Learning Microservices in the open with Game On!

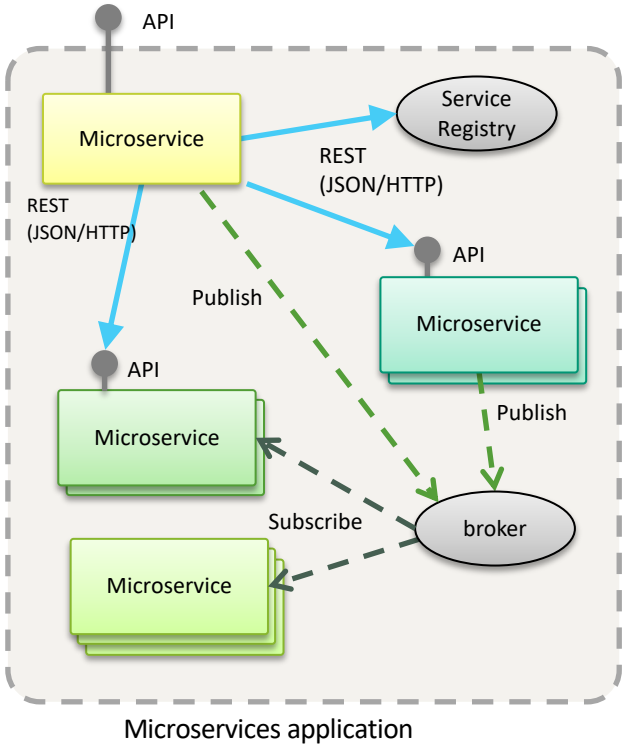
Erin Schnabel @ebullientworks
Kate Stanley @KateStanley91

October 2016

Microservices are used to…

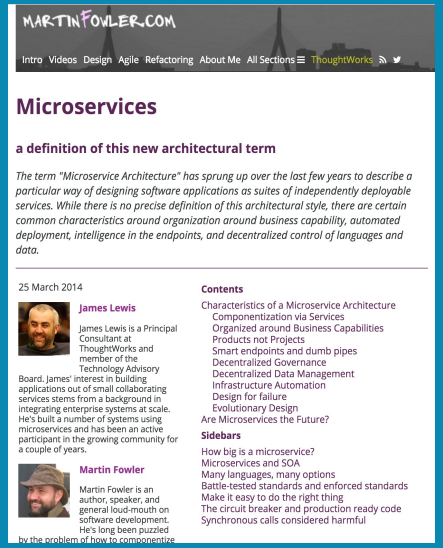
- compose a complex application using
  - “small”
  - independent (autonomous)
  - replaceable
  - processes

- that communicate via
  - language-agnostic APIs



Conway’s law

Bounded Contexts


MARTINFOWLER.COM

Intro Videos Design Agile Refactoring About Me All Sections ThoughtWorks

Microservices

a definition of this new architectural term

The term "Microservice Architecture" has sprung up over the last few years to describe a particular way of designing software applications as suites of independently deployable services. While there is no precise definition of this architectural style, there are certain common characteristics around organization around business capability, automated deployment, intelligence in the endpoints, and decentralized control of languages and data.

25 March 2014

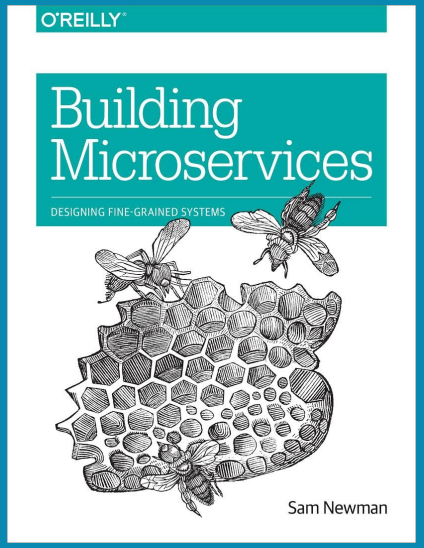
James Lewis

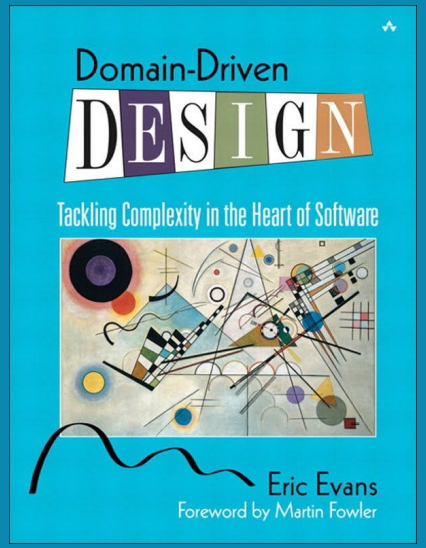
James Lewis is a Principal Consultant at ThoughtWorks and member of the Technology Advisory Board. James' interest in building applications out of small collaborating services stems from a background in integrating enterprise systems at scale. He's built a number of systems using microservices and has been an active participant in the growing community for a couple of years.

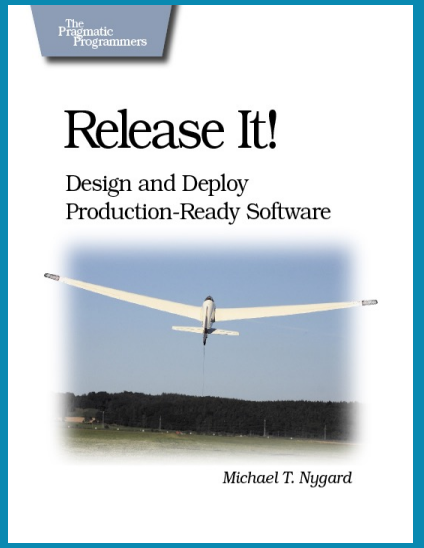
Martin Fowler

Martin Fowler is an author, speaker, and general loud-mouth on software development. He's long been puzzled by the problem of how to componentize

Contents







Eventual consistency

DevOps

Automation

Testing?

Microservices Sample Apps…

- Create a si…

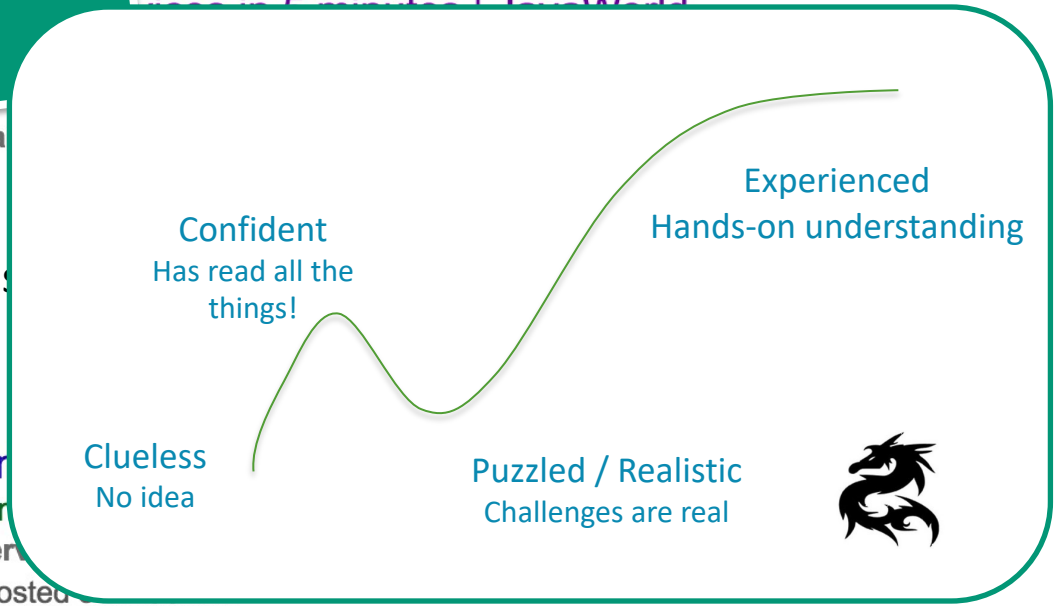
Microservices are so easy!

