Quarkus

Supersonic Subatomic Java

About me

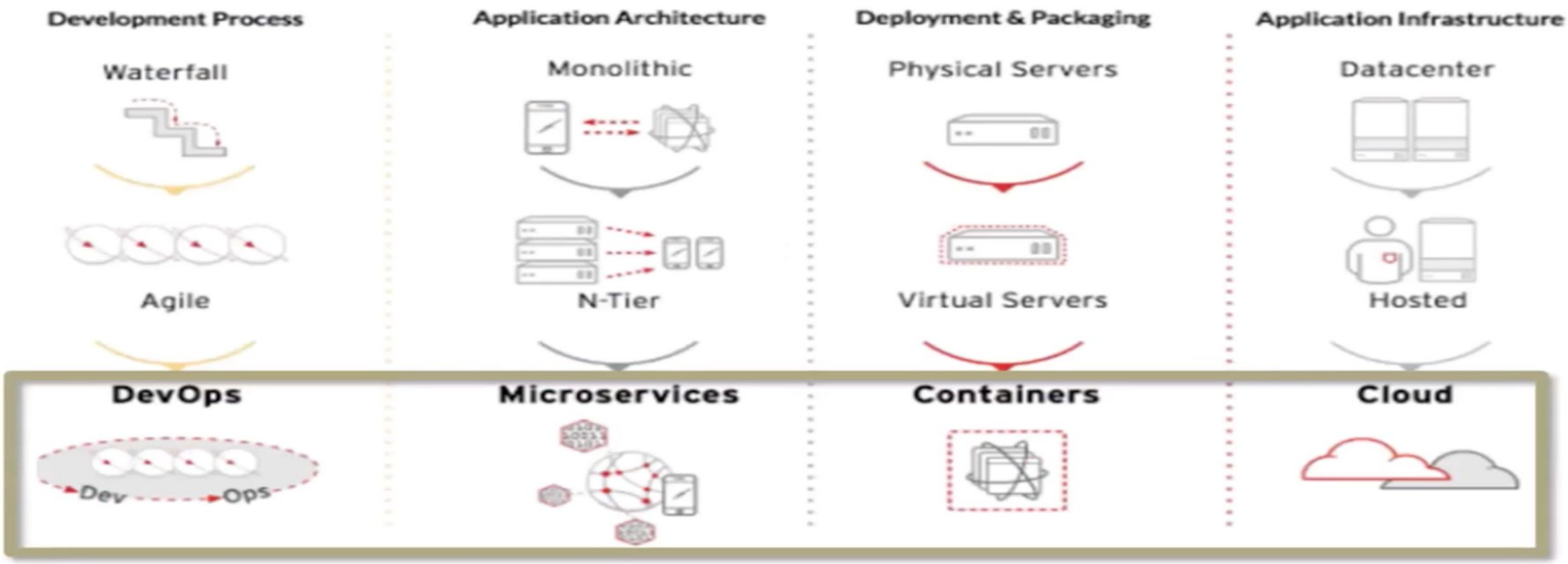
Name : Nagabhushanam

Trainer & Consultant

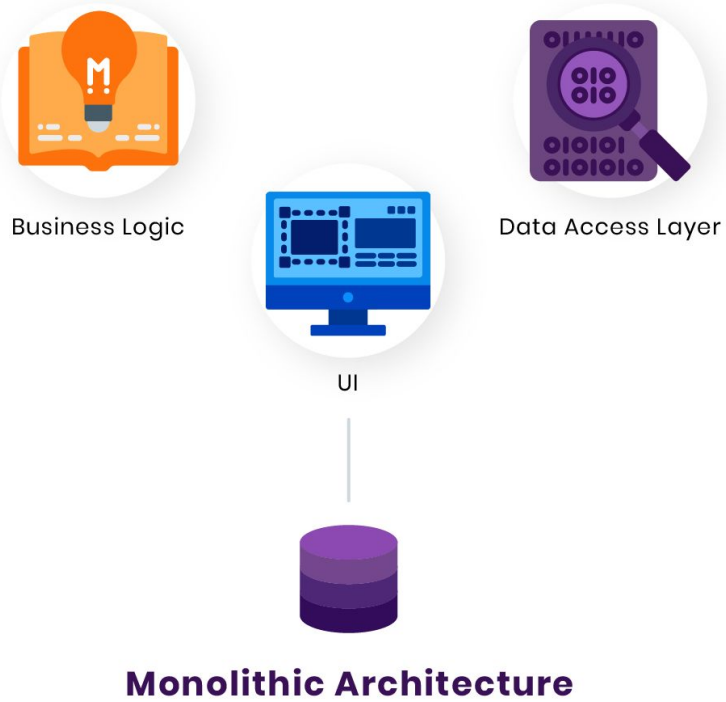
Exp : 13yrs

Location : chennai

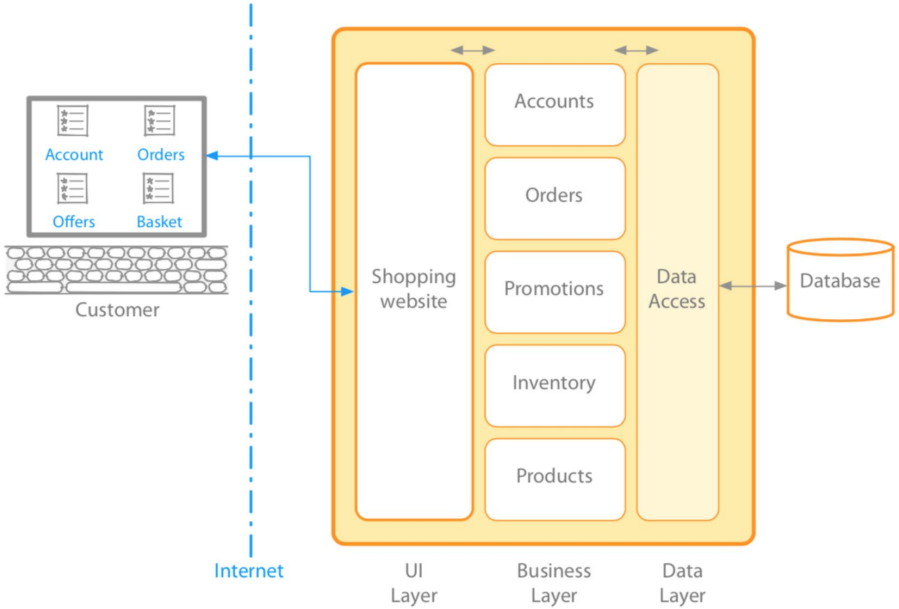
Software evolution



Monolithic architecture



Monolithic architecture - Ex



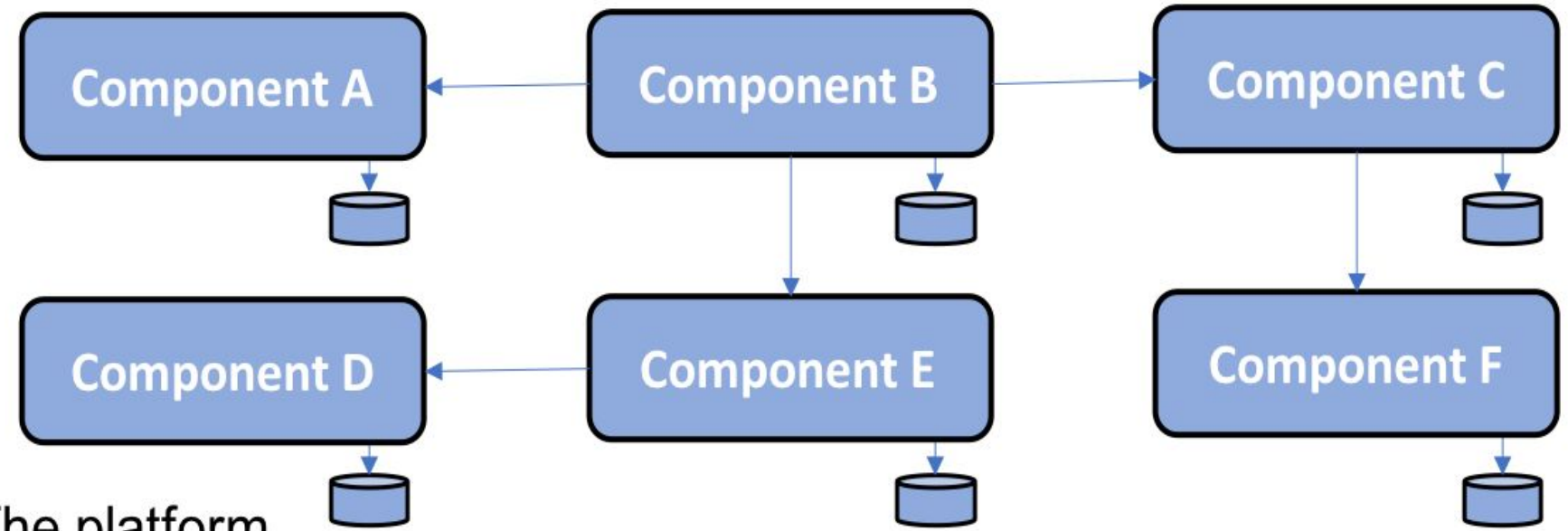
Monolithic architecture - pros

- Simpler development & deployment
- Fewer cross-cutting concerns
- Better performance

Monolithic architecture - cons

- Larger codebase
- Difficult to adopt new technologies
- Failure could affect whole system
- Scaling requires duplication of the whole
- Limited agility
  - every small update requires a full redeployment
  - all developers have to wait until it’s done
  - When several teams are working on the same project, agility can be reduced greatly.

