

Microservices communication and integration: what are my options?

Kate Stanley
@KateStanley91

Jfokus – February 2016

Introduction

Software engineer

WebSphere Liberty 

Microservices



@KateStanley91

Contents

What are microservices?

Why is communication important?

Microservice communication options

Producing and consuming APIs

Conclusion

What are microservices?

Microservices are used to...

...compose a complex application using:

“small”

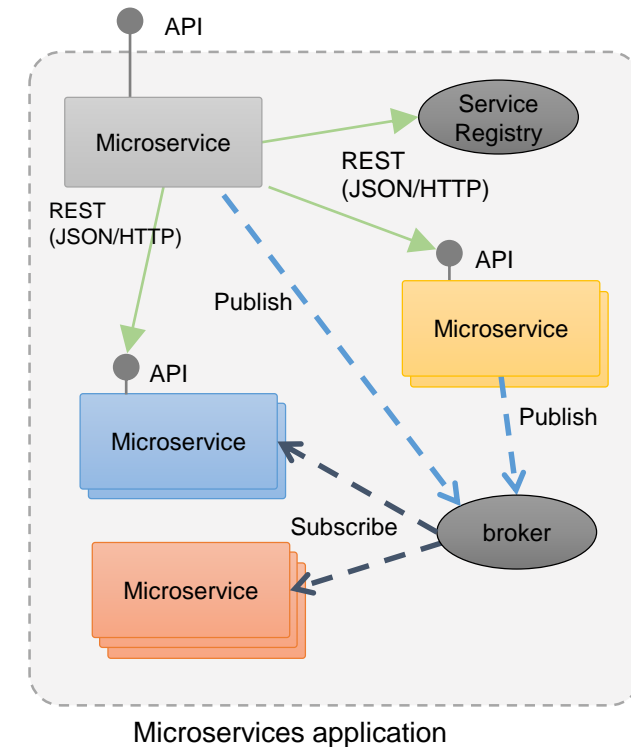
independent (autonomous)

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:

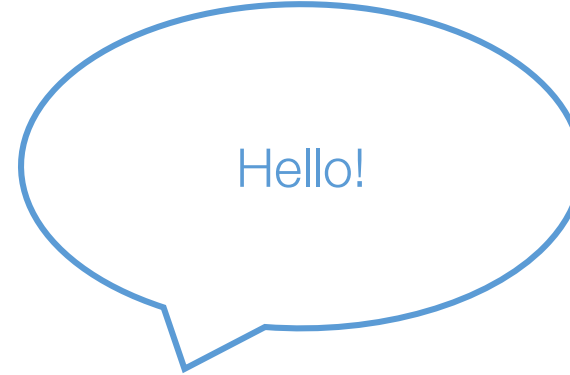
Text, speech, body language

Infrastructure:

Email, Slack, Skype

Characteristics:

Delayed (Email) / Immediate (Skype)



Microservices communication

Data type:

JSON / XML / binary

Infrastructure:

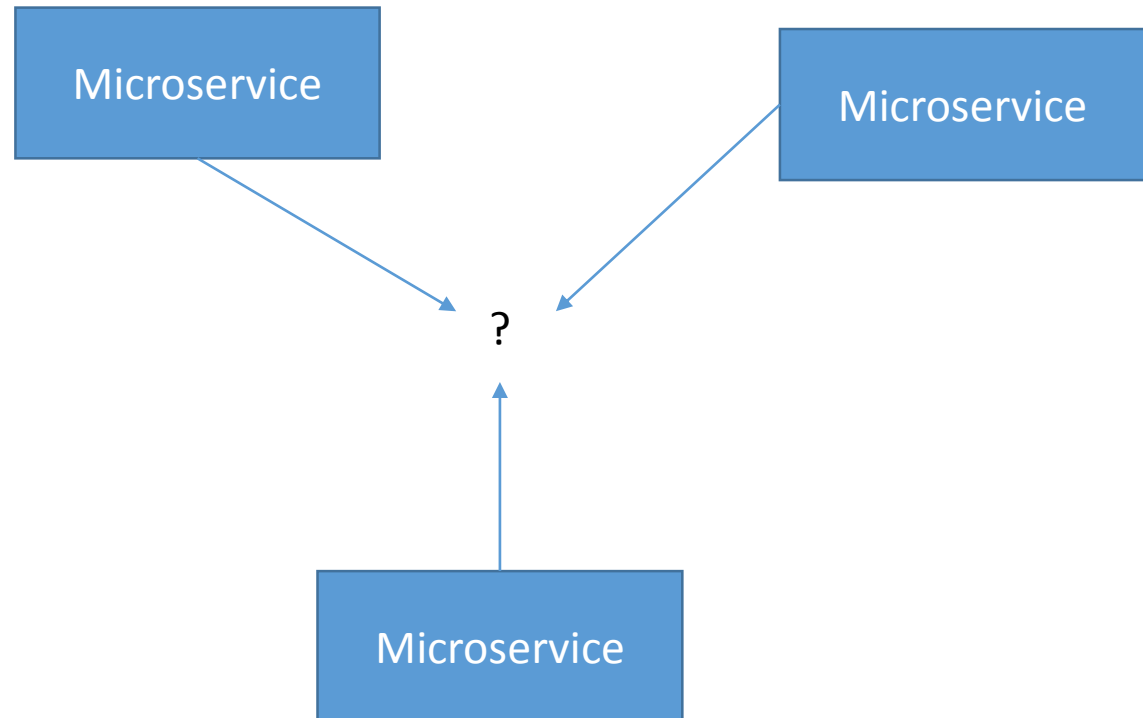
Service registry

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:

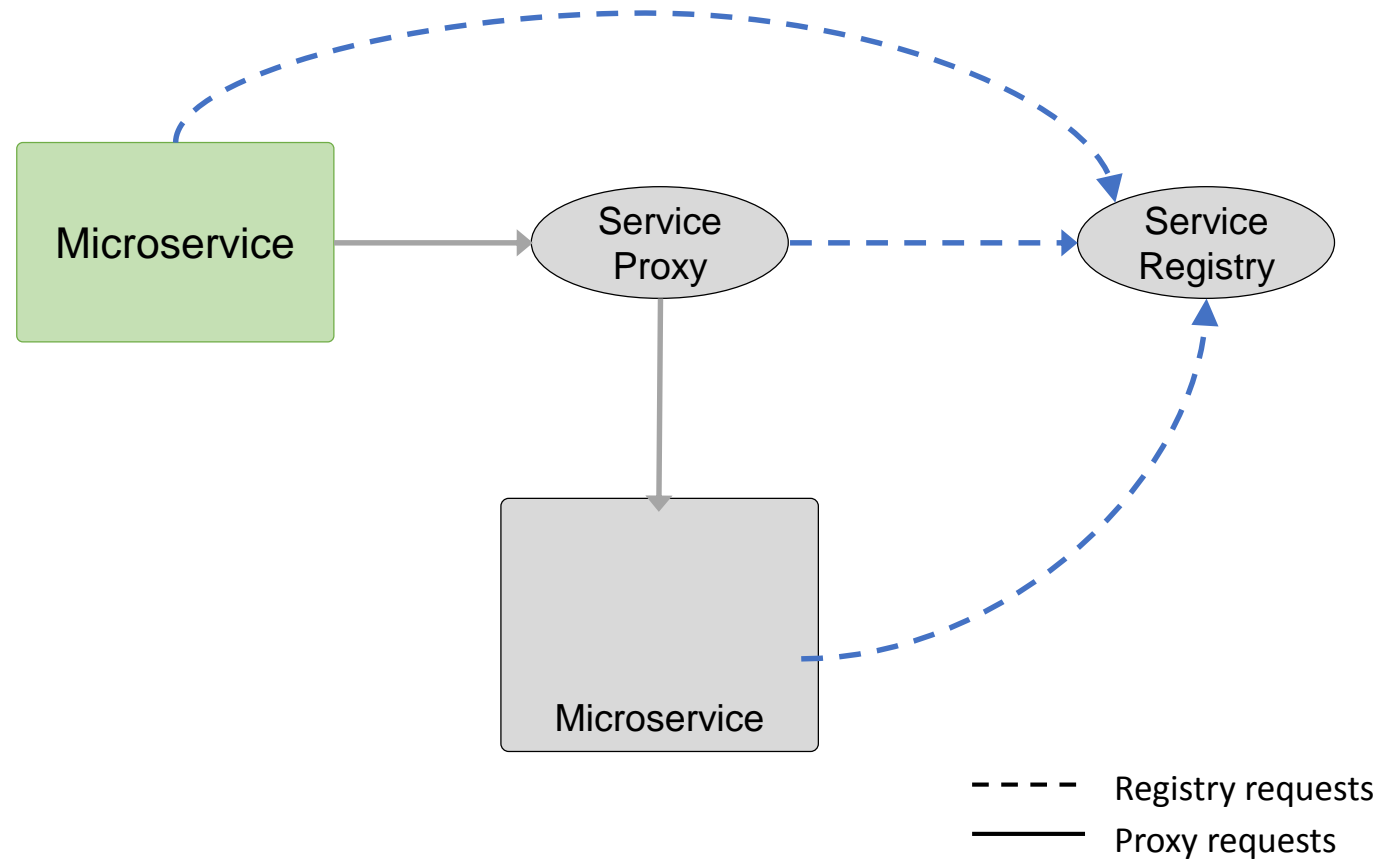
Eureka, Consul, Amalgam8

Routing via service proxy

Advantages

Simple requests

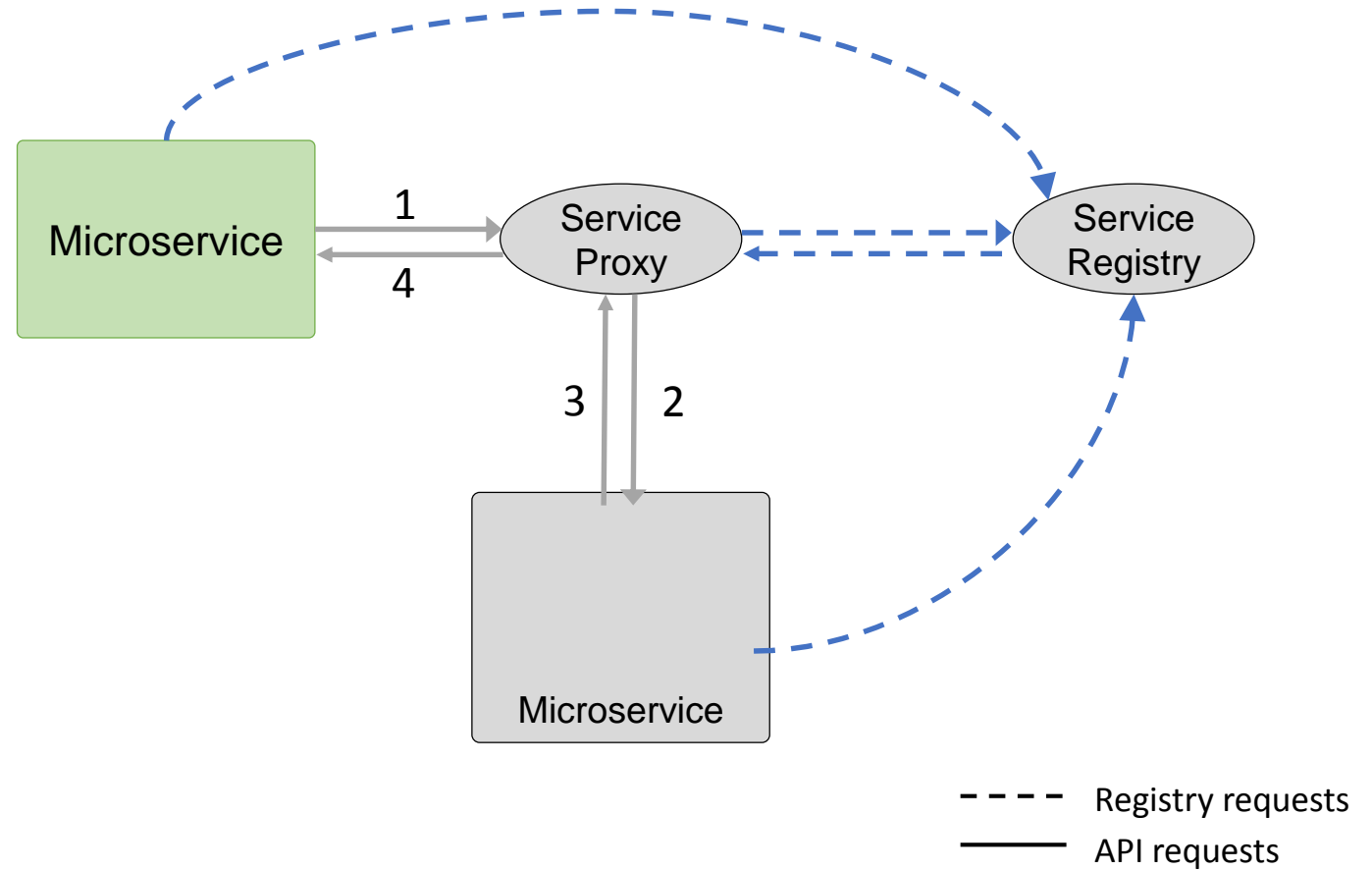
Simple tests



Routing via service proxy