…ices in 5 minutes | JavaWorld

…ve not imported the sa…

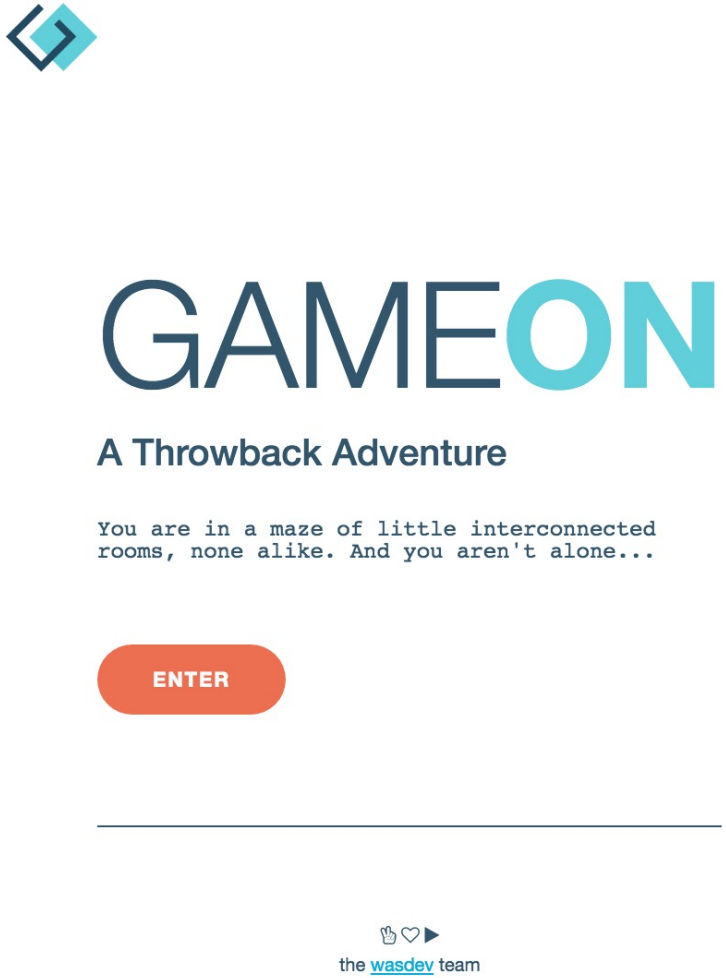
- Rebuild a pre-baked micros…

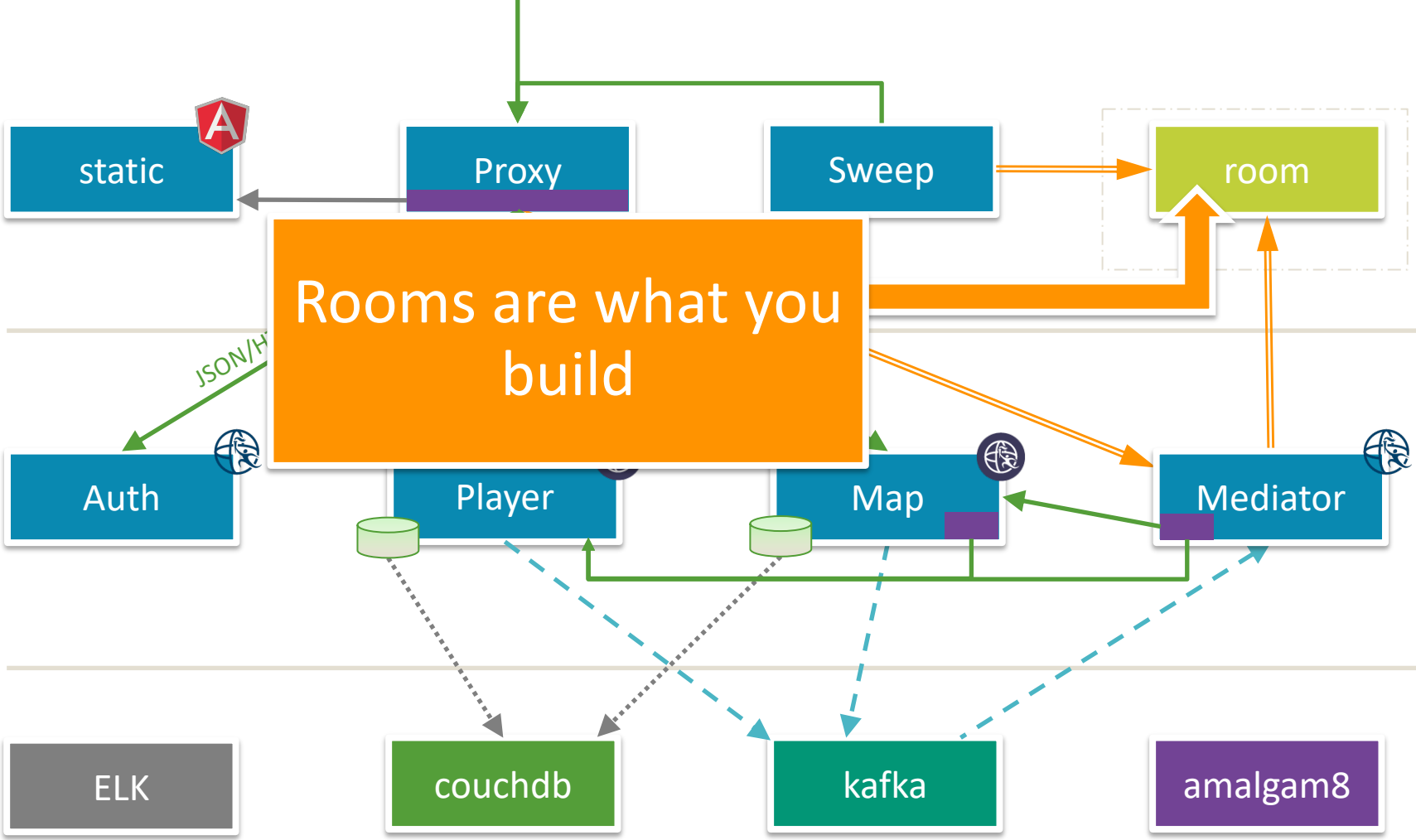
Microservices Onlin…
https://developer.ibm.cor…
Mar 16, 2015 - A microser…
Java JAX-RS, PHP and hosted …

