

Contract Testing

Toni Masotti
masotti@bikeleasing.de
Bikeleasing-Service

November 26, 2023

Outline

- 1 Problem statement
- 2 Option C : Contract Testing
- 3 Demo Time
- 4 Concluding thoughts

1 Problem statement

2 Option C : Contract Testing

3 Demo Time

4 Concluding thoughts



Let's start

It all started with Bob...



...and it continues with Bob

It all started with Bob...



Bob

They do work on
my local machine
though 😢

BOB ! the changes you
made do not work on the
live system 😡



Angry chef

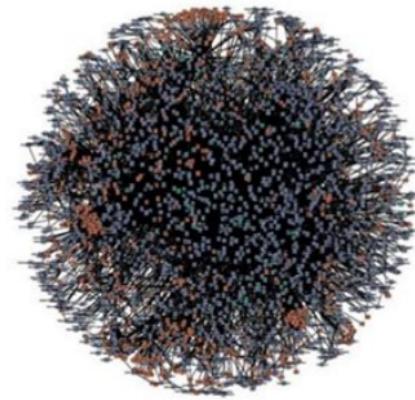
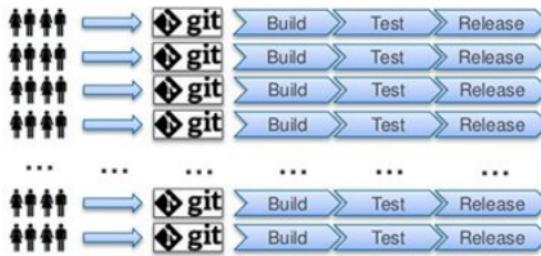
Bob has learnt the lesson...





Extreme examples...

- 50 Million Deployments a Year
- Software enhancements delivered every second



Gigantic Web of Micro-services at Amazon



Everybody loves microservices

- Big promise: Independent deployability

Everybody loves microservices

- Big promise: Independent deployability
- Fast feedback loops

Everybody loves microservices

- Big promise: Independent deployability
- Fast feedback loops
- Strong team focus, ideally crossfunctional

Everybody loves microservices

- Big promise: Independent deployability
- Fast feedback loops
- Strong team focus, ideally crossfunctional
- Loosely coupled architecture

Everybody loves microservices

- Big promise: Independent deployability
- Fast feedback loops
- Strong team focus, ideally crossfunctional
- Loosely coupled architecture
- Fast onboarding

Everybody loves microservices

- Big promise: Independent deployability
- Fast feedback loops
- Strong team focus, ideally crossfunctional
- Loosely coupled architecture
- Fast onboarding
- Cost-efficiency on the long run

Everybody loves microservices

- Big promise: Independent deployability
- Fast feedback loops
- Strong team focus, ideally crossfunctional
- Loosely coupled architecture
- Fast onboarding
- Cost-efficiency on the long run
- Cloud native

Everybody loves microservices

- Big promise: Independent deployability
- Fast feedback loops
- Strong team focus, ideally crossfunctional
- Loosely coupled architecture
- Fast onboarding
- Cost-efficiency on the long run
- Cloud native
- Tech-agnostic approach

Everybody loves microservices

- Big promise: Independent deployability
- Fast feedback loops
- Strong team focus, ideally crossfunctional
- Loosely coupled architecture
- Fast onboarding
- Cost-efficiency on the long run
- Cloud native
- Tech-agnostic approach
- Scalability out of the box

Everybody loves microservices

- Big promise: Independent deployability
- Fast feedback loops
- Strong team focus, ideally crossfunctional
- Loosely coupled architecture
- Fast onboarding
- Cost-efficiency on the long run
- Cloud native
- Tech-agnostic approach
- Scalability out of the box
- Improved Security (no SPOF, no SPOC)

Everybody loves microservices

- Big promise: Independent deployability
- Fast feedback loops
- Strong team focus, ideally crossfunctional
- Loosely coupled architecture
- Fast onboarding
- Cost-efficiency on the long run
- Cloud native
- Tech-agnostic approach
- Scalability out of the box
- Improved Security (no SPOF, no SPOC)
- Optimized Time to Market

Everybody loves microservices

“ Chief among the benefits of service-enabling an enterprise’s application landscape are **increased organizational agility and **reduced overall cost of implementing change**.**

*A SOA increases organizational agility by placing high-value business functions in **discrete, reusable services**, and then connecting and orchestrating these services to satisfy core business processes.*

*It reduces the cost of change by **reducing the dependencies** between services, allowing them to be rapidly recomposed and tuned in response to change or unplanned events.*

”

martinFowler.com

Refactoring Agile Architecture About Thoughtworks

Consumer-Driven Contracts: A Service Evolution Pattern

This article discusses some of the challenges in evolving a community of service providers and consumers. It describes some of the coupling issues that arise when service providers change parts of their contract, particularly document schemas, and identifies two well-understood strategies – adding schema extension points and performing “just enough” validation of received messages – for mitigating such issues. Both strategies help protect consumers from changes to a provider contract, but neither of them gives the provider any insight into the ways it is being used and the obligations it must maintain as it evolves. Drawing on the assertion-based language of one of these mitigation strategies – the “just enough” validation strategy – the article then describes the “Consumer-Driven Contract” pattern, which imbues providers with insight into their consumer obligations, and focuses service evolution around the delivery of the key business functionality demanded by consumers.

