Microservices

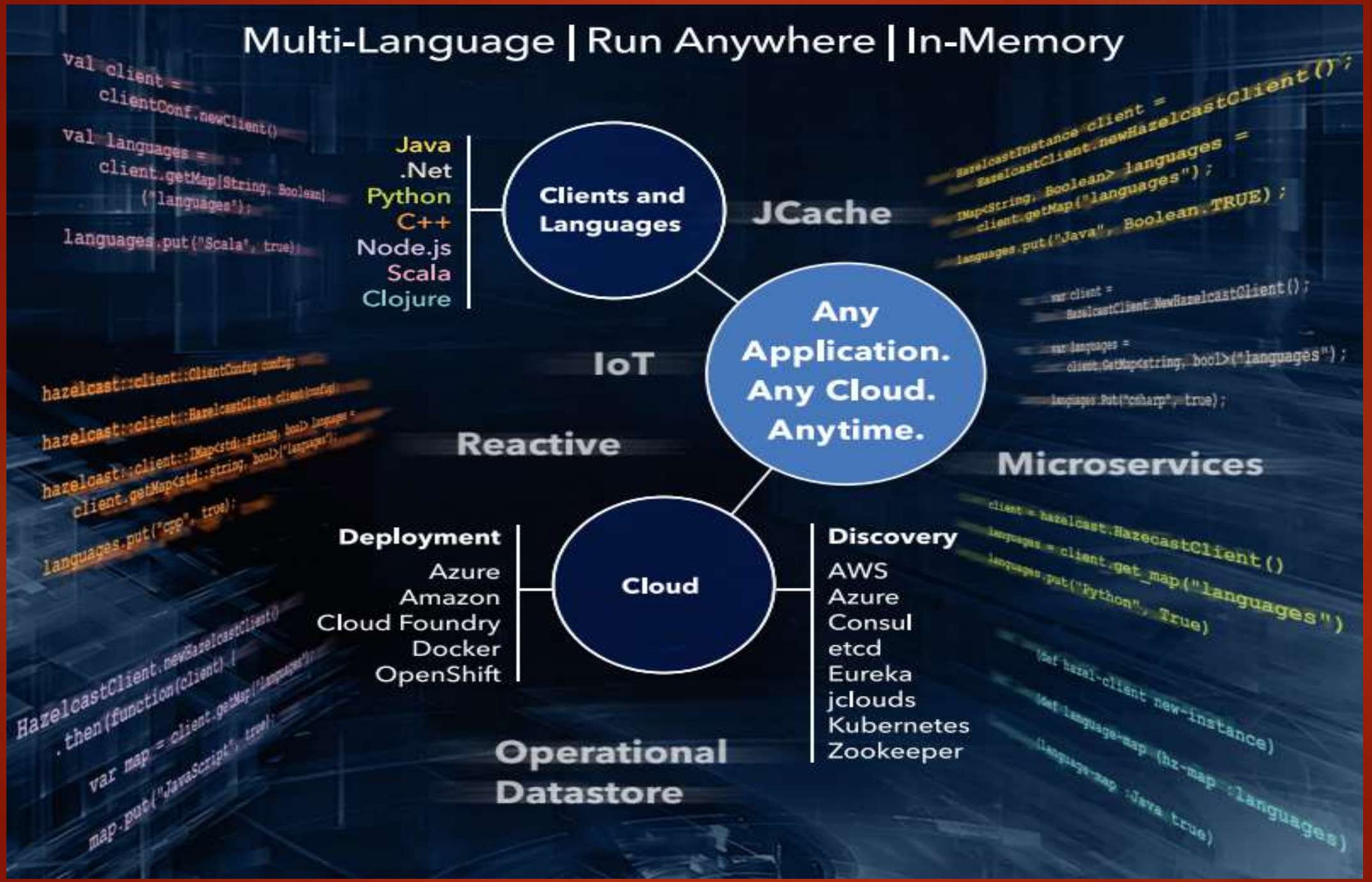
By Narendranath Reddy

Name: Narendranath Reddy Thota

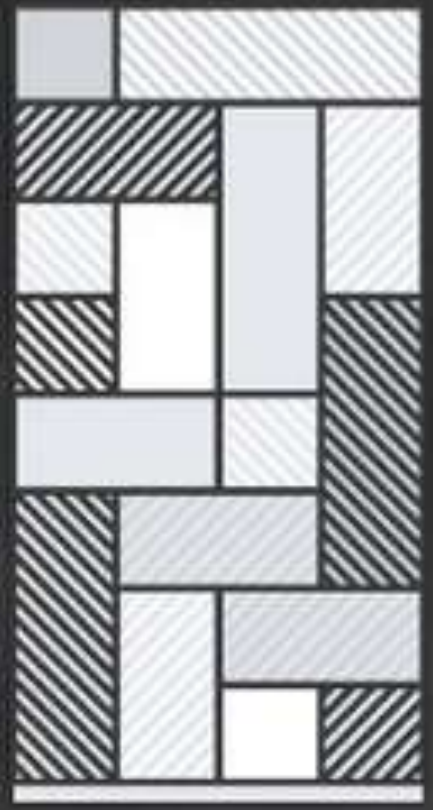
- d3 Whitepaper Core Team
- Trainer from Gama
- Blockchain Full Stack Developer
- Technology Analyst
- Software Developer
- Technology Speaker

MAERSK
Email: narendranath.thota@ust-global.com
P: +917288838869

UST Global®
Innovation • Information • Technology


Multi-Language | Run Anywhere | In-Memory
Java
.Net
Python
C++
Node.js
Scala
Clojure
Clients and Languages
JCache
IoT
Reactive
Any Application. Any Cloud. Anytime.
Microservices
Deployment
Azure
Amazon
Cloud Foundry
Docker
OpenShift
Cloud
Discovery
AWS
Azure
Consul
etcd
Eureka
jclouds
Kubernetes
Zookeeper
Operational Datastore

“The Monolith”



Challenges with monolithic software

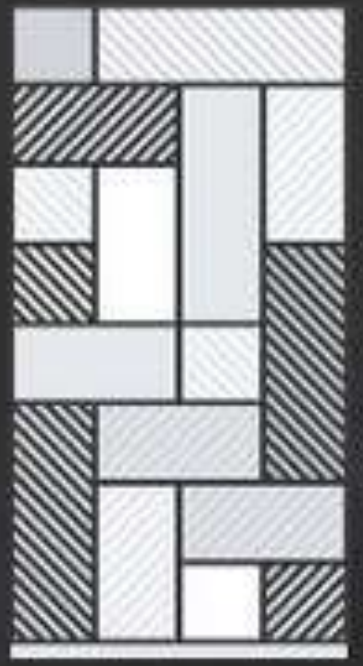
Difficult to scale	Architecture is hard to maintain and evolve	Lack of agility
Long Build/Test/Release Cycles (who broke the build?)	New releases take months	Lack of innovation
Operations is a nightmare (module X is failing, who's the owner?)	Long time to add new features	Frustrated customers

Monolith development lifecycle

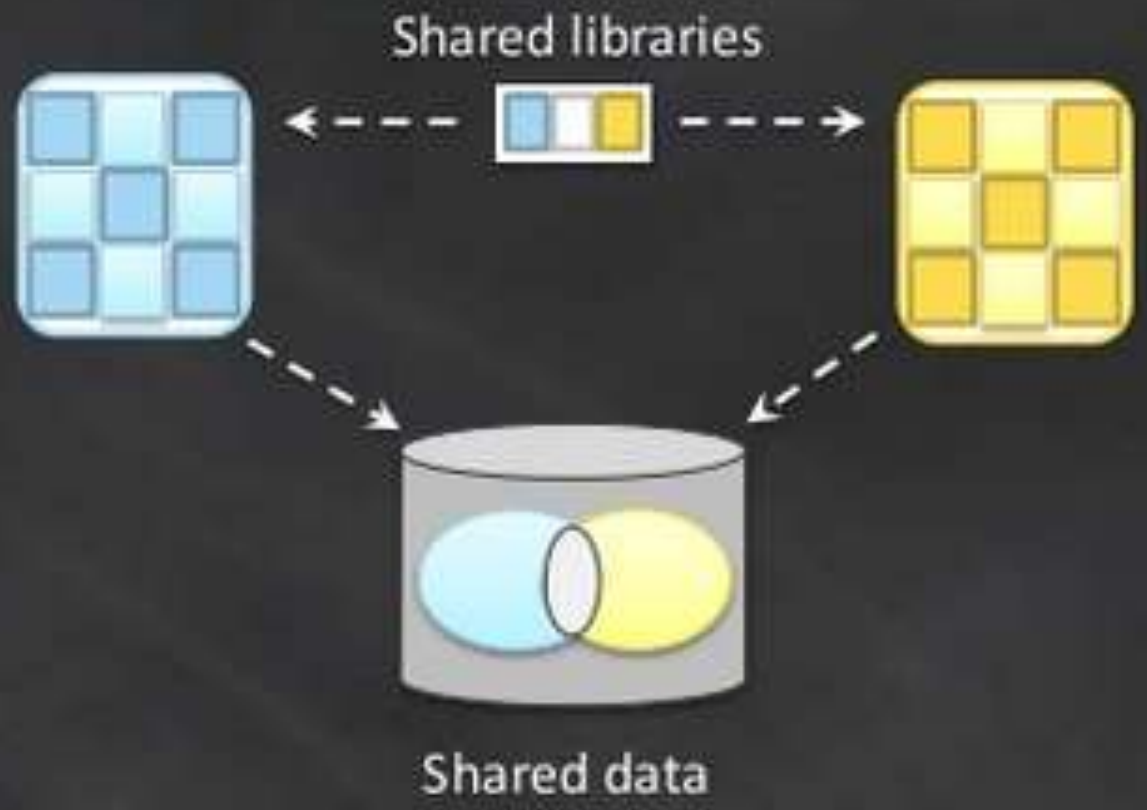
developers

delivery pipeline

app
(aka the“monolith”)



Too much software coupling



Monolithic Architecture

Load Balancer

Monolithic App

Account Component

Catalog Component

Recommendation Component

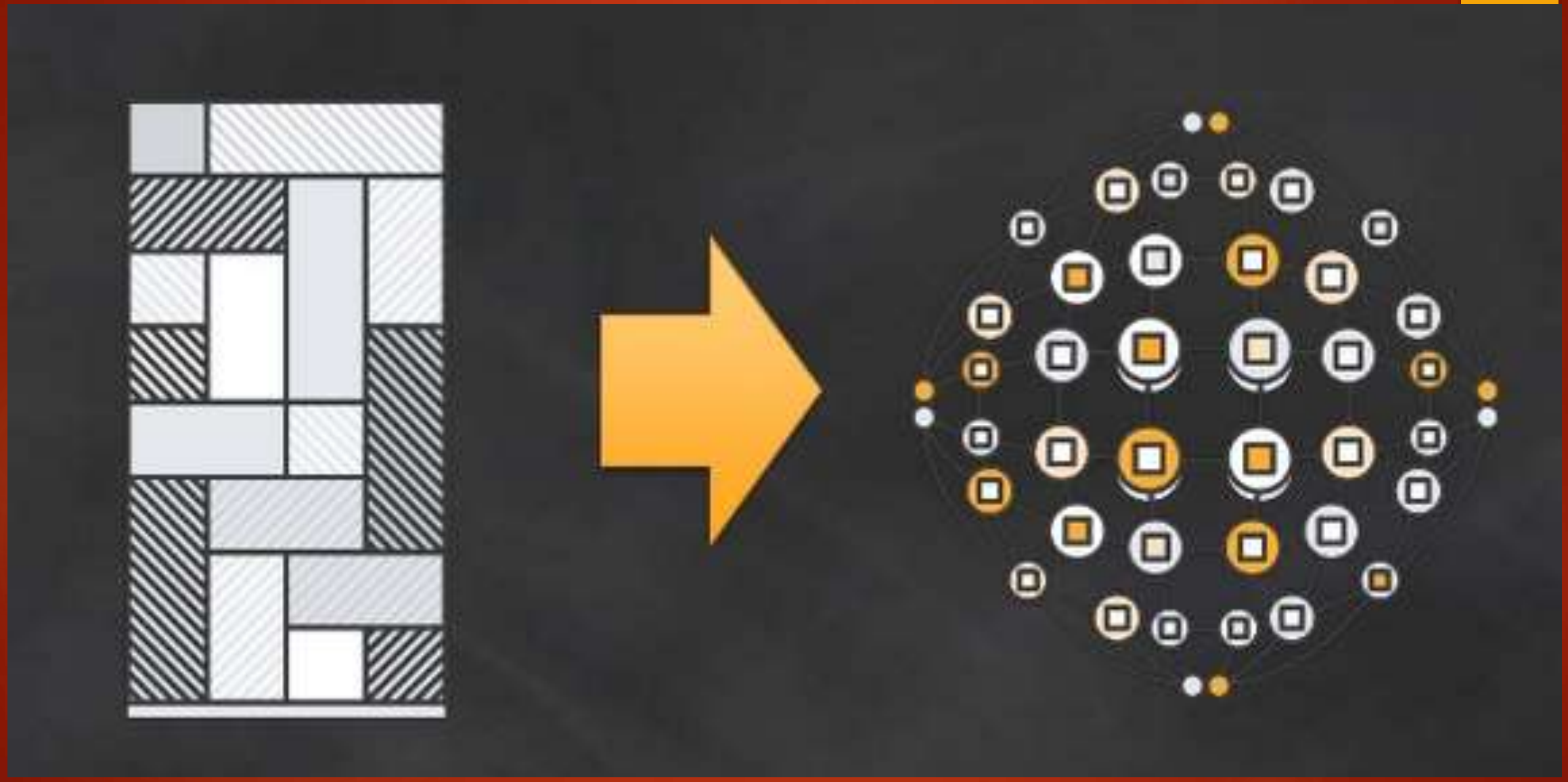
Customer Service Component

Database

Comparing Monolithic to MicroServices

MONOLITHIC APP (VARIOUS COMPONENTS LINKED TOGETHER)