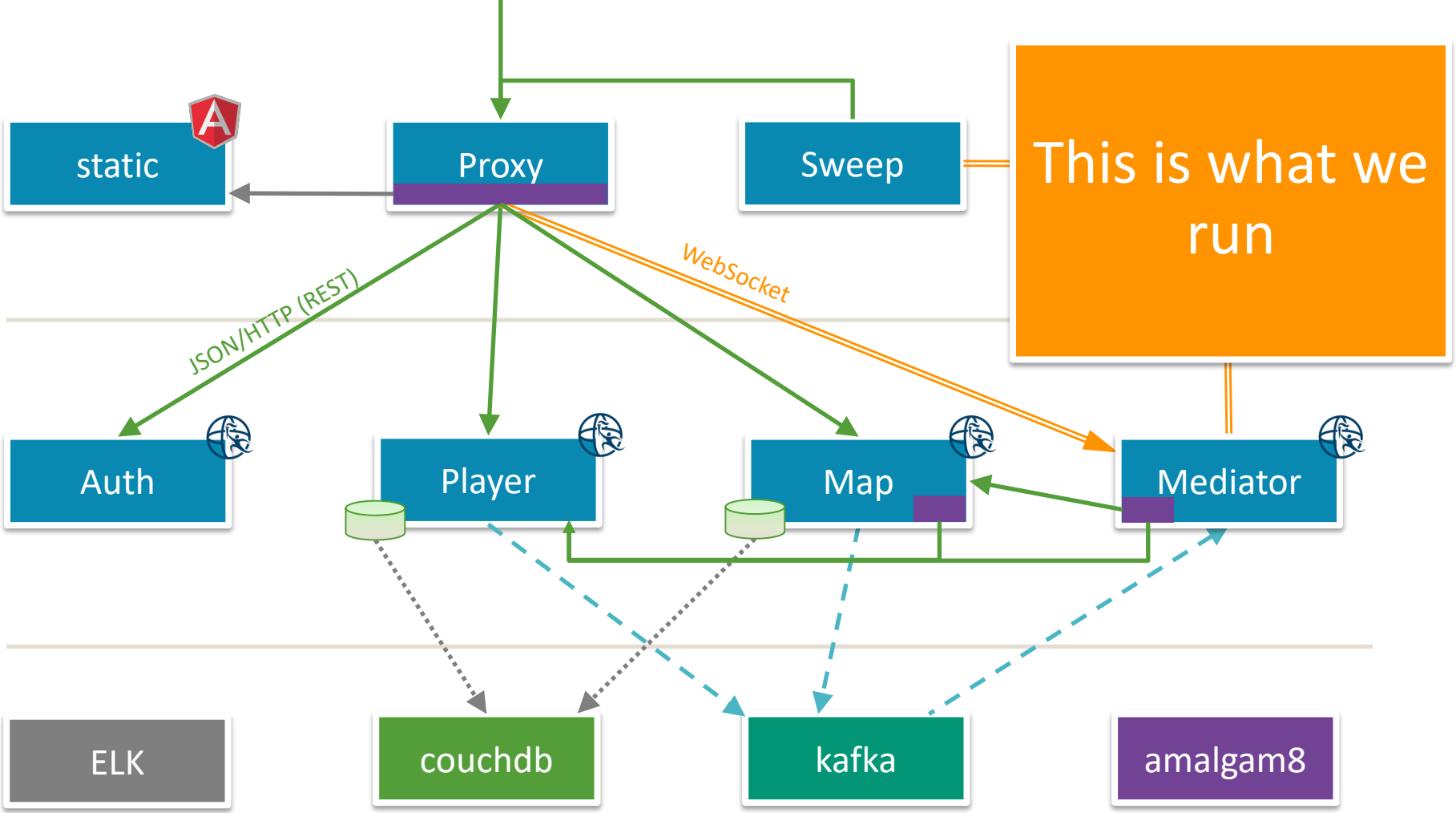


The premise …

- Hands on with microservices
- Stick with ’Hello World’ simplicity
- Choose your own adventure
- Fast path to the hard stuff
- Build something cool (to you!)
- Learn as you go




static
Proxy
Sweep
room
Rooms are what you build
JSON/H
Auth
Player
Map
Mediator
ELK
couchdb
kafka
amalgam8


static
Proxy
Sweep
This is what we run
JSON/HTTP (REST)
WebSocket
Auth
Player
Map
Mediator
ELK
couchdb
kafka
amalgam8

Game On! in action

connected: validating JWT
enter The First Room

Welcome to The First Room

The First Room

You've entered a vaguely squarish room, with walls of an indeterminate color.

TL;DR README (The extended edition is here):

- Commands start with '/'.
- Use `/help` to list all available commands. The lis
- Use `/exits` to list all available exits.
- Use `/sos` to return to First Room if you're stuck.
- Rooms might try to fool you, but these three commands will always work.

Retro, text-only interface

Simple text commands

/go N

/go N

You head North

So long, and thanks for all the fish.

exit The First Room
enter Look out what can you see

Connecting to Look out what can you see. Please hold.

Status updates

The room is strangely warm, expressing the malaise that comes with a fever as well as a room can. (1)

New microservice serving content

Look out what can you see

A hasty message has been taped to the wall, `Not feeling well, I've gone to lie down -- Look out what can you see`

You notice:

- Monitor
- Chart

How odd. The room has a stretched tense feeling, like it is desperately trying not to sneeze. (2)

The room emits a low and rhythmic rumble, like a congested chest. Is it breathing? (3)

How odd. The room has a stretched tense feeling, like it is desperately trying not to sneeze. (4)

</>

Demo: Creating a room

Microservices with a Game On! Room

codacy B

Game On! is both a sample microservices application, and a throwback text adventure brought to you by the WASdev team at IBM. This application demonstrates how microservice architectures work from two points of view:

1. As a Player: navigate through a network/maze of rooms. Each room is an autonomous service, supports chat, and may provide interaction with items (some of which may be in the room, some of which might be separately defined services as well).
2. As a Developer: learn about microservice architectures and their supporting infrastructure by creating your own microservices to extend the game.

You can learn more about Game On! at http://game-on.org/.

Deploy the room

mvn package

docker-compose.override.yml:

```
gojava:
  volumes:
    -  './gojava-wlpcfg/target/wlp/usr/servers/gojava-room:/opt/ibm/wlp/usr/servers/defaultServer'
```

docker-compose build

docker-compose up

Your room is running!

Consider this your primitive health check, well done!

Your websocket endpoint is **ws://192.168.99.100:9081/room**. Use browser extensions to play with the WebSocket directly, or go to http://game-on.org/ and check out the `/listmyrooms` command in First Room.

The WebSocket protocol used by Game On! is documented in the book.

Standing up this initial room was pretty easy, right?

So, what happens if you want to make this scale? How will the Map service find and connect to the instances of your service?

