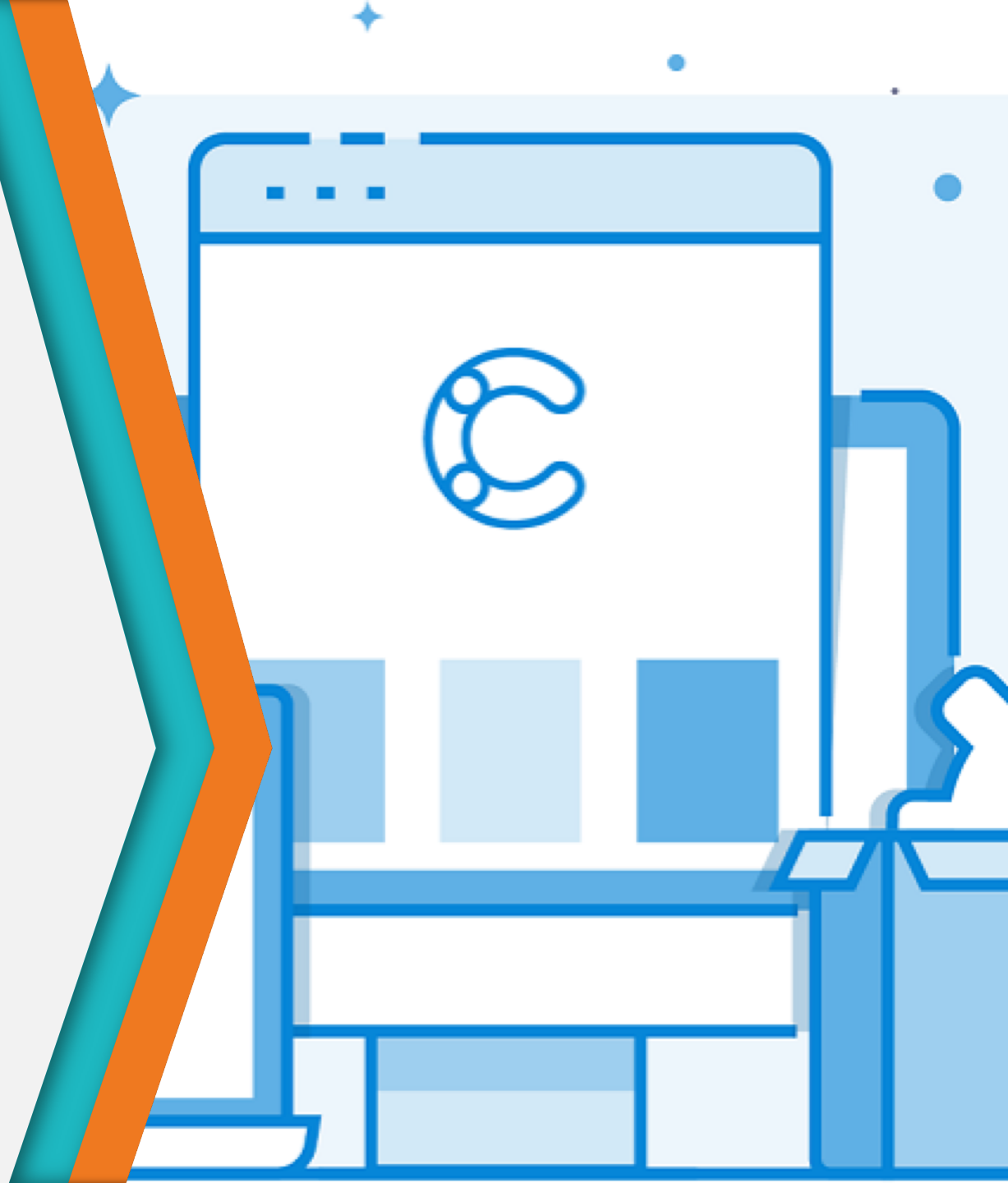