12 June 2006



Ian Robinson

Ian Robinson is a Principal Consultant with Thoughtworks. He specializes in helping clients create sustainable service-oriented development capabilities that align business and IT teams successfully through to operation. He is a frequent speaker at conferences on Microservices, cloud and service-oriented systems with Microsoft technologies, and has published articles on business-oriented development methodologies and distributed systems design – most recently in *The ThoughtWorks Guide to Microservices* co-authored a book on Web-friendly enterprise software.

CONTENTS

- Evening a Service: An Example
- Interface-Bordered With Services
- Schema Versioning
- Extension Points
- Breaking Changes
- Schematron
- Consumer-Driven Contracts
- Provider Contracts
- Contract Evolution
- Consumer-Driven Contracts
- Summary of Contract Characteristics
- Implementation
- Benefits
- Liabilities

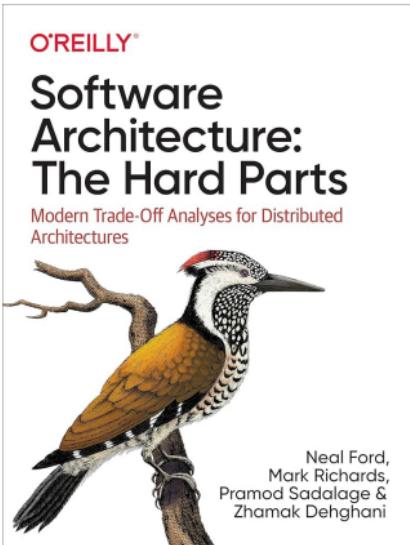
- APPLICATION INTEGRATION
- WEB SERVICES

Is always a tradeoff

“ There is only one hard rule in software development: Everything is a tradeoff.

– Neal Ford

”