CHANGE



MicroServices – separate single purpose services

Microservices Architecture

Load Balancer

API Gateway

Account Service

Catalog Service

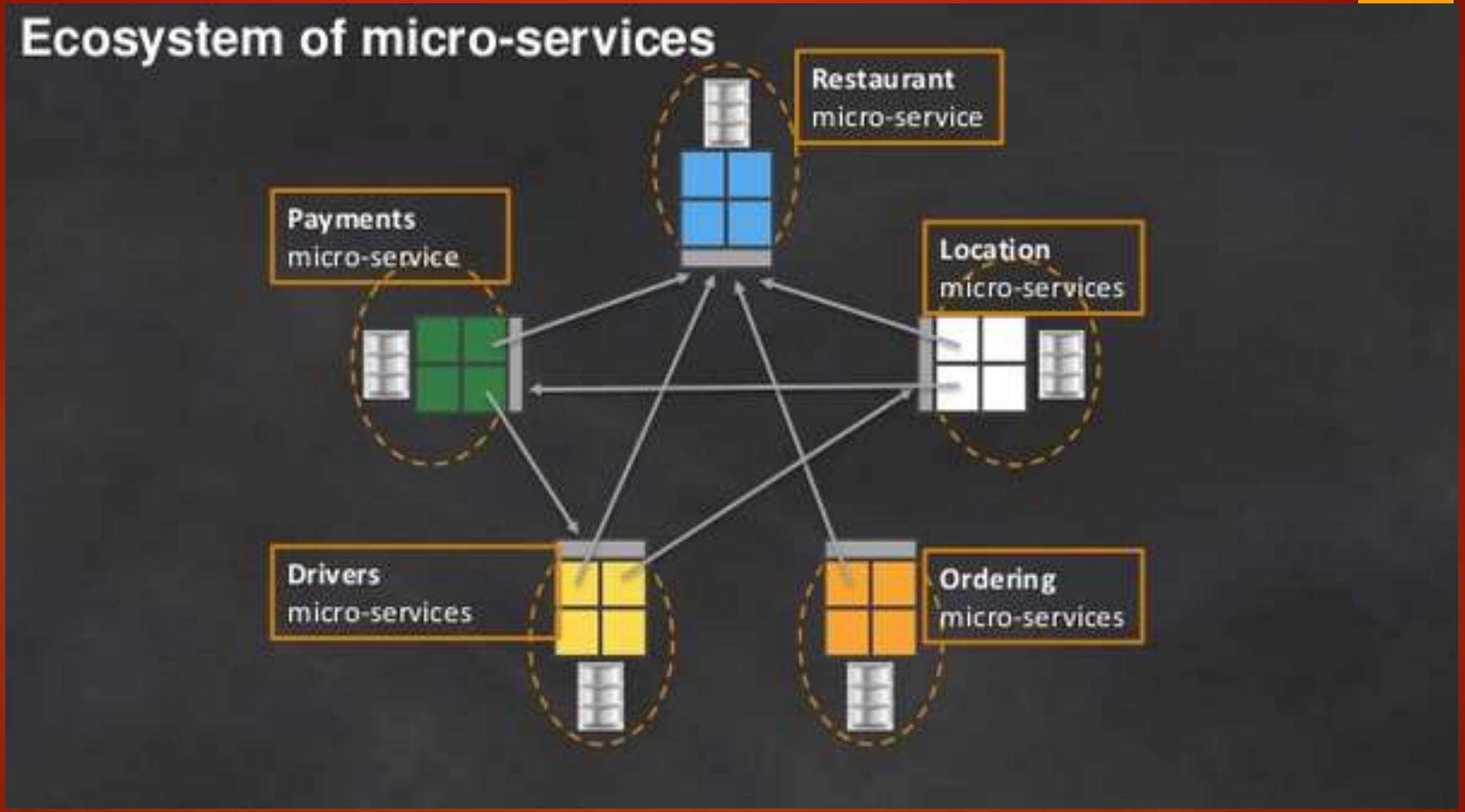
Catalog DB

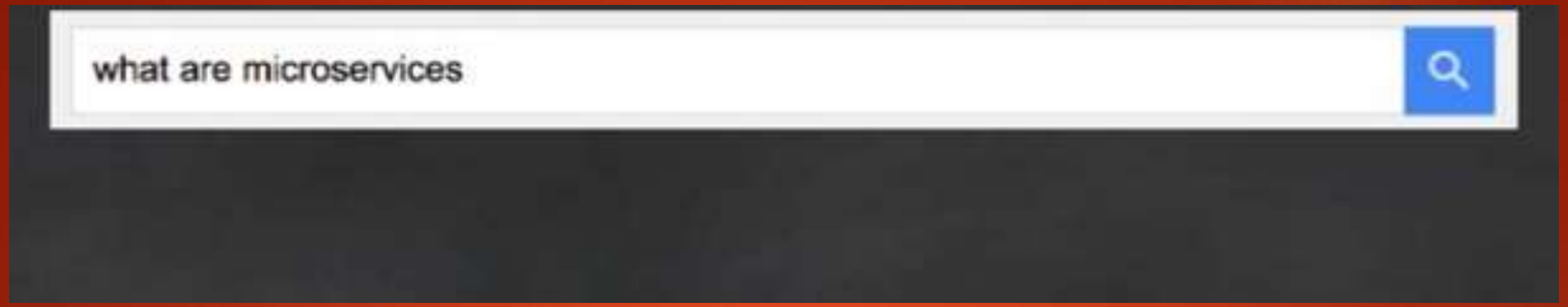
Recommendation Service

Customer Service Service

Customer DB

Ecosystem of micro-services





- Micro service architecture (MSA) is an approach to building software systems that decomposes business domain models into smaller, consistent, bounded-contexts implemented by services.
- Typically implemented and operated by small teams.
- Switching from SOAP to REST doesn’t make a micro services architecture.
- Micro services are not a technology-only discussion.

"service-oriented architecture

composed of

loosely coupled elements

that have

bounded contexts"

Services communicate with each other over the network

Adrian Cockcroft (former Cloud Architect at Netflix, now Technology Fellow at Battery Ventures)

"service-oriented architecture composed of **loosely coupled elements** that have **bounded contexts**"

Adrian Cockcroft (former Cloud Architect at Netflix, now Technology Fellow at Battery Ventures)

You can update the services independently; updating one service doesn't require changing any other services.

"service-oriented
architecture
composed of
loosely coupled
elements
that have
bounded contexts"

Adrian Cockcroft (former Cloud Architect at Netflix, now Technology Fellow at Battery Ventures)

Self-contained; you can update the code without knowing anything about the internals of other microservices

Design Principles for Monoliths:

- DDD
- SoC using MVC
- High cohesion, low coupling
- DRY
- CoC
- YAGNI

Design Patterns for Micro services:

- Aggregator Pattern
- Proxy Pattern
- Chained Pattern
- Branch Pattern
- Shared Resources
- Async Messaging and etc ..

Principle 1: Microservices only rely on each other's public API (Hide Your Data)

Principle 2: Use the right tool for the job
(Embrace polyglot programming frameworks)

Principle 2: Use the right tool for the job
(Embrace polyglot programming frameworks)

Disadvantages of Monolith:

- Difficult to deploy and maintain
- Obstacle to frequent deployments
- Dependency between unrelated features
- Makes it difficult to try out new technologies/framework

Advantages of micro services:

Easier to develop, understand, maintain

- Starts faster than a monolith, speeds up deployments
- Local change can be easily deployed, great enabler of CD
- Each service can scale on X- and Z-axis
- Improves fault isolation
- Eliminates any long-term commitment to a technology stack
- Freedom of choice of technology, tools, frameworks

Availability

- A single missing “;” brought down the Netflix website for many hours (~2008)

MONOLITHIC APPS – FAILURE & AVAILABILITY

Introduction to some popular technology components, how they help solve some of the problems of developing and delivering software using a micro services architecture.

- Spring Boot (spring cloud – Eureka, ZUUL, Hystrix, Ribbon, etc)
- Docker
- Nodejs
- Kubernetes/Openshift
- NetflixOSS
- Kafka , Rabbit MQ, CloudAMQP, Kestrel etc..
- Zipkin
- ELK stack
- Prometheus with Grafana
- AppDynamics
- Akana
- ForgeRock
- API Connect etc…

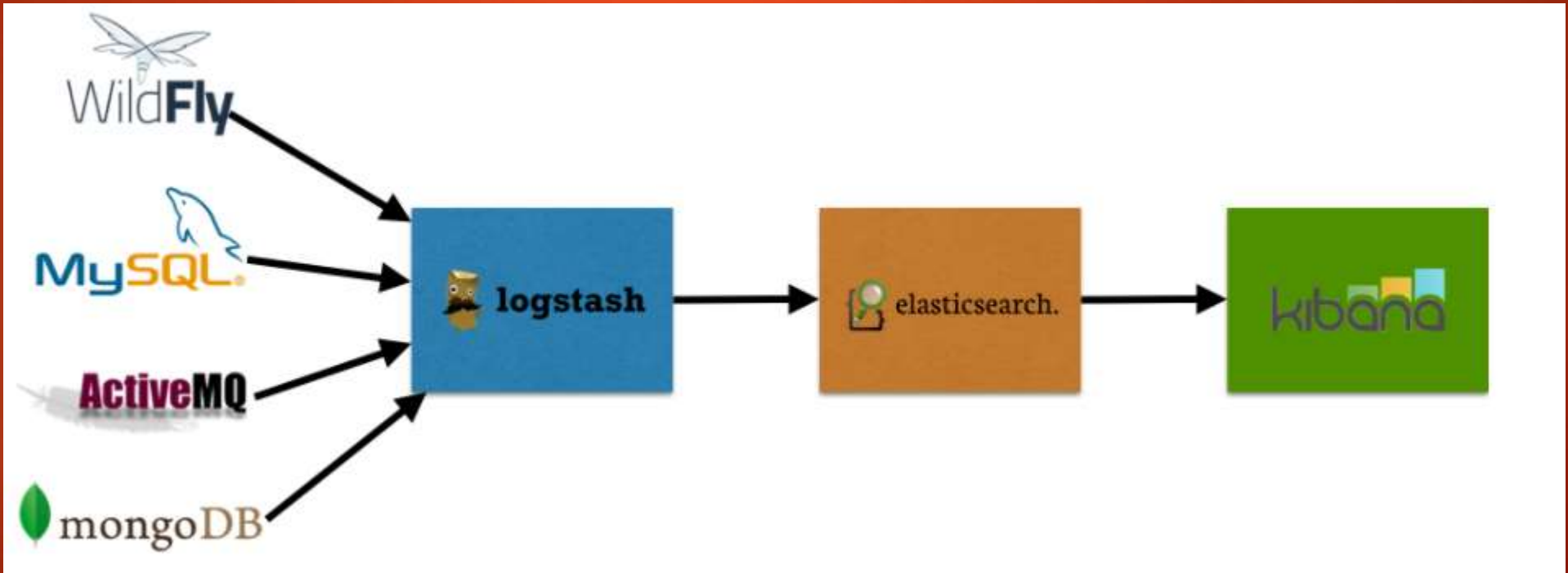
Monitoring:

- One of the most frequently mentioned challenges related to the creation of micro services-based architecture is monitoring. Each micro service should be run in an environment isolated from the other micro services so it does not share resources such as databases or log files with them.

- However, the essential requirement for micro services architecture is that it is relatively easy to access the call history, including the ability to look through the request propagation between multiple micro services. Grepping the logs is not the right solution for that problem. There are some helpful tools that can be used when creating micro services with Spring Boot and Spring Cloud frameworks.

- Zipkin. A distributed tracing system that helps gather timing data for every request propagated between independent services. It has simple management console where we can find a visualization of the time statistics generated by subsequent services.

- **ELK**. Elasticsearch, Logstash, and Kibana — three different tools usually used together. They are used for searching, analyzing, and visualizing log data in real-time.
- Distributed, independent micro services and centralized log monitoring make for the right solution. With tools like ELK and Zipkin, microservices monitoring seems to not be a very difficult problem to solve. There are also some other tools — for example, Hystrix and Turbine Prometheus with Grafana and etc.. — that provide real-time metrics for the requests processed by micro services.



Some Tools for Service Registry/Discovery:

- Zookeeper and
- Curator
- Kubernetes
- etcd
- Consul
- OSGi
- Snoop

NoOps:

- Service replication (Kubernetes)
- Dependency resolution (Nexus)
- Failover (Circuit Breaker)
- Resiliency (Circuit Breaker)
- Service monitoring, alerts and events (ELK)

Open Source

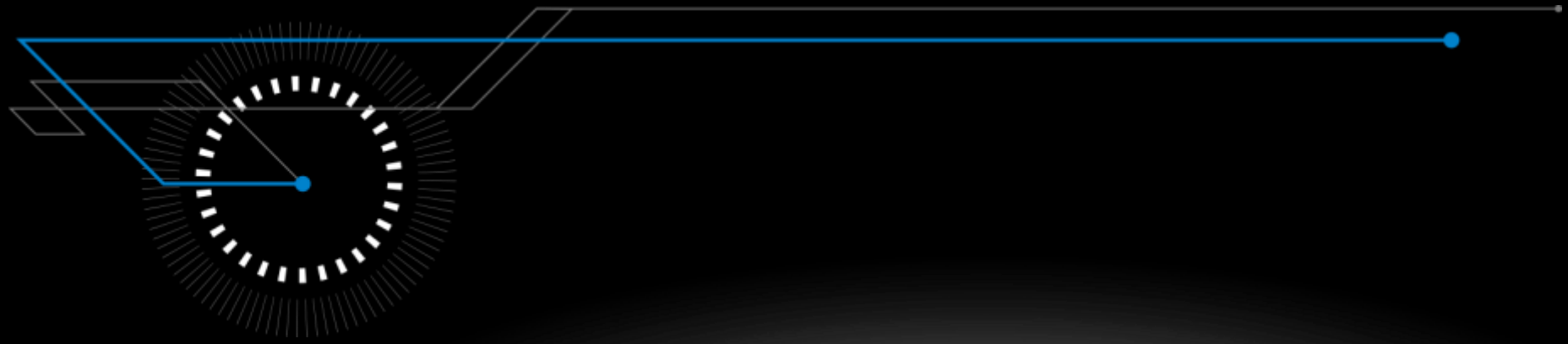
- Commoditization of Technology

  Open source is leading the charge in the technology space.

  This drives communities to build things like operating systems (Linux), programming languages (Go), message queues (Apache ActiveMQ), and web servers (httpd).

  As open source and open ecosystems have become the norm, we're starting to see a lot of innovation in software technology coming directly from open source communities (e.g., Apache Spark, Docker, and Kubernetes, ELK Stack, Zipkin).

Micro Services - Challenges



CHALLENGES

Can lead to chaos if not designed right …

CHALLENGES

Requires thinking differently about how to build, deploy and operate them.

- Design for Faults

  Building distributed systems is different from building shared memory, single process, monolithic applications.

  Networks are inherently unreliable.

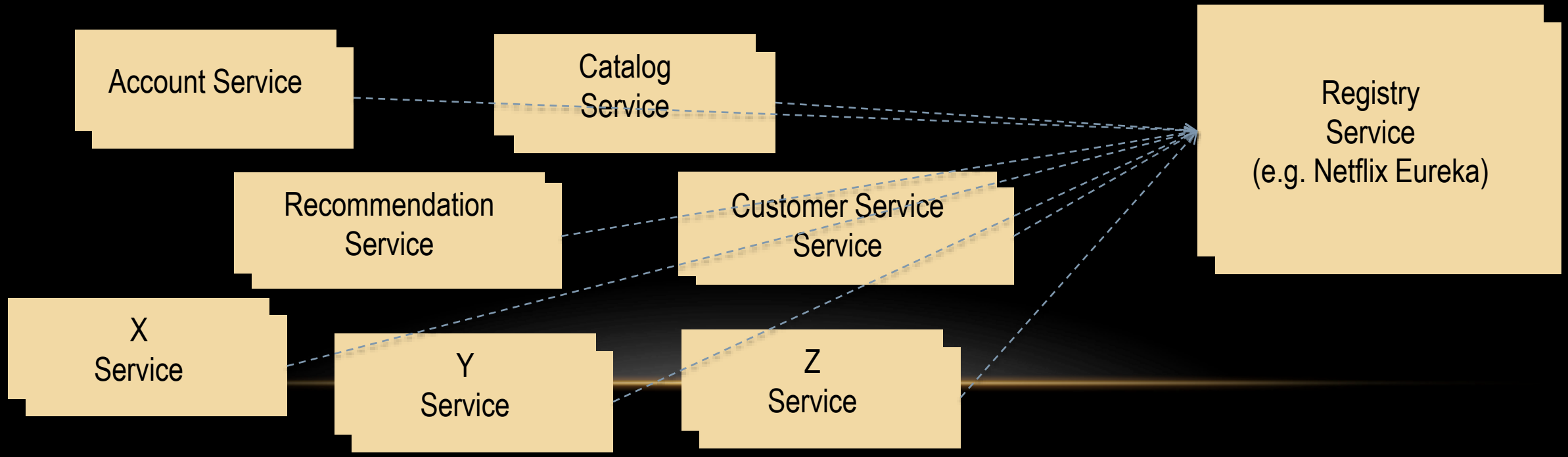
  Latent network calls can be very difficult to debug.

- Design with Dependencies in Mind

  Need loose coupling in our teams, in our technology, and our governance.

