Best practices for creating secure microservices

Erin Schnabel
@ebullientworks

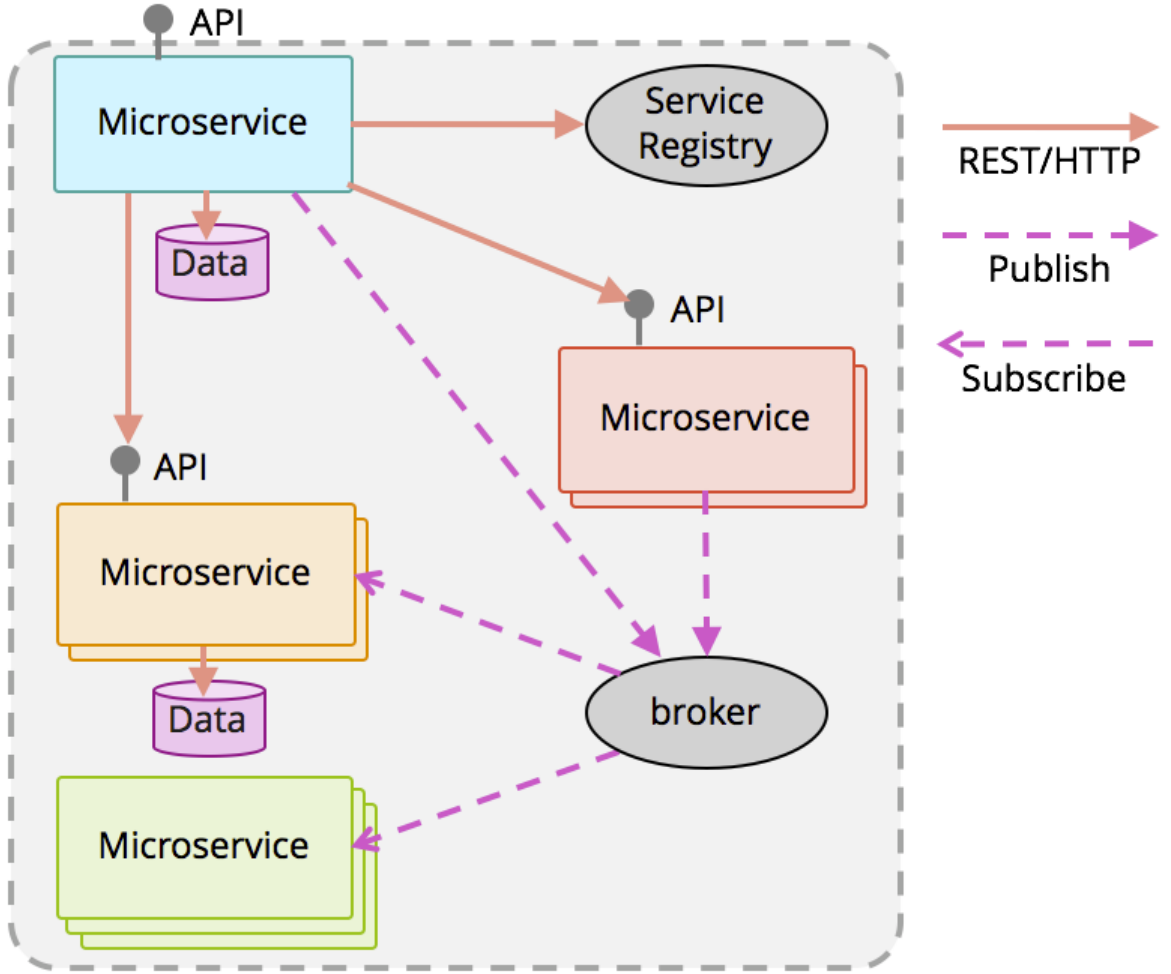
InterConnect
2017

Agenda

- Microservices
- Security Challenges
- Technologies
- Examples