1 Problem statement

2 Option C : Contract Testing

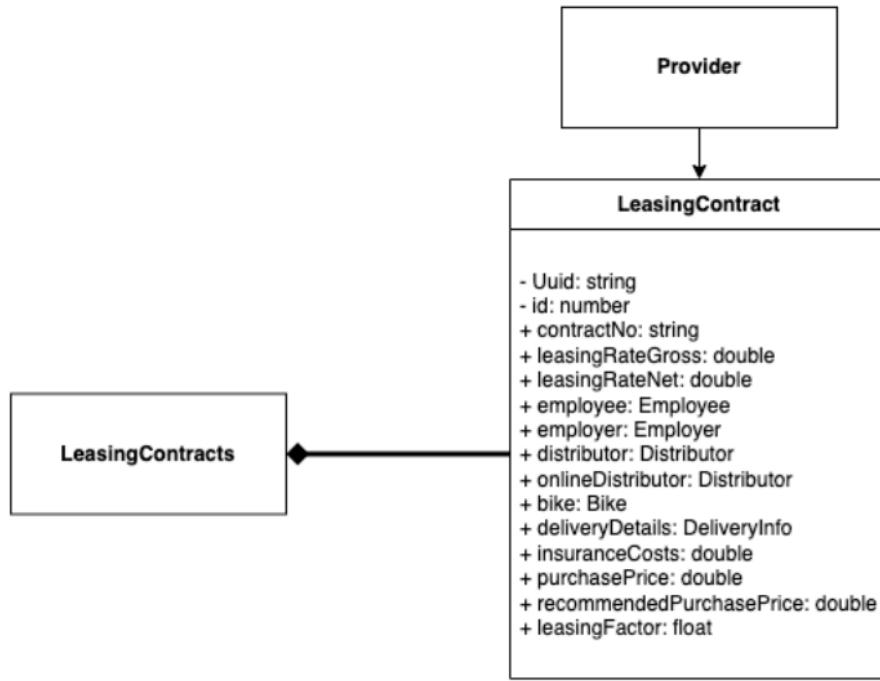
3 Demo Time

4 Concluding thoughts

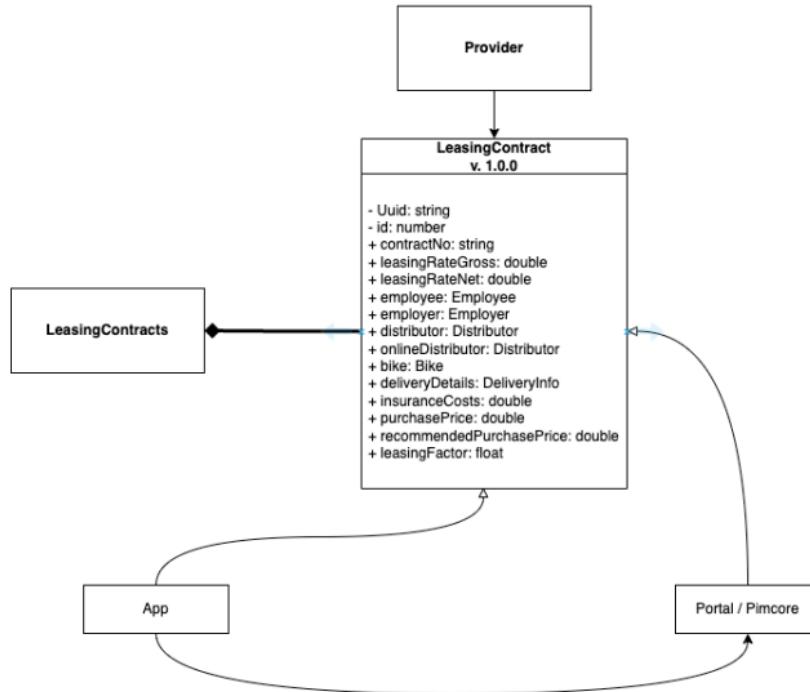
Bob's company is being very successfull...