SERVICE DISCOVERY

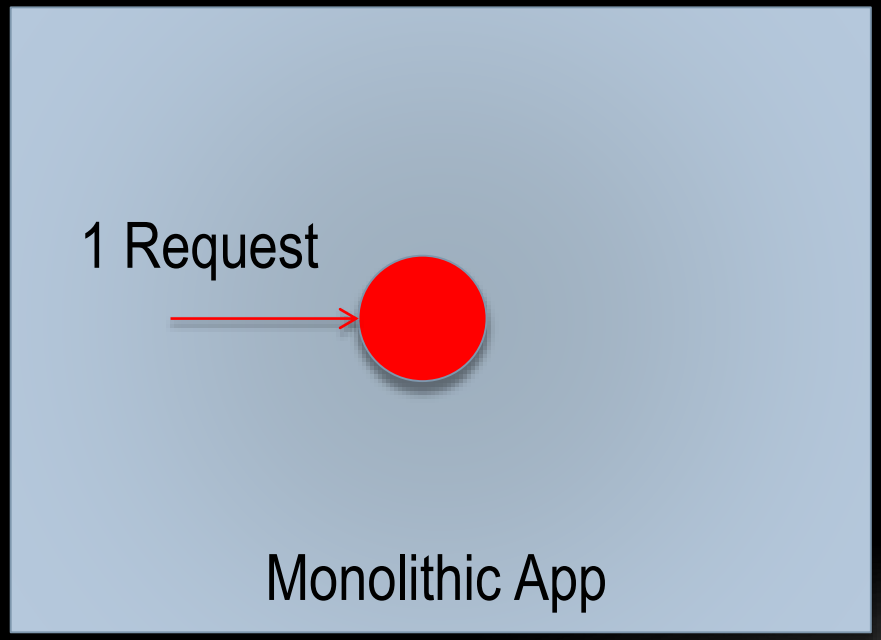
- 100s of MicroServices
  - Need a Service Metadata Registry (Discovery Service)

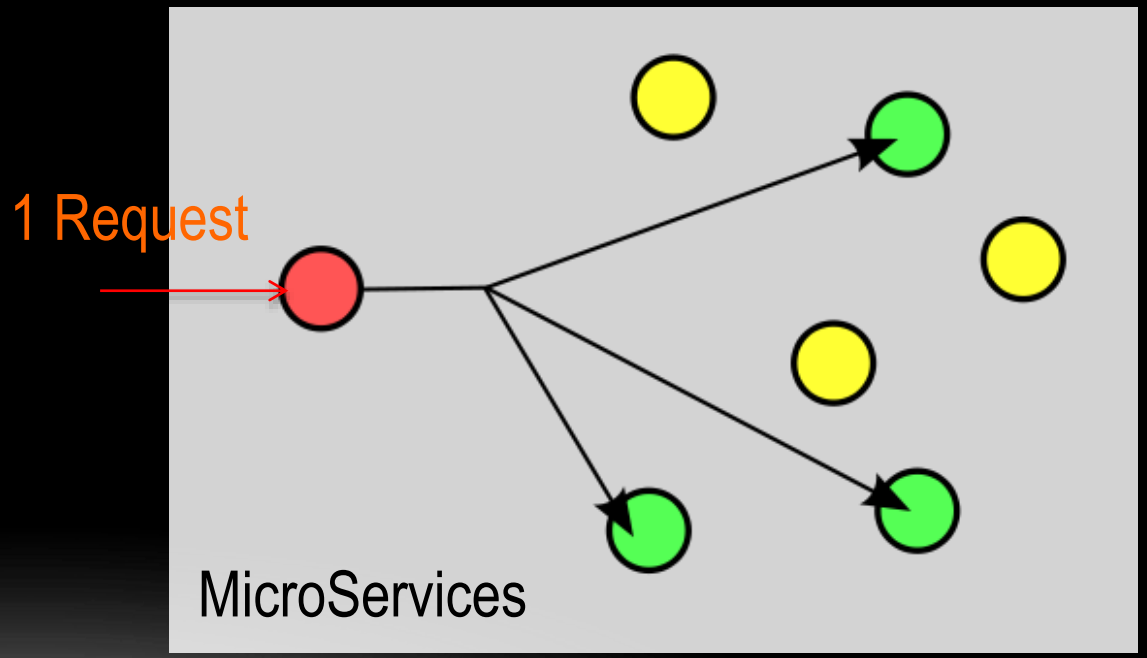


CHATTINESS (AND FAN OUT)

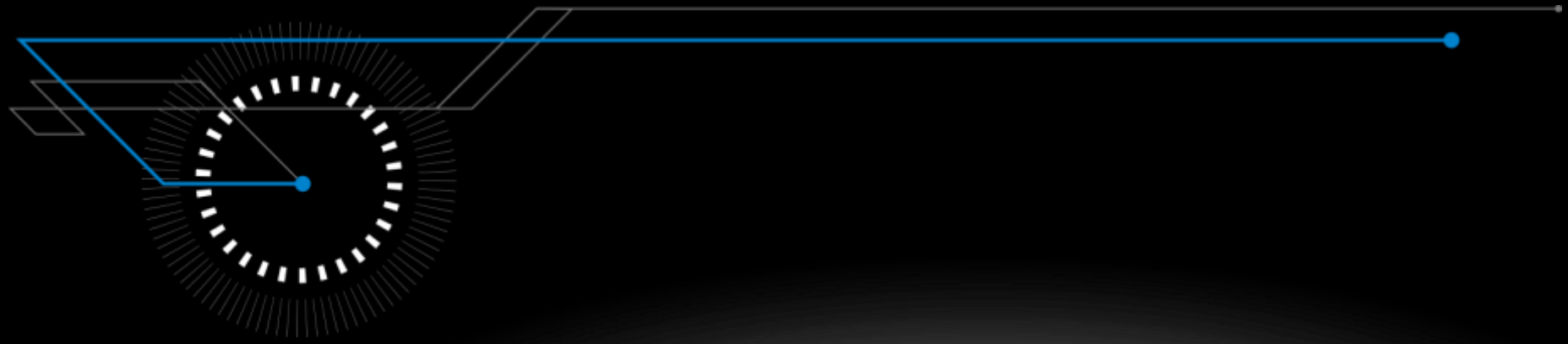
~2 Billion Requests per day on Edge Service

