hybris-as-a-service
A microservices architecture in action

Andrea Stubbe
Klaus Herrmann

Product and Technology @ hybris

Disclaimer

This presentation outlines our general product direction and should not be relied on in making a purchase decision. This presentation is not subject to your license agreement or any other agreement with SAP. SAP has no obligation to pursue any course of business outlined in this presentation or to develop or release any functionality mentioned in this presentation. This presentation and SAP's strategy and possible future developments are subject to change and may be changed by SAP at any time for any reason without notice. This document is provided without a warranty of any kind, either express or implied, including but not limited to, the implied warranties of merchantability, fitness for a particular purpose, or non-infringement. SAP assumes no responsibility for errors or omissions in this document, except if such damages were caused by SAP intentionally or grossly negligent.

Agenda

Deconstructing the Vision

The [y] Factors

Architecture

How to Write a Microservice

Sharing is Caring

Deconstructing the Vision

Why we wanted to build hybris as a service, and how we accomplish it

Why Microservices?

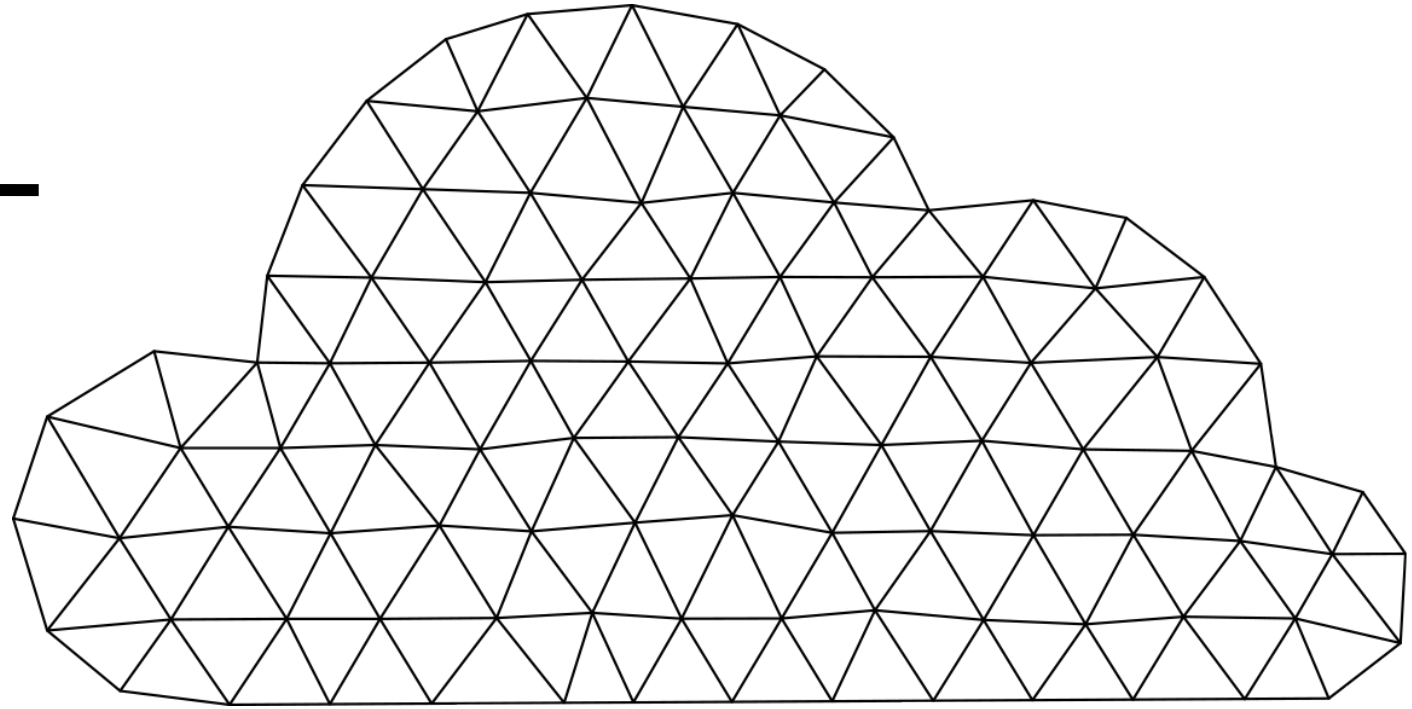
At hybris, we make enterprise commerce software for 15 years, and we learned a few things:

- **Cloud first**. Scaling of our solution has become a priority
- **Retain development speed**. Adding new features becomes increasingly difficult, same for testing and maintenance
- **Autonomy**. Many dependencies in the code lead to many dependencies between teams
- **Community**. Sharing extensions within our development community is hard to achieve

hybris as a Service // The Vision

THE VISION

A cloud platform that allows everyone to easily develop, extend and sell services and applications.



hybris as a Service // Deconstructing the Vision

THE VISION

A cloud platform that allows everyone to easily develop, extend and sell services and applications.

Designed to scale – Core services for storage, messaging, search and more are built with technologies which are known to scale

Open for innovation – Develop with the technologies you like and run your service everywhere you want

Multi-tenant – Infrastructure and core services are shared between all tenants

hybris as a Service // Deconstructing the Vision

THE VISION

A cloud platform that allows everyone to easily develop, extend and sell services and applications.

No secrets – SDK, core APIs and guidelines are visible to everyone

No sales contact – Just sign up and start

Community hub – Submit your service to our community and wait for the money - we take care of the rest

hybris as a Service // Deconstructing the Vision

THE VISION

A cloud platform that allows everyone to easily develop extend and sell services and applications.

