Microservices

1. Creating a Set of Cooperating Microservices

```
java -jar microservices/product-composite-service/build/libs/*.jar &
java -jar microservices/product-service/build/libs/*.jar &
java -jar microservices/recommendation-service/build/libs/*.jar &
java -jar microservices/review-service/build/libs/*.jar &
```

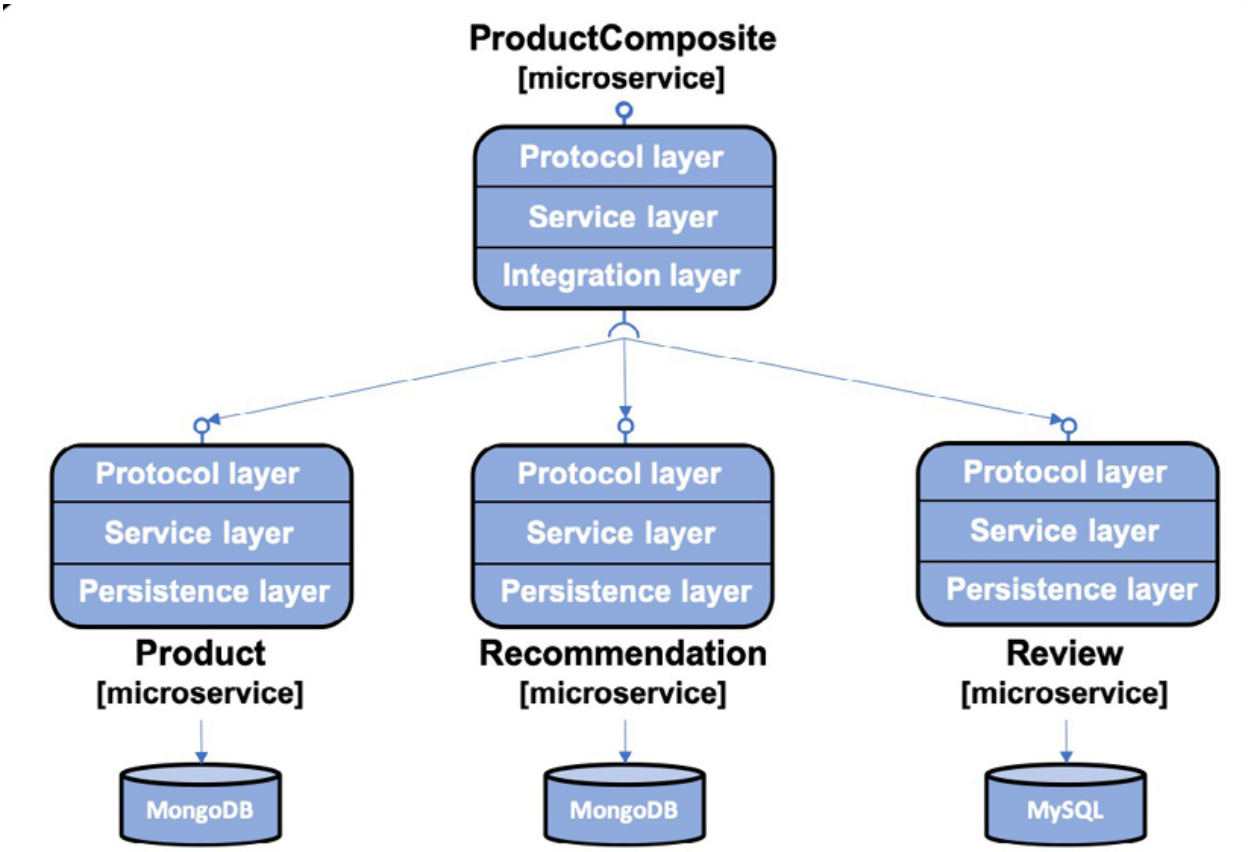
2. Microservices Using Docker

```
./gradlew build
docker-compose build

docker-compose up -d

docker-compose logs -f

curl localhost:8080/product-composite/123 -s | jq .

docker-compose down


./test-em-all.bash start stop

./gradlew clean build && docker-compose build && ./test-em-all.bash start stop
```

3. Adding an API Description Using OpenAPI

```
./gradlew build && docker-compose build && docker-compose up -d
docker-compose up -d
./test-em-all.bash


http://localhost:8080/openapi/swagger-ui.html
```