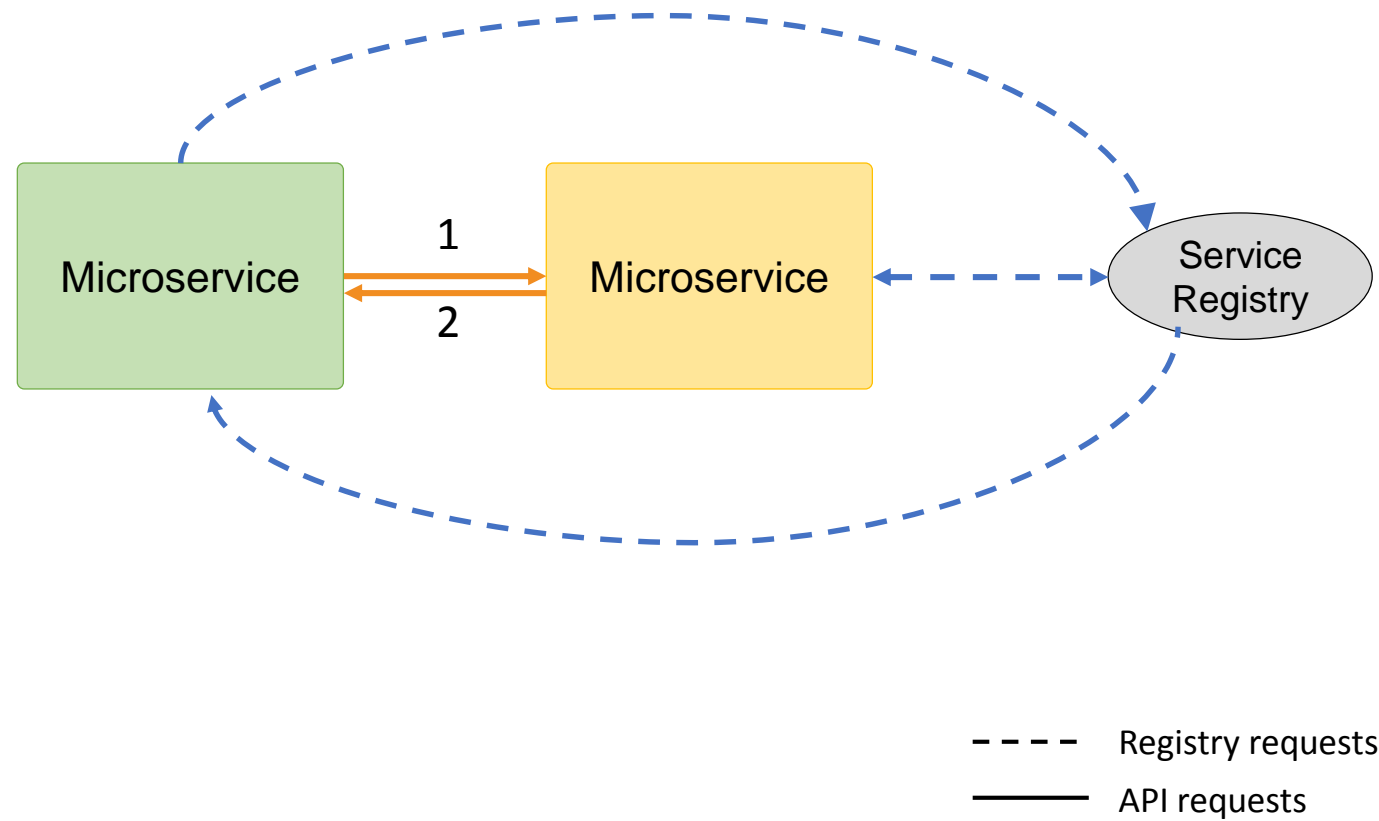
Disadvantage

More network hops



Client-side routing

Less network hops



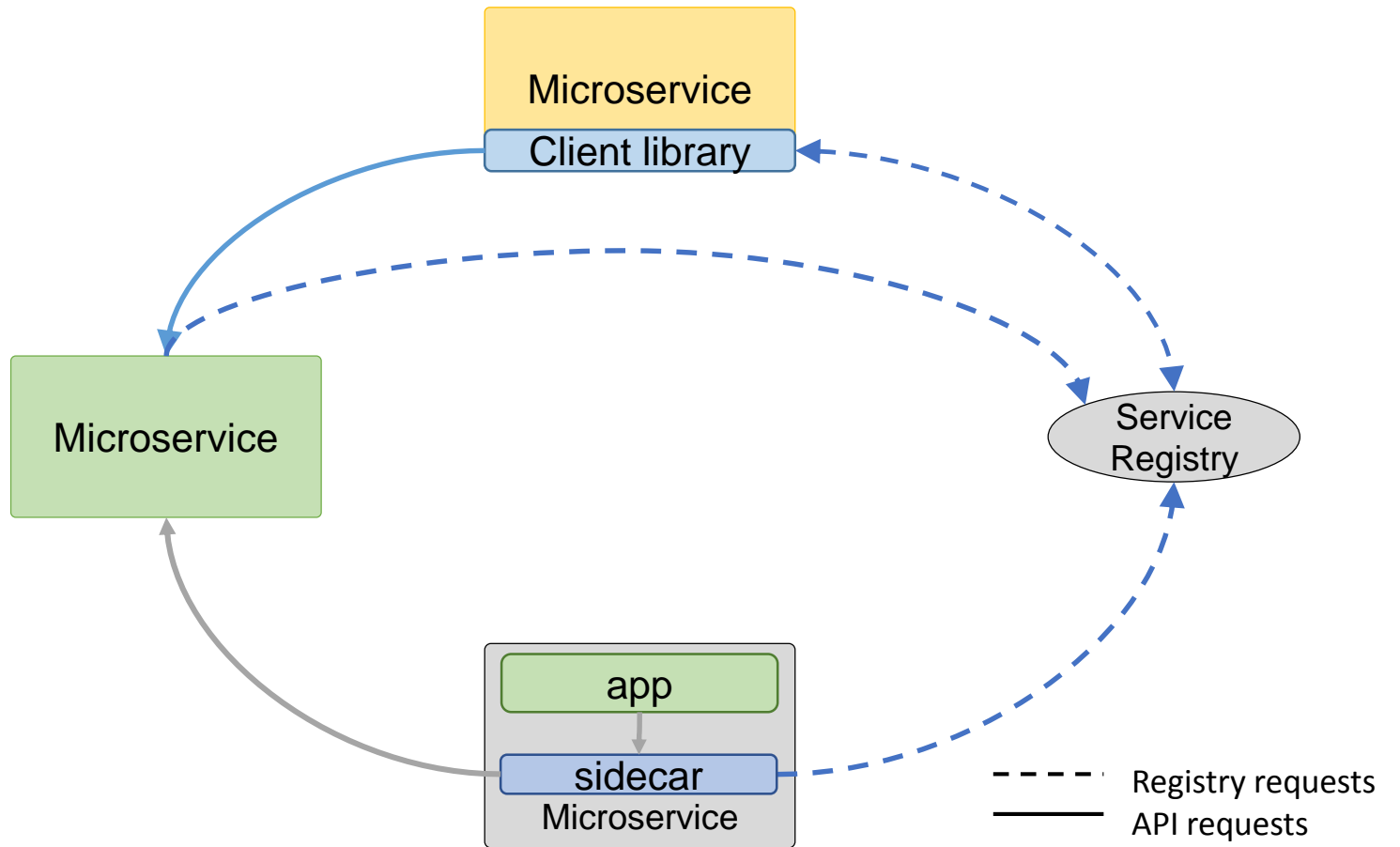
Client-side routing