Open – Use your favorite languages and technologies

Low learning curve – Tools and an active community help getting you started in minutes

Supportive – Core APIs and SDKs are there to help you, not to restrict you

hybris as a Service // Deconstructing the Vision

THE VISION

A cloud platform that allows everyone to easily develop, extend and sell services and applications

Microservices – Build scalable, resilient, and performant services

Applications – Use templates to build the next generation responsive applications

Backoffice – Build backoffice modules in any frontend technology and integrate them into a central view

The [y] Factors

How we changed the way we make software – the development process and the nature of the software

[y] Factors // Technology

Open Technology Landscape

Freedom to pick the right tool for the job

Scalability of Technology

Linear horizontal scalability: lower costs, less limits on maximal scalability

Design for Failure

If it can be down, it will be down. Design for failure and recovery.

Accessibility of Technology

API-consumer facing technologies must be easy to use and wide-spread

Release Early, Release Often

Establish a deployment pipeline that allows to deliver without fear of breaking things

[y] Factors // Design

API First

Focus on developing rich APIs and develop the functionality later.

Design the API for your customers

Don’t Surprise your Consumers

Use pre-defined patterns and best practices to ensure a consistent API and UI

Isolation of Design

The perfect service has zero dependencies – this ensures that the services, as well as the teams implementing them, can work independently

Simplicity of Design

No need to introduce complex layering or enterprise architecture patterns, as services are small

[y] Factors // Team

Independent, Self Sufficient Teams

Teams should be able to take a product from the concept and feature definition to production with limited dependencies outside of the team.

Self Service Infrastructure

All aspects of the technology infrastructure, required by a developer for any aspect of the development and operations lifecycle should be self-service

