Resiliente Microservices mit Spring Boot und Failsafe

JavaLand 2018

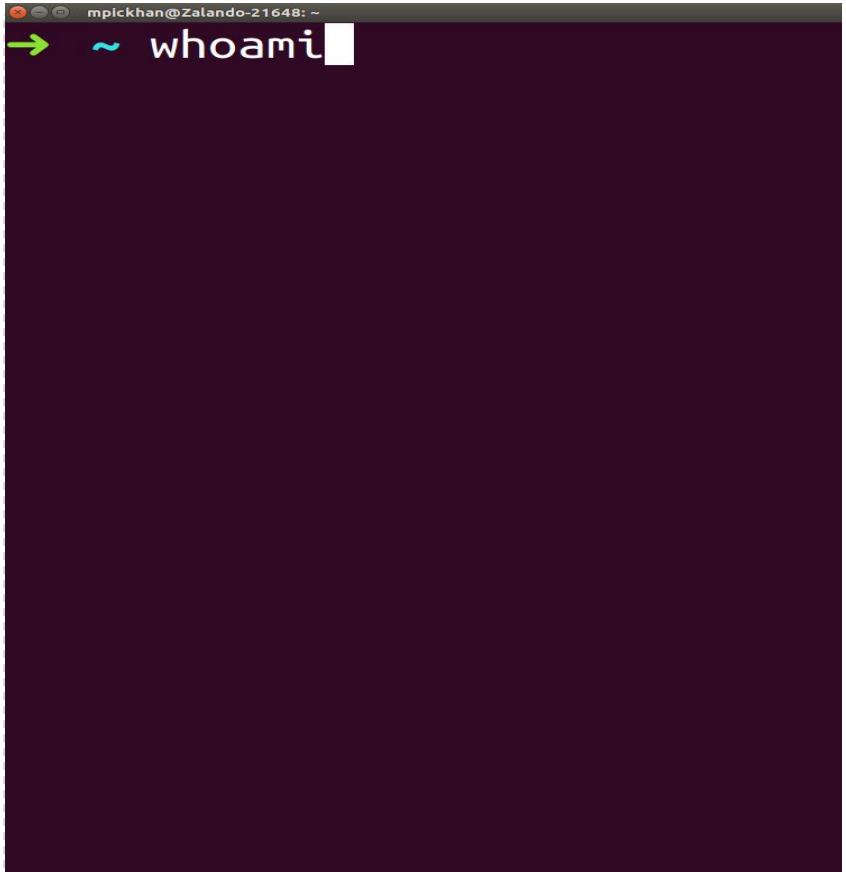
13.03.2018

Malte Pickhan

@mpickhan

Zalando Payments GmbH

WHOAMI



> 5 years experience as Java Backend Engineer

3 years in Telco industry

2 years in E-Commerce

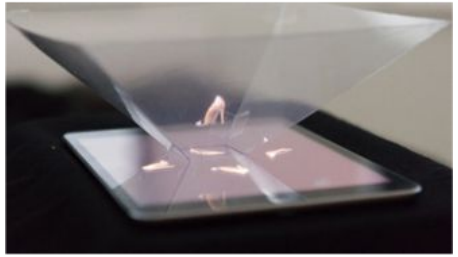
Contributor to open source Project (Failsafe-Actuator)

Published article in Java Magazin (10/17)