Client library

Language specific
Easy debugging

Sidcar

Language agnostic
Harder to debug failures



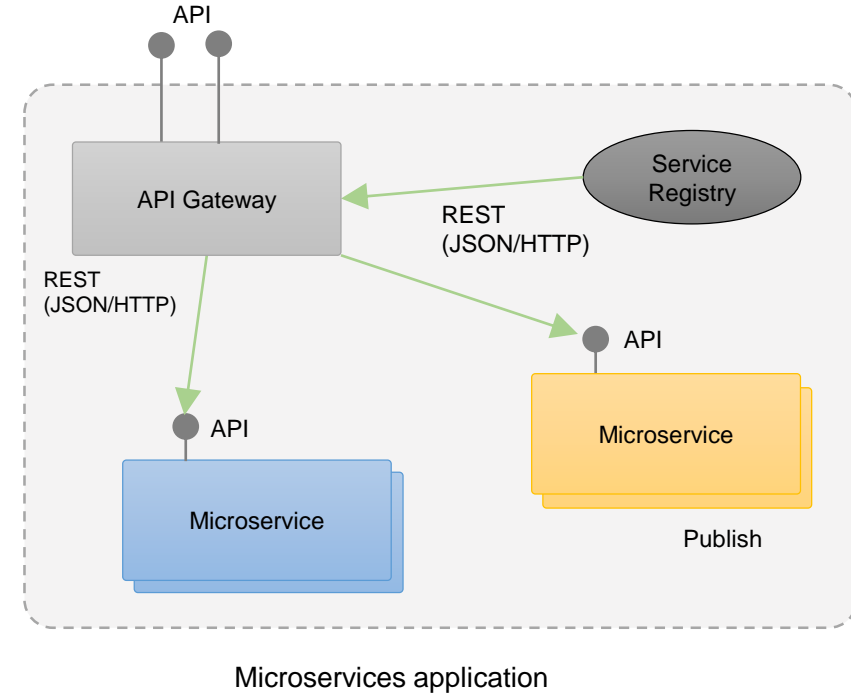
API Gateway

Provides coarse grained APIs

Useful for:

API metering

JWT injection



Communication characteristics

Synchronous vs asynchronous protocol

Synchronous e.g. REST, HTTP

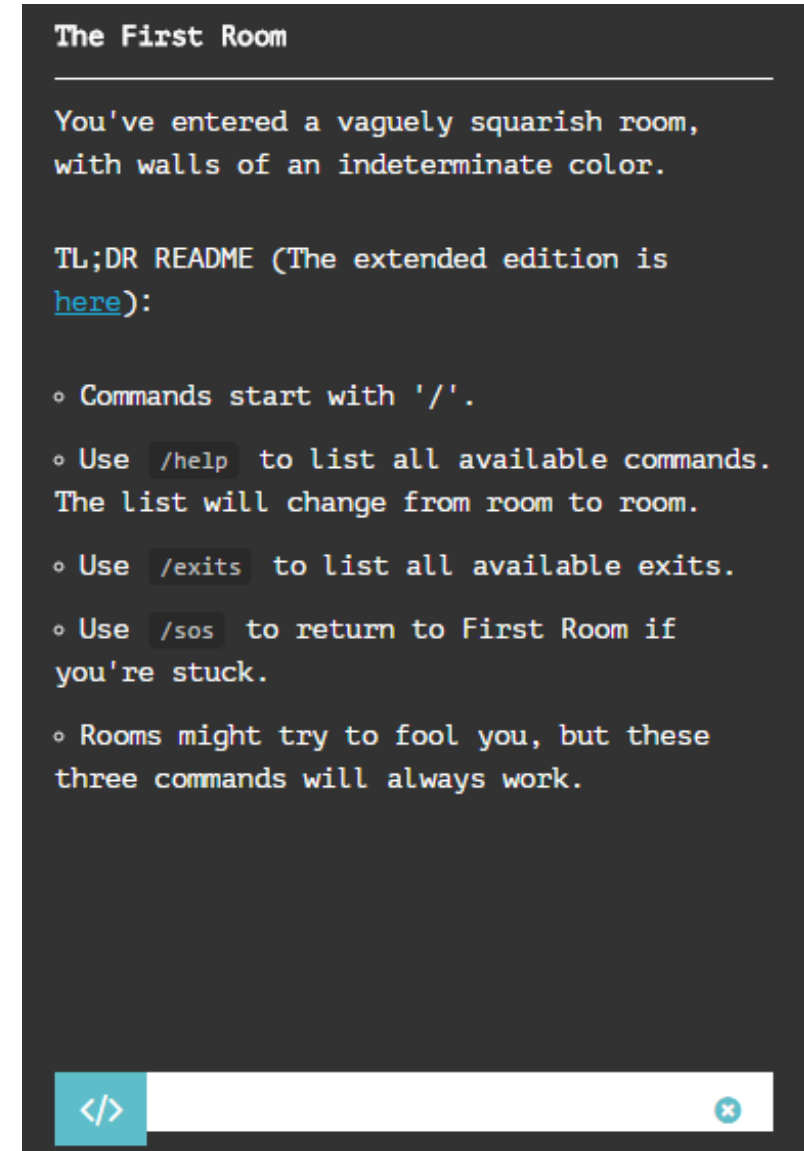
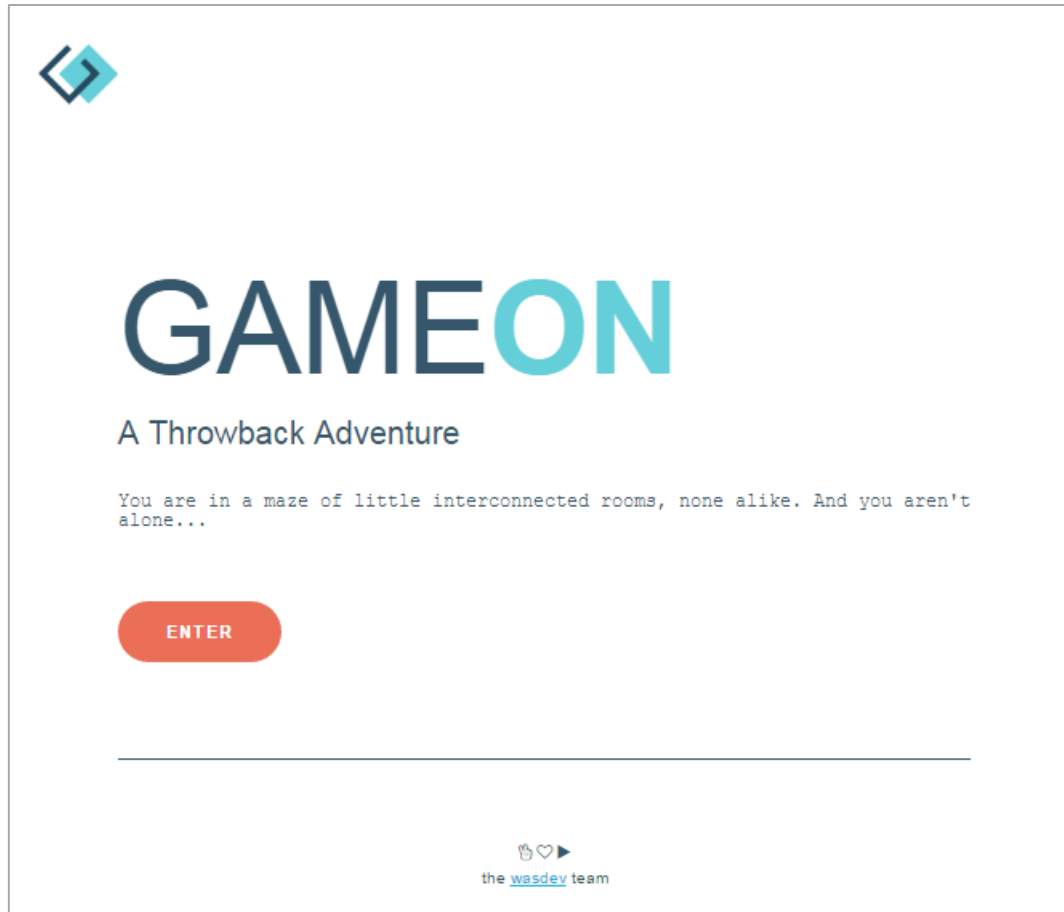
- Requires a response

