

Spring Cloud

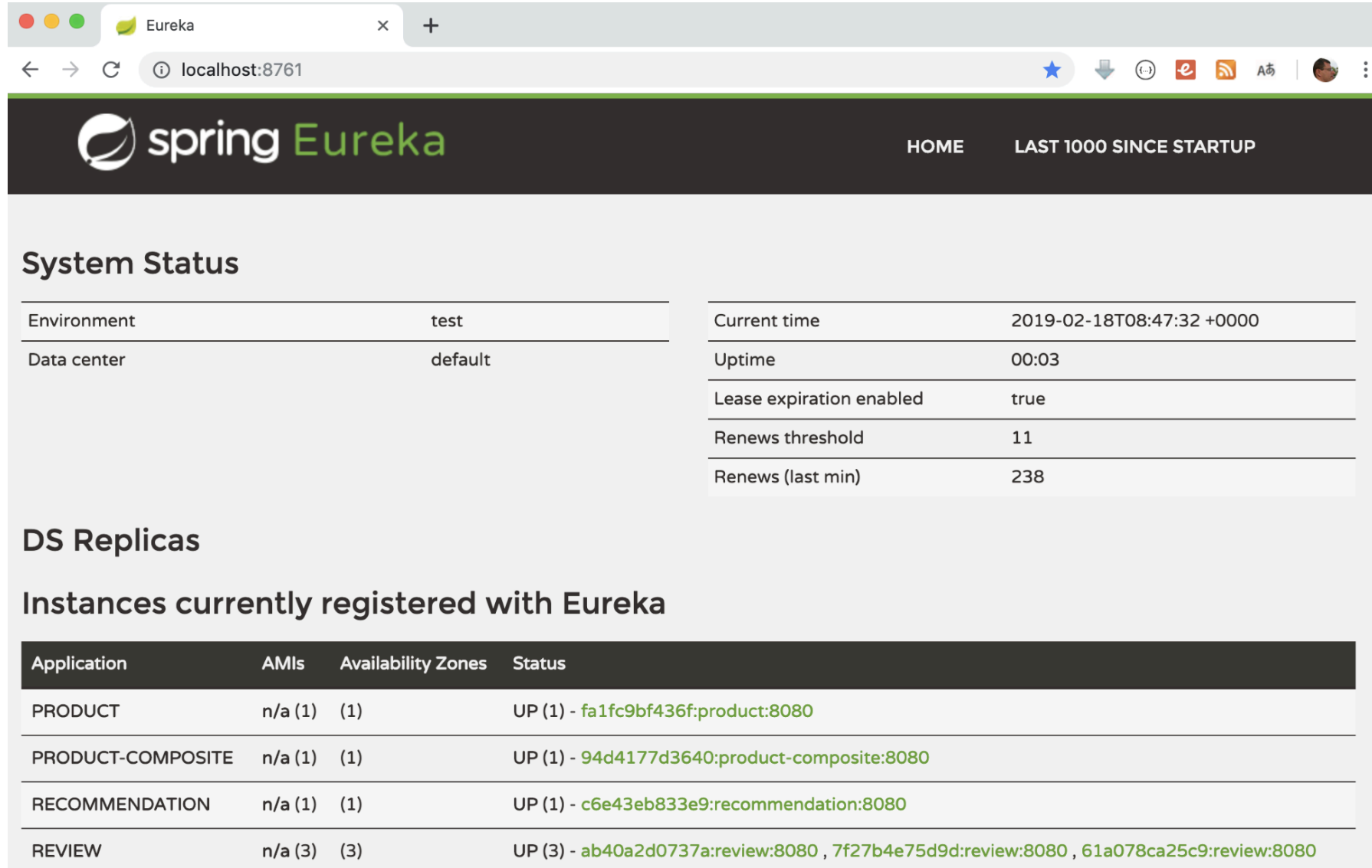
Parte Final

Spring Cloud Greenwich (V2.1) en Enero 2019

Current component	Replaced by
Netflix Hystrix	Resilience4j
Netflix Hystrix Dashboard/Netflix Turbine	Micrometer and monitoring system
Netflix Ribbon	Spring Cloud load balancer
Netflix Zuul	Spring Cloud Gateway

Design pattern	Software component
Service discovery	Netflix Eureka and Spring Cloud load balancer
Edge server	Spring Cloud Gateway and Spring Security OAuth
Centralized configuration	Spring Cloud Configuration Server
Circuit breaker	Resilience4j
Distributed tracing	Spring Cloud Sleuth and Zipkin

Netflix Eureka como discovery service



The screenshot shows the Netflix Eureka web interface in a browser window. The browser's address bar shows 'localhost:8761'. The page has a dark header with the 'spring Eureka' logo and navigation links for 'HOME' and 'LAST 1000 SINCE STARTUP'. The main content area is divided into sections: 'System Status', 'DS Replicas', and 'Instances currently registered with Eureka'. The 'System Status' section contains two tables. The first table shows 'Environment' as 'test' and 'Data center' as 'default'. The second table shows 'Current time' as '2019-02-18T08:47:32 +0000', 'Uptime' as '00:03', 'Lease expiration enabled' as 'true', 'Renews threshold' as '11', and 'Renews (last min)' as '238'. The 'DS Replicas' section is currently empty. The 'Instances currently registered with Eureka' section contains a table with four columns: 'Application', 'AMIs', 'Availability Zones', and 'Status'. It lists four applications: 'PRODUCT', 'PRODUCT-COMPOSITE', 'RECOMMENDATION', and 'REVIEW', each with its respective AMIs, availability zones, and status.

