Microservices communication and integration: what are my options?

Kate Stanley
@KateStanley91

Jfokus – February 2016

Introduction

Software engineer

WebSphere Liberty 

Microservices

GAMEON

A Throwback Adventure

@KateStanley91

Contents

What are microservices?

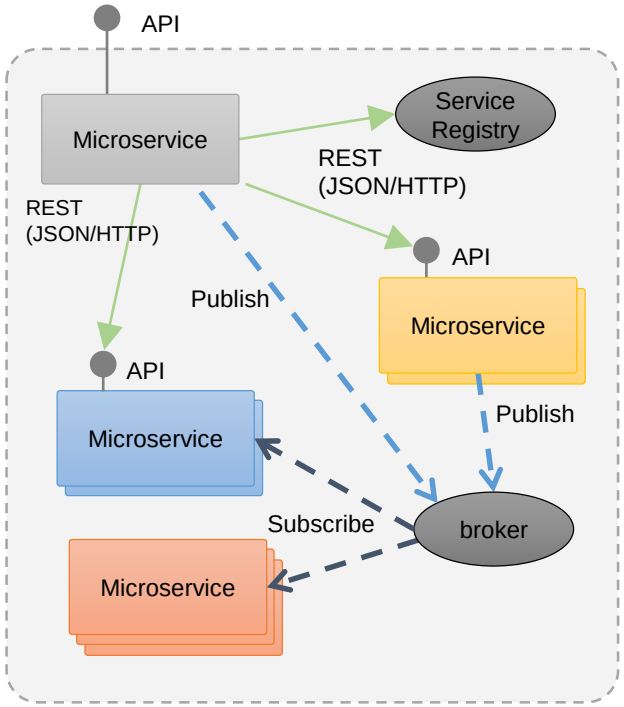
Microservices are used to…

…compose a complex application using:

- “small”
- independent (autonomous)
- replaceable
- processes

…that communicate via:

- language-agnostic APIs



Why is communication important?

How my team communicates

Different locations

Different timezones

Different languages…?!

"The United States and Great Britain are two countries separated by a common language."

How my team communicates

Data type:

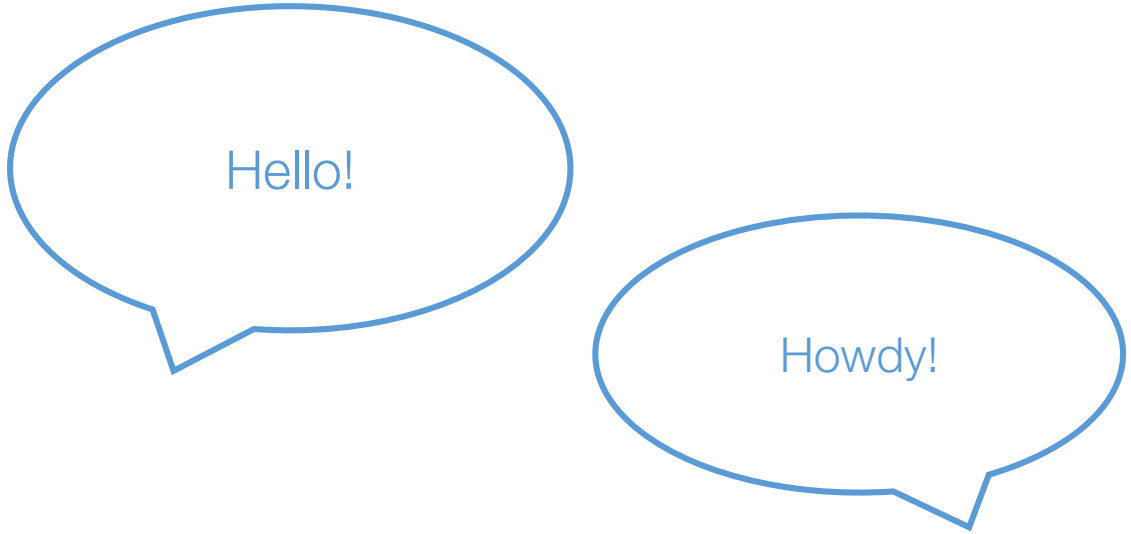
Text, speech, body language

Infrastructure:

Email, Slack, Skype

Characteristics:

Delayed (Email) / Immediate (Skype)



Microservices communication

Data type:

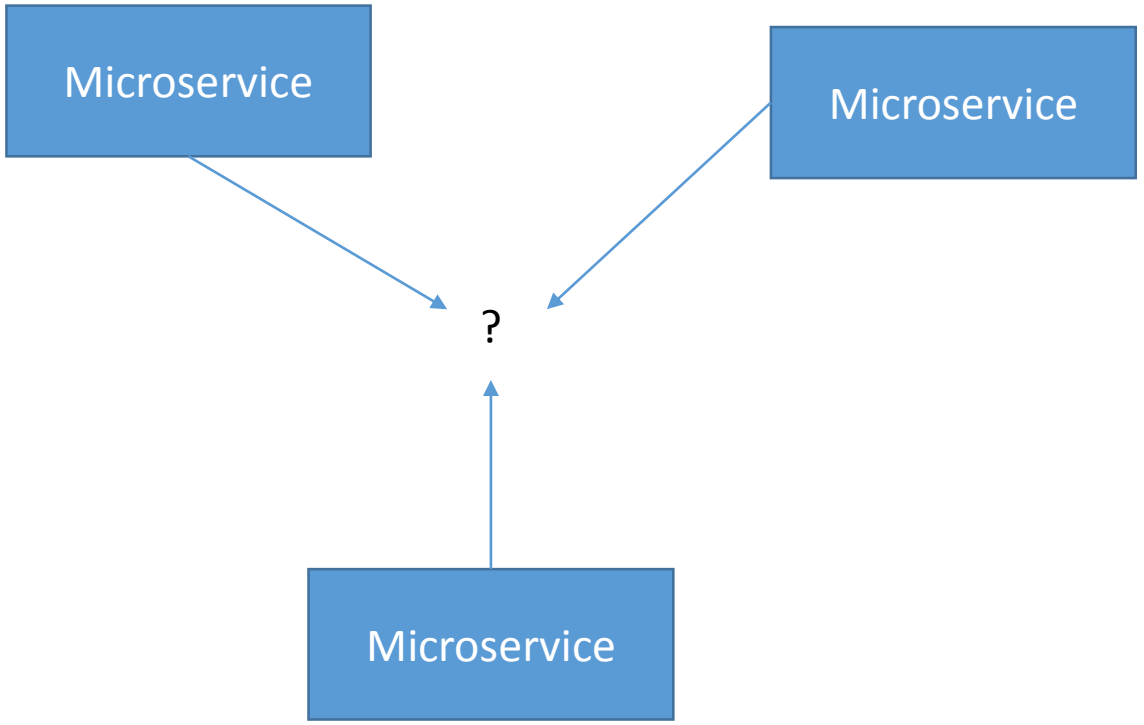
- JSON / XML / binary

Infrastructure:

- Service registry
- Service proxy

Characteristics:

- Asynchronous
- Synchronous



Communication infrastructure options

Service Registry

Store of available microservices

Provides:

Registration, Healthcheck, Service Discovery, De-registration

Consistency vs Availability

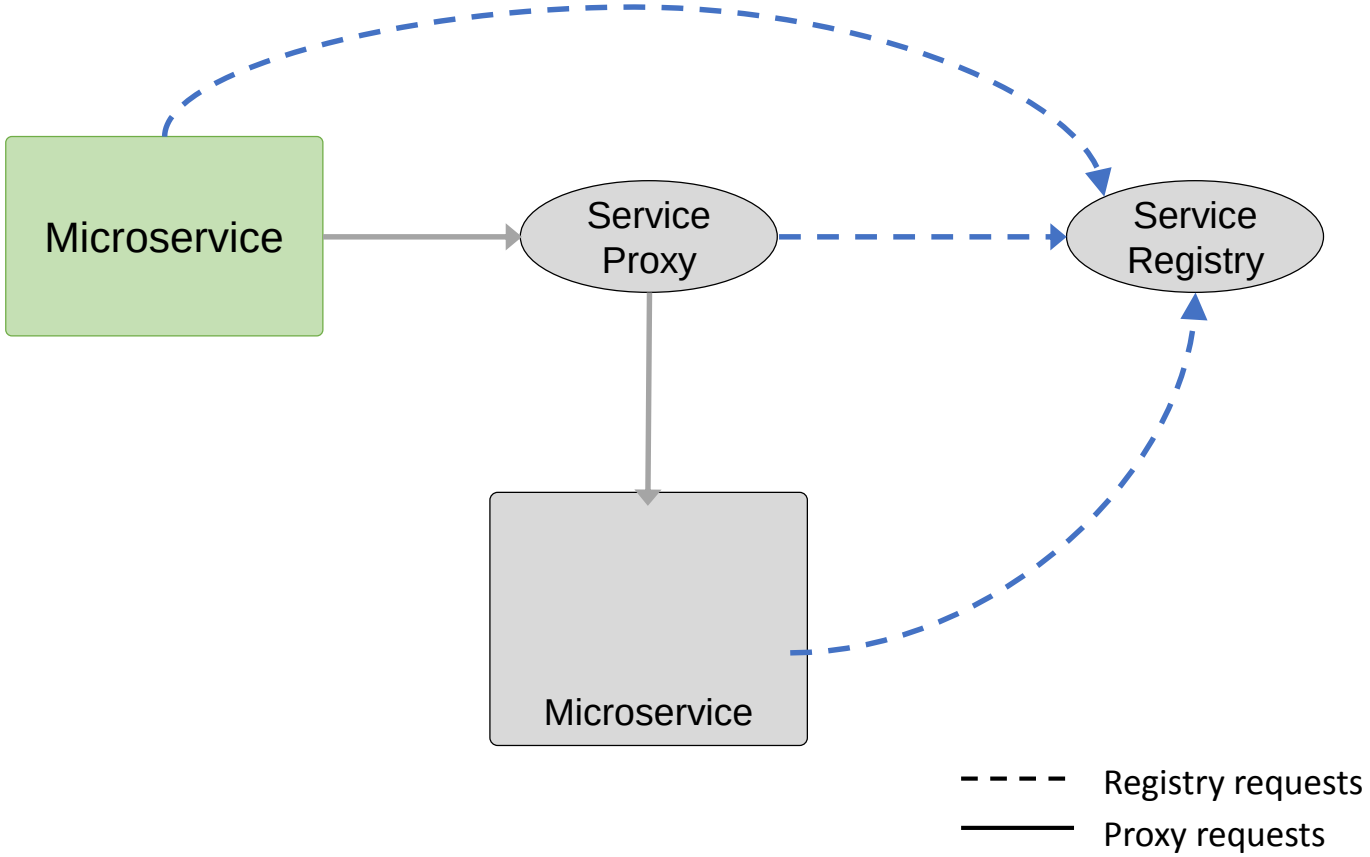
Examples:

Eureka, Consul, Amalgam8

Routing via service proxy

Advantages

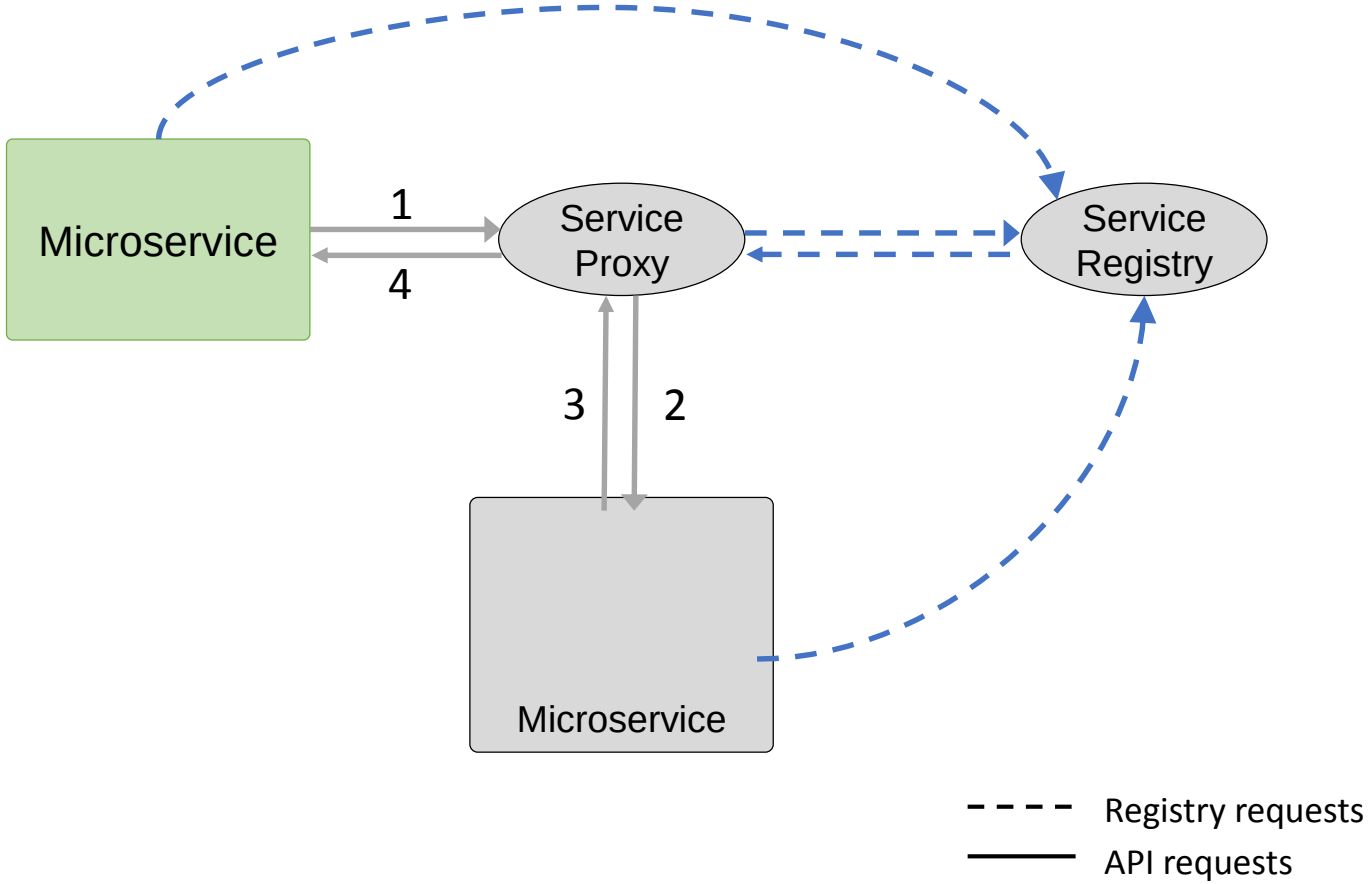
- Simple requests
- Simple tests



Routing via service proxy

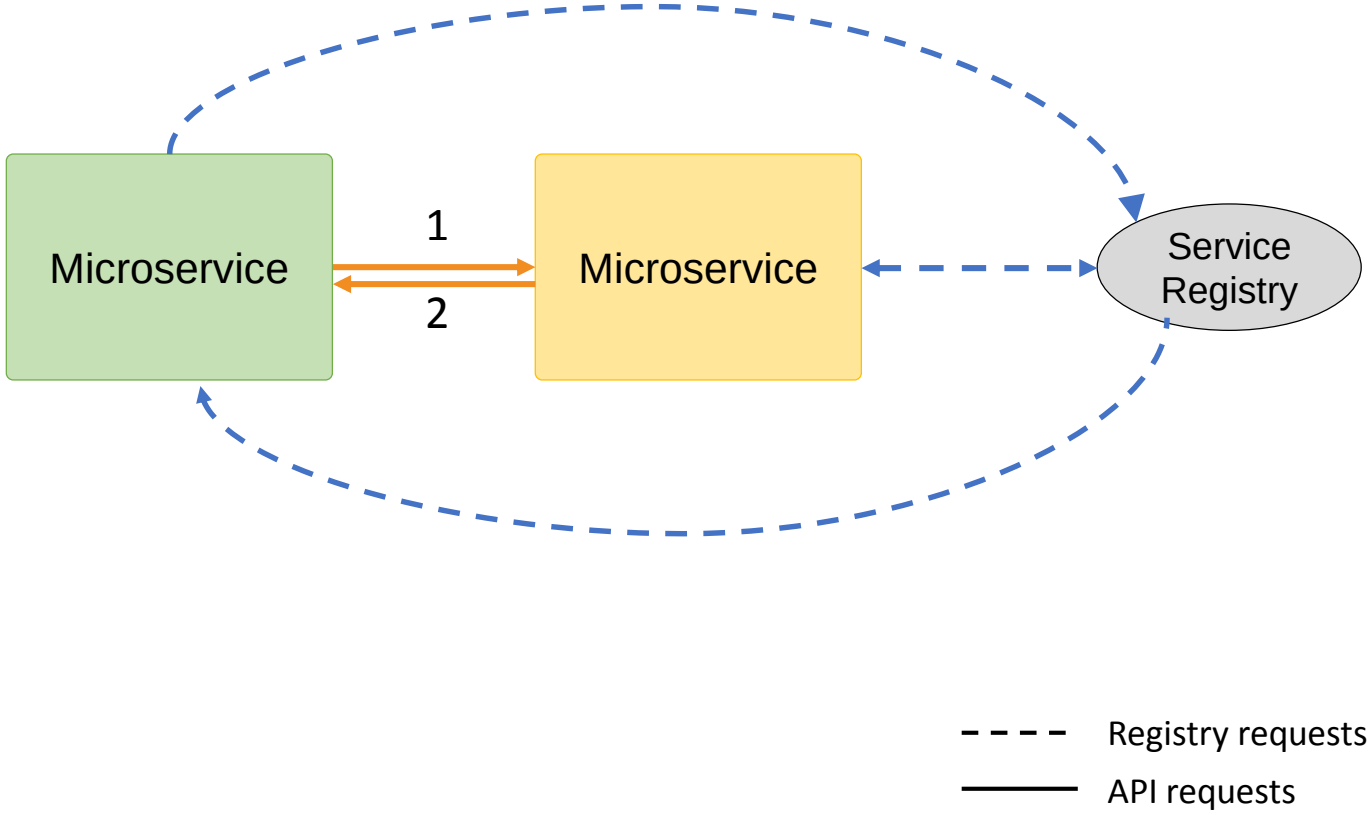
Disadvantage

More network hops



Client-side routing

Less network hops