Results in ~20 Billion Fan out requests in ~100 MicroServices





Best Practices/Tips



Best Practice -> Loadbalancers

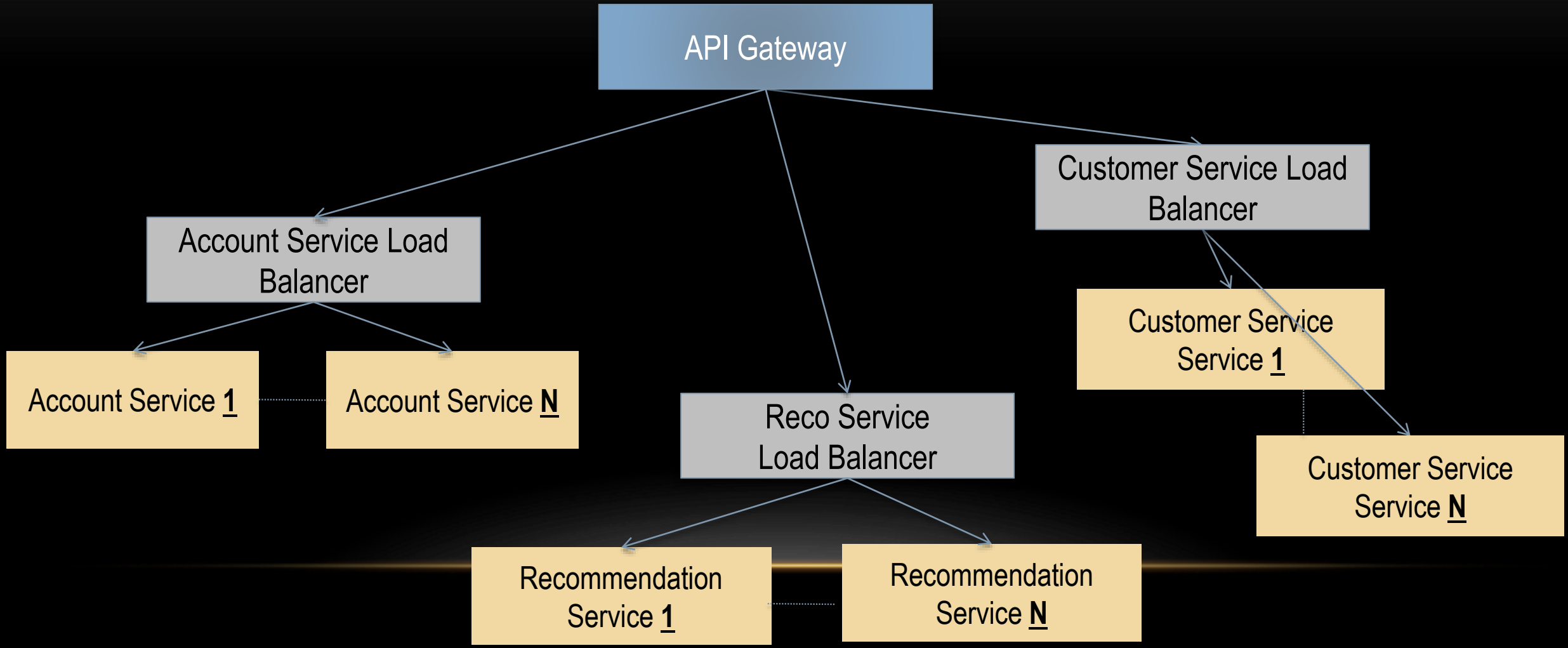
Choice

1. Central Loadbalancer?

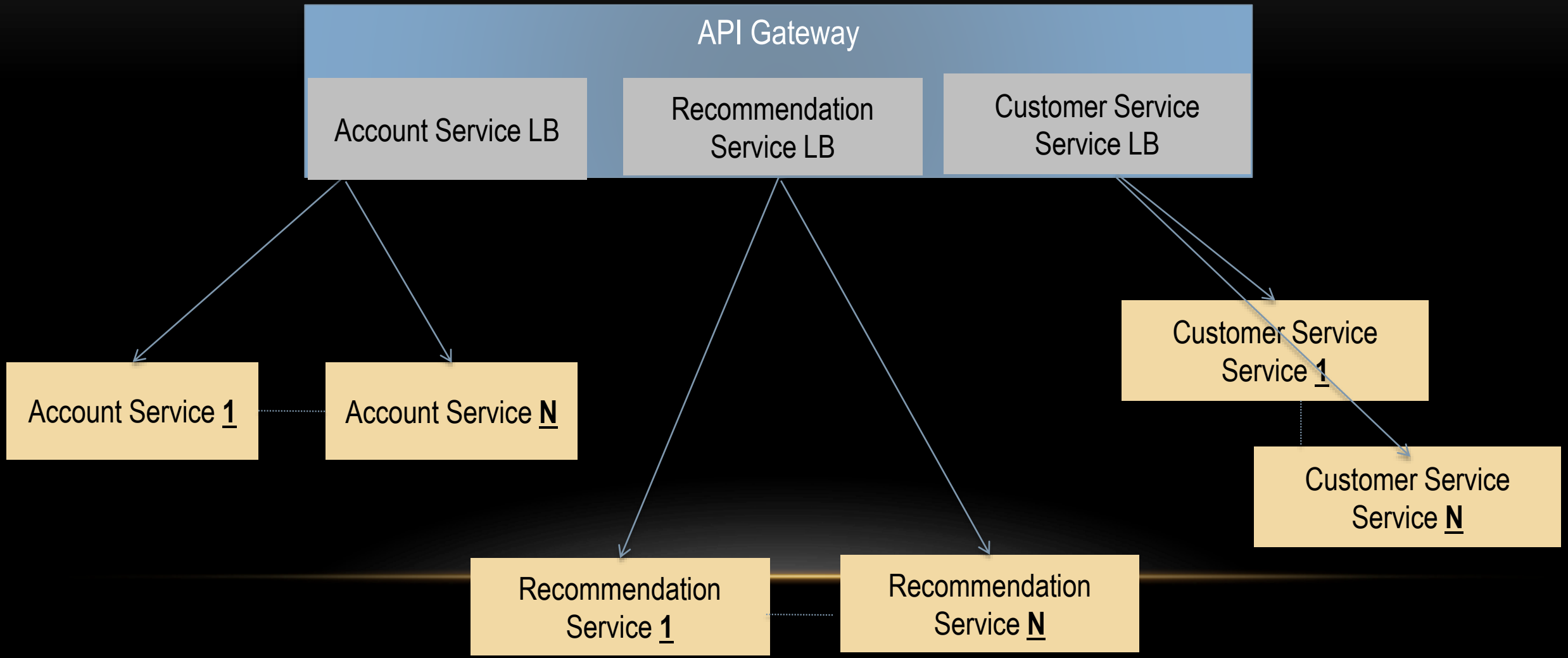
OR

2. Client based Loadbalancer?

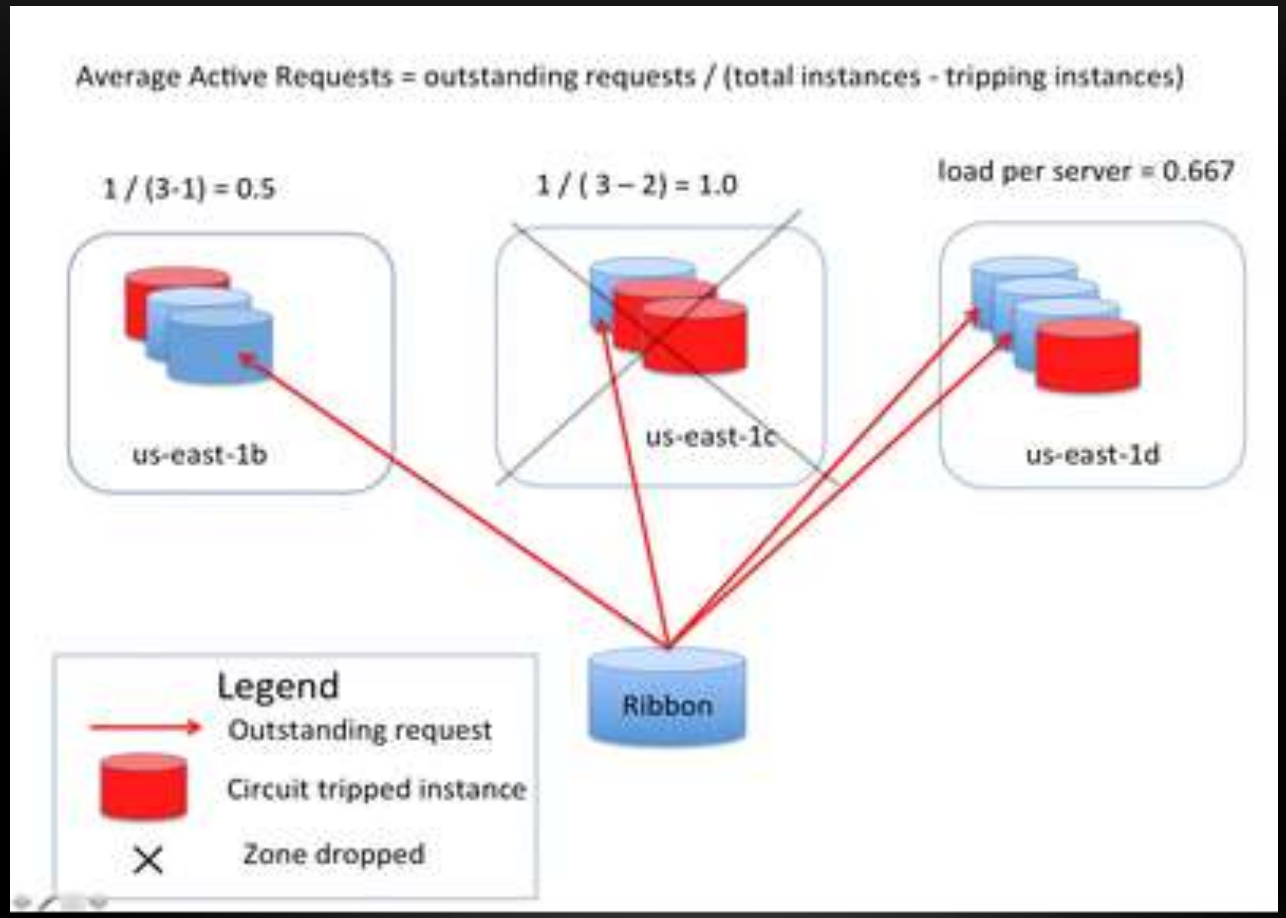
Central (Proxy) Loadbalancer



Client Loadbalancer

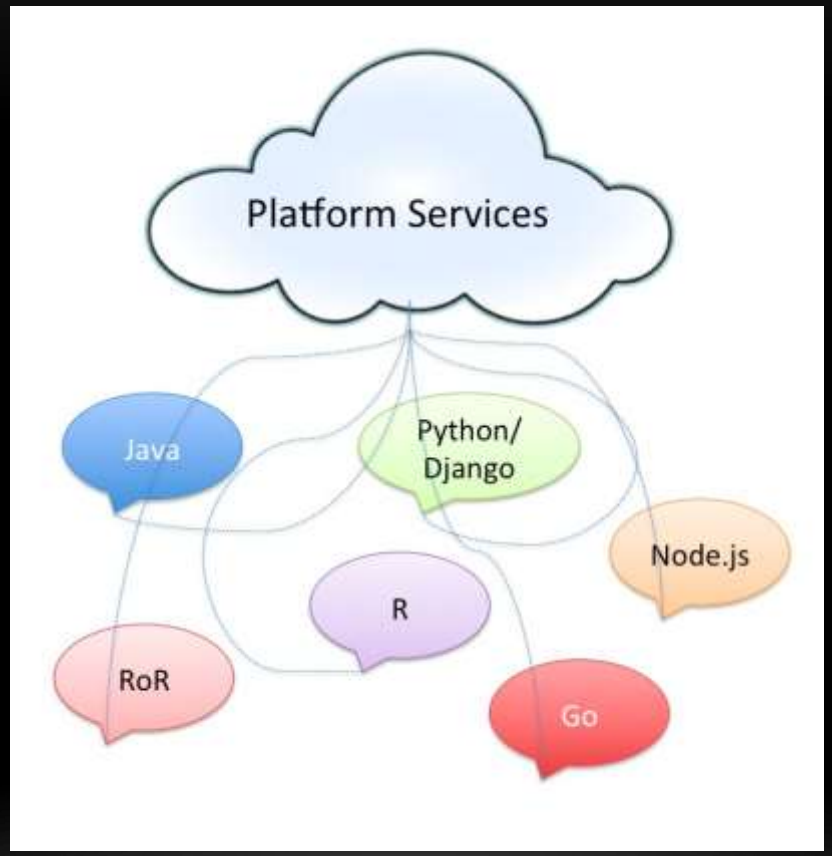


Client based Smart Loadbalancer

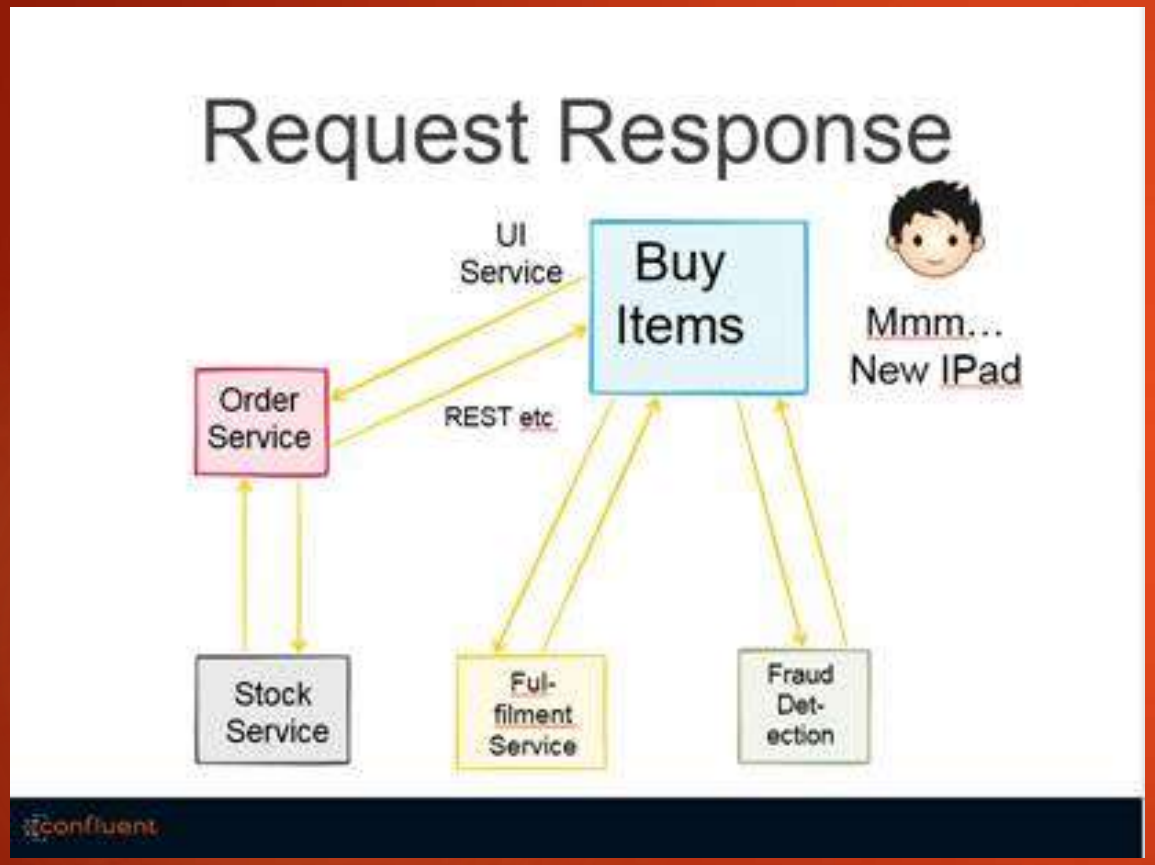


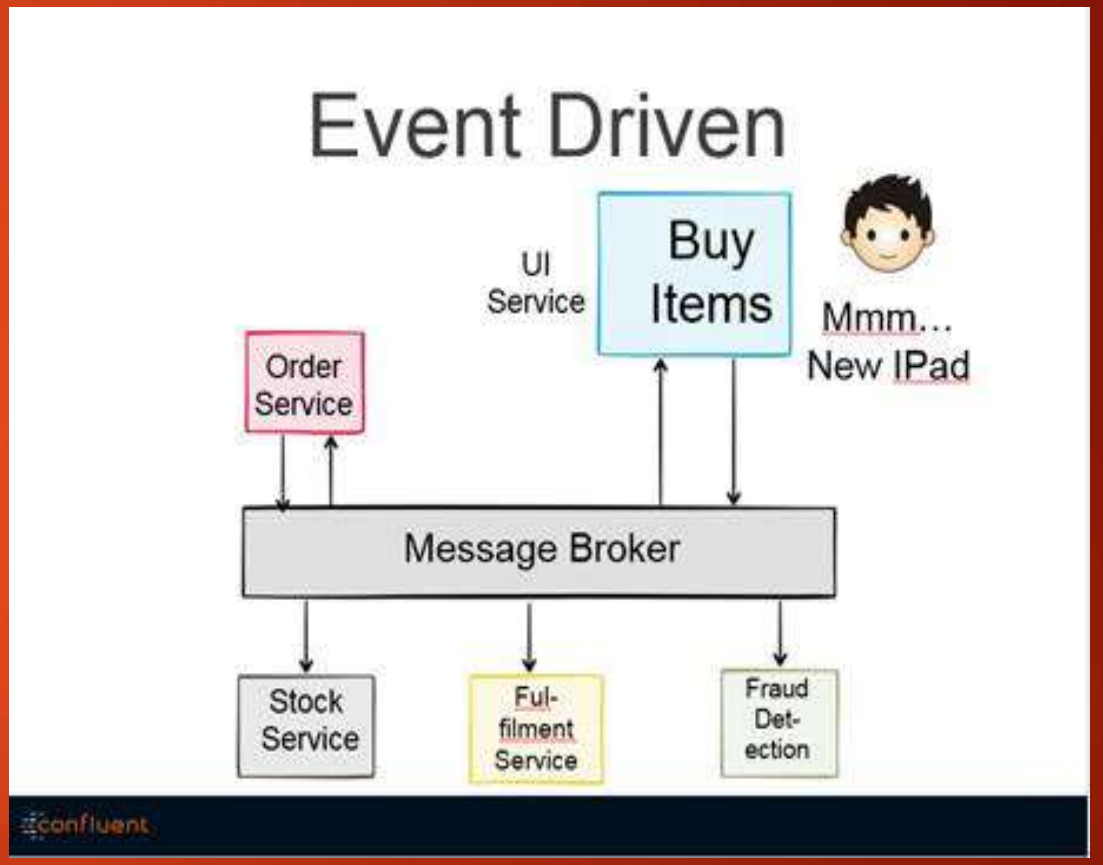
Use Ribbon (http://github.com/netflix/ribbon)