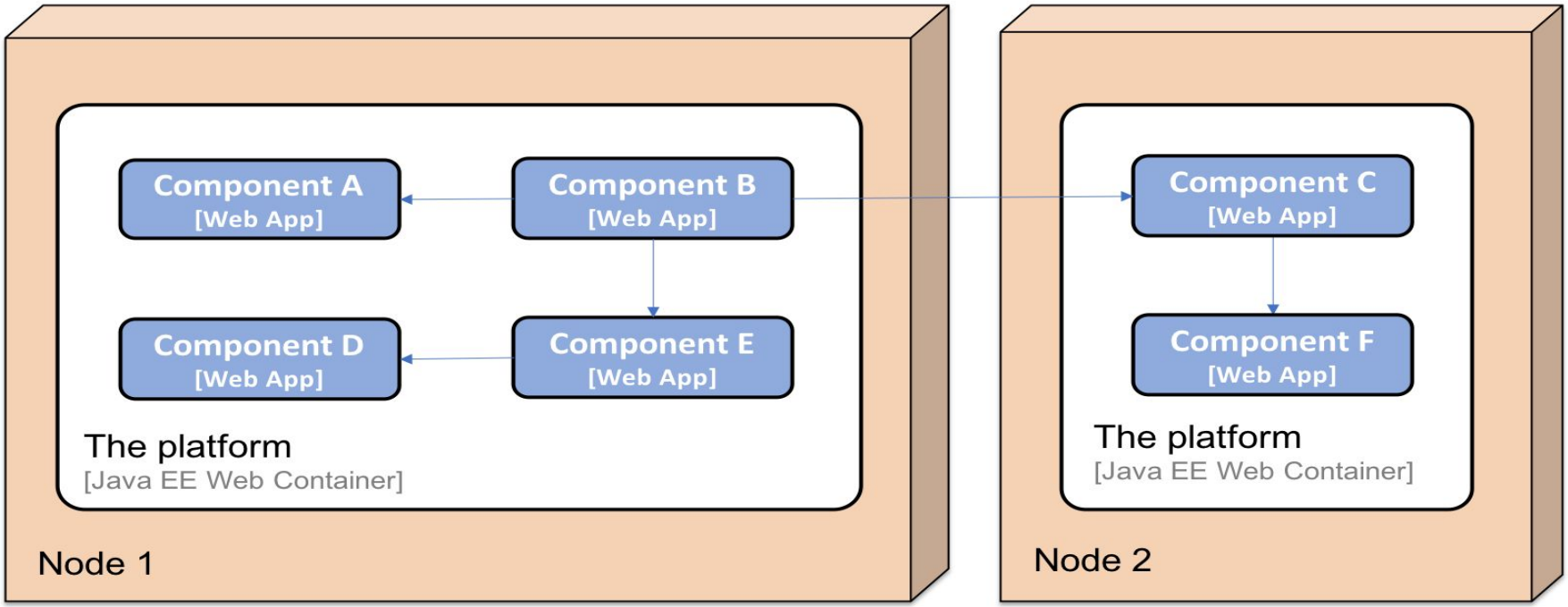
My way into Microservices



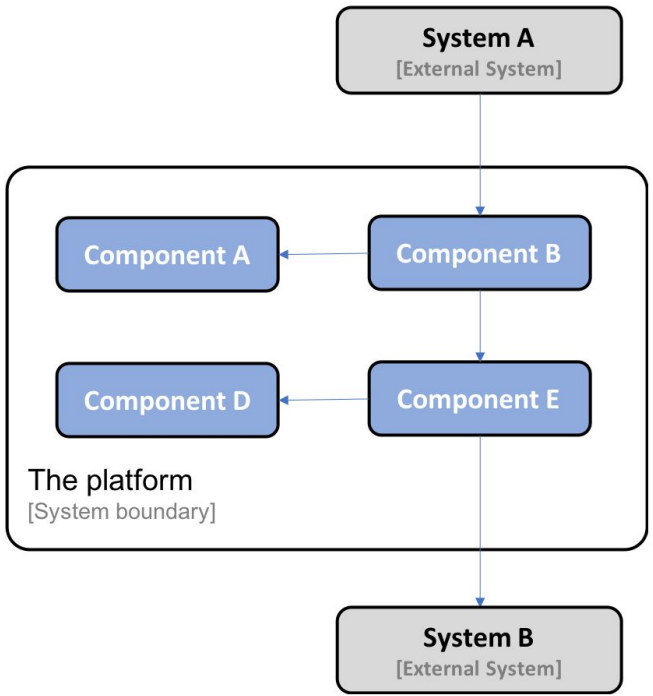
The platform

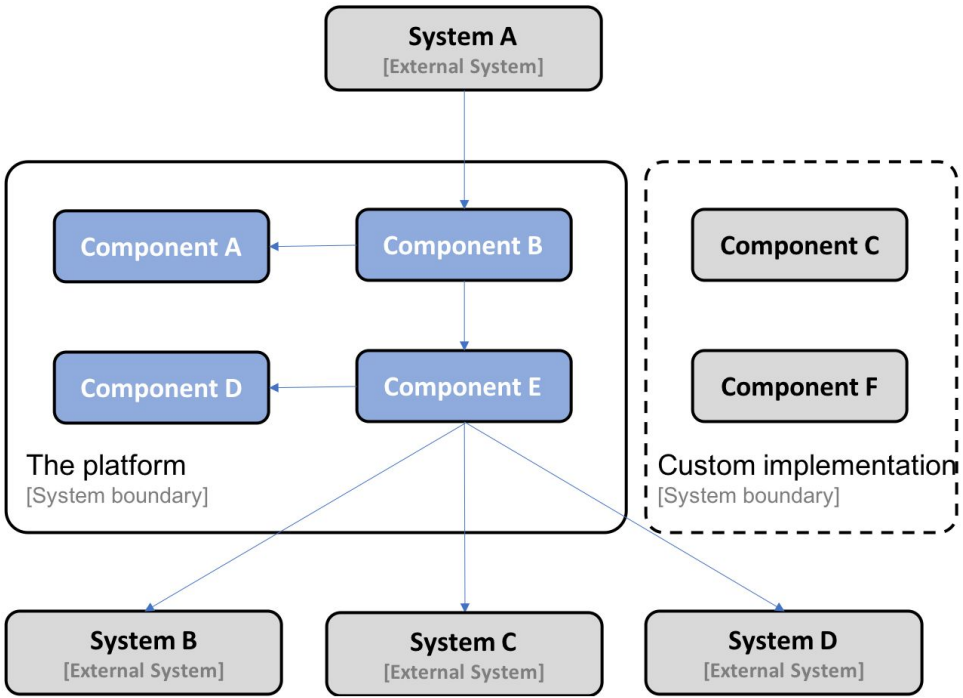
[System boundary]