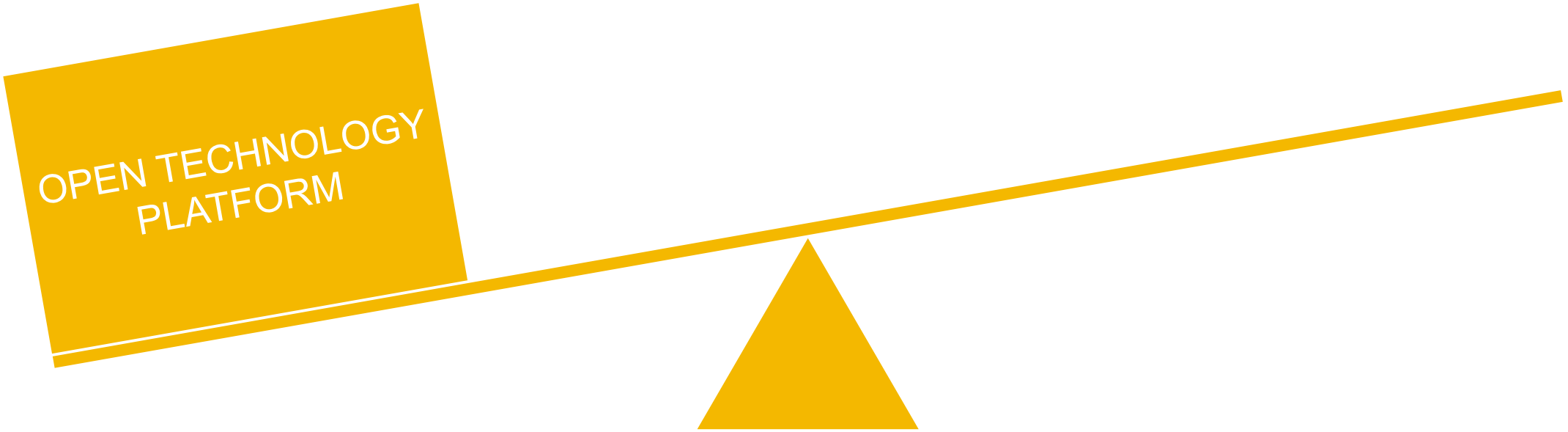
Responsibility

You build it, you release it, you run it, you scale it, you maintain it

[y] Factors // Balance

Open Technology Landscape

Freedom to pick the right tool for the job



[y] Factors // Balance

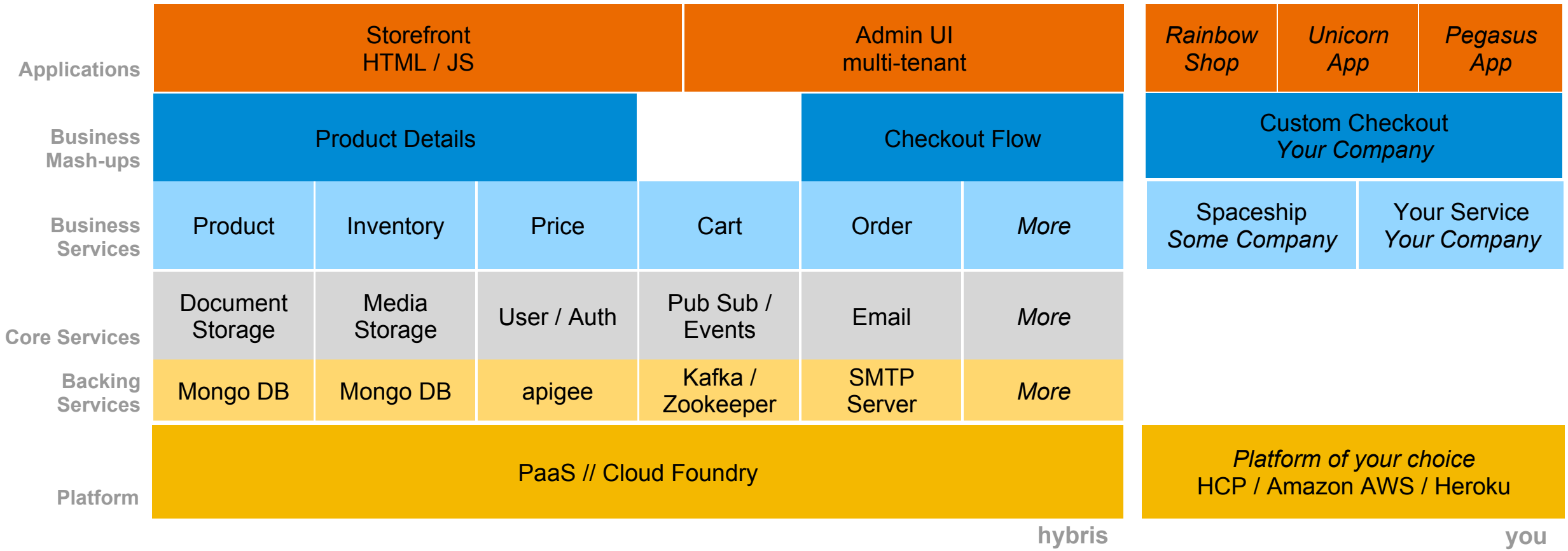
Responsibility

You build it, you release it, you run it, you scale it, you maintain it

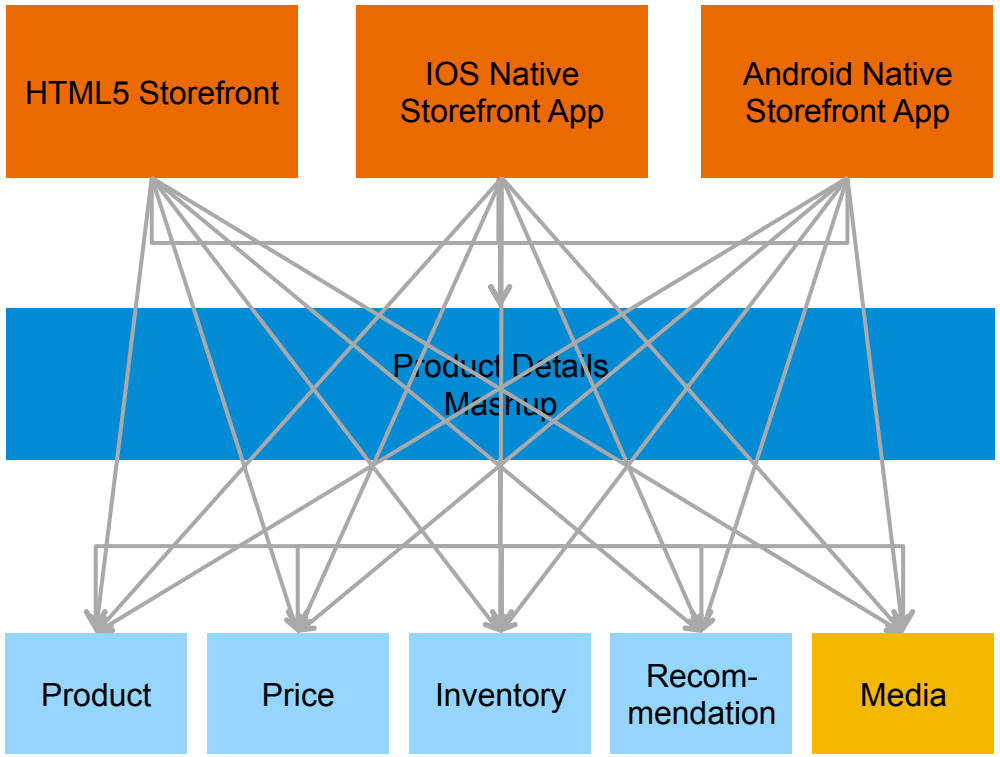
Mickey Mouse Architecture

Layers, and stuff.

Architecture // Layers



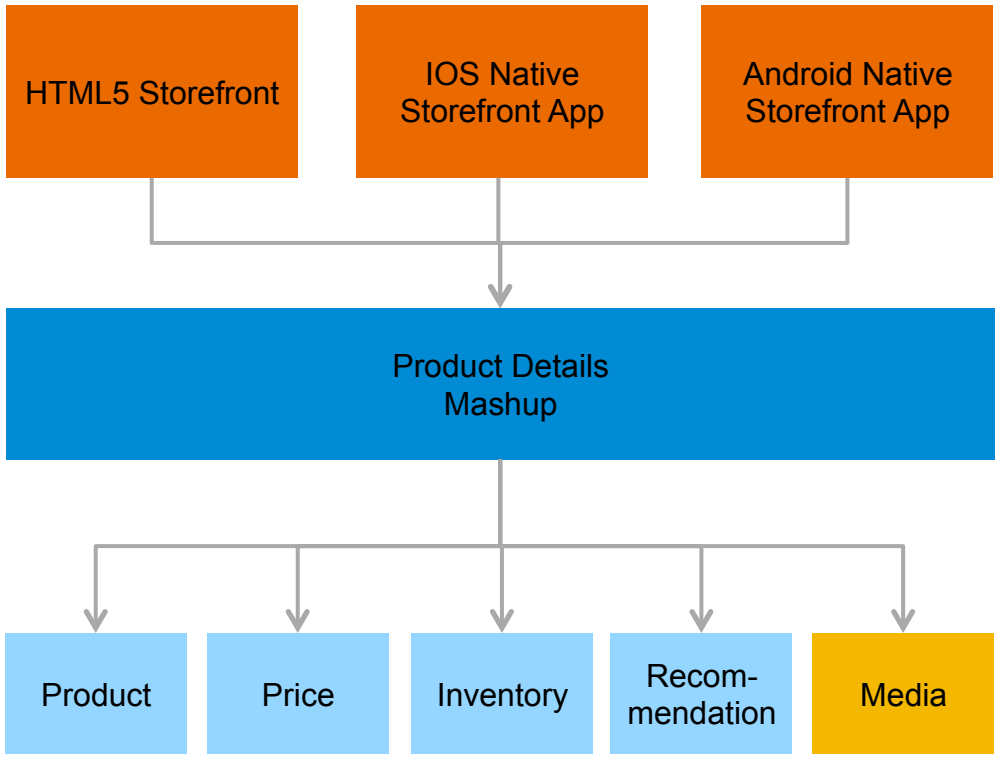
The Role of Mashups // Product Detail Page