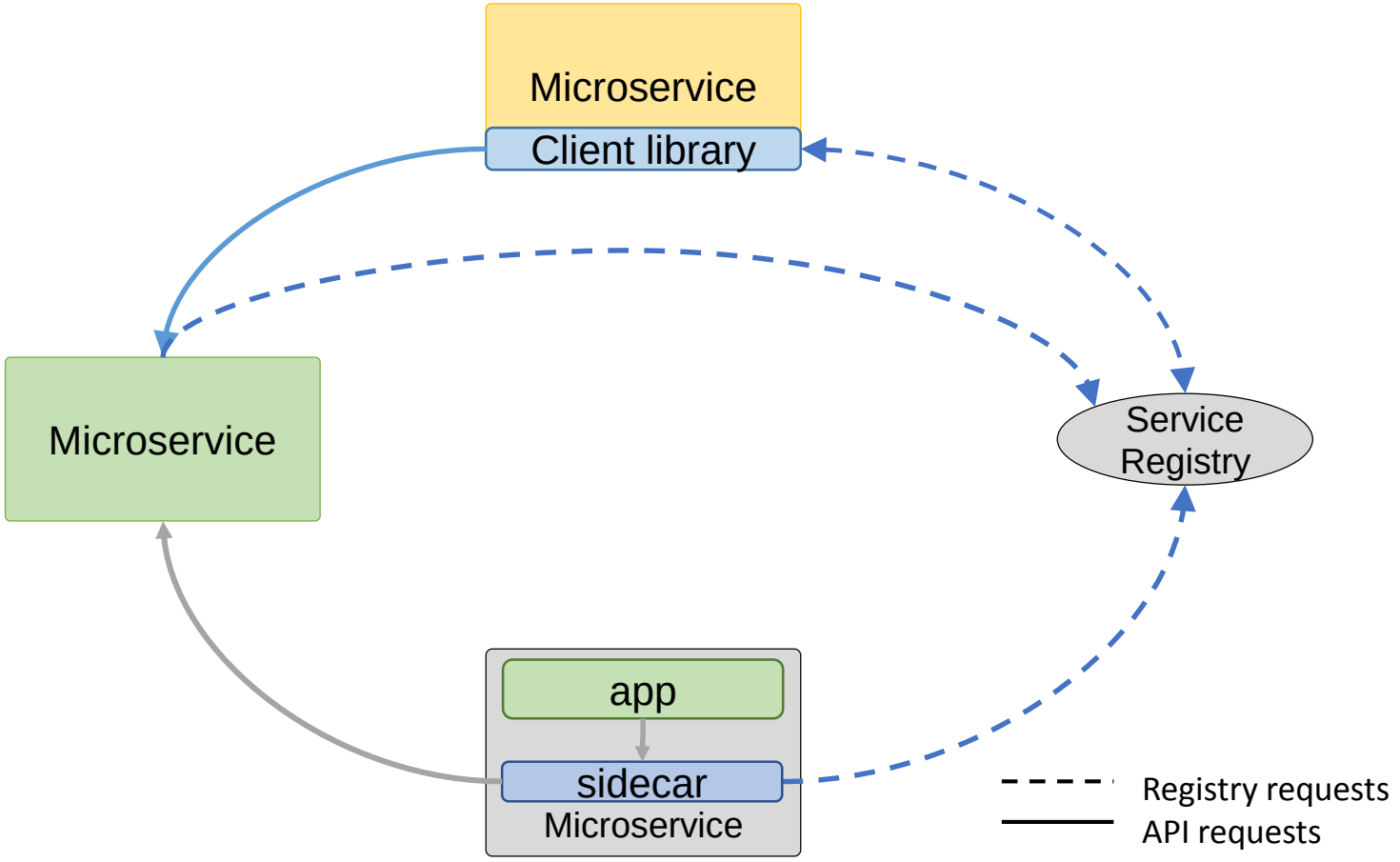
Client-side routing

Client library

- Language specific
- Easy debugging

Sidecar

- Language agnostic
- Harder to debug failures

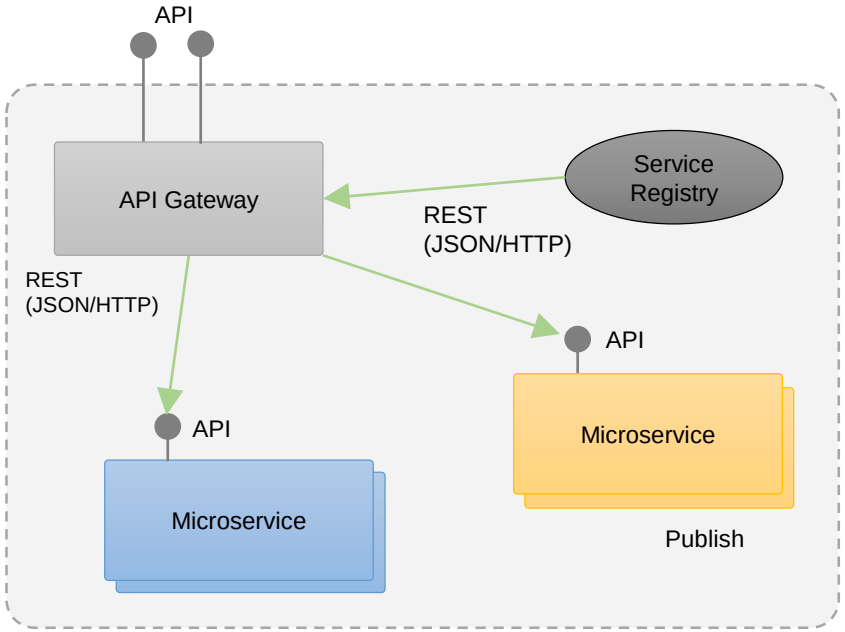


API Gateway

Provides coarse grained APIs

Useful for:

- API metering
- JWT injection



Microservices application

Communication characteristics

Synchronous vs asynchronous protocol

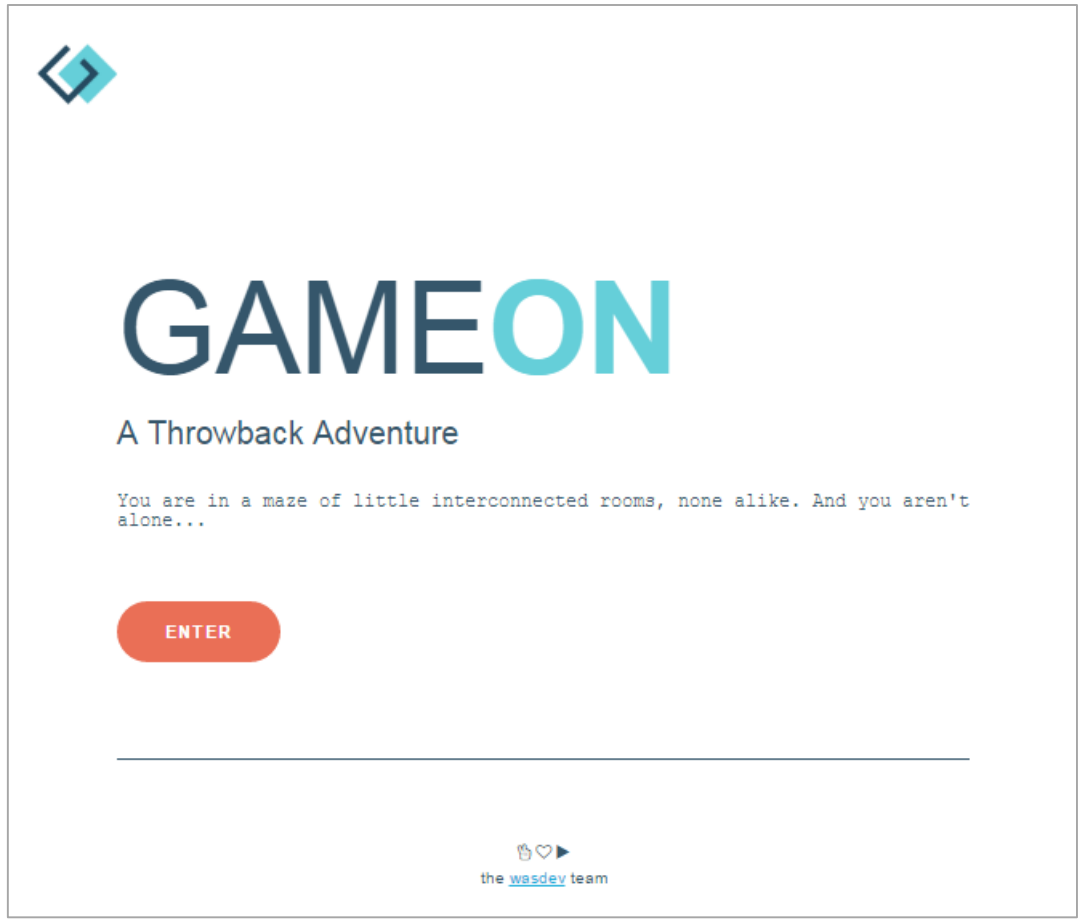
Synchronous e.g. REST, HTTP

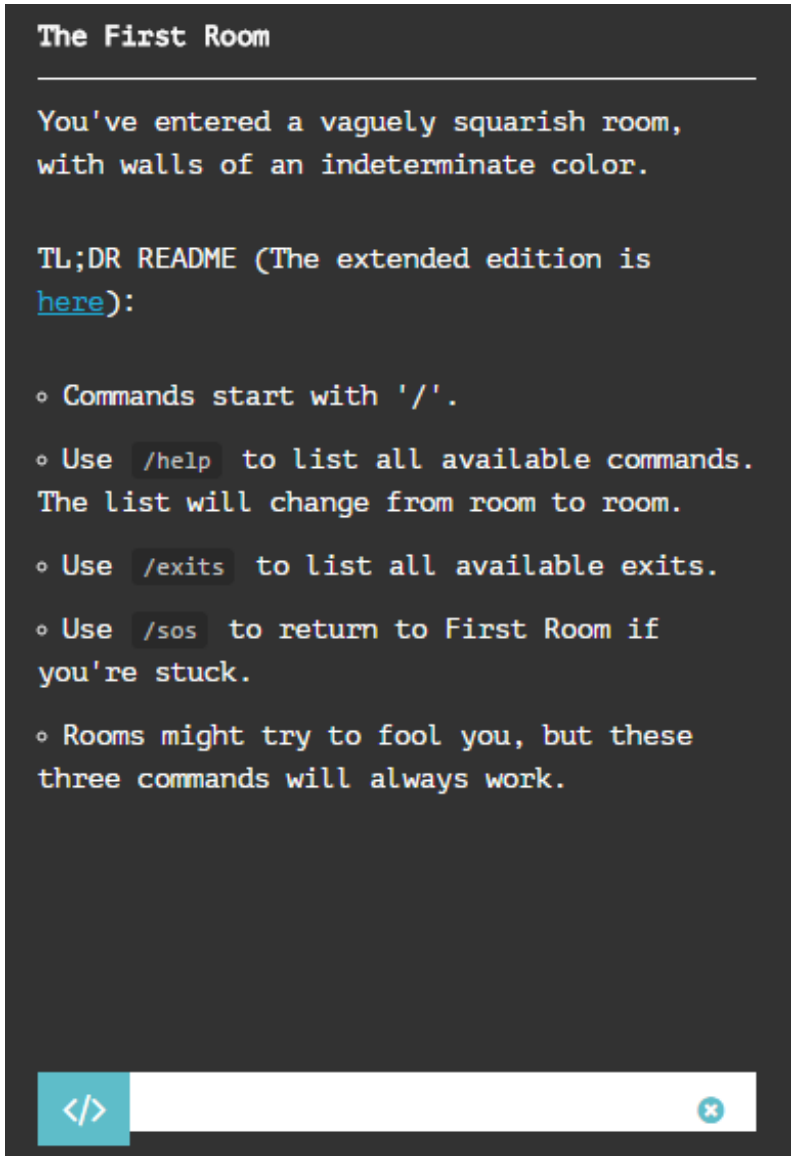
- Requires a response
- Utilise asynchronous programming model

