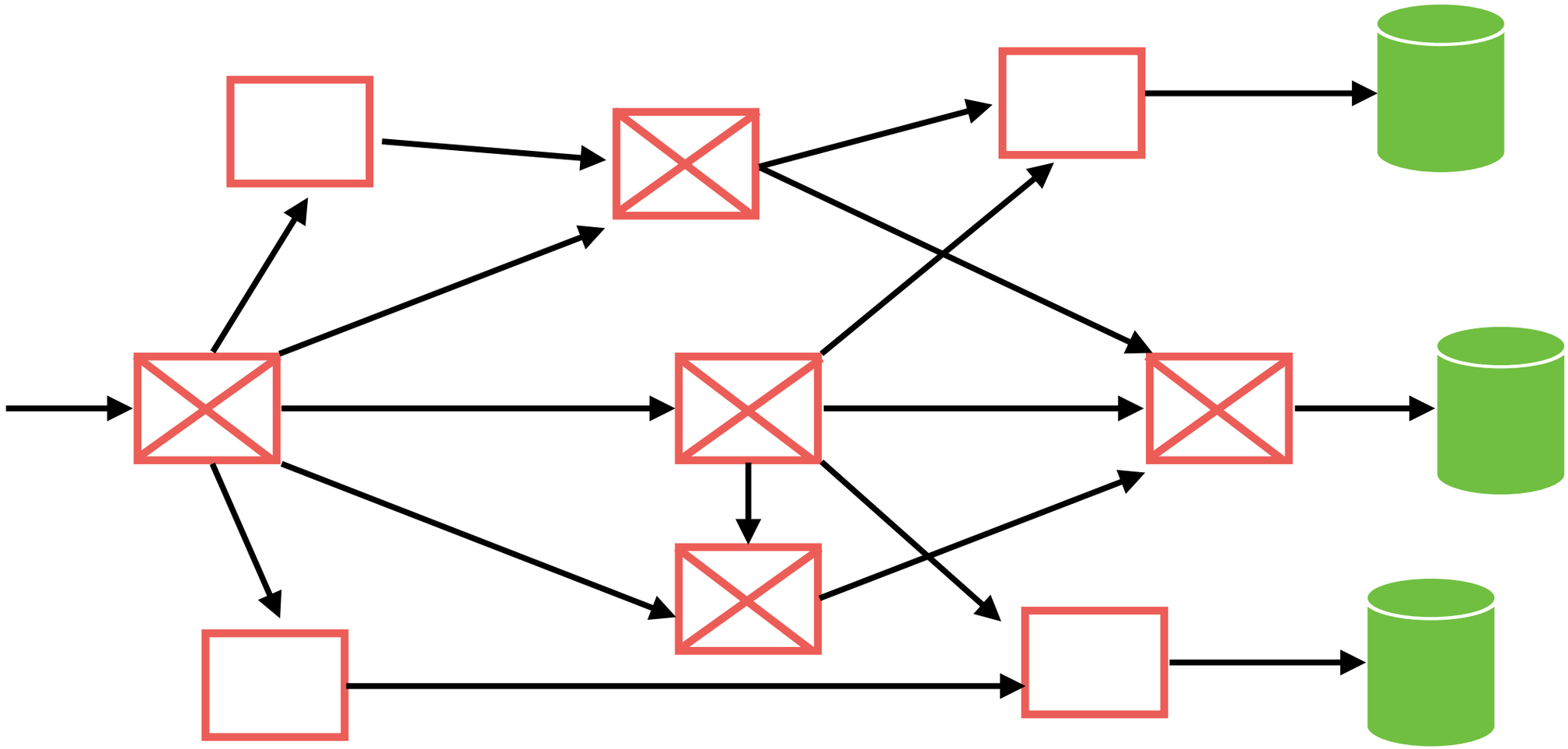
Multiple entry points



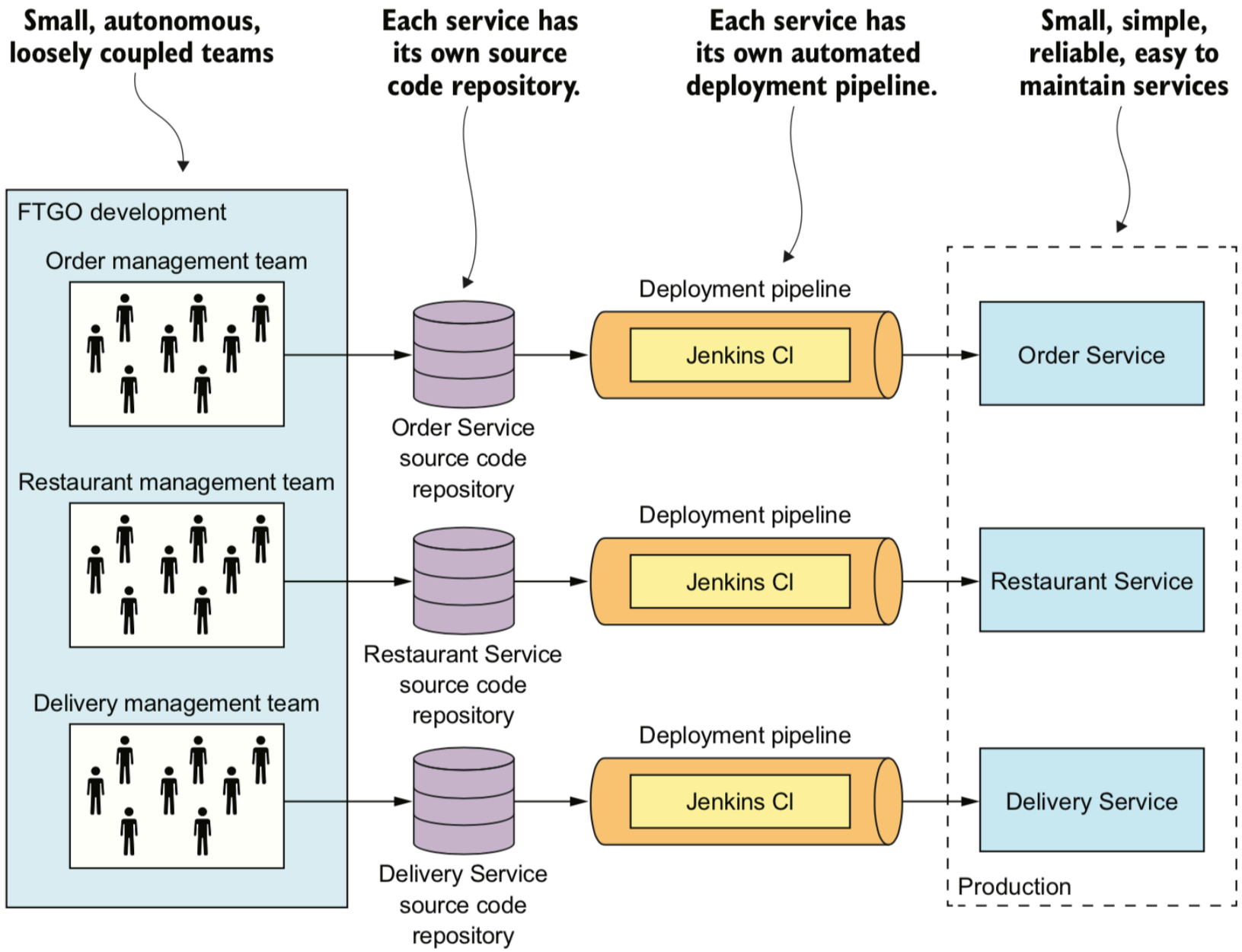
Failure !!



Cascading Failure !!