System Status

Environment	test
Data center	default

Current time	2019-02-18T08:47:32 +0000
Uptime	00:03
Lease expiration enabled	true
Renews threshold	11
Renews (last min)	238

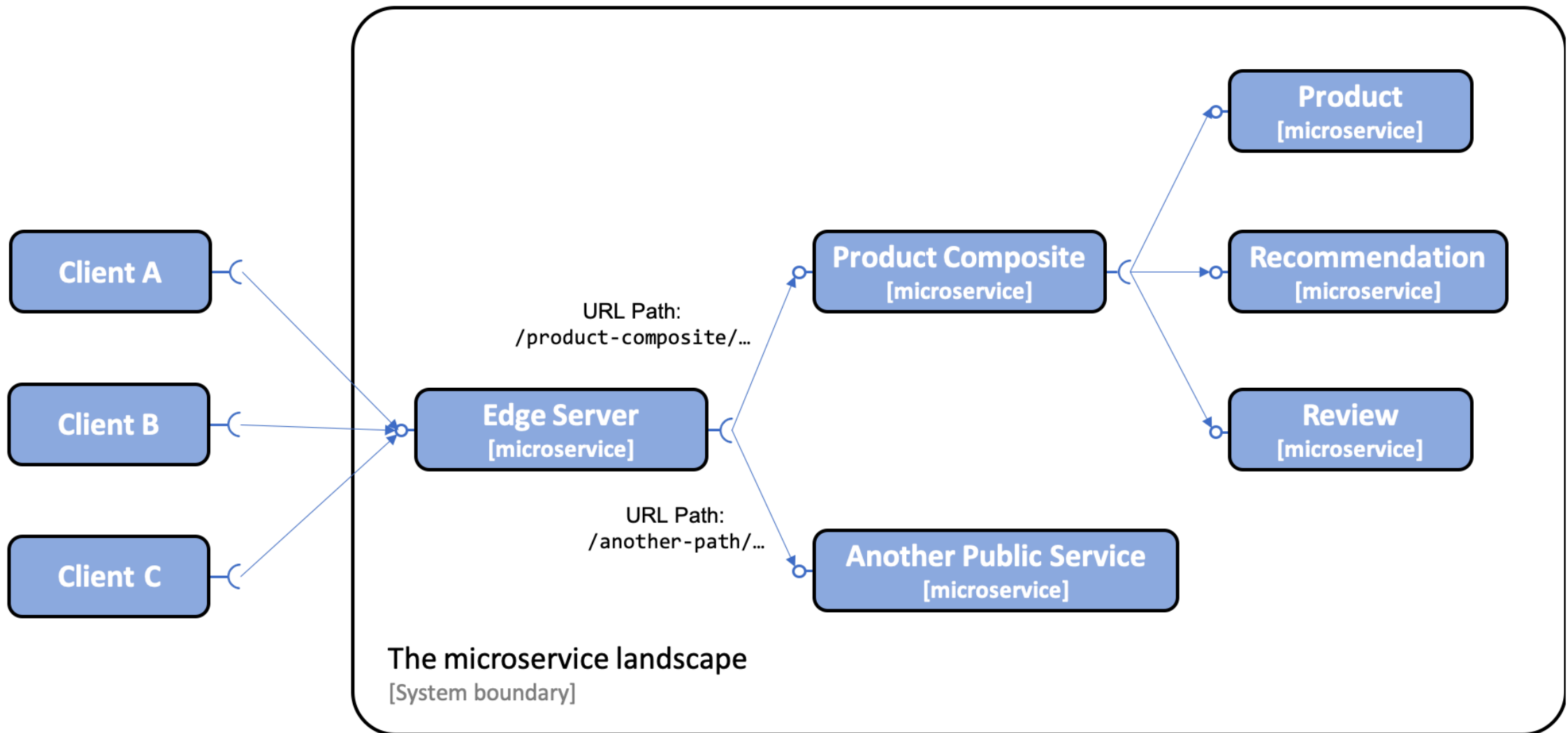
DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
PRODUCT	n/a (1)	(1)	UP (1) - fa1fc9bf436f:product:8080
PRODUCT-COMPOSITE	n/a (1)	(1)	UP (1) - 94d4177d3640:product-composite:8080
RECOMMENDATION	n/a (1)	(1)	UP (1) - c6e43eb833e9:recommendation:8080
REVIEW	n/a (3)	(3)	UP (3) - ab40a2d0737a:review:8080 , 7f27b4e75d9d:review:8080 , 61a078ca25c9:review:8080