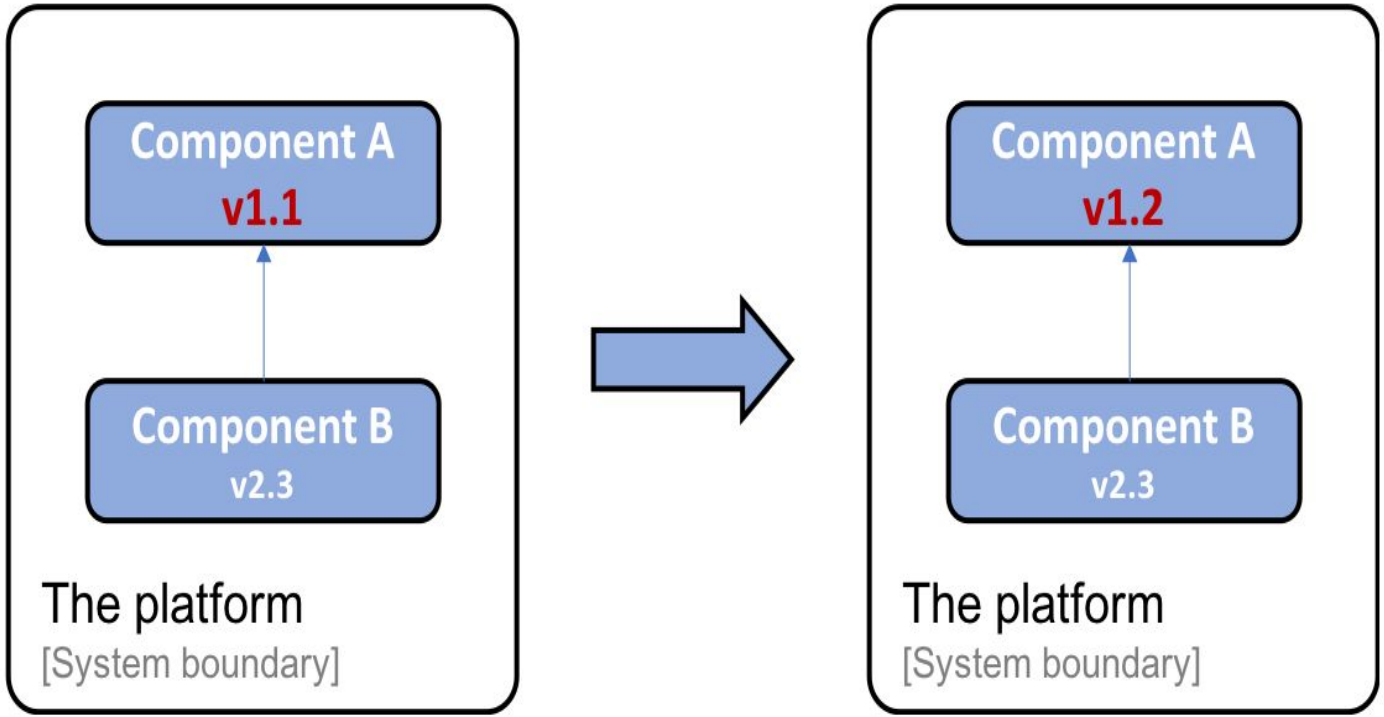
A two-node deployment

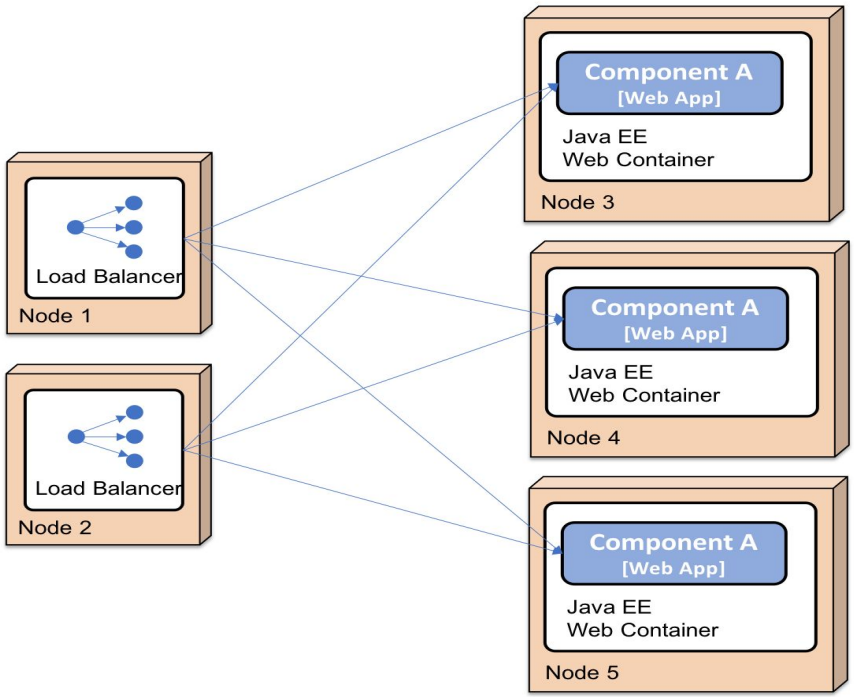


Benefits of autonomous software components


System A
[External System]
Component A
Component B
Component D
Component E
The platform
[System boundary]
System B
[External System]


System A
[External System]
Component A
Component B
Component C
Component D
Component E
Component F
The platform
[System boundary]
Custom implementation
[System boundary]
System B
[External System]
System C
[External System]
System D
[External System]


Component A
v1.1
Component B
v2.3