If clients always use Microservices directly it

- moves a lot of business logic & error handling logic to the clients
- requires multiple requests for standard flows

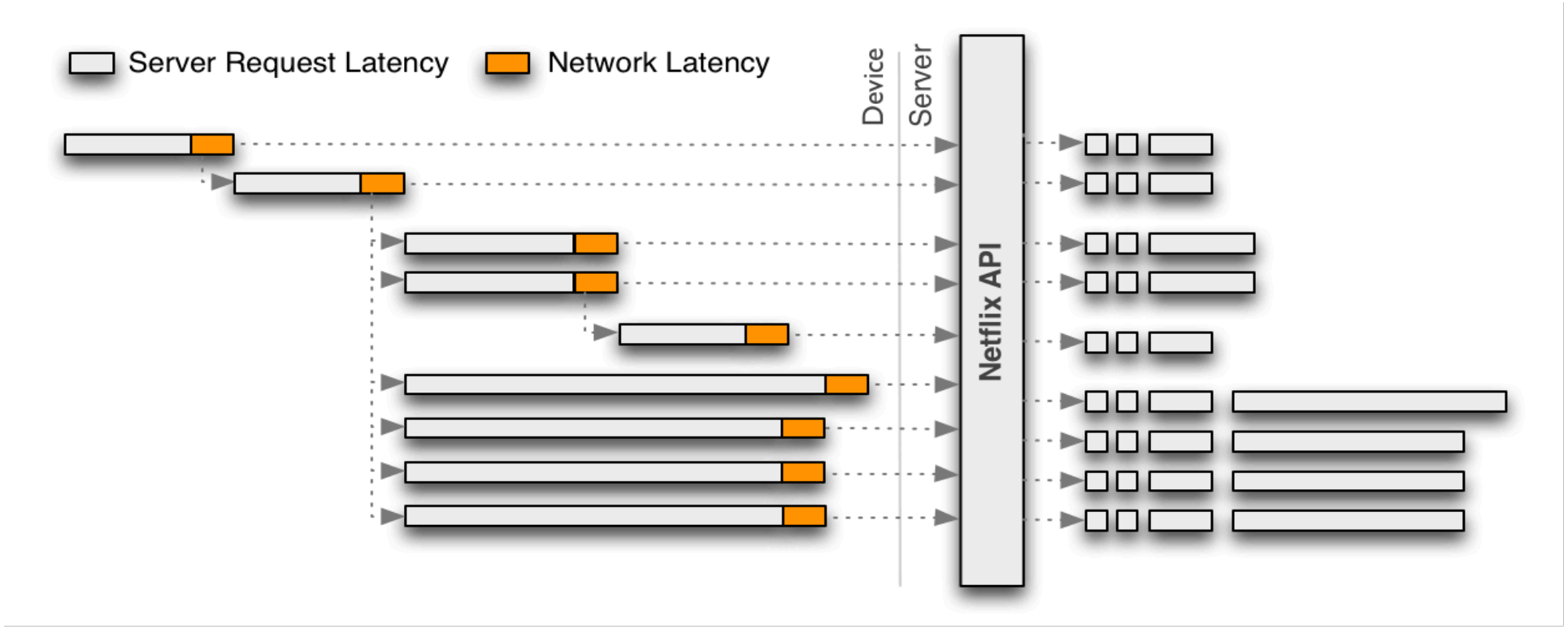
The Role of Mashups // Product Detail Page



Mashups can be used to aggregate service calls or to compose more complex service flows

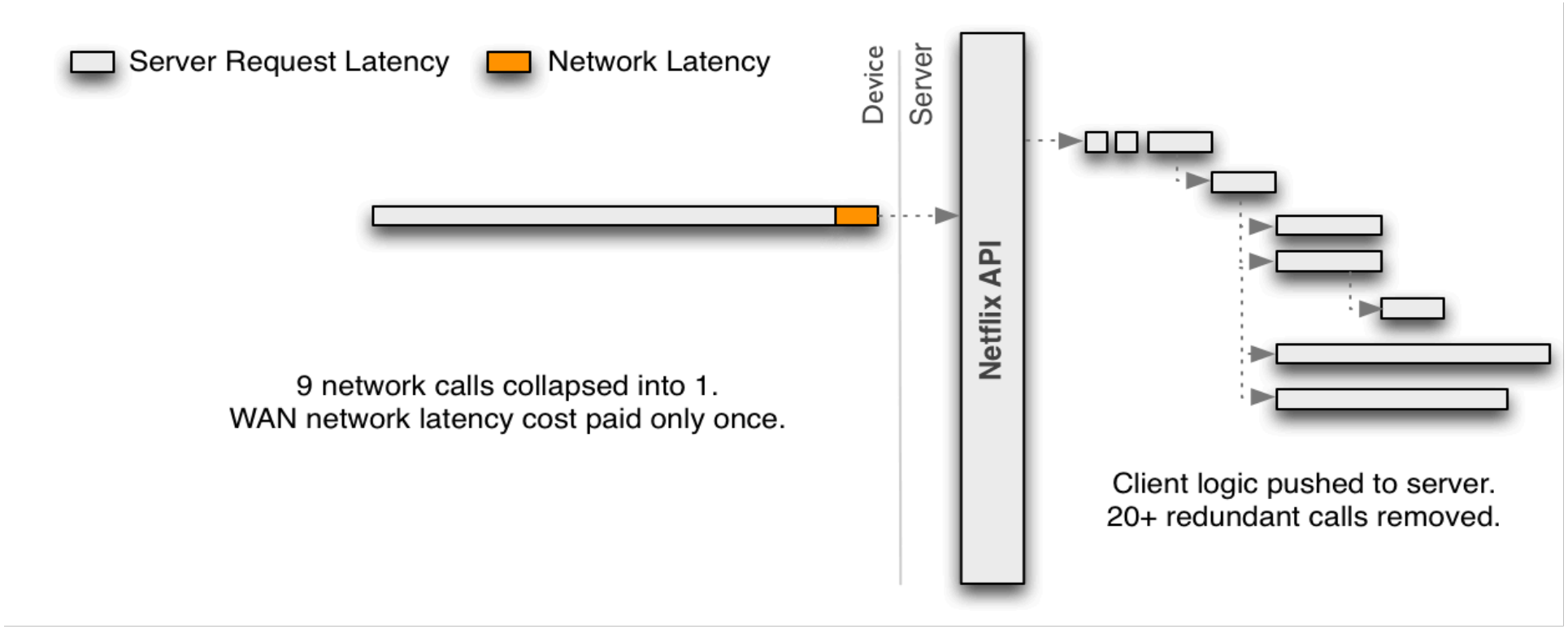
- higher **performance** & **resilience**
- improved **consistency** through fallback strategies
- optimized **APIs** for applications
- promotes **isolation** of individual services (moves most dependencies into mashup layer)

