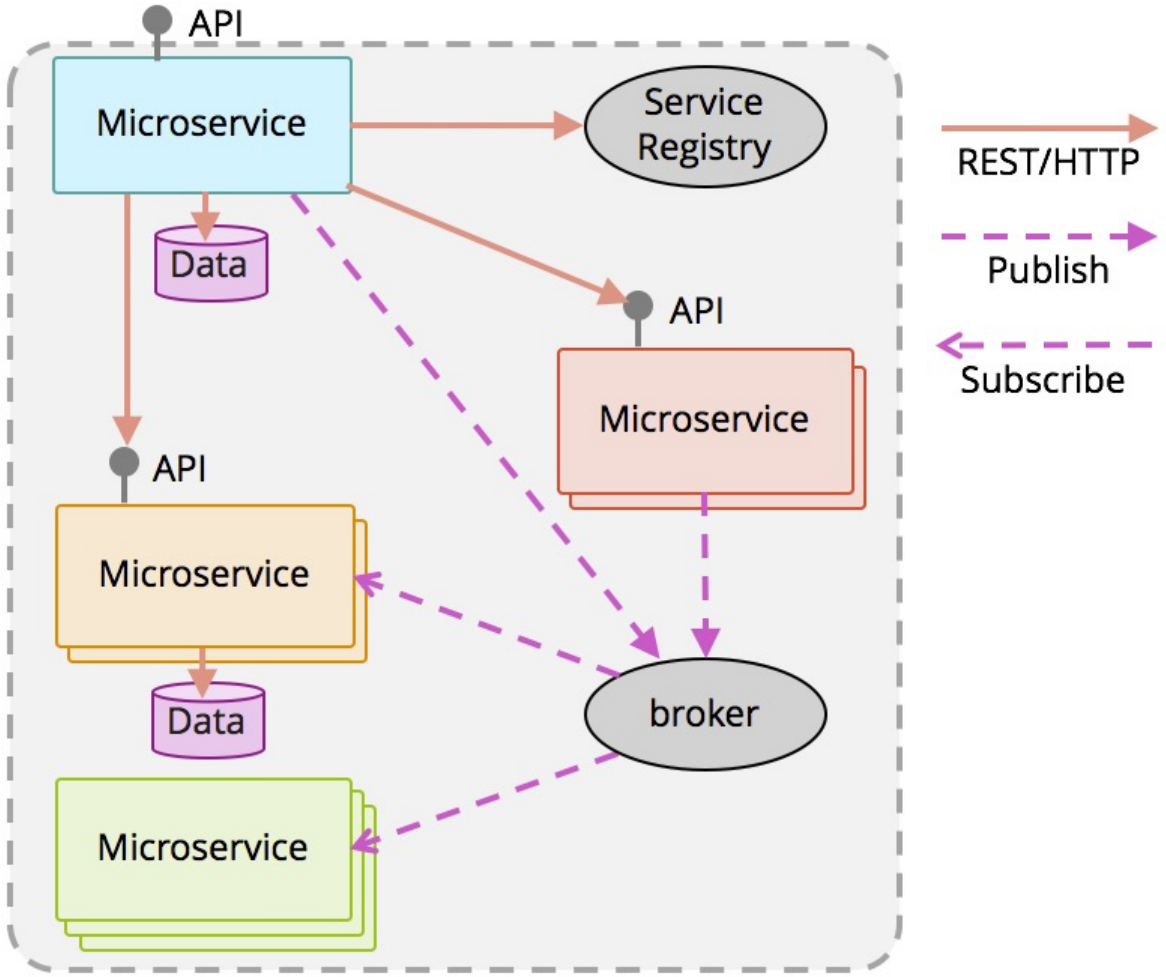
Monolithic Application to Microservice Architecture: How to Break Down Your Existing Applications

Erin Schnabel
@ebullientworks

InterConnect
2017

Microservices are used to…

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



Why microservices?

Microservices provide benefits…

- Strong Module Boundaries: Microservices reinforce modular structure, which is particularly important for larger teams.

- Independent Deployment: Simple services are easier to deploy, and since they are autonomous, are less likely to cause system failures when they go wrong.

- Technology Diversity: With microservices you can mix multiple languages, development frameworks and data-storage technologies.