Asynchronous e.g. messaging

- No response required

Example – Game On



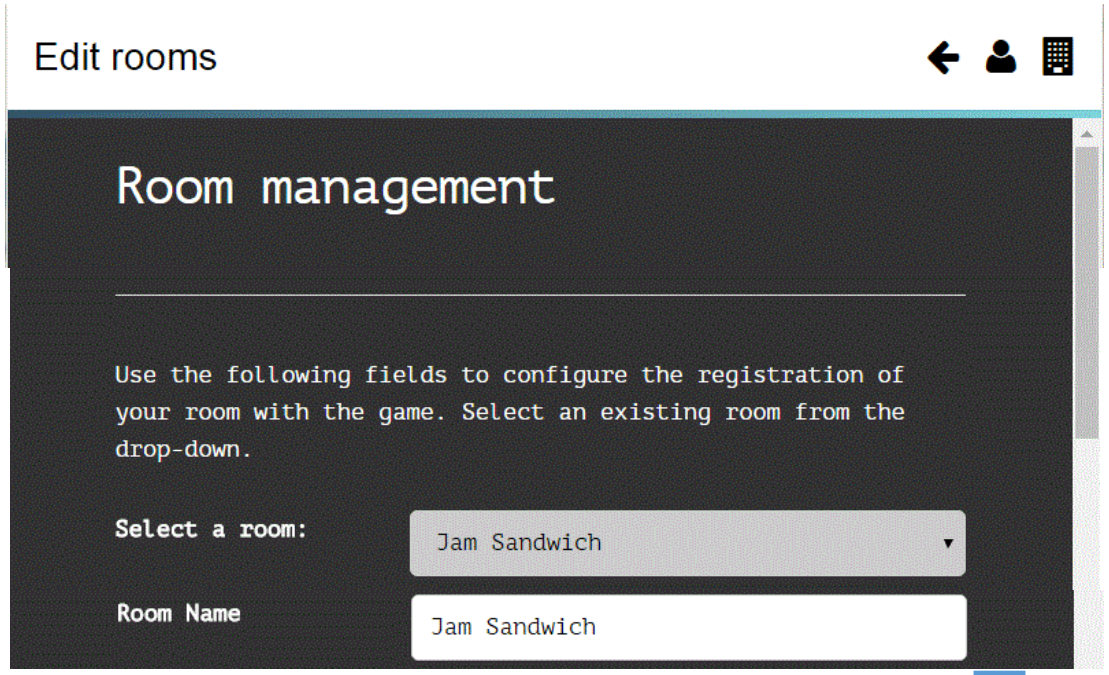


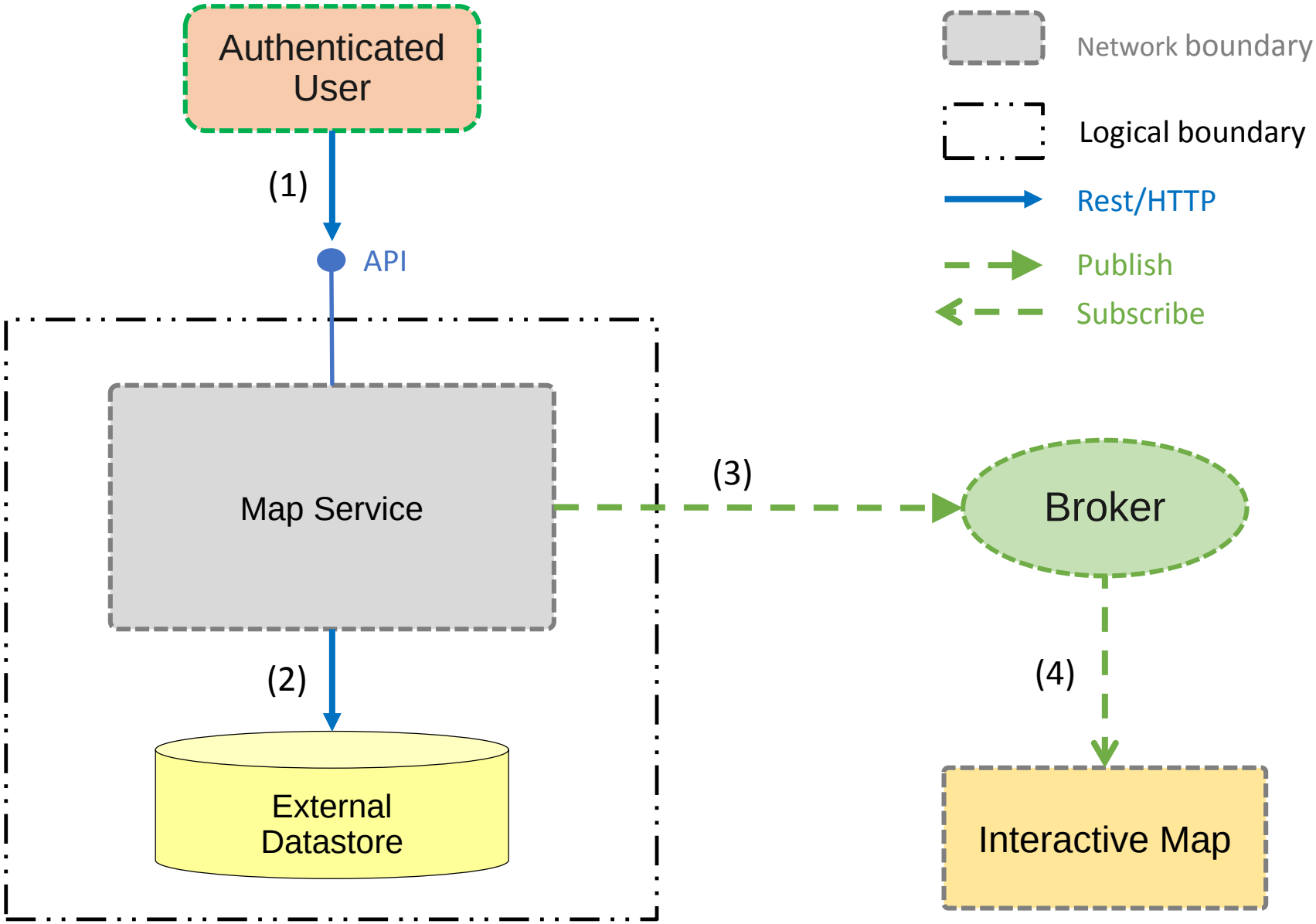
Example – Game On!