Microservices are used to…

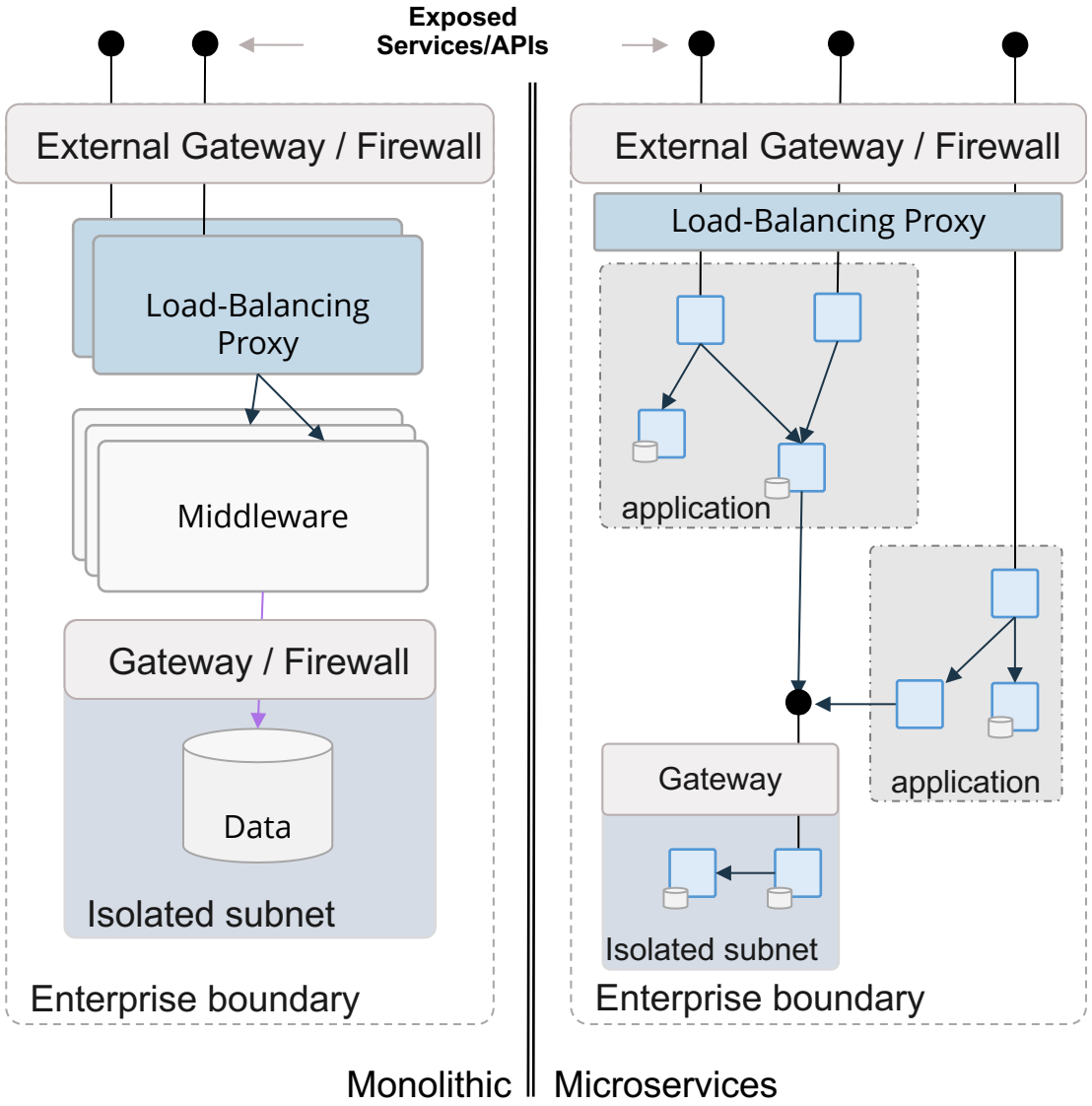
- compose a *complex application* using
  - "small"
  - independent (autonomous)
  - replaceable
  - processes
- that communicate via
  - language-agnostic APIs