zalando

ZALANDO TECHNOLOGY

HOME-BREWED, CUTTING-EDGE & SCALABLE
technology solutions



help our brand to
WIN ONLINE

1,800
employees from

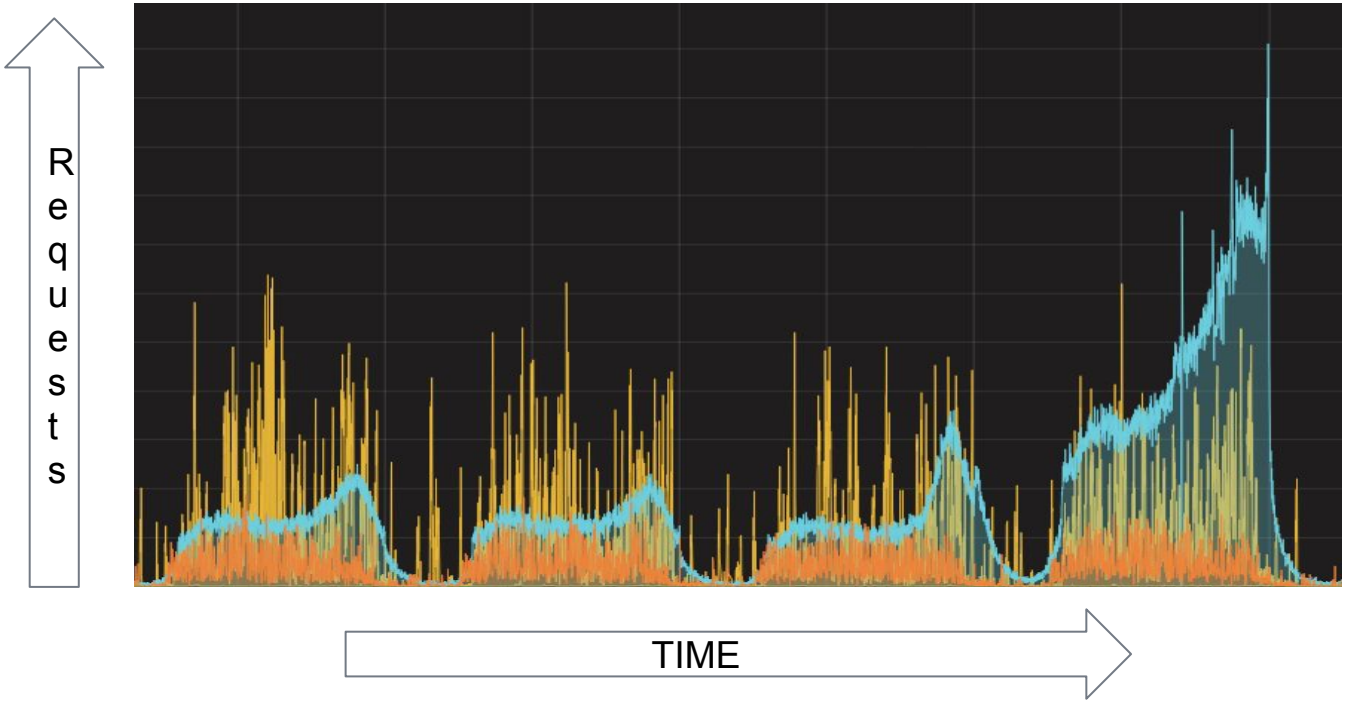
77
nations

6 tech locations
+ HQs in Berlin

zalando


HANDS ON EXPERIENCE ON HOW WE BUILD
RESILIENT MICROSERVICES WITH
SPRING BOOT AND FAILSAFE

BLACK FRIDAY 2017



zalando

RESILIENCE

In computer networking: **resilience** is the ability to provide and maintain an acceptable level of service in the face of faults and challenges to normal operation. [0]

MICROSERVICES

Accept failures instead of avoiding them

- Latency
- Network
- More clients
- Maybe even unknown clients

SPRING BOOT

Takes an opinionated view of building production-ready Spring applications. Spring Boot favors convention over configuration and is designed to get you up and running as quickly as possible. [1]

ACTUATOR

Actuator endpoints allow you to monitor and interact with your application. Spring Boot includes a number of built-in endpoints and you can also add your own. For example the health endpoint provides basic application health information. [2]

FAILSAFE

Failsafe is a lightweight, zero-dependency library for handling failures. It was designed to be as easy to use as possible, with a concise API for handling everyday use cases and the flexibility to handle everything else. [3]

FAILSAFE

Execution Context

Supports retries out of the box

Calls can be put to Lambda expressions or Callbacks

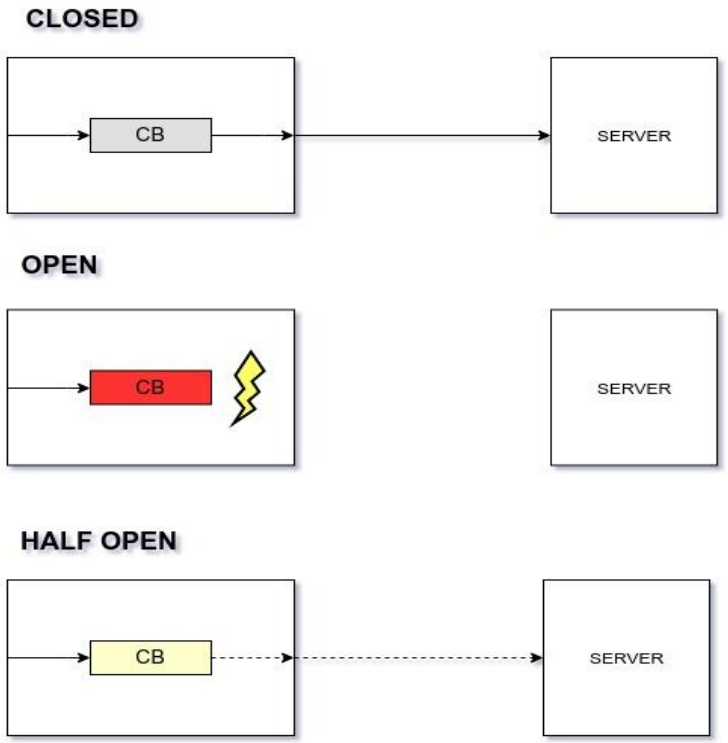
Async execution can be observed via Listener API