The platform
[System boundary]
Component A
v1.2
Component B
v2.3
The platform
[System boundary]


Component A
[Web App]
Java EE
Web Container
Node 3
Load Balancer
Node 1
Component A
[Web App]
Java EE
Web Container
Node 4
Load Balancer
Node 2
Component A
[Web App]
Java EE
Web Container
Node 5

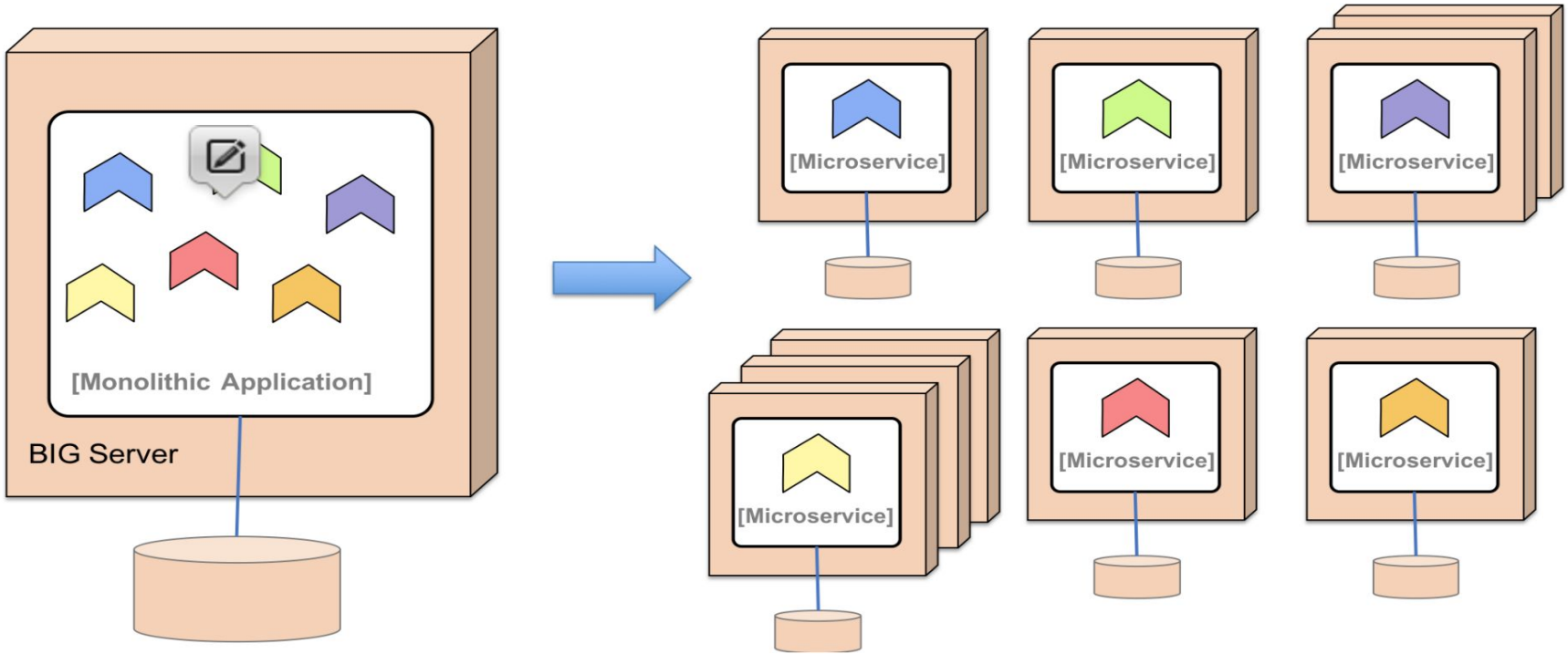
Challenges with autonomous software components

Challenges

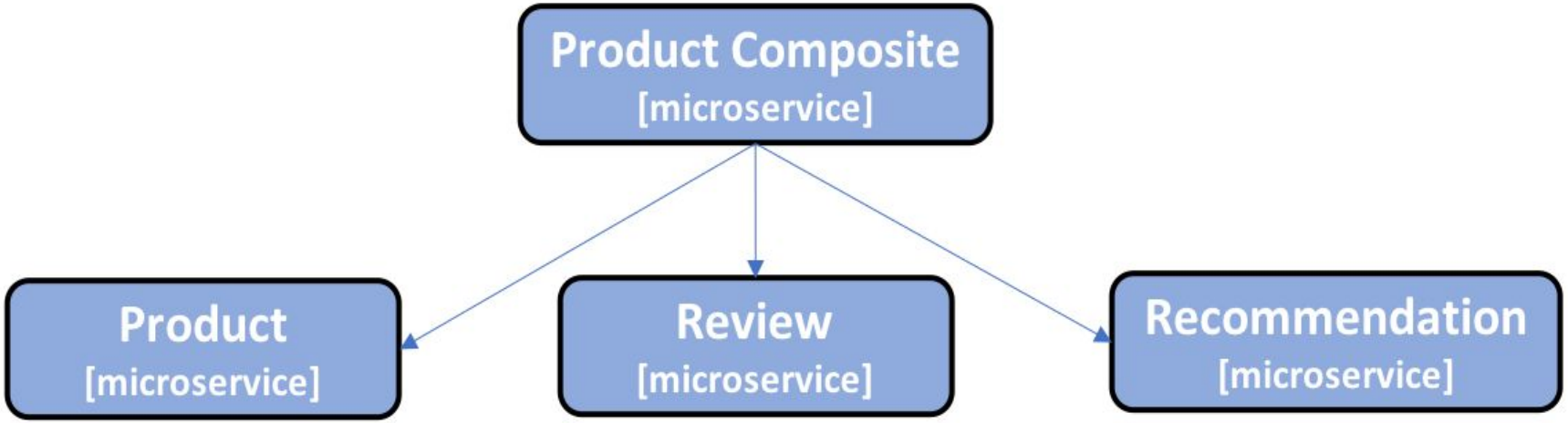
- Scalability
- Chain of failures
- Keeping the configuration consistent
- Monitoring the state of the platform
- Collecting log files