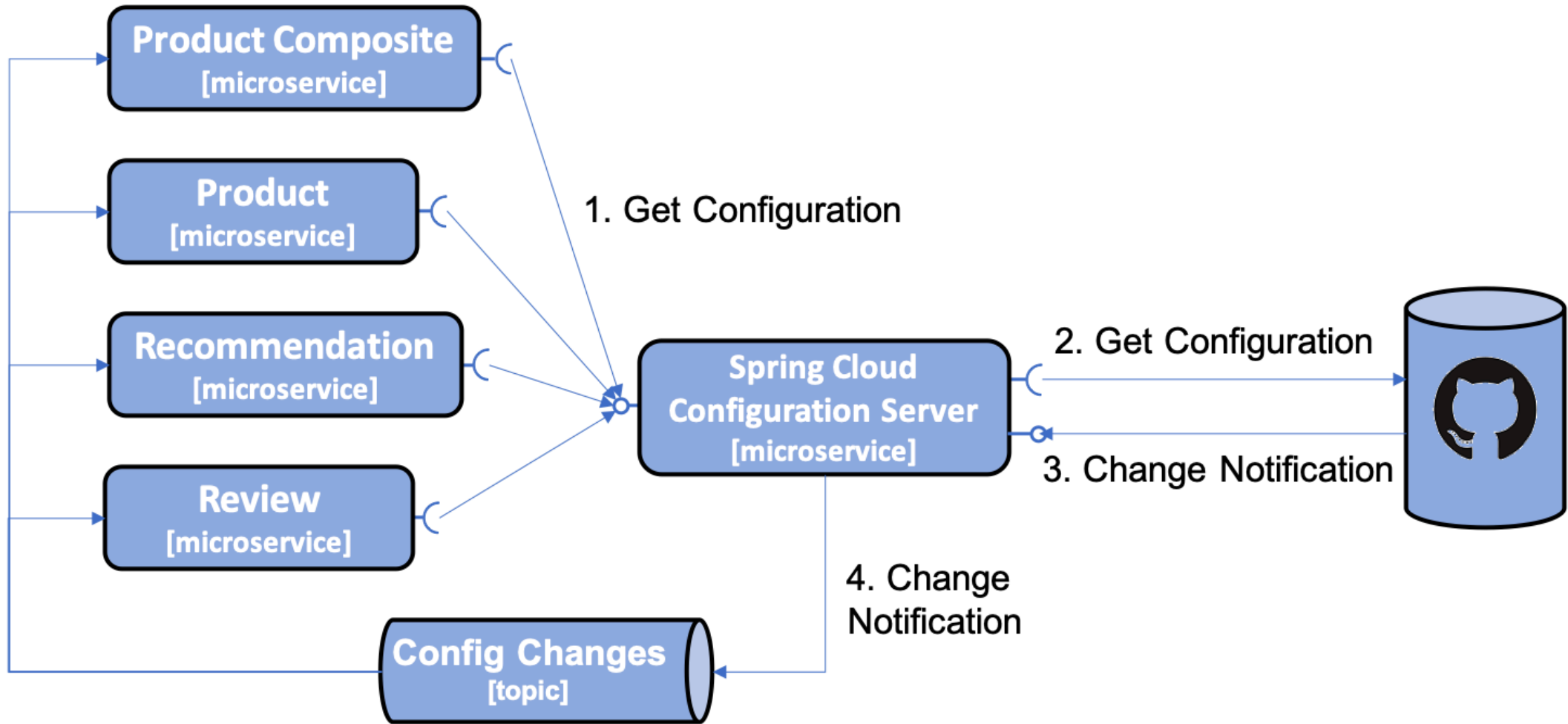
Spring Cloud Gateway como edge server

- Spring Cloud utilizó Netflix Zuul v1 como su servidor perimetral. Desde el lanzamiento de Spring Cloud Greenwich, se recomienda usar Spring Cloud Gateway en su lugar. Spring Cloud Gateway viene con un soporte similar para funciones críticas, como el enrutamiento basado en la ruta URL y la protección de puntos finales mediante el uso de OAuth 2.0 y OpenID Connect (OIDC).
- Una diferencia importante entre Netflix Zuul v1 y Spring Cloud Gateway es que Spring Cloud Gateway se basa en API sin bloqueo que utilizan Spring 5, Project Reactor y Spring Boot 2, mientras que Netflix Zuul v1 se basa en API de bloqueo. Esto significa que Spring Cloud Gateway debería poder manejar mayores cantidades de solicitudes concurrentes que Netflix Zuul v1, lo cual es importante para un servidor gateway por el que pasa todo el tráfico externo.



Spring Cloud Config para configuración centralizada

- Spring Cloud Config admite el almacenamiento de archivos de configuración de varios backends diferentes, como los siguientes:
 - Un repositorio de Git, por ejemplo, en GitHub o Bitbucket
 - Sistema de archivos local
 - HashiCorp Vault
 - Una base de datos JDBC