Homogeneity in A Polyglot Ecosystem

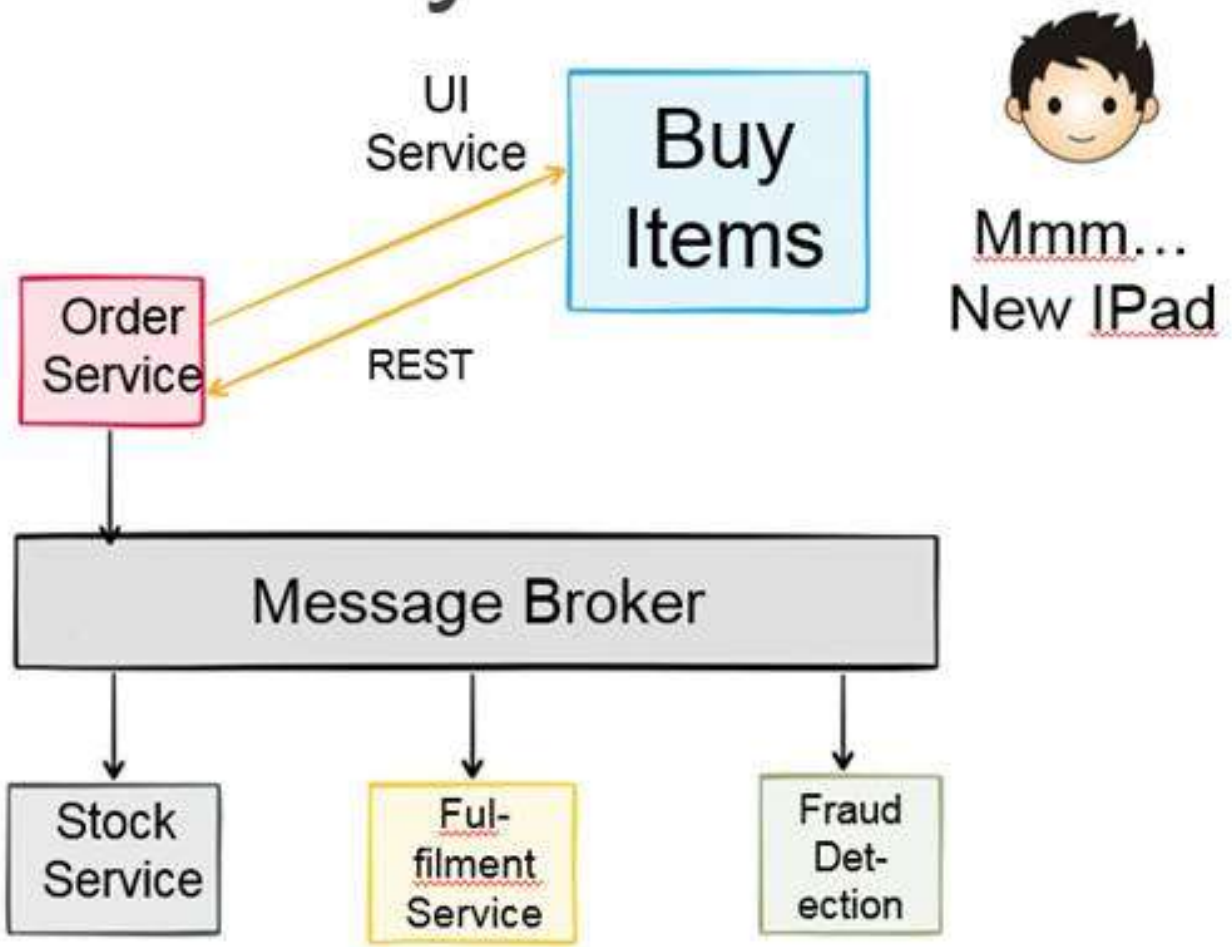


Message Queue

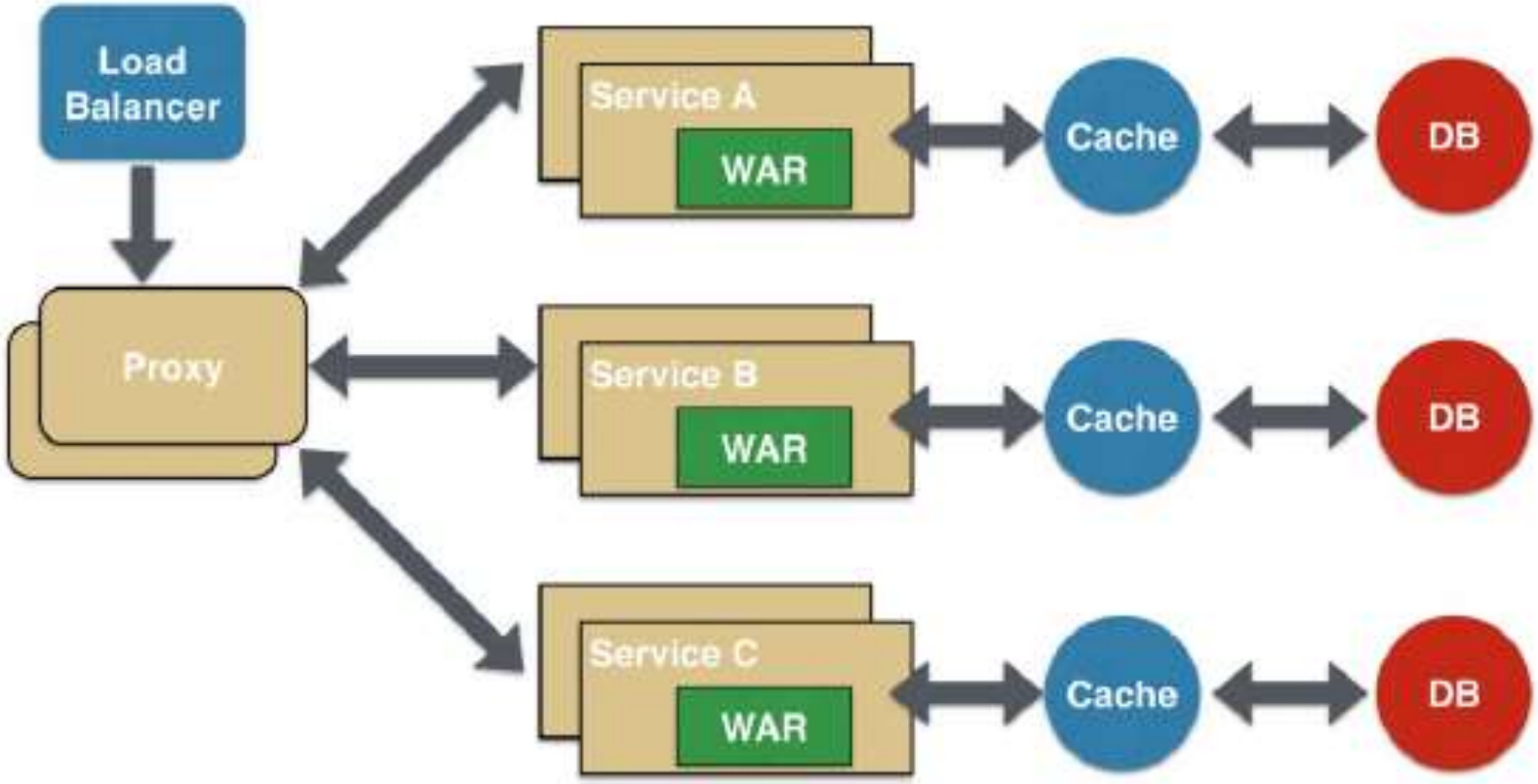




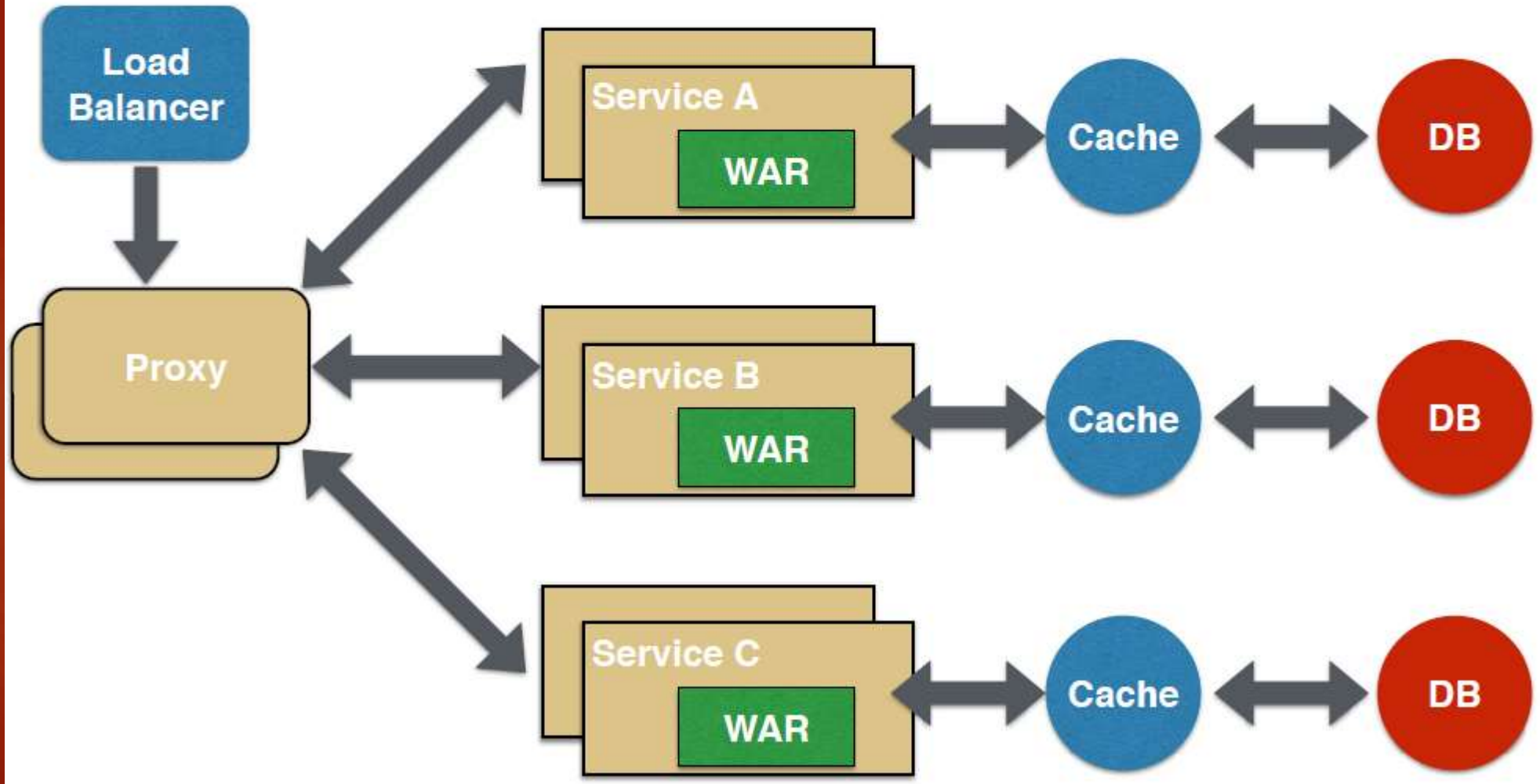
Hybrid



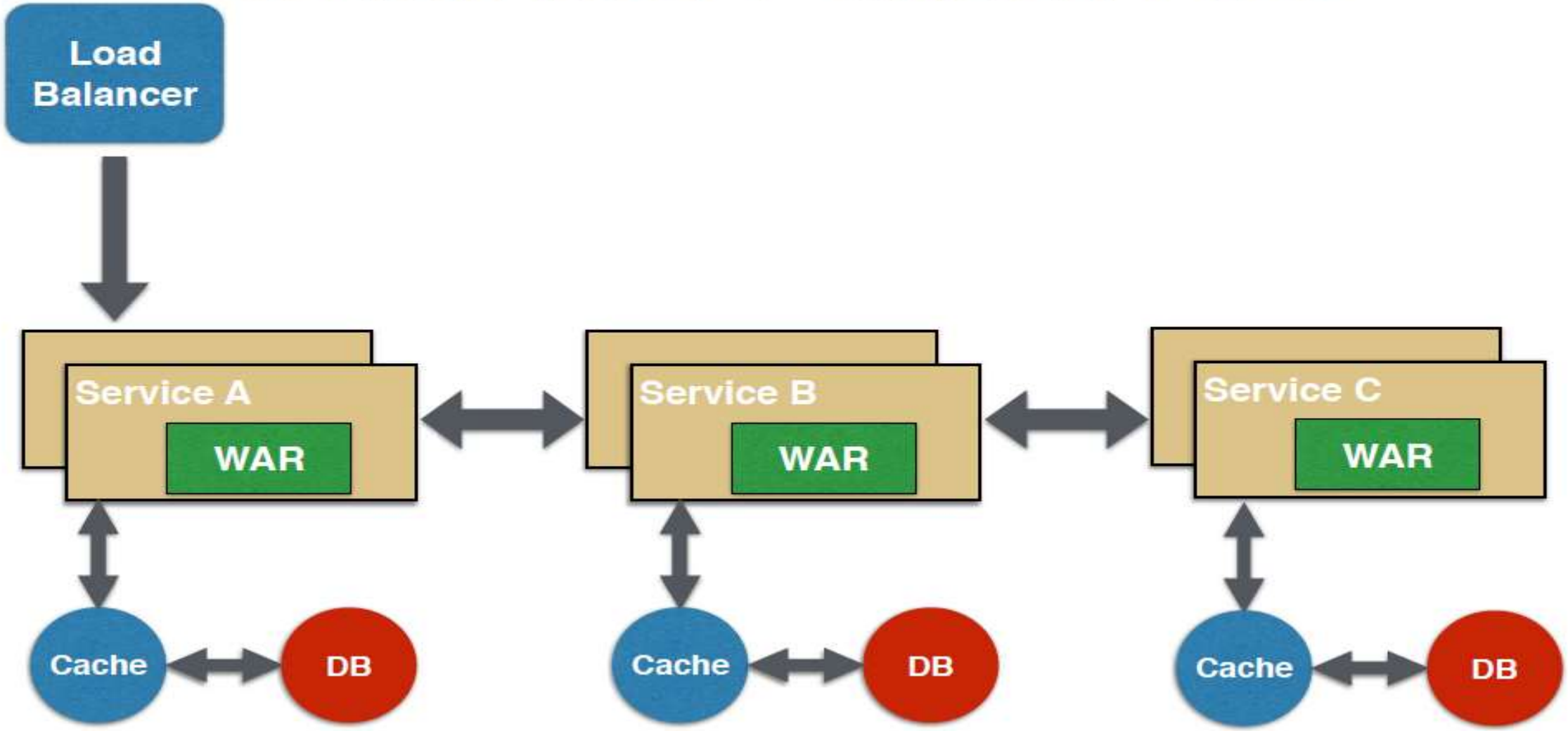
Mmm…
New IPad


Load Balancer
Proxy
Service A
WAR
Service B
WAR
Service C
WAR
Cache
Cache
Cache
DB
DB
DB