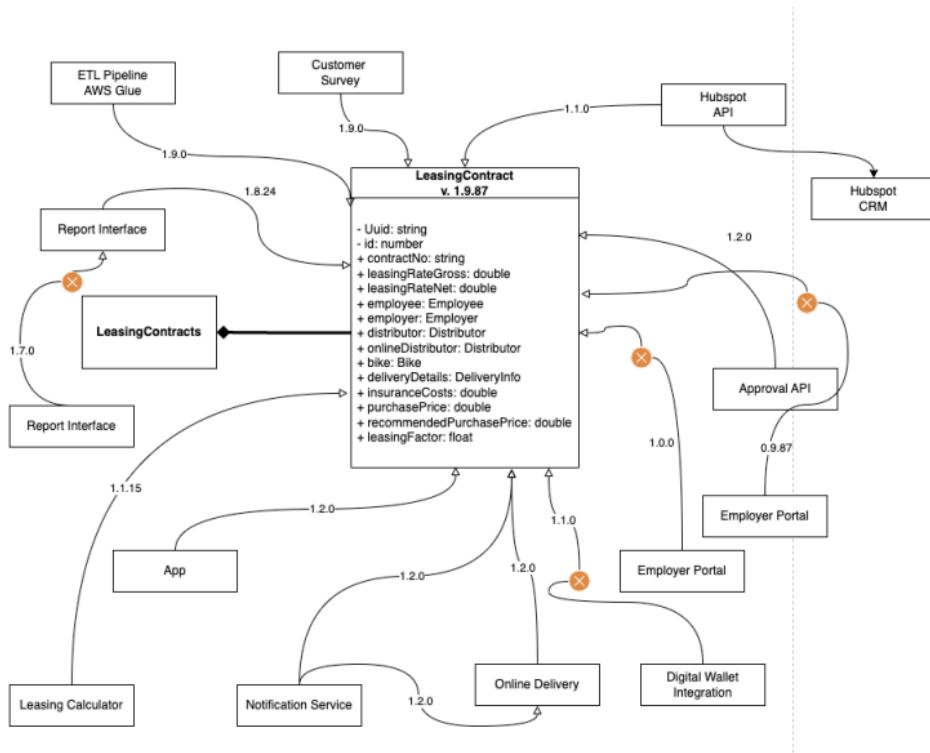
A company growth... so its architecture



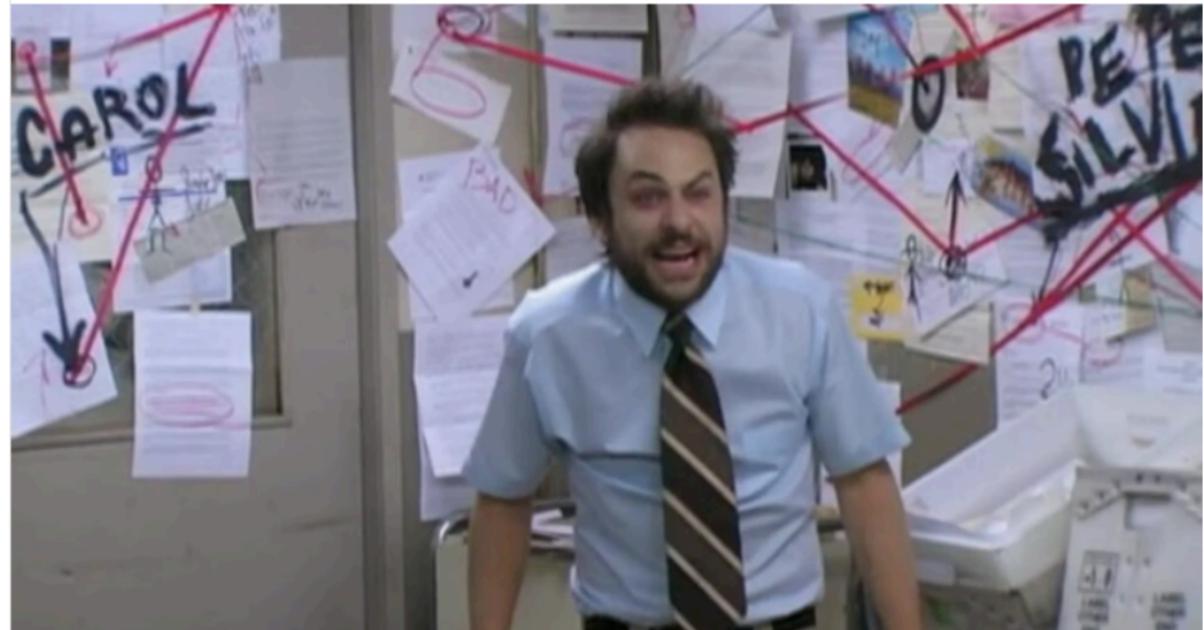
A company growth... so its architecture



A company growth... so its architecture



Bob is happy...



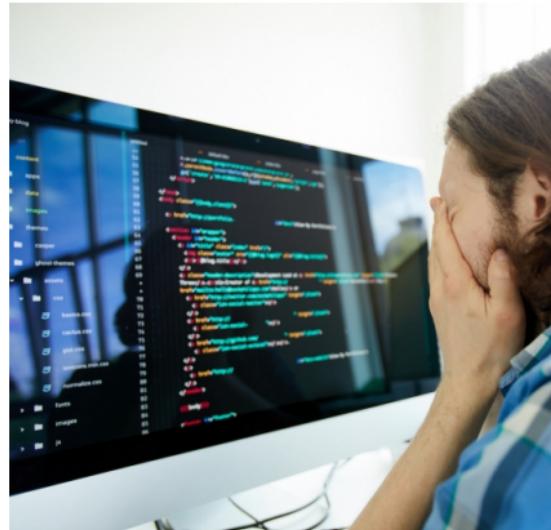
The Problem...

“

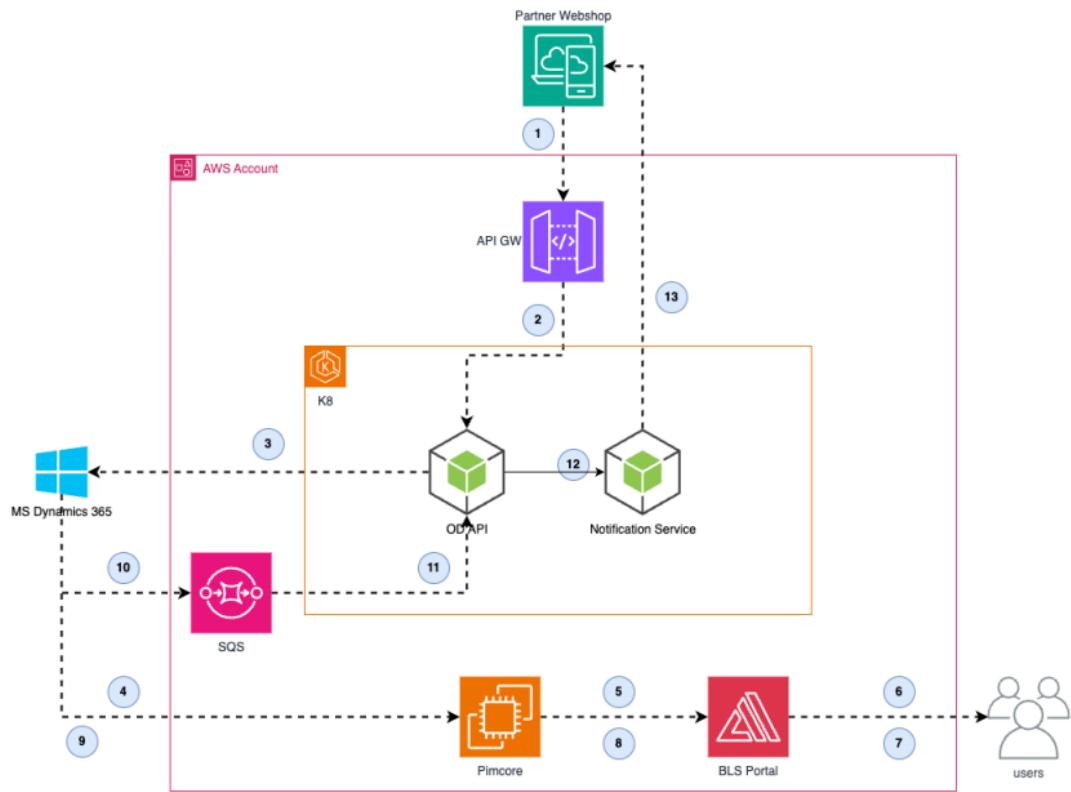
Every time we touch one service, all the others break :(