Security Challenges

API Surface Explosion

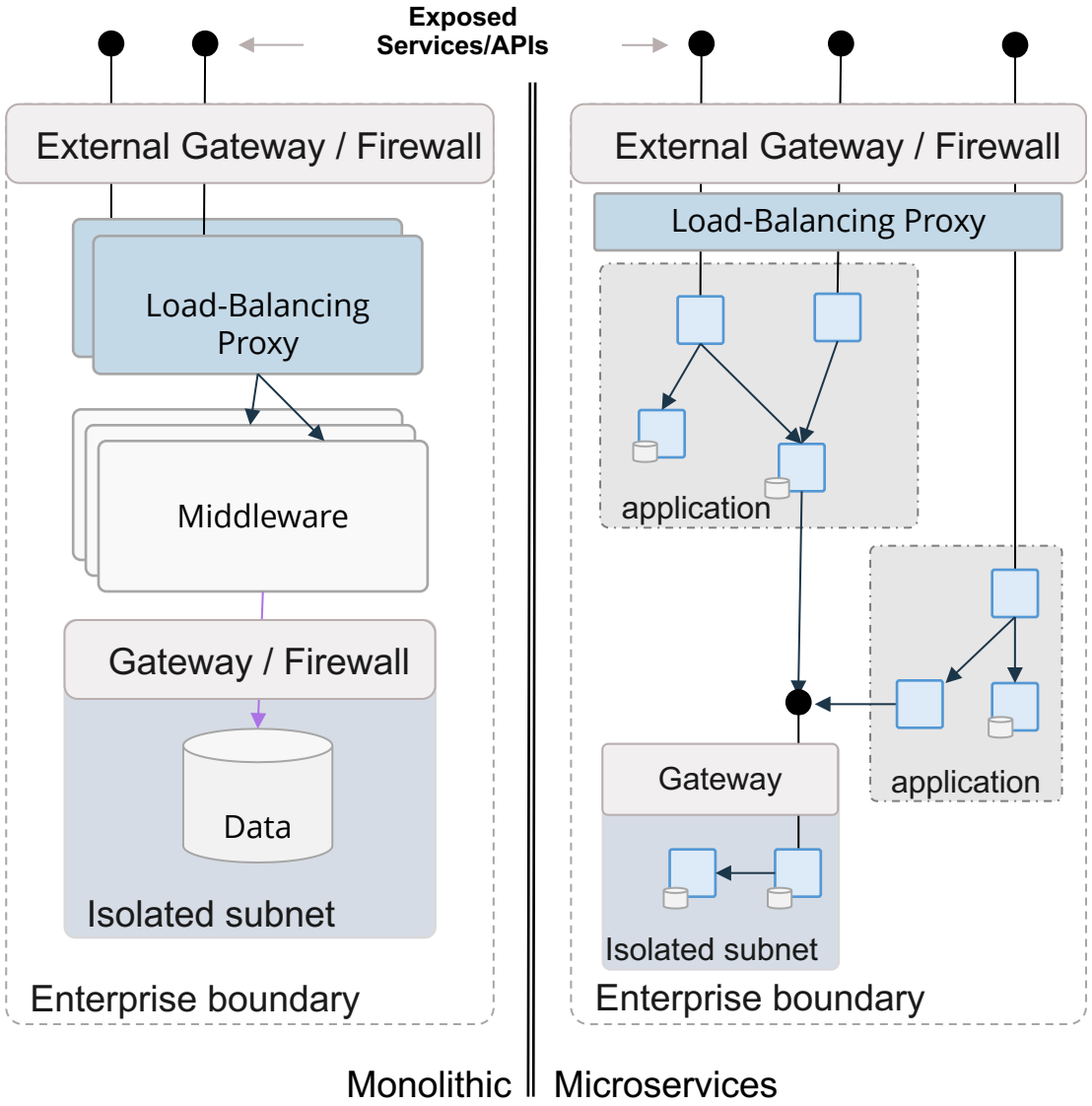
- Application composed of services
- Each service with its own API
- Each service with it's own security concerns.



Microservices Best Practices for Java, an IBM Redbooks publication
http://www.redbooks.ibm.com/redpieces/abstracts/sg248357.html?Open

Service to Service

- Services care about originating ID
- Services need to establish trust
  - Who is invoker?
- Avoid bottlenecks
  - E.g. calls to centralized service



Microservices Best Practices for Java, an IBM Redbooks publication
http://www.redbooks.ibm.com/redpieces/abstracts/sg248357.html?Open