Each room instance uses WebSockets to broadcast events and chat to connected players. What happens when you have multiple instances of your room? Do you want all of them to behave as a single room (such that all events are visible in all rooms), or is each an independent/transient sharded instance?

Want to try adding items? How do you track the state of those items: In memory? In a shared cache? ...

Go play. Game On!

Room management

Game On! id: google:109552283218504689773

Use the following secret for room registration:

REGENERATE SHARED SECRET

Select or create a room:

Room id

Room Name Kate's room

Full Name Kate's fun new room

WebSocket endpoint ws://192.168.99.100:9081/room

Description

You are in a brightly lit room full of sofas. They all look very comfy!

/listmyrooms

You have registered the following rooms...

- 'Kate's fun new room' with id Kate room (long id: c7365d0c1dbfc17a9b347fcbd6c8f33c)

You can go directly to a room using /teleport <roomid>

/teleport Kate room

You punch the coordinates into the console, a large tube appears from above you, and you are sucked into a maze of piping.

Katherine_Stanley has gone

exit The First Room
enter Kate's fun new room

Connecting to Kate's fun new room. Please hold.

You have entered the room

A Very Simple Room.

You are in the worlds most simple room, there is nothing to do here.

exit The First Room
enter Kate's fun new room

Connecting to Kate's fun new room. Please hold.

You have entered the room

Kate's fun new room

You are in the a brightly lit room full of sofas. They all look very comfy!

What next?

What happens when…

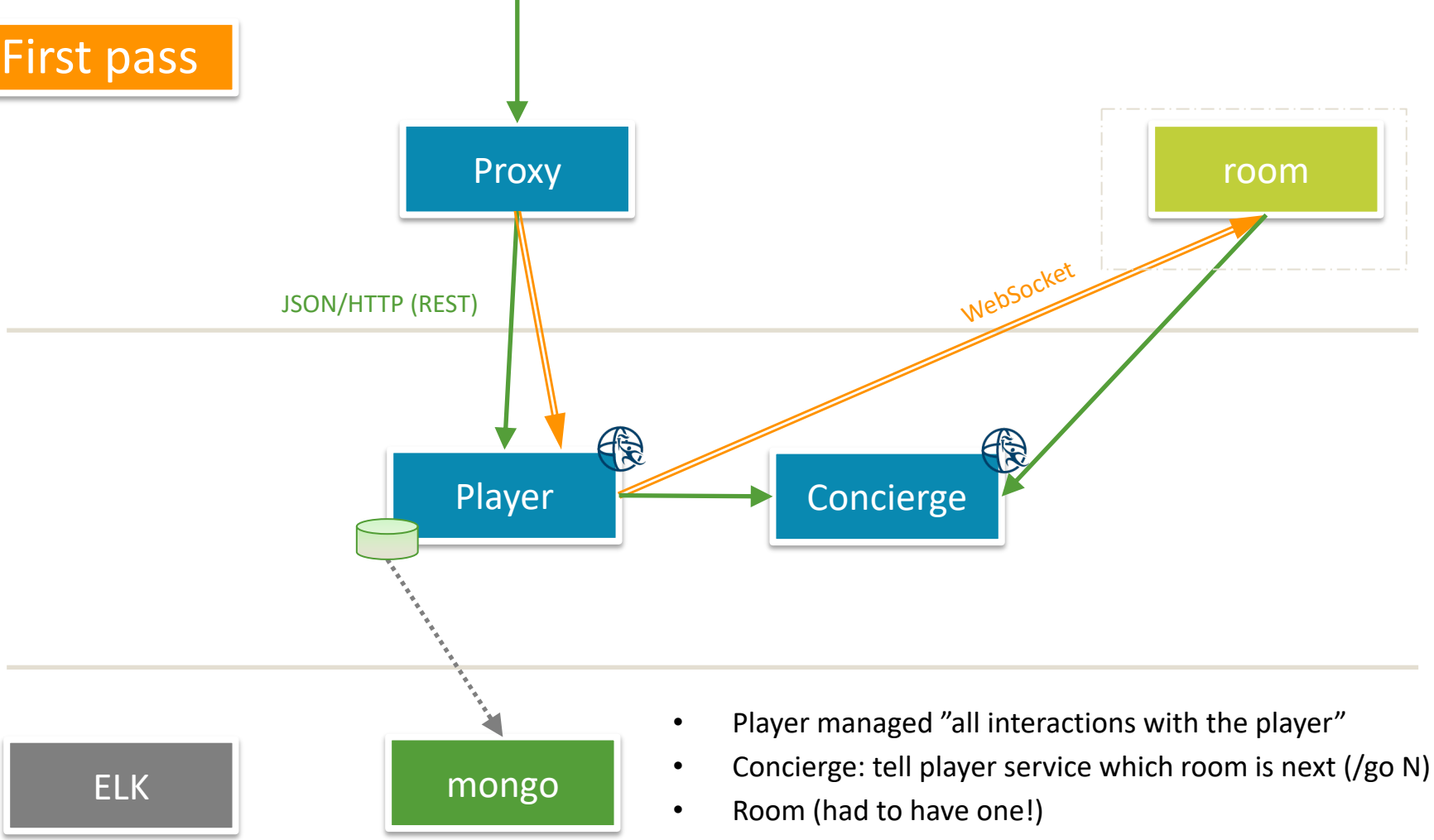
1. As a player, navigate from room to room
   - Notice latency?
   - What is impact of calls to additional services?
     - Fallacies of distributed computing! Gotcha!

2. You have the most popular room ever!!
   - How do you scale this thing?
   - What about items or shared state?
   - Where are players?