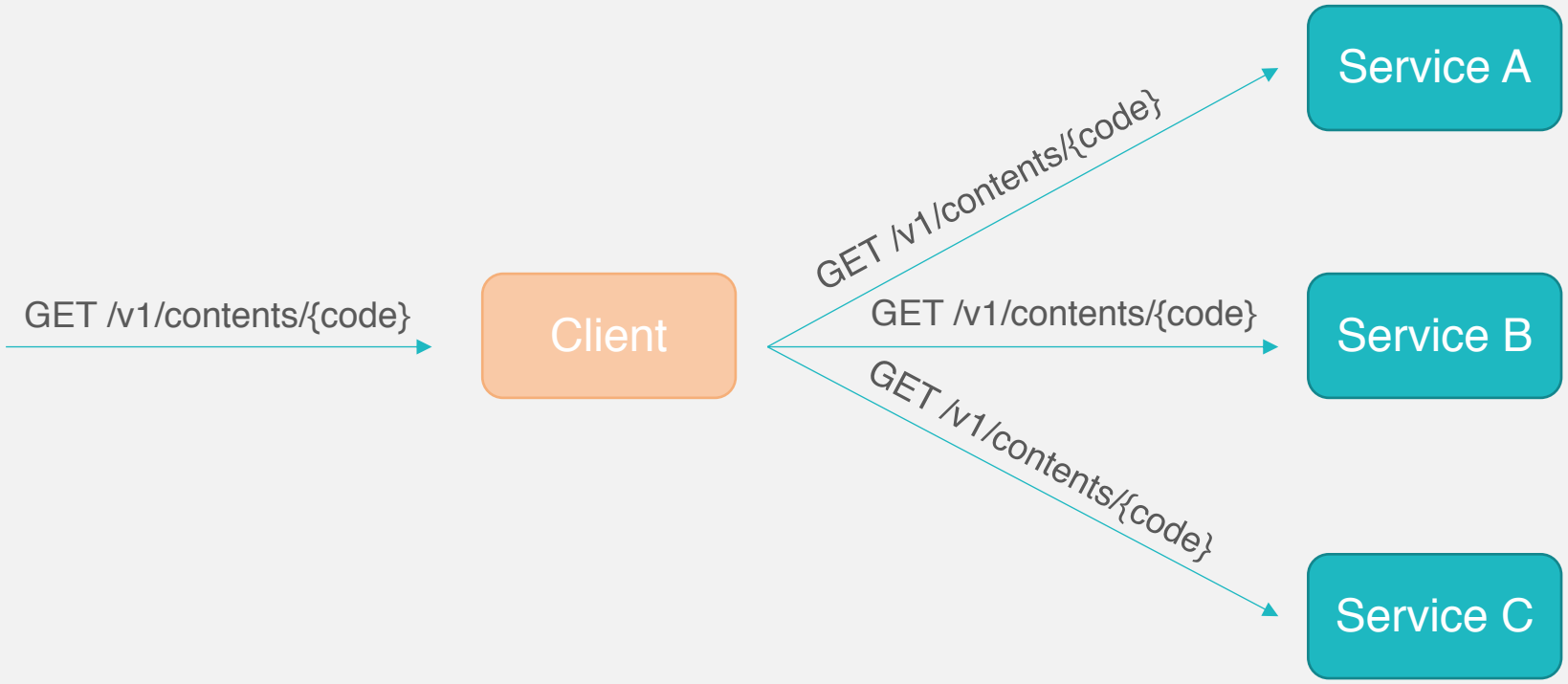
Fault tolerant microservices