The World Without API Mashups



http://techblog.netflix.com/2013/01/optimizing-netflix-api.html

The World With API Mashups

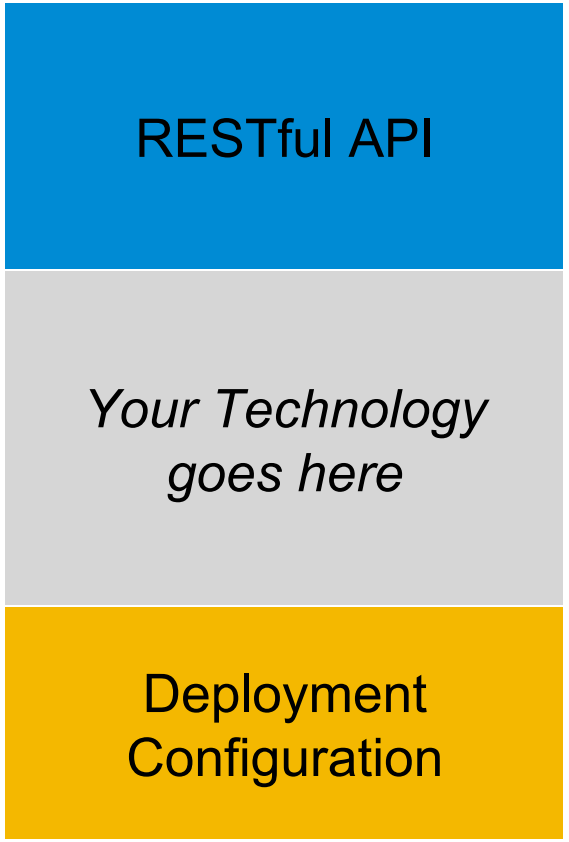


http://techblog.netflix.com/2013/01/optimizing-netflix-api.html

How to Write a Microservice

The anatomy of a service and how hybris as a service reduces your “time to first hello world”