”

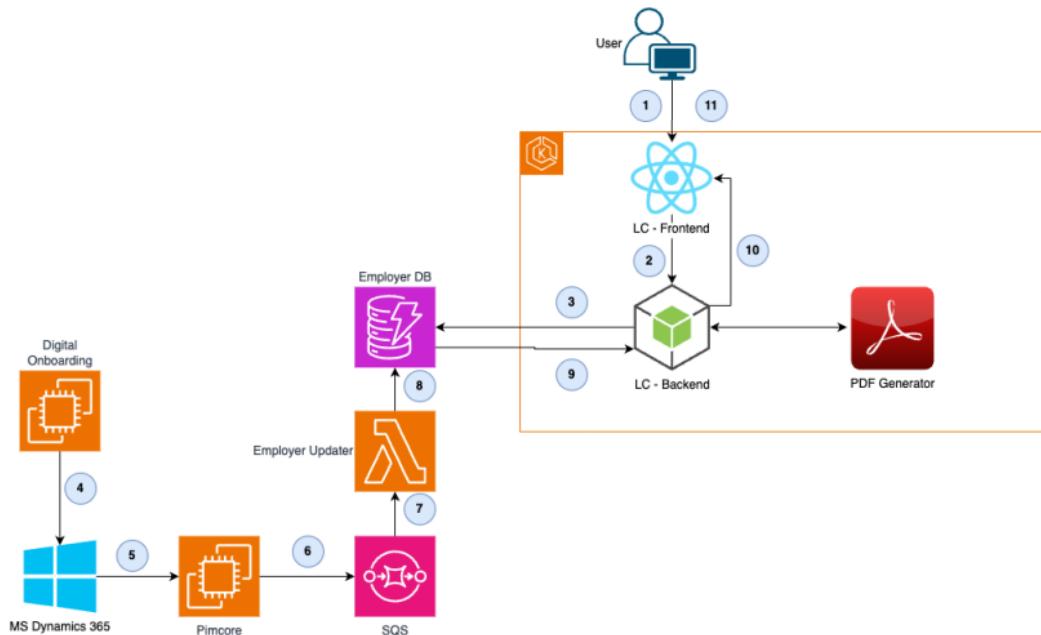
- Dependency hell
- Devs don't have confidence in deploying changes
 - Consequences of changes are hard to track
 - Distributed architectures are hard to test



Architecture example: Online Delivery



Architecture example: Leasing Calculator



Updating a microservice: the sad Atlassian case

```
{  
  "users": "Mike",  
  "address": "Something"  
}
```

```
1 {  
2   "user": "Mike",  
3   "address": "Something"  
4 }  
5
```



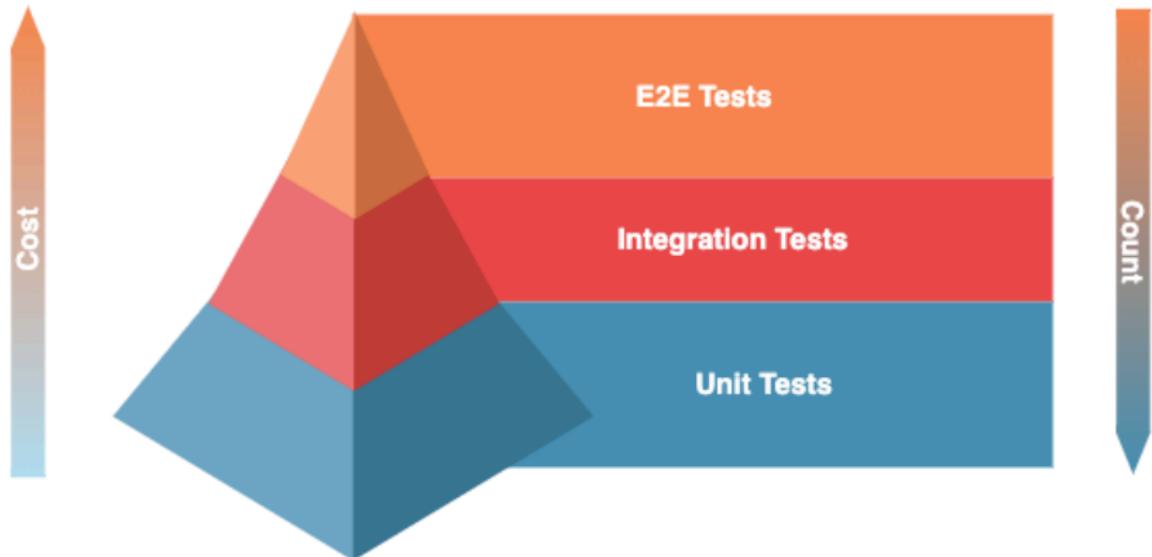
Something's gone wrong



Our team has been notified. If the problem persists,
please contact Atlassian Support.