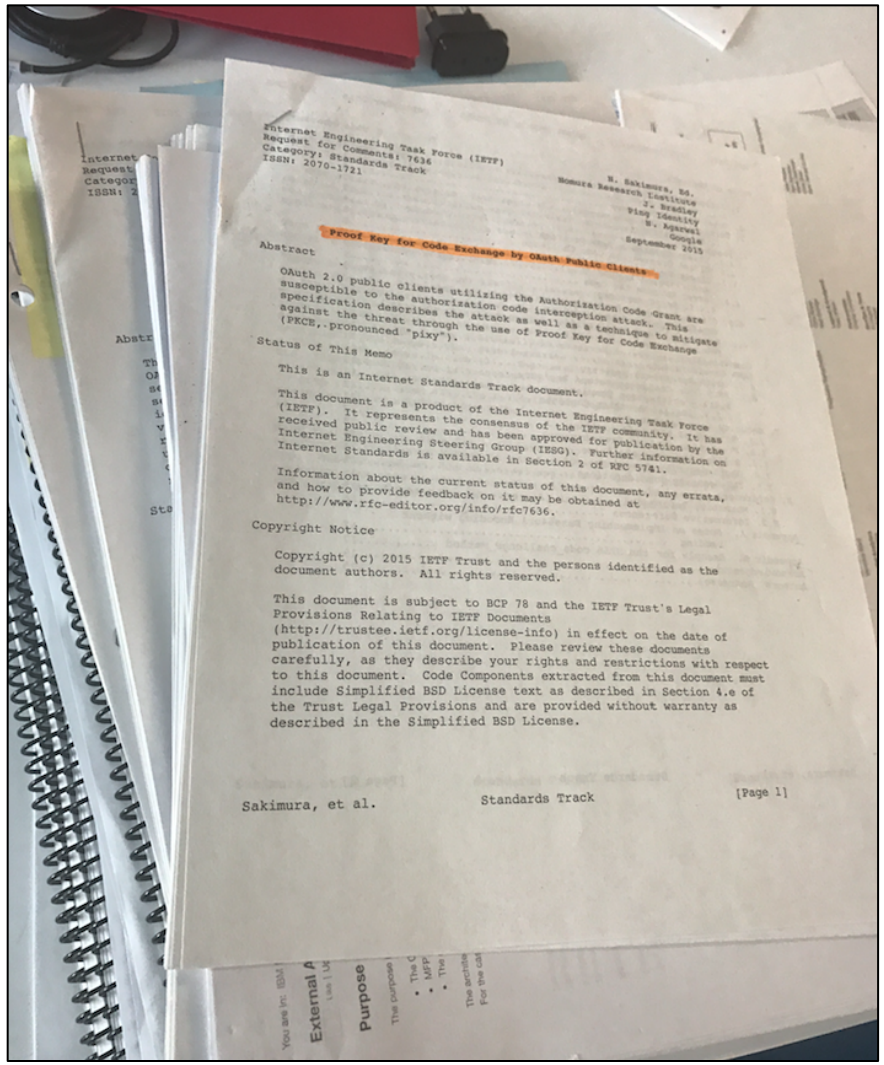
Technologies

Identity, authorization and authentication are not piece of cake

!=

Care to read some RFCs?

1. RFC 6749 - The OAuth 2.0 Authorization Framework
2. RFC 6750 - The OAuth 2.0 Authorization Framework: Bearer Token Usage
3. Open ID Connect
    1. Core
    2. Discovery
    3. Dynamic registration
4. RFC 7591 - OAuth 2.0 Dynamic Client Registration Protocol
5. RFC 7592 - OAuth 2.0 Dynamic Client Registration Management Protocol
6. RFC 7518 - JSON Web Algorithm (JWA)
7. RFC 7519 - JSON Web Token (JWT)
8. RFC 7515 - JSON Web Signature (JWS)
9. RFC 7636 - Proof Key for Code Exchange by OAuth Public Clients
10. RFC 7662 - OAuth 2.0 Token Introspection
11. RFC 7521 - Assertion Framework for OAuth 2.0 Client Authentication and Authorization Grants
12. RFC 7523 - JSON Web Token (JWT) Profile for OAuth 2.0 Client Authentication and Authorization Grants
13. RFC 7522 - Security Assertion Markup Language (SAML) 2.0 Profile for OAuth 2.0 Client Authentication and Authorization Grants
14. RFC 7642 - System for Cross-domain Identity Management: Definitions, Overview, Concepts, and Requirements
15. RFC 7643 - System for Cross-domain Identity Management: Core Schema
16. RFC 7644 - System for Cross-domain Identity Management: Protocol



Everything revolves around two standards - OAuth2 and OIDC

Authorization

Authentication

OAuth2 / OIDC

- Authentication Server
- Used by most Social Login today
  - (Facebook, Google, GitHub etc)

- Auth request by User results in Access Token
- Access Token supplied by User as part of secured requests
- Access Token validated by invoked Service querying OAuth server.
  - Allows for token invalidation
  - Consider token invalidation mid-request impact on overall request flow.

- Open ID Connect (OIDC) as an identity layer over OAuth2

JWT

- JSON Web Tokens
- JSON Claims + Digital Signature Block
  - Base64 Encoded for transmission.
- Can be encrypted, but usually not.
- Signature can be signed with a Shared Secret, or Public/Private Key.
- Standard time related fields restrict validity period of JWT

- Self Verifying. No need to call external server.

JWK

- JSON Web Key

- Similar to JWT:

- Makes validating with PKI across languages deterministic
  - No obscurity

API Key / Shared Secret

- Common for WebHooks, User “Apps” (Facebook, Google, etc)
- Establish trust because both parties know the same Key/Secret
- Identity can be established in parallel.
  - Request transmits an ID along with a Key or Secret
    - Essentially UserID & Password
  - Request signs something including the ID using the Key / Secret
    - ID is used to recreate signature, verifying the sender knew ID & Secret, establishing trust.
    - Can be used to further protect additional request content, Eg AWS Auth.