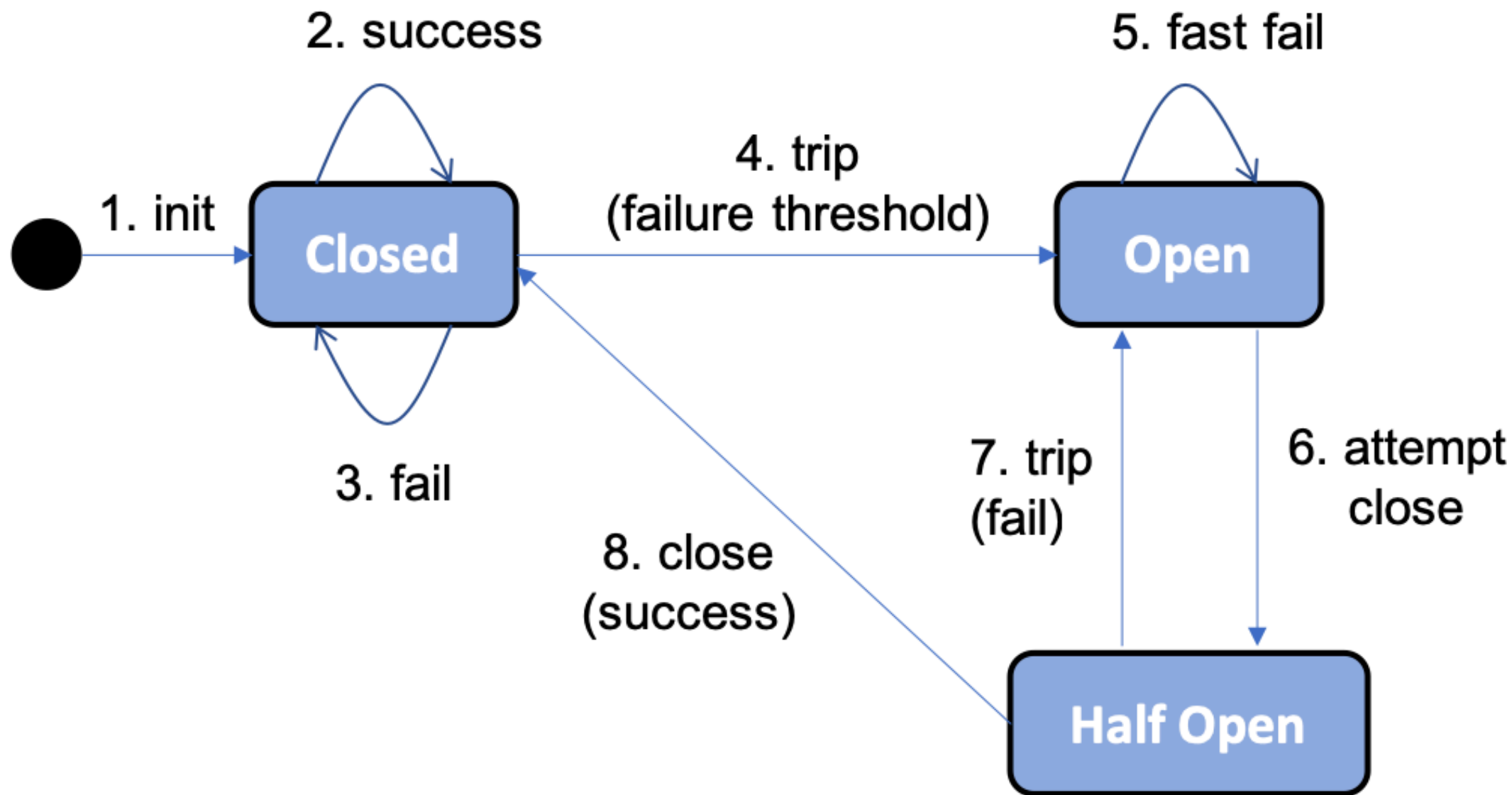
The microservice landscape
[System boundary]

Using Resilience4j para una mejor resiliencia

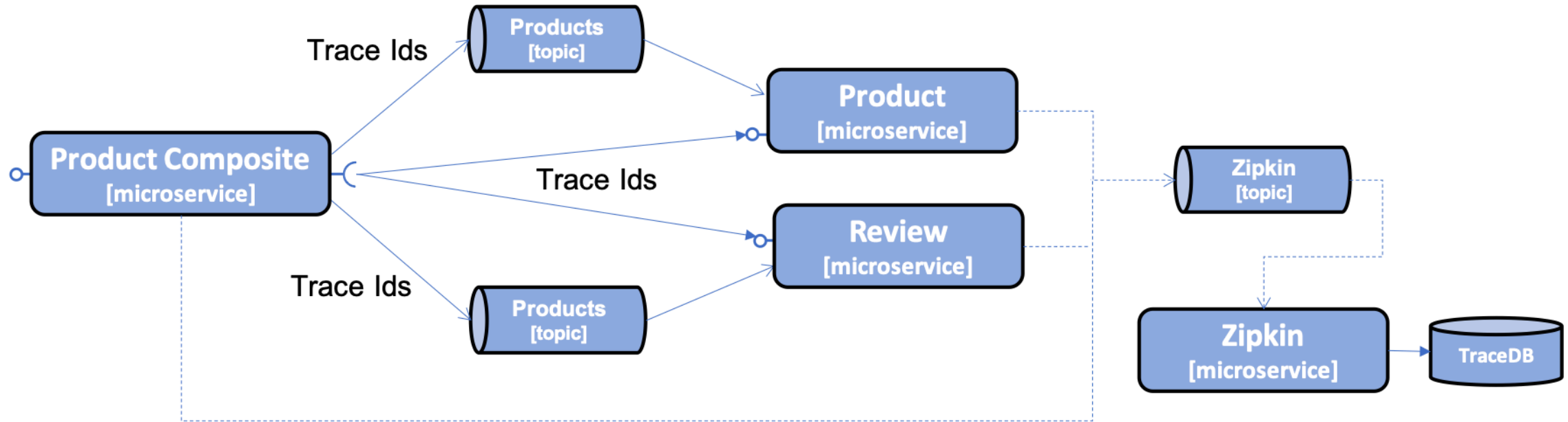
- Inicialmente, Spring Cloud vino con Netflix Hystrix, un interruptor de circuito bien probado. Pero desde el lanzamiento de Spring Cloud Greenwich, se recomienda reemplazar Netflix Hystrix con **Resilience4j**. La razón de esto es que Netflix recientemente puso a Hystrix en modo de mantenimiento. Para obtener más detalles, consulte <https://github.com/Netflix/Hystrix#hystrix-status>.
- Resilience4j es una biblioteca de tolerancia a errores basada en código abierto. Puede descubrir más al respecto en <https://github.com/resilience4j/resilience4j>. Viene con los siguientes mecanismos de tolerancia a fallas incorporados:

Circuit breaker is used to prevent a chain of failure reactions if a remote service stops to respond.