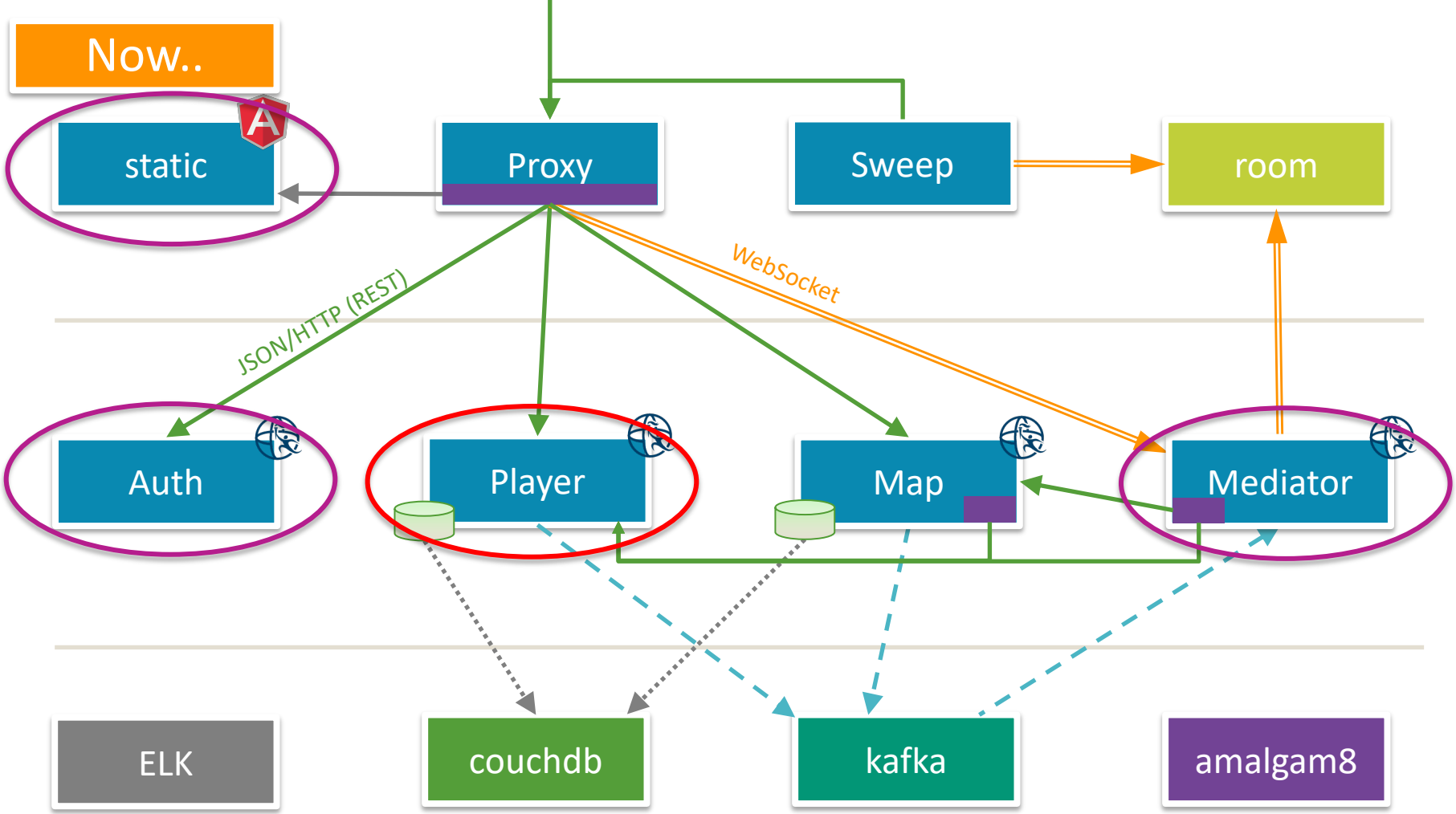
Exploration of solutions for caching, circuit breakers, service interaction patterns

