# BUILDING CLOUD NATIVE APPLICATIONS: BEST PRACTICES IN ACTION

ERIN SCHNABEL

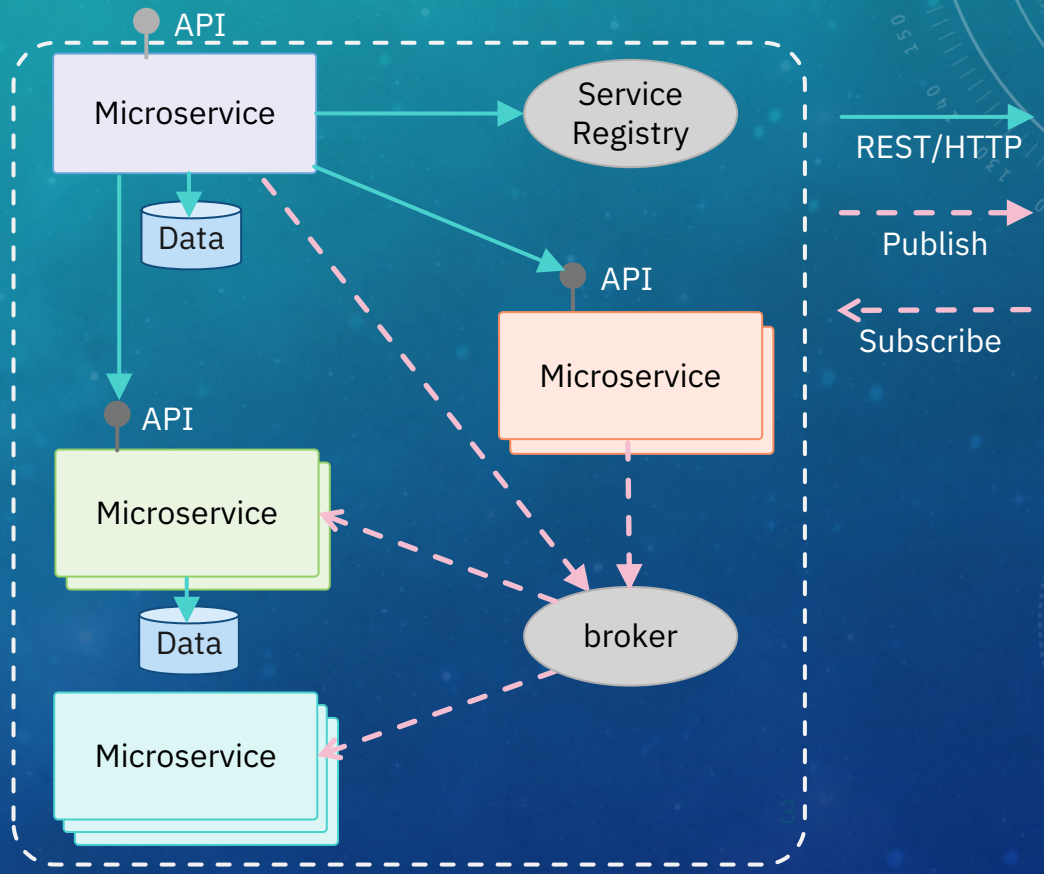
@EBULLIENTWORKS

# CLOUD NATIVE

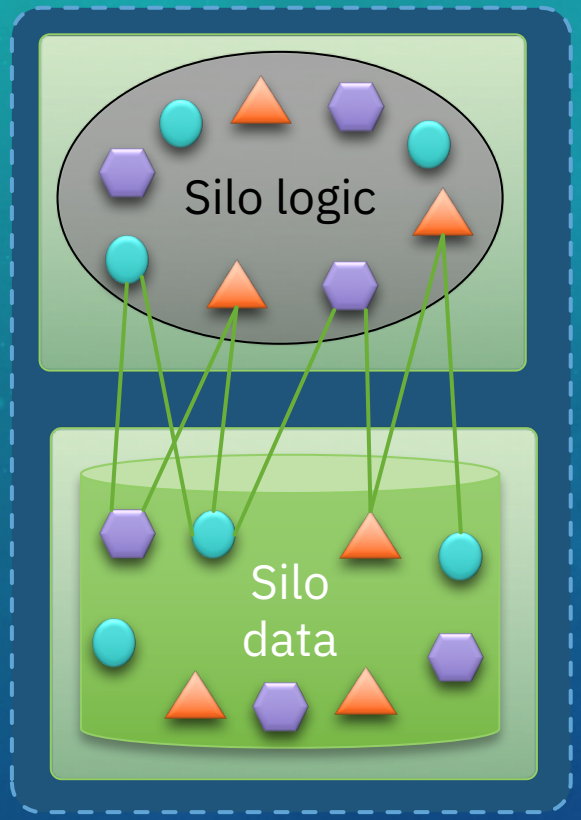
An application architecture designed to leverage the strengths and accommodate the challenges of a standardized cloud environment, including concepts such as elastic scaling, immutable deployment, disposable instances, and less predictable infrastructure.[2]

# MICROSERVICES ARE USED TO...

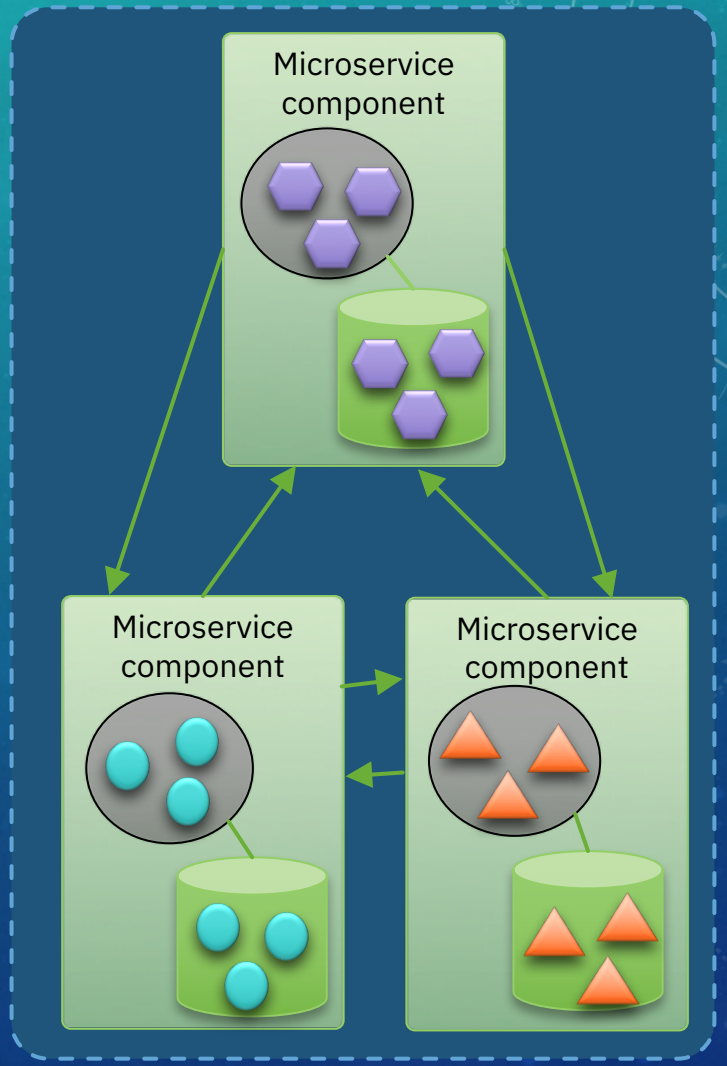
- compose a complex application using
  - “small”
  - independent (autonomous)
  - replaceable
  - processes
- that communicate via
  - language-agnostic APIs