…but come with costs

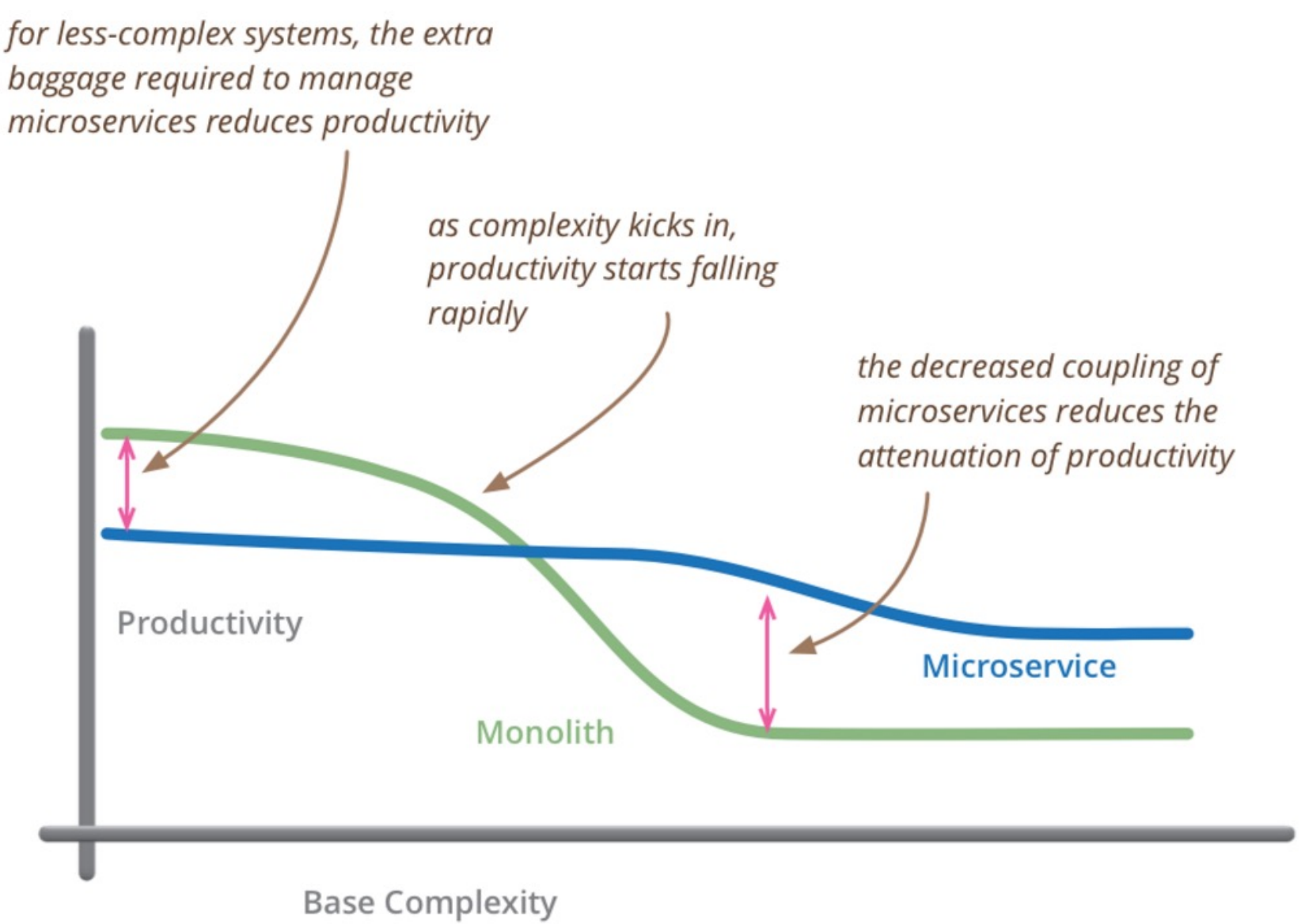
- Distribution: Distributed systems are harder to program, since remote calls are slow and are always at risk of failure.

- Eventual Consistency: Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency.

- Operational Complexity: You need a mature operations team to manage lots of services, which are being redeployed regularly.