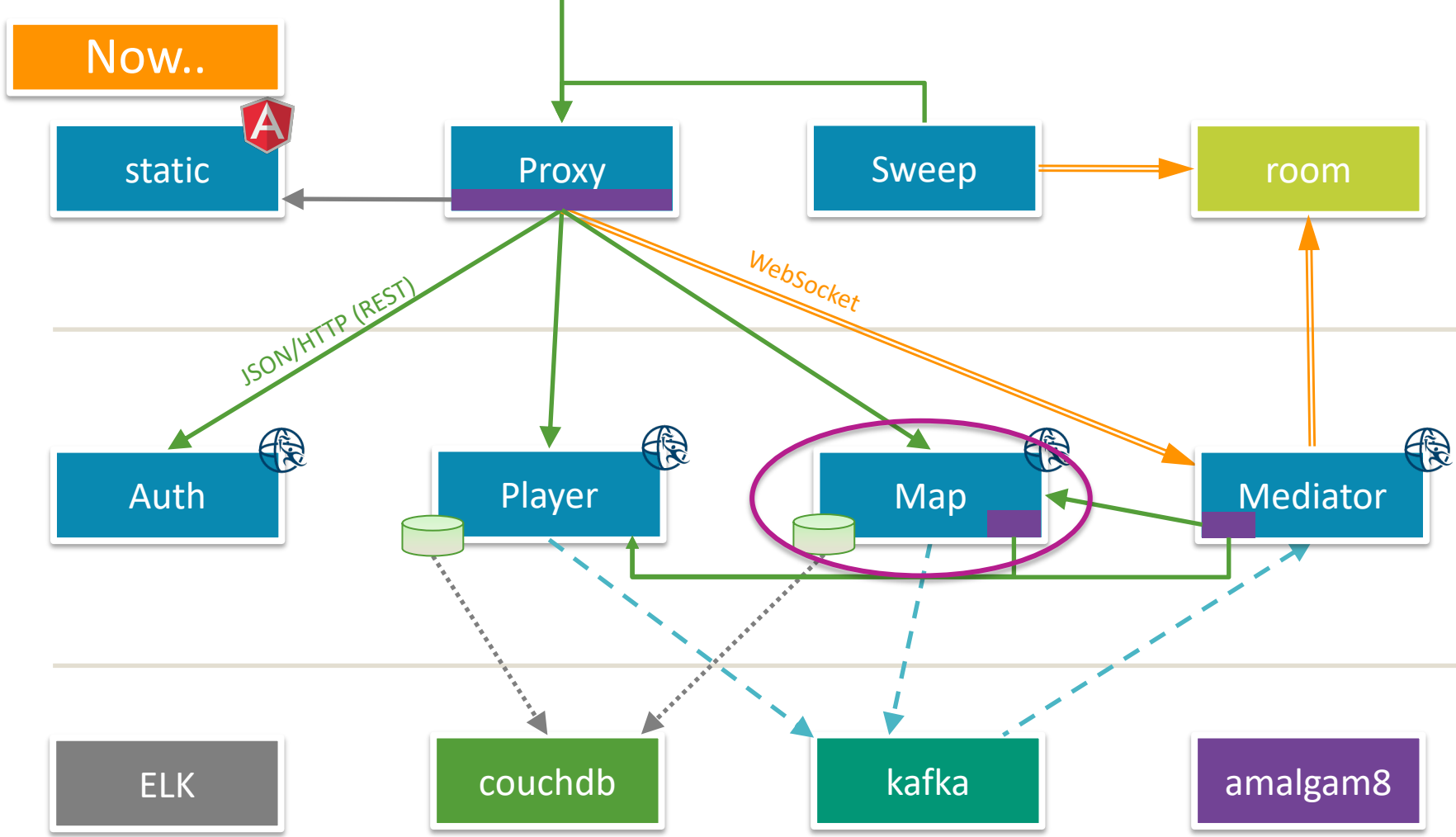
Service composition

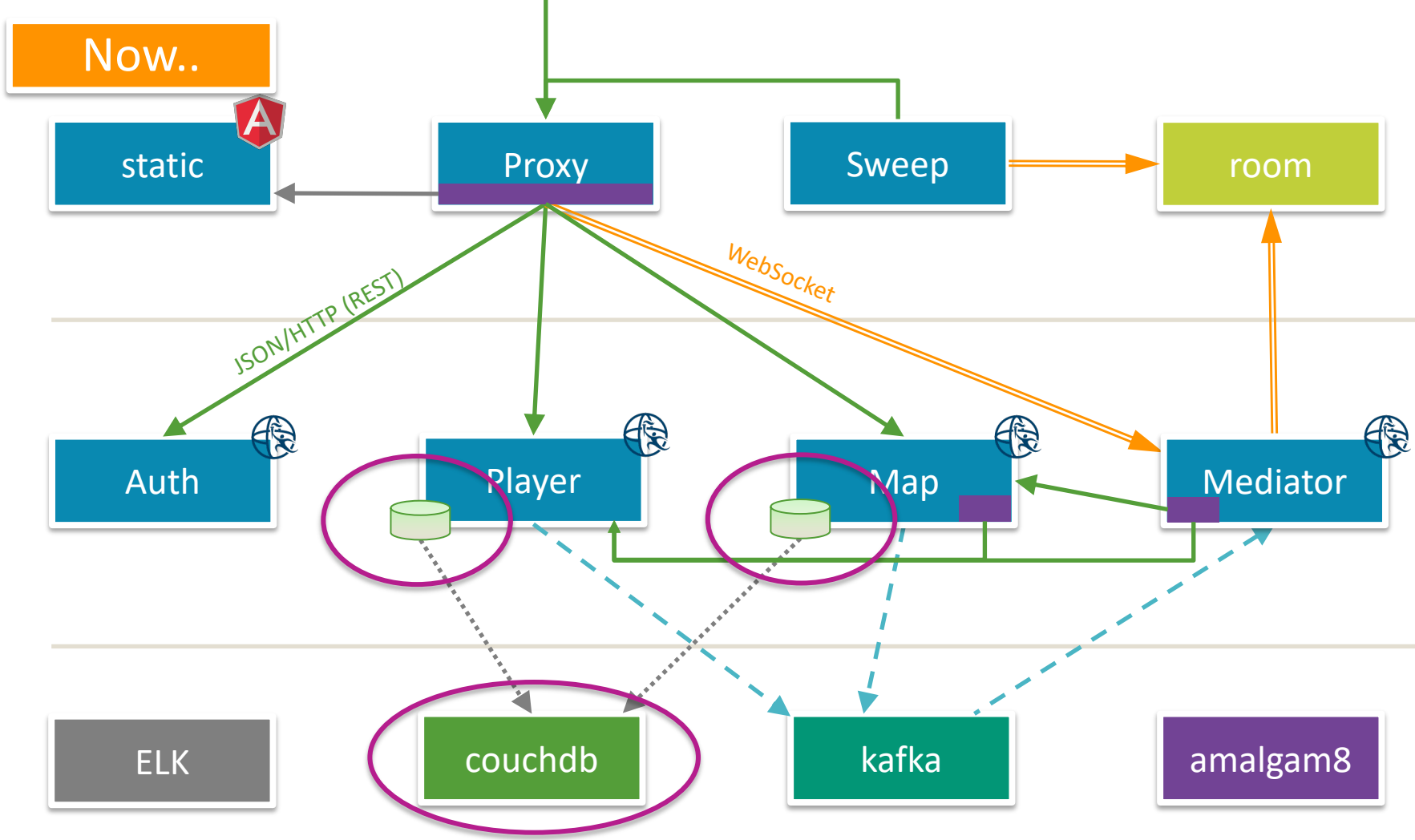
First pass



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


Now..
static
Proxy
Sweep
room
JSON/HTTP (REST)
WebSocket
Auth
Player
Map
Mediator
ELK
couchdb
kafka
amalgam8


Now..
static
Proxy
Sweep
room
JSON/HTTP (REST)
WebSocket
Auth
Player
Map
Mediator
ELK
couchdb
kafka
amalgam8


Now..
static
Proxy
Sweep
room
JSON/HTTP (REST)
WebSocket
Auth
Player
Map
Mediator
ELK
couchdb
kafka
amalgam8

Twelve Factors

Twelve factor applications

- "a methodology for building software-as-a-service applications"
  - Created by developers at Heroku

- Factors are independent of
  - programming language,
  - backing services,
  - cloud provider

- http://12factor.net/

The Twelve Factors

I. Codebase
One codebase tracked in revision control, many deploys

II. Dependencies
Explicitly declare and isolate dependencies

III. Config
Store config in the environment

IV. Backing Services
Treat backing services as attached resources

V. Build, release, run
Strictly separate build and run stages

VI. Processes
Execute the app as one or more stateless processes

VII. Port binding
Export services via port binding

VIII. Concurrency
Scale out via the process model

IX. Disposability
Maximize robustness with fast startup and graceful shutdown

X. Dev/prod parity
Keep development, staging, and production as similar as possible

XI. Logs
Treat logs as event streams

XII. Admin processes
Run admin/management tasks as one-off processes

Git + Submodules (Factor 1: codebase)

- Root repository: https://github.com/gameontext/gameon
  - Optional use of submodules

- Key: Only builds update submodule commit levels
  - Prevents conflicts and confusion caused by humans

Containers (Factor 2: dependencies, 5: build/release/run, 6: Processes, 8: concurrency, 10: dev/prod parity)

- Encapsulation of all dependencies
- Parity: dev -> test -> prod
- Configuration passed in via environment

- Local: Docker Compose or Vagrant
  - Pre-built images in dockerhub (this came later..)
  - Overlays for local editing

- Independent build pipelines per service to deploy containers

Liberty (Factor 2, 10, 3: config, 4: backing services, 7: port binding, 9: disposability)

- Java services are Liberty-based
- Customizable features: Cachable Docker Layers
  - Explicit app server dependencies
  - Self-contained immutable artifact
  - Smaller war (smaller delta)

- Environment variables in server config
  - Common configuration across environments
  - Config munging not necessary
  - Composable configuration w/ dropins if required

```
# Install required features
RUN /opt/ibm/wlp/bin/installUtility install \
    apiDiscovery-1.0 \
    bluemixLogCollector-1.1 \
    cdi-1.2 \
    concurrent-1.0 \
    couchdb-1.0 \
    localConnector-1.0 \
    jaxrs-2.0 \
    jndi-1.0 \
    jsonp-1.0 \
    ssl-1.0 \
    websocket-1.1
```

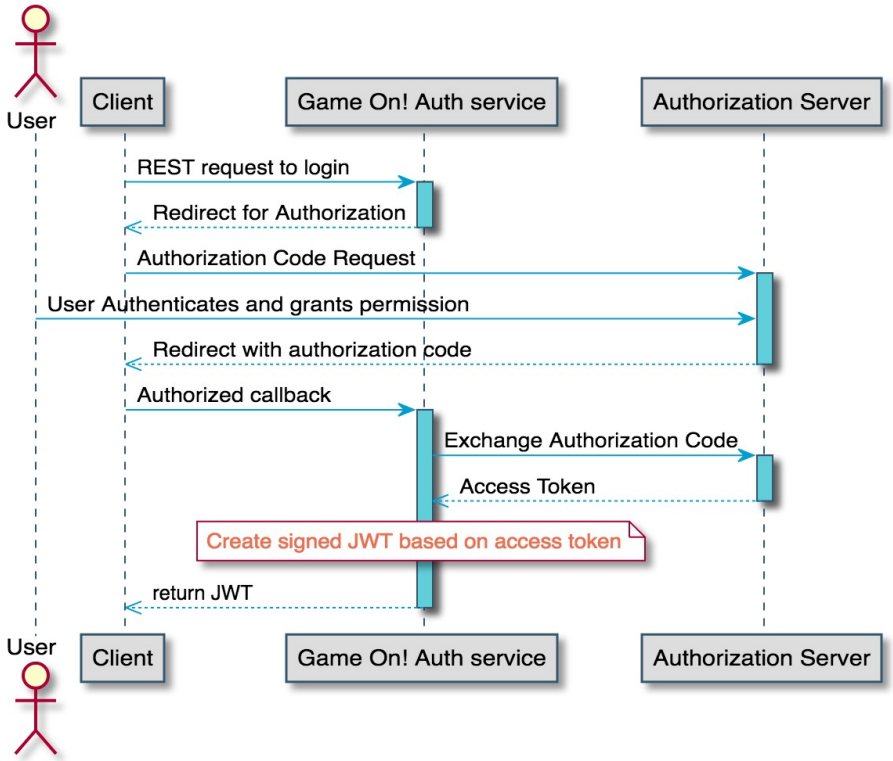
```
<couchdb id="couchdb"
         jndiName="couchdb/connector"
         libraryRef="couchdb-lib"
         password="${env.COUCHDB_PASSWORD}"
         url="${env.COUCHDB_SERVICE_URL}"
         username="${env.COUCHDB_USER}"/>
```