# ISOLATE ALL THE THINGS!

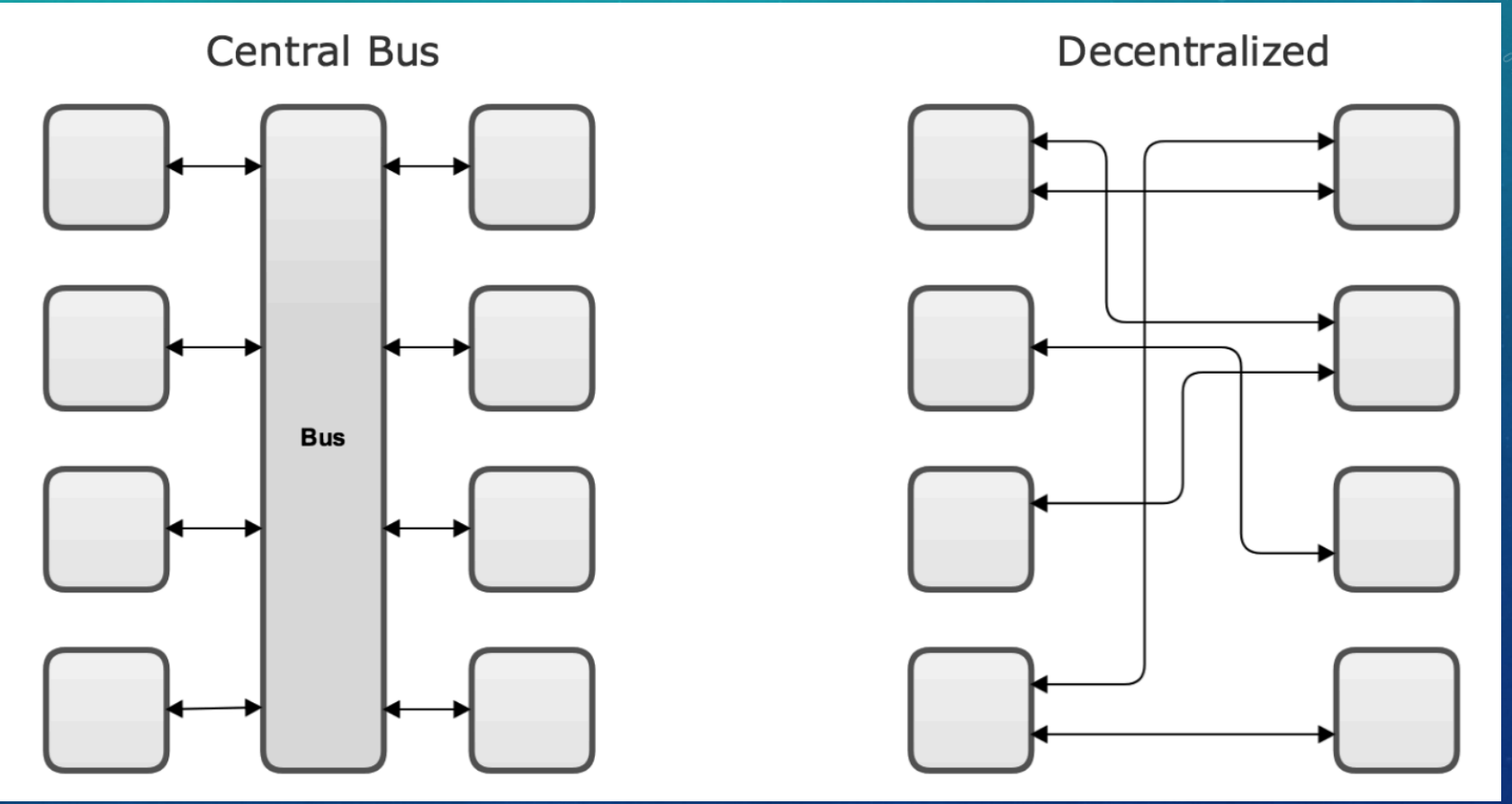



Monolithic application




Microservices application

# SMART ENDPOINTS, DUMB PIPES

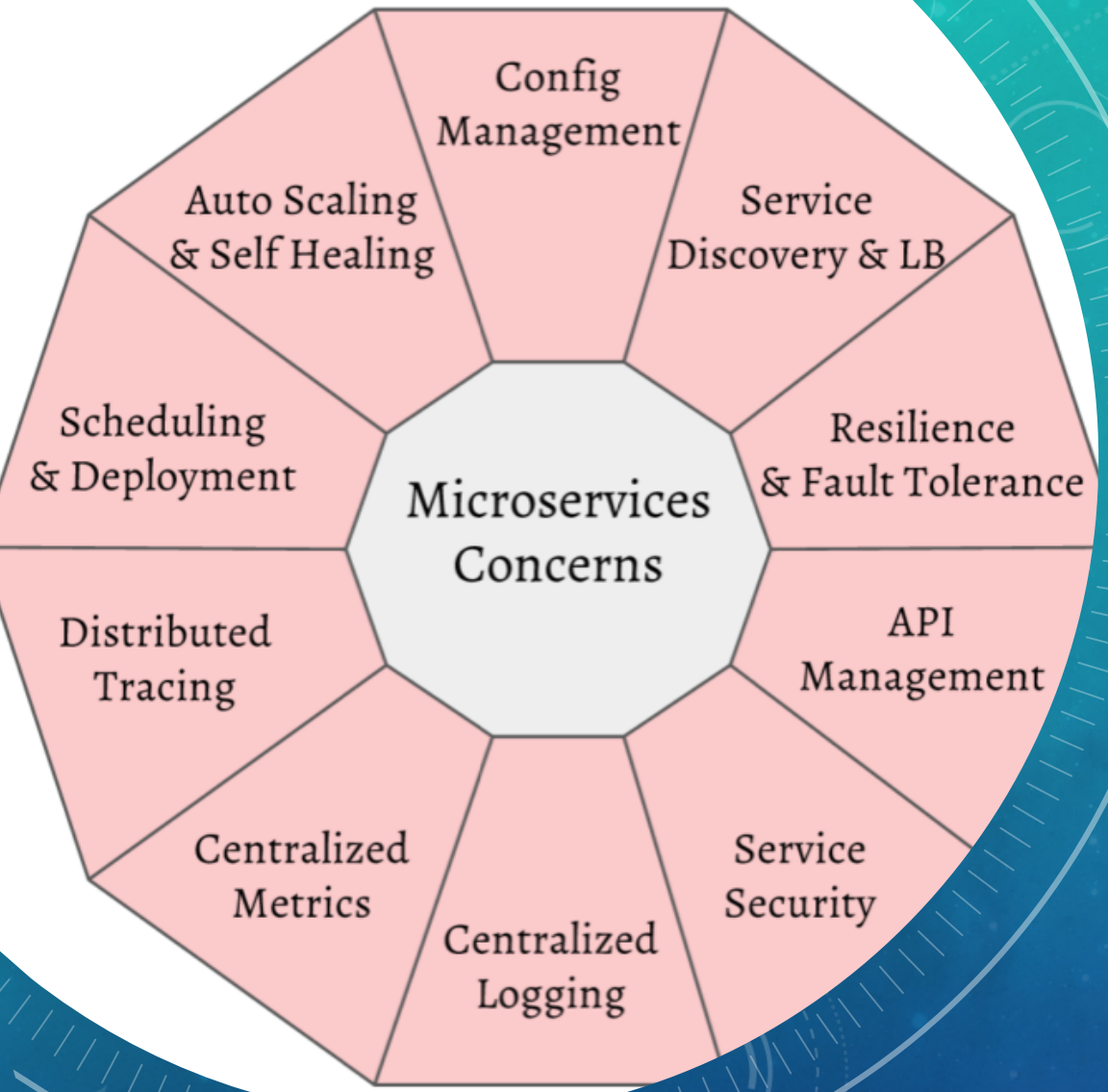



https://medium.com/@nathankpeck/microservice-principles-smart-endpoints-and-dumb-pipes-5691d410700f

"Twitter microservices map looks just like the Netflix one. "We called this the 'Death Star' diagram"