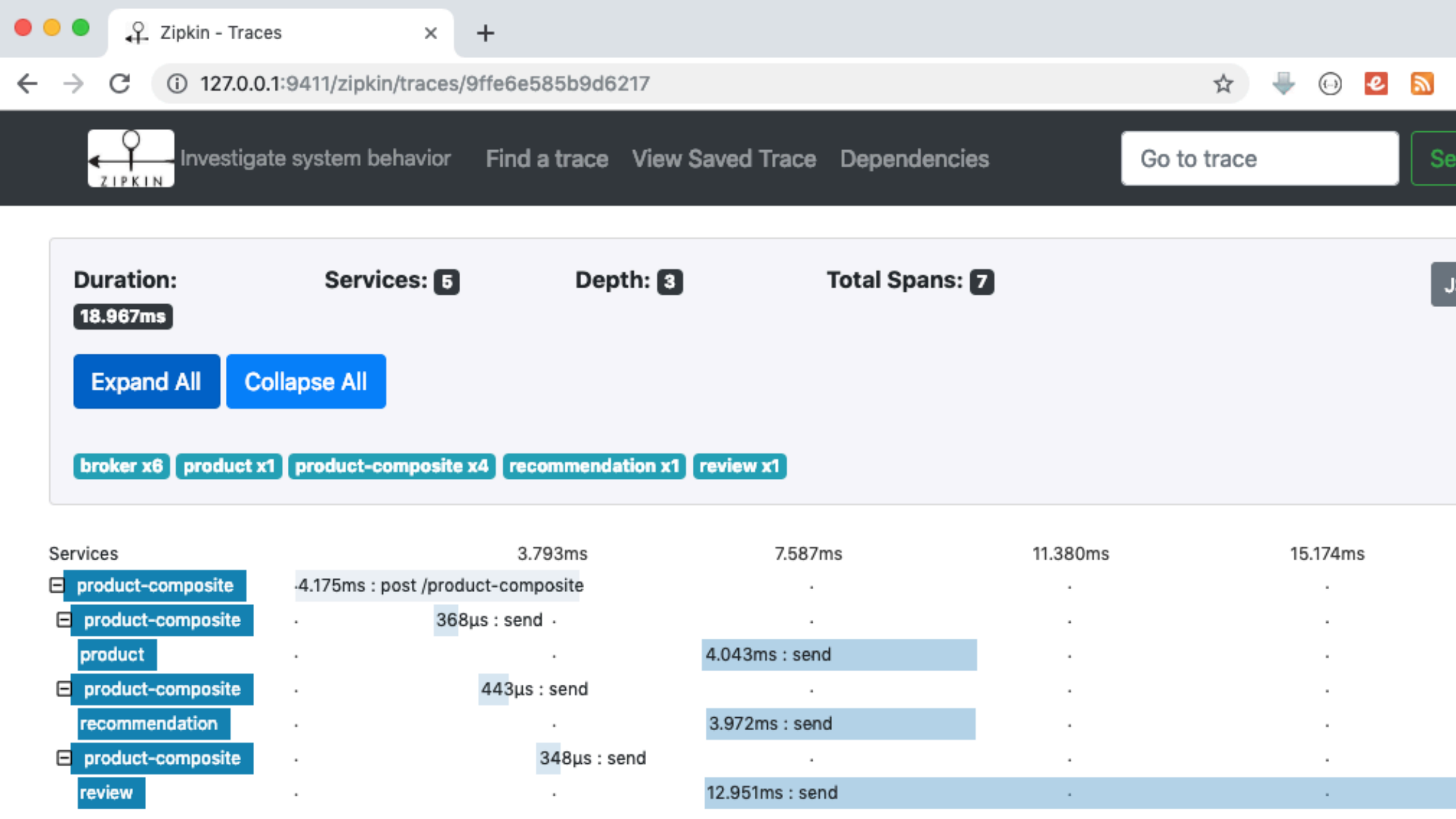
- **Rate limiter** is used to limit the number of requests to service during a specified time period.
- **Bulkhead** is used to limit the number of concurrent requests to a service.
- **Retries** is used to handle random errors that might happen from time to time.
- **Timeout** is used to avoid waiting too long for a response from slow or not responding service.