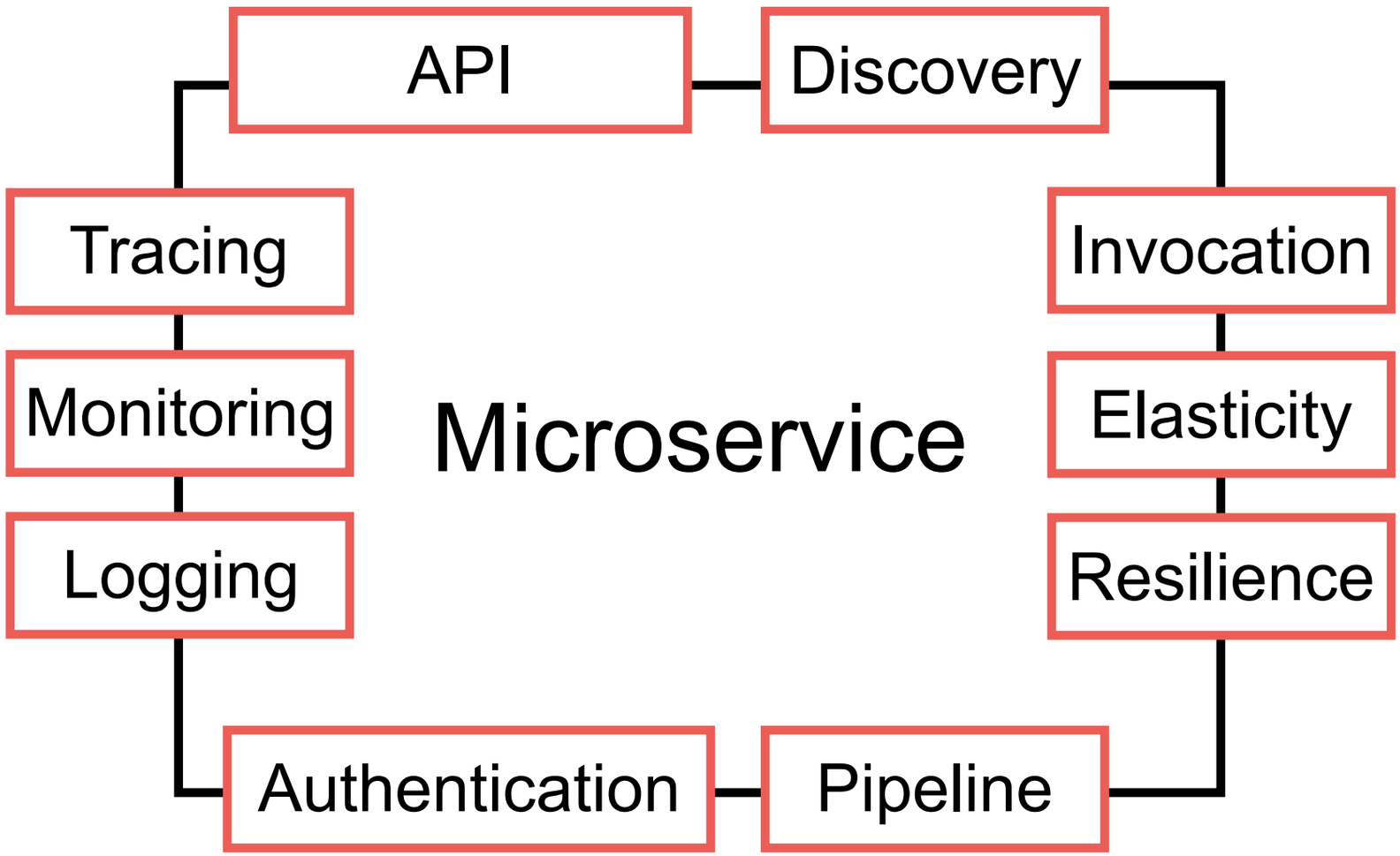


Each team **develop**, test, and deploy their services independently

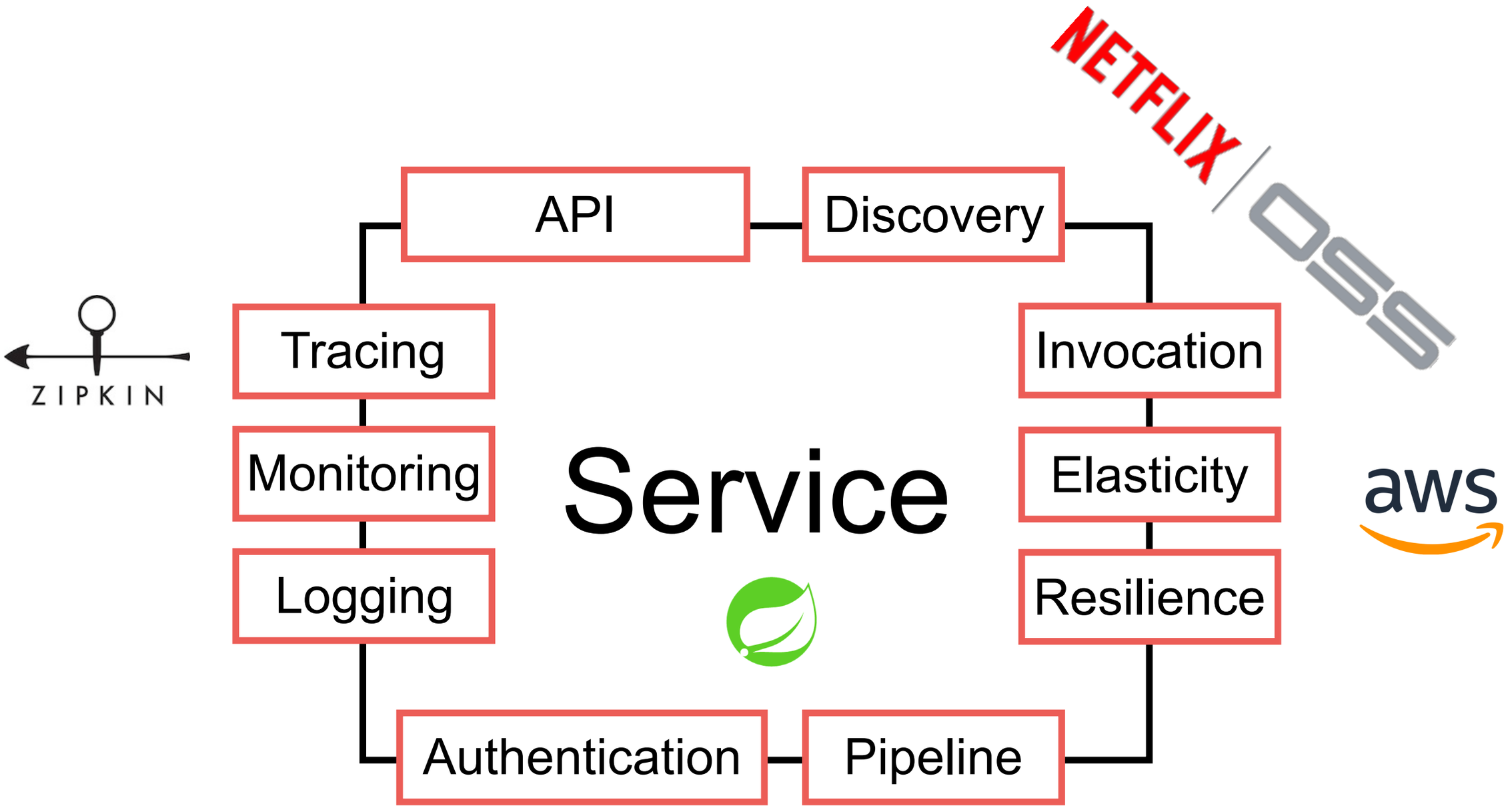


Develop Microservice

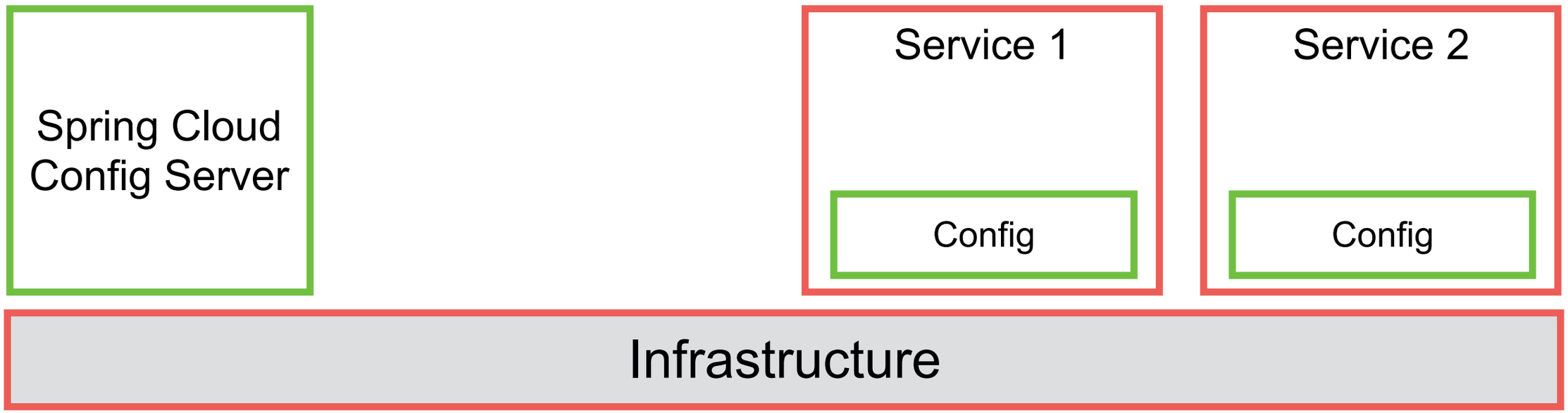
Properties of Microservice



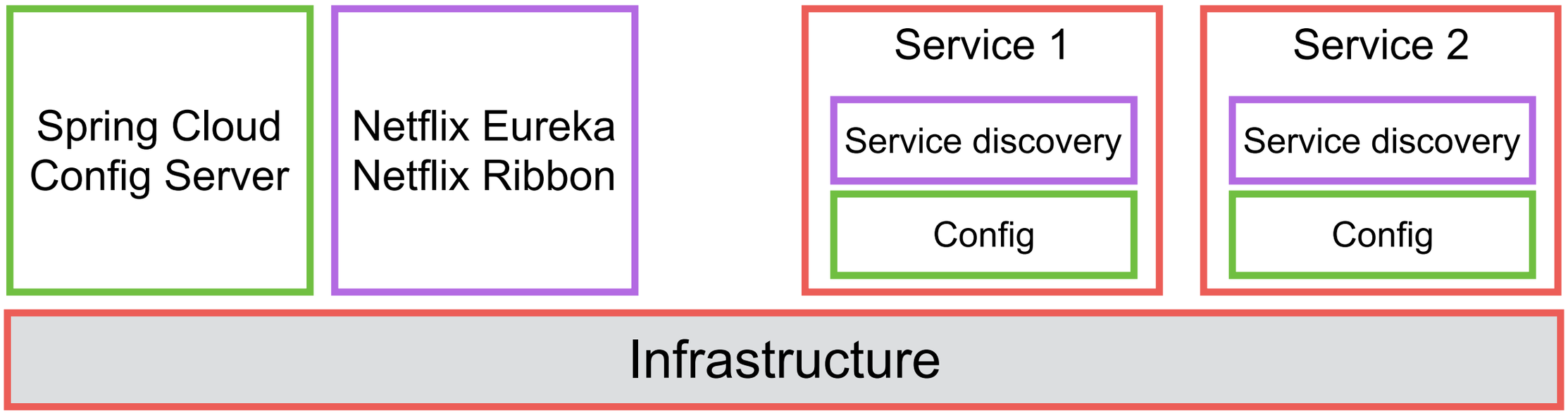
Microservice 1.0



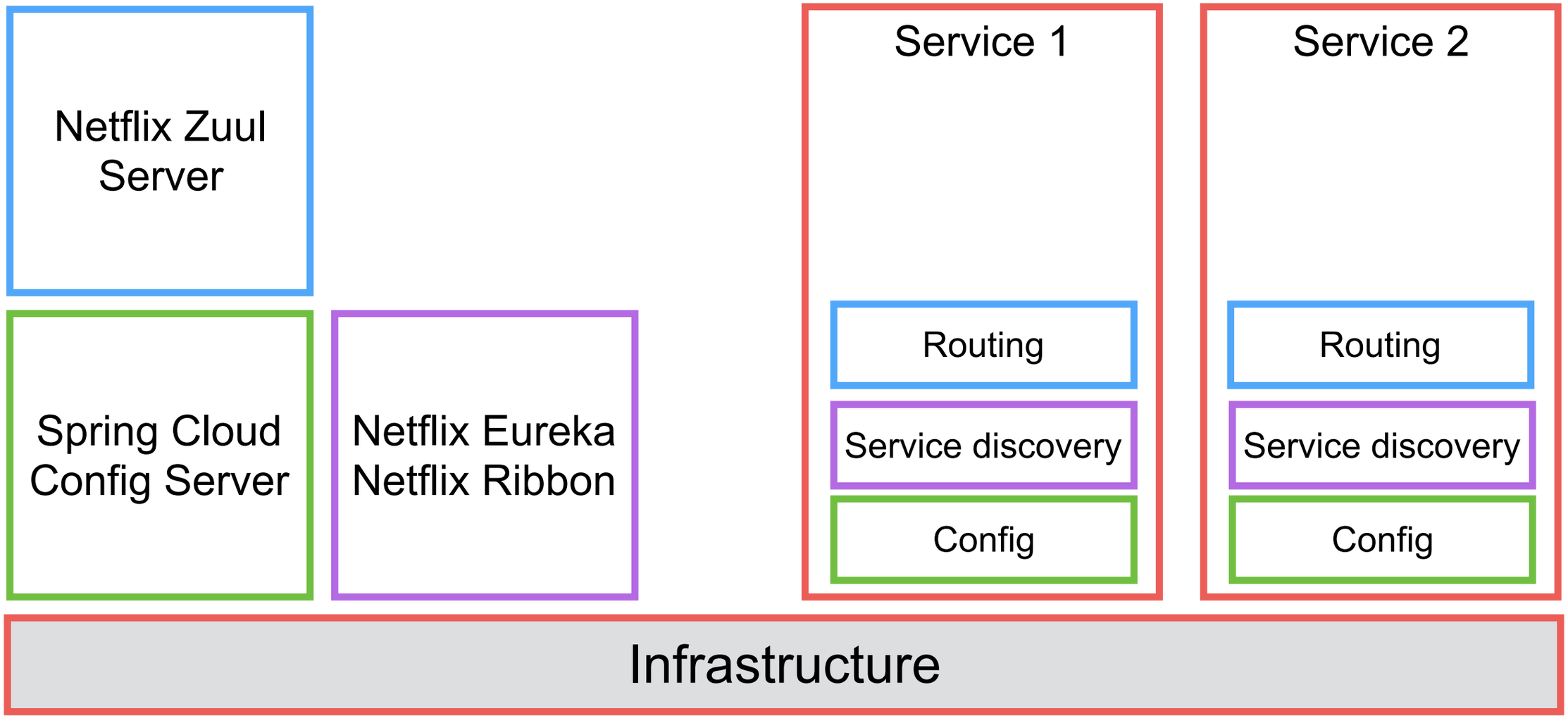
Configuration



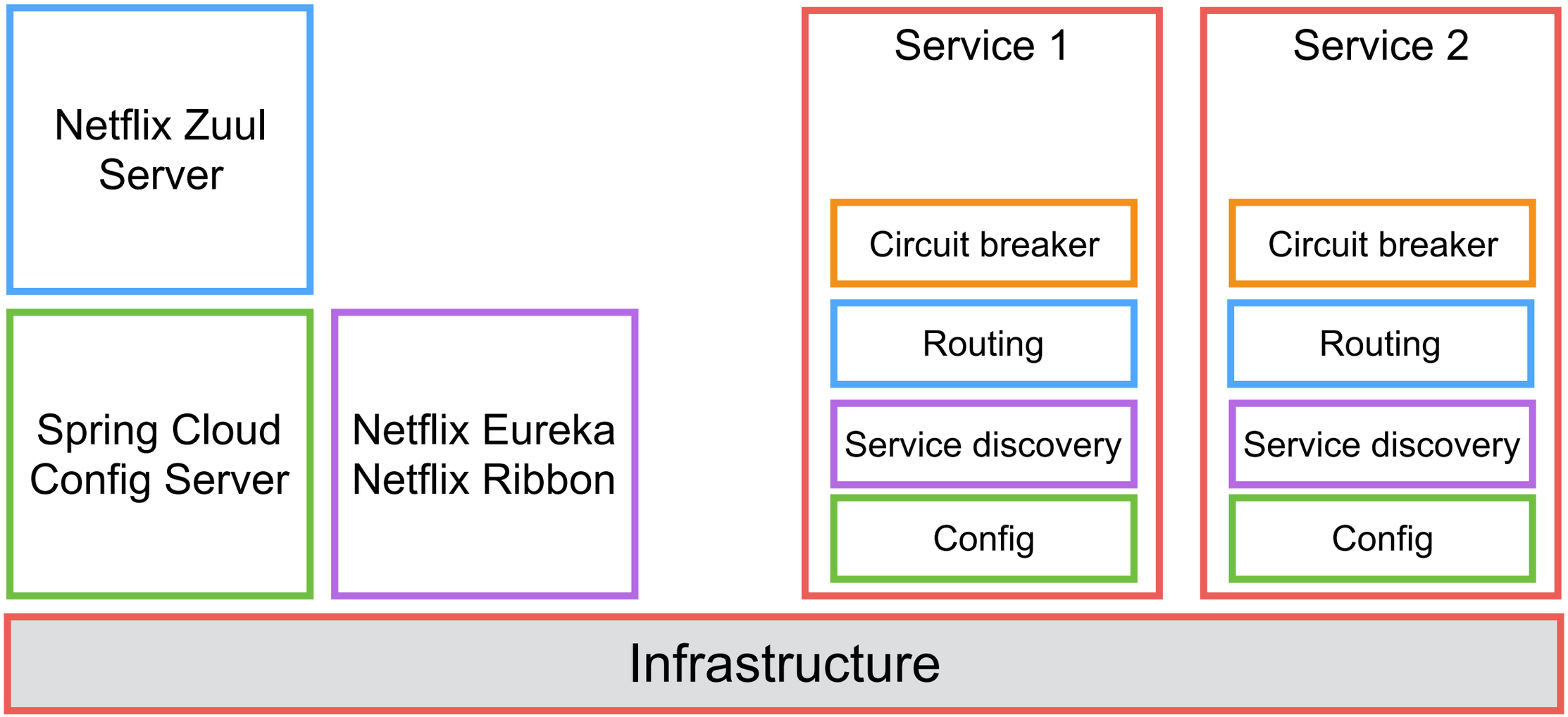
Service Discovery



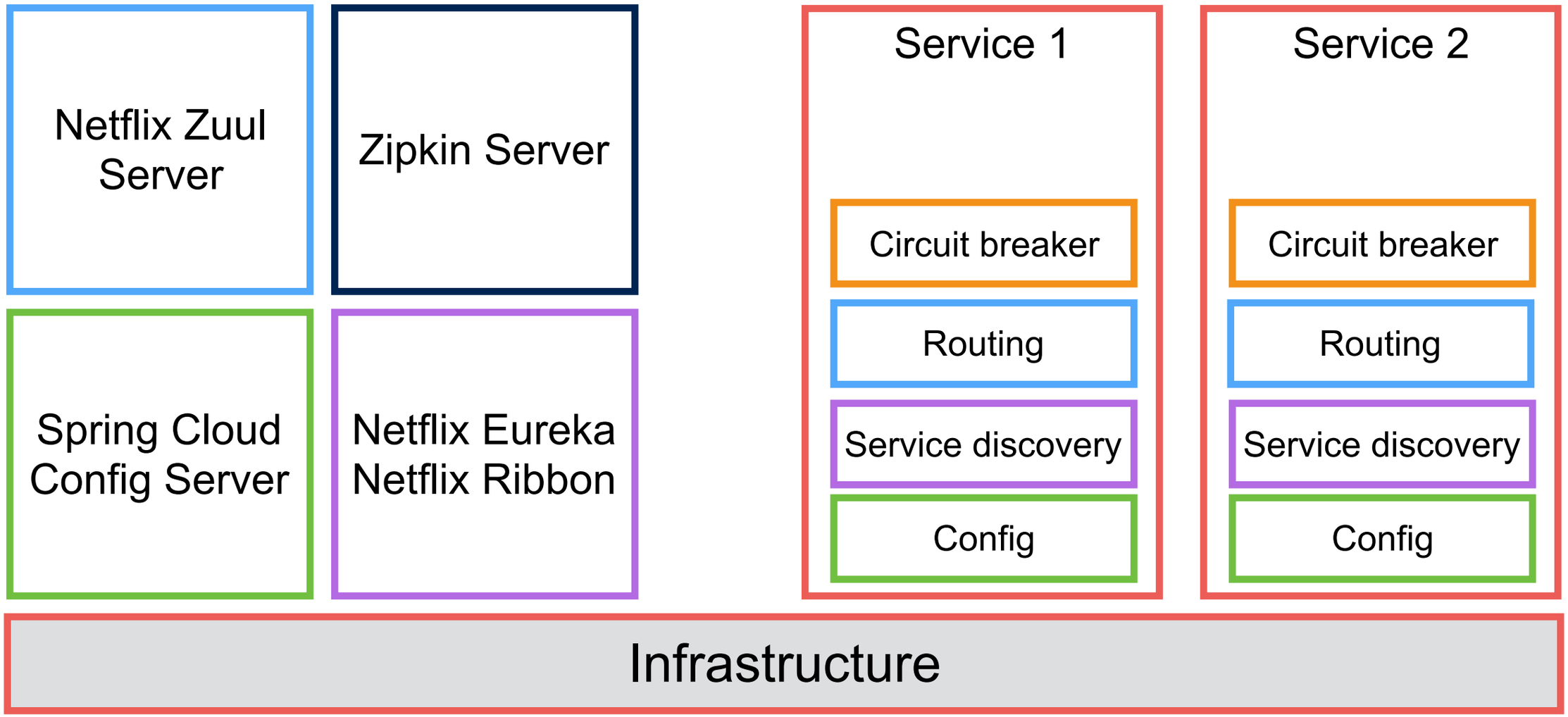
Dynamic Routing



Fault Tolerance



Tracing and Visibility

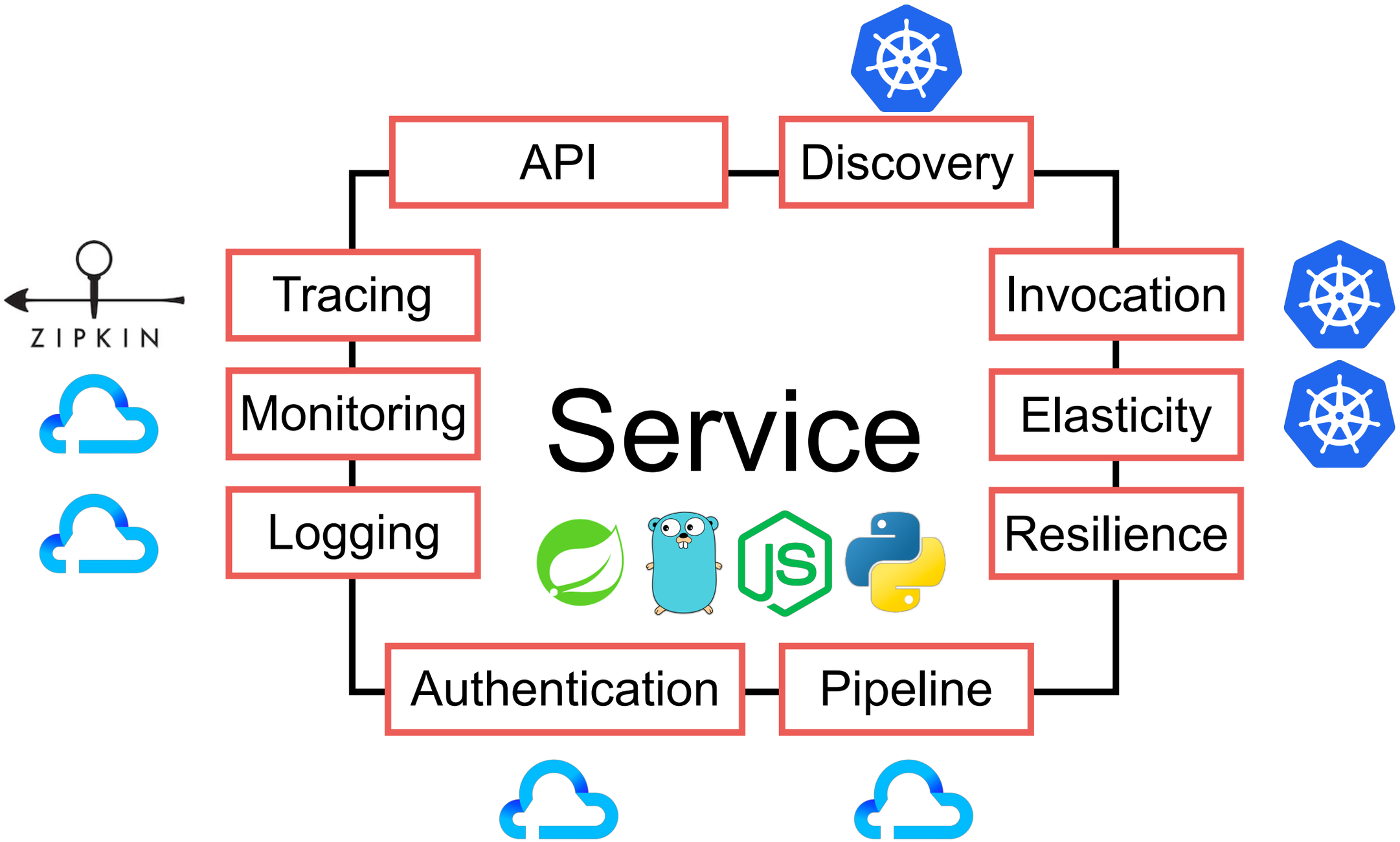


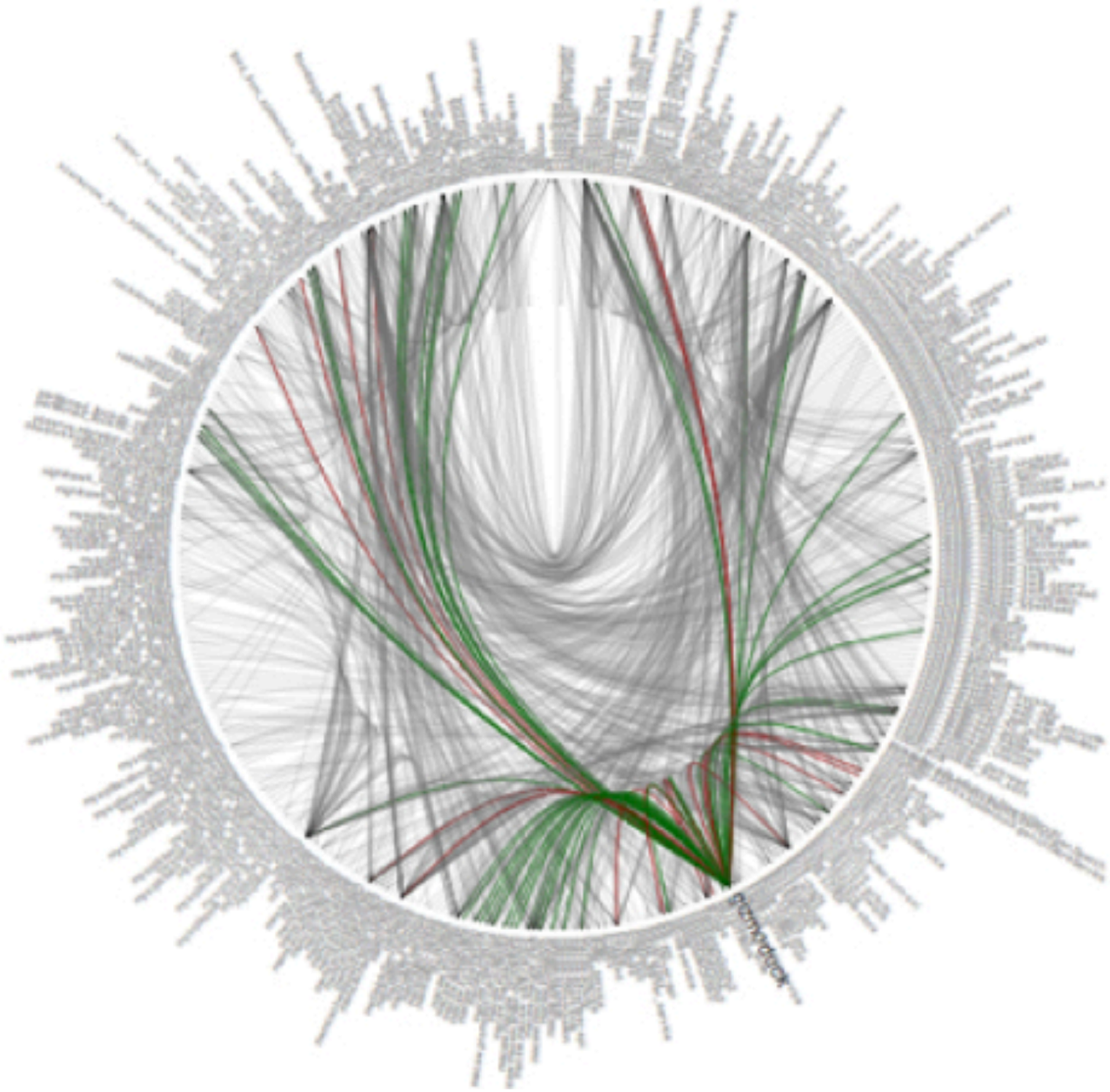
Microservice 1.0

JVM only
Add libraries to your code/service

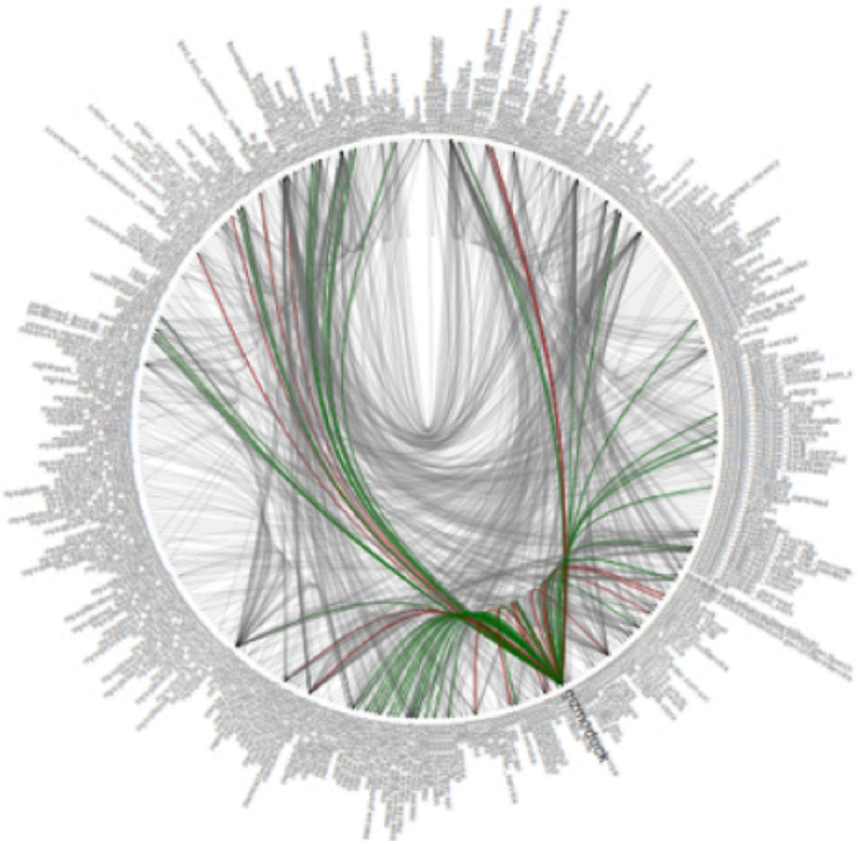
kubernetes

Microservice 2.0



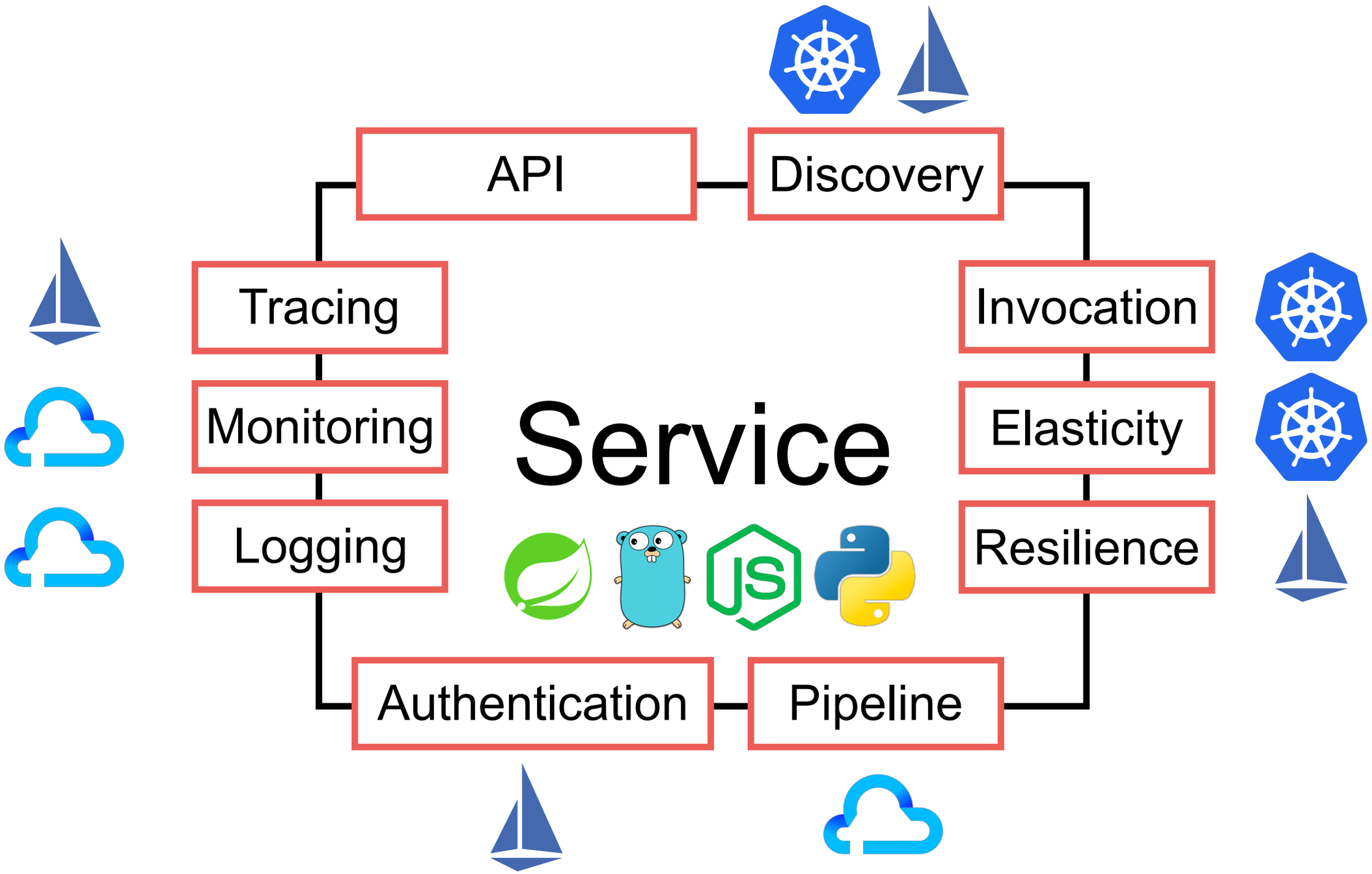


Service Mesh

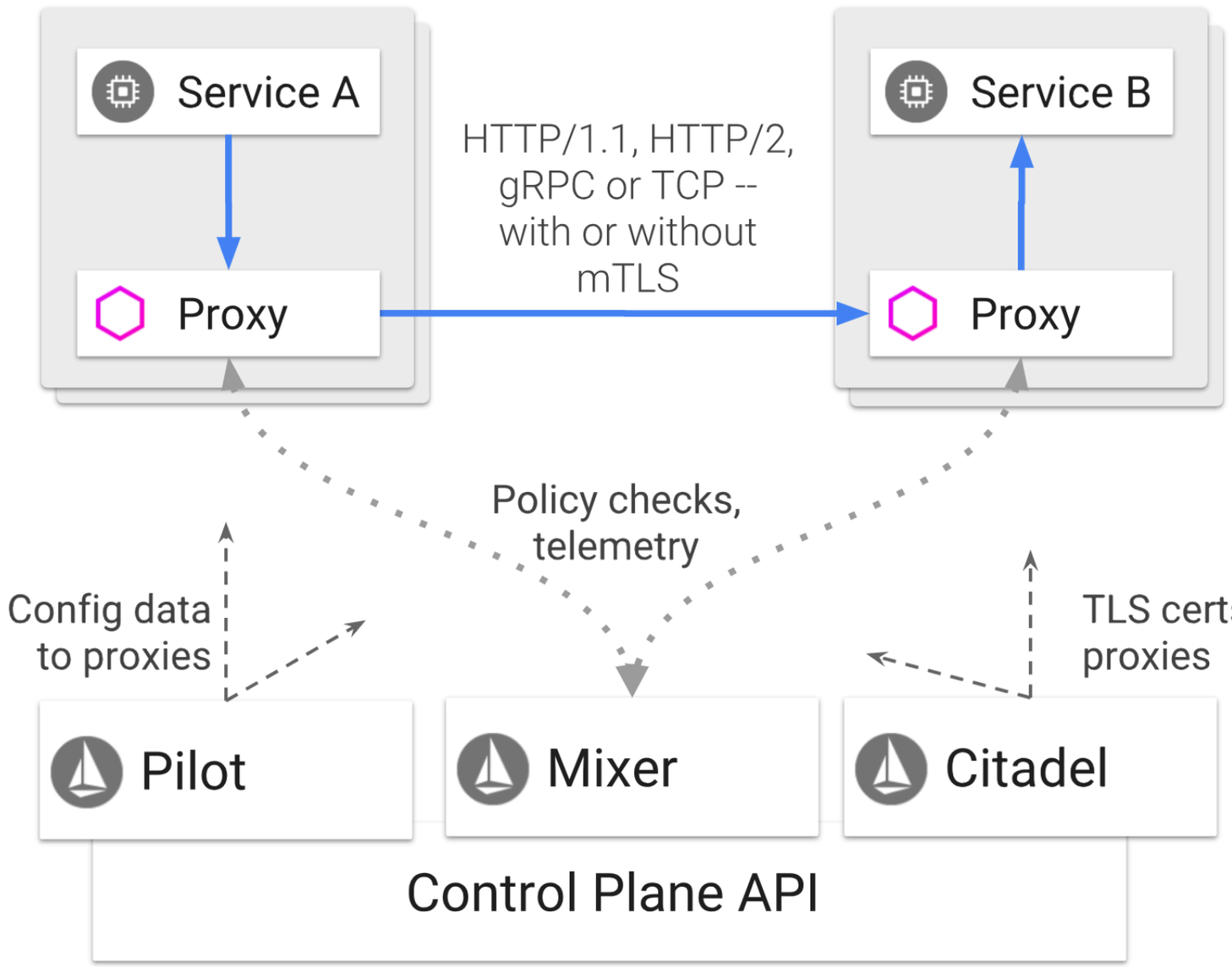


Service Mesh with Istio

Microservice 3.0

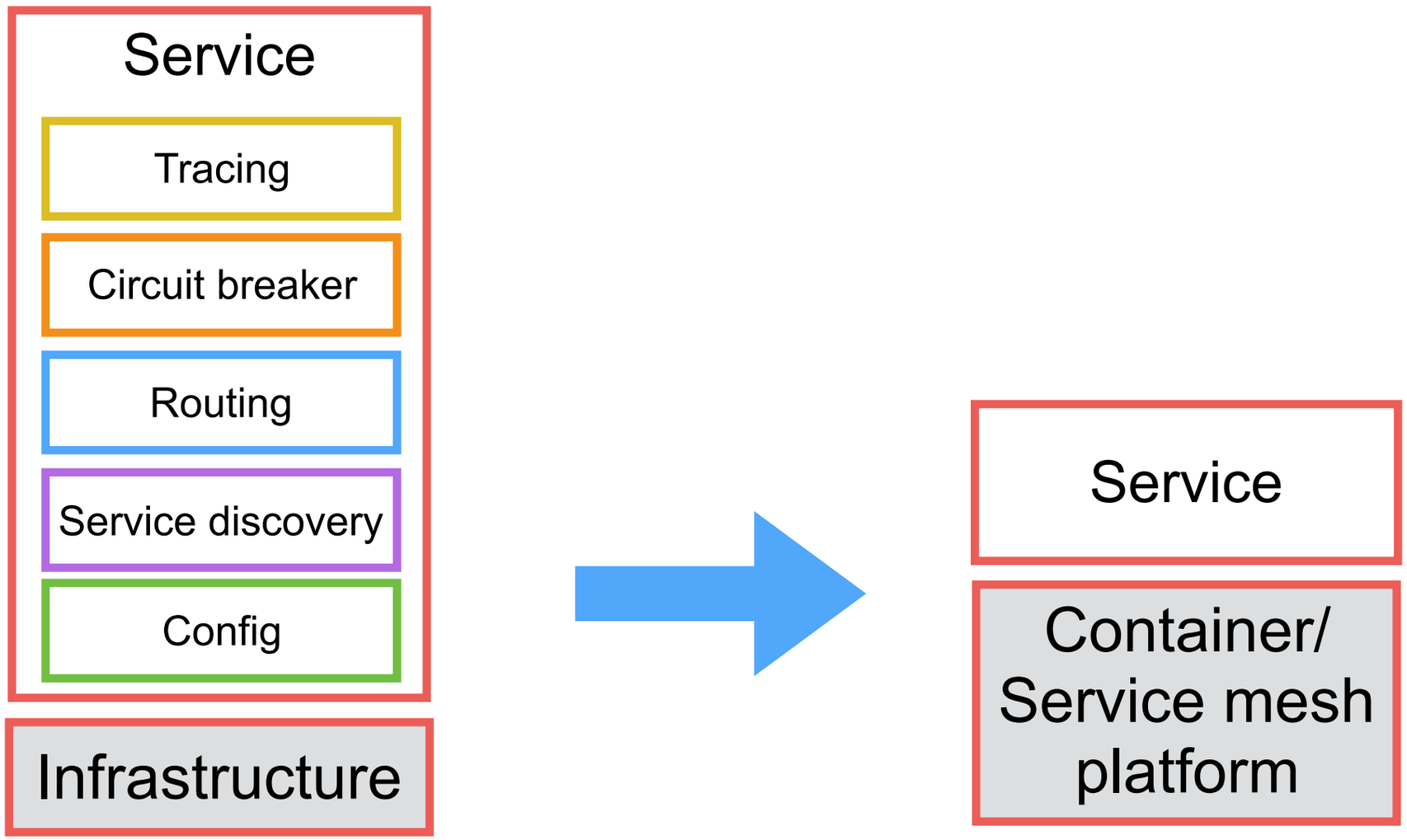


Istio



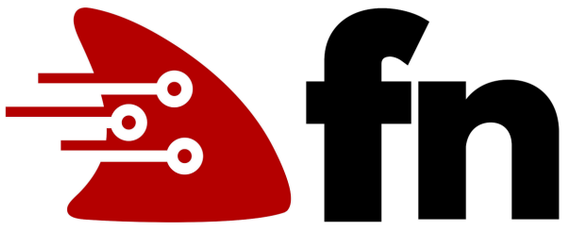
https://istio.io/docs/concepts/what-is-istio/

Microservice Evolution



Function-as-a-Service (FaaS)

Microservice 4.0 == FaaS



OPENFAAS

APACHE
OpenWhisk™

fission

Kubeless

Workshop
Microservice 1.5

Twelve-Factor App

Twelve-Factor App

https://12factor.net/

Initial to build app for Heroku

Principles to cloud and container native app