– Adrian Cockcroft via twitter

https://twitter.com/adrianco/status/441883572618948608

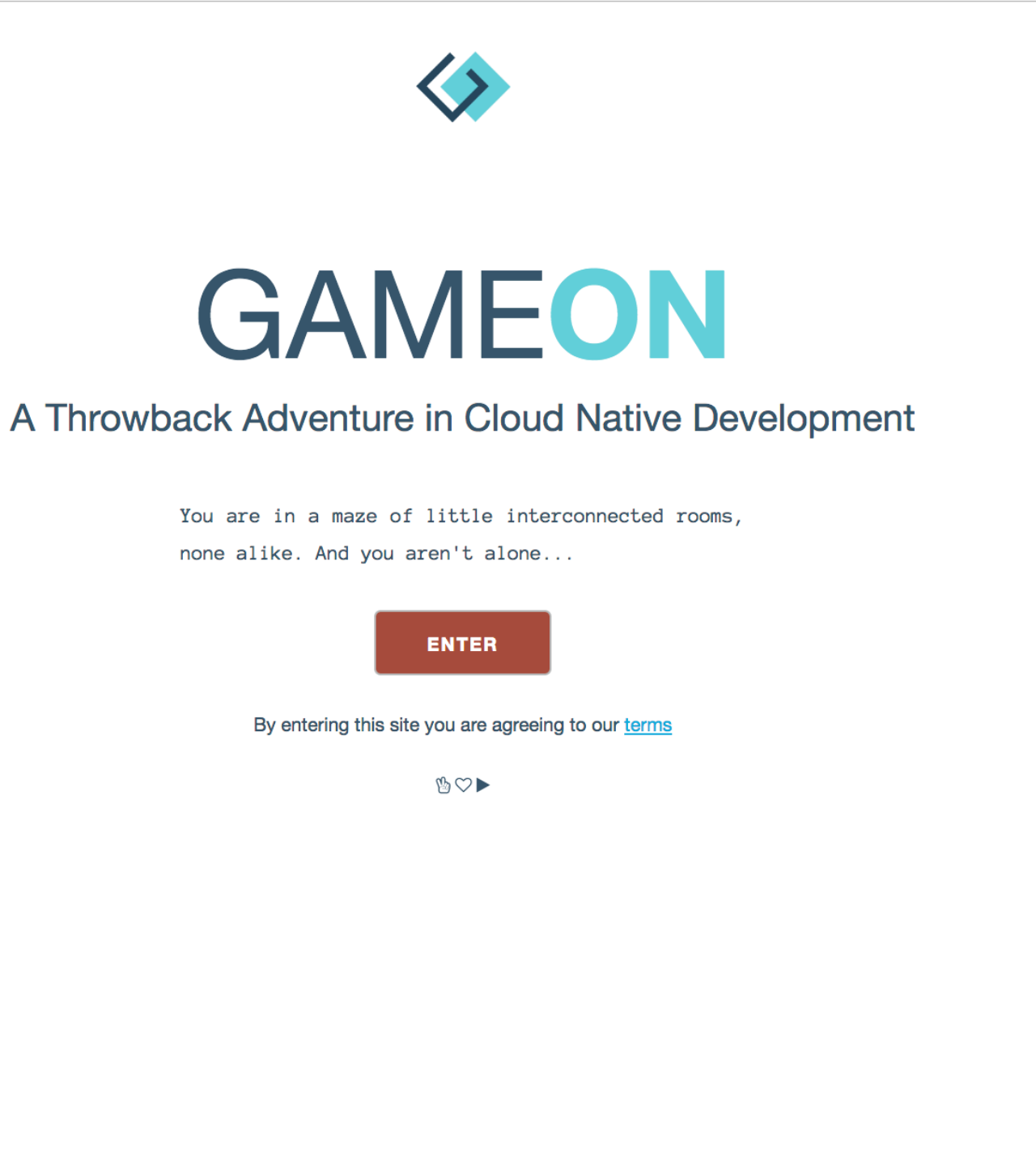




# THIS IS NOT EASY

# 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



**Join us on Slack!**

slack 7/141

## Book

What is this game, and what does it have to do with microservices? Read on!

## Blog

Extra! Extra! Get the latest on our adventures, whether they be at events or just in code.

## Swagger APIs

Our callable REST APIs. Cleanly documented with Swagger. Ready to be poked with sticks.

## GitHub Projects

Game On! is open source. All the code. All the words.

## Docker Images

Game On! has pre-built images on Docker Hub to make local development easier.

## Contributors

Lots of people helped to build this game. We hope you will, too.

# GAMEON

## A Throwback Adventure in Cloud Native Development

You are in a maze of little interconnected rooms,
none alike. And you aren't alone...

ENTER

By entering this site you are agreeing to our terms

https://gameontext.org

**Join us on Slack!**

slack 7/141

## Book

What is this game, and what does it have to do with microservices? Read on!

## Blog

Extra! Extra! Get the latest on our adventures, whether they be at events or just in code.

## Swagger APIs

Our callable REST APIs. Cleanly documented with Swagger. Ready to be poked with sticks.

## GitHub Projects

Game On! is open source. All the code. All the words.

## Docker Images

Game On! has pre-built images on Docker Hub to make local development easier.

## Contributors

Lots of people helped to build this game. We hope you will, too.

Sign in

TWITTER

GOOGLE

FACEBOOK

GITHUB

**Join us on Slack!**

slack 7/141

## Book

What is this game, and what does it have to do with microservices? Read on!

## Blog

Extra! Extra! Get the latest on our adventures, whether they be at events or just in code.

## Swagger APIs

Our callable REST APIs. Cleanly documented with Swagger. Ready to be poked with sticks.

## GitHub Projects

Game On! is open source. All the code. All the words.

## Docker Images

Game On! has pre-built images on Docker Hub to make local development easier.

## Contributors

Lots of people helped to build this game. We hope you will, too.

### User Profile

| Username | Favorite color |
| --- | --- |
| GlutenFreePastrySlice | Fuschia |
| GENERATE A USERNAME | GENERATE A COLOR |
| GlutenFreePastrySlice moves the chair. | GlutenFreePastrySlice loves the color Fuschia |

DONE!

*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. A note is pinned to the wall.

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

- Commands start with '/'.
- Use `/help` to list all available commands. The list will change from room to room.
- 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.

You notice:

- Note

The First Room

*connected: validating JWT*

*enter The First Room*

Welcome to The First Room

**The First Room**

---

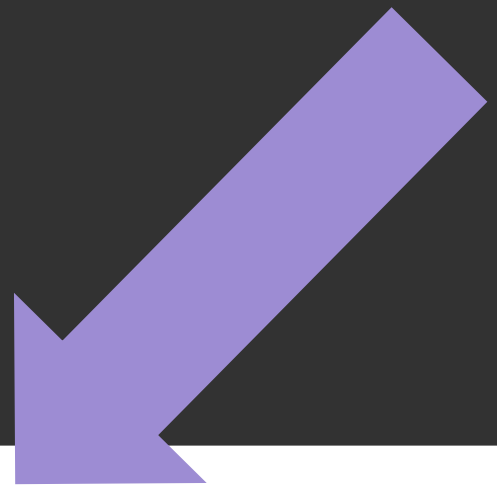
You've entered a vaguely squarish room, with walls of an indeterminate color. A note is pinned to the wall.

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

- Commands start with '/'.
- Use `/help` to list all available commands. The list will change from room to room.
- 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.

You notice:

- Note



</> /examine Note

# THINGS TO TRY

- **/go <direction: E|W|N|S>**
- **/help** – Rooms provide additional/custom commands
- **/sos** – Emergency return to First Room

- From First Room:
  - **/go W** to ”Junky Place” – written by an 8 year old
  - **/go E** to “Rec Room” – this room has a puzzle

# PAUSE AND THINK …

- How would you build this?

- **Where do we start?**
- **How big** is a Microservice?
- Should every Microservice **own its own data**?

2 minutes … GO!

