Reactive Socket - the future of microservices communication

Maciej Ciołek

CTO @ codeheroes_

~~Reactive Socket~~ - the future of microservices communication

Maciej Ciołek

CTO @ codeheroes_

RSocket - the future of microservices communication

Maciej Ciołek

CTO @ codeheroes_

RSocket maintainers:

Who we are?

We are working on different components of same product

We are familiar with HTTP

We need to have communication between components

We need to implement integrations with external parties

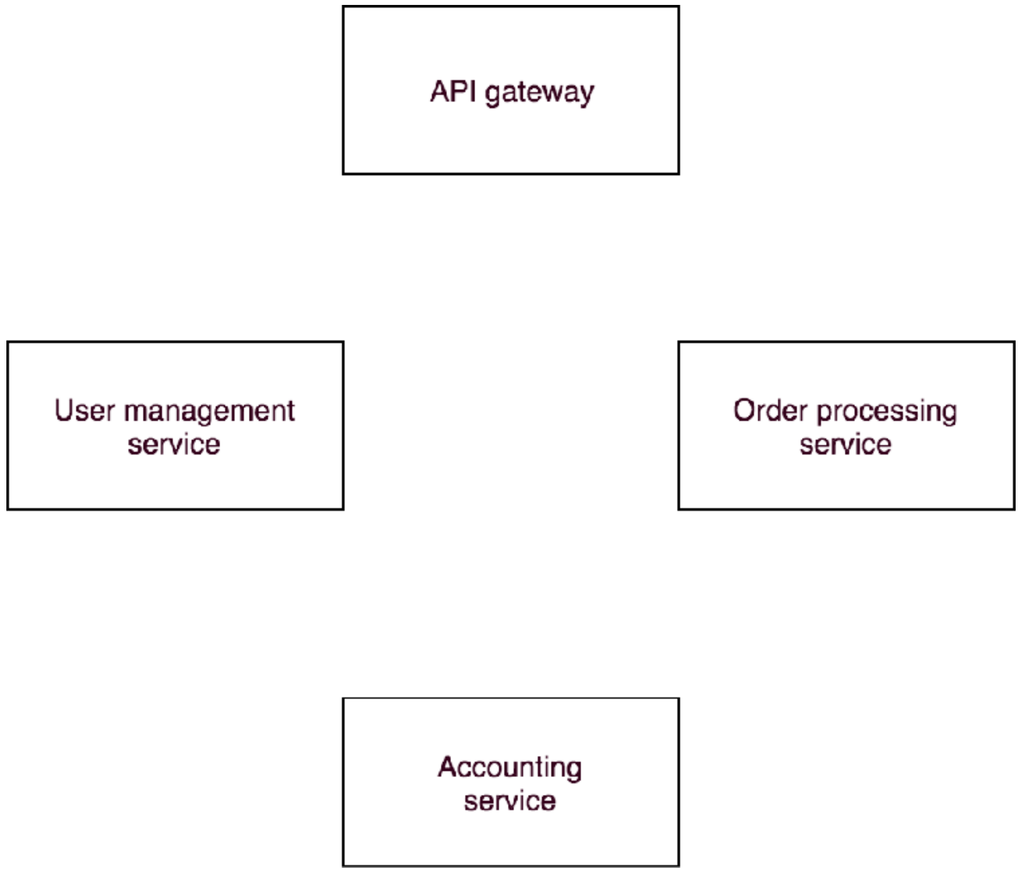
Microservice

responsible only for specific part of business domain

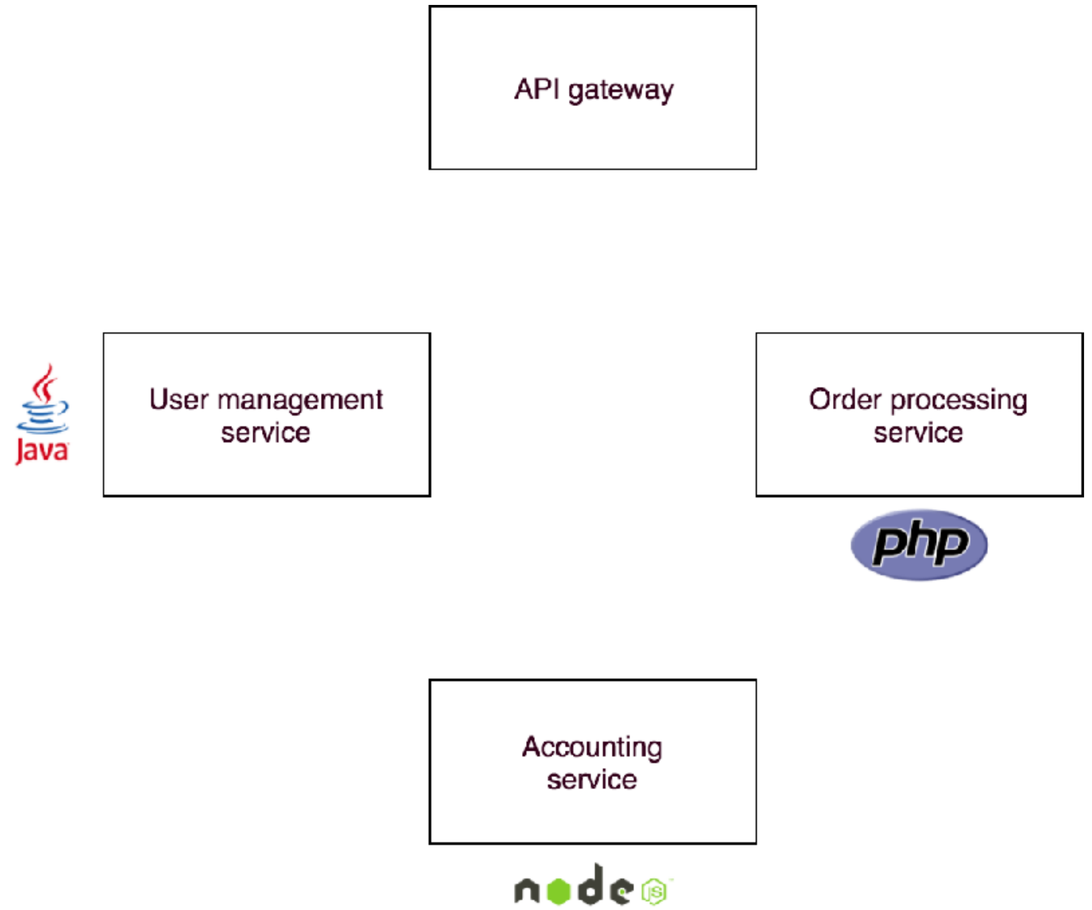
loosely coupled

lightweight protocols

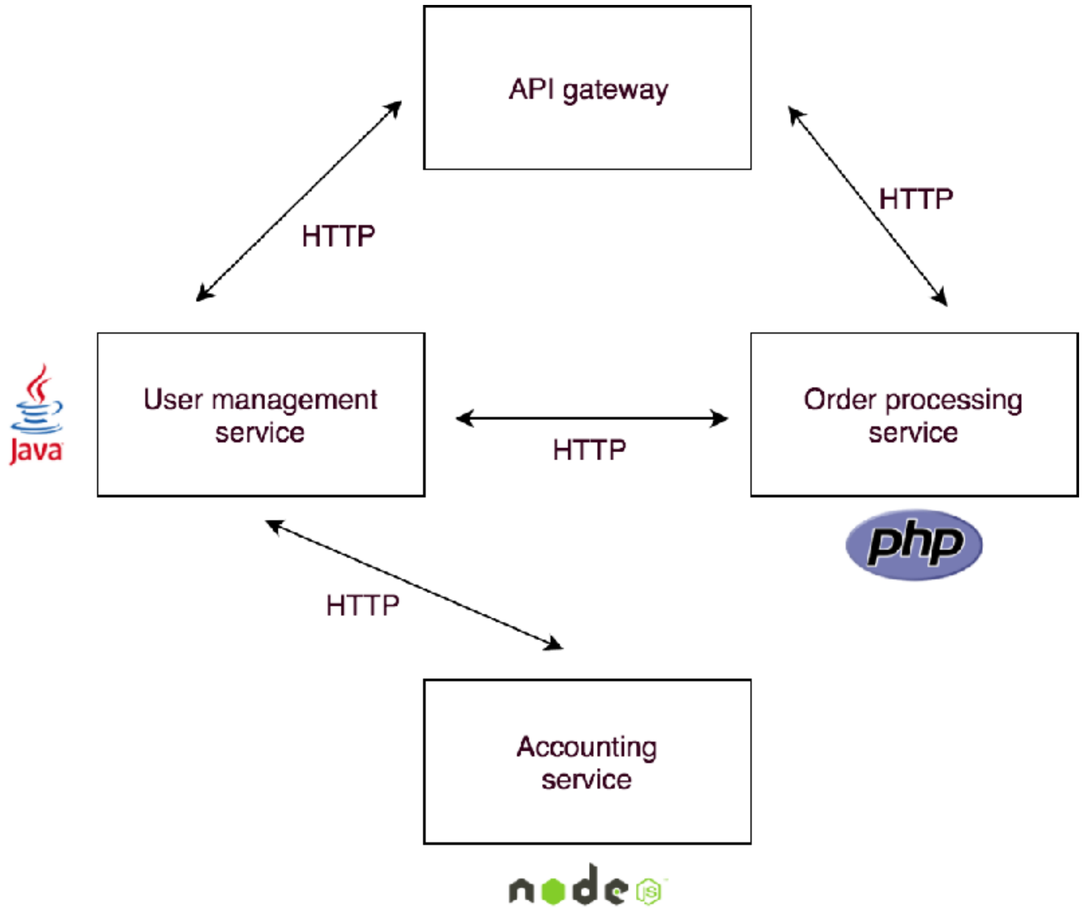