http://martinfowler.com/articles/microservice-trade-offs.html

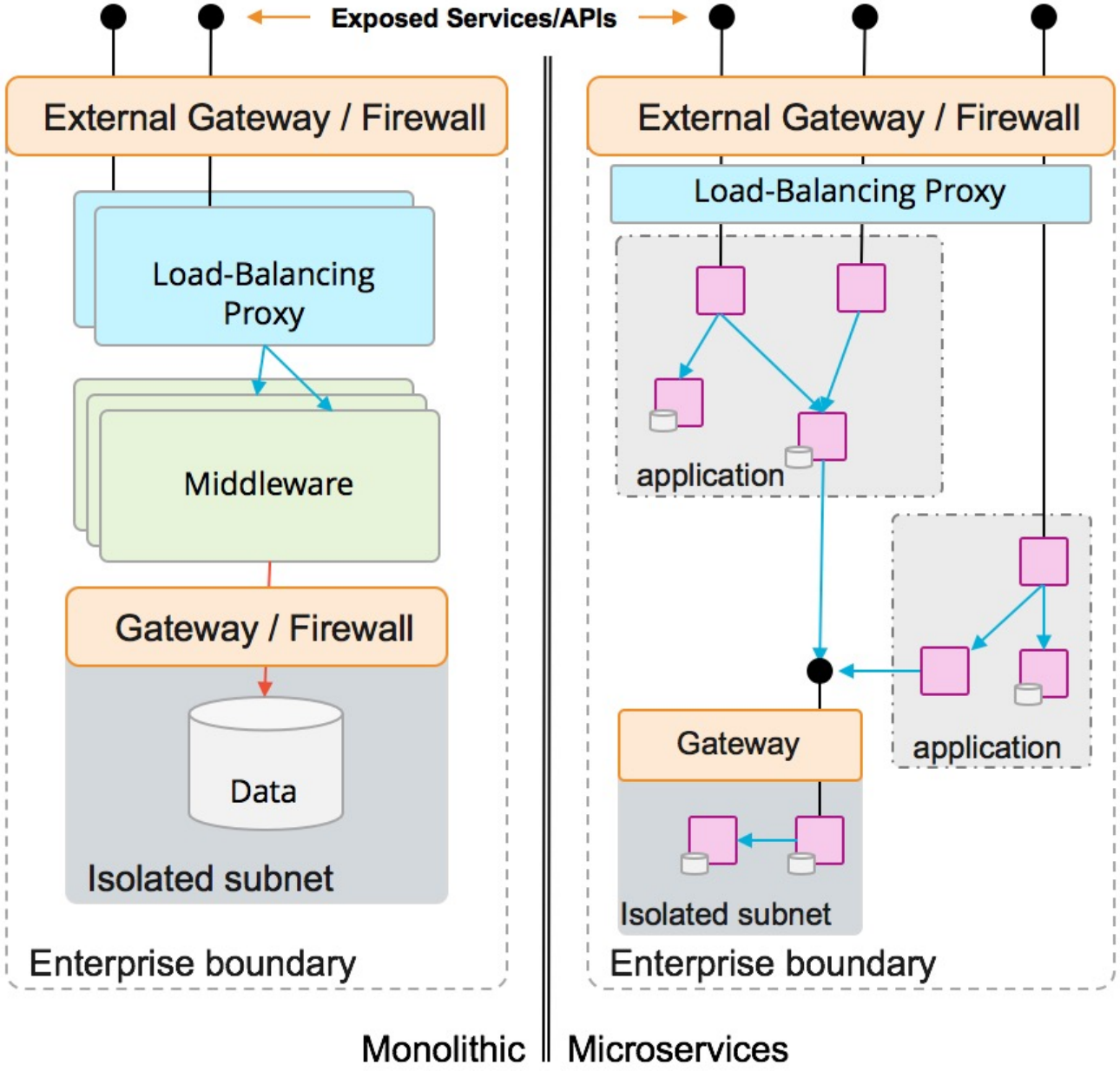
The Premium



but remember the skill of the team will outweigh any monolith/microservice choice

http://martinfowler.com/bliki/MicroservicePremium.html

Monolith vs. Microservice



Should we use microservices?

- Resource utilization
  - Conflicting scaling requirements
- Lifecycle mismatch
  - New vs. old
  - Some parts of application change more frequently
  - Coordinating rollouts introduces delay

Are you ready?

- DevOps / Infrastructure
  - Build pipelines / toolchains
  - Deployment environment (and all that entails)
    - Bluemix (PaaS or Containers)
    - IBM Spectrum CfC / Microservices Builder (Beta)
    - Liberty Collectives, or Liberty in existing WAS ND

Configuration for microservices

Make configuration part of your DevOps process:

- Treat infrastructure as code, such as in a software-defined data center
- All application deployment must be automated
- Start small with simple scripts and build from there

Refactoring an existing application

- Each REST service is potentially a microservice

- Each SOAP web service or EJB is potentially a microservice
  - Especially stateless session beans
  - Redesign function-oriented interfaces to asset-oriented interfaces
  - Make the interface RESTful, such as using command objects

- Use domain-driven design to discover your assets, which might be microservices

Use tools to analyze your application

Download

Migration Toolkit for Application Binaries

ASSET TYPE: TOOL

The Migration Toolkit for Application Binaries provides a command line tool that quickly evaluates application binaries for rapid deployment on newer versions of WebSphere Application Server traditional or Liberty.

Application Evaluation Report

Identifies the Java™ EE programming models in the application and provides a recommendation for the right-fit IBM WebSphere Application Server edition.