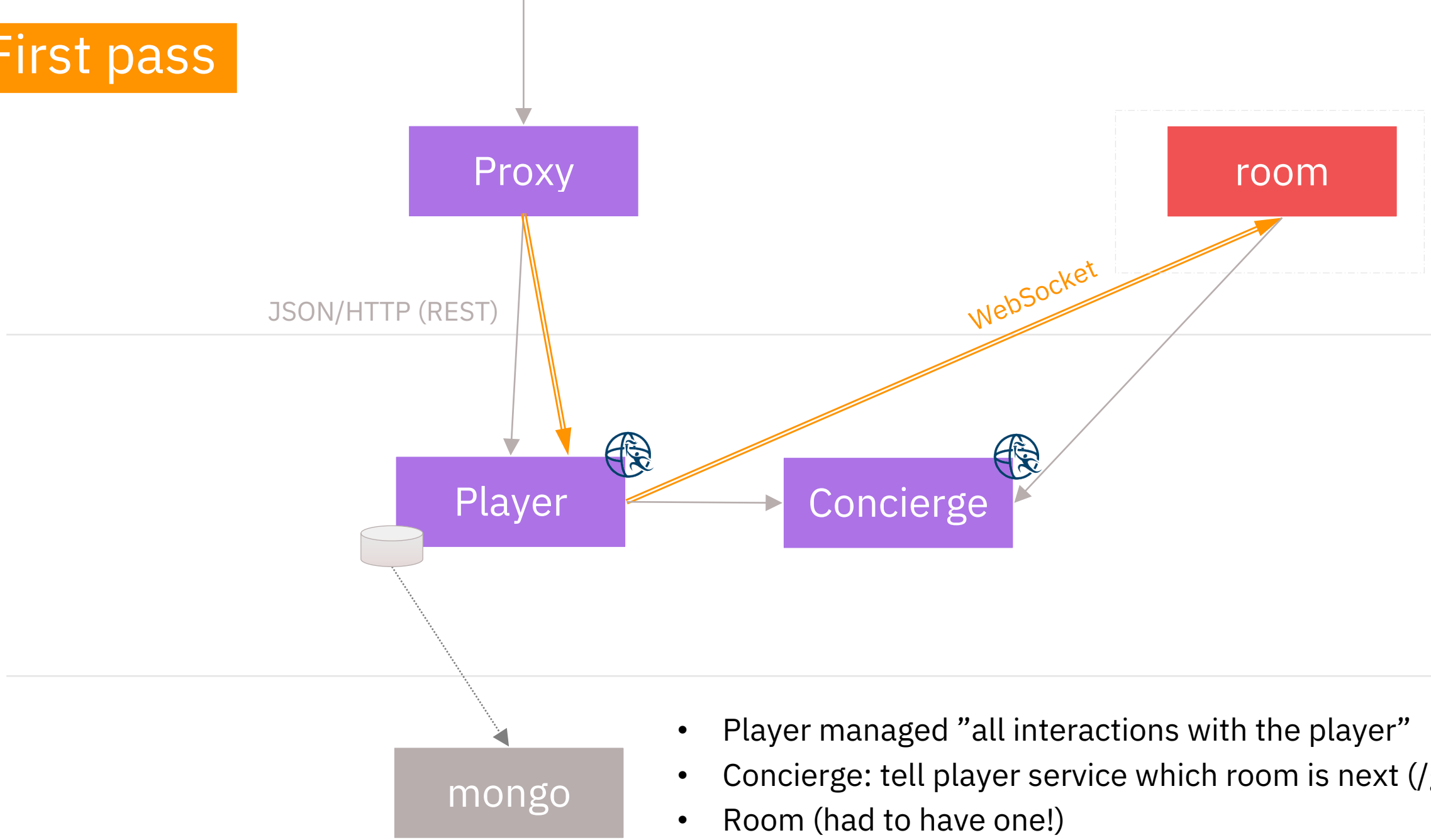
**https://gameontext.org**

# PAUSE AND THINK …

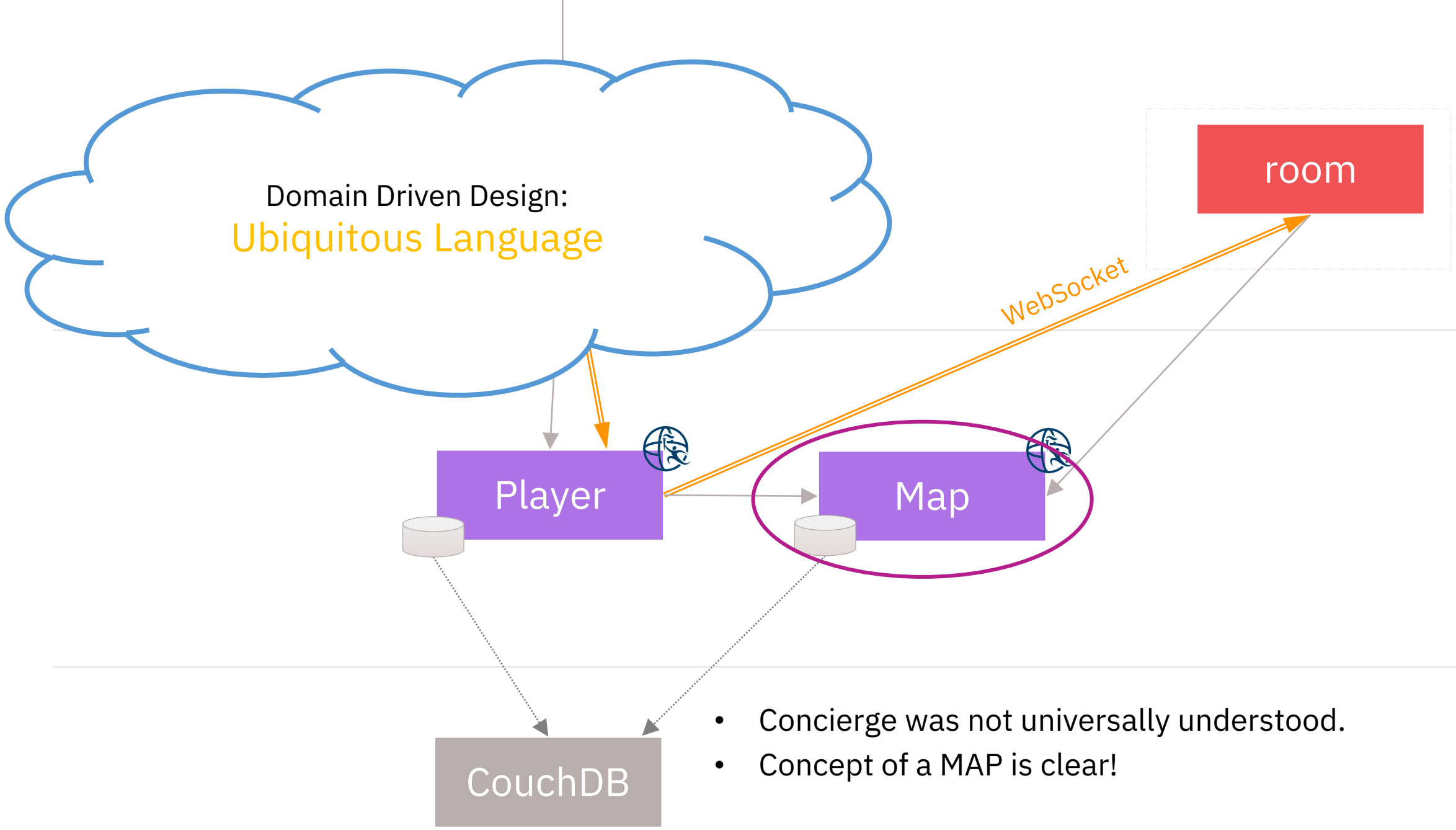
- How would you build this?