Author: Łukasz Wojtaszek



Case study


GET /v1/contents/{code}
Client
GET /v1/contents/{code}
GET /v1/contents/{code}
GET /v1/contents/{code}
Service A
Service B
Service C

Potential problems

1. Latency
   - What can we do in parallel?
   - „Asynchronous by default, synchronous when necessary"
2. Network issues
   - General network problems (network overloaded, peaks)
   - DNS problems
   - Wrong service host/port used for communication with Service
3. Applications issues
   - Service is not working
   - Service has performance issues (huge latentcies while processing requests)
4. Endpoints issues
   - Service endpoints were changed without telling Client
5. Error codes
   - Resource not found – 404
   - Other codes

Worst case scenario: Problem in one Service breaks everything and error is propagated to the whole system

Solutions

1. Latency
   - Solution: **Feign** + **RxJava** for async calls
2. Network issues

   We can't guarantee that network won't have any problems but we can be prepared for this.
   - Solution: **Feign** (retryer) + **Hystrix** (fallback)
3. Applications issues
   - Solution: **Hystrix** (fallback + circuit breaker)
4. Endpoints issues
   - Solution: API versioning
5. Error codes
   - Solution: **Feign** (decode404) + **Hystrix** (fallback)

API Versioning – how to

1. Use internal model structures for logic in the application
2. Create api-model library (generated from swagger yaml file) per API version

```
<dependency>
  <groupId>com.luwojtaszek.client</groupId>
  <artifactId>client-api-model-v1</artifactId>
  <version>1.0.25</version>
</dependency>
<dependency>
  <groupId>com.luwojtaszek.client</groupId>
  <artifactId>client-api-model-v2</artifactId>
  <version>2.0.1</version>
</dependency>
```

3. Create separate controllers (generated from swagger yaml file), mappers per API version

API Versioning – pros and cons

1. Pros:
   - ✓ Modules are separated – you can deploy them separately at any time, not in one bundle when you work on some feature
   - ✓ It is easier to rollback if something went wrong (just decrease api version in application.properties or Docker or Rancher)
2. Cons:
   - ❖ pom.xml grows up (dependency per API version)
   - ❖ code grows up (mappers, controllers per API version)

You can provide some cleanup policy that if all dependend modules (microservices) are bumped up to the latest version of some API then in this API you will remove old code and api-model libraries.

Asynchrony – what can we use and why RxJava?

1. What can we use?
   - Future / ListenableFuture
   - CompletableFuture
   - RxJava
   - Project Reactor (spring 5)

2. Why RxJava?
   - Well known and cross-language solution (Angular 2+ uses this)
   - Nice syntax (chaining - builder pattern)
   - Huge list of operators
   - Easy way of jumping between thread pools
   - Supported by Feign and Hystrix!

Feign

„Feign is a declarative web service client. It makes writing web service clients easier. To use Feign create an interface and annotate it.”

Feign

```
@FeignClient(name = "service1", url = "${client.service1.url}")
public interface Service1Client {

  @RequestLine("GET /contents/{contentsCode}")
  Single<Optional<ServiceResponse>> getContents(@Param("contentsCode") String contentsCode);
}
```

Feign + Spring

```
@FeignClient(name = "service1", url = "${client.service1.url}")
public interface Service1Client {

  @GetMapping(value = "/contents/{contentsCode}")
  Single<Optional<ServiceResponse>> getContents(@PathVariable("contentsCode") String contentsCode);
}
```

Feign – features

1. No implementation
2. Supports Spring annotations
3. Integrates with other Netflix libraries (Ribbon, Hystrix, etc.)
4. Resolves 404 error codes as empty values (null, emtpy list)
5. Contains retryer mechanism
6. Allows to declare own interceptors
   - PropagateIncomingHeaders interceptor
   - Oauth2 interceptor

Hystrix

„Hystrix is a latency and fault tolerance library designed to isolate points of access to remote systems, services and 3rd party libraries, stop cascading failure and enable resilience in complex distributed systems where failure is inevitable.”

Features:

- Creates separate thread-pool for each Feign client (it uses name attribute)
- Wraps each request into separate thread (async by default)
- Supports: Future and RxJava types as a response
- Fallback methods (handling timeouts, error codes, etc.)
- Circuit Breaker (cut of failing service for a few seconds and let it heal itself)
- Hystrix Dashboard – shows current status of calls made to services

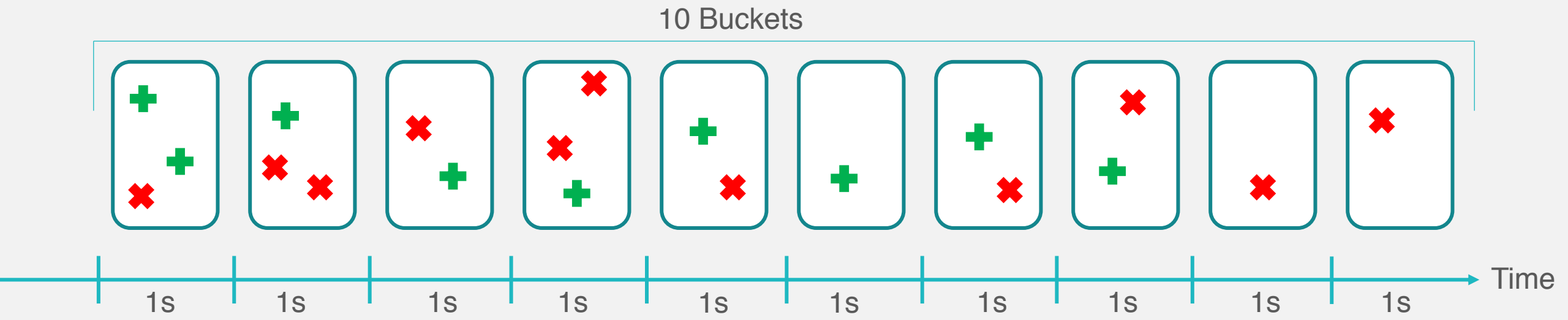
Hystrix – usage with Feign

```
@FeignClient(name = "service1", url = "${client.service1.url}", fallbackFactory = Service1ClientFallbackFactory.class)
public interface Service1Client {

  @GetMapping(value = "/contents/{contentsCode}")
  Single<Optional<ServiceResponse>> getContents(@PathVariable("contentsCode") String contentsCode);
}
```

```
@Component
public class Service1ClientFallbackFactory implements FallbackFactory<Service1Client> {

  @Override
  public Service1Client create(Throwable cause) {
    return new Service1ClientFallback(cause);
  }
}
```

```
@RequiredArgsConstructor
public class Service1ClientFallback implements BaseClientFallback, Service1Client {

  private final Throwable cause;

  @Override
  public Single<Optional<ServiceResponse>> getContents(String contentsCode) {
    handleCommonExceptions(cause);
    // Do whatever you want like: fetch from cache, return dummy object, etc.
    return Single.just(Optional.of(ServiceResponse.builder()
      .message("ServiceA - mocked response")
      .build()));
  }
}
```