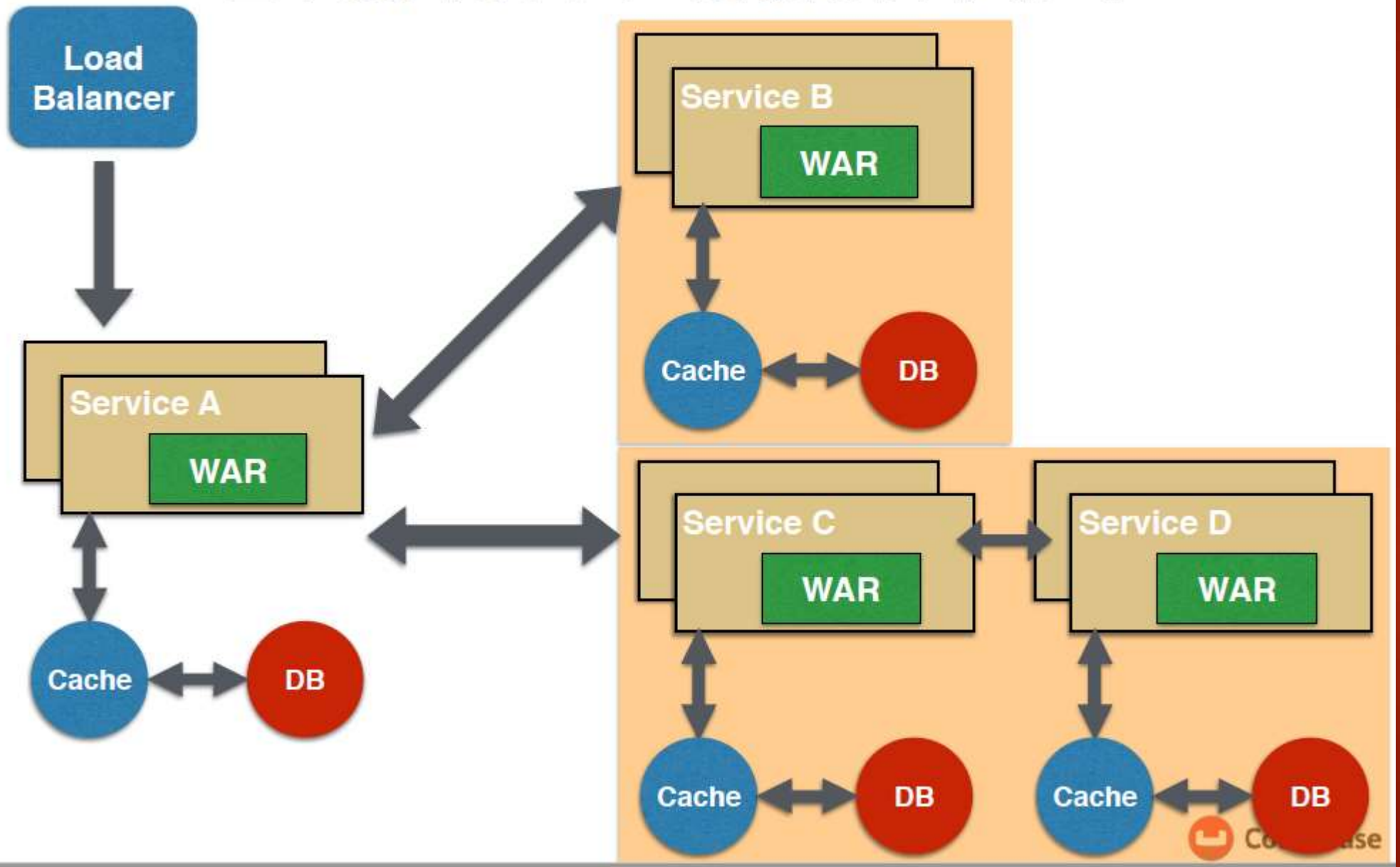
Proxy Pattern #2



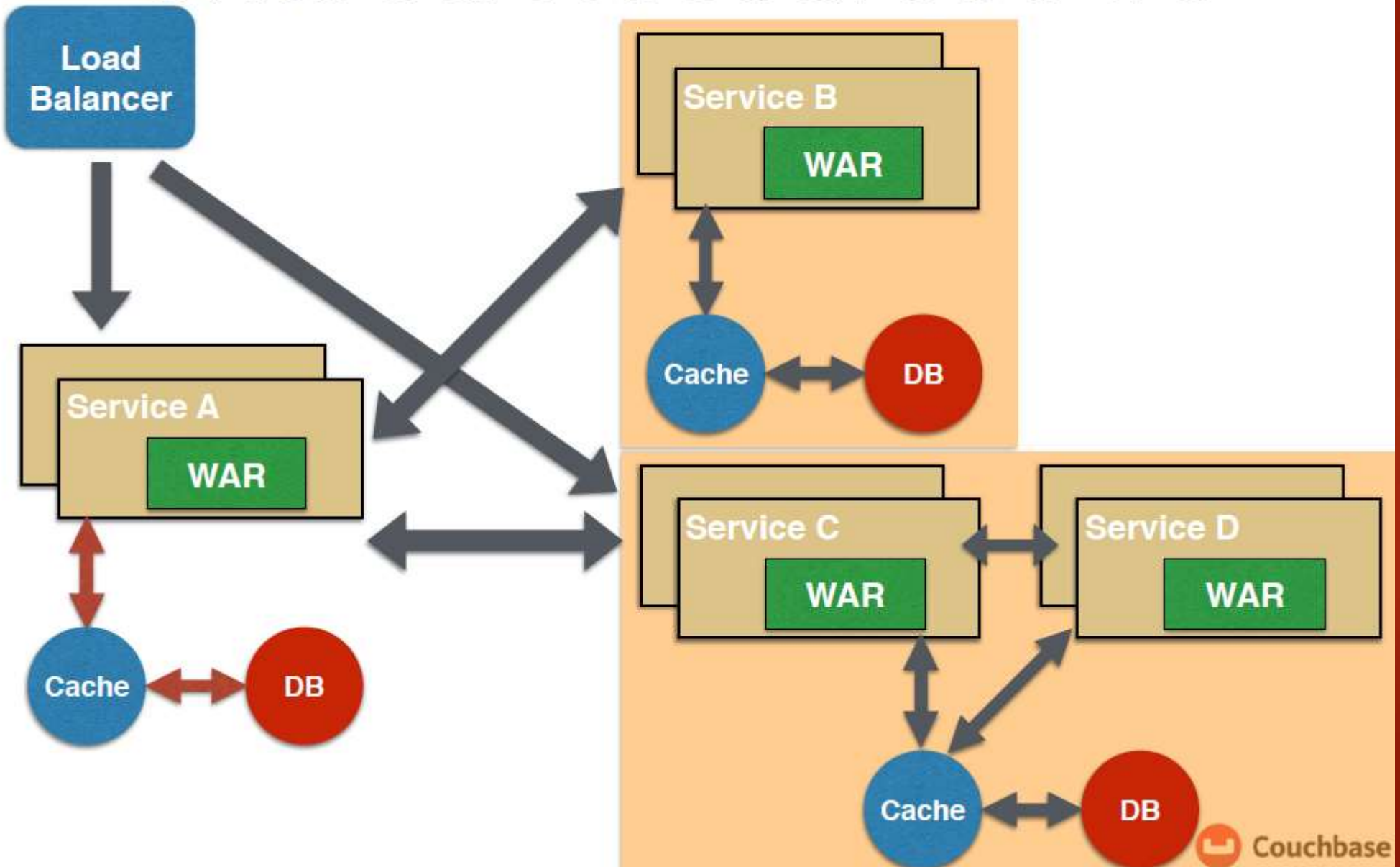
Chained Pattern #3



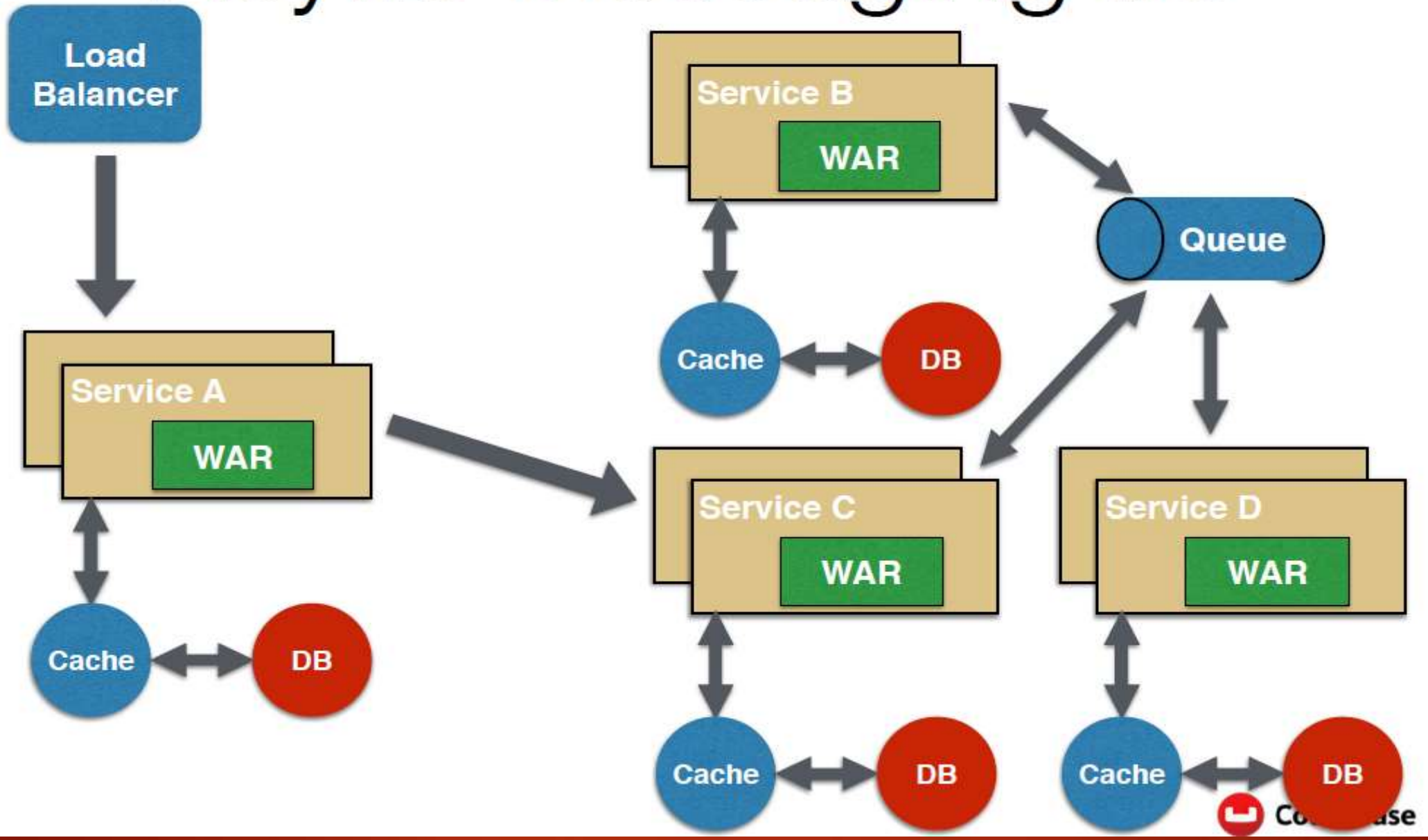
Branch Pattern #4



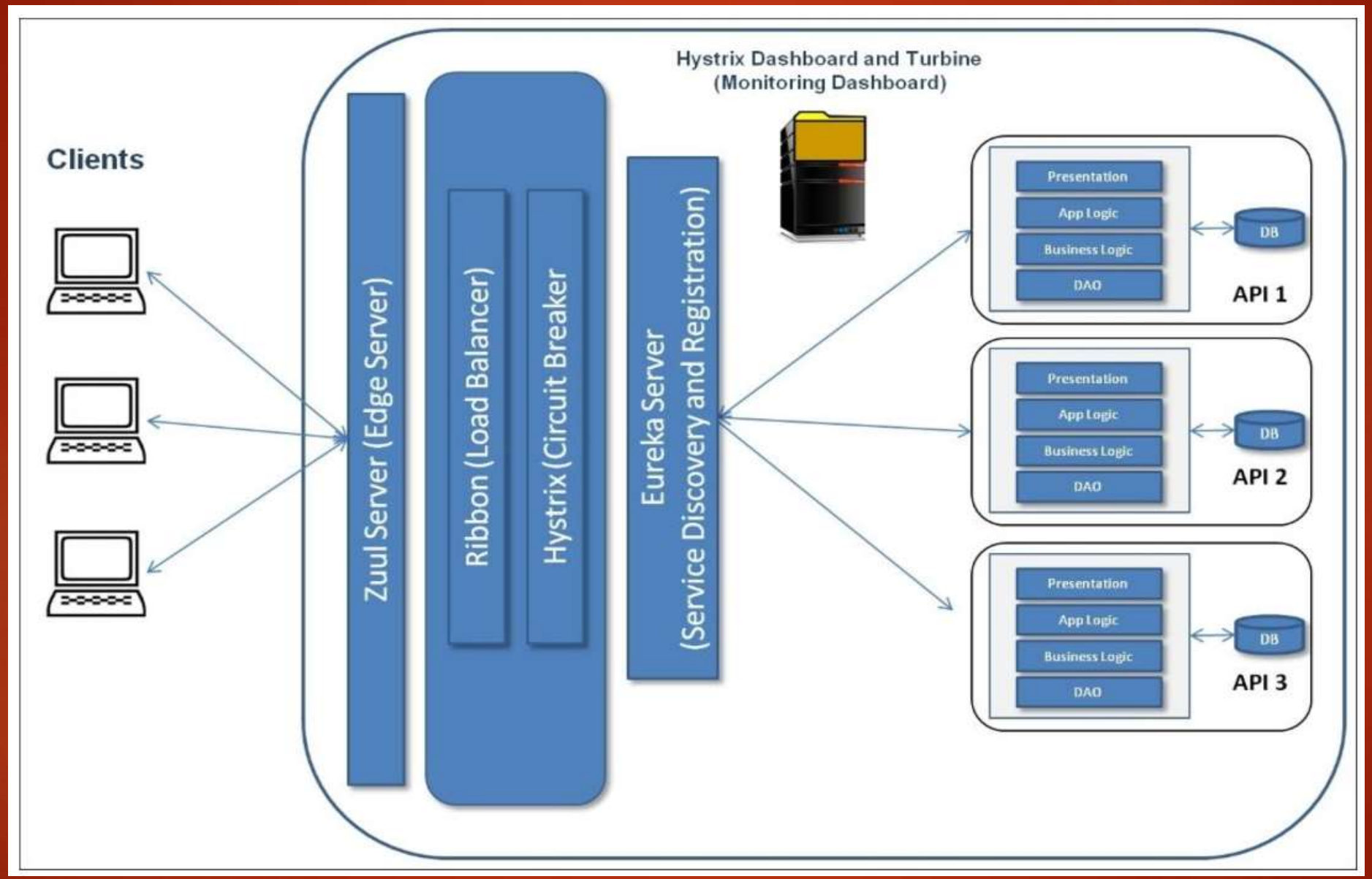
Shared Resources #5



Async Messaging #5



NetflixOSS


Hystrix Dashboard and Turbine
(Monitoring Dashboard)
Clients
Zuul Server (Edge Server)
Ribbon (Load Balancer)
Hystrix (Circuit Breaker
Eureka Server
(Service Discovery and Registration)
Presentation
App Logic
Business Logic
DAO
DB
API 1
Presentation
App Logic
Business Logic
DAO
DB
API 2
Presentation
App Logic
Business Logic
DAO
DB
API 3

localhost:8765/api/account/accounts/customer/2


```
[
  - {
        id: 2,
        customerId: 2,
        number: "222222"
    },
  - {
        id: 6,
        customerId: 2,
        number: "666666"
    },
  - {
        id: 7,
        customerId: 2,
        number: "777777"
    }
]
```


localhost:8765/api/customer/customers/1

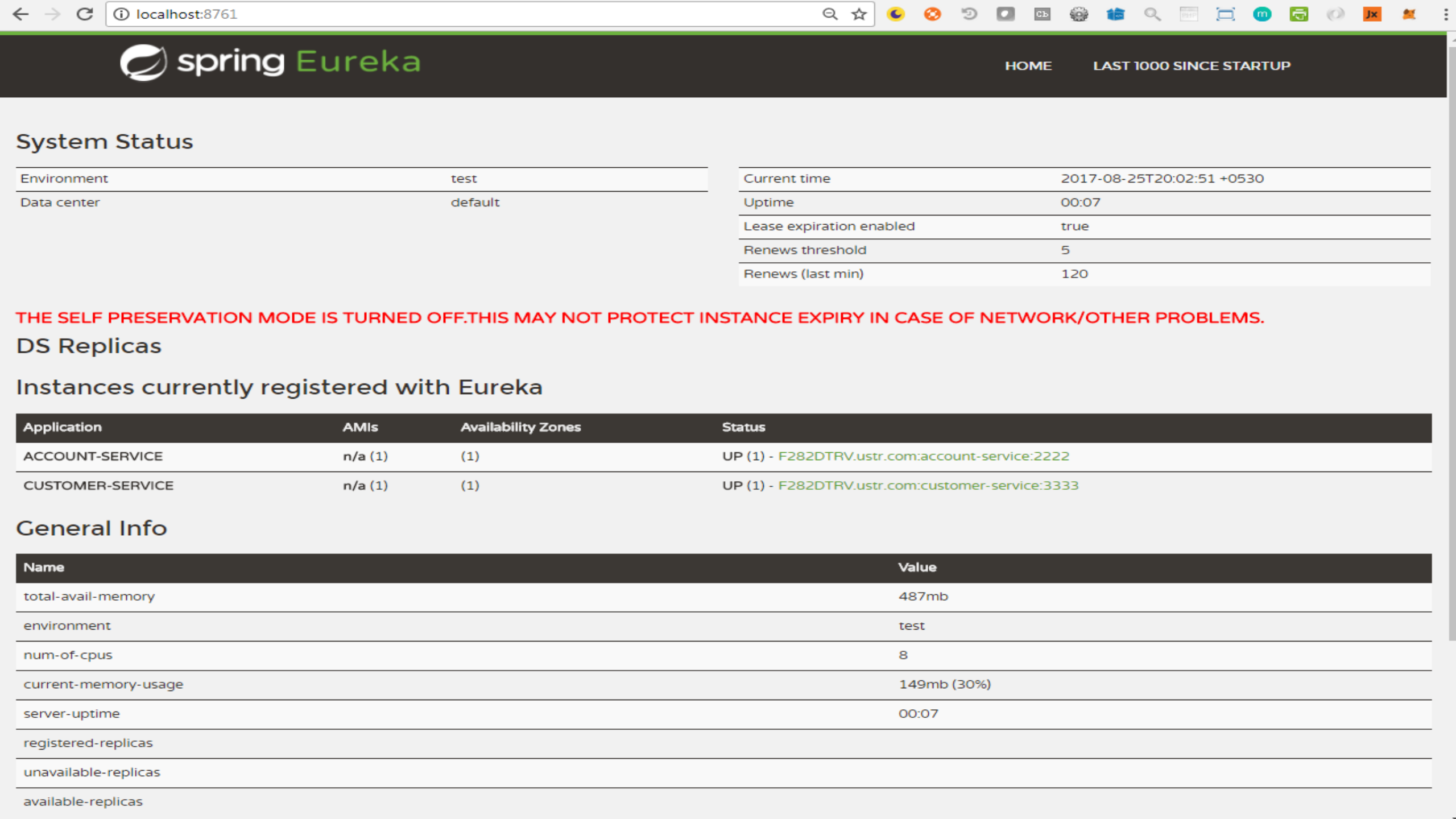

```
{
    id: 1,
    pesel: "12345",
    name: "Adam Kowalski",
    type: "INDIVIDUAL",
  - accounts: [
      - {
            id: 1,
            number: "111111"
        },
      - {
            id: 5,
            number: "555555"
        }
    ]
}
```