Application Inventory Report

Contains a high-level inventory of the content and structure of each application which can be useful in determining the complexity of applications. It also displays information about potential deployment problems and performance considerations.

Detailed Migration Analysis Report

Highlights Java EE programming model and WebSphere API differences between the profile types. This report contains advice and potential solutions to assess the ease of moving applications to Liberty or to newer versions of WebSphere traditional. It also informs you of any Java EE specification implementation differences that could affect your applications.

The tool can either display an HTML or text report or save the report as an HTML, JSON, or text file.

https://developer.ibm.com/wasdev/downloads/#asset/tools-Migration_Toolkit_for_Application_Binaries

Analyze your monolith

Application Inventory Report

8/23/16 10:55 AM

/apps/PlantsByWebSphereV7.ear
/apps/WebServicesExperiment.war

Jump To Application

1	3	0	1	2	0
EAR files	WAR files	RAR files	EJB JAR files	Utility JAR files	Application client JAR files

Summary

Java Servlets	4
JSP files	13
JPA entities	7
BMP entity beans	0
CMP entity beans	0
Message-driven beans	0
Singleton session beans	0
Stateful session beans	1
Stateless session beans	8
Web Services	1

Inventory Details by Application

Expand all | Collapse all

PlantsByWebSphereV7.ear Show details

Refactor your data

- Refactor your data for the microservices' code
    - Each microservice manages its own data
        - YES, REALLY!
    - Refactor the tables to be used by only one module

Optimize for query time, not storage efficiency

Refactoring data for microservices

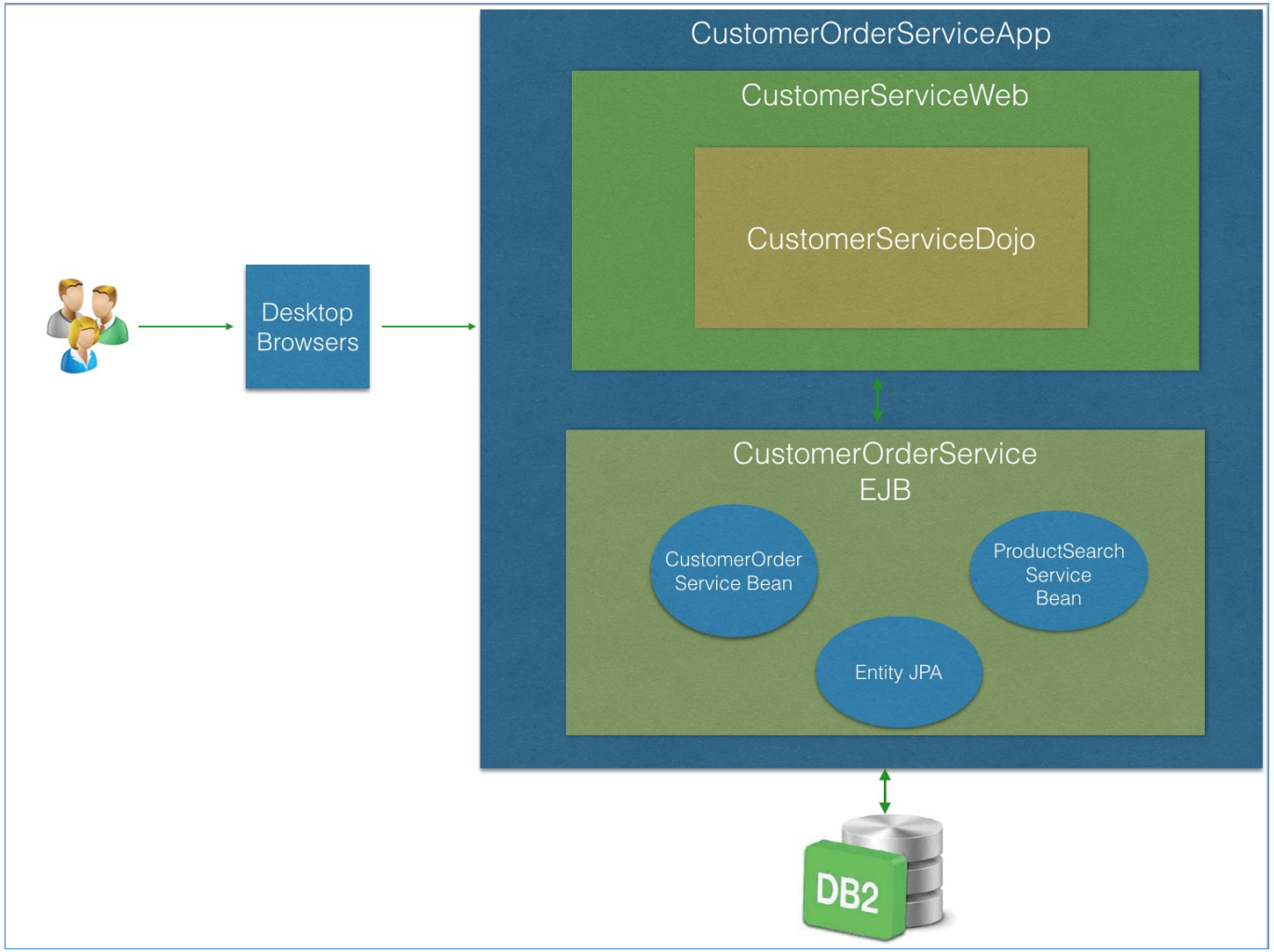
- Master data management
  - Form a single, consistent view of widely used data entities
    - Domain Driven Design
  - Develop microservices to work with that