Hystrix – circuit breaker



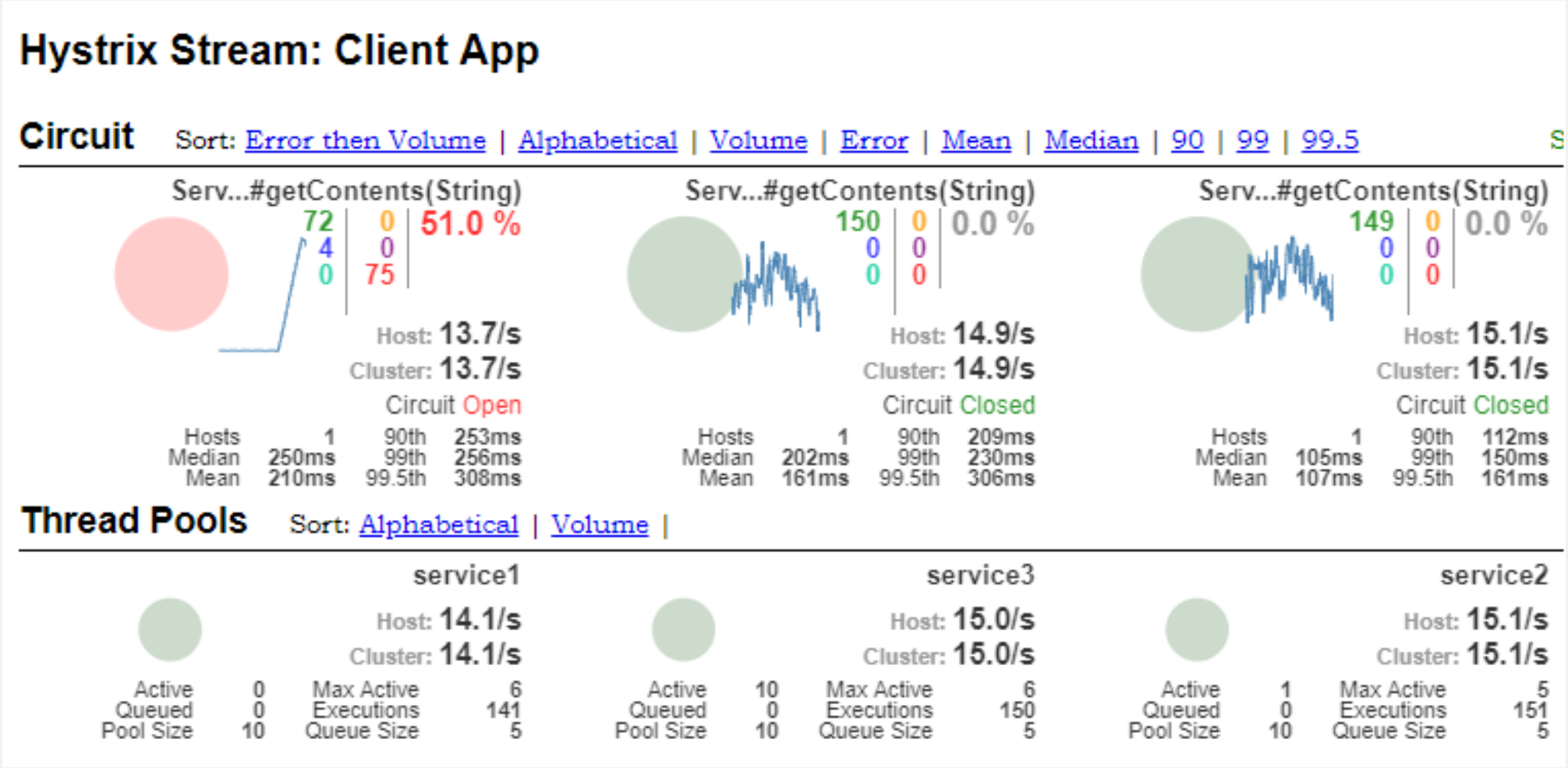
Configuration:

```
hystrix.command.default.circuitBreaker.requestVolumeThreshold: 20
hystrix.command.default.circuitBreaker.sleepWindowInMilliseconds: 5000
hystrix.command.default.circuitBreaker.errorThresholdPercentage: 50
hystrix.command.default.metrics.rollingStats.timeInMilliseconds: 10000
hystrix.command.default.metrics.rollingStats.numBuckets: 10
hystrix.command.default.metrics.healthSnapshot.intervalInMilliseconds: 500
```