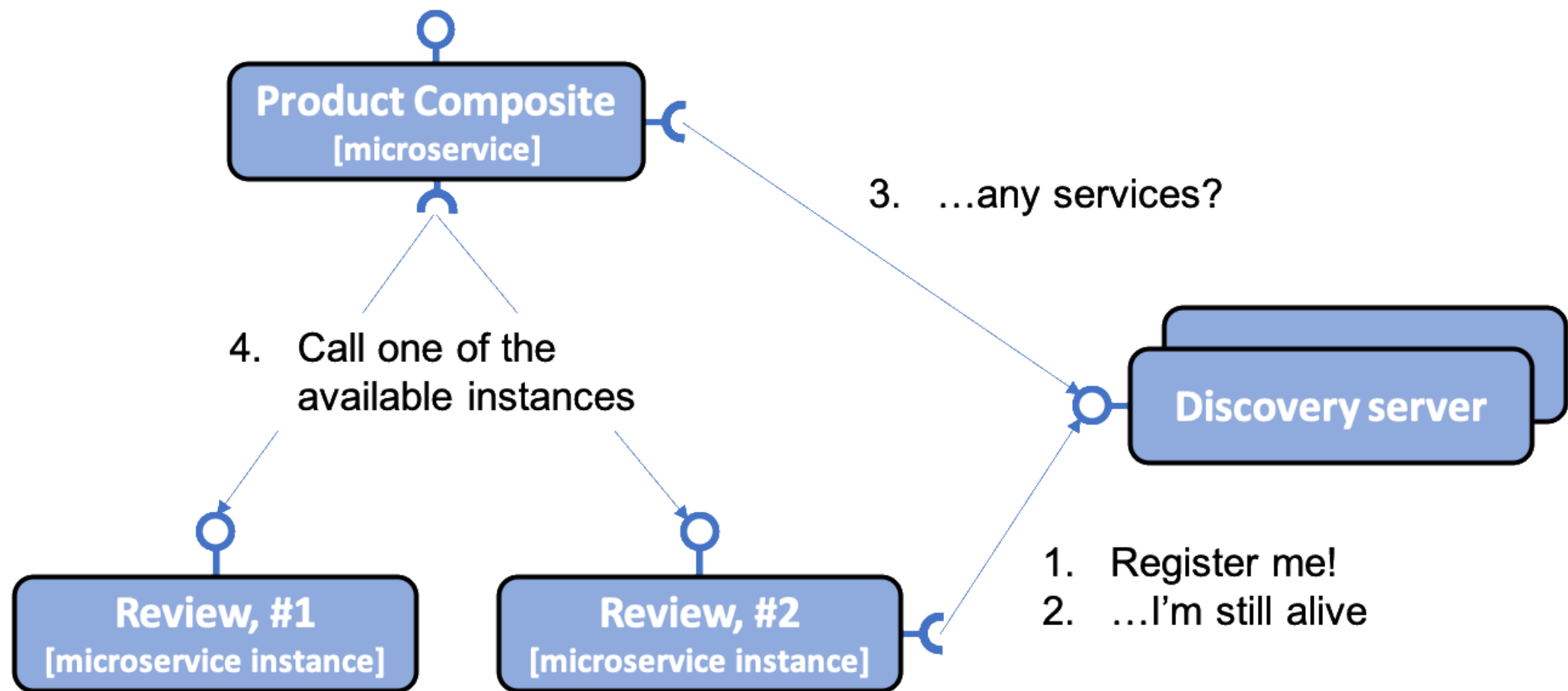
Spring Cloud Sleuth y Zipkin para distributed tracing



The microservice landscape
[System boundary]



Eureka Server



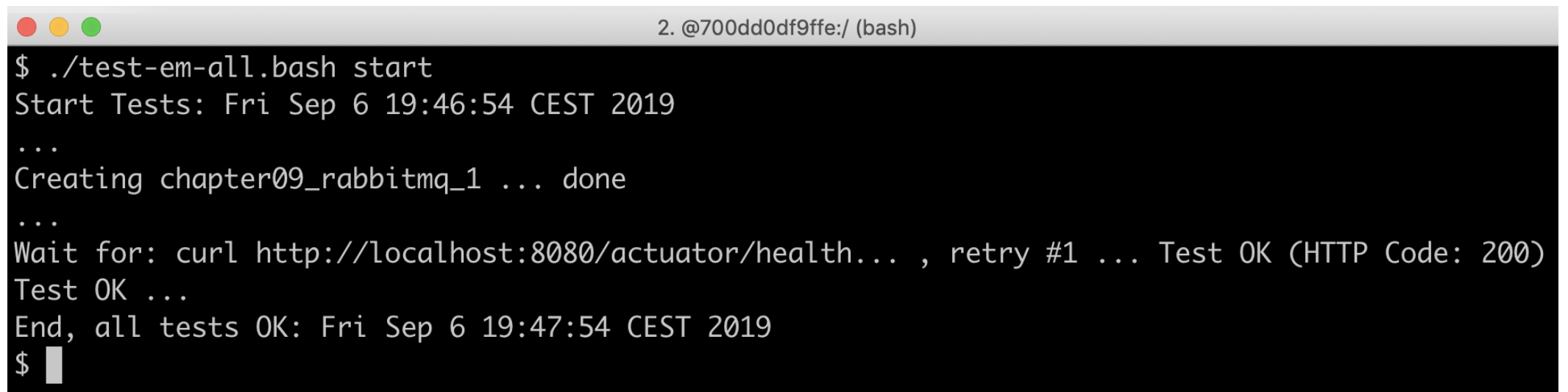
The microservice landscape

[System boundary]

Pasos para ejecutar con eureka:

1) **`./gradlew build && docker-compose build`**

2) **`./test-em-all.bash start`**

A terminal window with a dark background and light gray text. The window title bar shows three colored circles (red, yellow, green) on the left and the text "2. @700dd0df9ffe:/ (bash)" on the right. The terminal content shows the execution of the command `./test-em-all.bash start`. The output includes the start time "Start Tests: Fri Sep 6 19:46:54 CEST 2019", three dots, the message "Creating chapter09_rabbitmq_1 ... done", three dots, the command "Wait for: curl http://localhost:8080/actuator/health... , retry #1 ... Test OK (HTTP Code: 200)", "Test OK ...", and the end time "End, all tests OK: Fri Sep 6 19:47:54 CEST 2019". The prompt "\$" is followed by a cursor bar.

```
2. @700dd0df9ffe:/ (bash)
$ ./test-em-all.bash start
Start Tests: Fri Sep 6 19:46:54 CEST 2019
...
Creating chapter09_rabbitmq_1 ... done
...
Wait for: curl http://localhost:8080/actuator/health... , retry #1 ... Test OK (HTTP Code: 200)
Test OK ...
End, all tests OK: Fri Sep 6 19:47:54 CEST 2019
$
```

Pasos para ejecutar con eureka:

3) docker-compose up -d --scale review=3

Eureka

×

+

←

→

↺

localhost:8761

★

⬇

⌚


e

📡

Aa

👤

⋮

spring

Eureka

HOME

LAST 1000 SINCE STARTUP

System Status

Environment	test	Current time	2019-02-18T08:47:32 +0000
Data center	default	Uptime	00:03
		Lease expiration enabled	true
		Renews threshold	11
		Renews (last min)	238

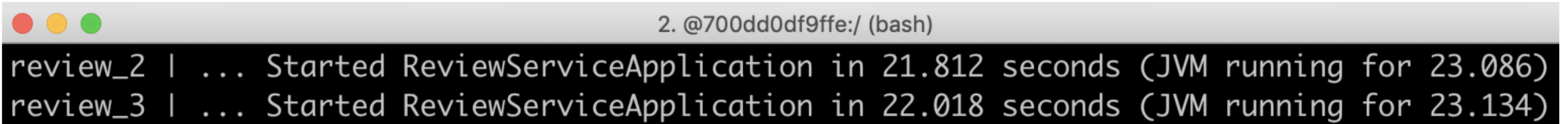
DS Replicas

Instances currently registered with Eureka

Application	AMIs	Availability Zones	Status
PRODUCT	n/a (1)	(1)	UP (1) - fa1fc9bf436f:product:8080
PRODUCT-COMPOSITE	n/a (1)	(1)	UP (1) - 94d4177d3640:product-composite:8080
RECOMMENDATION	n/a (1)	(1)	UP (1) - c6e43eb833e9:recommendation:8080
REVIEW	n/a (3)	(3)	UP (3) - ab40a2d0737a:review:8080 , 7f27b4e75d9d:review:8080 , 61a078ca25c9:review:8080

Pasos para ejecutar con eureka:

4) docker-compose logs -f review

A terminal window with a title bar showing three colored circles (red, yellow, green) and the text "2. @700dd0df9ffe:/ (bash)". The terminal output shows two lines of logs: "review_2 | ... Started ReviewServiceApplication in 21.812 seconds (JVM running for 23.086)" and "review_3 | ... Started ReviewServiceApplication in 22.018 seconds (JVM running for 23.134)".

```
2. @700dd0df9ffe:/ (bash)
review_2 | ... Started ReviewServiceApplication in 21.812 seconds (JVM running for 23.086)
review_3 | ... Started ReviewServiceApplication in 22.018 seconds (JVM running for 23.134)
```

5) Obtener la lista de instancias IDs:

`curl -H "accept:application/json" localhost:8761/eureka/apps -s | jq -r .applications.application[].instance[].instanceId`