1. Codebase

Codebase must be tracked in version control and will have many deploy

2. Dependencies

Dependencies are explicitly declared and isolated

Use dependency management tools to shared libraries

3. Configuration

Store configuration in the environment

Add configuration in environment variables or config files

4. Backing services

Treat backing services as an attached resource

Should easy to deploy and change

5. Build, Release, Run

Always have a build and deploy strategy

Build strategies for repeated builds, versioning of running system and rollback

6. Processes

Execute the application as a stateless process

7. Port Binding

Expose services via port bindings

8. Concurrency

Scale out with the process model

9. Disposability

Quick application startup and shutdown times
Graceful shutdown

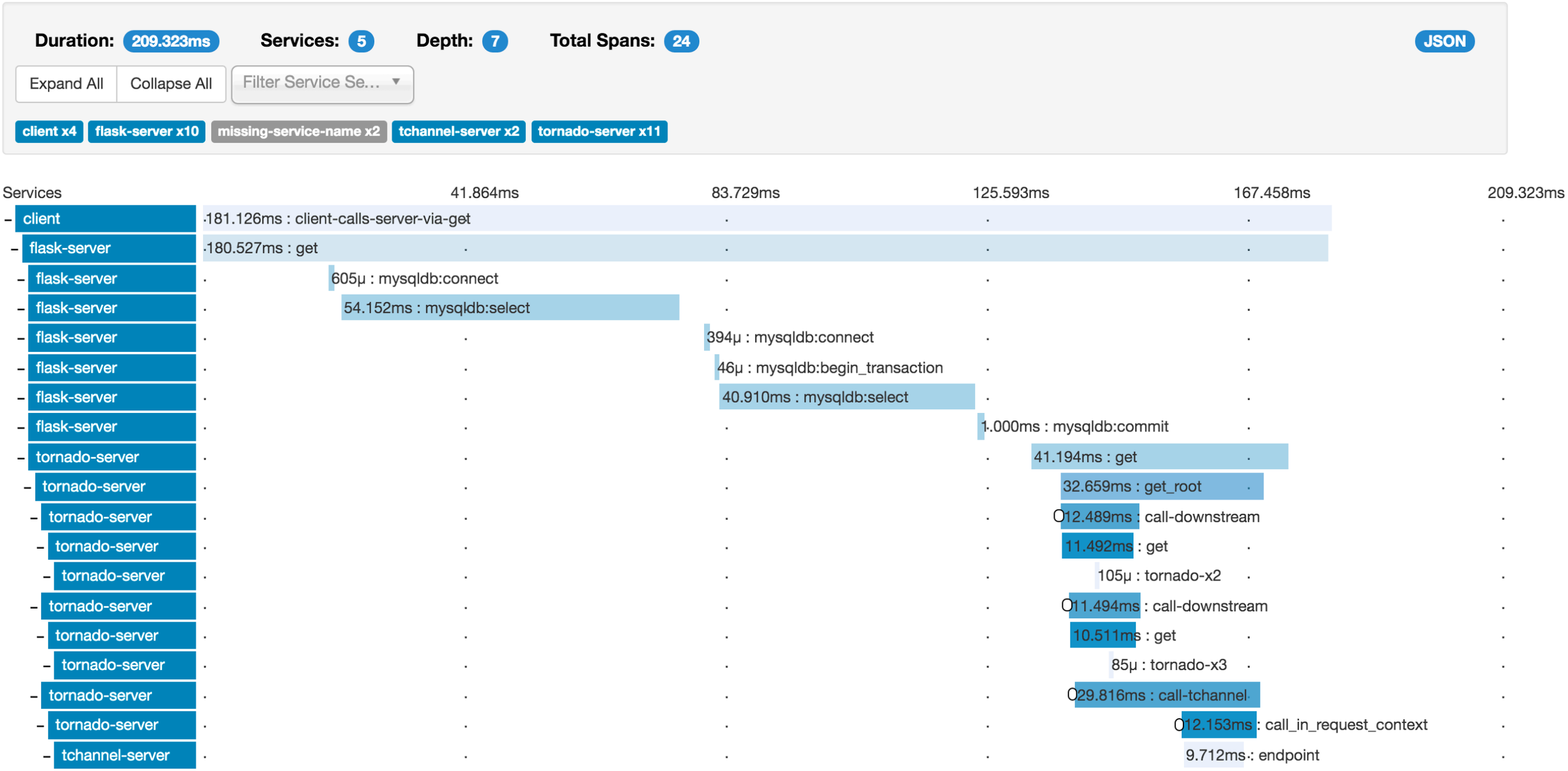
10. Dev/prod parity

Application is treated the same way in dev, staging and production

11. Log management

Treated as an event stream

Logging/Tracing



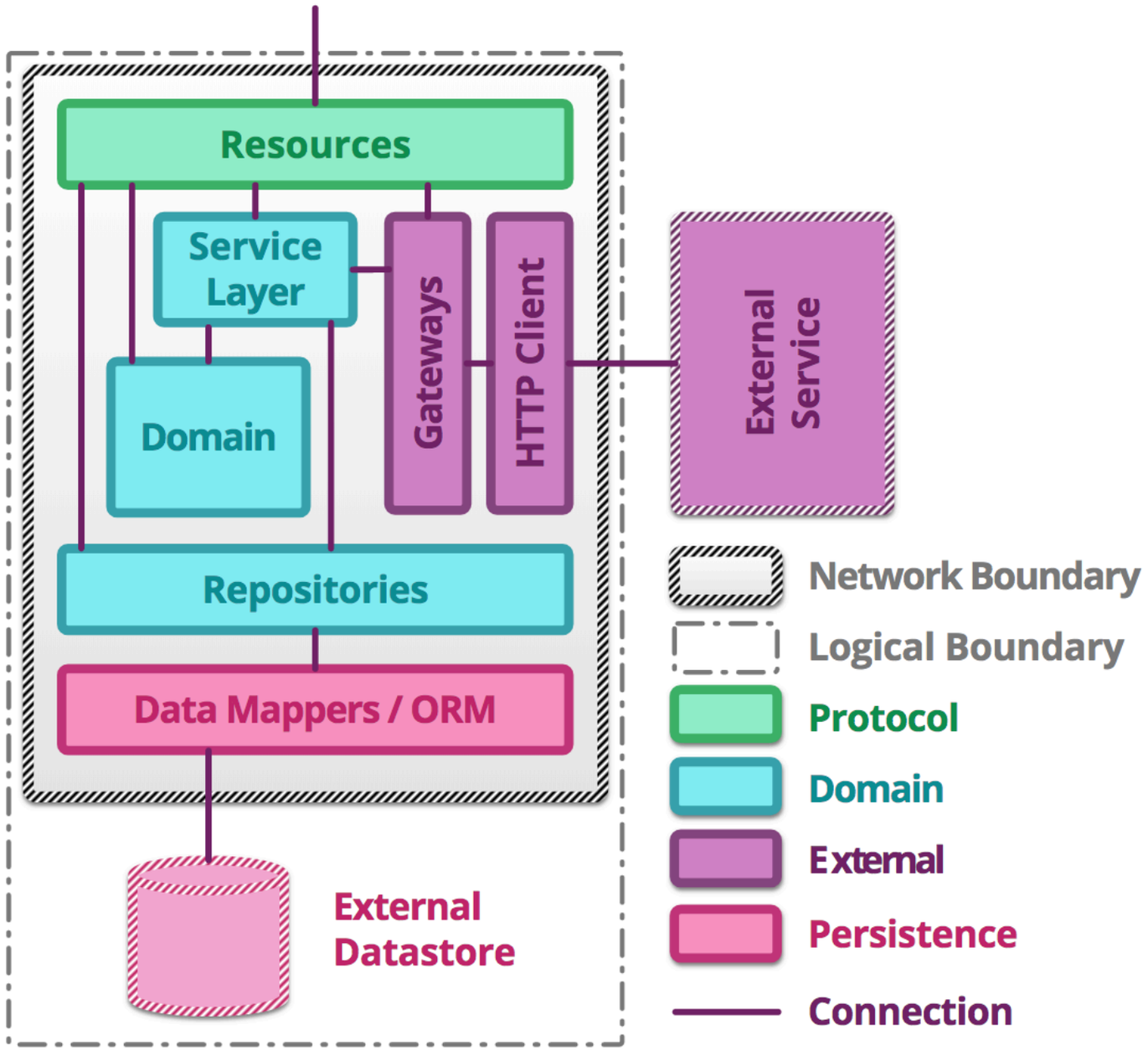
https://zipkin.io/

12. Admin tasks

Treated the same way like the rest of the application

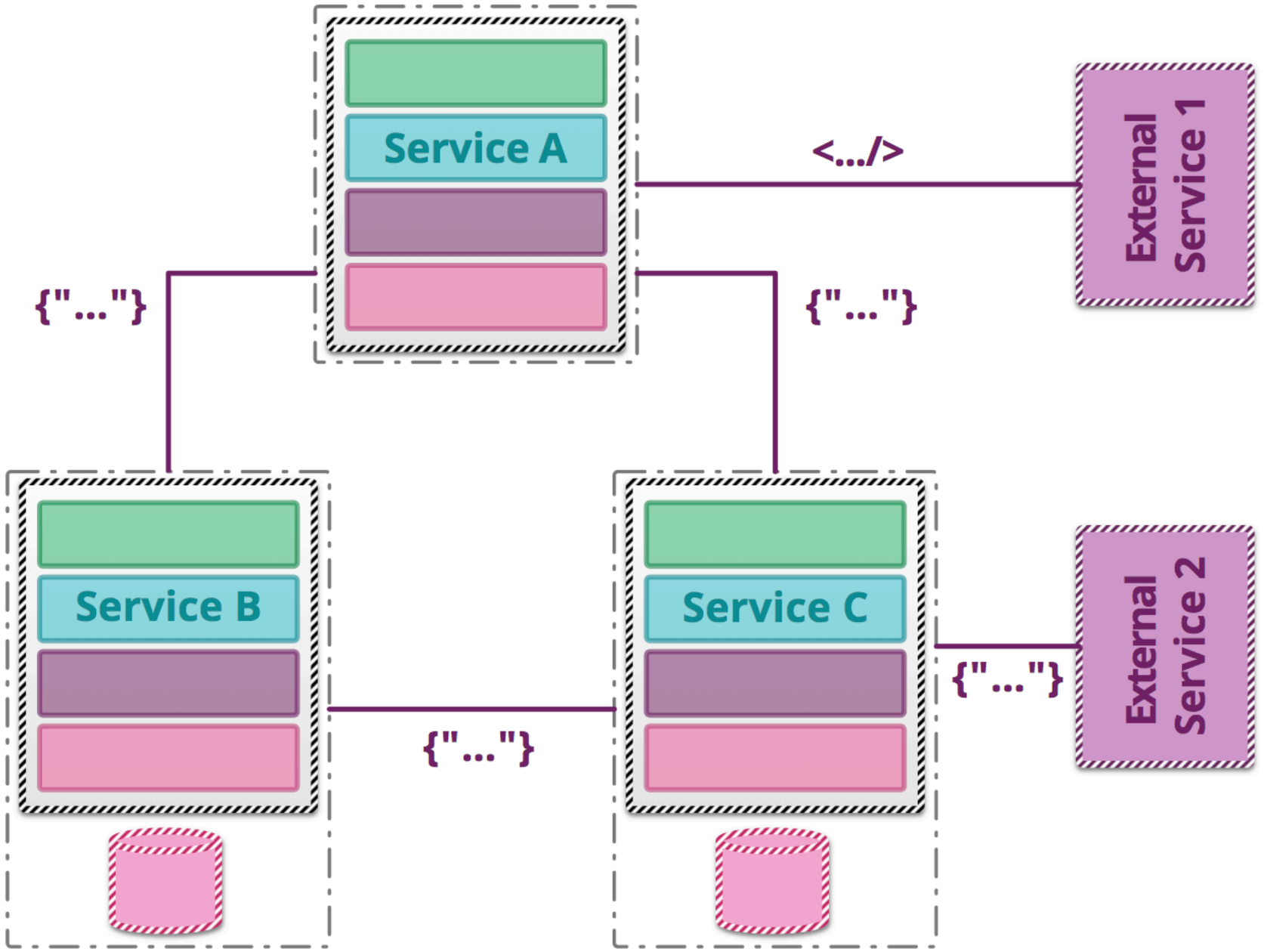
Microservice Testing

Service structure



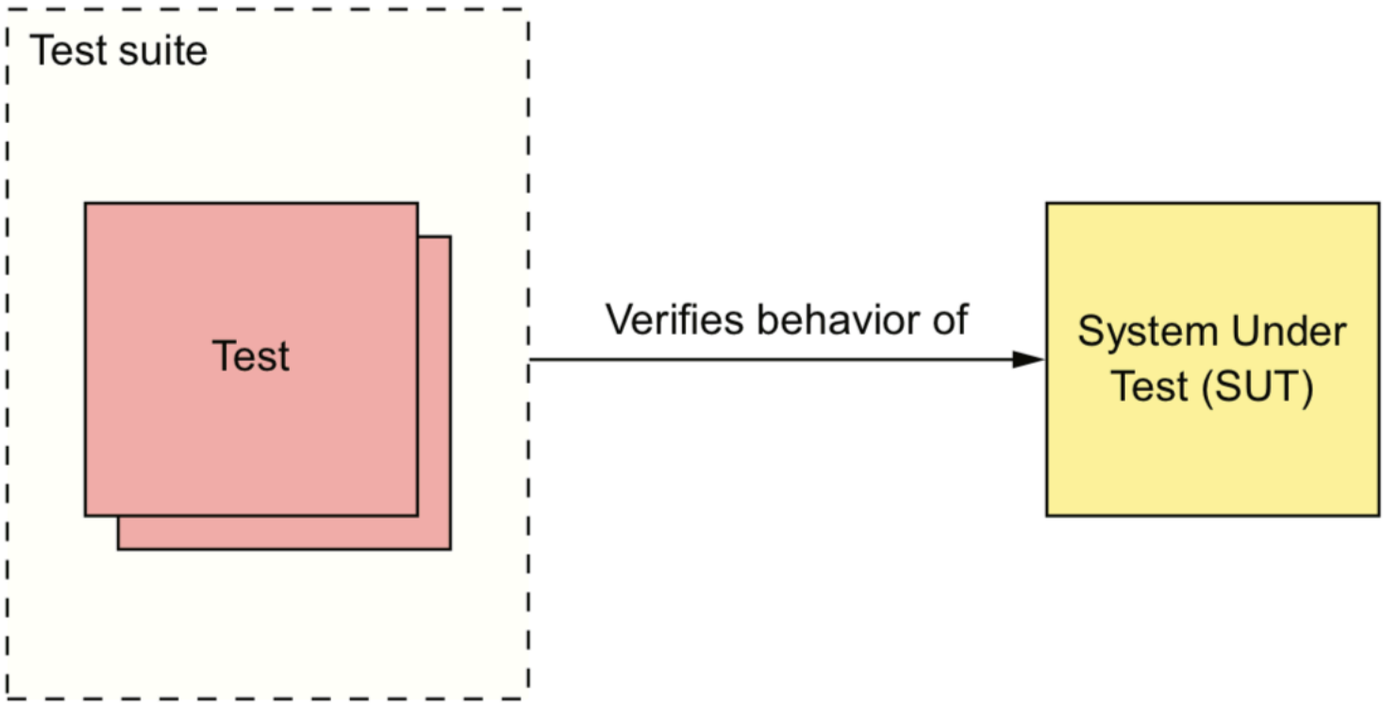
https://martinfowler.com/articles/microservice-testing