Interactive map

- Outside of core microservices
- Displays room location and name

Example – Game On




Authenticated User
(1)
API
Map Service
(2)
External Datastore
(3)
Broker
(4)
Interactive Map
Network boundary
Logical boundary
Rest/HTTP
Publish
Subscribe

Messaging in Game On!

Apache Kafka locally

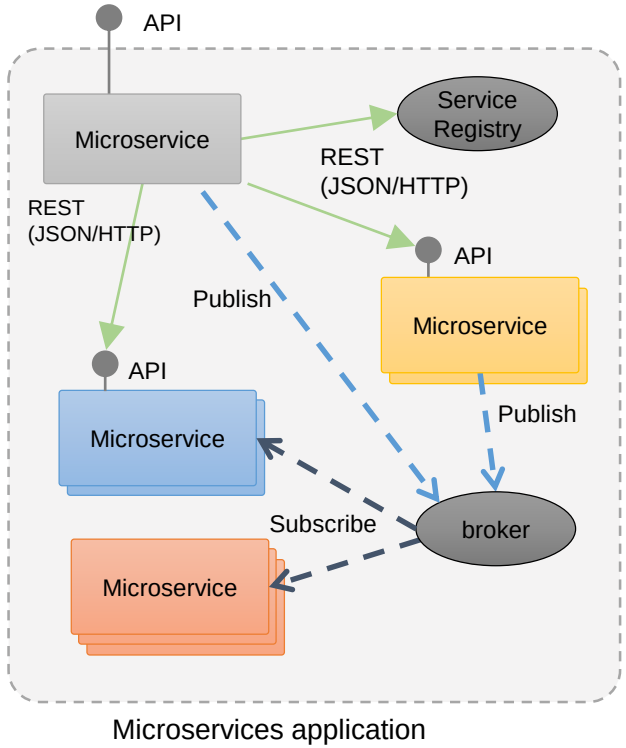
MessageHub in the cloud

```
111     public void publishMessage(String topic, String key, String message){
112       if(producer!=null){
113           Log.log(Level.FINER, this, "Publishing Event {0} {1} {2}",topic,key,message);
114           ProducerRecord<String,String> pr = new ProducerRecord<String,String>(topic, key, message);
115           producer.send(pr);
116           Log.log(Level.FINER, this, "Published Event");
117       }else{
118           Log.log(Level.FINER, this, "Kafka Unavailable, ignoring event {0} {1} {2}",topic,key,message);
119       }
120     }
```