Security

OAuth & JWTs

- OAuth proxy
  - Application id w/ different front-end
  - Could be a gateway instead

- Access token converted into signed JWT
- System services deal only with JWT
  - game-on.org SSL certificate
  - Well-known public key



Hashed message authentication codes (HMACs)

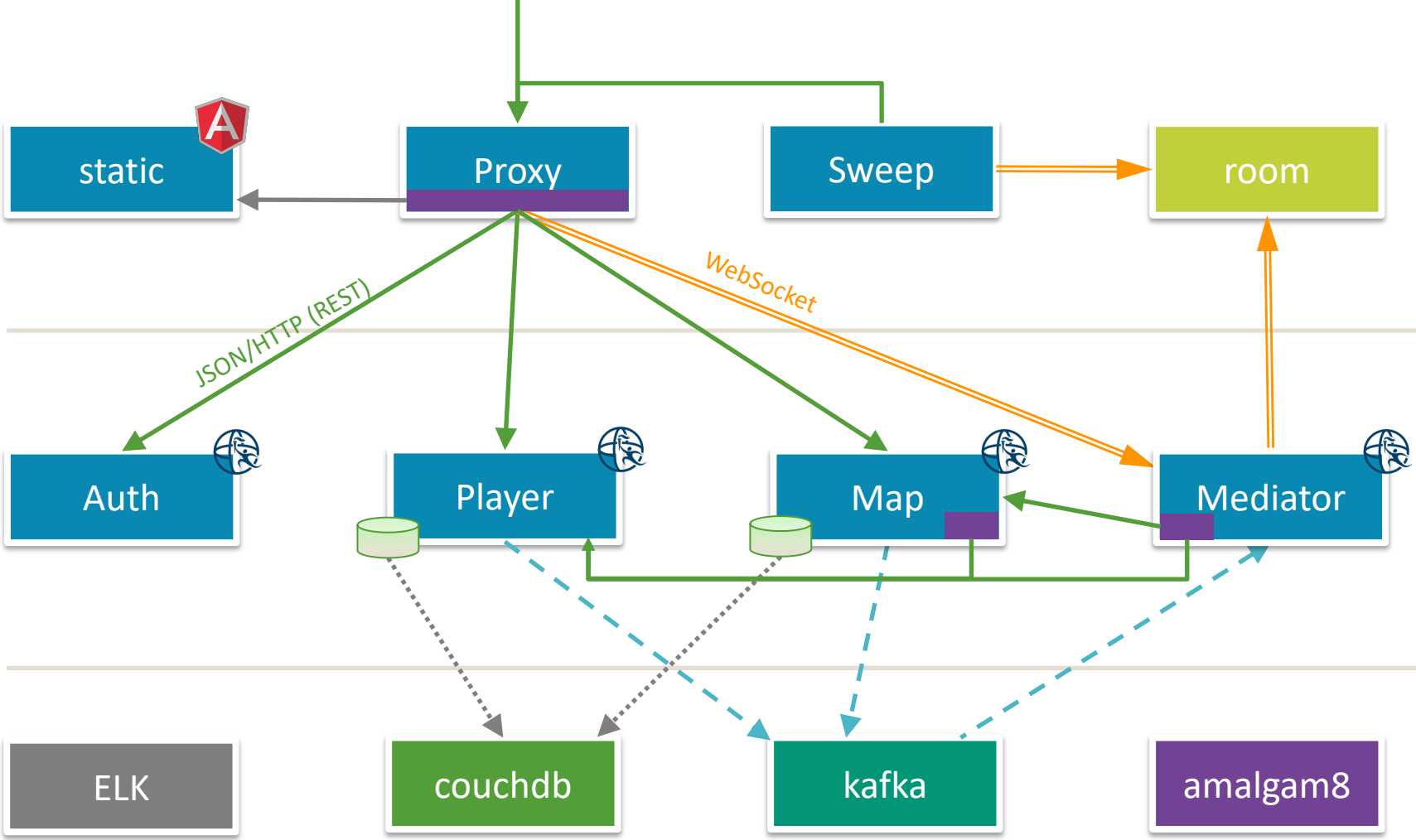
- Shared secrets
  - Credentials not sent on the wire
  - Used to verify identity of sender

- Map operations
  - Mutable operations require HMAC signature
  - Hashed signature used to prevent replays

- Room handshake for WebSocket
  - It is the game calling the room
  - Room answering the game

- https://book.game-on.org/microservices/ApplicationSecurity.html

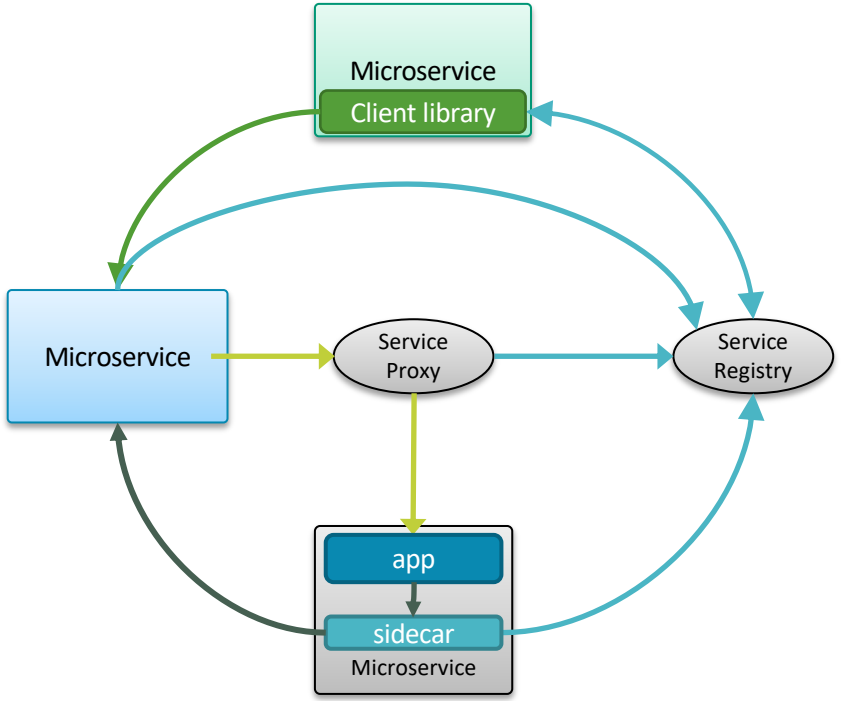
Service discovery


static
Proxy
Sweep
room
JSON/HTTP (REST)
WebSocket
Auth
Player
Map
Mediator
ELK
couchdb
kafka
amalgam8