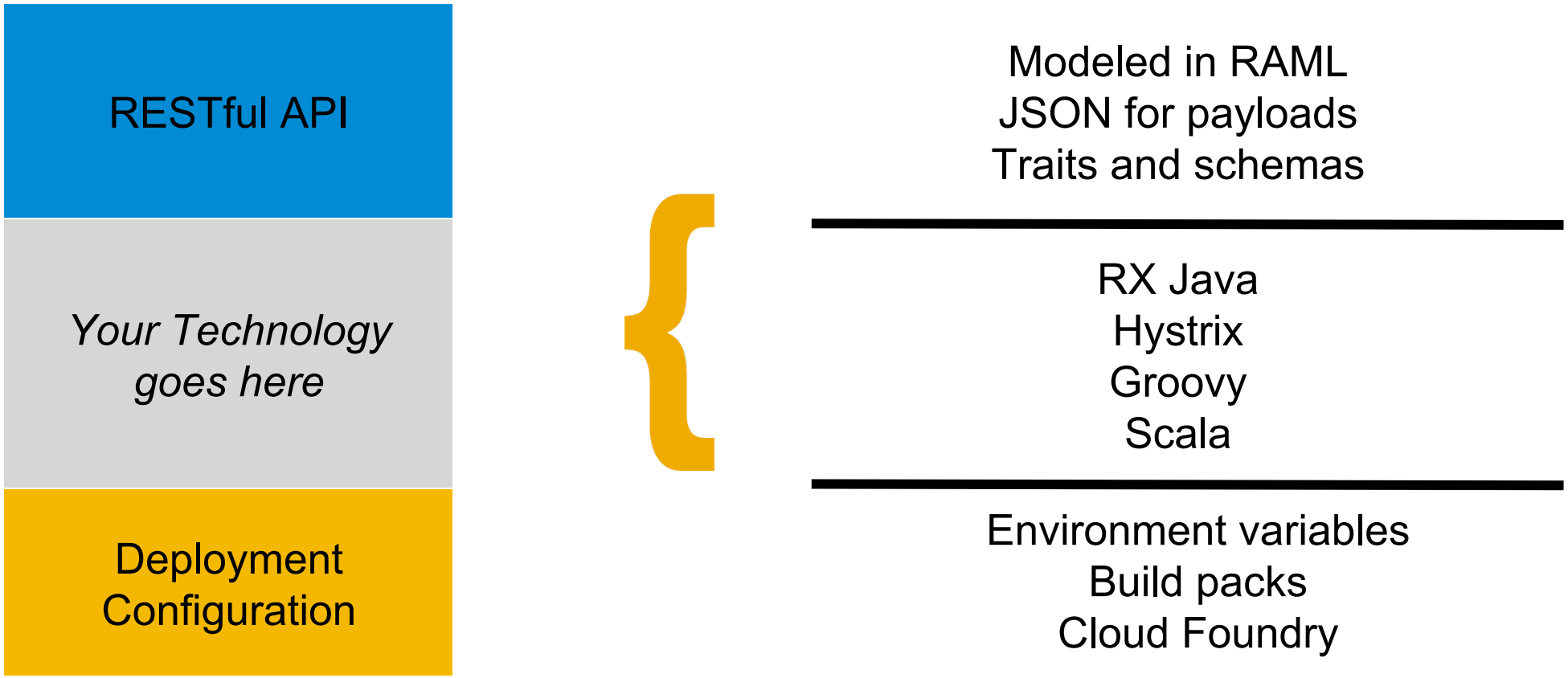
The Anatomy of a Service



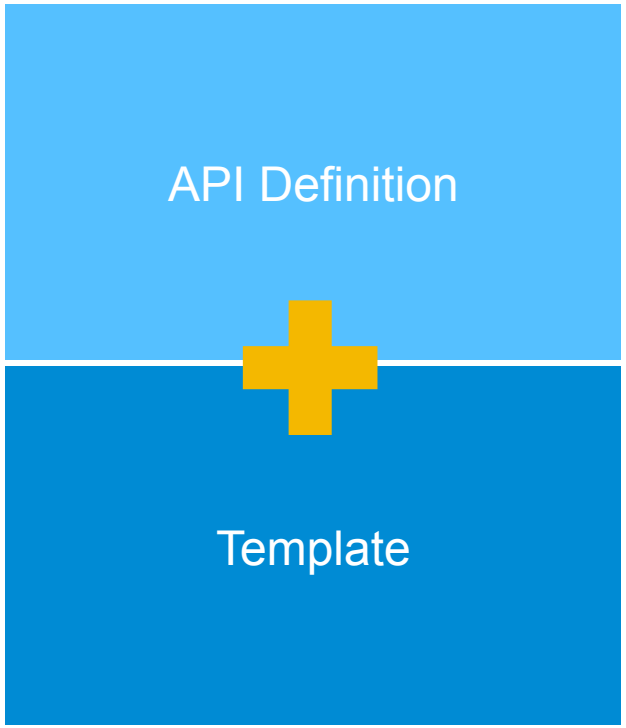
The Aspects of a Service

- Services are consumed over RESTful APIs
- Deployment Configuration matching your containers / infrastructure
- Everything in between is up to you!

The Anatomy of a hybris Service



SDK



- Basic Java Project
- API Implementation
- Documentation

SDK // API Definition

```
/products:
  type: collection
  get:
    is: [paged]
    description: Gets all products
  post:
    description: Creates a new product

  /{productId}:
    type: element
    get:
      description: Gets a product

    put:
      description: Updates a product

    delete:
      description: Deletes a product
```

RAML

- The RESTful API Modeling Language is an open spec, built on standards such as YAML and JSON
- It encourages reuse through pattern-sharing (schemas, traits, types)
- Broad tool support to design and test APIs, and to generate server and client code

SDK // Patterns - Traits

```
traits:
- !include http://api.yaas.io/patterns/v1/trait-paged.yaml

/products:
   get:
     is: [ paged ]
```

RAML

http://api.yaas.io/products?**pageNumber**=2&**pageSize**=10

- Use traits to model recurring properties of methods
- Less repetition, more consistency

SDK // Patterns - Schemas

RAML

```
schemas:
- error: !include http://api.yaas.io/patterns/v1/schema-error.json

 ...400:
      body:
        application/json:
            schema: error
```

JSON

```
{
    "status": 400,
    "moreInfo": "https://developer.yaas.io/errors/missing.header”,
    "message": "Missing header"
}
```

Error response body

- Define common schemas and share them across all APIs
- Can be used for input or output