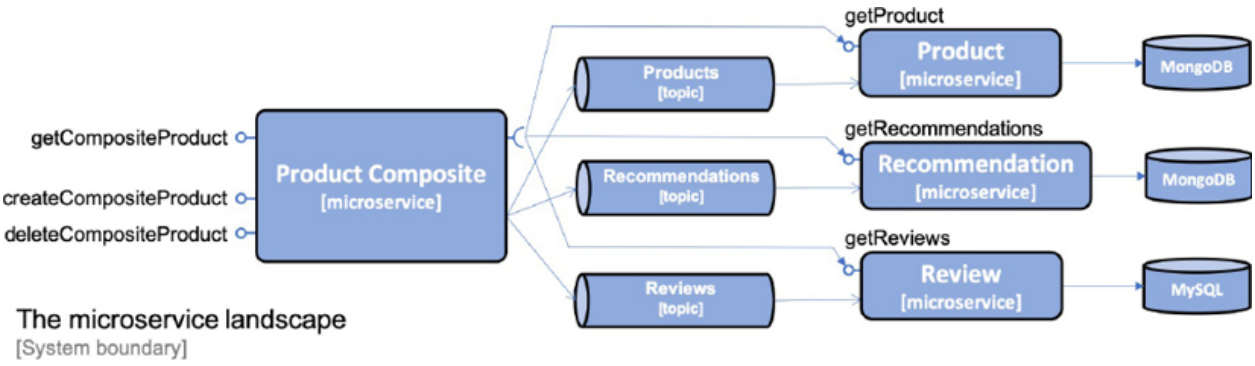
4. Adding Persistence



The microservice landscape
[System boundary]

```
./gradlew microservices:product-service:test --tests PersistenceTests
./gradlew build && docker-compose build && docker-compose up

./test-em-all.bash start stop
```

5. Developing Reactive Microservices




```
./gradlew build && docker-compose build && docker-compose up -d
curl -s localhost:8080/actuator/health | jq -r .status
```

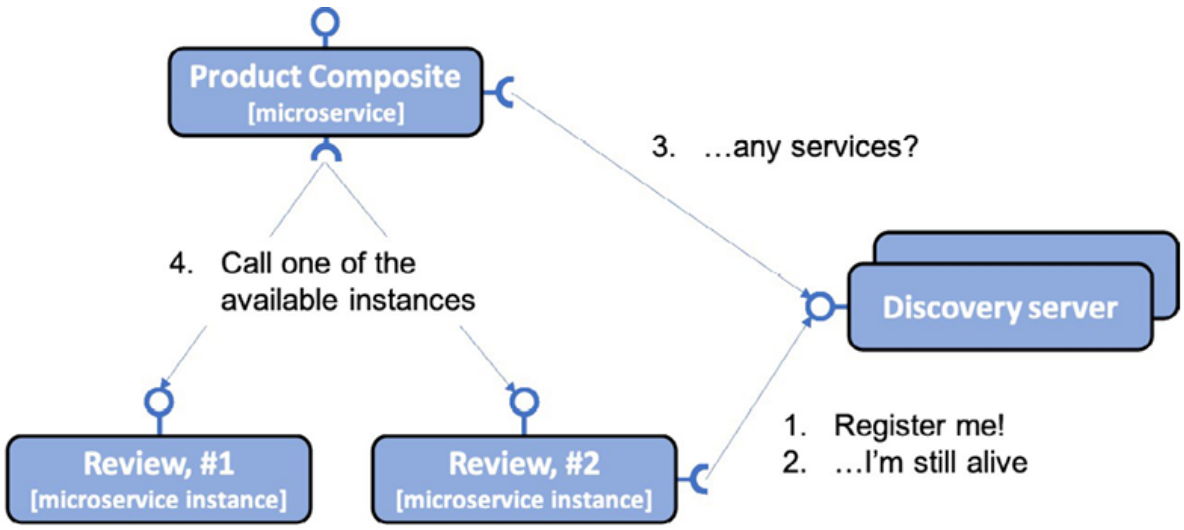
```
body='{"productId":1,"name":"product name C","weight":300, "recommendations":[
{"recommendationId":1,"author":"author 1","rate":1,"content":"content 1"},
 {"recommendationId":2,"author":"author 2","rate":2,"content":"content 2"},
 {"recommendationId":3,"author":"author
3","rate":3,"content":"content 3"}
], "reviews":[
 {"reviewId":1,"author":"author 1","subject":"subject 1","content":"content 1"},
 {"reviewId":2,"author":"author 2","subject":"subject 2","content":"content 2"},
 {"reviewId":3,"author":"author 3","subject":"subject 3","content":"content 3"}
```

```
]}'

curl -X POST localhost:8080/product-composite -H "Content-Type: application/json" --data "$body"
```

```
http://localhost:15672/#/queues
```

```
curl -s localhost:8080/product-composite/1 | jq
curl -X DELETE localhost:8080/product-composite/1
docker-compose down
```

6. Adding Service Discovery Using Netflix Eureka



The microservice landscape
[System boundary]

```
./gradlew build && docker-compose build
./test-em-all.bash start

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

```
http://localhost:8761/
```

```
docker-compose logs review | grep Started
```

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

```
curl localhost:8080/product-composite/1 -s | jq -r .serviceAddresses.rev
```

First, stop the Eureka server and keep the two `review` instances up and running:

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

Try a couple of calls to the API and extract the service address of the `review` service:

```
curl localhost:8080/product-composite/1 -s | jq -r .serviceAddresses.rev
```

Terminate one of the two review instances with the following command:

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

Let's try starting a new instance of the `product` service:

```
docker-compose up -d --scale review=1 --scale eureka=0 --scale product=2
```

Call the API a couple of times and extract the address of the product

service with the following command:

```
curl localhost:8080/product-composite/1 -s | jq -r .serviceAddresses.pro
```

Start the Eureka server with the following command:

```
docker-compose up -d --scale review=1 --scale eureka=1 --scale product=2
```

Make the following call a couple of times to extract the addresses of the product and the review service:

```
curl localhost:8080/product-composite/1 -s | jq -r .serviceAddresses
```