Service registration and discovery

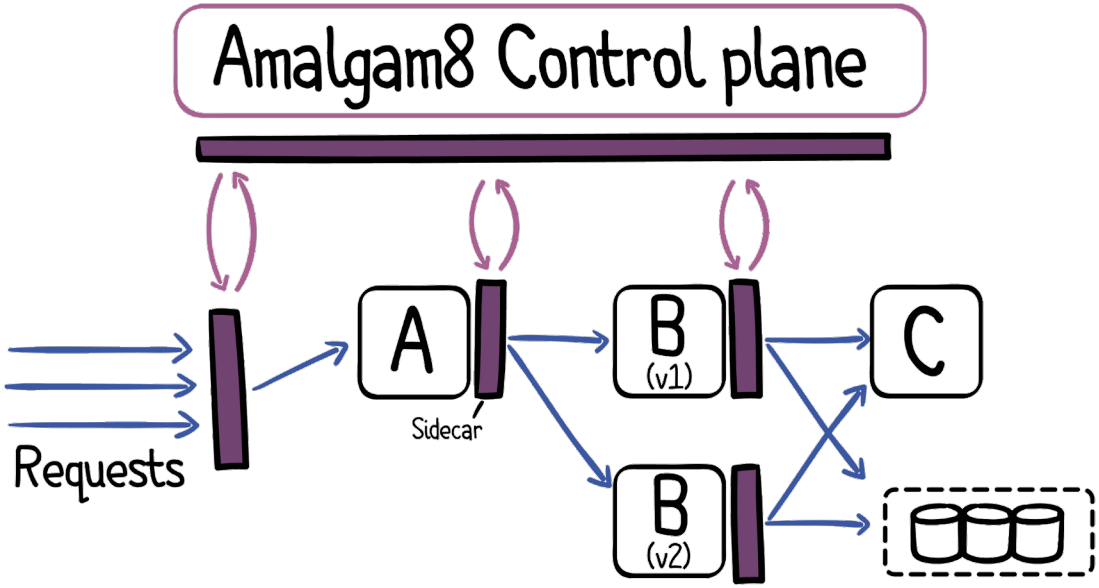
- Required for load balancing and scaling
- Services need to find each other
- Environment changes constantly

- Client-side or server-side?
- Client library, sidecar, or proxy?



Amalgam8

- Basics
  - Service registration
  - Service discovery
  - Client-side load balancing



- How
  - Sidecar model. 2 options:
    - An independent process running in the same container as the app
    - An independent container running in the same pod as the app container

Successful?

Example rooms

- Map room
- Weather room
- Liberty car
- Pi-Cam

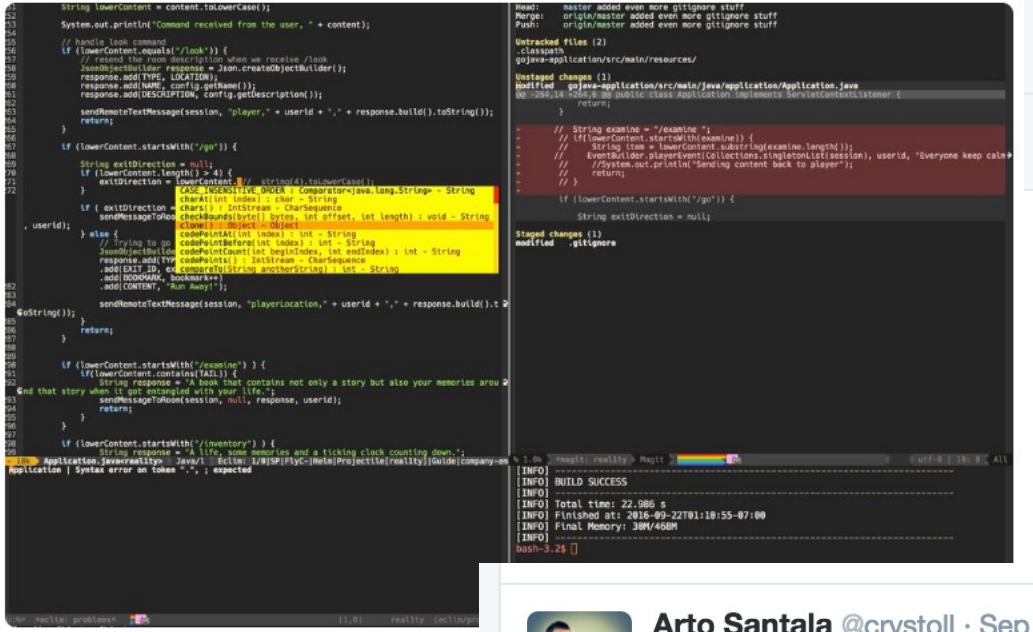
	-4	-3	-2	-1	0	1	2	3
3					Empty			
2			Empty	Empty	WalkThruThur (GWC)	Empty		
1		Empty	A Very Simple Room. (Alvin_Uy)	MyRoom (Simon_Boyden)	GO.CONTAINER.4 (GWC)	The Room with The Mug (game-on.org)	Empty	
0	Empty	The Node-JS Room (game-on.org)	WalkthruWed (GWC)	Rec Room (game-on.org)	The first Room (game-on.org)	GoMonday (GWC)	This is a CF in bluemix (Ant_Mazzone)	Empty
-1		Empty	A Very Simple Room. (hongsign)	A dusty room beneath the capstone... (Ozzy)	Basement (game-on.org)	This is a CF in bluemix (GWC)	Empty	
-2			Empty	Empty	A Very Simple Room. (LargeMuffin)	Empty		
-3					Empty			

(LastUpdated: 2016-04-14T21:46:43.829Z)

swardley @swardley · Aug 2

One I want to try -> Learning microservices in the open with GameOn! by **@ebullientworks** conferences.oreilly.com/oscon/open-sou… #oscon

ha! got it working in emacs so I dont have to much around in those ides that drive me up the wall! Now I can go to sleep.



4h

Learning microservices in the open with GameOn! -...

There are plenty of talks out there about how to get started with microservices, but in reality you learn by doing. Erin Schnabel and Katherine Stanley explore l...

conferences.oreilly.com

2

Arto Santala @crystoll · Sep 19

@ebullientworks @Dev_Events @gameontext Cool, looking forward to try it!

1 1

Thank You!

Play – http://game-on.org

Learn more – http://book.game-on.org

Erin Schnabel | schnabel@us.ibm.com | @ebullientworks
Kate Stanley | katheris@uk.ibm.com | @KateStanley91