SDK // Documentation as Part of the Code

.md

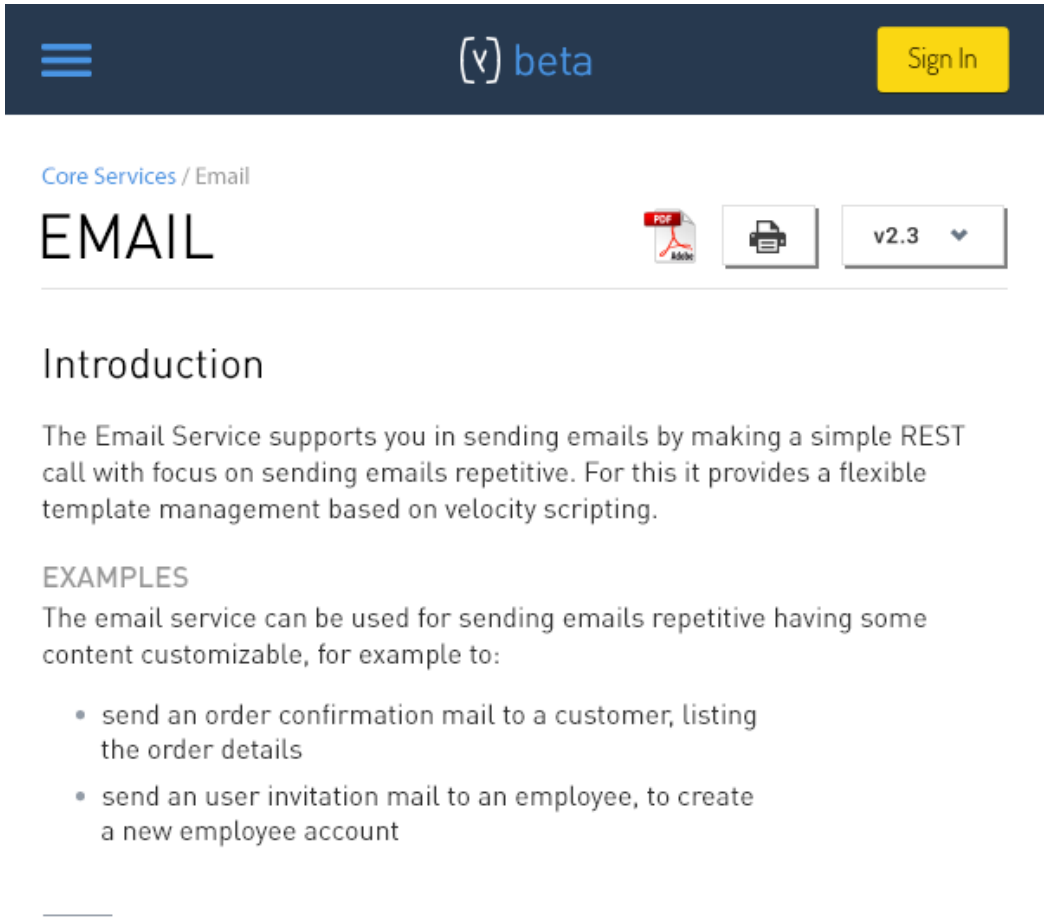
```
---
title: 'About'
layout: document
service: 'Email'
type: General
order: 1
---

# Introduction

The Email Service supports you in sending emails
by making a simple REST call with focus on sending
emails repetitive. For this it provides a flexible
template management based on velocity scripting.

### Examples

The email service can be used for sending emails
repetitive having some content customizable, for
example to:
* send an order confirmation mail to a customer,
listing the order details
* send an user invitation mail to an employee, to
create a new employee account
```



SDK // Template

Three simple steps to get you started

```
mvn archetype:generate [group, artifact, version]

mvn clean install

mvn jetty:run
```

Maven

Try it using the API Console

```
http://localhost:8080
```

- Use Maven commands to generate the basic project and API implementation
- Explore and test your API using Mulesoft's API Console

Demo

Write a service. Really fast.

Ready to Use APIs

Core Services

Everything which is domain agnostic

- Storing content and data
- Messaging and events
- Users and roles
- Search
- ...

Commerce Services

Everything about commerce

- Customers
- Orders
- Shipping and tax
- Products
- ...

Now add yours.

Code Samples // How to Use the APIs

Get an access token

```
curl –i -X POST https://api.yaas.io/auth/anonymous/login?project=YOUR_PROJECT_ID;
```

HTTP

Make calls

```
curl -H "Authorization: Bearer YOUR_AUTHTOKEN" https://api.yaas.io/product/v1/products
```

HTTP

- APIs are secured with OAuth 2
- One auth-token works for all APIs published in the community hub

Multi-Tenancy && Extensibility

Data

Extensible Schemas and flexible data stores for core entities (customers, orders, …)

Data isolation per tenant

Functionality

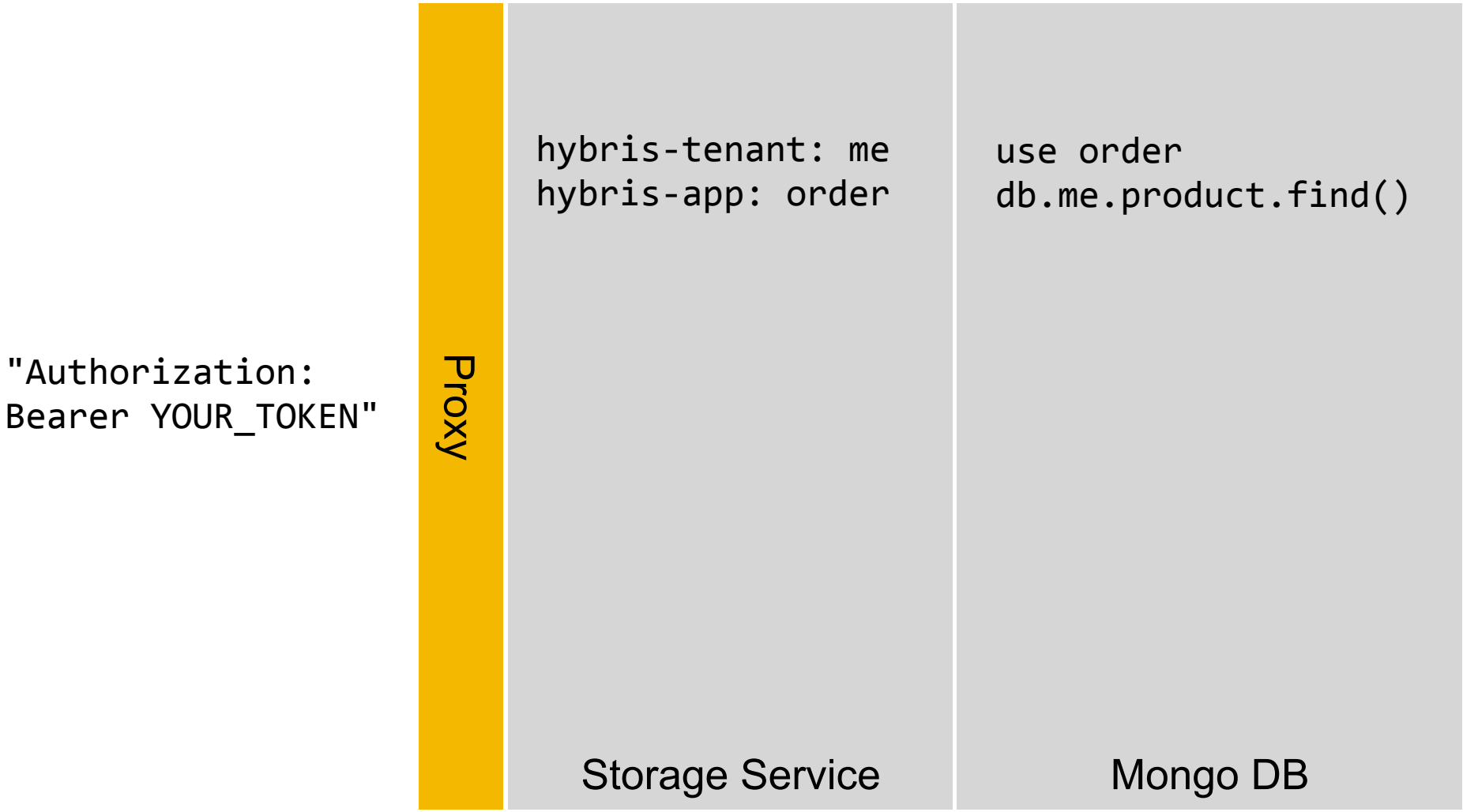
Key aspects and behaviors are configurable per tenant