Microservices Architecture



Teams = different languages



HTTP communication



HTTP drawbacks

Synchronous

Cannot handle many parallel request per connection

Pipelining does not work well

Supports only request-response model

HTTP is not enough

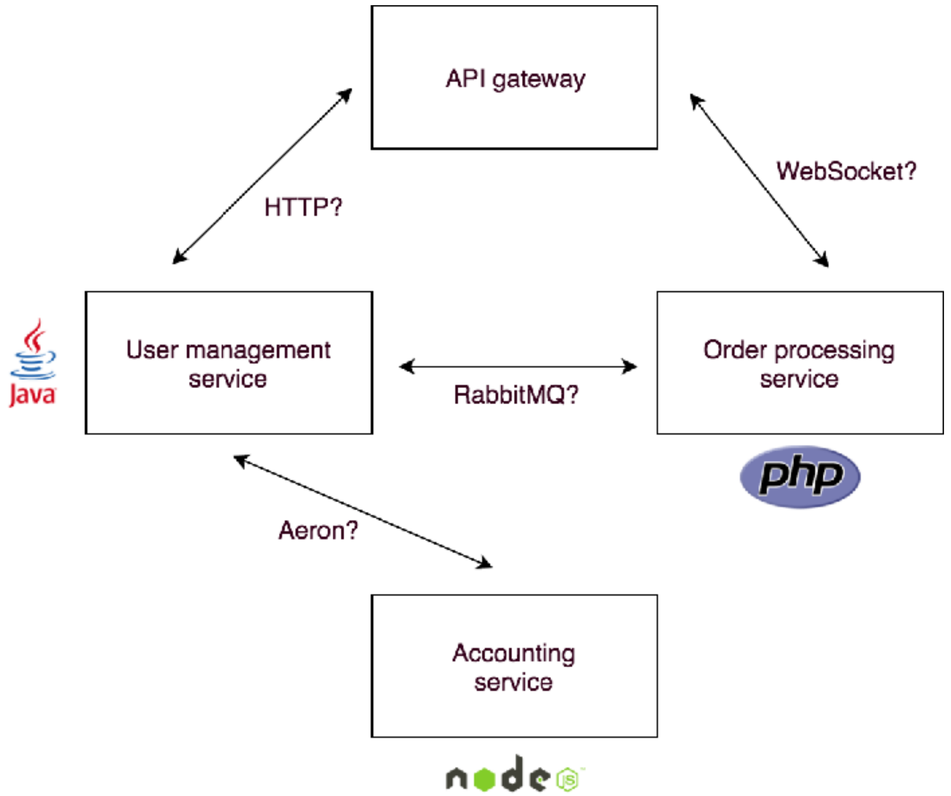
We need asynchronous approach

We need streaming API

We need publish and subscribe approach

We need persistent subscriptions

Communication coordination



This approach causes difficulties

RSocket - why?

Because of all I have presented so far

Reactive Manifesto

Reactive Streams

Because being reactive is great choice nowadays

RSocket - basic concept

asynchronous communication

multiplexed streams over single connection

no synchronous blocks waiting for response

RSocket - interaction models

fire-and-forget

request -> response (single response)

request -> stream (many responses)

channel (bi-directional)

Same as HTTP

RSocket - fire-and-forget

single request with no response

```
Future<Void> result = client.fireAndForget(message);
```

RSocket - request / response

single request with single response

```
Future<Payload> response = client.requestResponse(request);
```

RSocket - request / multi-response

single request with stream of responses

```
Publisher<Payload> output = client.requestStream(request);
```

RSocket - request channel

bi-directional communication channel

```
Publisher<Payload> output = client.requestChannel(Publisher<Payload> input);
```