EXAMPLE

```
1  public boolean checkSolvency(final ConsumerResource consumerResource) throws URISyntaxException {
2     log.debug("Running solvency check for {}", consumerResource.toString());
3     ResponseEntity<Boolean> booleanResponseEntity =
4       restTemplate.postForEntity(new URI(BASE_URL), consumerResource, Boolean.class);
5     log.debug("Received response {}", booleanResponseEntity.toString());
6     return booleanResponseEntity.getBody().booleanValue();
7   }
```

CIRCUIT BREAKER

- Fail fast
- Be excellent to others
- Three states
  - Closed
  - Open
  - Half Open



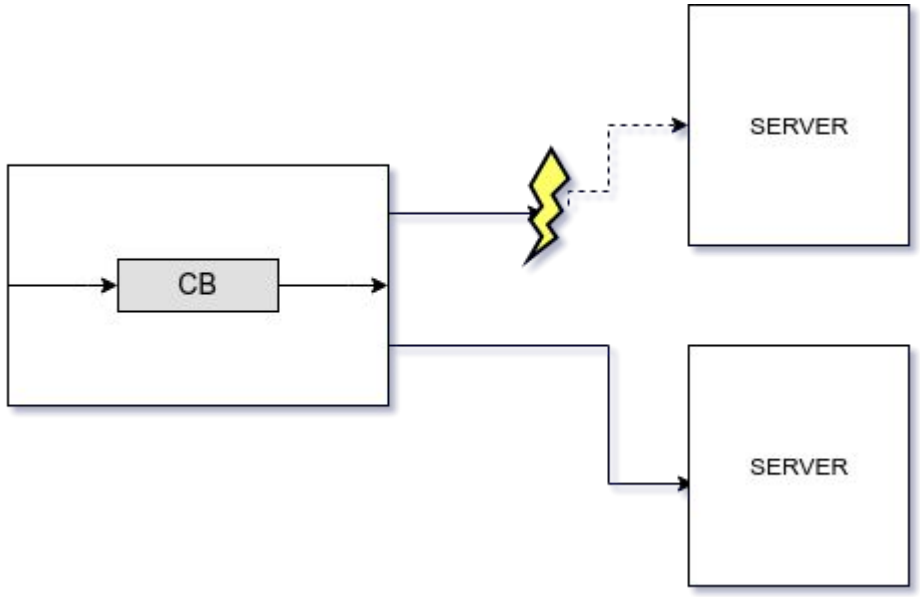
CIRCUIT BREAKER

```
1   private void configureCircuit() {
2     this.circuitBreaker
3             .withFailureThreshold(3, 6)
4             .withDelay(10, TimeUnit.SECONDS)
5             .withSuccessThreshold(3);
6
7     this.circuitBreaker.failOn(Arrays.asList(InterruptedIOException.class, ResourceAccessException.class));
8   }
9
10  public boolean checkSolvency(final ConsumerResource consumerResource) throws URISyntaxException {
11    log.debug("Running solvency check for {}", consumerResource.toString());
12    ResponseEntity<Boolean> booleanResponseEntity = Failsafe
13            .with(this.circuitBreaker)
14            .get(() -> restTemplate.postForEntity(new URI(BASE_URL), consumerResource, Boolean.class));
15    log.debug("Received response {}", booleanResponseEntity.toString());
16    return booleanResponseEntity.getBody().booleanValue();
17   }
```