## How many services did you guess?

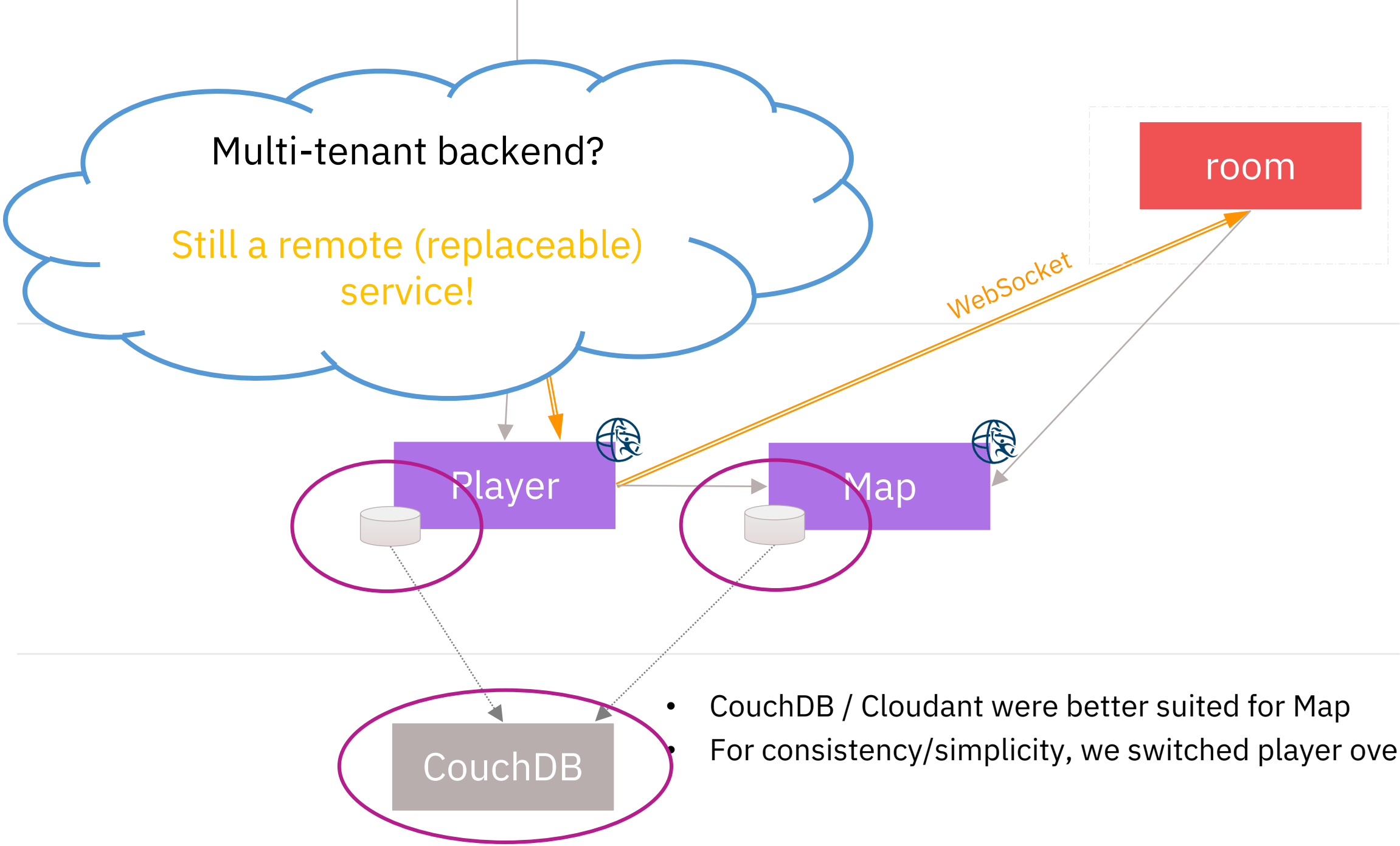
# First pass




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

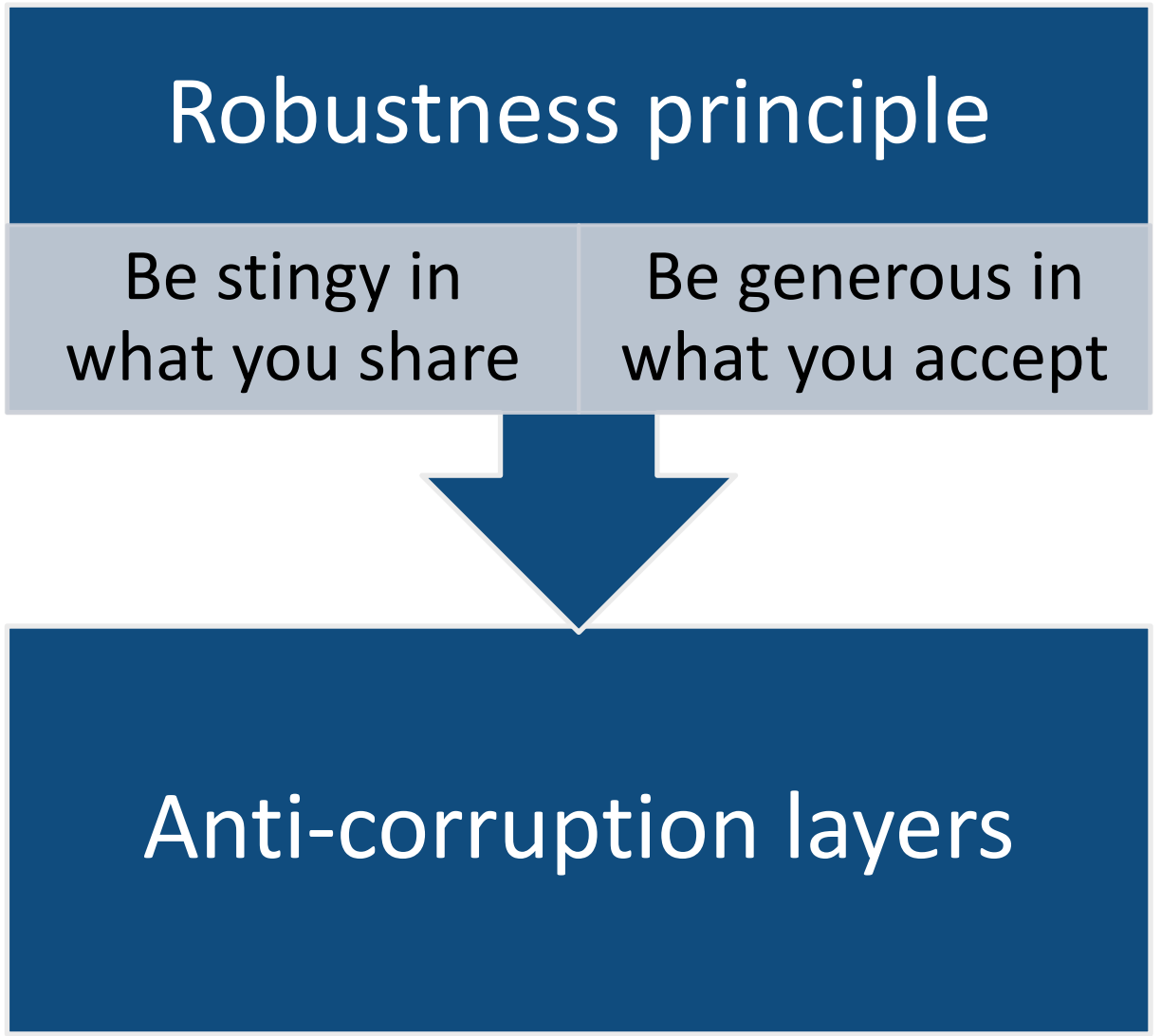



- Concierge was not universally understood.
- Concept of a MAP is clear!