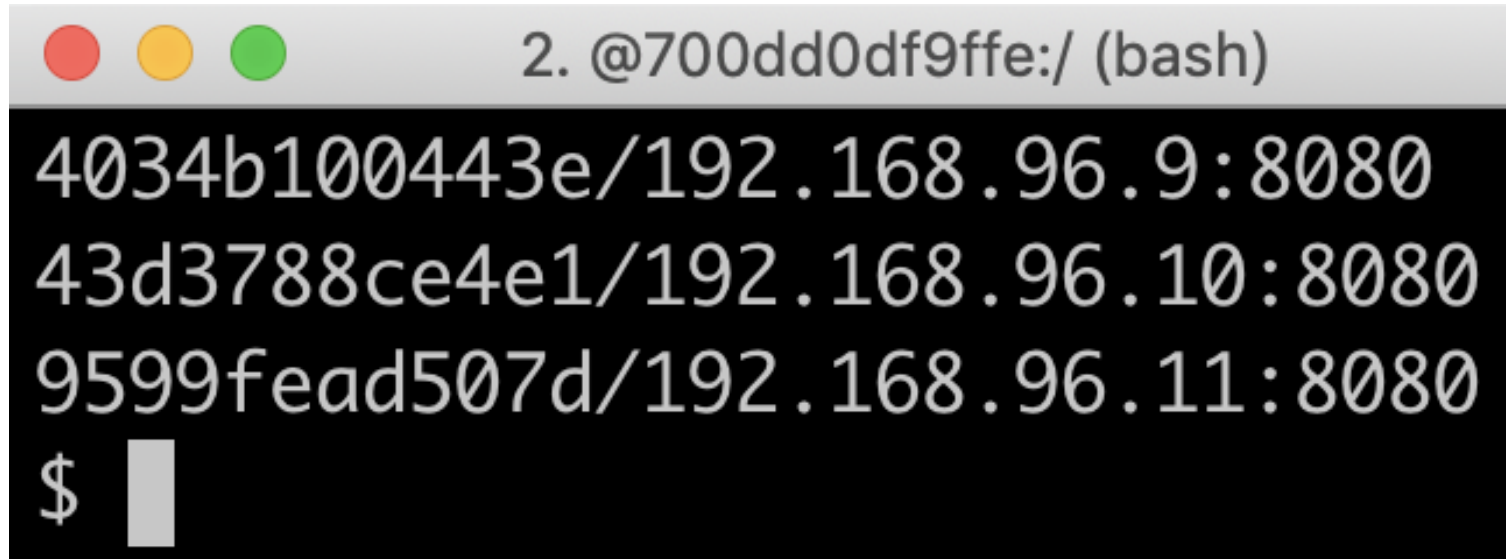
A terminal window with a title bar showing three colored circles (red, yellow, green) and the text "2. @700dd0df9ffe:/ (bash)". The terminal output shows a list of instance IDs and their corresponding application names and ports, one per line: "3c7a676d8a5c:product-composite:8080", "ac0a272e4488:product:8080", "4034b100443e:review:8080", "9599fead507d:review:8080", "43d3788ce4e1:review:8080", and "f50156c9882a:recommendation:8080". The prompt "\$" is visible at the bottom.

```
2. @700dd0df9ffe:/ (bash)
3c7a676d8a5c:product-composite:8080
ac0a272e4488:product:8080
4034b100443e:review:8080
9599fead507d:review:8080
43d3788ce4e1:review:8080
f50156c9882a:recommendation:8080
$
```

Pasos para ejecutar con eureka:

5) Hagamos algunos request y veamos que servicios review responde:

```
curl localhost:8080/product-composite/2 -s | jq -r .serviceAddresses.rev
```



```
2. @700dd0df9ffe:/ (bash)
4034b100443e/192.168.96.9:8080
43d3788ce4e1/192.168.96.10:8080
9599fead507d/192.168.96.11:8080
$
```

A terminal window with a title bar containing three colored circles (red, yellow, green) and the text "2. @700dd0df9ffe:/ (bash)". The terminal output shows three lines of JSON data, each representing a service address: "4034b100443e/192.168.96.9:8080", "43d3788ce4e1/192.168.96.10:8080", and "9599fead507d/192.168.96.11:8080". The prompt "\$" is followed by a vertical bar cursor.

Pasos para ejecutar con eureka:

6) Aplicar un scale down:

```
docker-compose up -d --scale review=2
```

7) Llamar al servicio, pero, con un timeout de tolerancia:

```
curl localhost:8080/product-composite/2 -m 2
```

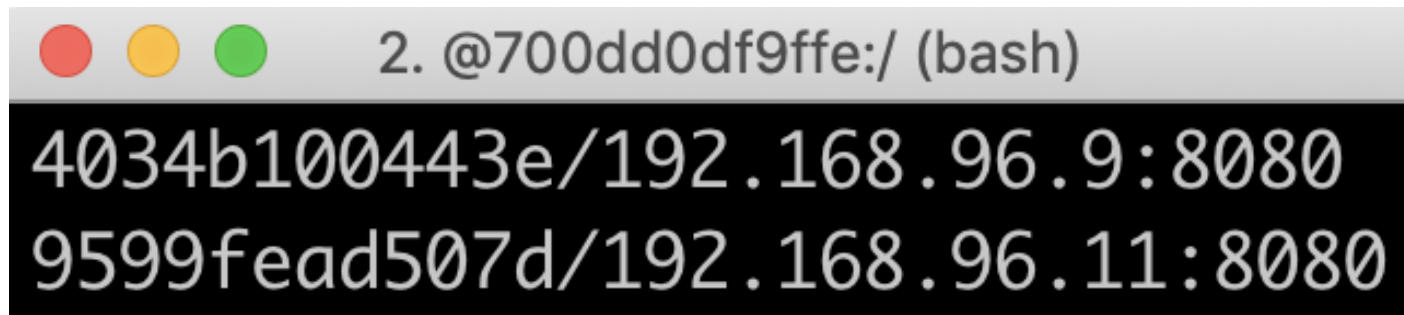
Pasos para probar los microservicios cuando el eureka esta detenido:

8) eureka=0

```
docker-compose up -d --scale review=2 --scale eureka=0
```

9) Llamar al servicio, para ver si sigue funcionando:

```
curl localhost:8080/product-composite/2 -s | jq -r .serviceAddresses.rev
```

A terminal window with a title bar containing three colored circles (red, yellow, green) and the text "2. @700dd0df9ffe:/ (bash)". The terminal has a black background with white text. It displays the output of the curl command from the previous block, showing two lines of JSON data: "4034b100443e/192.168.96.9:8080" and "9599fead507d/192.168.96.11:8080".

```
2. @700dd0df9ffe:/ (bash)  
4034b100443e/192.168.96.9:8080  
9599fead507d/192.168.96.11:8080
```

Pasos para probar los microservicios cuando el eureka esta detenido:

10) eureka=0 & review=1

`docker-compose up -d --scale review=1 --scale eureka=0`