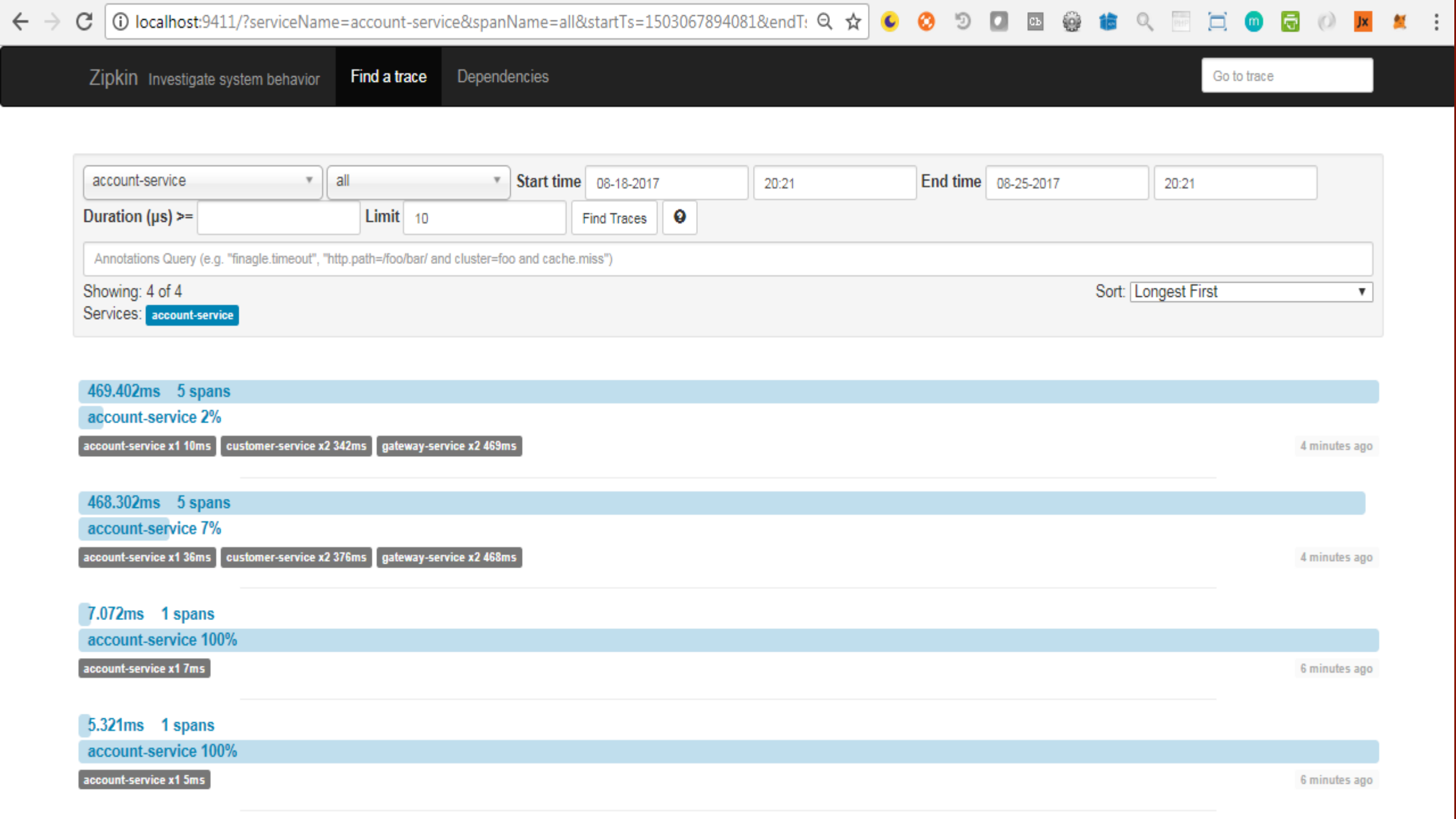

localhost:2222/accounts/

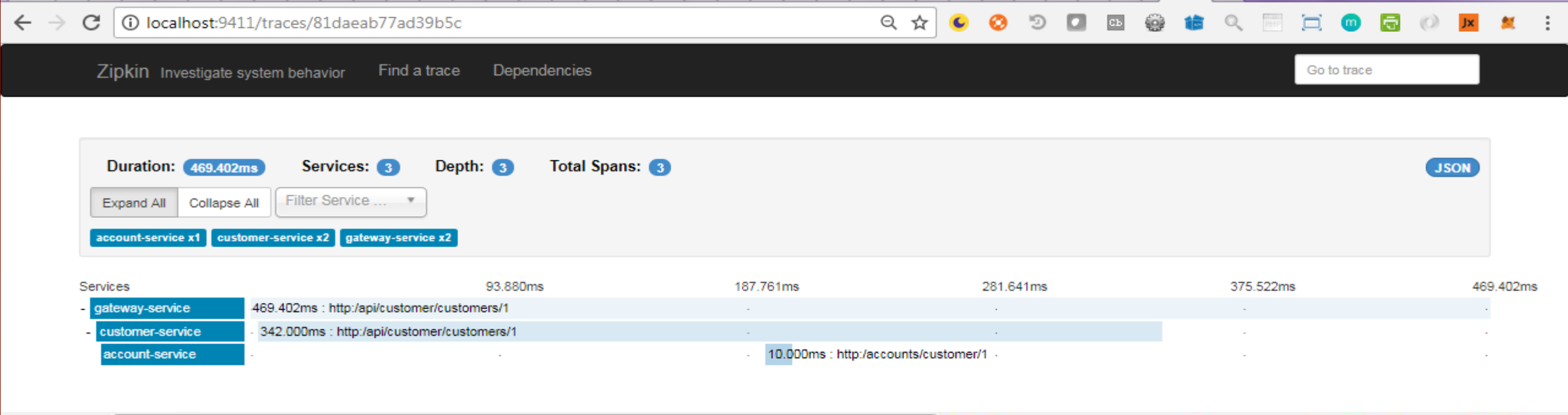

```
[
  - {
        id: 1,
        customerId: 1,
        number: "111111"
    },
  - {
        id: 2,
        customerId: 2,
        number: "222222"
    },
  - {
        id: 3,
        customerId: 3,
        number: "333333"
    },
  - {
        id: 4,
        customerId: 4,
        number: "444444"
    },
```

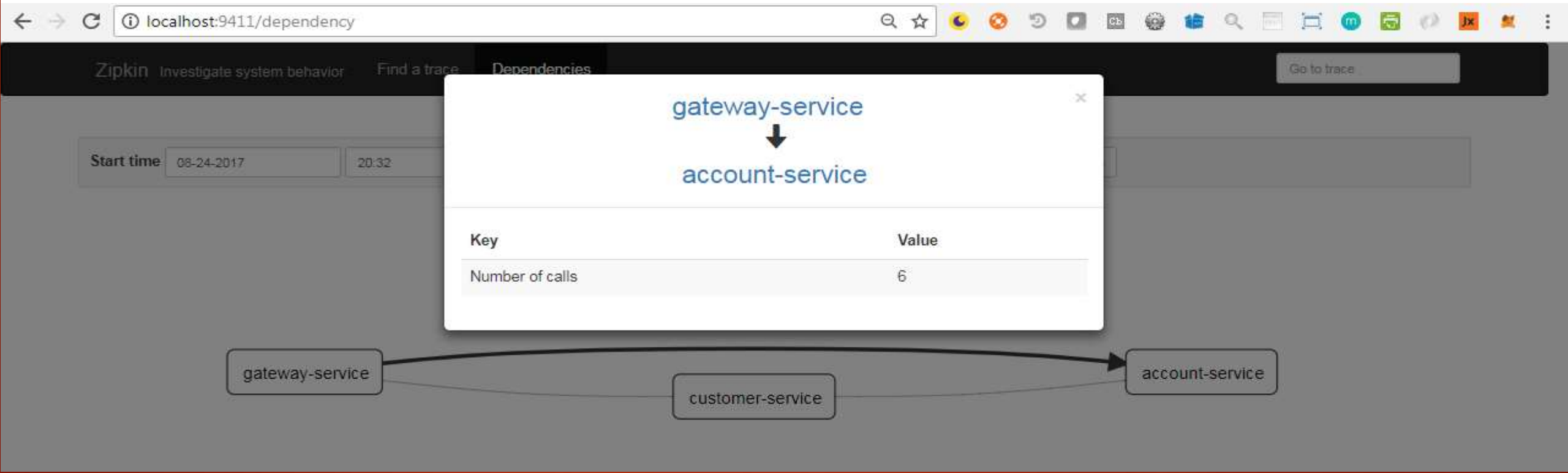

localhost:3333/customers/


```
[
  - {
        id: 1,
        pesel: "12345",
        name: "Adam Kowalski",
        type: "INDIVIDUAL",
      - accounts: [
          - {
                id: 1,
                number: "111111"
            },
          - {
                id: 5,
                number: "555555"
            }
        ]
    },
  - {
        id: 2,
        pesel: "12346",
        name: "Anna Malinowska",
```


localhost:8761
spring Eureka
HOME
LAST 1000 SINCE STARTUP
System Status
Environment test
Data center default
Current time 2017-08-25T20:02:51 +0530
Uptime 00:07
Lease expiration enabled true
Renews threshold 5
Renews (last min) 120
THE SELF PRESERVATION MODE IS TURNED OFF.THIS MAY NOT PROTECT INSTANCE EXPIRY IN CASE OF NETWORK/OTHER PROBLEMS.
DS Replicas
Instances currently registered with Eureka
Application AMIs Availability Zones Status
ACCOUNT-SERVICE n/a (1) (1) UP (1) - F282DTRV.ustr.com:account-service:2222
CUSTOMER-SERVICE n/a (1) (1) UP (1) - F282DTRV.ustr.com:customer-service:3333
General Info
Name Value
total-avail-memory 487mb
environment test
num-of-cpus 8
current-memory-usage 149mb (30%)
server-uptime 00:07
registered-replicas
unavailable-replicas
available-replicas


localhost:9411/?serviceName=account-service&spanName=all&startTs=1503067894081&endT
Zipkin
Investigate system behavior
Find a trace
Dependencies
Go to trace
account-service
all
Start time
08-18-2017
20:21
End time
08-25-2017
20:21
Duration (μs) >=
Limit
10
Find Traces
Annotations Query (e.g. "finagle.timeout", "http.path=/foo/bar/ and cluster=foo and cache.miss")
Showing: 4 of 4
Sort: Longest First
Services:
account-service
469.402ms 5 spans
account-service 2%
account-service x1 10ms
customer-service x2 342ms
gateway-service x2 469ms
4 minutes ago
468.302ms 5 spans
account-service 7%
account-service x1 36ms
customer-service x2 376ms
gateway-service x2 468ms
4 minutes ago
7.072ms 1 spans
account-service 100%
account-service x1 7ms
6 minutes ago
5.321ms 1 spans
account-service 100%
account-service x1 5ms
6 minutes ago

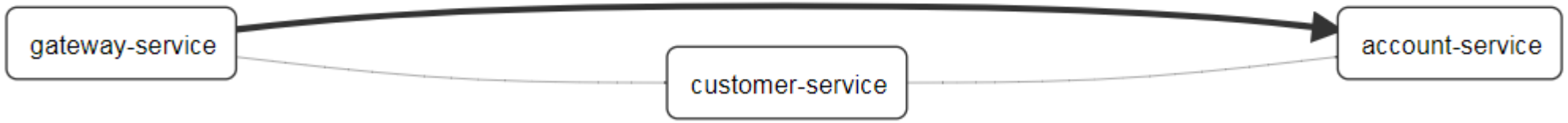

localhost:9411/traces/81daeab77ad39b5c
Zipkin
Investigate system behavior
Find a trace
Dependencies
Go to trace
Duration: 469.402ms
Services: 3
Depth: 3
Total Spans: 3
JSON
Expand All
Collapse All
Filter Service ...
account-service x1
customer-service x2
gateway-service x2
Services
93.880ms
187.761ms
281.641ms
375.522ms
469.402ms
gateway-service
469.402ms : http:/api/customer/customers/1
customer-service
342.000ms : http:/api/customer/customers/1
account-service
10.000ms : http:/accounts/customer/1


localhost:9411/dependency
Zipkin
Investigate system behavior
Find a trace
Dependencies
Go to trace
Start time
08-24-2017
20:32
gateway-service
account-service
Key
Value
Number of calls
6
gateway-service
customer-service
account-service

localhost:9411/dependency?endTs=1503672900000&startTs=1503586355010

Zipkin Investigate system behavior | Find a trace | Dependencies | Go to trace

Start time 08-24-2017 20:22 End time 08-25-2017 20:25 Analyze Dependencies



localhost:9411/dependency?endTs=1503672900000&startTs=1503586355010

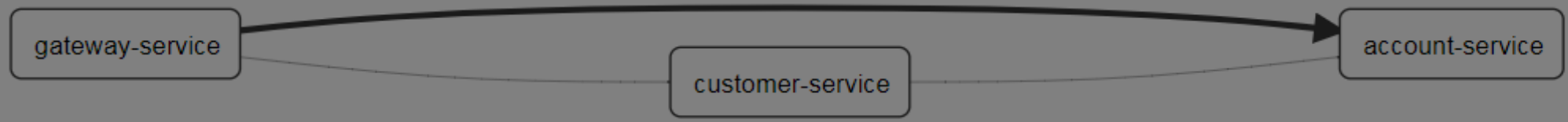
Zipkin Investigate system behavior | Find a trace | Dependencies | Go to trace