Multiple services

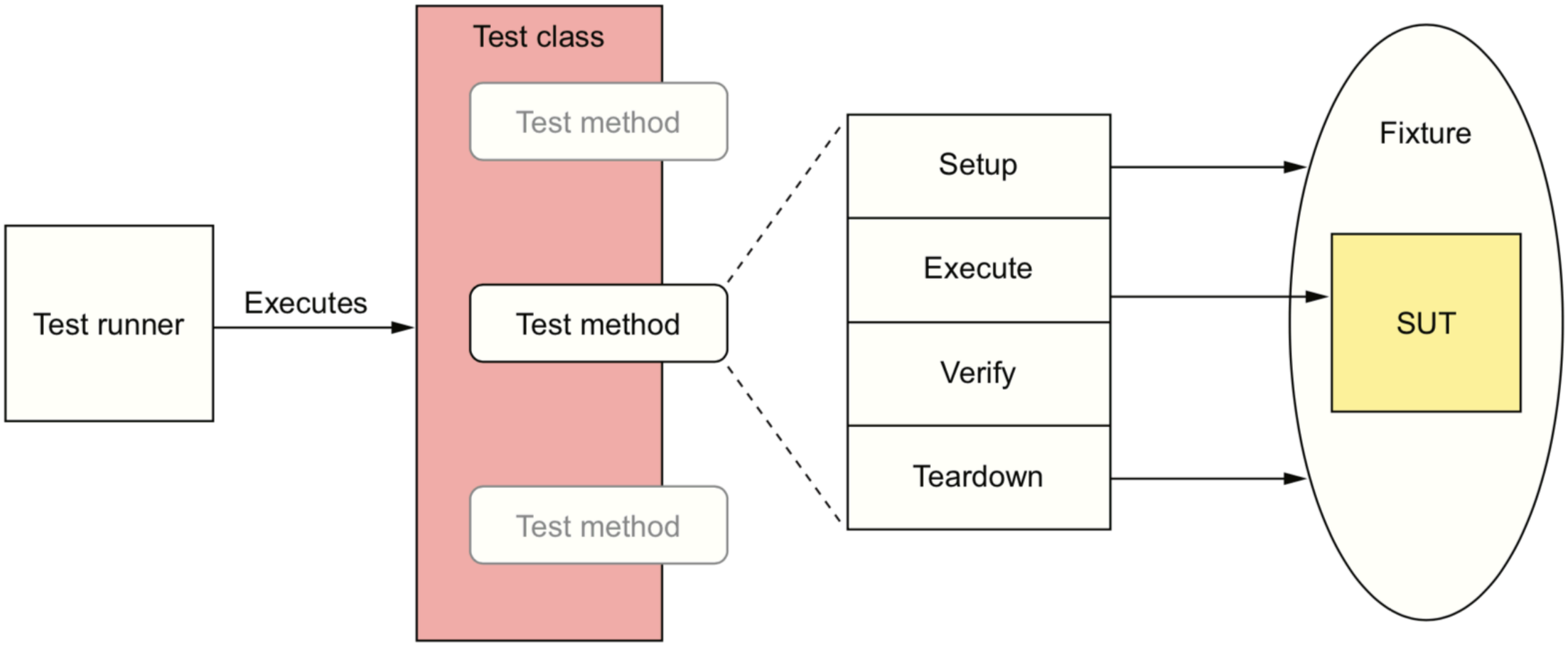


https://martinfowler.com/articles/microservice-testing

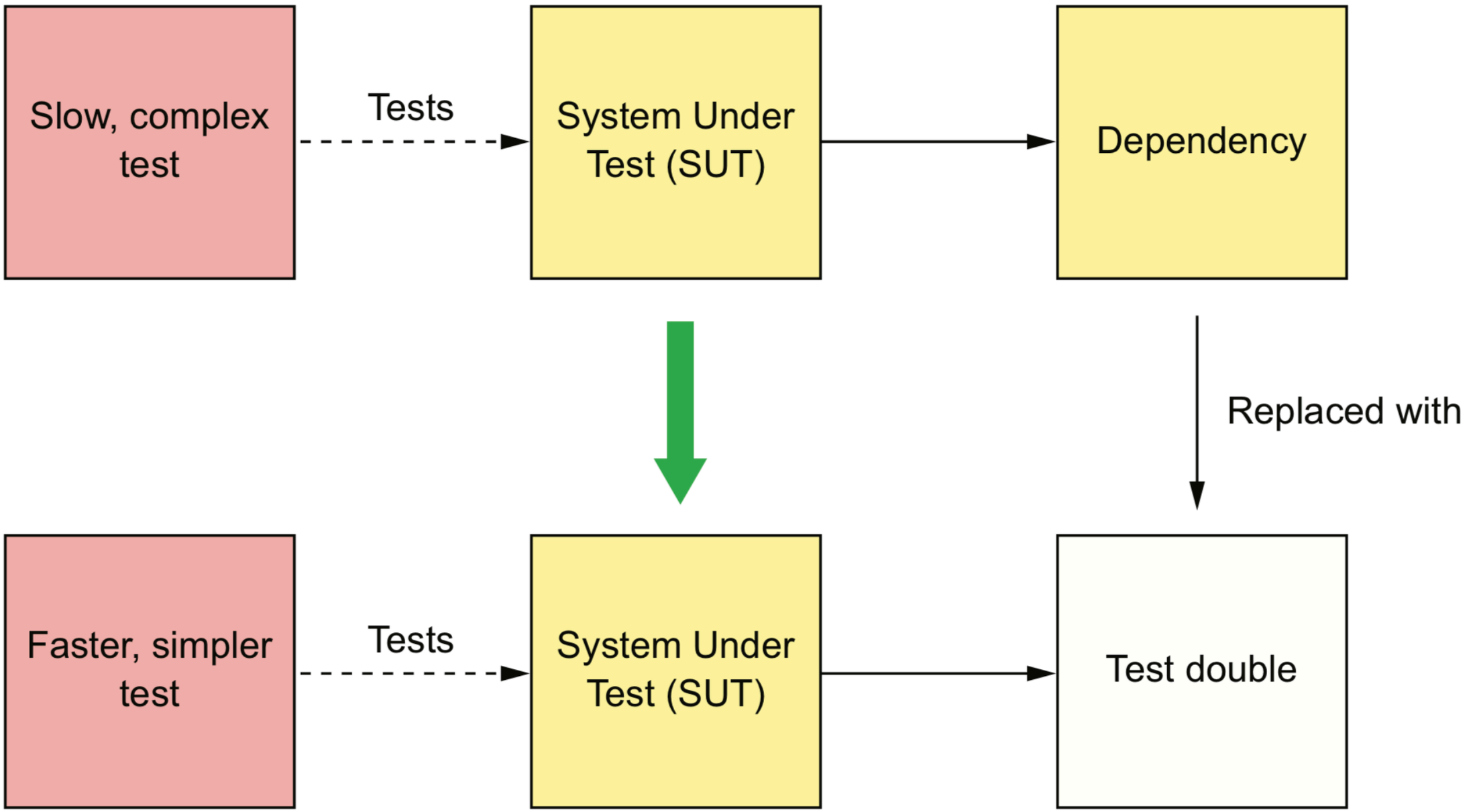
Testing

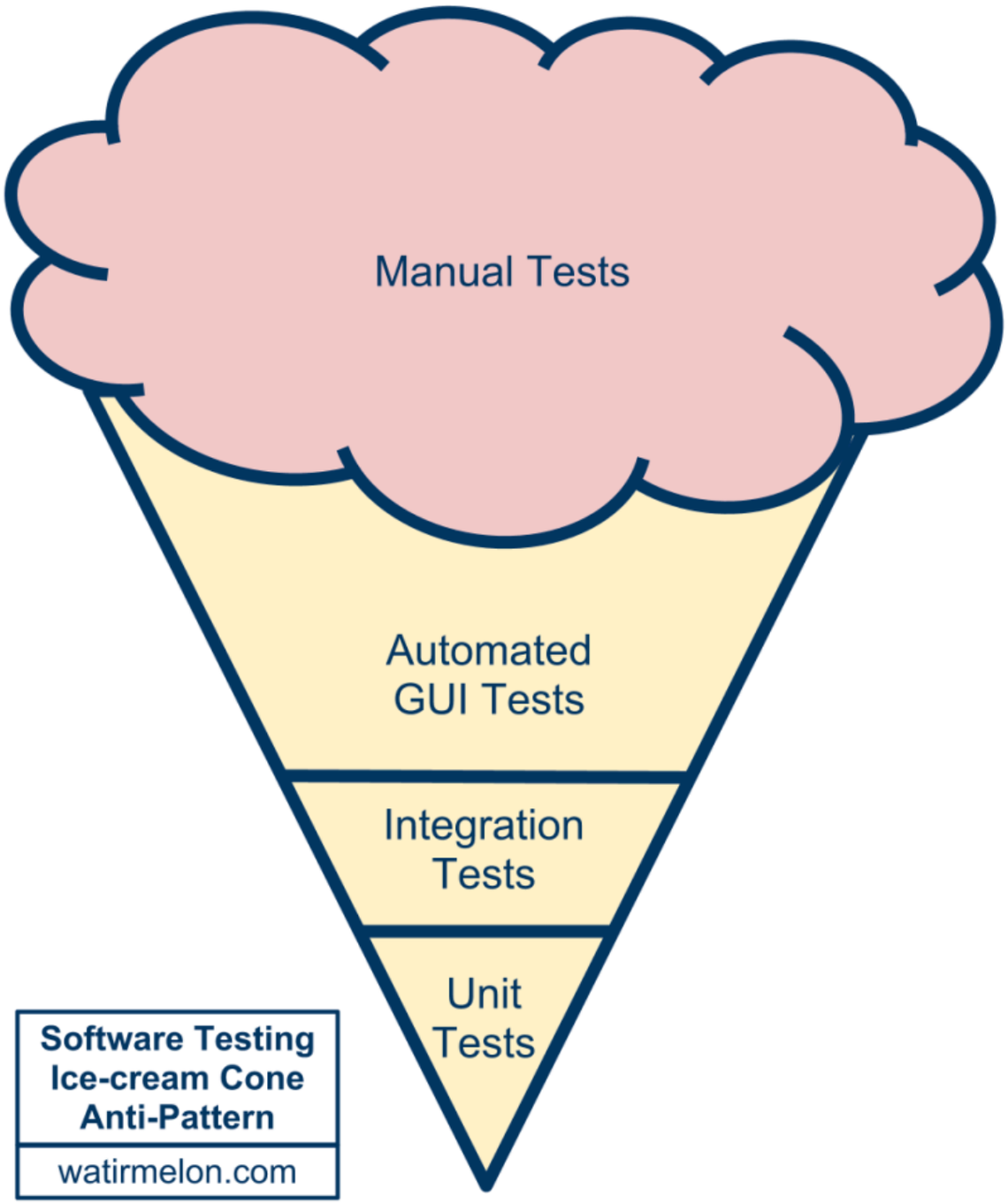


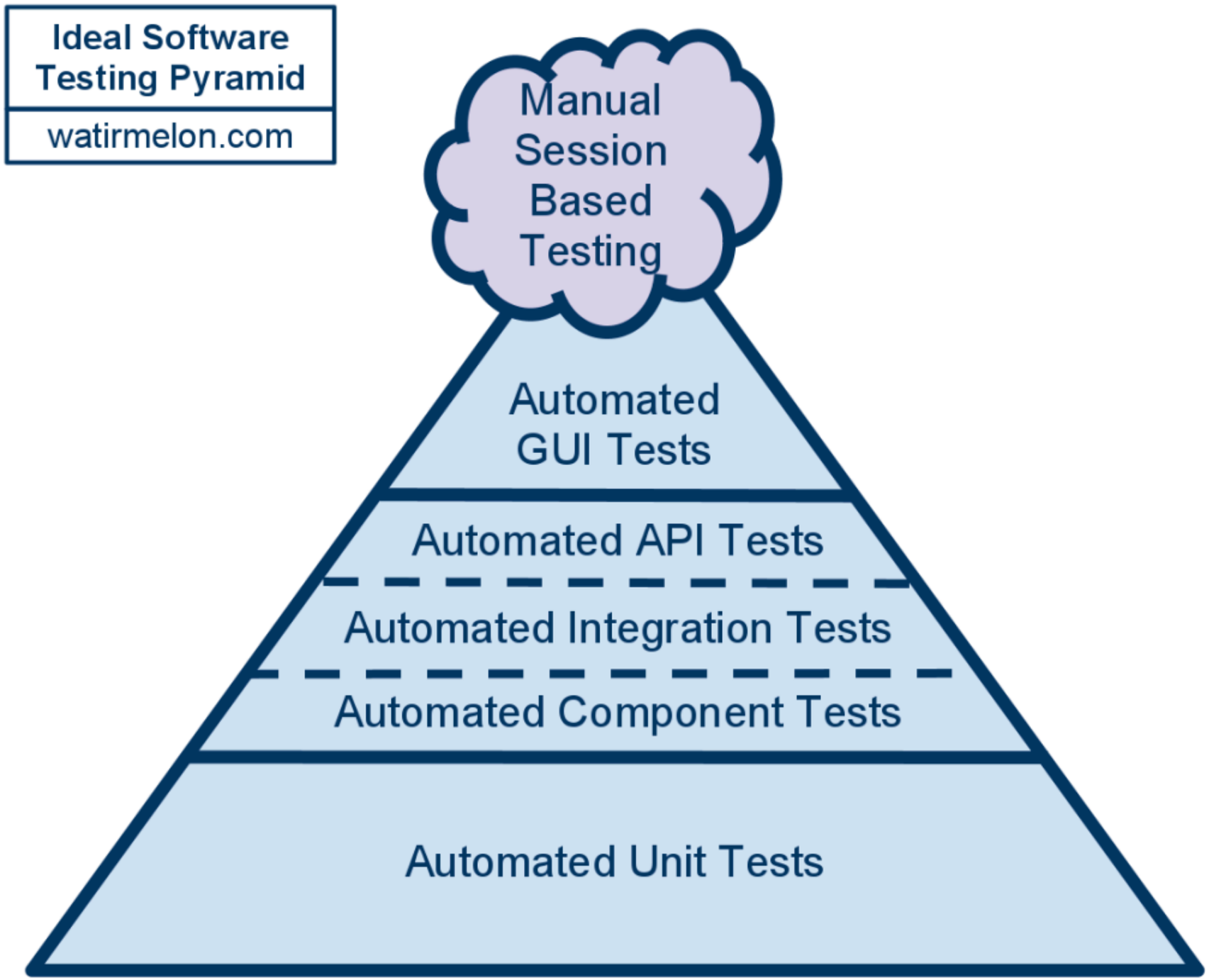
Structure of Automated tests

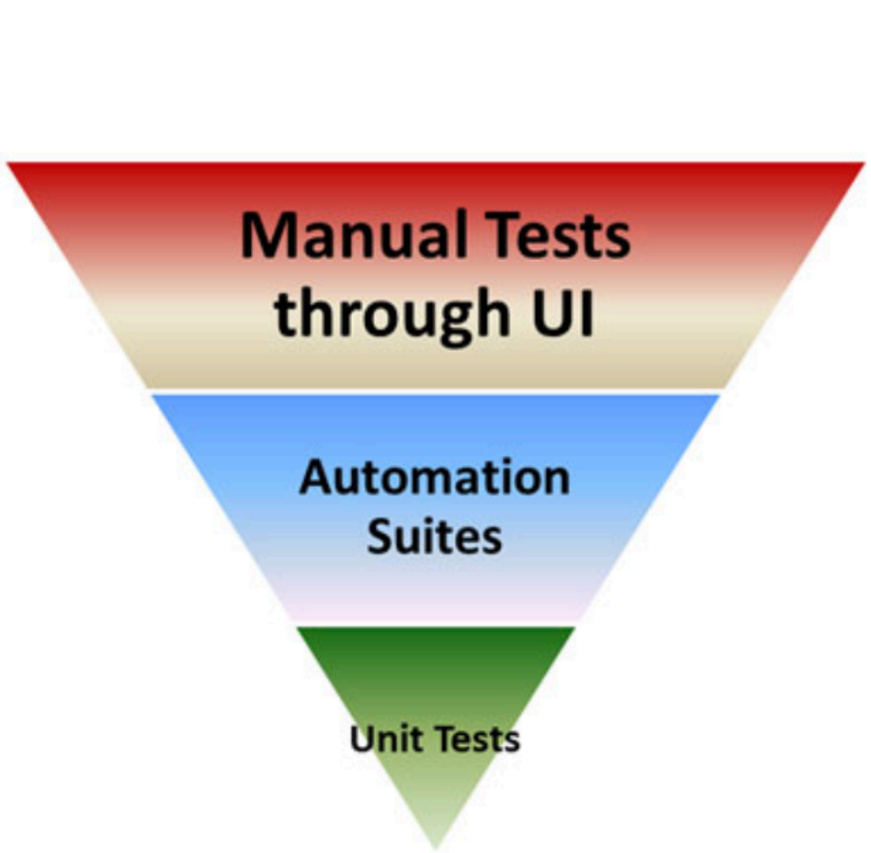


Working with Test double

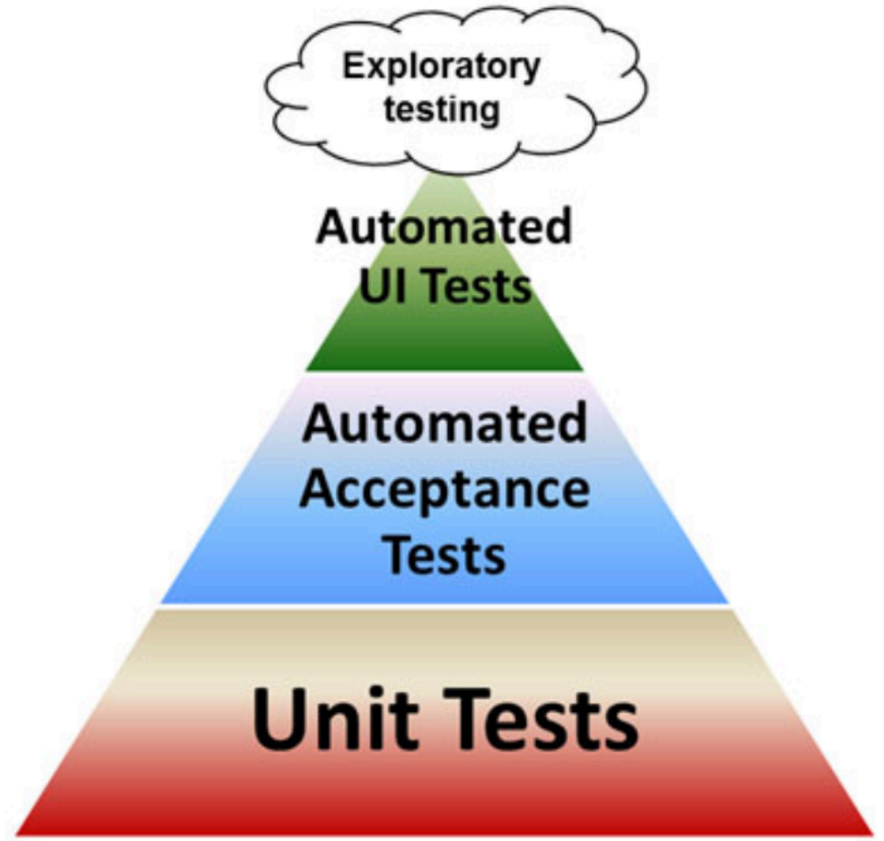




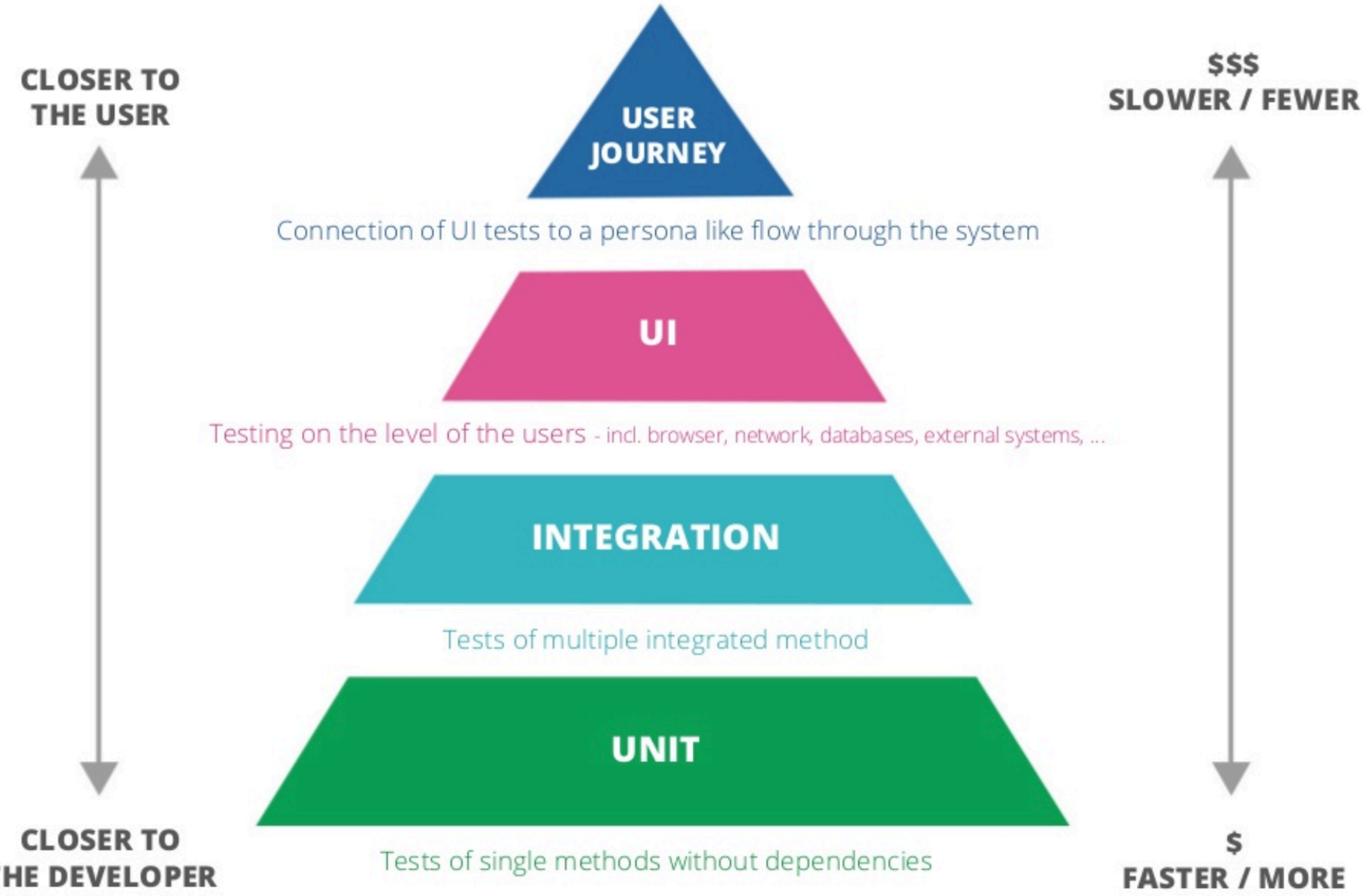




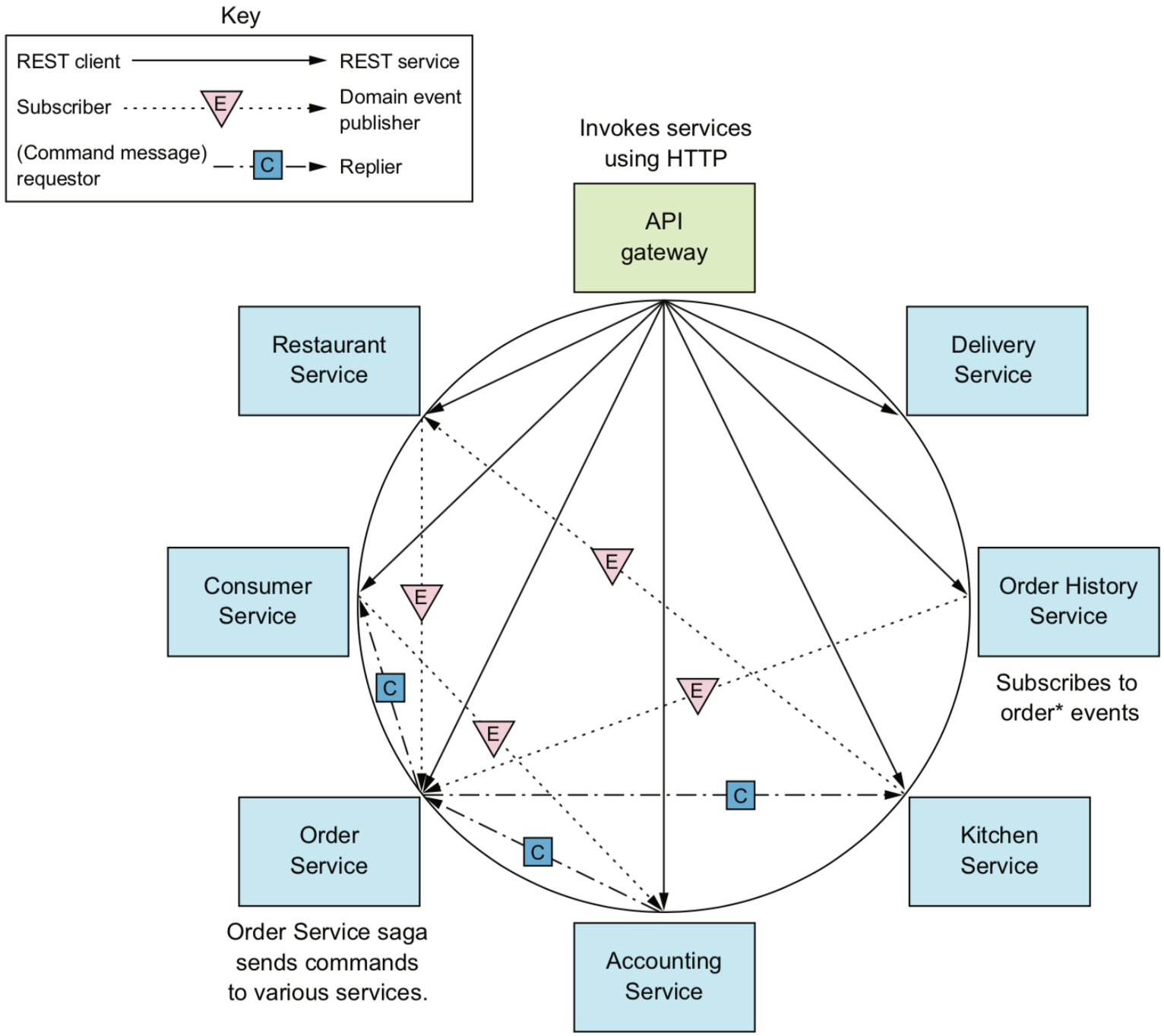
Traditional
(find bugs)



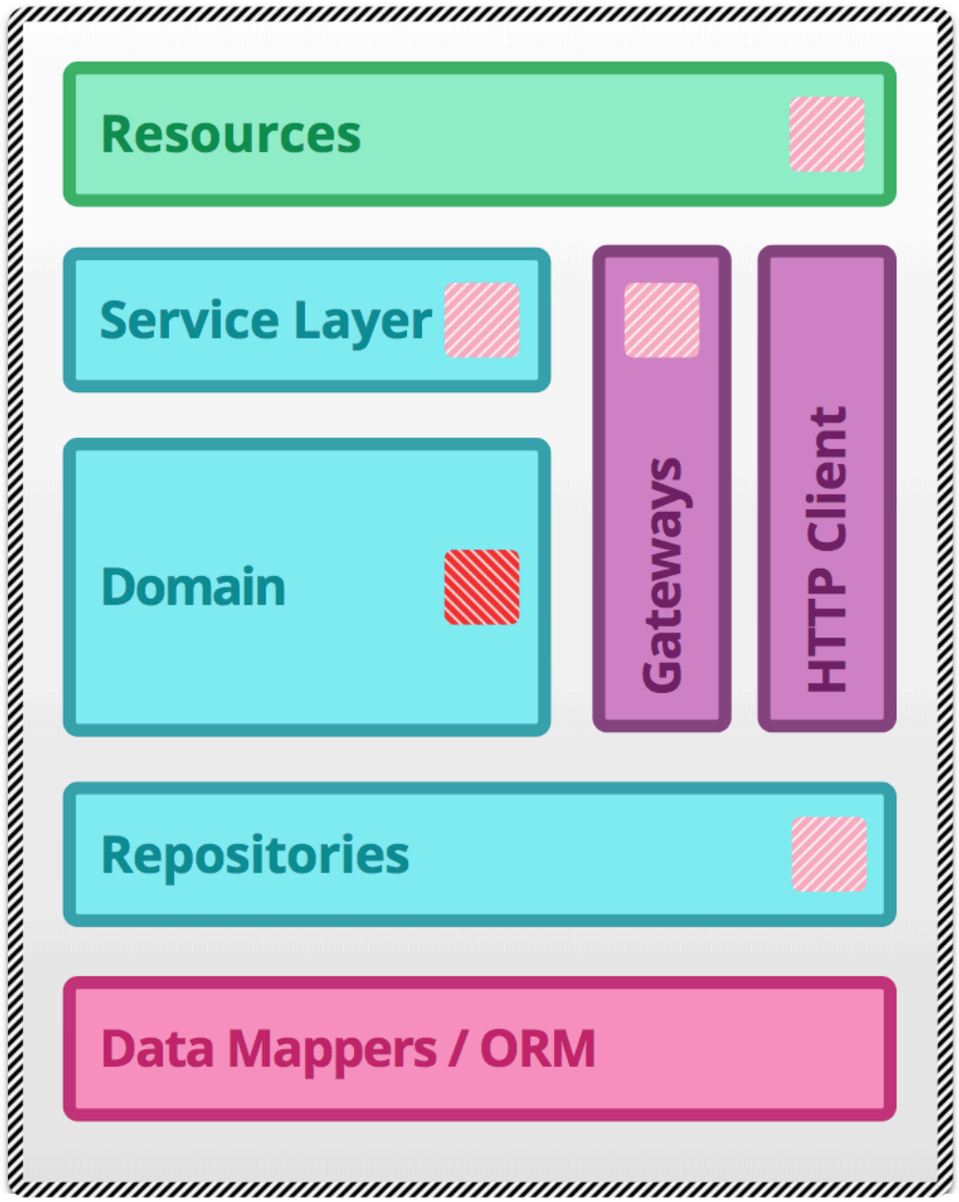
Agile
(prevent bugs)



Testing Microservices ?