- 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



The screenshot shows the 'GameOn! Map' interface. It features a grid of rooms, each with a name and a color-coded background. A legend on the left side of the grid allows users to filter the map by 'Points of Interest' (Rooms, My Rooms, Simple Rooms) and by map type (Game On! Map, Sweep Map). The grid is organized into rows and columns, with room names displayed in a monospaced font. The interface also includes a search bar and a menu icon in the top right corner, and a zoom control in the bottom right corner.

GameOn! Map								
		mike_pub	VeryAbsur	AnotherSi	AnotherSi	Bootcamp	RatpackRo	AnotherSi
		CalypsoRo	TomsRoom	IainsSimp	AnotherSi	invisible	AnotherSi	AnkUnique
AnotherSi	AnotherSi	roncastel	IoTRoom	davichbroo	creepyroo	CowRoom	missing	closet
A Pretty	Rock n Ro	Rajroom	Basement	First Roo	RecRoom	room14	aca_room	AnotherSi
Adams Roo	JavaOneHC	MugRoom	zyclone	GOTest1	REAL	Creepy Ro	Jam Sandw	AnotherSi
(missing)	BakeBit	CarRoom	AnotherSi	CleanLigh	marcoshri	testashok	bookRoom	TheNodeRo
Hello Wor	asmita-te	missing	Patrocini	wine cell	wonderlan	AnotherSi	HelloWorl	(missing)

Example – Game On

Edit rooms

Room management

Use the following fields to configure the registration of your room with the game. Select an existing room from the drop-down.

Select a room:

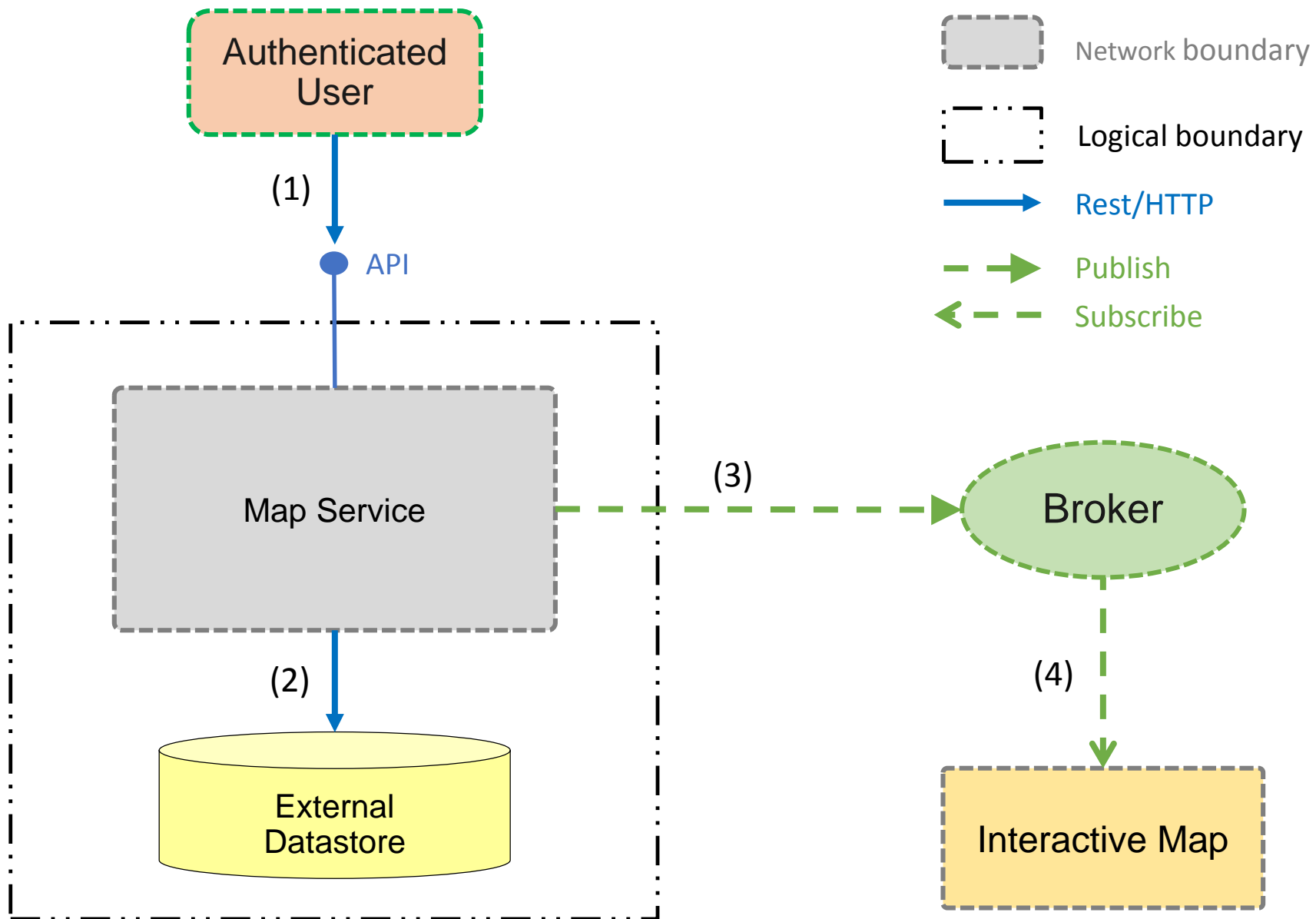
Jam Sandwich

Room Name

Jam Sandwich

GameOn! Map								
<input type="radio"/> Game On! Map	mike_pub	VeryAbsur	AnotherSi	AnotherSi	Bootcamp	RatpackRo	AnotherSi	
<input checked="" type="radio"/> Sweep Map								
Points of Interest								
<input type="checkbox"/> Rooms	CalypsoRo	TomsRoom	IainsSimp	AnotherSi	invisible	AnotherSi	AnkUnique	
<input type="checkbox"/> My Rooms								
<input type="checkbox"/> Simple Rooms								
AnotherSi	AnotherSi	roncastel	IoTRoom	davicbroo	creepyroo	CowRoom	missing	closet
A Pretty	Rock n Ro	Rajroom	Basement	First Roo	RecRoom	room14	aca_room	AnotherSi
Adams Roo	JavaOneHC	MugRoom	zyclone	GOTest1	REAL	Creepy Ro	Jam Sandw	AnotherSi
(missing)	BakeBit	CarRoom	AnotherSi	CleanLigh	marcoshri	testashok	bookRoom	TheNodeRo
Hello Wor	asmita-te	missing	Patrocini	wine cell	wonderlan	AnotherSi	HelloWorl	(missing)





Messaging in Game On!

Apache Kafka locally

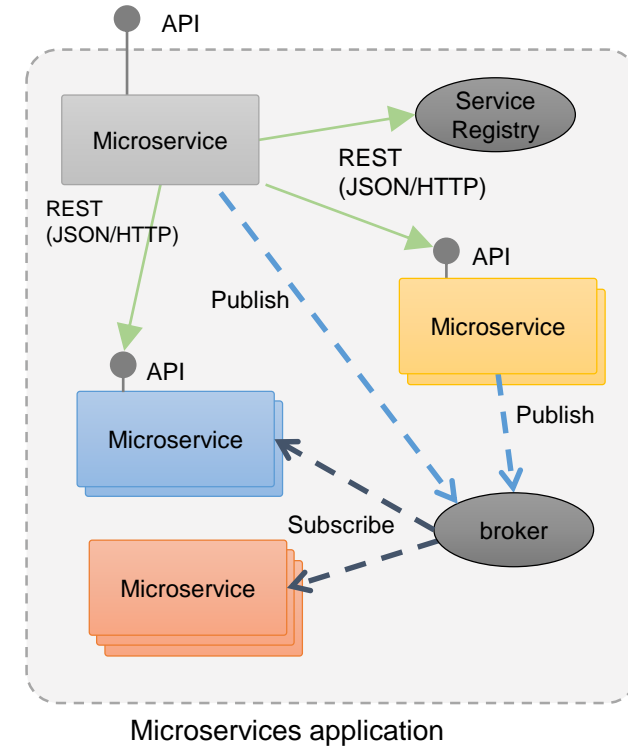
MessageHub in the cloud

```
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     }
```