Enter Microservices

Defining a microservice



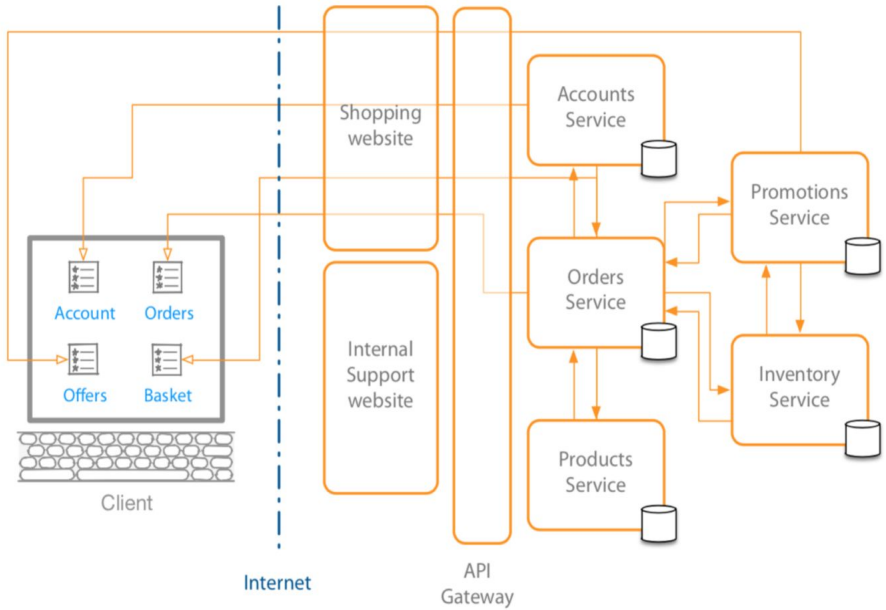
A sample microservice landscape



The microservice landscape

[System boundary]

Microservice - Ex



Microservice - characteristics

- Must conforms to a shared-nothing architecture
- Must only communicate with well defined interfaces
- Must be deployed as separate runtime process
- Microservice instances are stateless

Microservice - benefits

- Easy to develop,test and deploy
- Increased agility
- Ability to scale horizontally

Challenges with microservices

- Synchronous communication
- Keeping the configuration up to date
- It's hard to track a request
- Analyzing the usage of hardware resources on a component level
- Management of many small components can become costly and error-prone

The 8 fallacies of distributed computing

1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

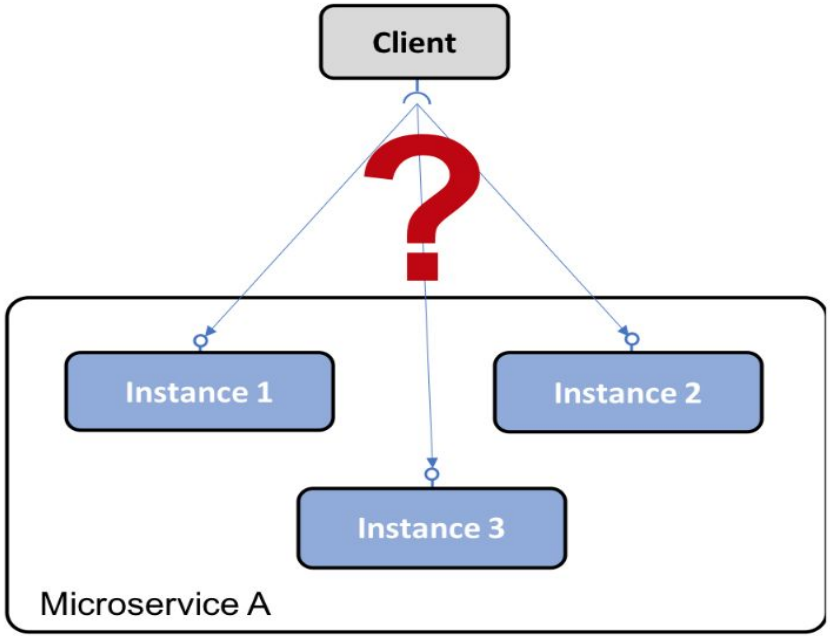
Note: The eighth fallacy was actually added by James Gosling at a later date. For more details, please go to https://www.rgoarchitects.com/Files/fallacies.pdf.

Design patterns for microservices

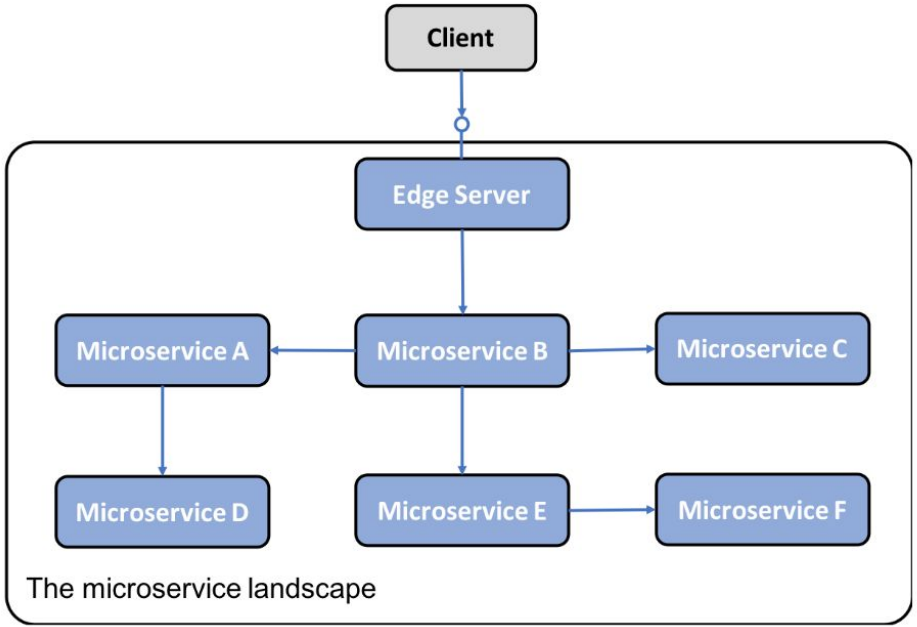
design patterns

- Service discovery
- Edge server
- Reactive microservices
- Central configuration
- Centralized log analysis
- Distributed tracing
- Circuit Breaker
- Control loop
- Centralized monitoring and alarms