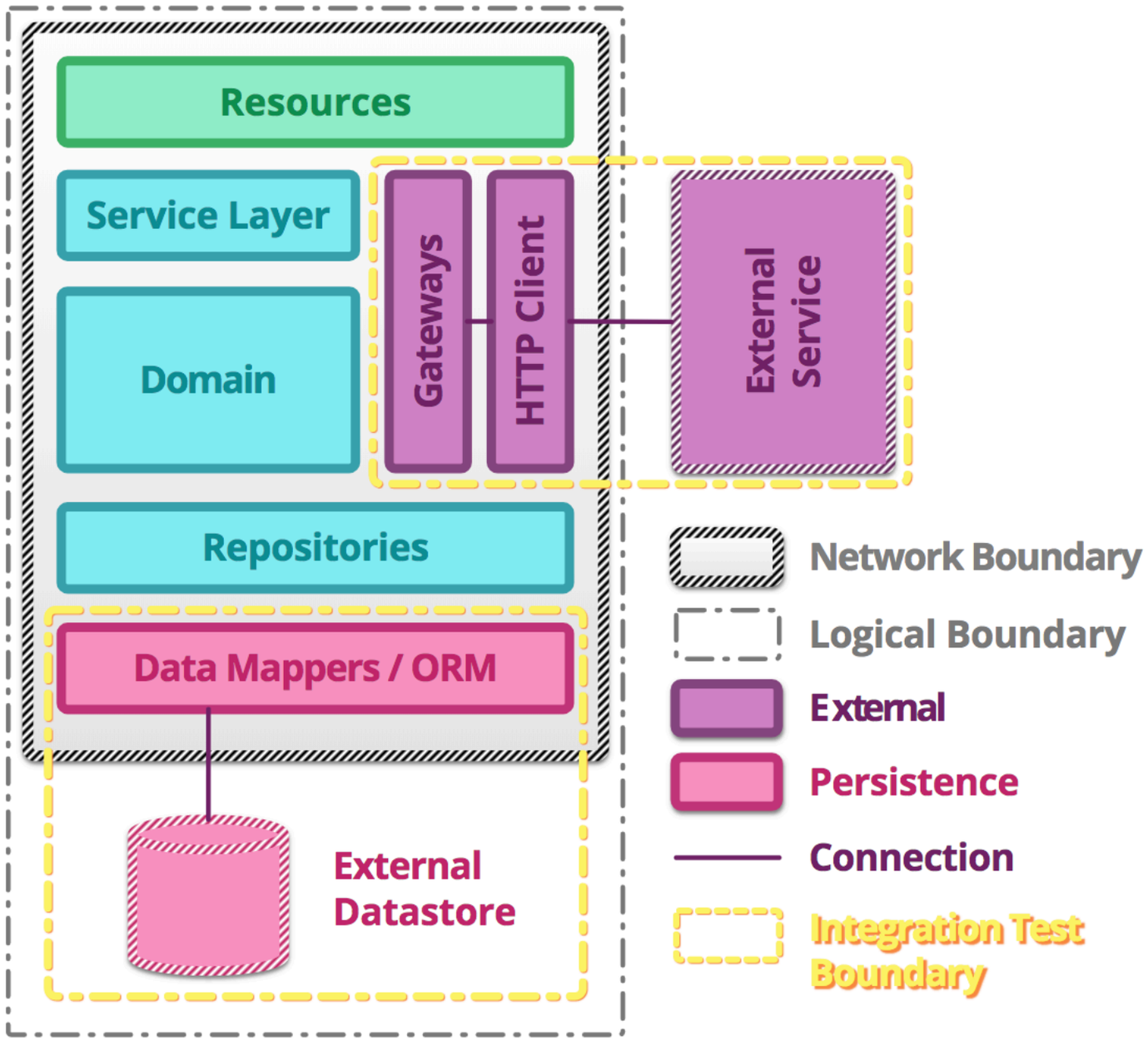
Unit testing



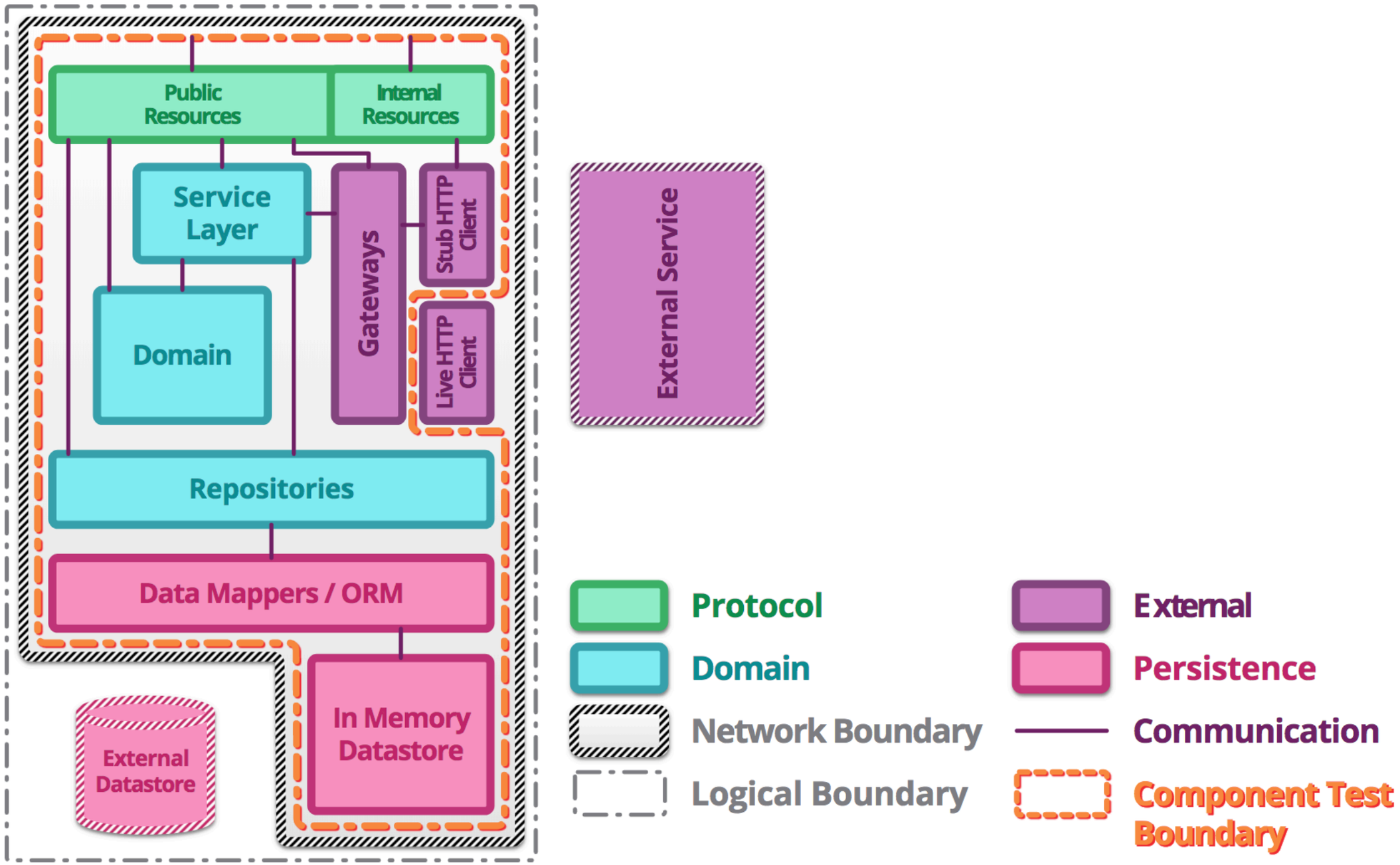
Unit - Solitary

Unit - Sociable

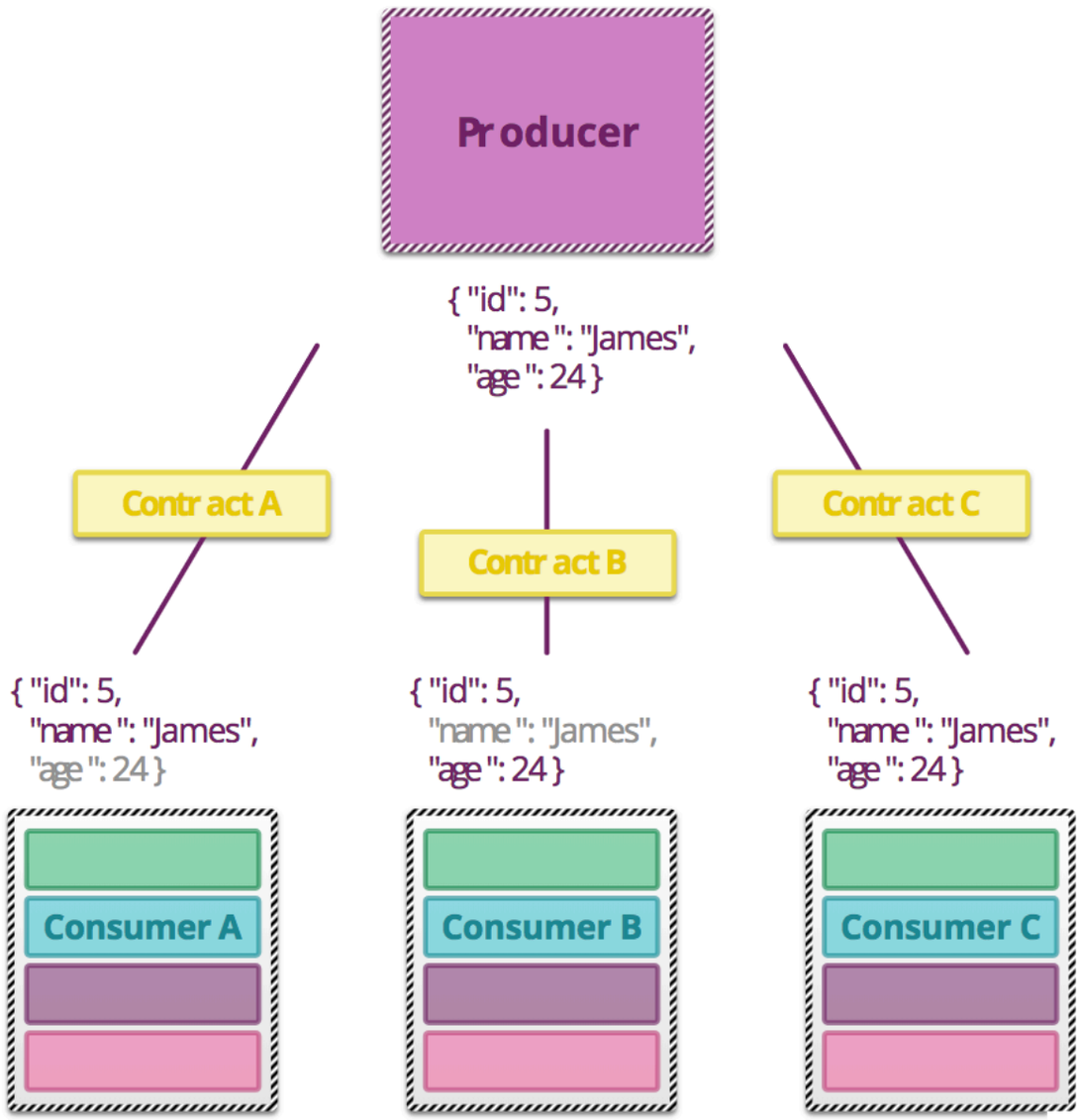
Integration testing



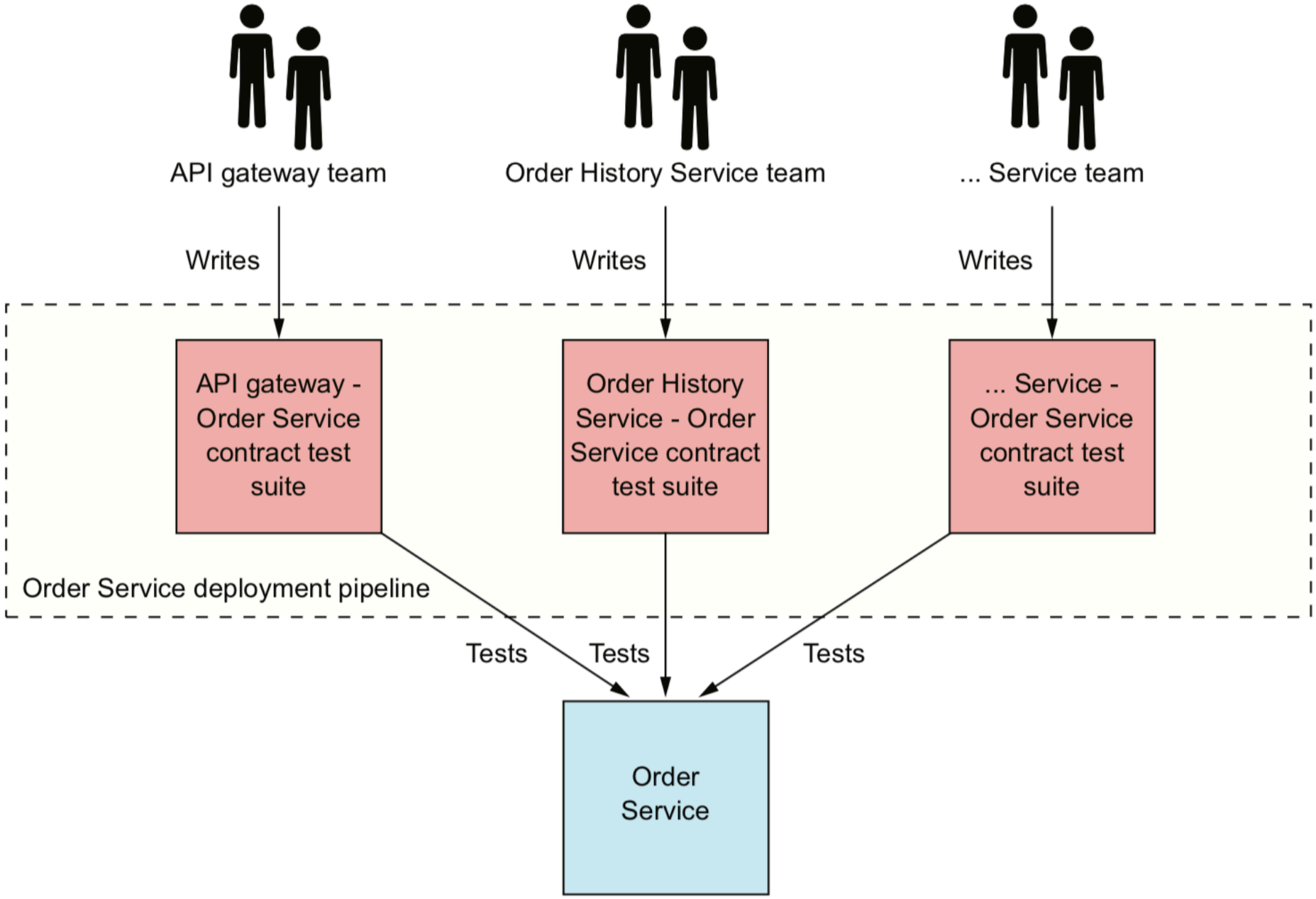
Component testing



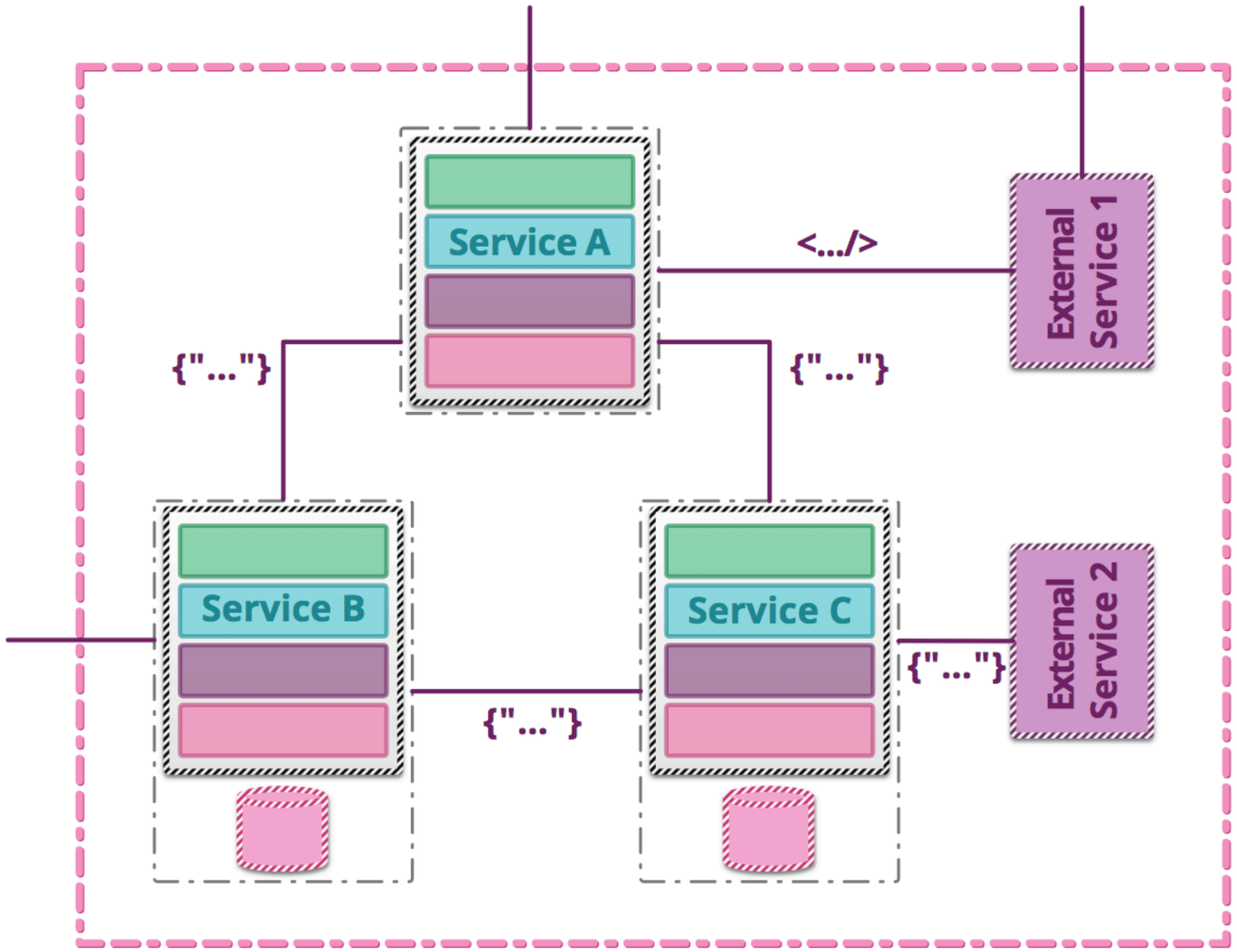
Contract testing



Consumer Contract testing



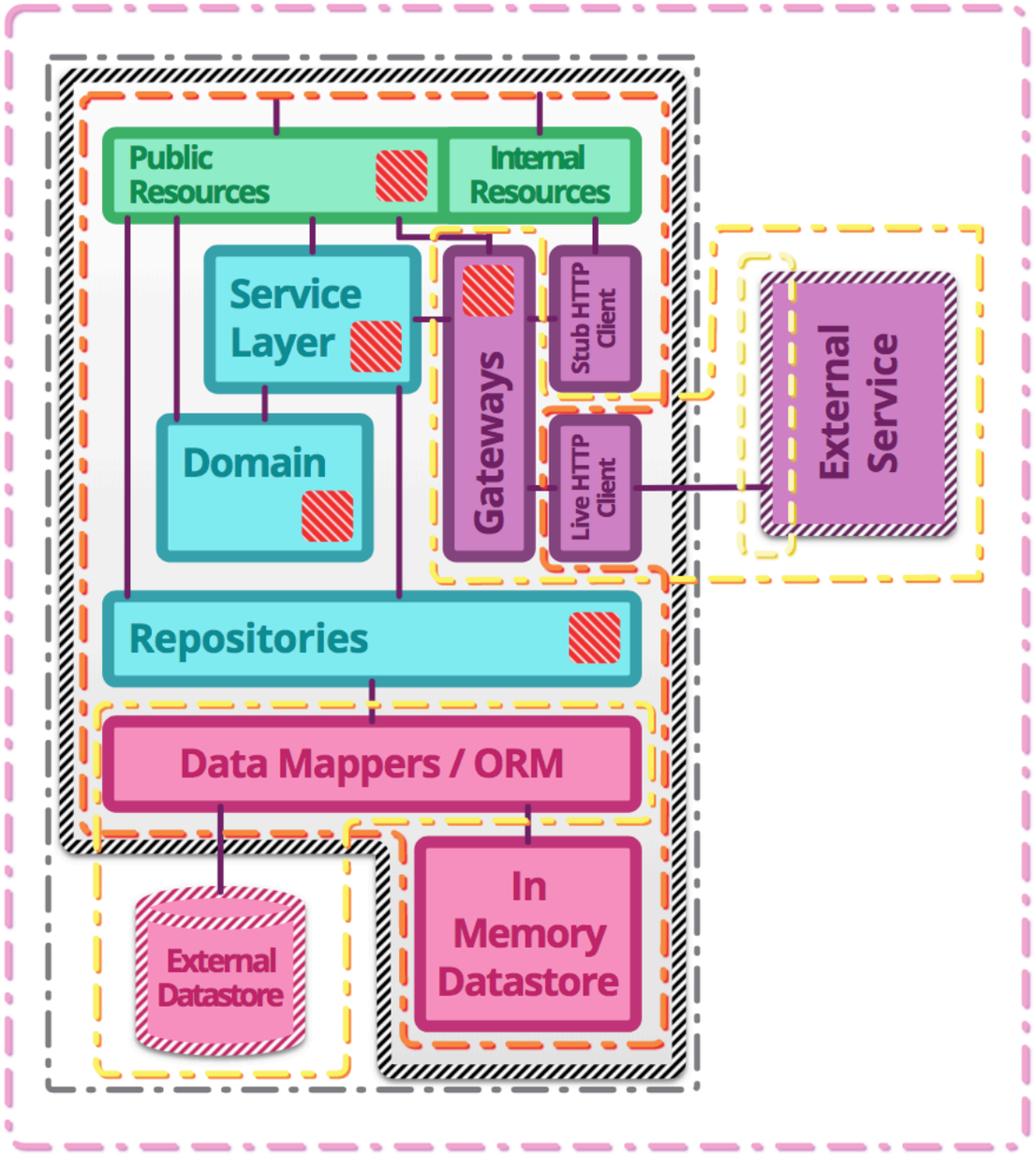
End-to-End testing



Summary

Unit tests : exercise the smallest pieces of testable software in the application to determine whether they behave as expected.

Integration tests : verify the communication paths and interactions between components to detect interface defects.



Component tests : limit the scope of the exercised software to a portion of the system under test, manipulating the system through internal code interfaces and using test doubles to isolate the code under test from other components.

Contract tests : verify interactions at the boundary of an external service asserting that it meets the contract expected by a consuming service.