Examples

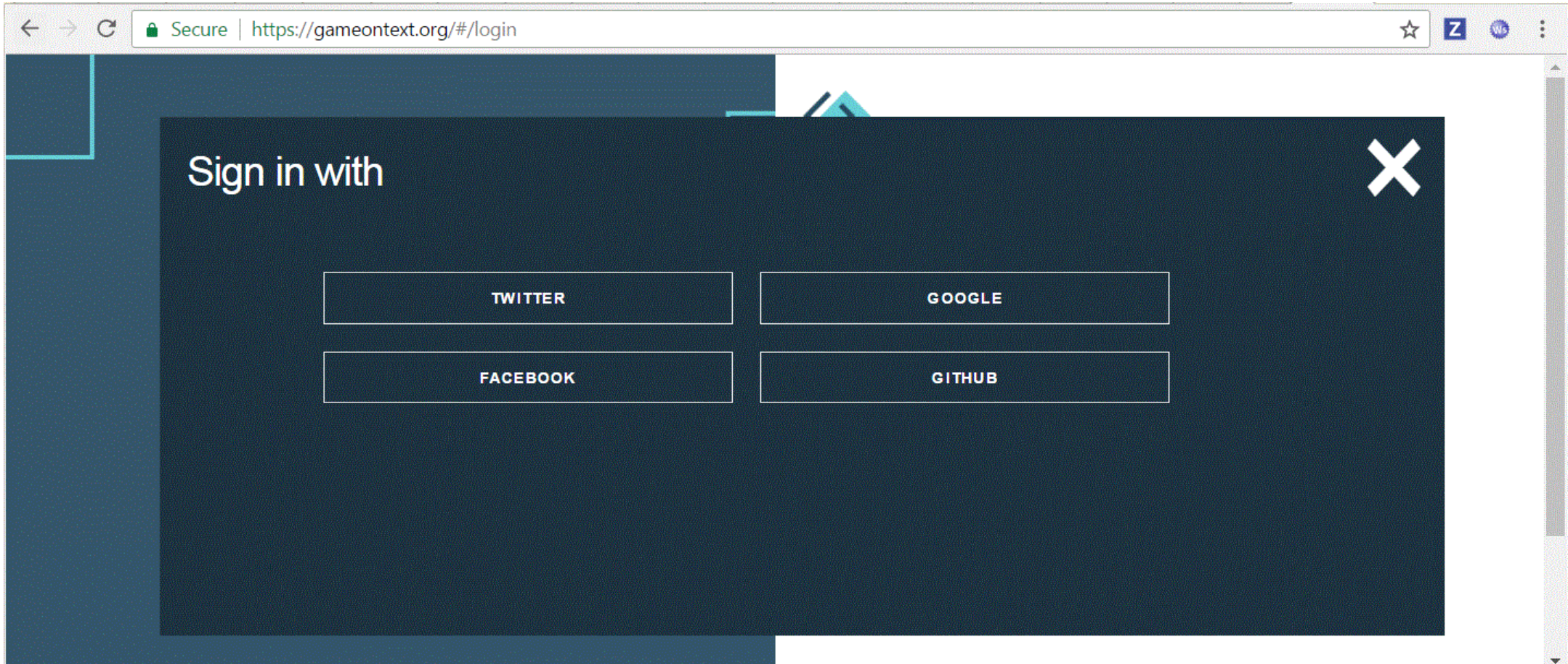
The premise …

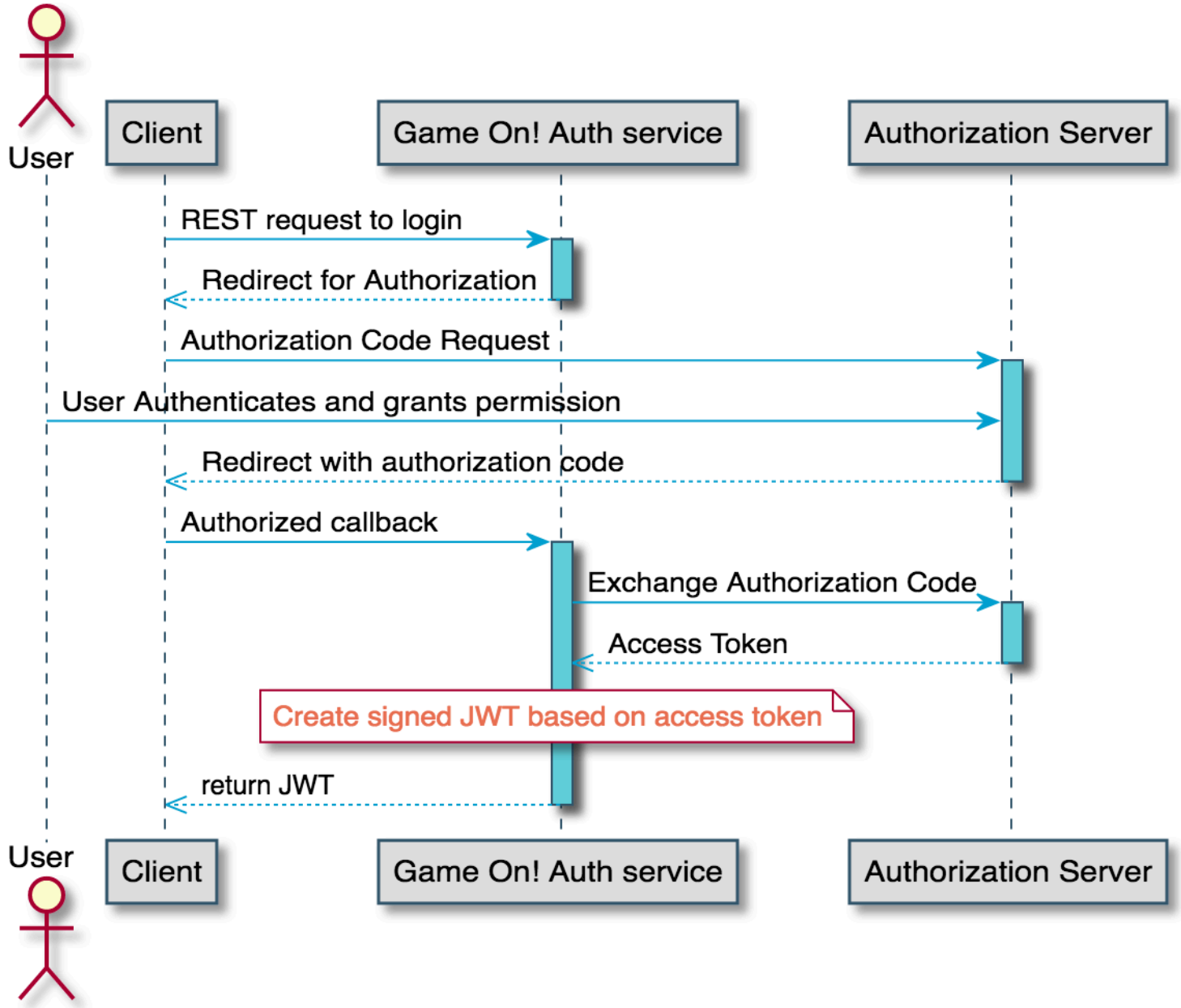
- Hands on with microservices
- Stick with ’Hello World’ simplicity
- Choose your own adventure
- Fast path to the hard stuff
- Build something cool (to you!)
- Learn as you go