- For code storing blobs in your SQL database
  - Store those objects in a No SQL database instead
  - Key-value store, such as Redis or Data Cache

- Active Record pattern: Flat objects unrelated to others
  - Islands of data unrelated to other data
  - Use a document model database, such as Cloudant or Mongo

Implementation Example: Shopping Application

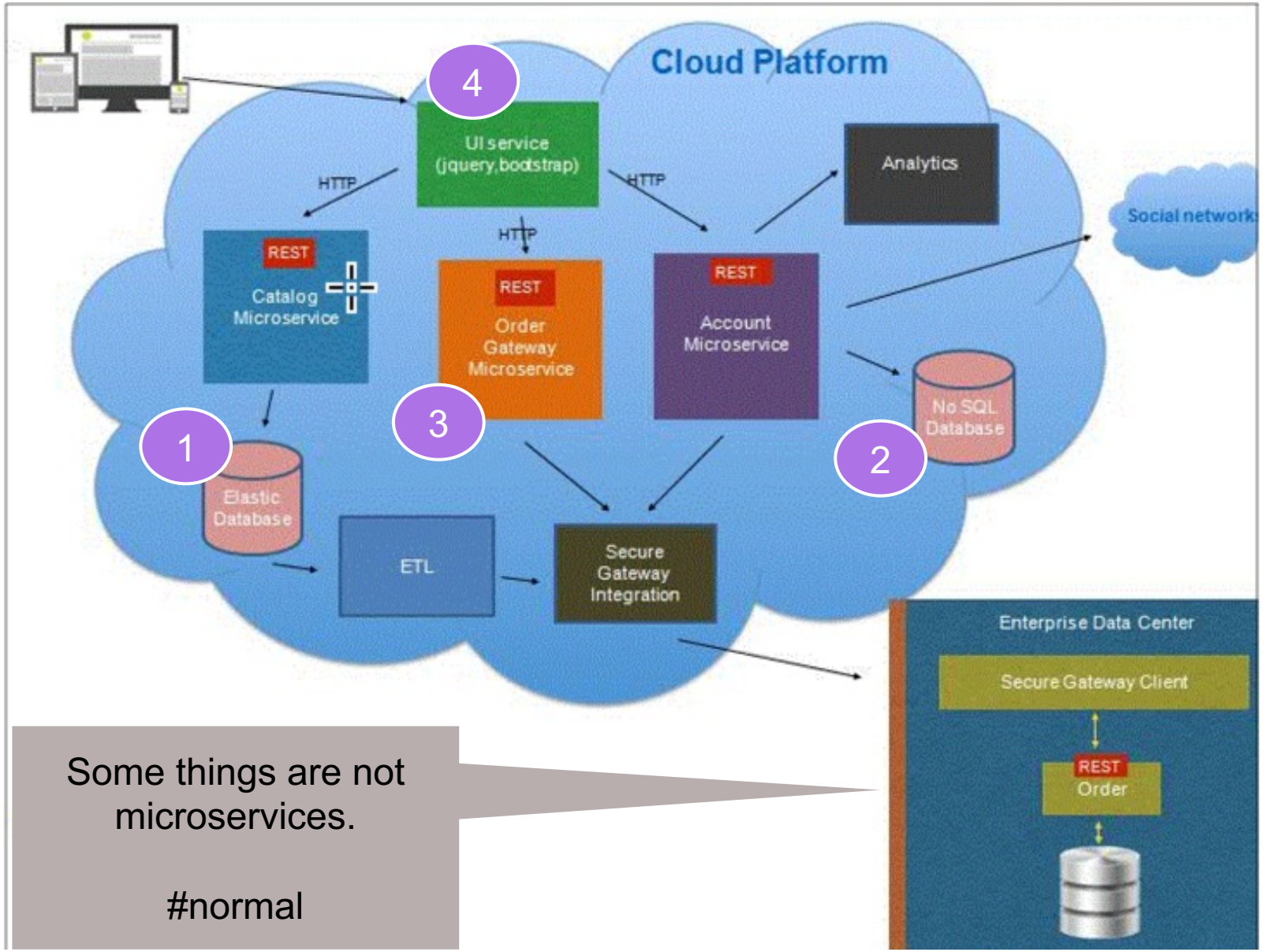
- Current Architecture
  - Single Relational DB Schema
    - Products, Customer, etc
  - Single EAR file
    - EJB and JPA persistence
    - Dojo Web Application
- Business Problems
  - Limited search capability
  - Not focused on external customer experience
  - Complex ordering system
    - difficult to add product and customer analytics to site without breaking Order System.