THIS WAS GOOD, BUT WE CAN DO BETTER

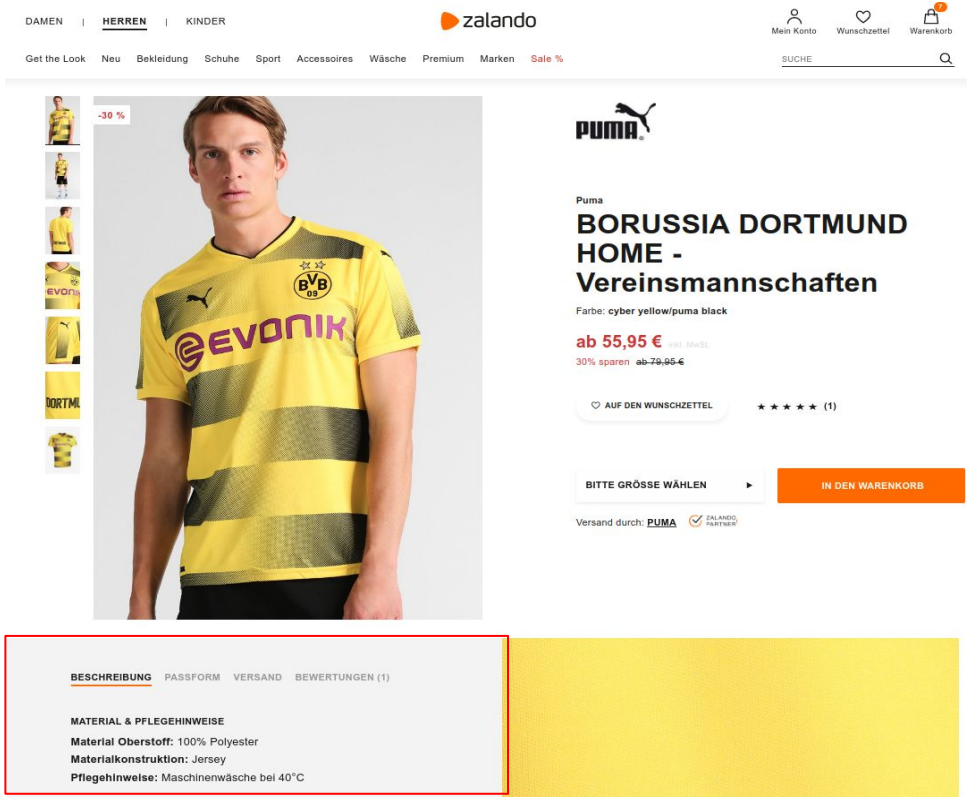
FALLBACK

- Call alternative service in case primary is unavailable
- Not only be available, but also deliver valid results



REAL WORLD EXAMPLE

- Product detail page
- Customer might not be interested in details at all
- Why should we serve a 500 because of that and miss the sale?



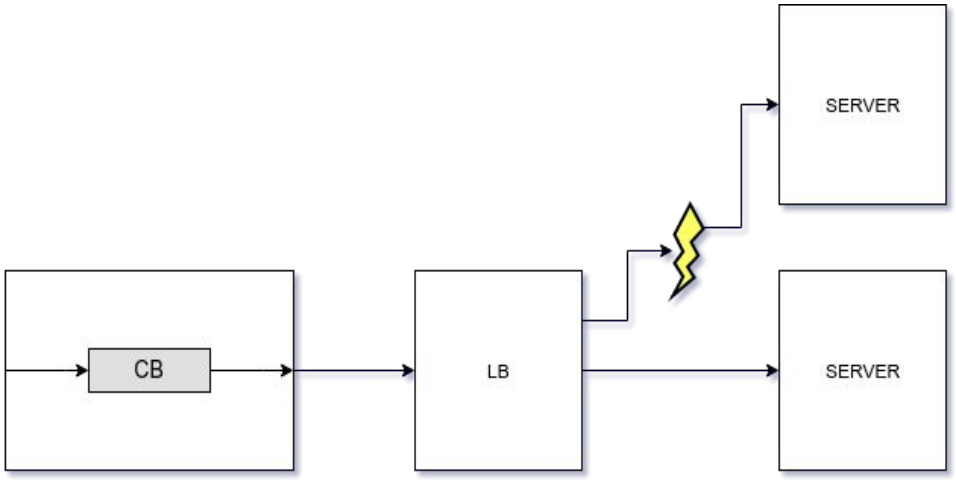
zalando

FALLBACK EXAMPLE

```
1 public boolean checkSolvency(final ConsumerResource consumerResource) throws URISyntaxException {
2     log.debug("Running solvency check for {}", consumerResource.toString());
3     ResponseEntity<Boolean> booleanResponseEntity = Failsafe
4             .with(this.circuitBreaker)
5             .withFallback(callAlternativeProvider())
6             .get(() -> restTemplate.postForEntity(new URI(BASE_URL), consumerResource, Boolean.class));
7     log.debug("Received response {}", booleanResponseEntity.toString());
8     return booleanResponseEntity.getBody().booleanValue();
9   }
```