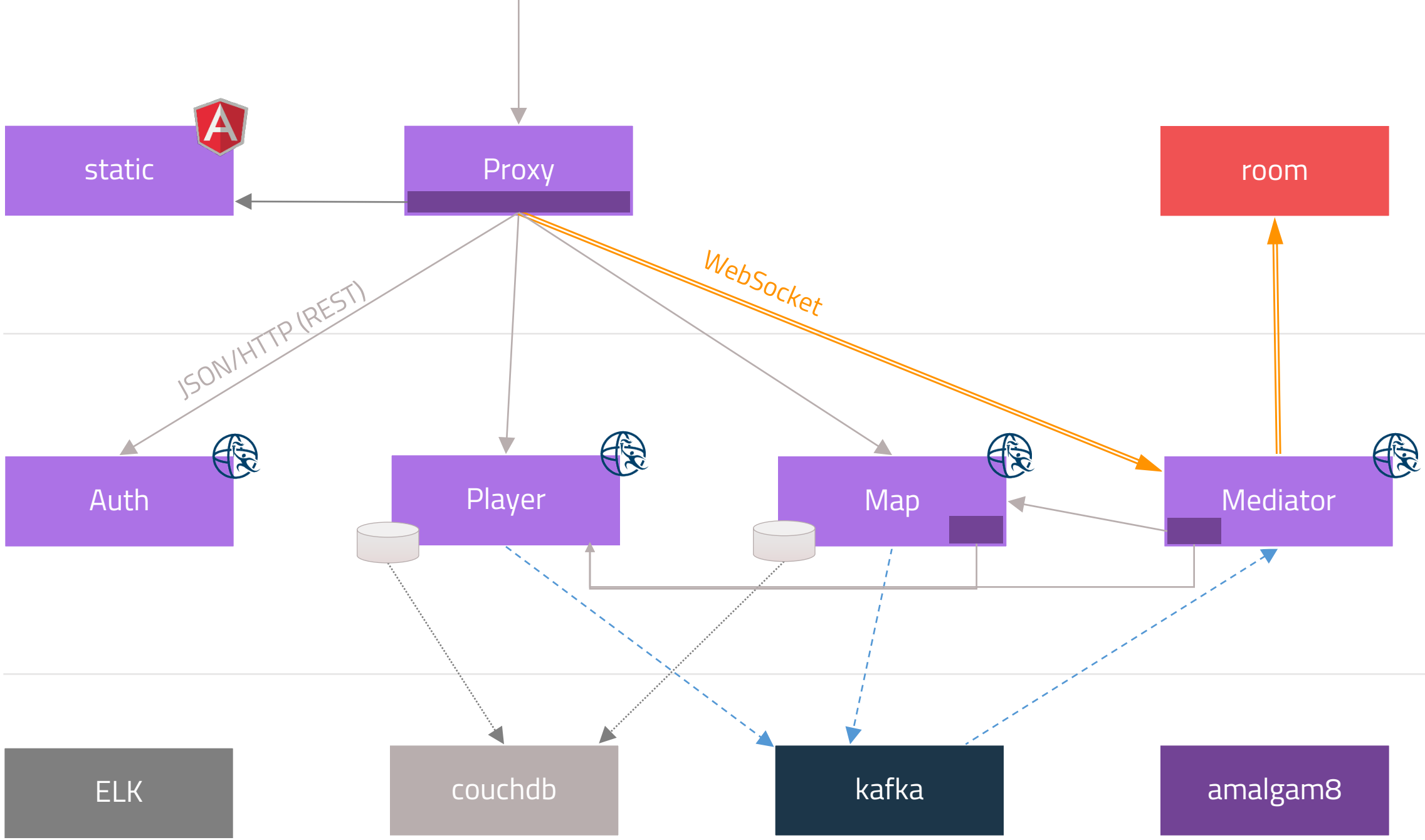
Identifying users

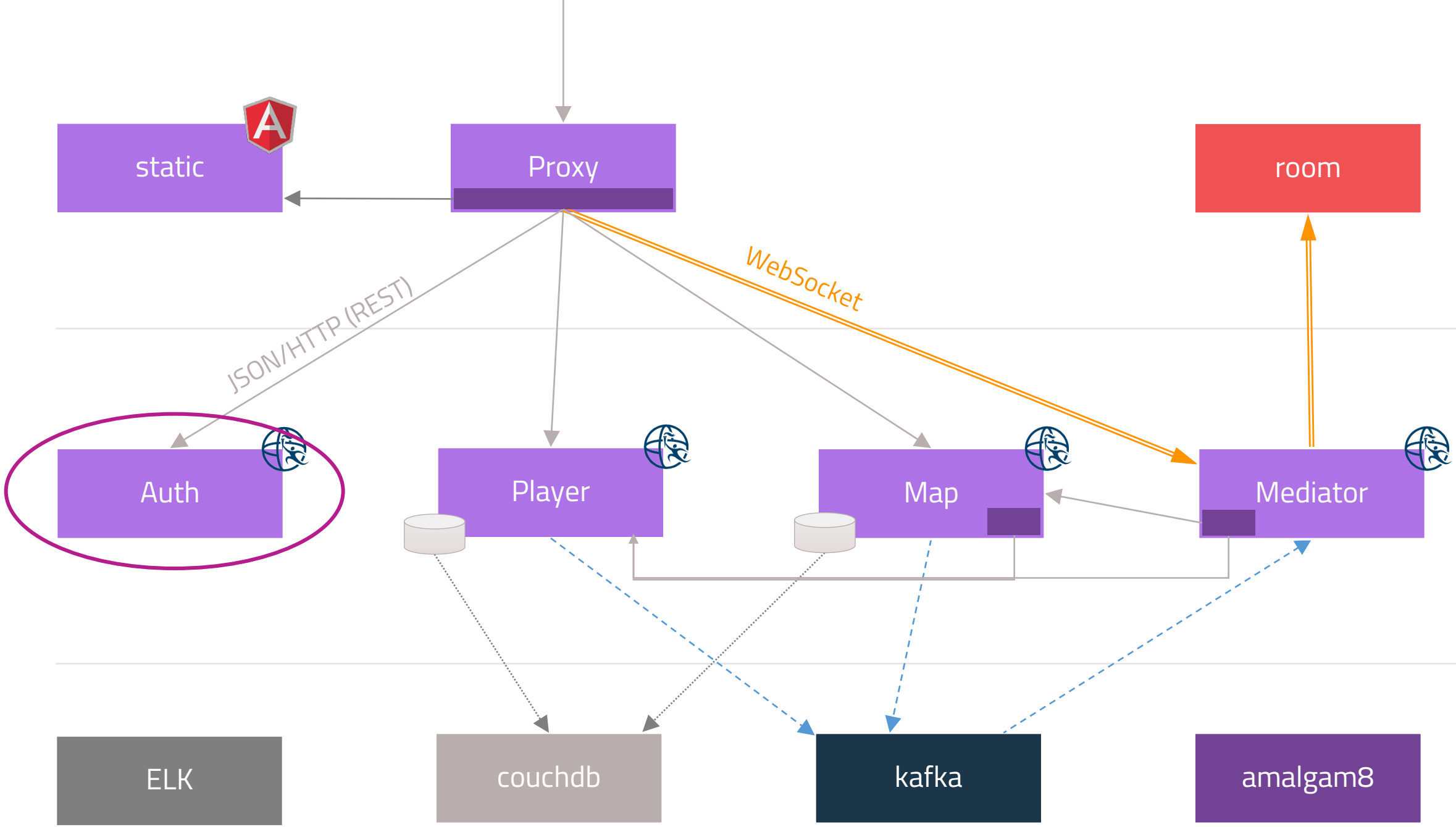
Manage username and password

Social sign-on




User
Client
Game On! Auth service
Authorization Server
REST request to login
Redirect for Authorization
Authorization Code Request
User Authenticates and grants permission
Redirect with authorization code
Authorized callback
Exchange Authorization Code
Access Token
Create signed JWT based on access token
return JWT
User
Client
Game On! Auth service
Authorization Server