End-to-end tests : verify that a system meets external requirements and achieves its goals, testing the entire system, from end to end.

What is your testing strategy ?

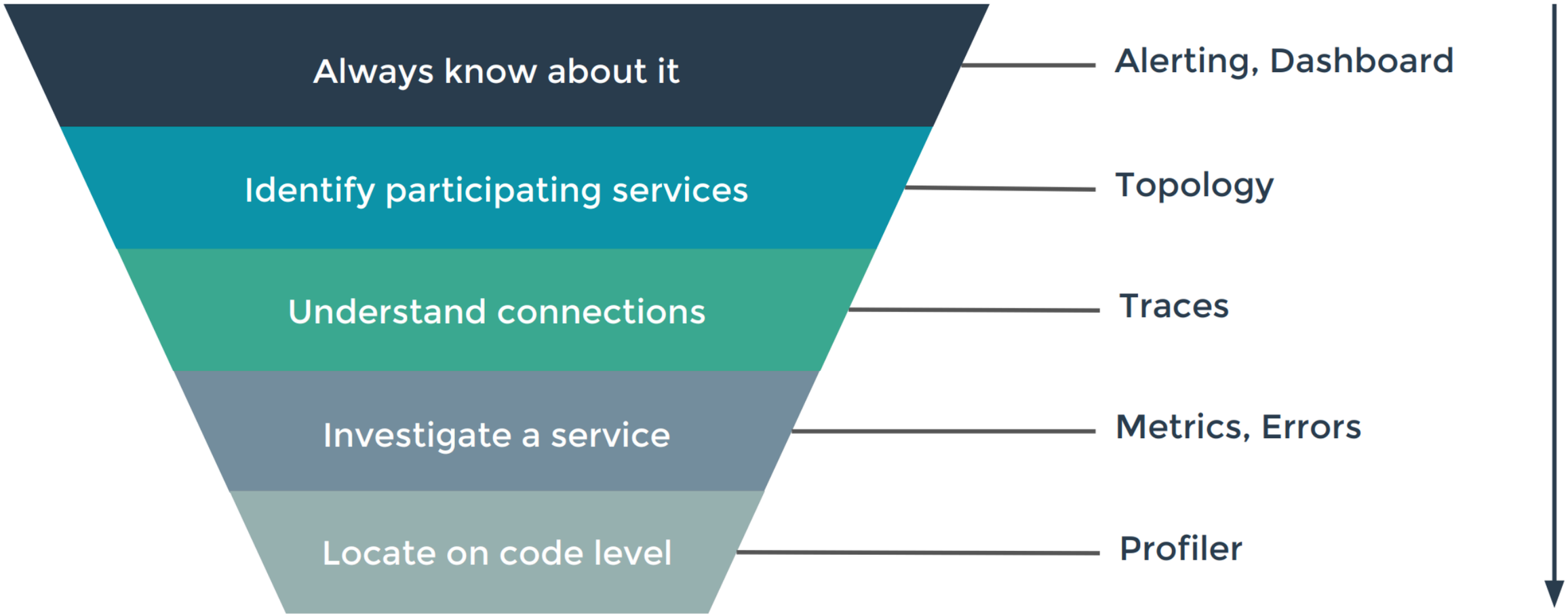
Workshop

Performance testing ?

Security testing ?

How to find an issue ?

How to find an issue ?



Develop production-ready services

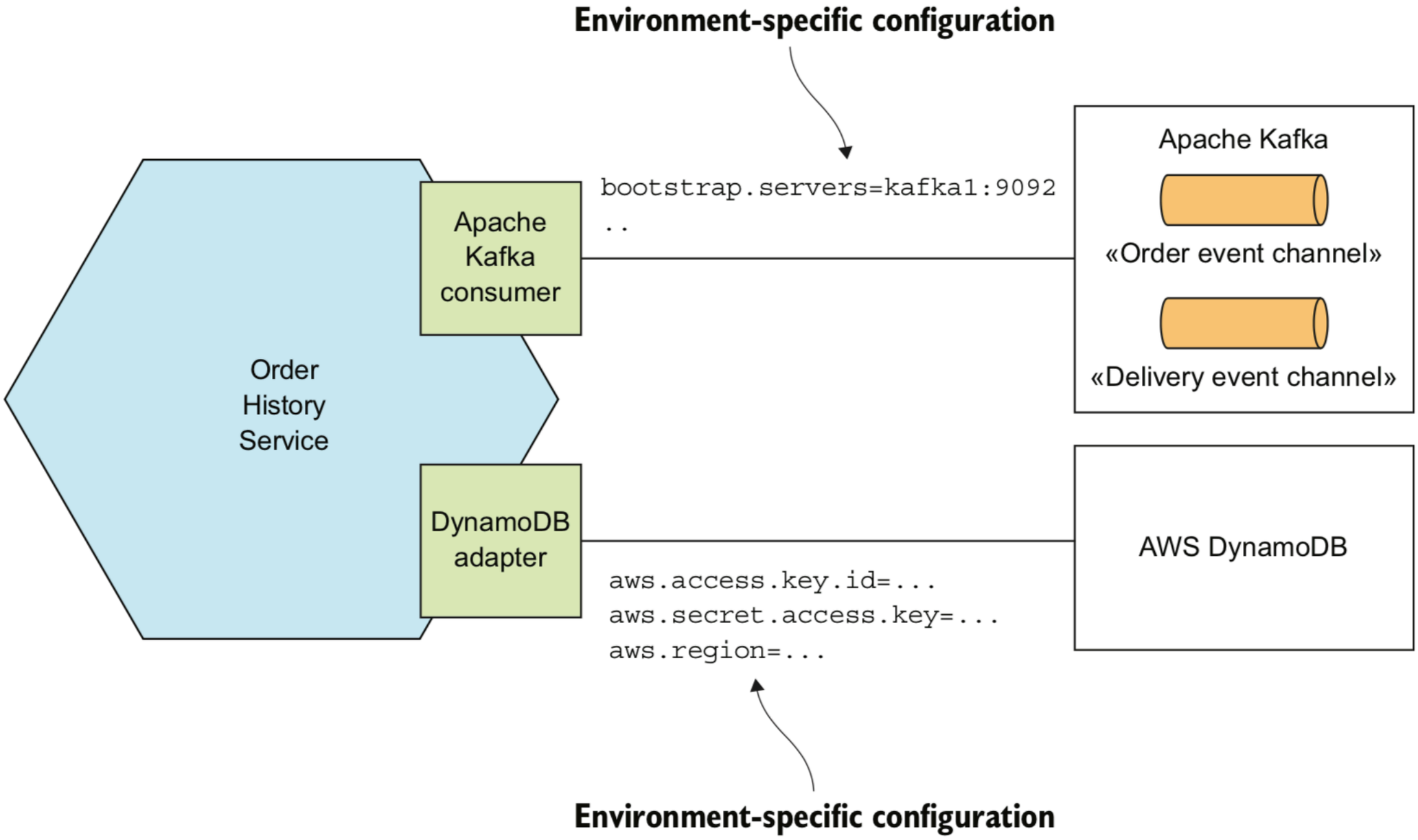
Important quality attributes

Security
Configurability
Observability

Develop secure services

Authentication
Authorization
Auditing
Secure interprocess communication

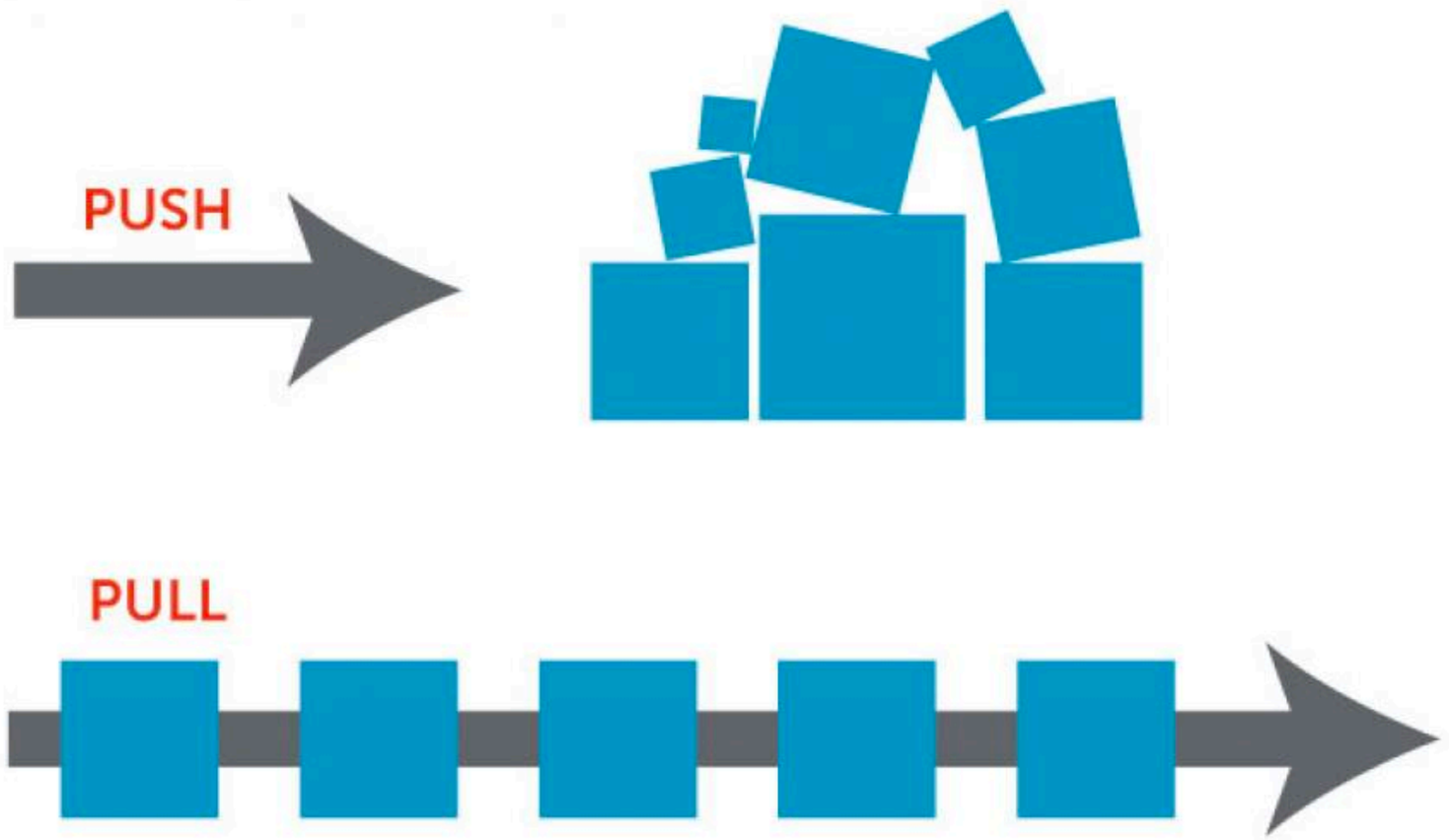
Design configurable services



External configuration models

Push model

Pull model

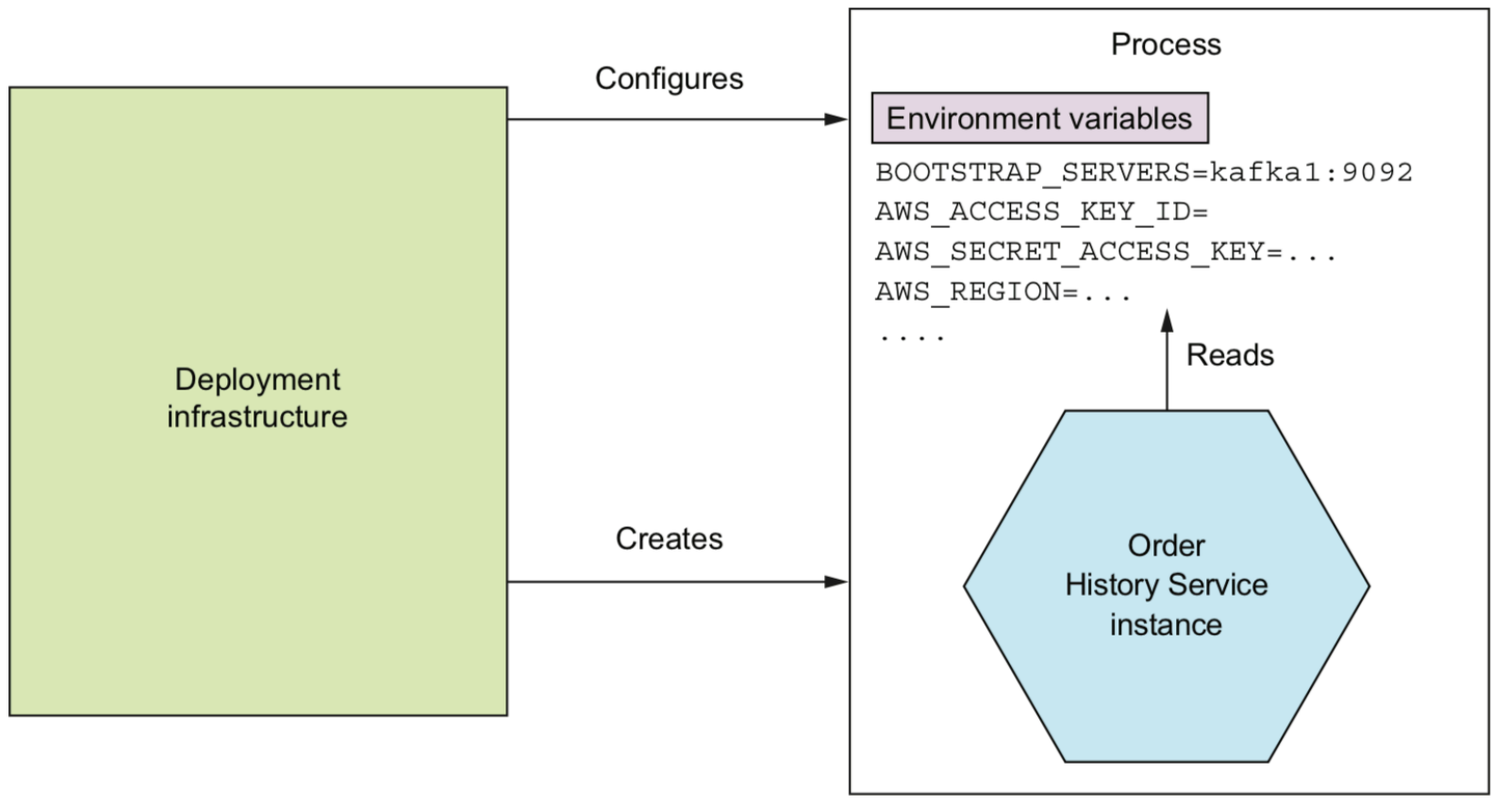


Push model

Pass the configuration to service

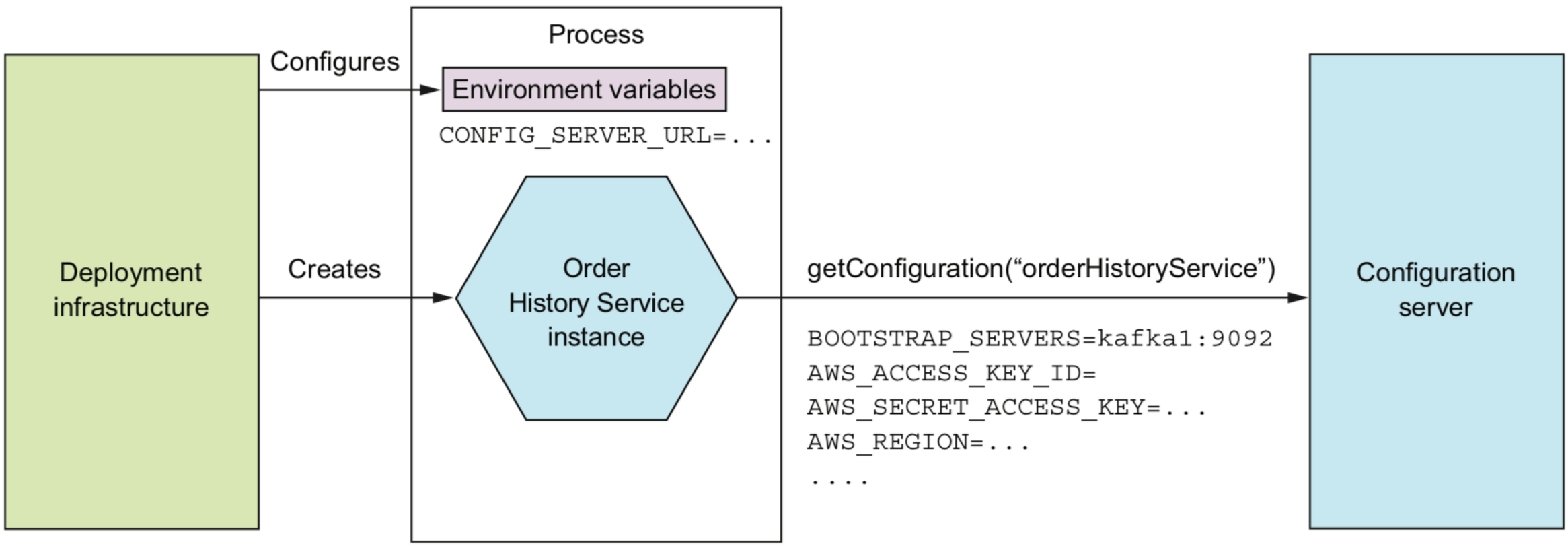
OS environment variables

Configuration files



Pull model

Service read configuration from configuration server



Benefits of configuration server

Centralized configuration
Transparent decryption of sensitive data
Dynamic reconfiguration