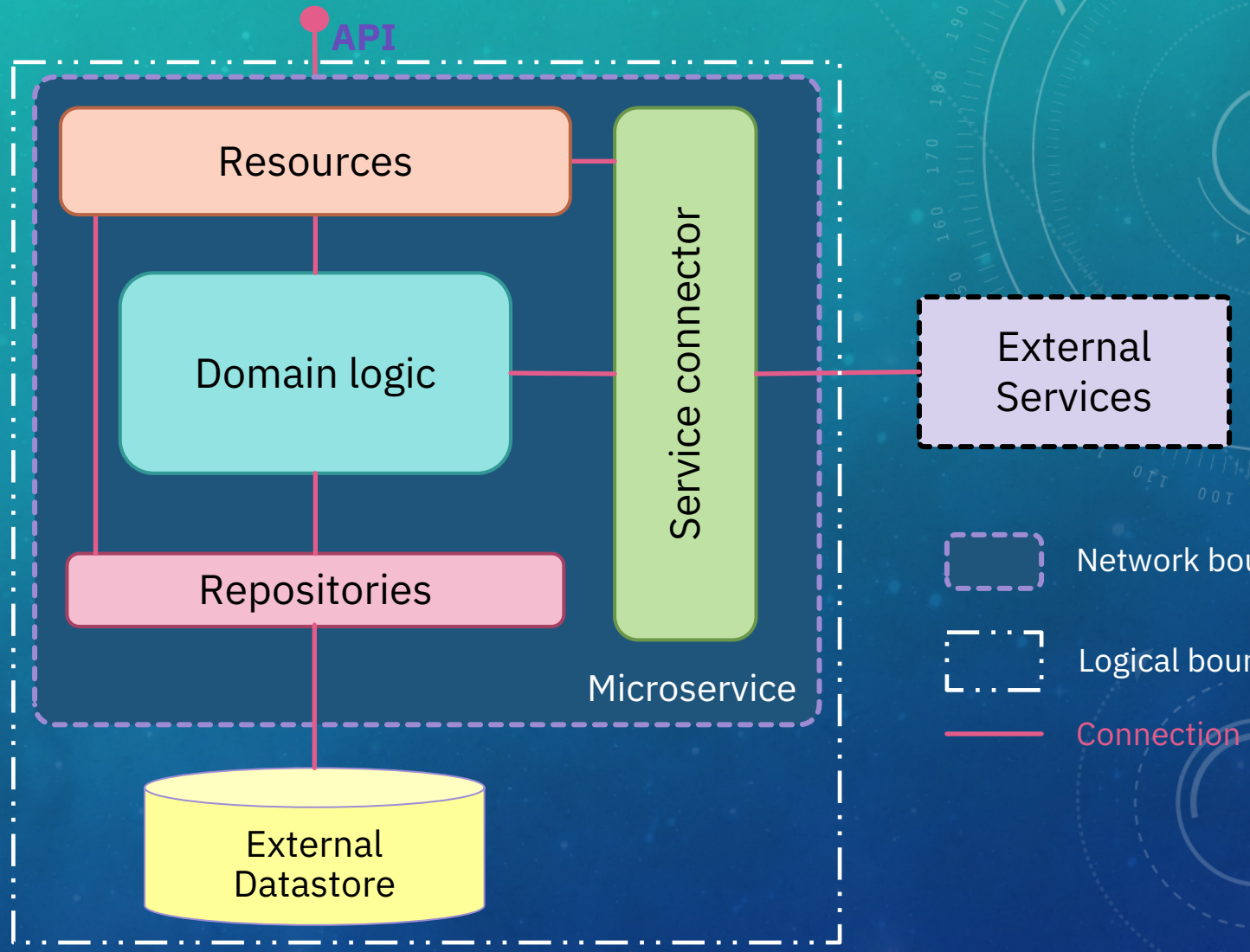


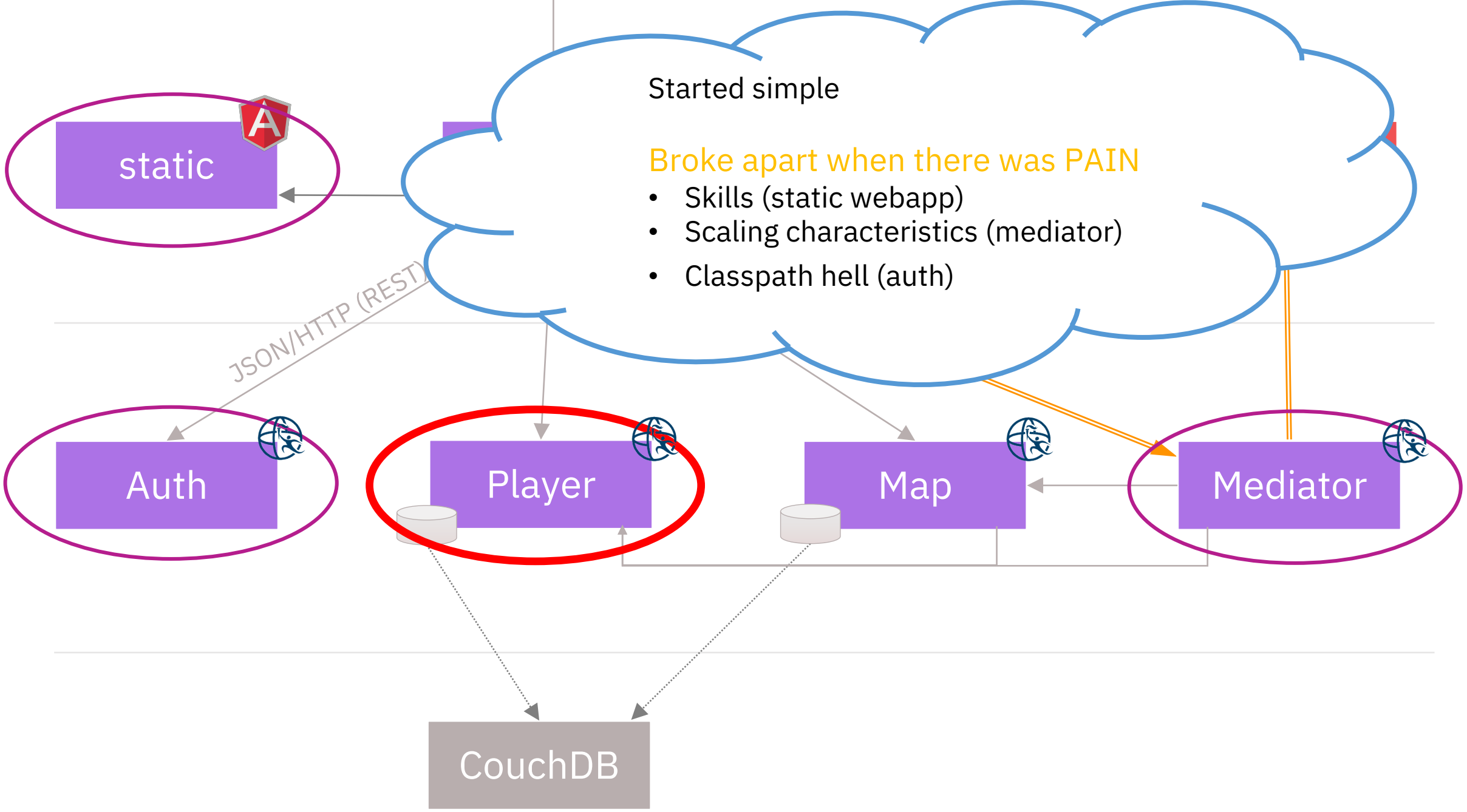

- CouchDB / Cloudant were better suited for Map
- For consistency/simplicity, we switched player over, too