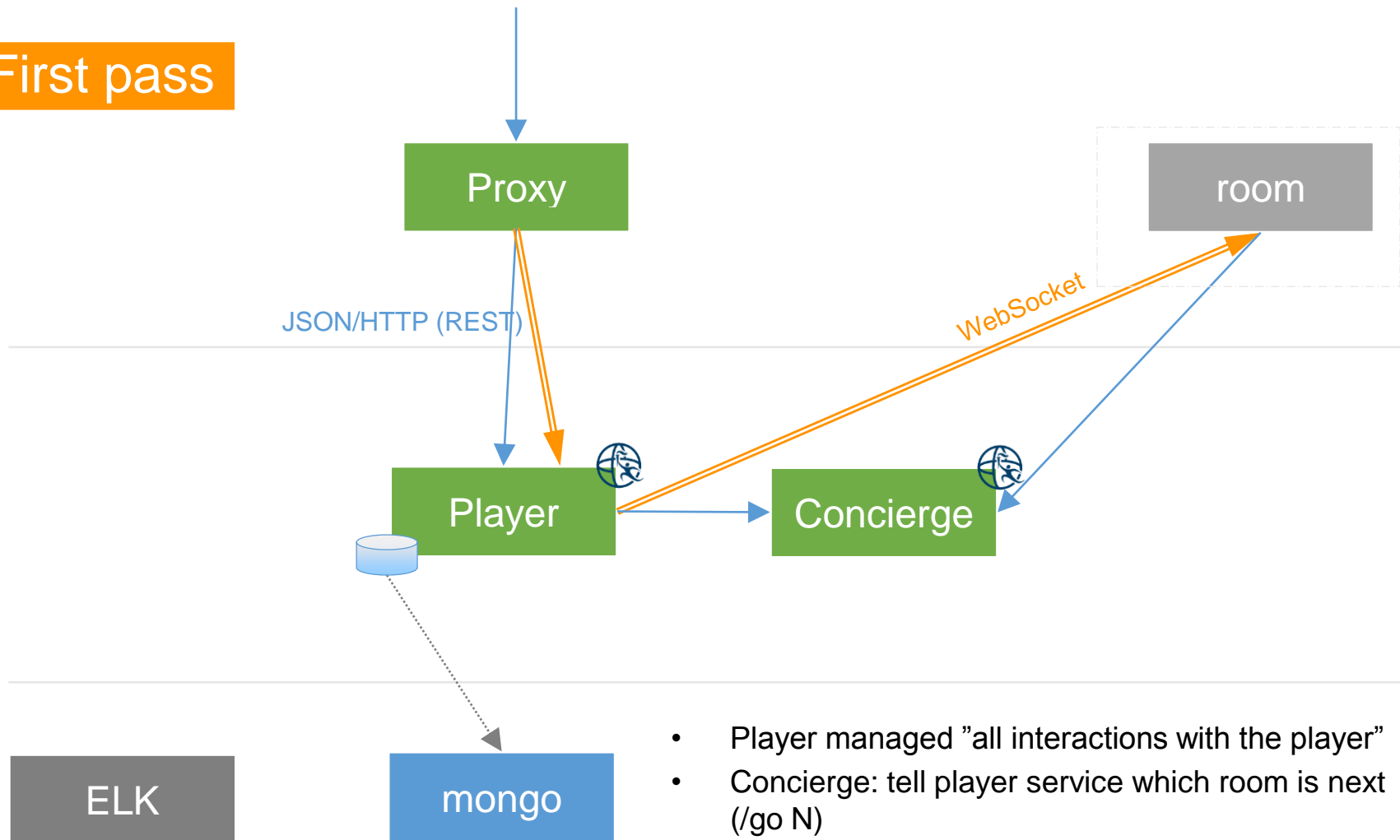
Microservice design

Design influenced by communication

Consider latency

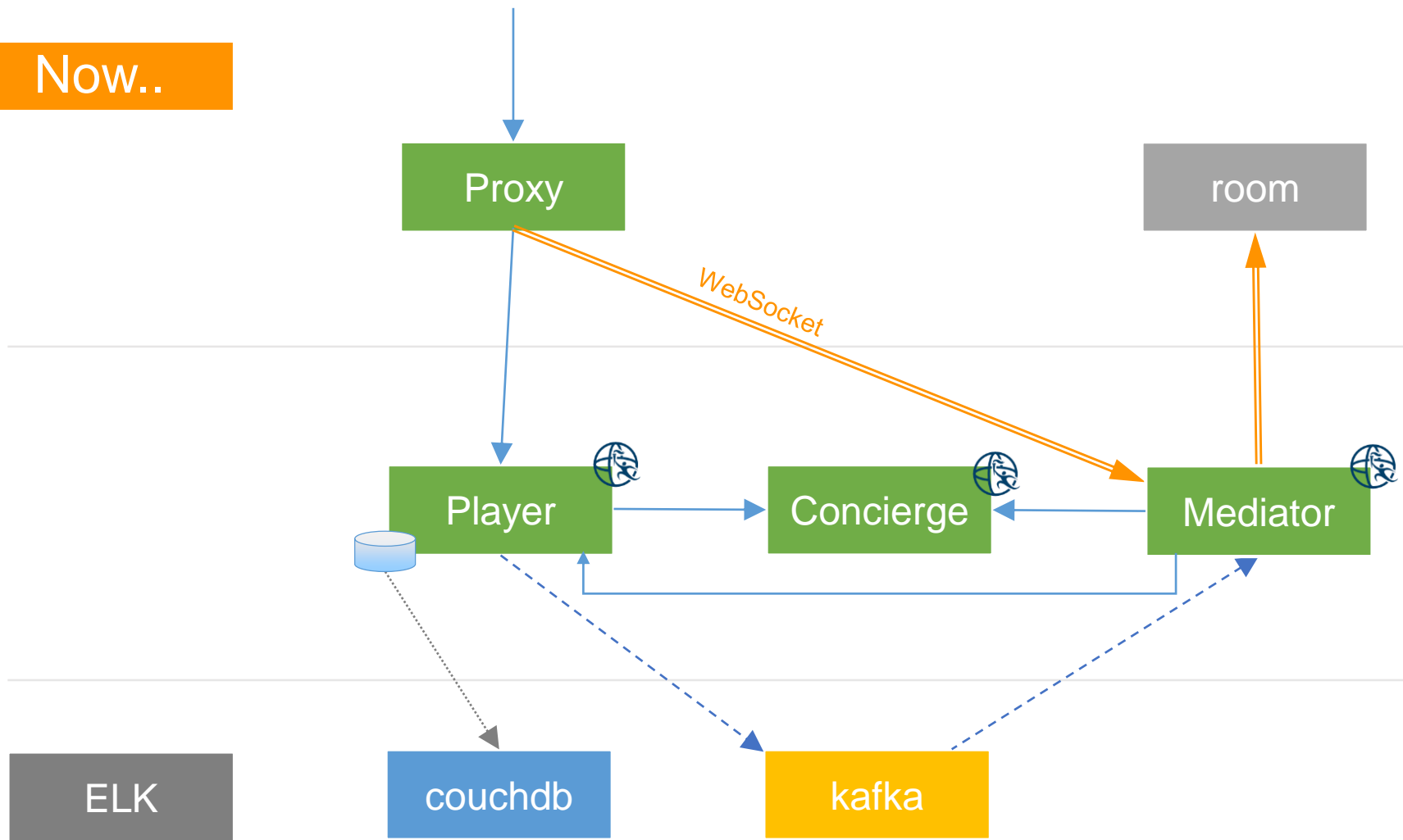


First pass



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

Now..



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”

Documenting APIs

Consumer driven contract tests

Swagger



The image shows the Swagger UI for the 'Game On' API. The top bar is green and contains the Swagger logo, the API URL 'https://gameontext.org/swagger/swagger.json', an 'api_key' input field, and an 'Explore' button. Below the bar, the title 'Game On' is displayed, followed by a description: 'A throwback text application. Check [Game-On.org](https://gameontext.org) for more info!'. A section titled 'map' lists four API endpoints. Each endpoint is represented by a colored bar (blue for GET, green for POST) with the HTTP method, the path, and a description. The endpoints are: GET /map/v1/, GET /map/v1/health (Check application health), GET /map/v1/sites (List sites), and POST /map/v1/sites (Create a room). To the right of the 'map' section, there are links for 'Show/Hide', 'List Operations', and 'Expand Operations'.

Game On
A throwback text application. Check [Game-On.org](https://gameontext.org) for more info!

map Show/Hide List Operations Expand Operations

GET	/map/v1/	
GET	/map/v1/health	Check application health
GET	/map/v1/sites	List sites
POST	/map/v1/sites	Create a room

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](https://twitter.com/KateStanley91)