Mash up multiple services to create APIs to model complex use cases

Integration

Event / message driven communication allows externals systems to observe and react

Multi-Tenancy // Data Isolation



Sharing is Caring

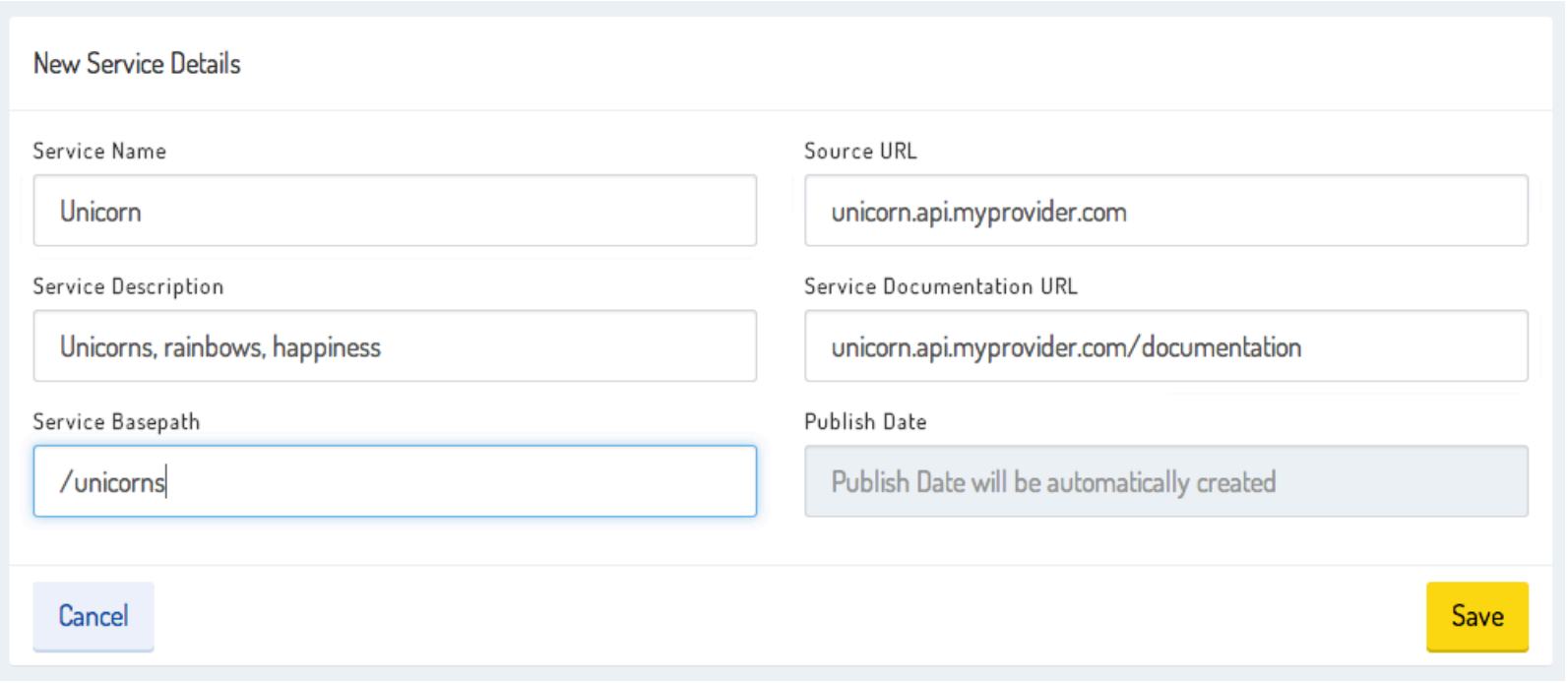
How to offer your services to others

Your Service is Finished – Now What?

- Offer it to customers (marketing and sales channels)
- Control access
- Make it easy to integrate with other services
- Measure usage
- Invoice users and get money

We make this easy for you

API Management // Publish your Service



- Creates proxy for authentication, metering, billing
- Use the community hub to publish and sell your APIs

Favorite Bookmarks

- http://12factor.net/
- http://www.reactivemanifesto.org/
- http://martinfowler.com/articles/microservices.html
- http://raml.org/ - API modeling
- https://www.youtube.com/watch?v=R8SIxZVaai4 - from apigee, about API design

Thanks for listening.
We need to talk.

© 2014 SAP SE or an SAP affiliate company. All rights reserved.