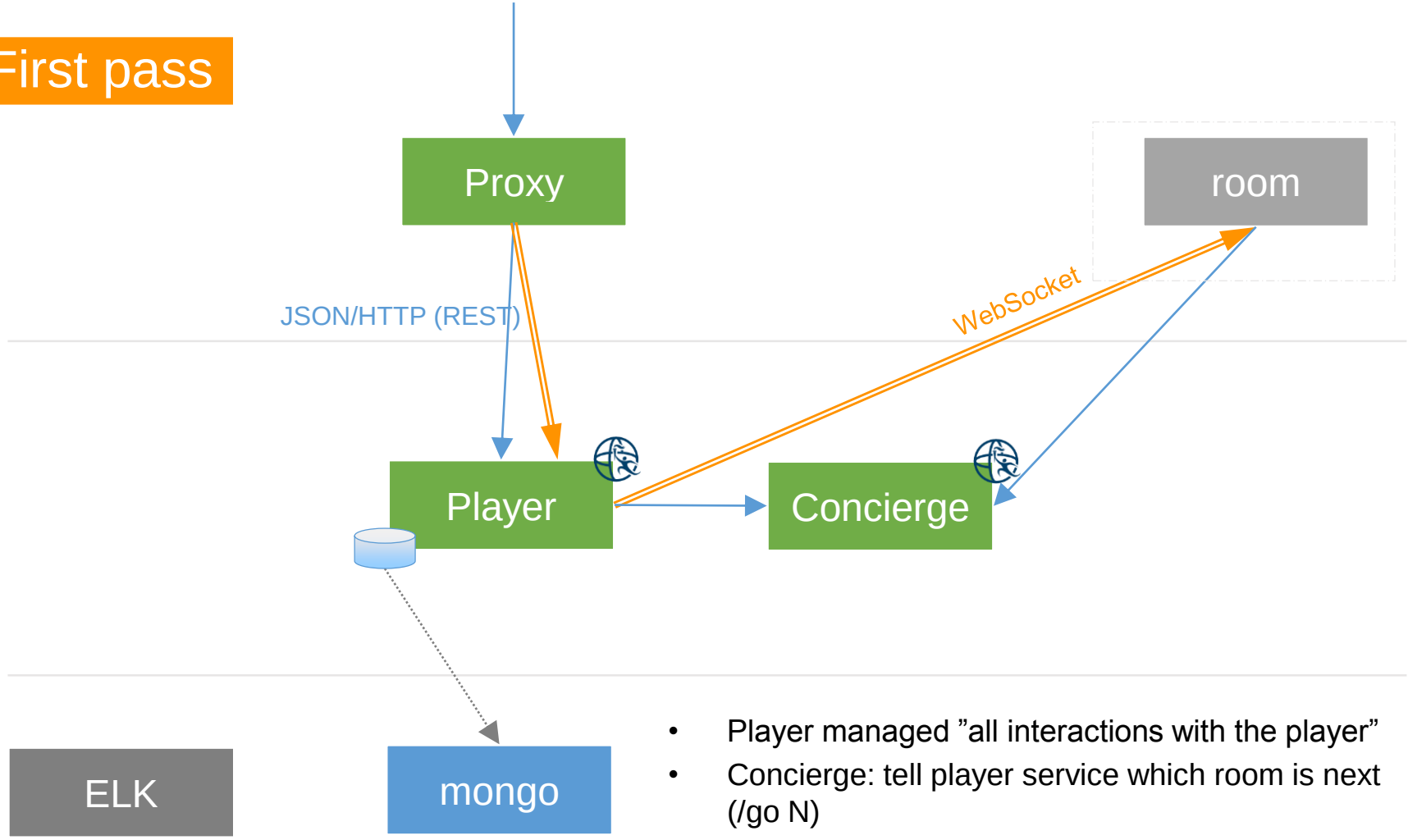
Microservice design

Design influenced by communication

Consider latency

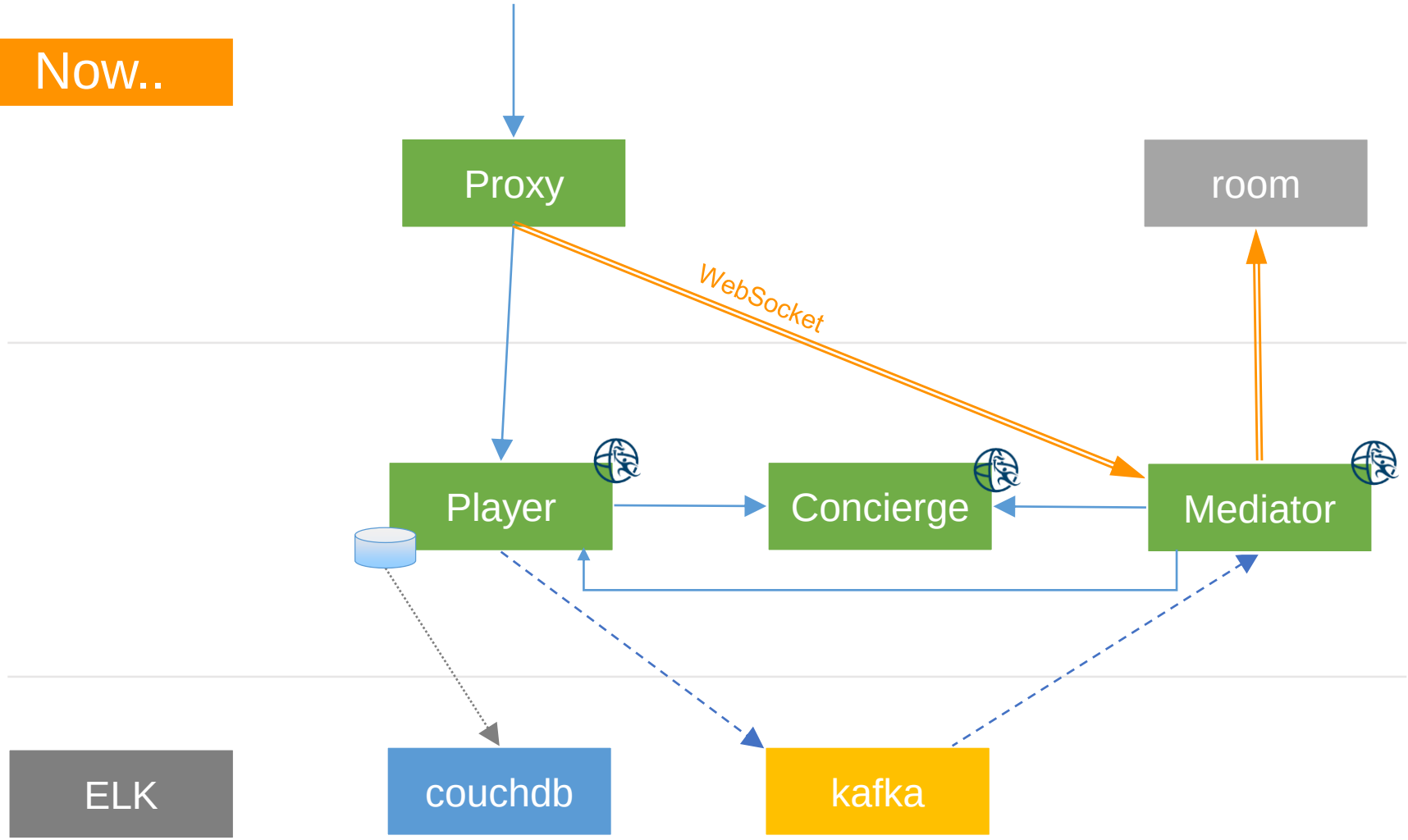


First pass



- Player managed ”all interactions with the player”
- Concierge: tell player service which room is next (/go N)
- Room (had to have one!)

Now..


Proxy
room
WebSocket
Player
Concierge
Mediator
ELK
couchdb
kafka

Producing and consuming APIs

APIs change!

Breaking changes

- Alter methods and data

Non-breaking changes (theoretically!)

- Add data or methods

Being a good consumer

Accept the unknown

```
26  @JsonInclude(Include.NON_EMPTY)
27  @JsonIgnoreProperties(ignoreUnknown = true)
28  public class RoomInfo {
29
30      /** name of room (short / url-friendly) */
31      private String name;
```

Only validate against the required

Fault tolerance important – another talk!

Being a good producer

Golden rule: don’t break your consumers

If you need to break something - use versioning

Accept unknown attributes

Only return attributes relevant to the request

Robustness principle

“Be conservative in what you send,
be liberal in what you accept”

https://en.wikipedia.org/wiki/Robustness_principle

Documenting APIs

Consumer driven contract tests

Swagger

Conclusion

Conclusion

Choose service discovery and invocation method

Use both asynchronous and synchronous protocols

Design influenced by communication

Robustness principle

Thank you!

Play the game – http://gameontext.org

Build rooms – http://github.com/gameontext

Learn more:

http://wasdev.net

Microservices Best Practices for Java
http://www.redbooks.ibm.com

Kate Stanley | katheris@uk.ibm.com | @KateStanley91