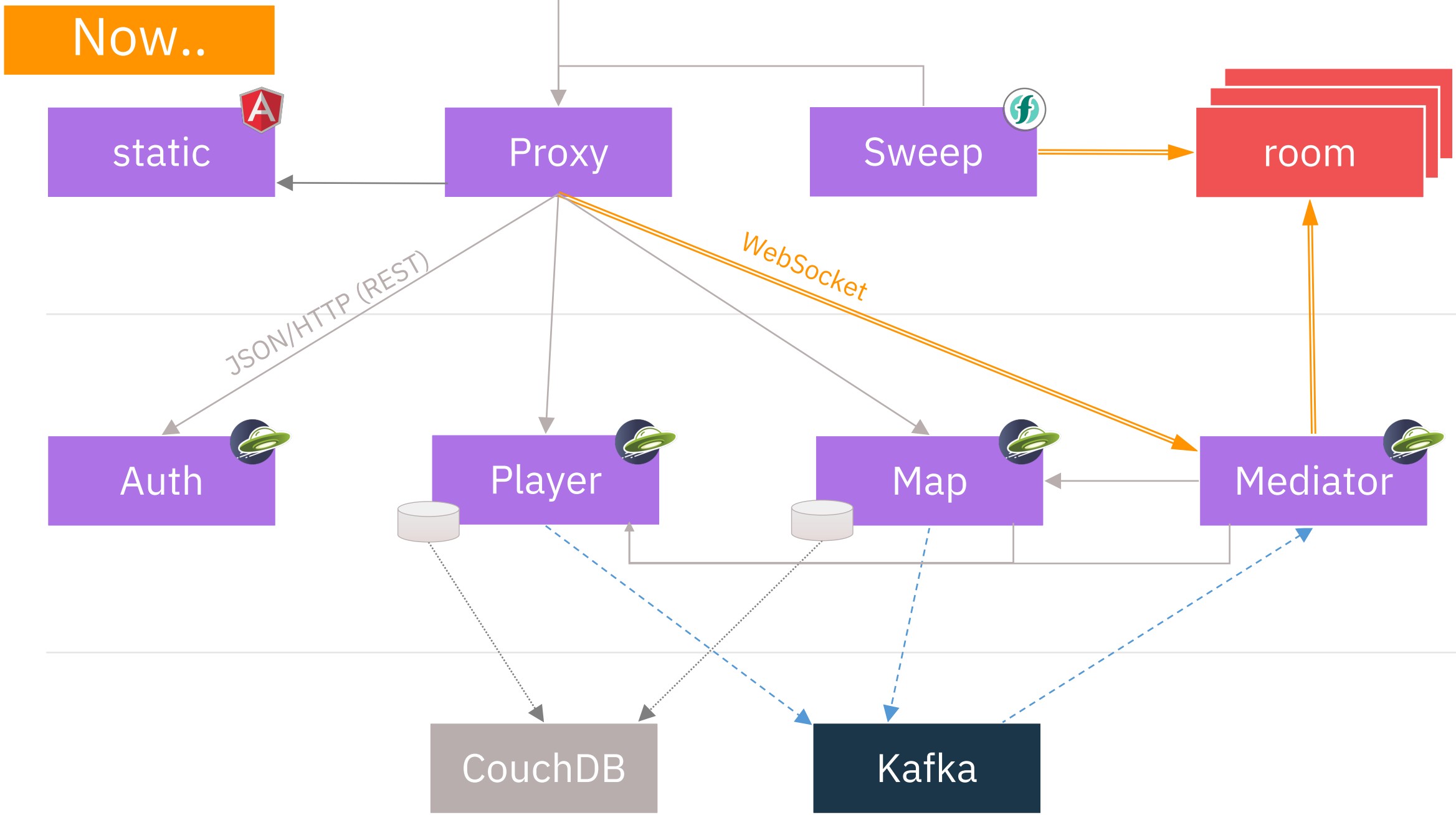
# THE ANATOMY OF A MICROSERVICE



# THE ANATOMY OF A MICROSERVICE




Started simple
Broke apart when there was PAIN
• Skills (static webapp)
• Scaling characteristics (mediator)
• Classpath hell (auth)
static
JSON/HTTP (REST)
Auth
Player
Map
Mediator
CouchDB


Now..
static
Proxy
Sweep
room
JSON/HTTP (REST)
WebSocket
Auth
Player
Map
Mediator
CouchDB
Kafka

# CODEBASE
# DEPENDENCIES
# CONFIG

# DEV/PROD PARITY

## 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 + TRAVIS

## Git submodules

- Root /umbrella project for running core services locally
- Bridge between “monolithic” system view and microservices
- RULE: only CI/CD pipeline updates submodule versions
- Scripts (go-admin.sh, go-run.sh) for Docker Compose OR Kubernetes OR Helm

## Similar services use common build scripts

- Common CI/CD scripts stored in root repository
- Simplifies maintenance
- Git Ops – Everything checked into and/or triggered by git

# LOCAL DEVELOPMENT

- Root /umbrella project for running core services locally
  - Scripts (go-admin.sh, go-run.sh)
  - Docker Compose OR Kubernetes OR Helm
- Local overrides for fast iteration with Docker Compose
  - docker-compose.override.xml
  - Volume mounts
- Common base container for liberty image
  - Caching / Patching vulnerabilities

# TL;DR

```
$ git clone https://github.com/gameontext/gameon.git
$ cd gameon                          # cd into the project directory
$ ./go-admin.sh choose               # choose Docker Compose (1)
$ eval $(./go-admin.sh env)          # set aliases for admin scripts
$ alias go-run                       # confirm path (docker)
$ go-run setup
$ go-run up
$ go-run wait                        # make sure things are good to go
```

# E.G. FIXING THE WEB FRONT END (1/2)

Click on the terms link: https://127.0.0.1

```
$ git submodule update --init webapp
$ cd webapp
$ ./build.sh
```

# USING A CONTAINER FOR BUILD

- In gameon/webapp:
  - Dockerfile-node
  - docker/docker-build.sh (npm & gulp)
  - build.sh

- Key elements:
  - Specified user & group to avoid file permission issues
  - Specific volume for installed node modules
    - Avoid fighting over binaries between host and container

# FIXING THE WEB FRONT END (2/2)

- In gameon/webapp directory:
  1. Open app/templates/default.html
  2. Around line 27, remove “templates/” before “terms.html”
- In gameon/docker directory:
  1. Copy docker-compose.override.yml.example → **docker-compose.override.yml**
  2. Open docker-compose.override.yml in editor of choice
  3. Uncomment webapp section

```
$ go-run rebuild webapp
```

→ Refresh browser!

# PROCESSES
# DISPOSABILITY
# CONCURRENCY

## 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

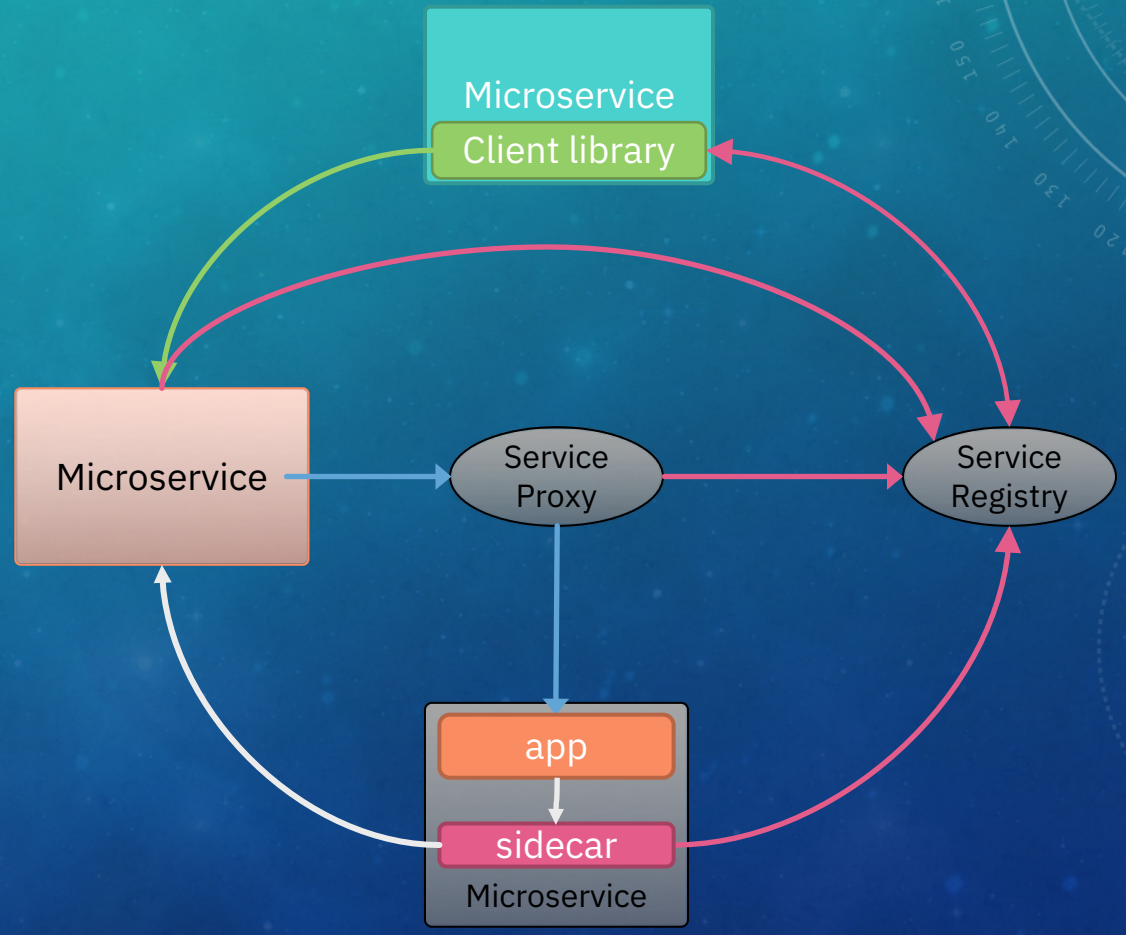
**XI. Logs**
Treat logs as event streams

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

# SERVICE REGISTRATION AND DISCOVERY

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

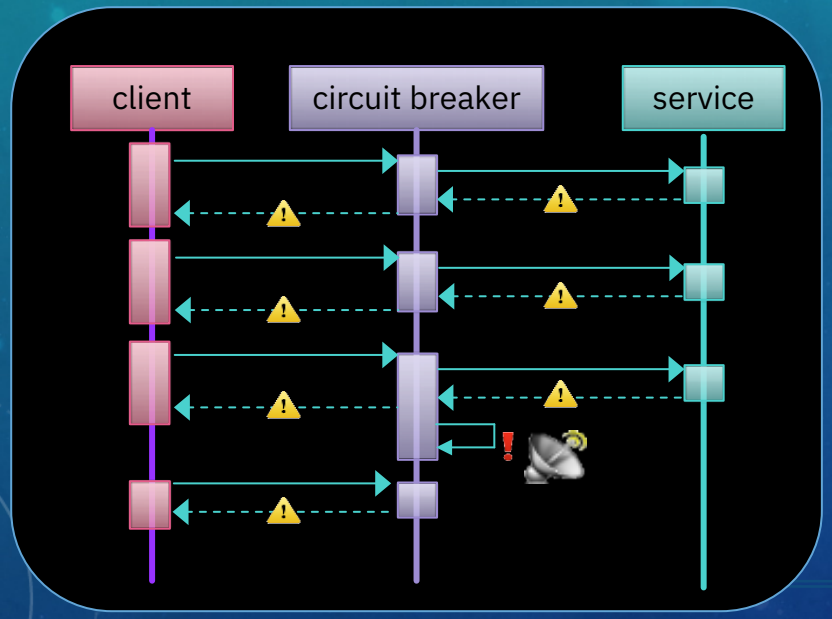
- Client-side or server-side?
- Client library, sidecar, or proxy?