RETRY

- Retry call in case of **transient** errors
- Try to get as much through at possible
- Every request is a paying customer



zalando

RETRY EXAMPLE

```
1  private void configureRetries() {
2    this.retryPolicy
3            .withMaxRetries(3)
4            .withBackoff(1,5, TimeUnit.SECONDS)
5            .withJitter(10, TimeUnit.MILLISECONDS);
6
7    this.retryPolicy.retryOn(Arrays.asList(InterruptedIOException.class, ResourceAccessException.class));
8  }
9
10 public boolean checkSolvency(final ConsumerResource consumerResource) throws URISyntaxException {
11   log.debug("Running solvency check for {}", consumerResource.toString());
12   ResponseEntity<Boolean> booleanResponseEntity = Failsafe
13           .with(this.circuitBreaker)
14           .withFallback(callAlternativeProvider())
15           .with(this.retryPolicy)
16           .get(() -> restTemplate.postForEntity(new URI(BASE_URL), consumerResource, Boolean.class));
17   log.debug("Received response {}", booleanResponseEntity.toString());
18   return booleanResponseEntity.getBody().booleanValue();
19 }
```

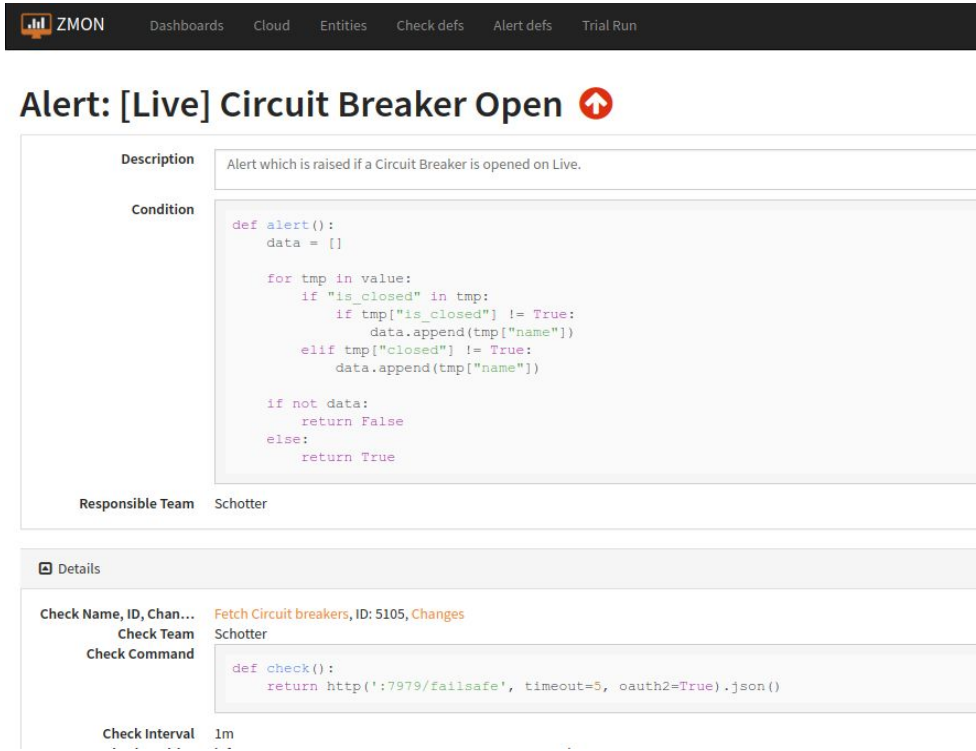
RETRY

delay = backoff + jitterFactor

- backoff is growing exponentially with each retry (1 .. 5 seconds)
- jitterFactor is randomly picked in range (-10 .. 10 ms)
- retries will be equally distributed

MONITORING

- Monitor Circuit Breaker to identify problems early
- Returns JSON containing the state of all registered Circuit Breaker
- Failsafe-Actuator provides this functionality
  - Spring Boot 2 support close to come



zalando

DEMO

zalando

QUESTIONS?

BIBLIOGRAPHY

[0]
https://en.wikipedia.org/wiki/Resilience_(network)
[1]
https://projects.spring.io/spring-boot/
[2]
https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-endpoints.html
[3]
https://github.co/mjhalterman/failsafe