Summary:

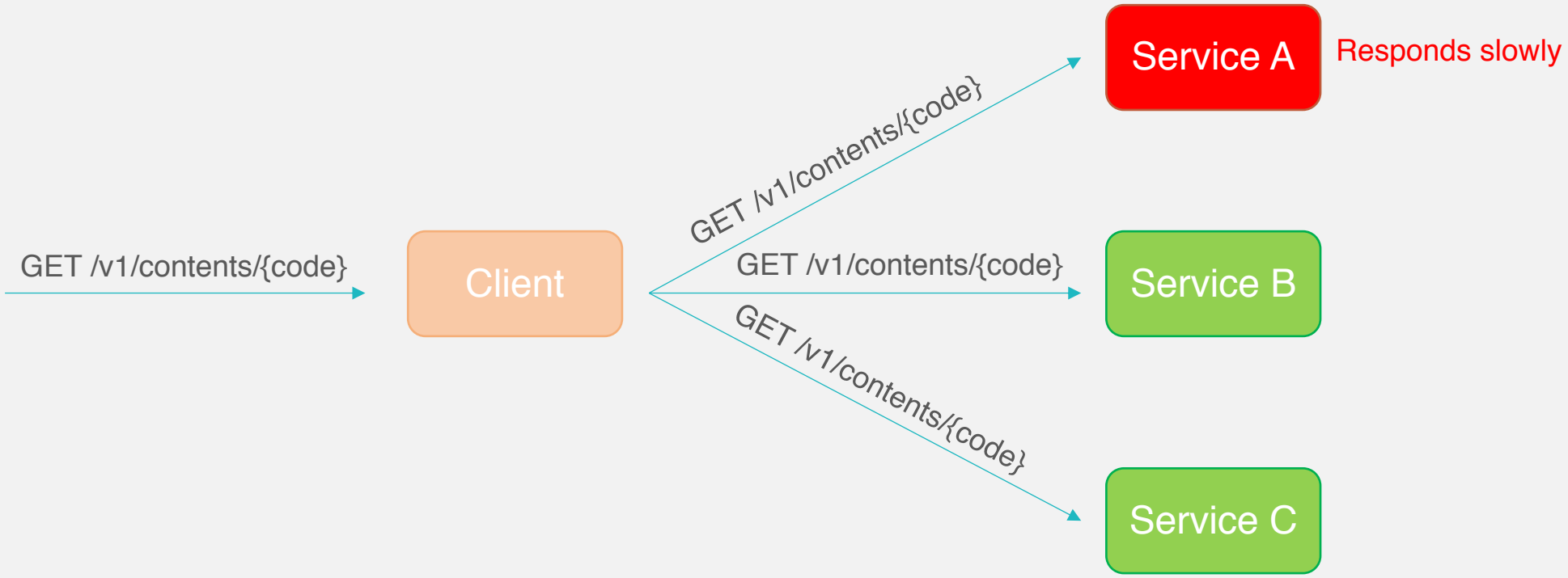
- All requests: 20
- Success requests: 9 = 45%
- Failed requests: 11 = 55%
- Circuit: Open (cut off for 5s)

Hystrix – dashboard



Let’s see the code

Case 1 – calls are not dependent



We can send all requests asynchronously at the same time ☺