New Architecture

1. Catalog data imported into Elasticsearch
   - Fuzzy!
2. Customer data modeled and stored in Document NoSQL store with analytic and social data.
3. Order microservice wraps on-prem ordering and uses integration.
4. New Mobile App uses new microservices.

Existing Website used with routing / Strangler pattern to evolve.



Some things are not microservices.

#normal

Evolving to microservices

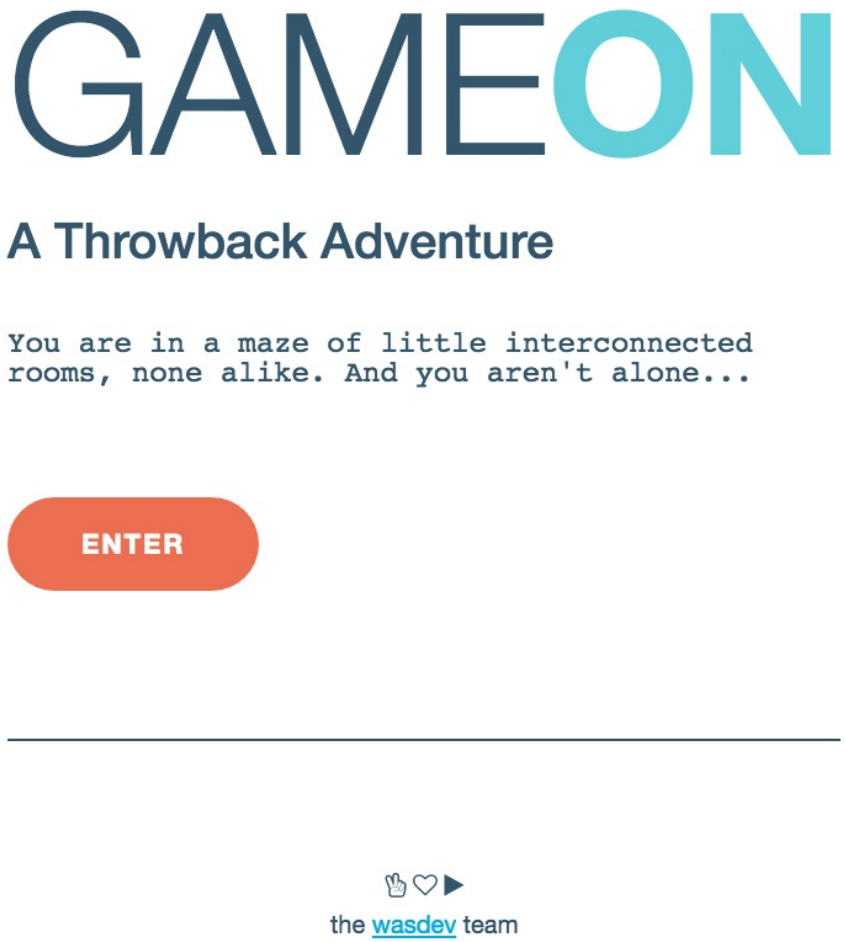
- Agile development: What's the simplest thing that could possibly work?
  - A monolith is simpler

- Start with a monolith or a duolith pair
  - Make it modular, and plan that modules can become microservices
  - Each module should be a vertical slice, a mini-app with its own data