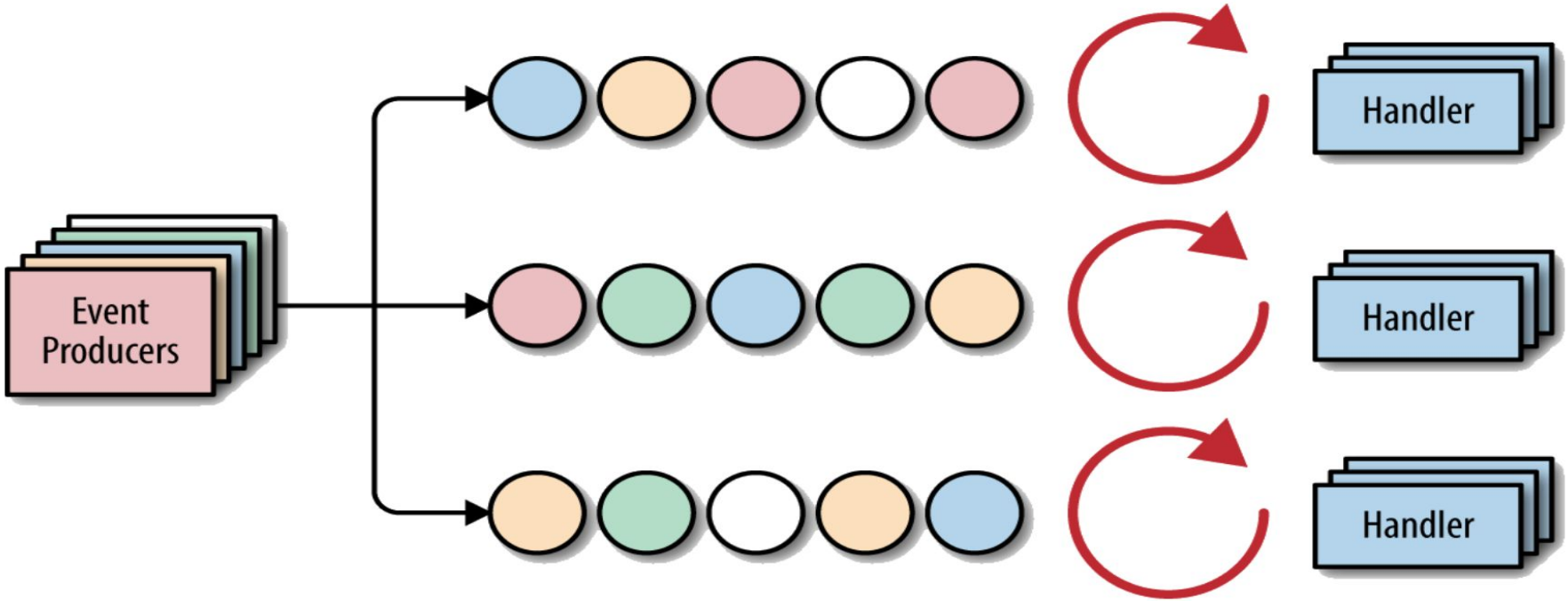
1. Service Discovery



2. Edge Server



3. Reactive Microservices

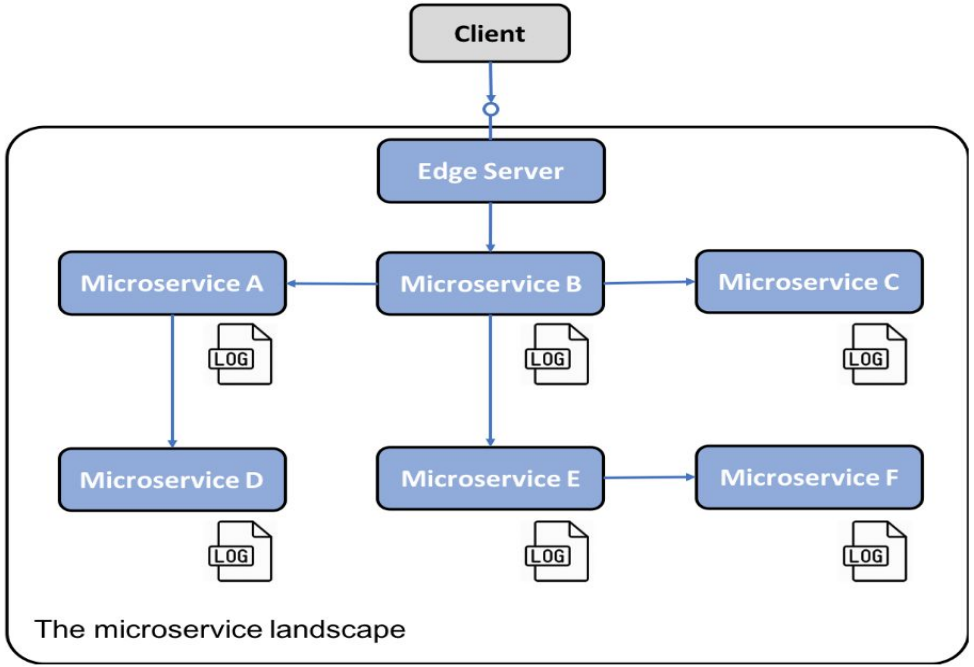


4. Central Configuration

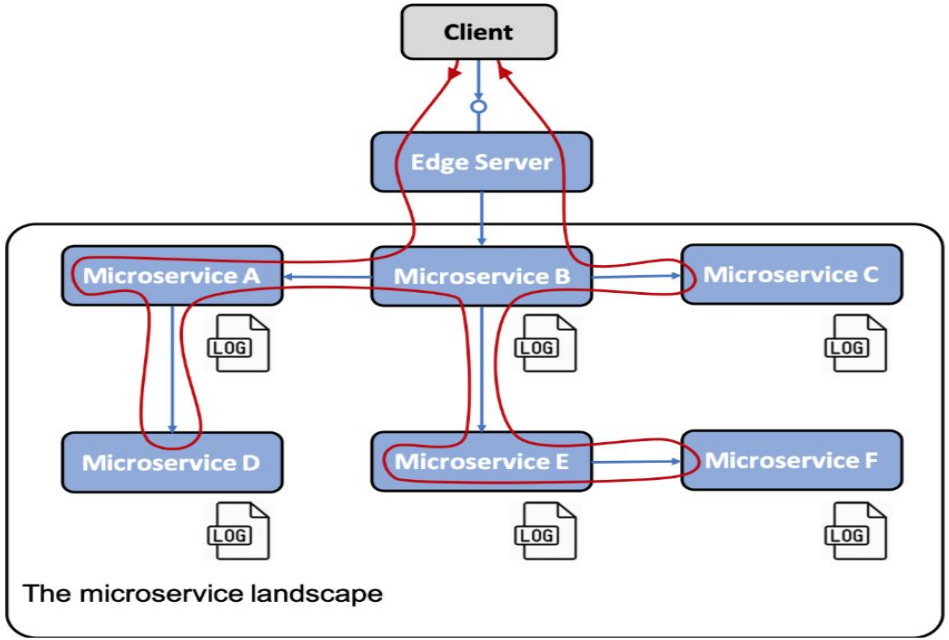
Add a new component,

a **configuration** server, to the system landscape to store the configuration of all the microservices.