[Reload page](#)

Test Typology



The problem with Unit Tests alone

- Local changes, even if tested, do not assure that the system as whole is working
- Mocks are not guaranteed to really represent the other part of the system

The bad side of mocks

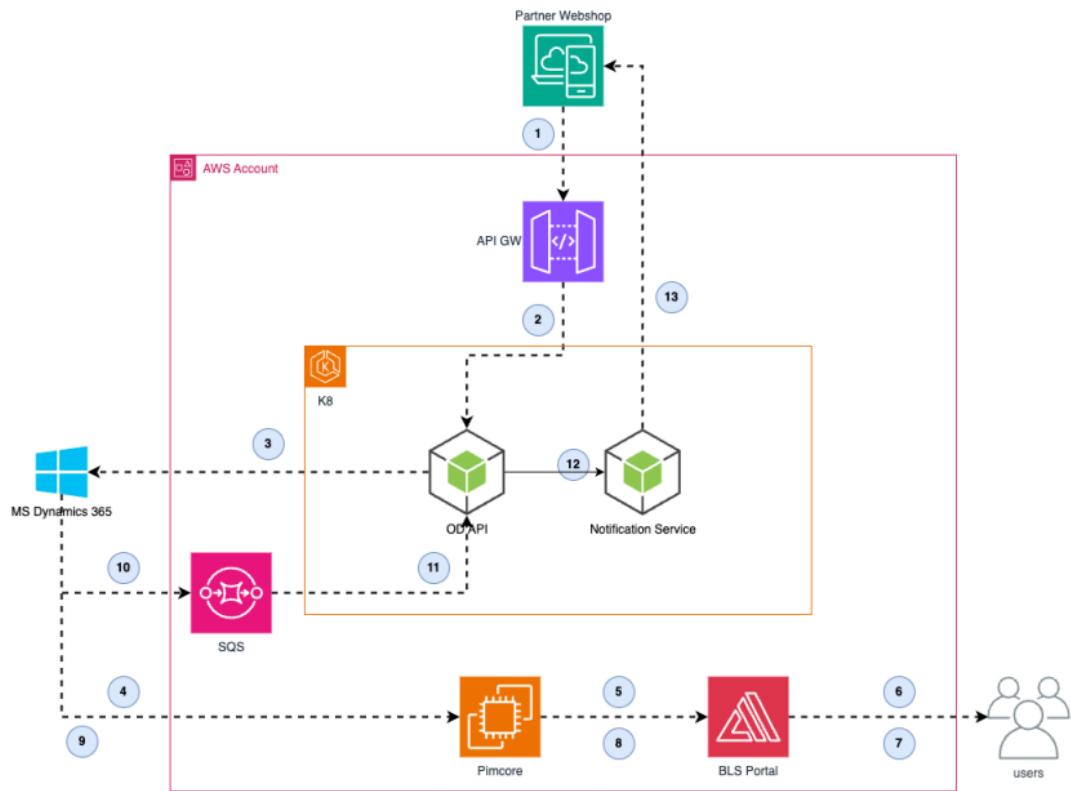
Embedded Animation

Problem with E2E Tests

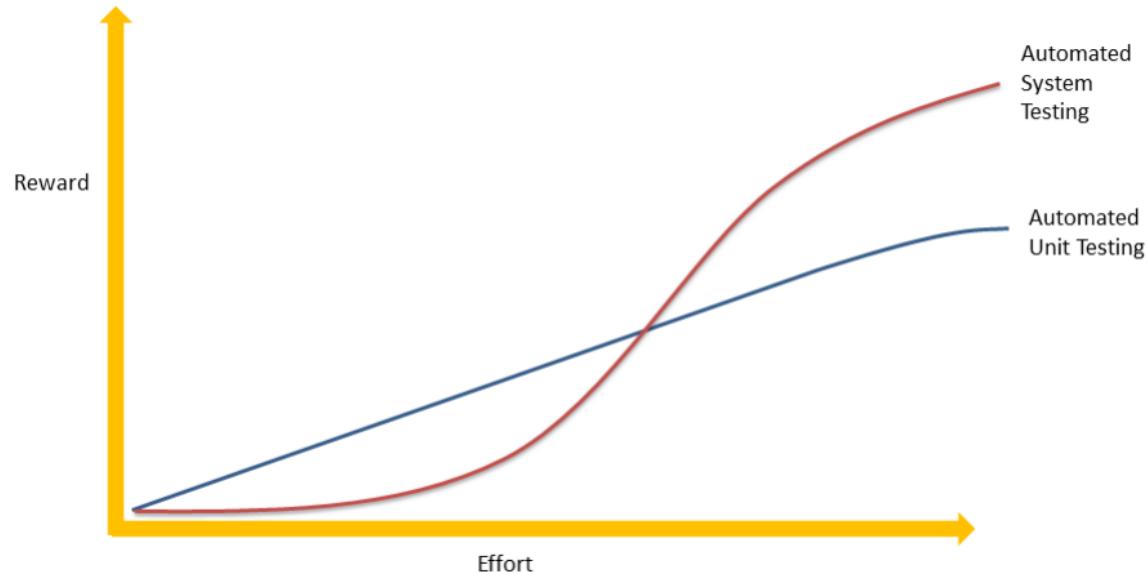
They're very important (*qua* realistic), **but...**