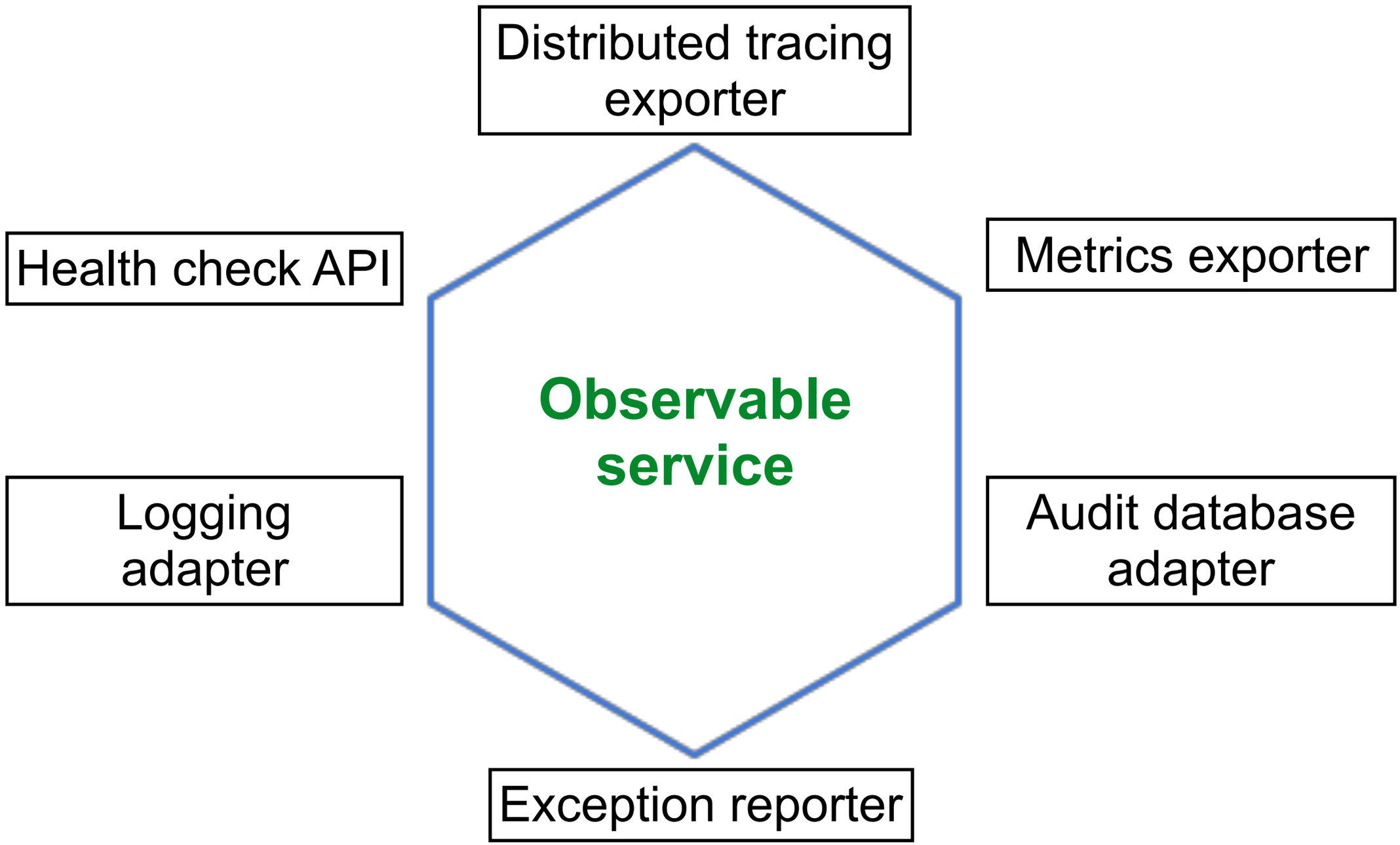
Drawbacks of configuration server

Need setup and maintain !!

Design observable services

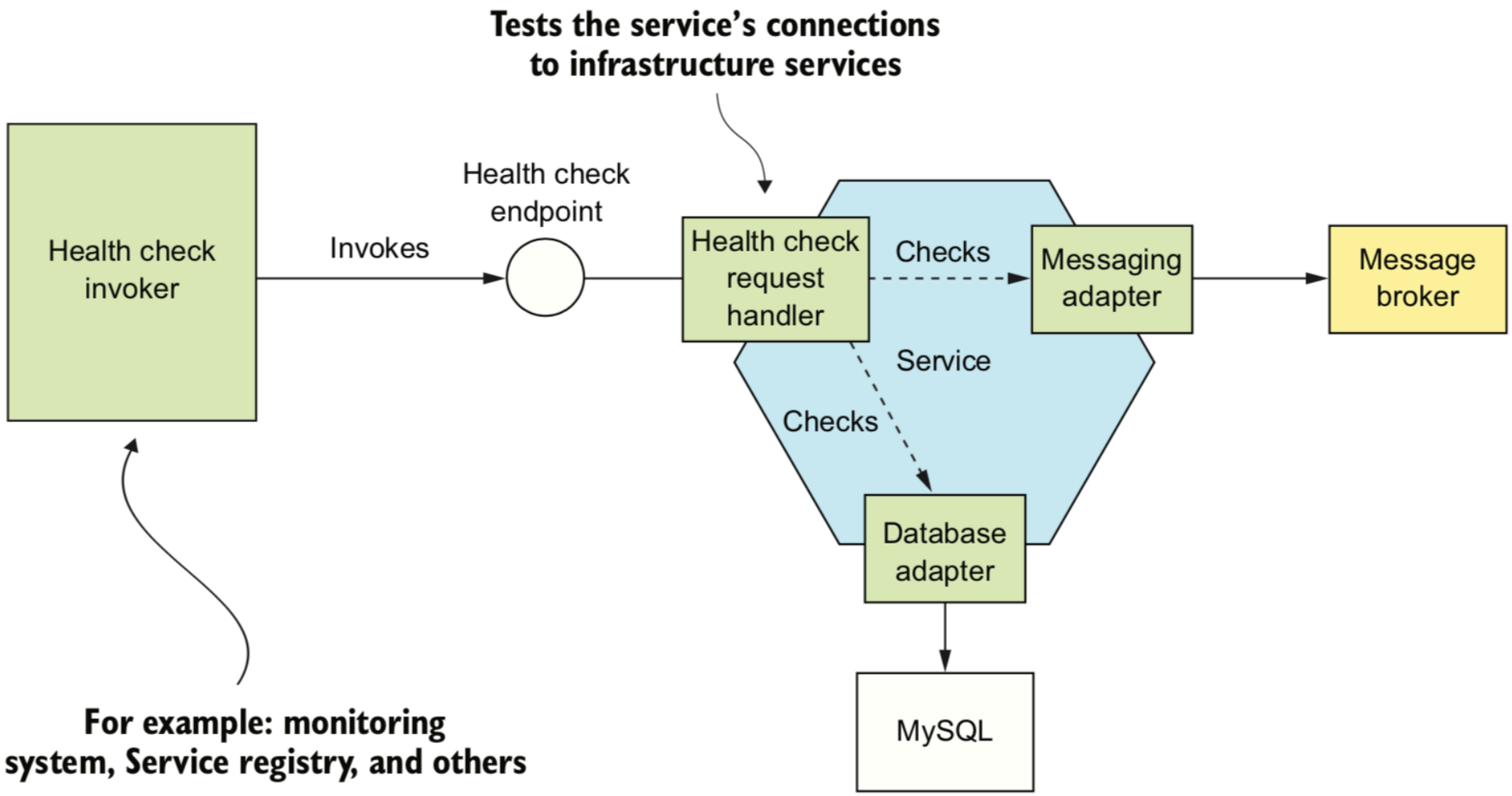
Health check API
Log aggregation
Distributed tracing
Exception tracking
Application metrics
Audit logging

Observable services



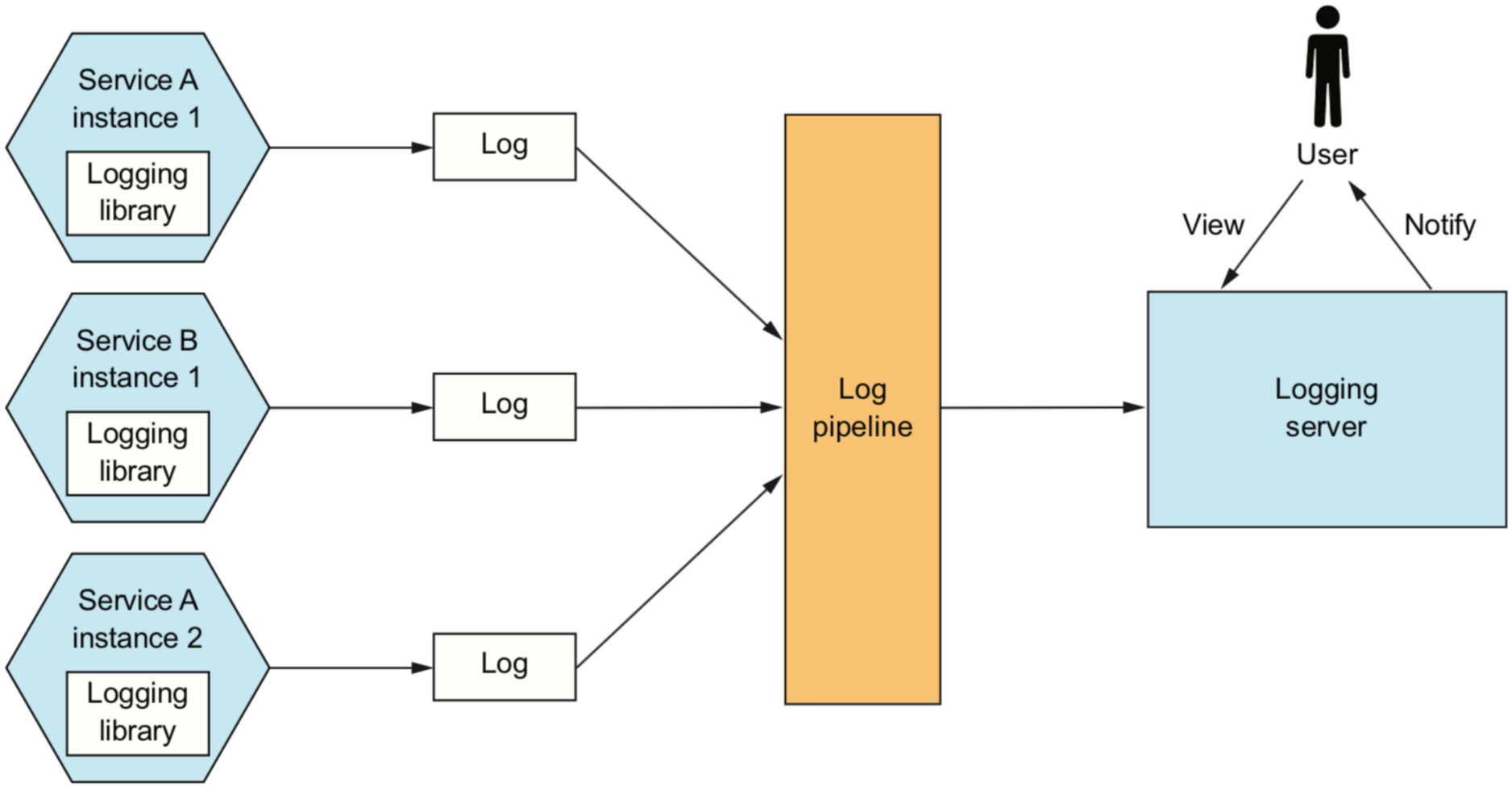
Health check API

Expose an endpoint that return the health of service



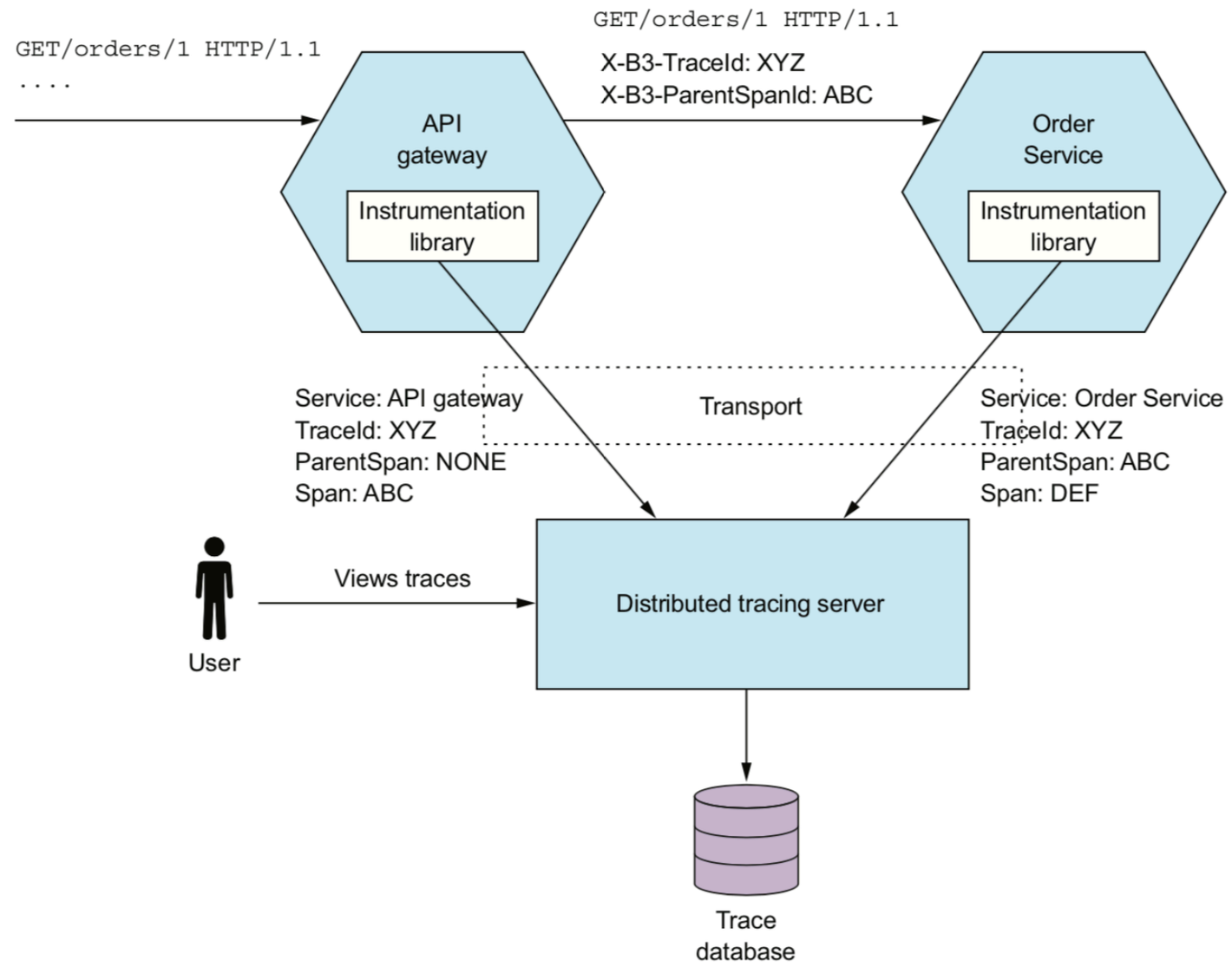
Log aggregation

Log service activity and write logs into a centralized logging server. (searching, alerting)

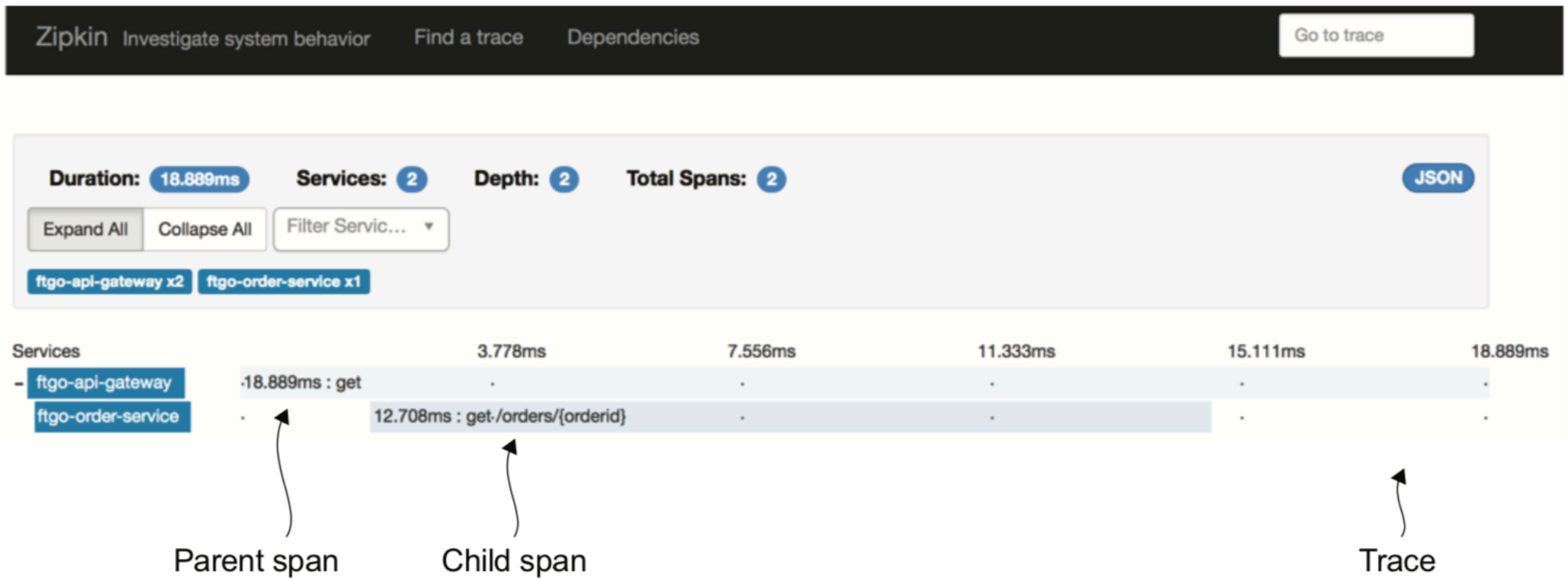


Distributed tracing

Assign each external request a unique ID and trace requests as flow between services