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


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

JWTs in Game On

- OAuth authenticates & identifies user
- User identity embedded in JWT issued back to browser
- JWT is signed with gameontext.org certificate
- System services deal with JWT
  - validate JWT & trust embedded identity
- Tricksy: Signed JWT as a the Random State string during OAuth

```
public String createJwt(Key key) throws Exception {
    // create and sign the JWT, including a hint
    // for the key used to sign the request (kid)
    String newJwt = Jwts.builder()
    .setHeaderParam("kid", "meaningfulName")
    .setSubject("user-12345")
    .setAudience("user")
    .setIssuedAt(Date.from(Instant.now()))
    .setExpiration(Date.from(Instant.now().plus(15, ChronoUnit.MINUTES)))
    .signWith(SignatureAlgorithm.RS256, key)
    .compact();
    return newJwt;
}

public void validateJwt(String jwtParameter, Key key) throws Exception {
    // Validate the Signed JWT!
    // Exceptions thrown if not valid
    Jws<Claims> jwt = Jwts.parser()
    .setSigningKey(key)
    .parseClaimsJws(jwtParameter);
    // inspect the claims, like make a new JWT
    // (need a signing key for this)
    Claims jwtClaims = jwt.getBody();
    System.out.println(jwtClaims.getAudience(), jwtClaims.getIssuer());
}
```

Claims

Signed with private key

Validate against public key

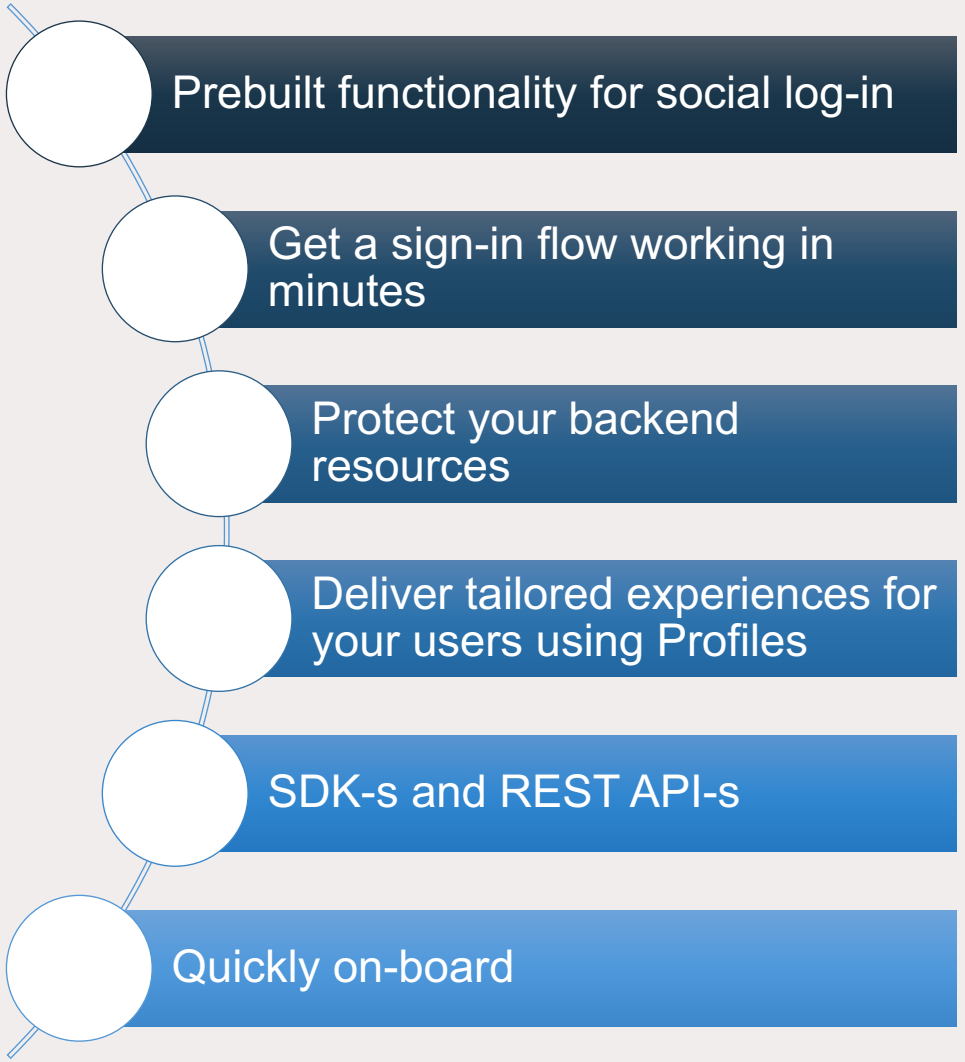
Signed Requests in Game On

- Service-service operations (internal)
  - Signed with shared secret
- Room handshake for WebSocket
  - Signed with room shared secret
- Room registration
  - Signed with room shared secret
- Uses a shared library - https://github.com/gameontext/signed

IBM Bluemix App ID

Helps developers to easily add authentication to their web and mobile apps with few lines of code, and secure their cloud-native applications & services on Bluemix.

App ID also helps manage user specific data that developers can use to build personalized app experiences.



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.

InterConnect
2017