# FAULT TOLERANCE

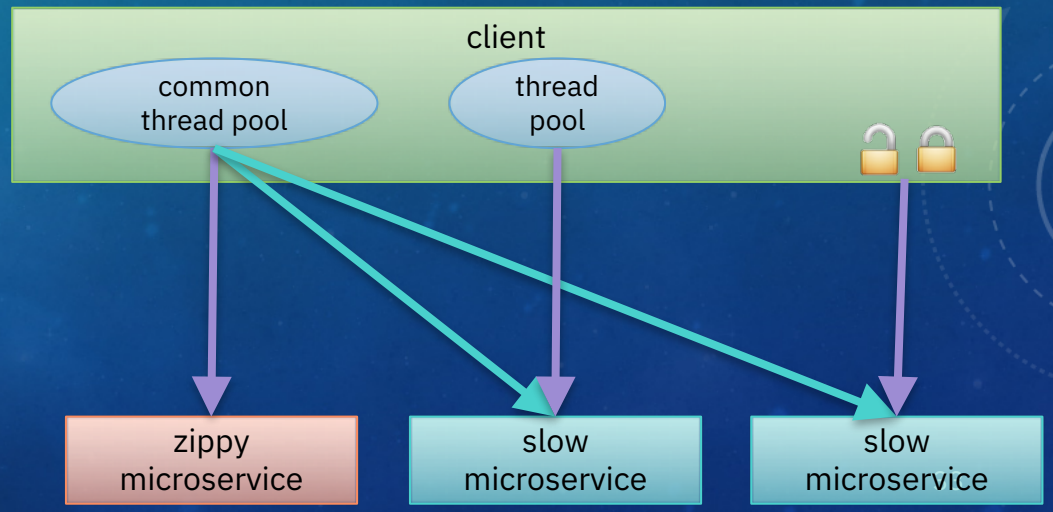
## Circuit Breakers

- Wrap remote calls
- Monitor for failures
- Notify when circuit is tripped
- Retry or Fallback?
- When is circuit reset?




## Bulkheads

- Ensure at most ‘n’ threads waiting for a slow resource
  - Thread isolation
    - With or without a queue
    - Timeout / fallback
  - Semaphore isolation
    - Request sent if lock obtained



# EXAMPLE OF A FALLBACK

**/go <direction: E|W|N|S>**

- Repeat until you reach a “sick room”

Examine things in the room:

- What is the Mediator doing?
- Would something in the infrastructure be able to do this?

Connecting to And if you ask me how I'm feeling... Please hold.

There is a distinctly green tinge to this room. It doesn't seem like itself at all. (1)

**And if you ask me how I'm feeling..**

---

A hasty message has been taped to the wall:

> Not feeling well, I've gone to lie down
> -- And if you ask me how I'm feeling..

This room is not healthy.

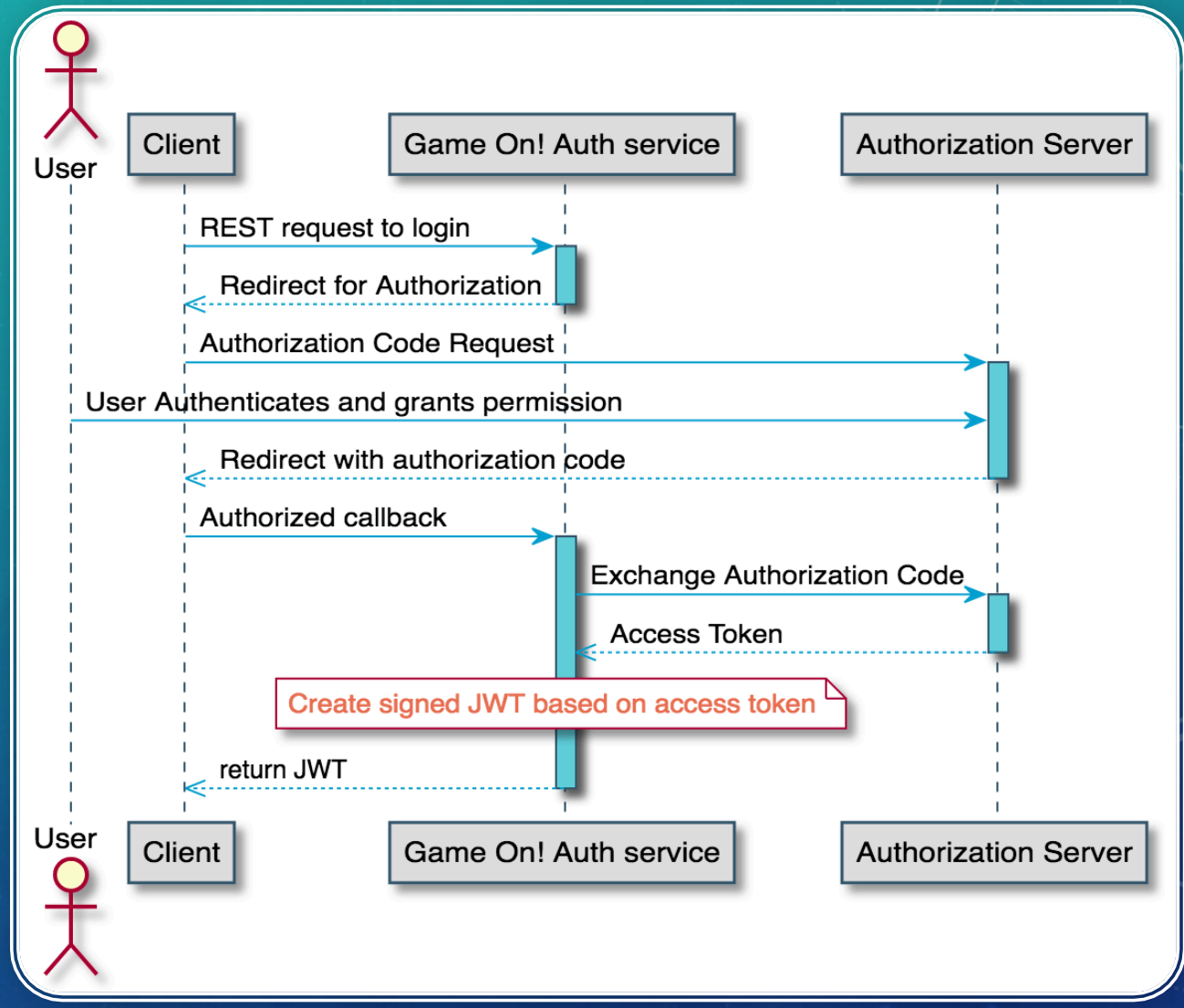
You notice:

- Monitor
- Chart

The room emits a low and rhythmic rumble, like a congested chest. Is it breathing? (2)

# 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

Shared Library



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

# REFERENCES

- Game On! https://gameontext.org/#/
  - Evolution of the Architecture: https://book.gameontext.org/chronicles/
- IBM Redbooks: Microservices Best Practices for Java http://www.redbooks.ibm.com/abstracts/sg248357.html?Open
- Toby Clemson: Testing Strategies in a Microservice Architecture https://martinfowler.com/articles/microservice-testing/#anatomy-modules
- Istio: https://istio.io/
  - Istio Distributed Tracing: https://istio.io/docs/tasks/telemetry/distributed-tracing.html
- Kubernetes: https://kubernetes.io/
  - https://brancz.com/2018/01/05/prometheus-vs-heapster-vs-kubernetes-metrics-apis/