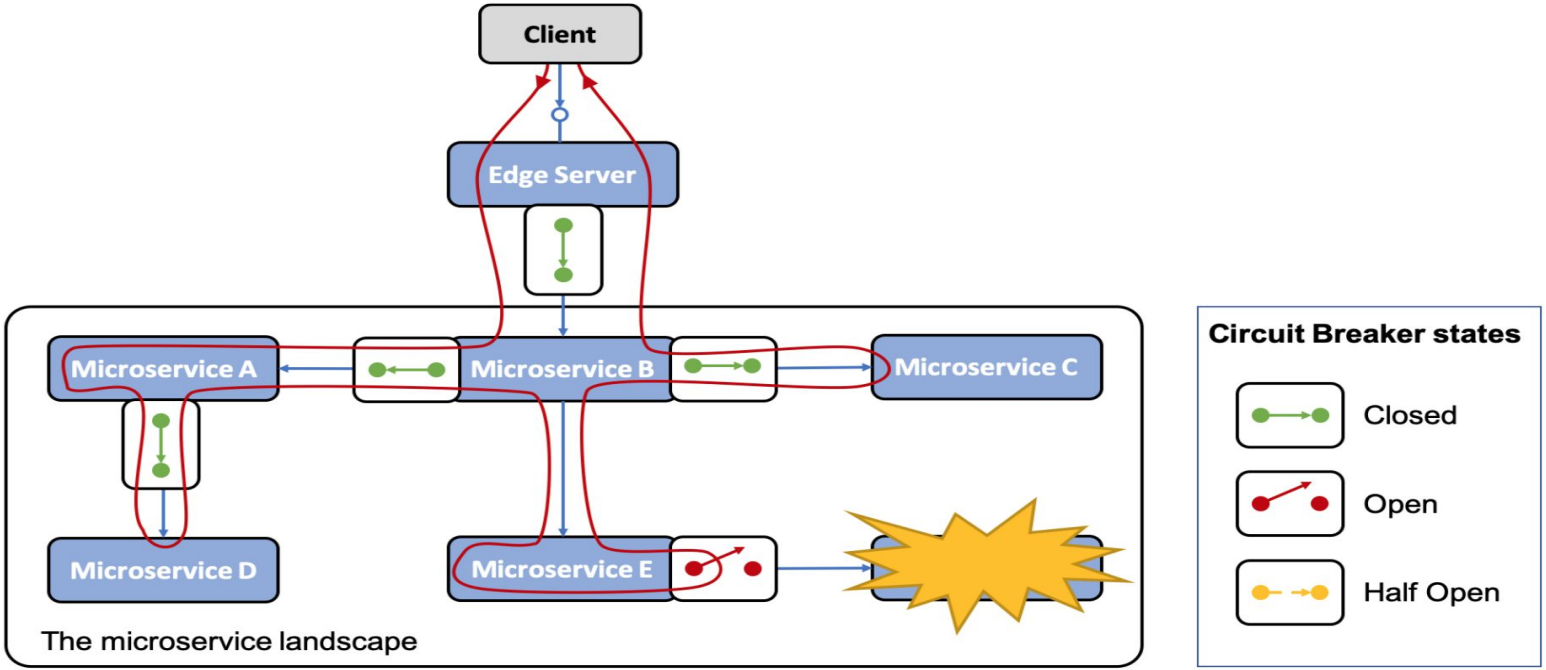
5. Centralized Log Analysis



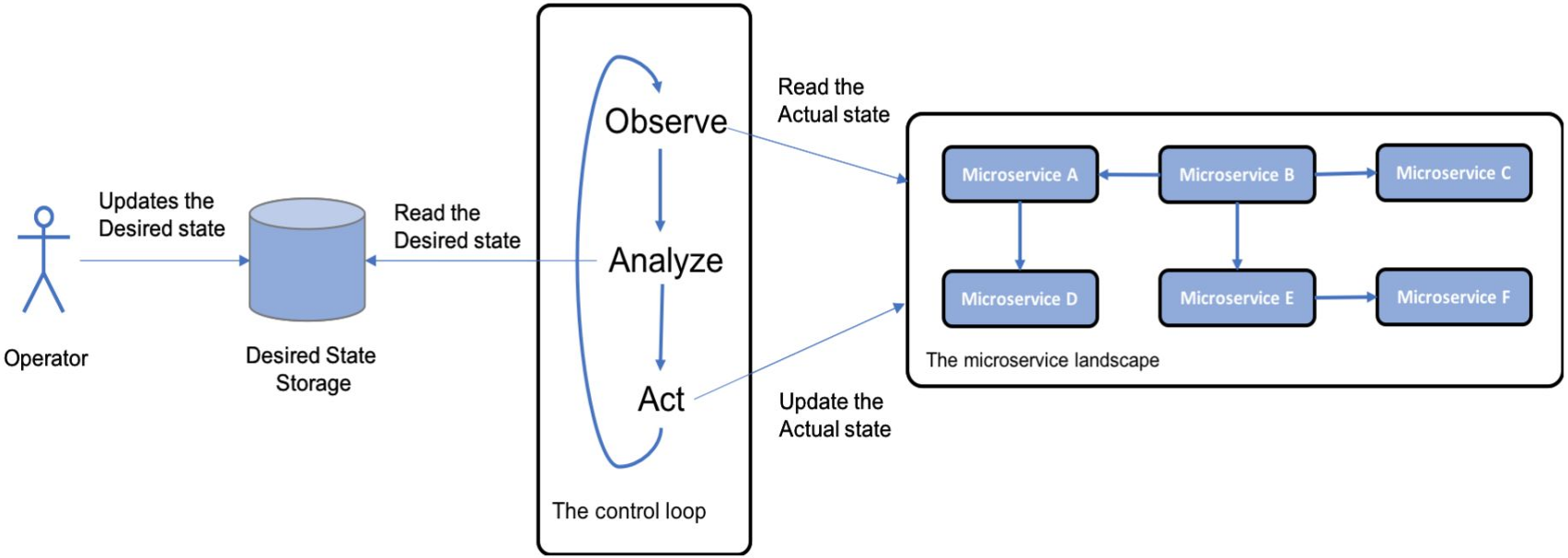
6. Distributed Tracing



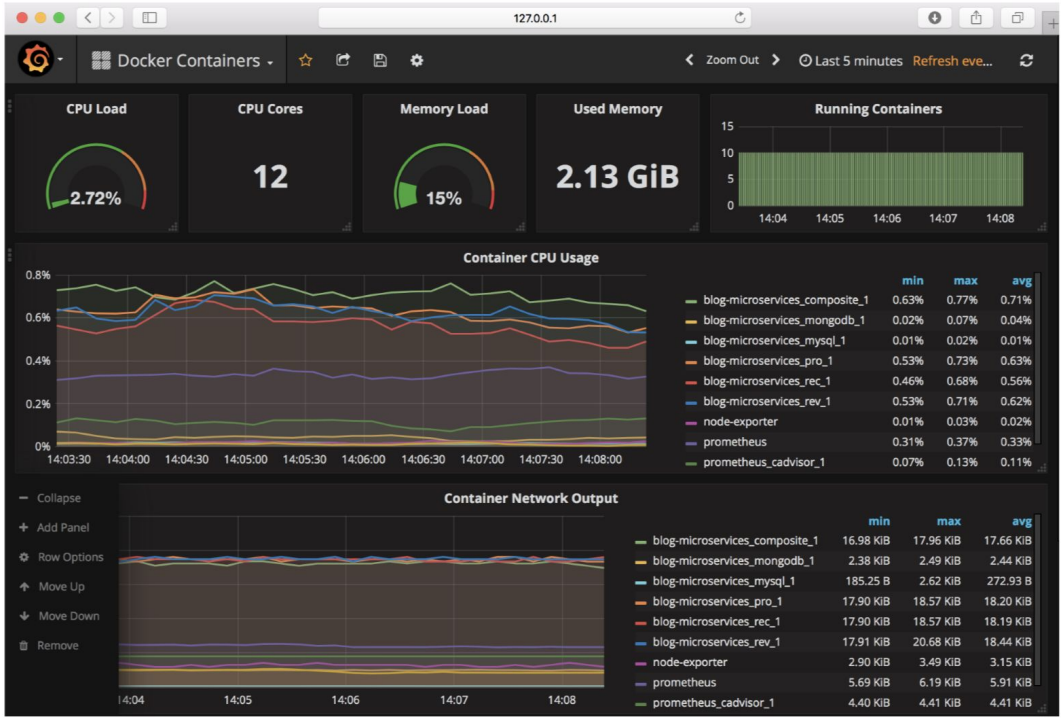
7. Circuit Breaker



8. Control Loop



9. Centralized monitoring & alarms



Software Enablers

Design Pattern	Spring Boot	Spring Cloud	Kubernetes	Istio
Service discovery		Netflix Eureka and Netflix Ribbon	Kubernetes `kube-proxy` and service resources	
Edge server		Spring Cloud and Spring Security OAuth	Kubernetes Ingress controller	Istio ingress gateway
Reactive microservices	Spring Reactor and Spring WebFlux			
Central configuration		Spring Config Server	Kubernetes `ConfigMaps` and Secrets	
Centralized log analysis			Elasticsearch, Fluentd, and Kibana **Note**: Actually not part of Kubernetes but can easily be deployed and configured together with Kubernetes	

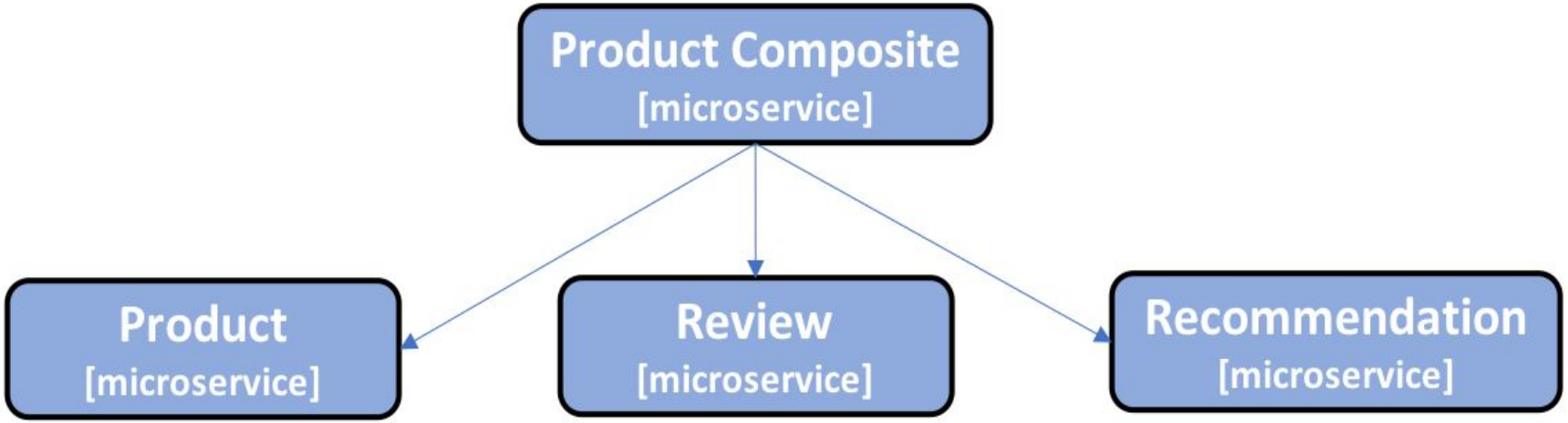
Software Enablers Cont...

Distributed tracing		Spring Cloud Sleuth and Zipkin		Jaeger
Circuit Breaker		Resilience4j		Outlier detection
Control loop			Kubernetes controller manager	
Centralized monitoring and alarms			Grafana and Prometheus **Note:** Actually not part of Kubernetes but can easily be deployed and configured together with Kubernetes	Kiali, Grafana, and Prometheus

Other important consideration

- Importance of Dev/Ops
- Organizational aspects
- Decomposing a monolithic application into microservices
- Importance of API design
- Migration paths from on-premise to the cloud
- Good design principles for microservices, the 12-factor app
    - https://12factor.net

A sample microservice landscape - demo



The microservice landscape

[System boundary]

Demo

- Creating a Set of Cooperating Microservices

Demo

- Deploying Our Microservices Using Docker

What does framework do at startup time

- Parse config files
- Classpath & classpath scanning
  - For annotations ,getters or other metadata
- Build framework metamodel objects
- Prepare reflection and build proxies
- Start and open IO, threads, etc

- Move startup time to build time

Build time benefits

- Do the work once, not each start
- All the bootstrap classes are no longer needed
- Less time to start , less memory used
- Less or no reflection nor dynamic proxy

Reactive Programming and Vert.x

Imperative Programming

```
URL url = new URL("http://acme.com/");
BufferedReader in = new BufferedReader(new InputStreamReader(url.openStream()));

String line;
while ((line = in.readLine()) != null) {
  System.out.println(line);
}
in.close();
```