- they tend to be not reproduceable locally
- they're hard or impossible to test some scenarios (e.g. system not available)
- they're expensive (time and money)
 - The longer we need to find out what's wrong, the more expensive it is
- Not few companies report *excessive test setup*
- they're prone to flakiness

Architecture example



Architecture example



Link: Trinitatum Blog (2015-03-02)

Distributed ≠ Decoupled



Is this one or three pipelines?

What do we need?

We need a way to **test and ensure correct data exchange** between services, with the following characteristics:

- **Fast feedback** on changes
- **Confidence** in our changes
- **Reproducibility** of tests
- **Realistic** tests
- **Cheap** tests
- **Easy** to write and maintain

What do we need?

Seen so far:

- **Option A:** Mocks
- **Option B:** E2E / System Tests with real infrastructure

Is there an option C?

1 Problem statement

2 Option C : Contract Testing

3 Demo Time

4 Concluding thoughts

Contracts

Contracts

A formal agreement between parties or individual. Synonyms: pact, agreement, protocol, deal

— Oxford English Dictionary

Contract Testing

Contract testing is a technique for testing an integration point by checking each application in isolation to ensure the messages it sends or receives conform to a shared understanding that is documented in a "contract".

— Pact Docs

A pact between who?

- **Consumer:** the service that consumes the information (typically closer to the user)
- **Provider:** the service that provides the information (typically closer to the data)

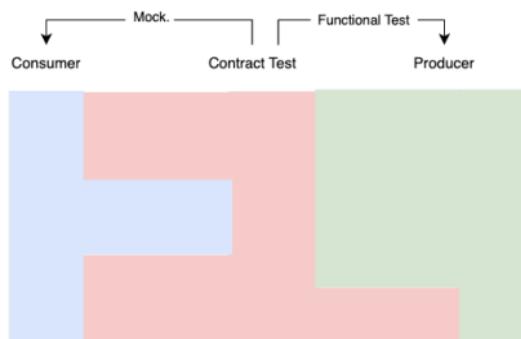
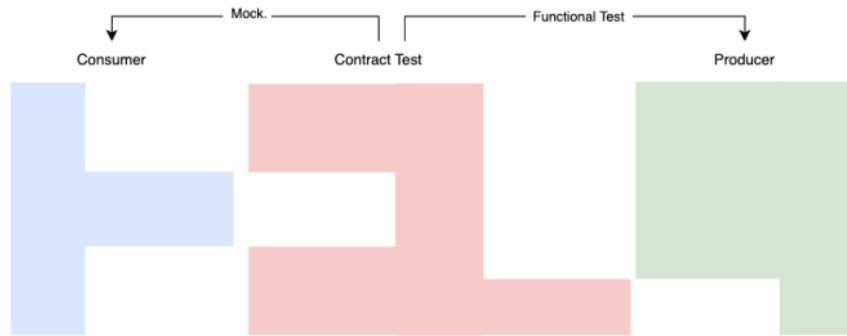
A pact between who?

- **Consumer:** the service that consumes the information (typically closer to the user)
- **Provider:** the service that provides the information (typically closer to the data)

Where do we find this architecture?

- REST, GraphQL, gRPC, SOAP ...APIs
- Microservices communication (over TCP, HTTP, Quic, AMQP, ...)
- SOA
- Other kinds of distributed systems (IoT, ...)
- Client-Server
- Messaging systems (Kafka, RabbitMQ, SQS ...)
- Notifying systems (Webhooks, SNS, ...)