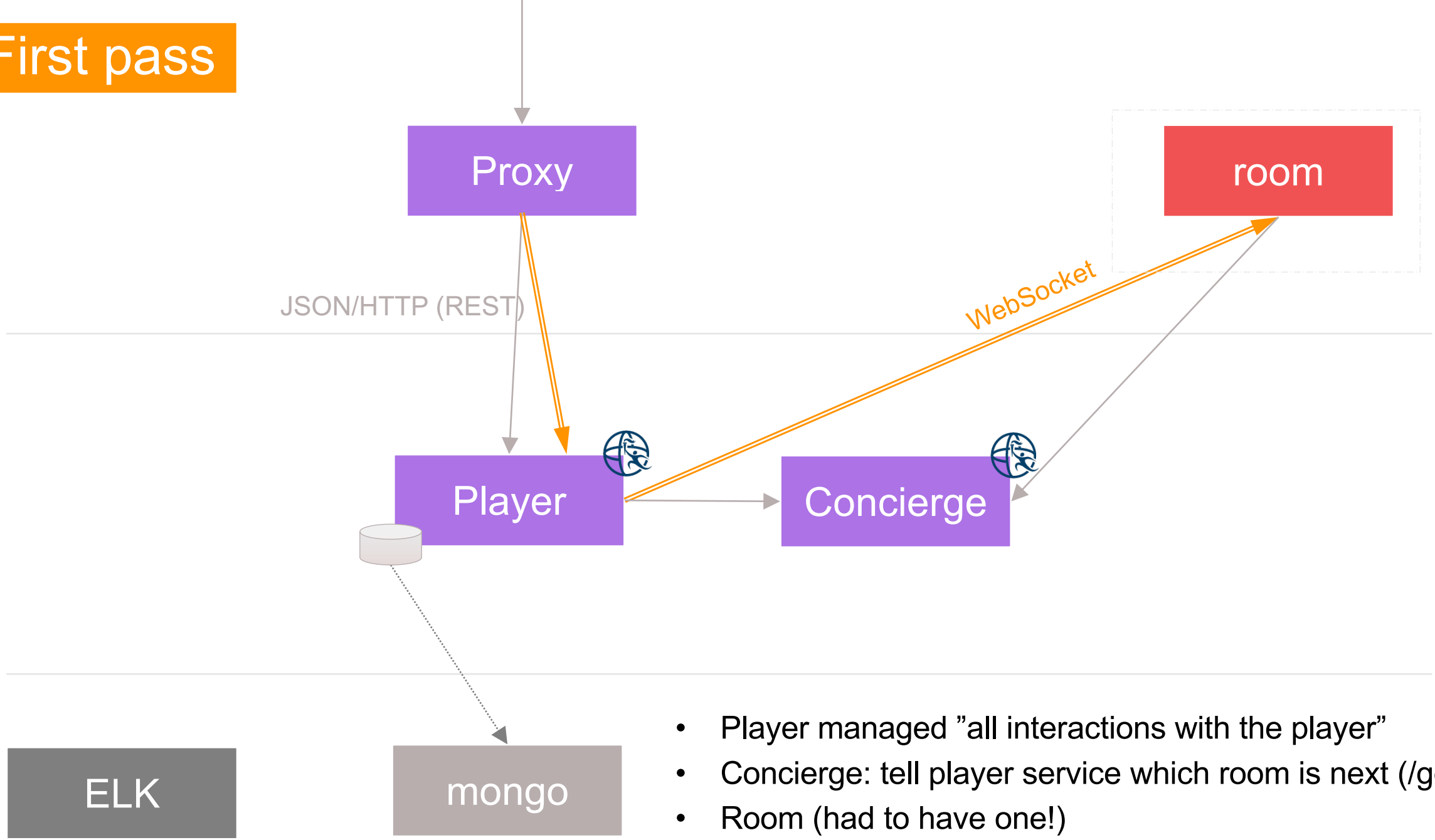
- Start with a minimally viable product (MVP)
  - MVP is a module
  - Improvements to existing features go in the same module
  - Implement new functional tasks in new modules
  - When a module implements more than one task, refactor into separate modules