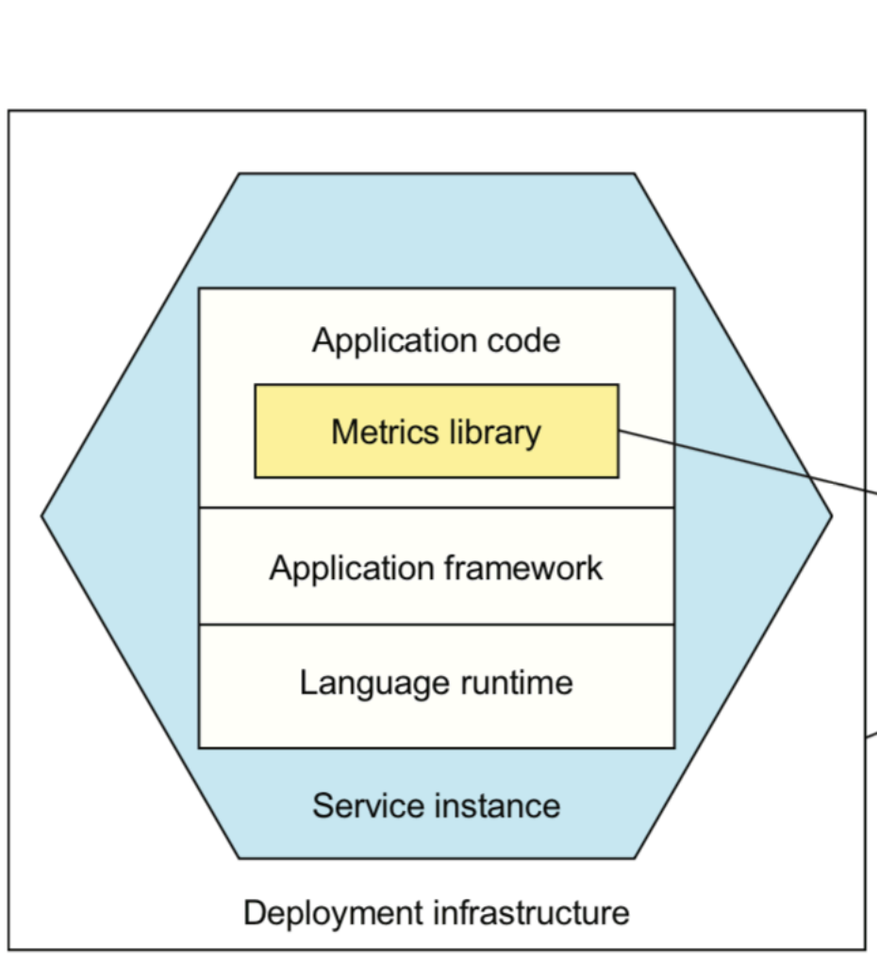
Distributed tracing with Zipkin

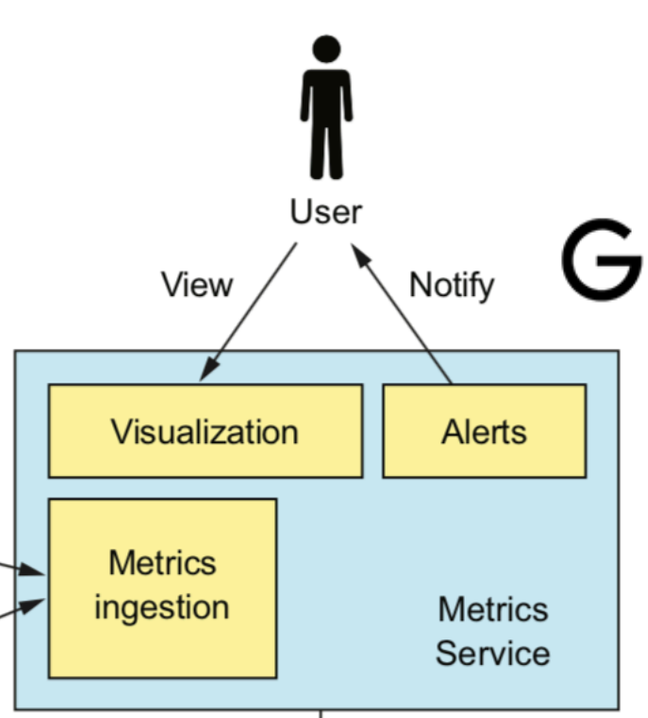


https://zipkin.io/

Application metrics

Services maintain metrics and expose to metric server (counters, gauges)





Grafana

Metrics sample:

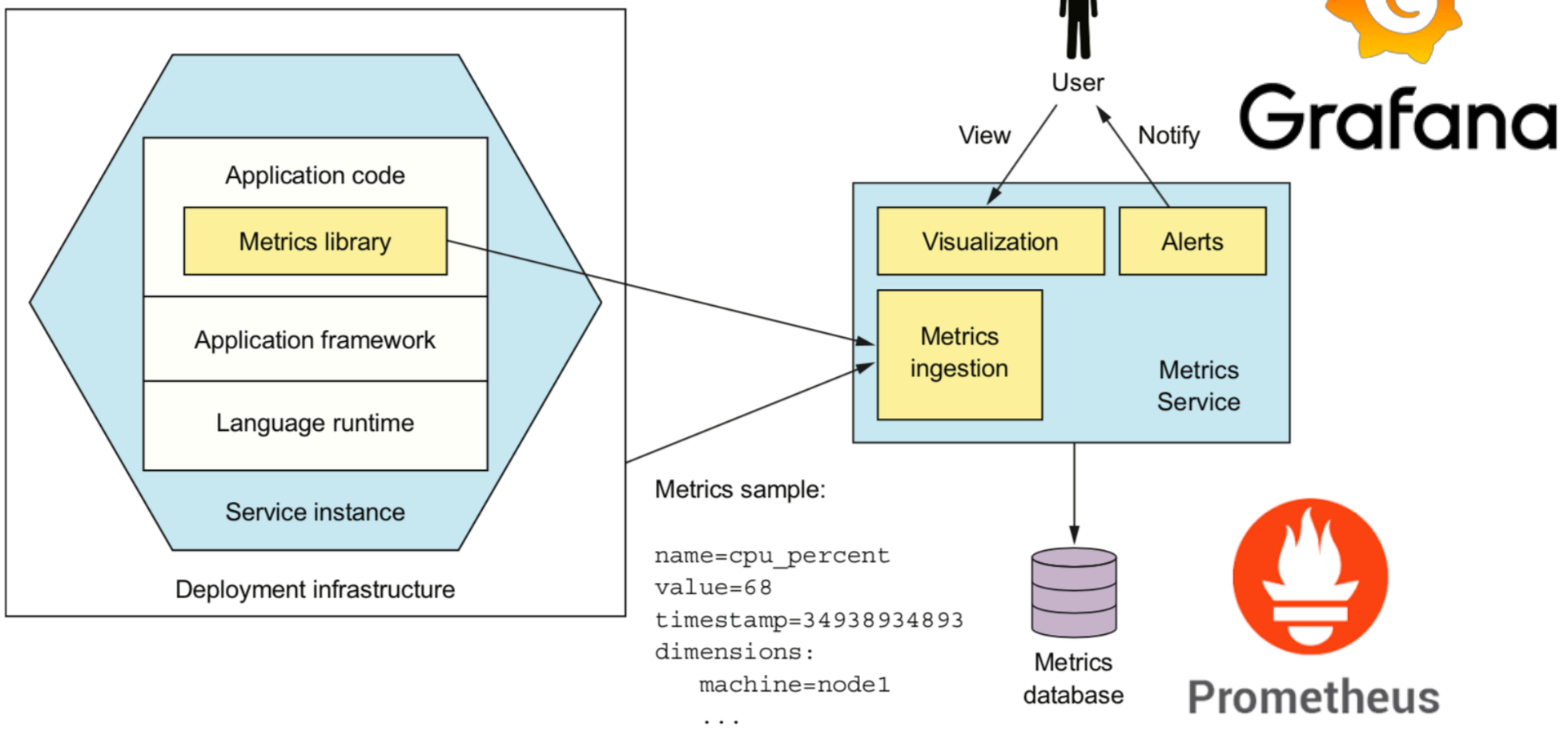
```
name=cpu_percent
value=68
timestamp=34938934893
dimensions:
   machine=node1
   ...
```

Metrics database

Prometheus

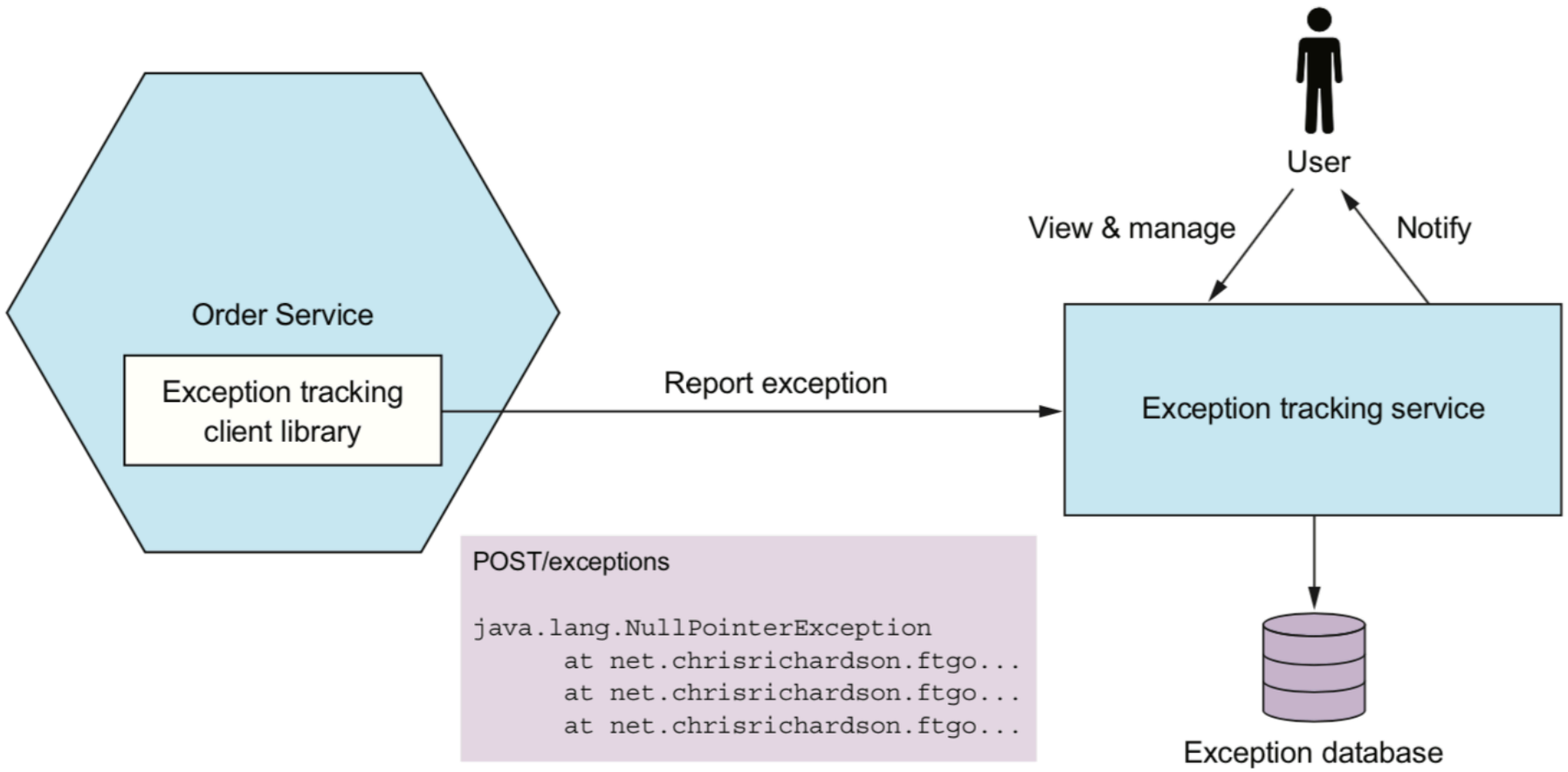
Application metrics

Metrics database => Prometheus
Visualization, Alert => Grafana



Exception tracking

Report exceptions to exception tracking service
Help to identify the root cause



Exception tracking services

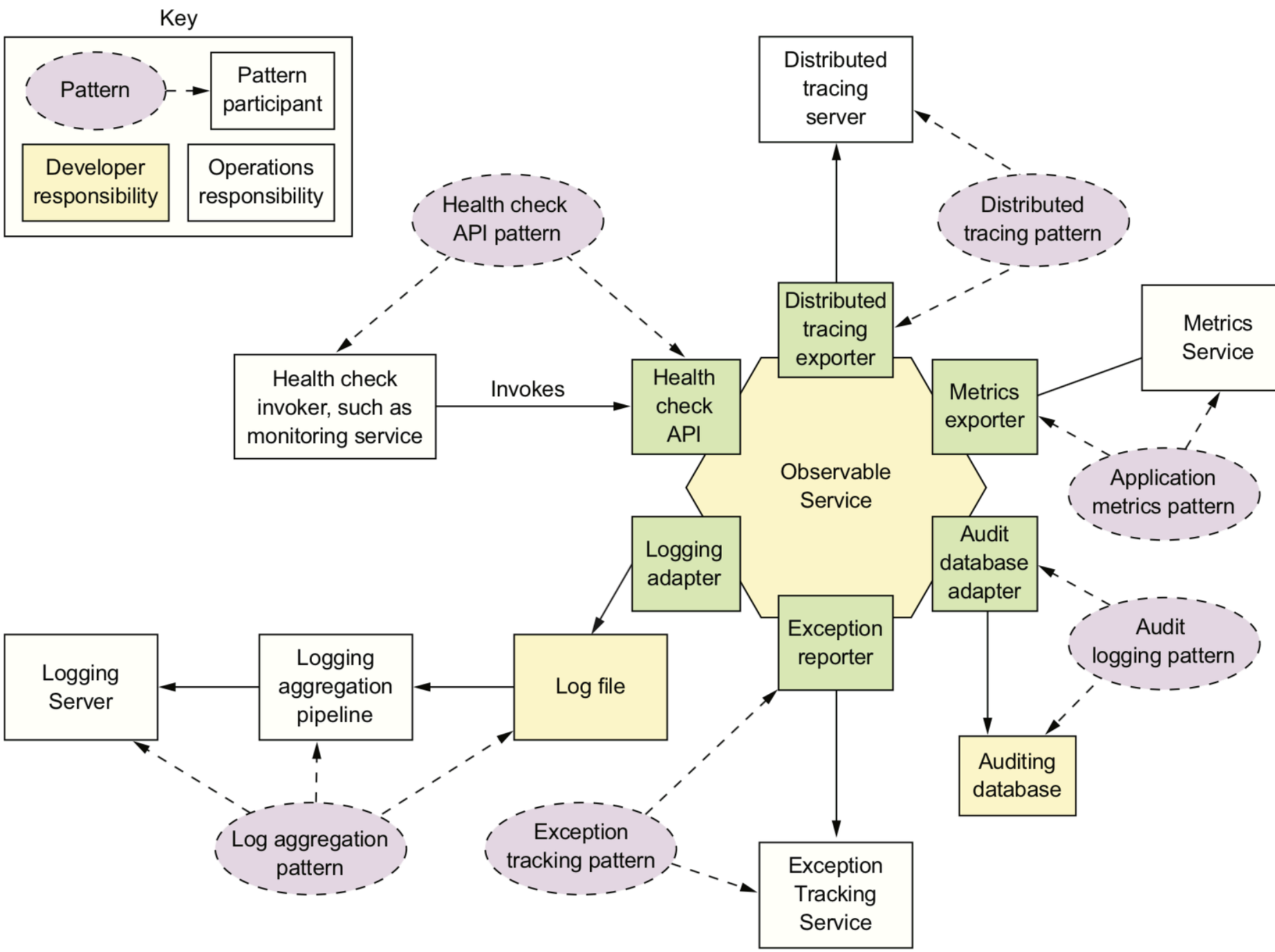
Audit logging

Log of user actions
Help customer support
Ensure compliance
Detect suspicious behaviour

How to implement the audit logging ?

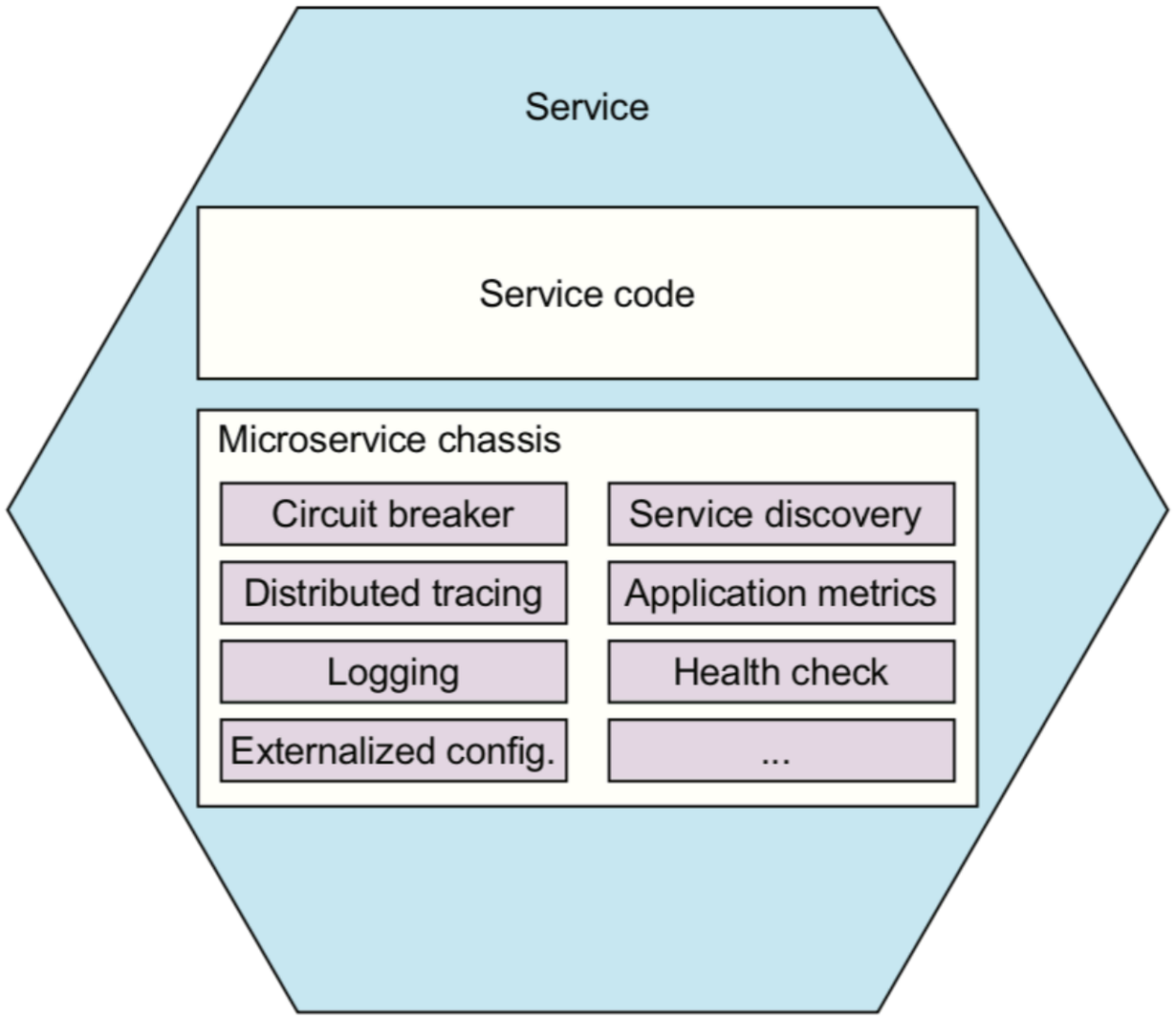
Add logging code in business logic
Use AOP (Aspect-Oriented Programming)
Use event sourcing

Observable services



Microservice framework

Microservice framework

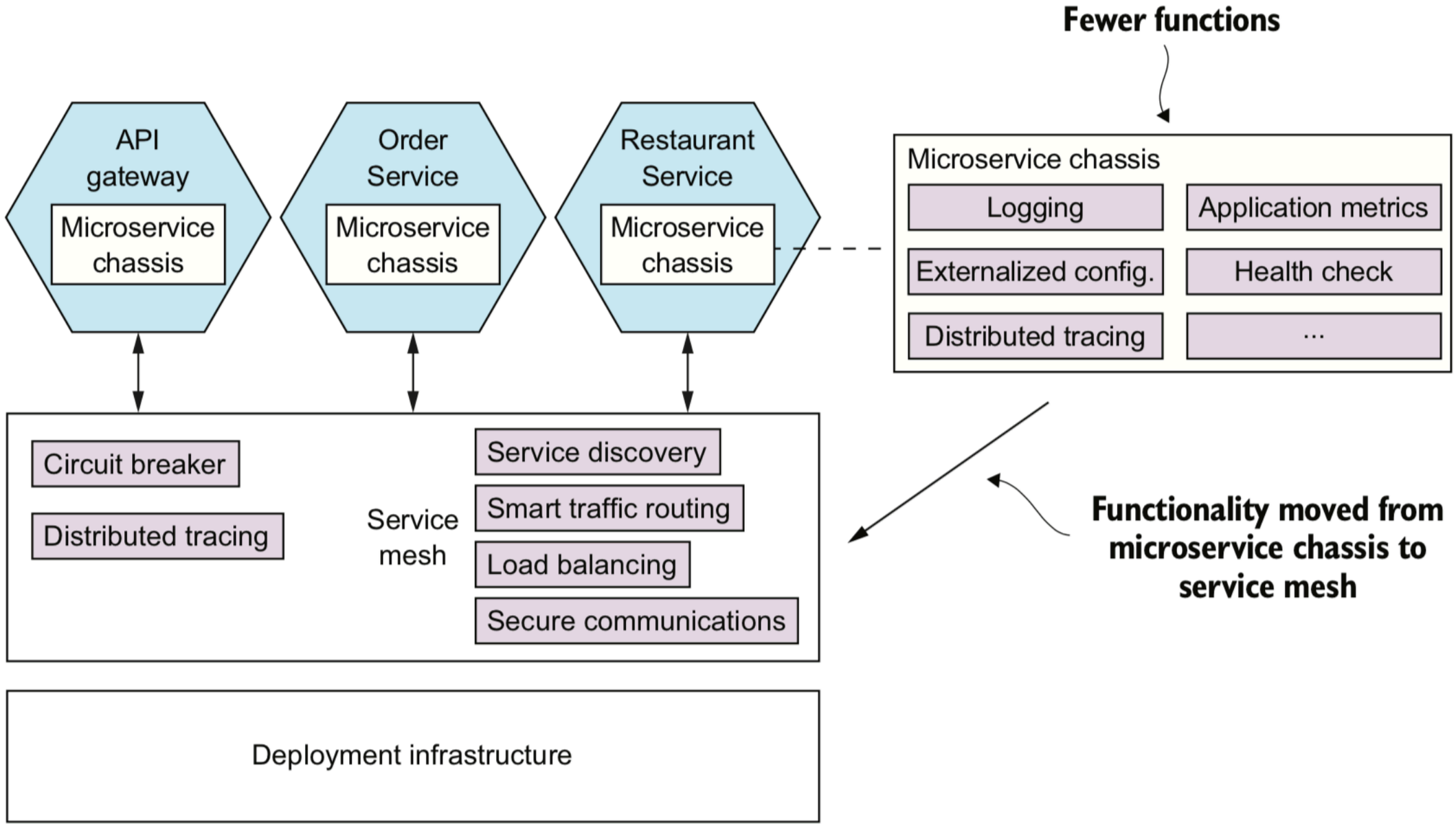


Microservice framework

Programming language	Framework
Java	Spring boot Spring cloud
Golang	Go-kit Micro

Next …

Service Mesh



Service Mesh

Istio

Connect, secure, control, and observe services.

CONDUIT

linkerd

More

Deployment pipeline in each service