RSocket - flow control

RSocket - flow control

RSocket - flow control

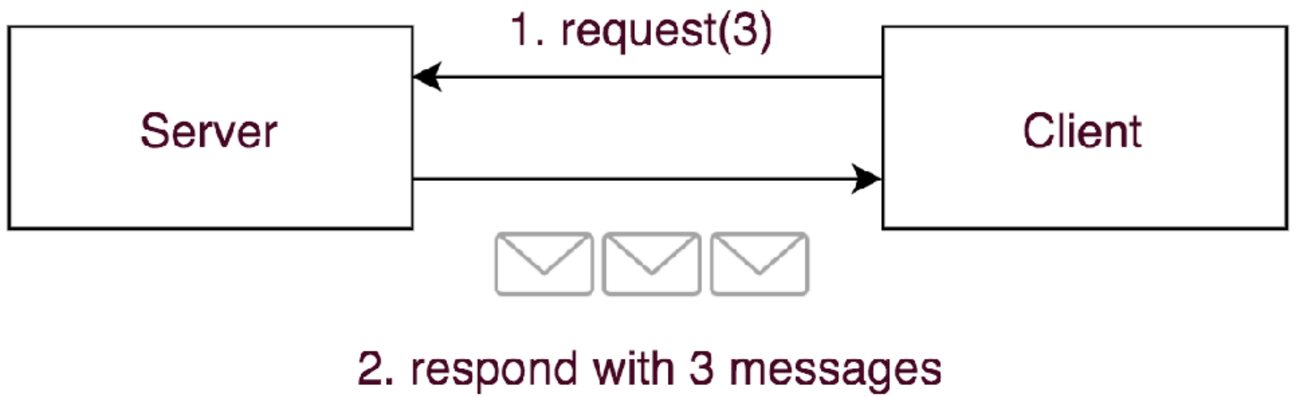
RSocket - flow control

RSocket - flow control

RSocket - flow control

Async Pull - request(n)

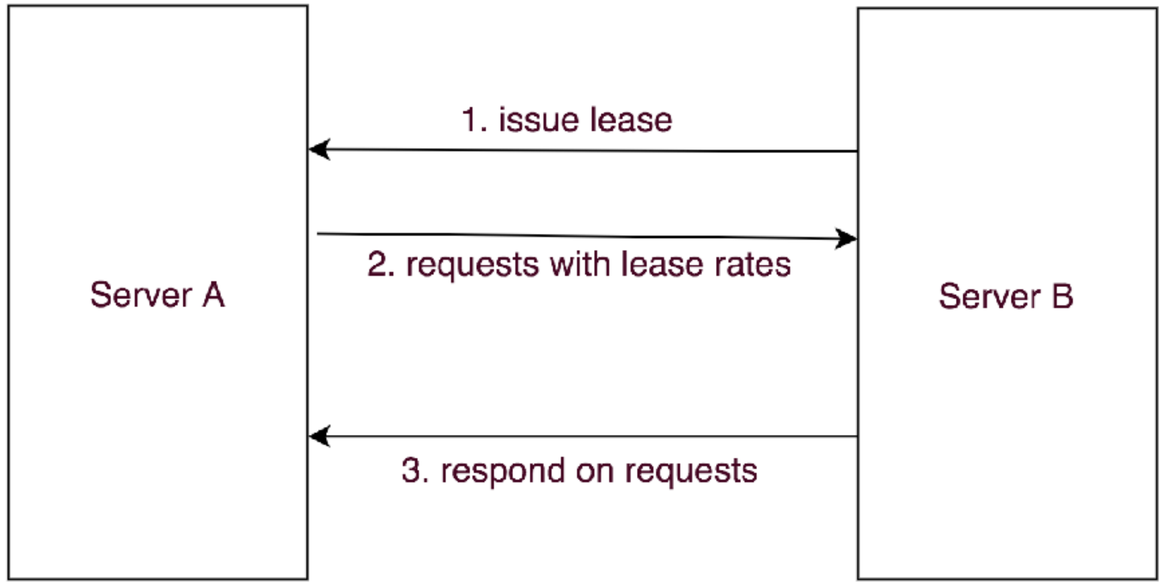
Client request n messages from server



RSocket - flow control

Leasing

In server-to-server scenario one side can issue lease
based on it`s knowledge in order to control requests rates



RSocket - transport layer

WebSockets

TCP

Aeron

etc.

DEMO

RSocket - status

Under development

Need to be battle tested

Almost production ready

Visit: http://rsocket.io

RSocket - why you should try?

1 protocol with 4 interaction models

build-in flow control mechanism

build on top of battle tested TCP or WebSockets

Thank you!

We are using RSocket in internal project
and in future we are going to provide Scala API.

Code available here:
https://github.com/codeheroesdev/rsocket-example

Maciej Ciołek

maciej@codeheroes.io

linkedin.com/in/maciejciolek

@MaciejCiolek