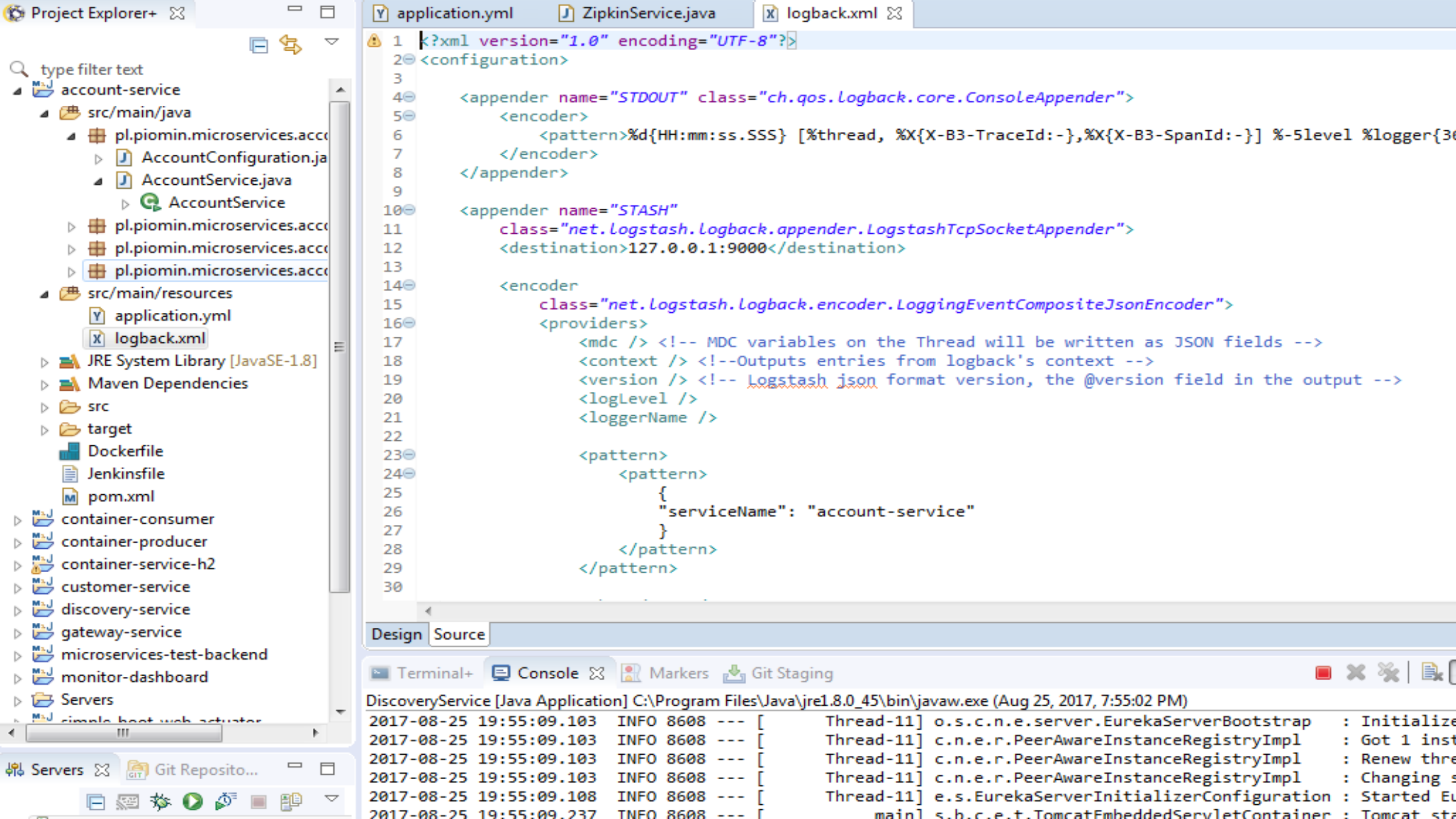
Start time 08-24-2017 20:22

customer-service

Used by
- gateway-service

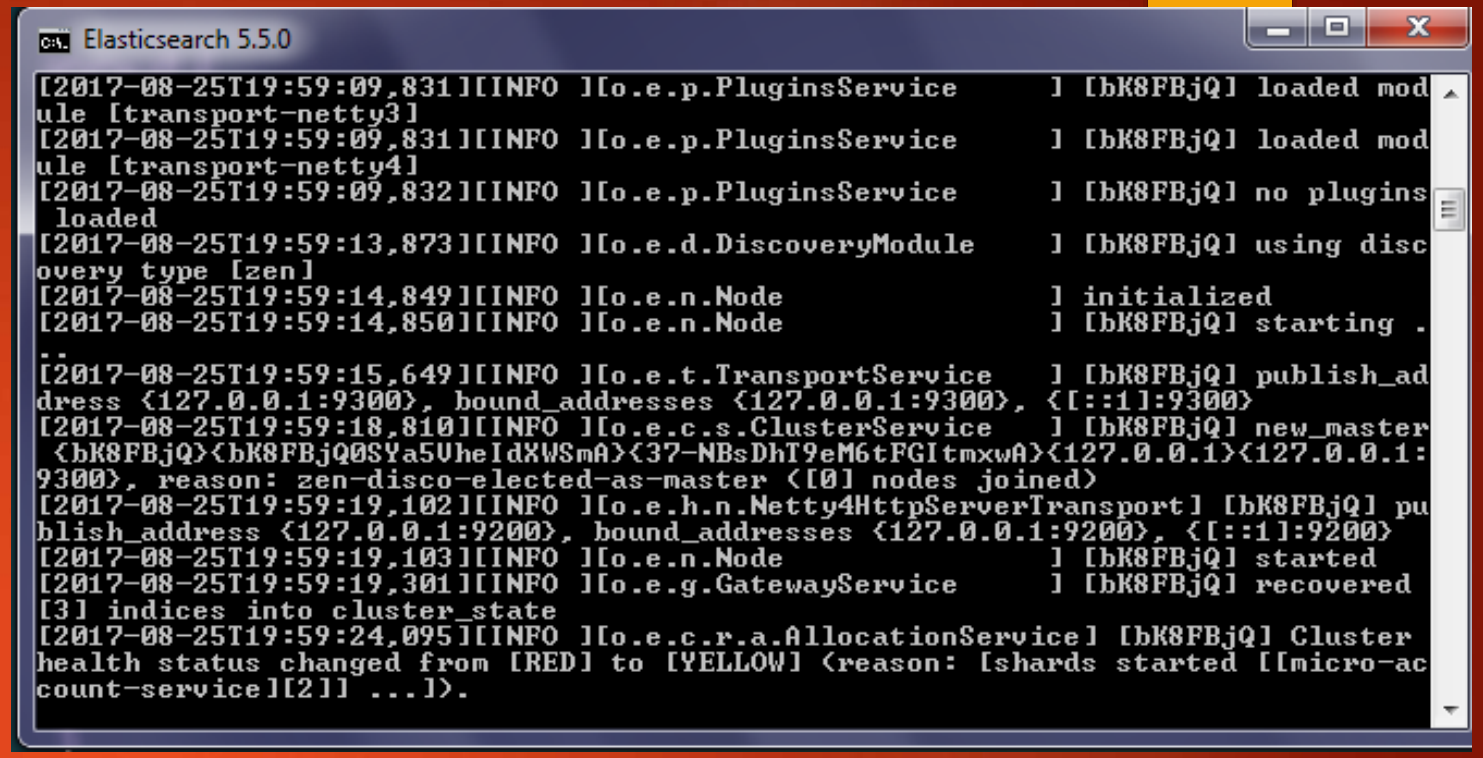
Uses
- account-service

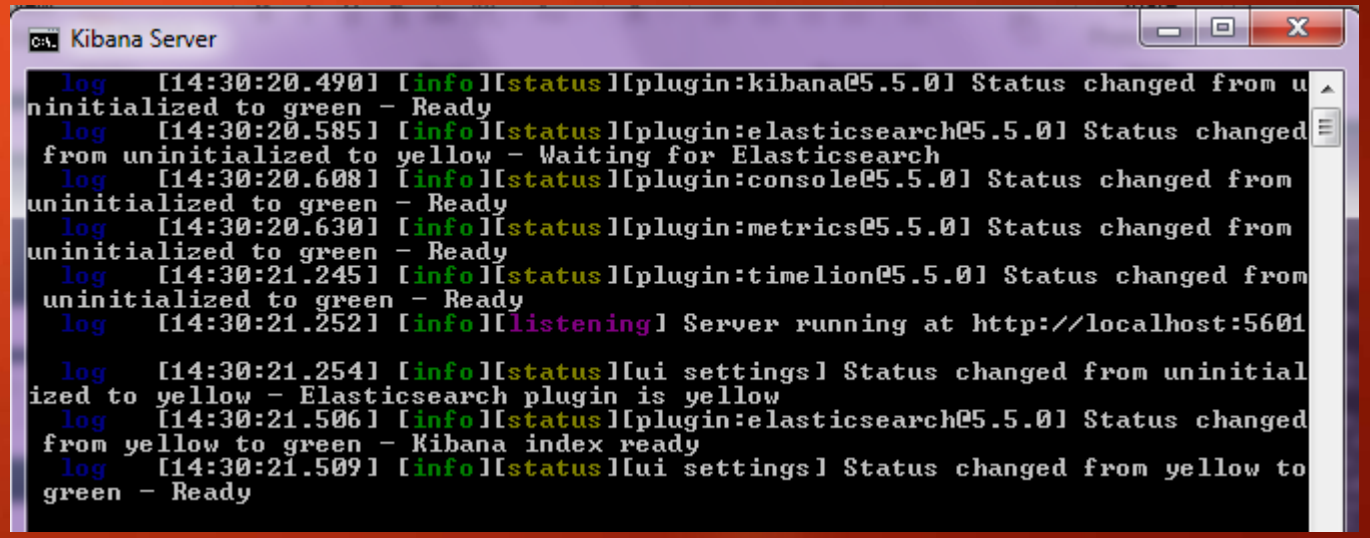



Project Explorer+
type filter text
account-service
src/main/java
pl.piomin.microservices.acc
AccountConfiguration.ja
AccountService.java
AccountService
pl.piomin.microservices.acc
pl.piomin.microservices.acc
pl.piomin.microservices.acc
src/main/resources
application.yml
logback.xml
JRE System Library [JavaSE-1.8]
Maven Dependencies
src
target
Dockerfile
Jenkinsfile
pom.xml
container-consumer
container-producer
container-service-h2
customer-service
discovery-service
gateway-service
microservices-test-backend
monitor-dashboard
Servers
Servers
Git Reposito...
application.yml
ZipkinService.java
logback.xml
1 <?xml version="1.0" encoding="UTF-8"?>
2 <configuration>
3
4 <appender name="STDOUT" class="ch.qos.logback.core.ConsoleAppender">
5 <encoder>
6 <pattern>%d{HH:mm:ss.SSS} [%thread, %X{X-B3-TraceId:-},%X{X-B3-SpanId:-}] %-5level %logger{36
7 </encoder>
8 </appender>
9
10 <appender name="STASH"
11 class="net.logstash.logback.appender.LogstashTcpSocketAppender">
12 <destination>127.0.0.1:9000</destination>
13
14 <encoder
15 class="net.logstash.logback.encoder.LoggingEventCompositeJsonEncoder">
16 <providers>
17 <mdc /> <!-- MDC variables on the Thread will be written as JSON fields -->
18 <context /> <!--Outputs entries from logback's context -->
19 <version /> <!-- Logstash json format version, the @version field in the output -->
20 <logLevel />
21 <loggerName />
22
23 <pattern>
24 <pattern>
25 {
26 "serviceName": "account-service"
27 }
28 </pattern>
29 </pattern>
30
Design
Source
Terminal+
Console
Markers
Git Staging
DiscoveryService [Java Application] C:\Program Files\Java\jre1.8.0_45\bin\javaw.exe (Aug 25, 2017, 7:55:02 PM)
2017-08-25 19:55:09.103 INFO 8608 --- [Thread-11] o.s.c.n.e.server.EurekaServerBootstrap : Initialize
2017-08-25 19:55:09.103 INFO 8608 --- [Thread-11] c.n.e.r.PeerAwareInstanceRegistryImpl : Got 1 inst
2017-08-25 19:55:09.103 INFO 8608 --- [Thread-11] c.n.e.r.PeerAwareInstanceRegistryImpl : Renew thre
2017-08-25 19:55:09.103 INFO 8608 --- [Thread-11] c.n.e.r.PeerAwareInstanceRegistryImpl : Changing s
2017-08-25 19:55:09.108 INFO 8608 --- [Thread-11] e.s.EurekaServerInitializerConfiguration : Started Eu
2017-08-25 19:55:09.237 INFO 8608 --- [main] s.b.c.e.t.TomcatEmbeddedServletContainer : Tomcat sta

logstash-simple.conf

```
input {
 tcp {
 port => 9000 codec => "json"
 }
 }
output {
 elasticsearch {
 hosts => ["localhost:9200"] index =>
"micro-%{serviceName}"
 }
 }
```





micro-*

Selected Fields

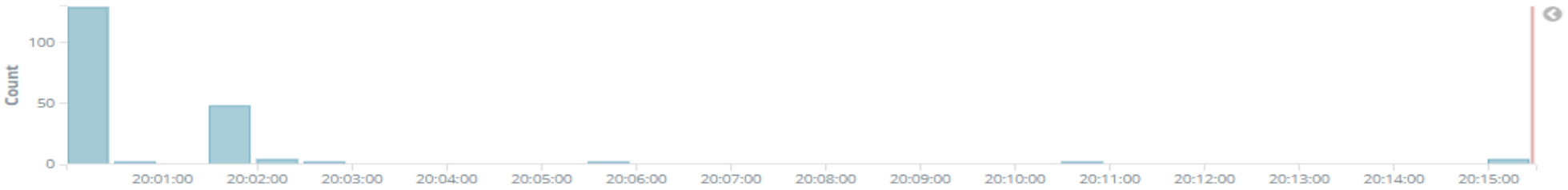
- t logger_name
- t message
- t serviceName

Available Fields

Popular

- @timestamp
- t X-B3-ParentSpanId
- t _id
- t _index
- # _score
- t _type
- t host
- t level
- # port
- t thread_name
- # @version
- t X-B3-SpanId
- t X-B3-TraceId
- t X-Span-Export
- ? stack_trace

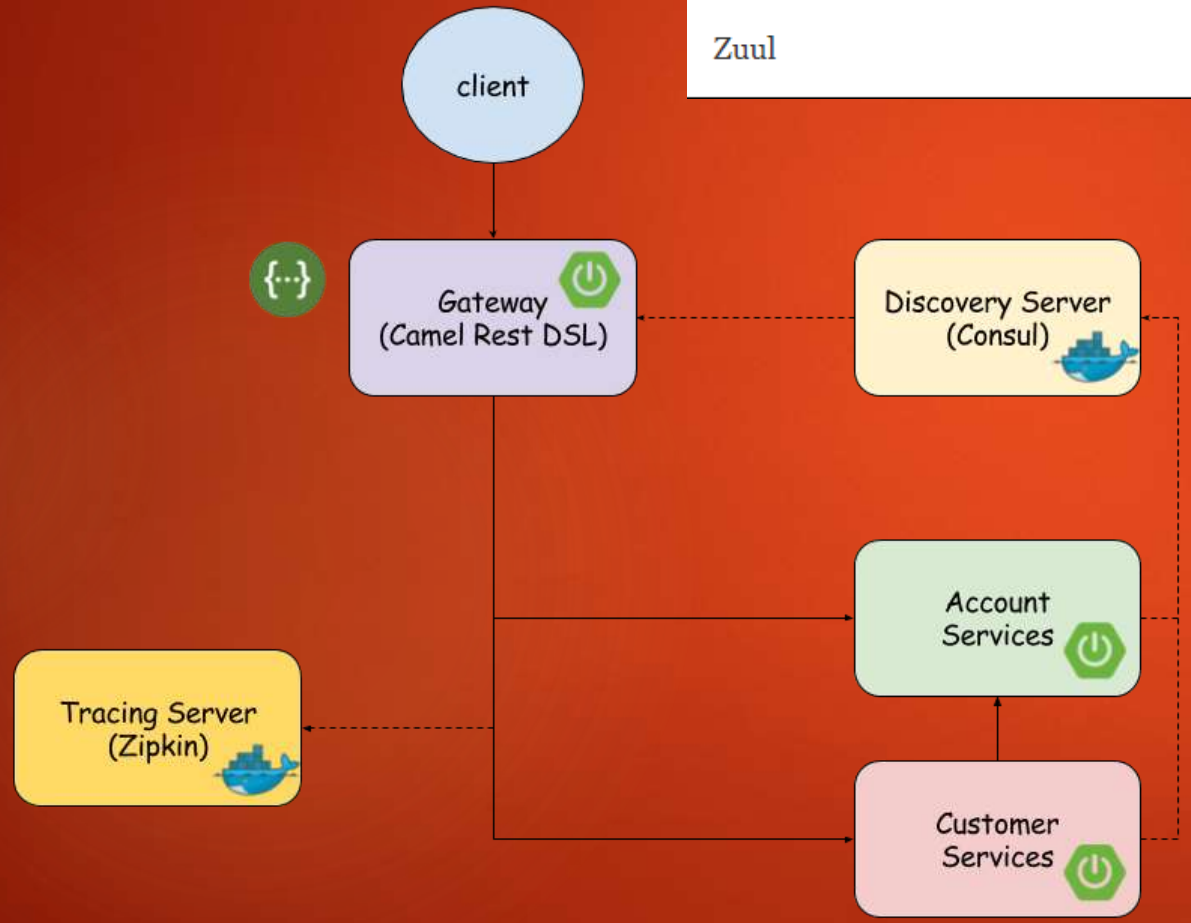
August 25th 2017, 20:00:26.646 - August 25th 2017, 20:15:26.646 — Auto



@timestamp per 30 seconds

Time	serviceName	logger_name	message
August 25th 2017, 20:15:19.489	account-service	pl.piomin.microservices.account.api.Api	Account.findAll: [Account [id=1, number='111111', customerId=1], Account [id=2, number='222222', customerId=2], Account [id=3, number='333333', customerId=3], Account [id=4, number='444444', customerId=4], Account [id=5, number='555555', customerId=1], Account [id=6, number='666666', customerId=2], Account [id=7, number='777777', customerId=2]]
August 25th 2017, 20:15:19.489	account-service	pl.piomin.microservices.account.api.Api	Account.findAll()
August 25th 2017, 20:15:19.238	account-service	pl.piomin.microservices.account.api.Api	Account.findAll()
August 25th 2017, 20:15:19.238	account-service	pl.piomin.microservices.account.api.Api	Account.findAll: [Account [id=1, number='111111', customerId=1], Account [id=2, number='222222', customerId=2], Account [id=3, number='333333', customerId=3], Account [id=4, number='444444', customerId=4], Account [id=5, number='555555', customerId=1], Account [id=6, number='666666', customerId=2], Account [id=7, number='777777', customerId=2]]
August 25th 2017, 20:10:49.859	customer-service	com.netflix.discovery.shared.resolver.aws.ConfigClusterResolver	Resolving eureka endpoints via configuration
August 25th 2017, 20:10:35.477	account-service	com.netflix.discovery.shared.resolver.aws.ConfigClusterResolver	Resolving eureka endpoints via configuration
August 25th 2017, 20:05:49.807	customer-	com.netflix.dis	Resolving eureka endpoints via configuration

Netflix Component Name	Functionality
Eureka	Service Registration and Discovery
Ribbon	Dynamic Routing and Load Balancer
Hystrix	Circuit Breaker
Zuul	Edge Server




Characteristics
php
ORACLE
DATABASE
MySQL
Java
ceylon
mongoDB
PostgreSQL
Couchbase
Scala
JS
redis
cassandra
Groovy
JRuby
WildFly
Apache TomEE

UST Global®
Innovation • Information • Technology

Thank You