The premise …

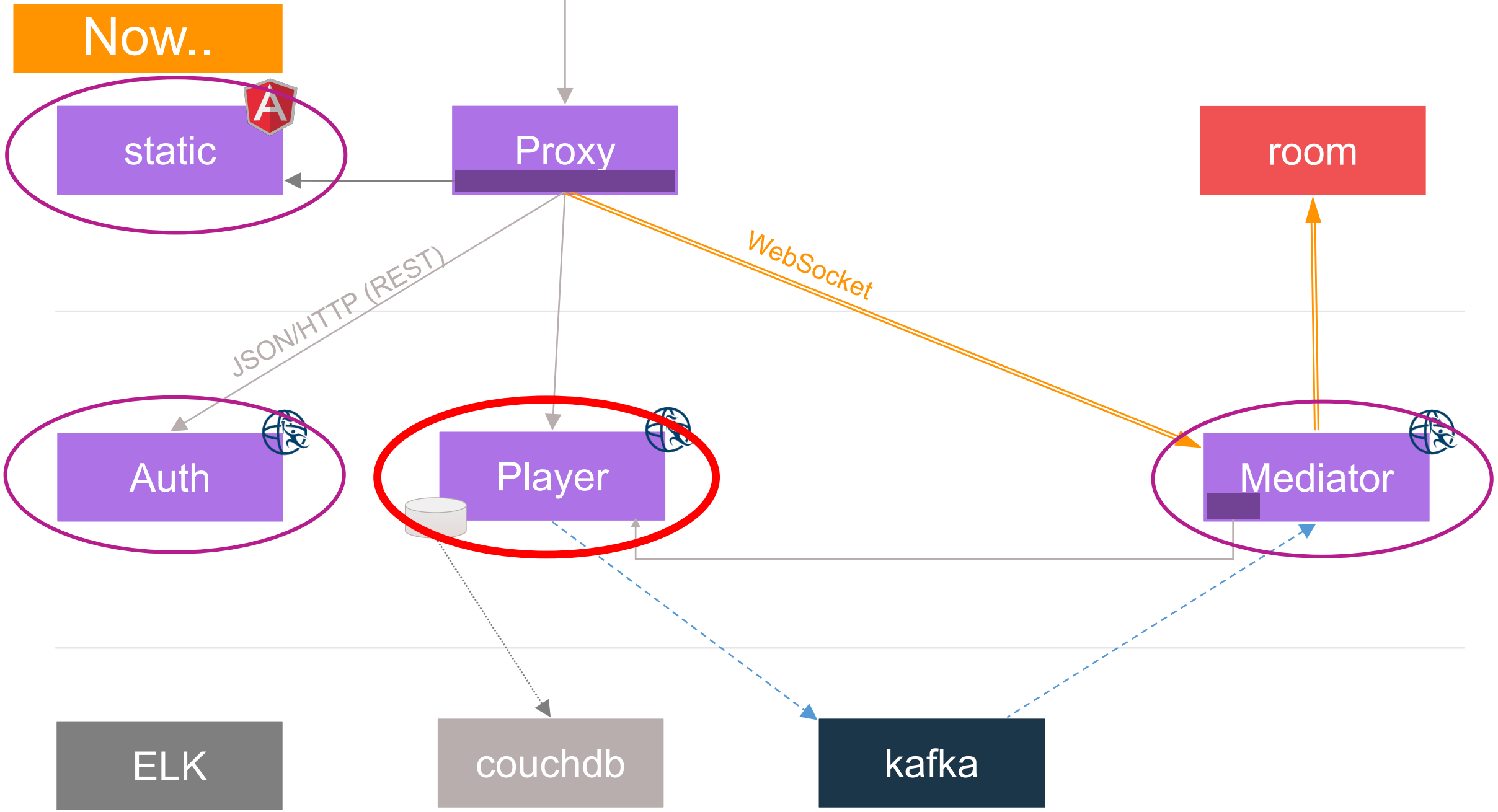
- Hands on with microservices
- Stick with ’Hello World’ simplicity
- Choose your own adventure
- Fast path to the hard stuff
- Build something cool (to you!)
- Learn as you go



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
room
JSON/HTTP (REST)
WebSocket
Auth
Player
Mediator
ELK
couchdb
kafka

Summary

- Don’t treat a happy monolith like a piñata
- Keep things coarse if you’re starting fresh
- Focus on business goals and alleviating pain if refactoring

Notices and disclaimers

Notices and disclaimers continued

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products. IBM does not warrant the quality of any third-party products, or the ability of any such third-party products to interoperate with IBM's products. **IBM expressly disclaims all warranties, expressed or implied, including but not limited to, the implied warranties of merchantability and fitness for a particular, purpose.**

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents, copyrights, trademarks or other intellectual property right.

IBM, the IBM logo, ibm.com, Aspera®, Bluemix, Blueworks Live, CICS, Clearcase, Cognos®, DOORS®, Emptoris®, Enterprise Document Management System™, FASP®, FileNet®, Global Business Services®, Global Technology Services®, IBM ExperienceOne™, IBM SmartCloud®, IBM Social Business®, Information on Demand, ILOG, Maximo®, MQIntegrator®, MQSeries®, Netcool®, OMEGAMON, OpenPower, PureAnalytics™, PureApplication®, pureCluster™, PureCoverage®, PureData®, PureExperience®, PureFlex®, pureQuery®, pureScale®, PureSystems®, QRadar®, Rational®, Rhapsody®, Smarter Commerce®, SoDA, SPSS, Sterling Commerce®, StoredIQ, Tealeaf®, Tivoli® Trusteer®, Unica®, urban{code}®, Watson, WebSphere®, Worklight®, X-Force® and System z® Z/OS, are trademarks of International Business Machines Corporation, registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at: www.ibm.com/legal/copytrade.shtml.

InterConnect
2017