Contract Test Role

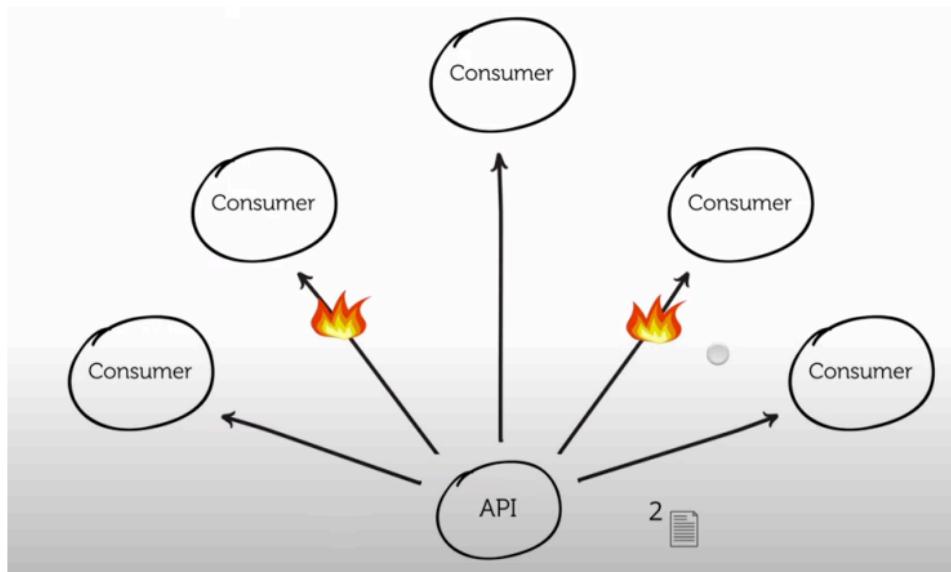


Who initiates the contract?

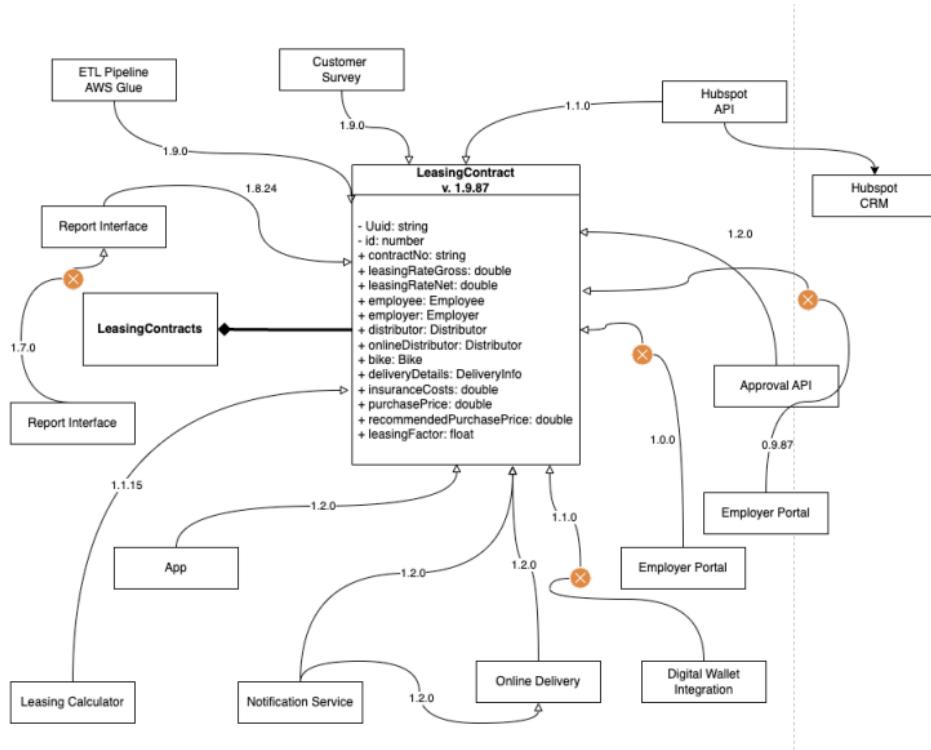
Both parties can initiate the contract, but...

Here I will present you the **Consumer Driven Contracts** approach.

Provider driven contracts



Provider driven contracts

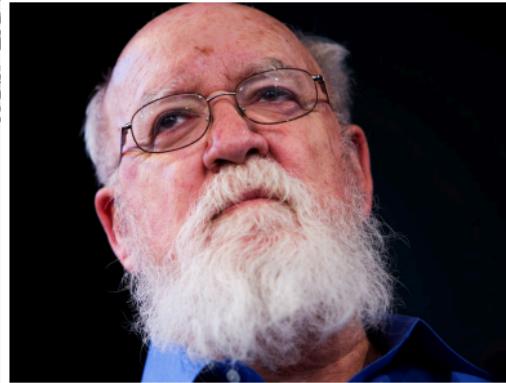
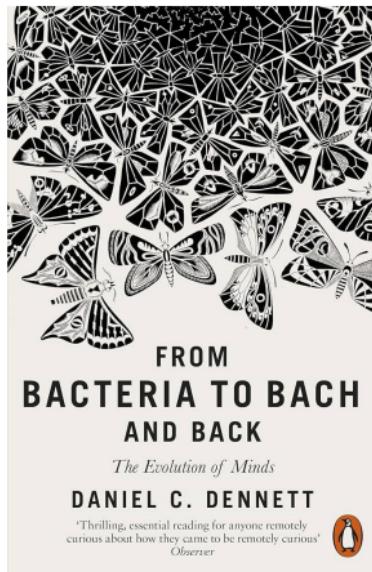


Consumer Driven Contracts

“

A strange inversion of reasoning... – D. Dennet

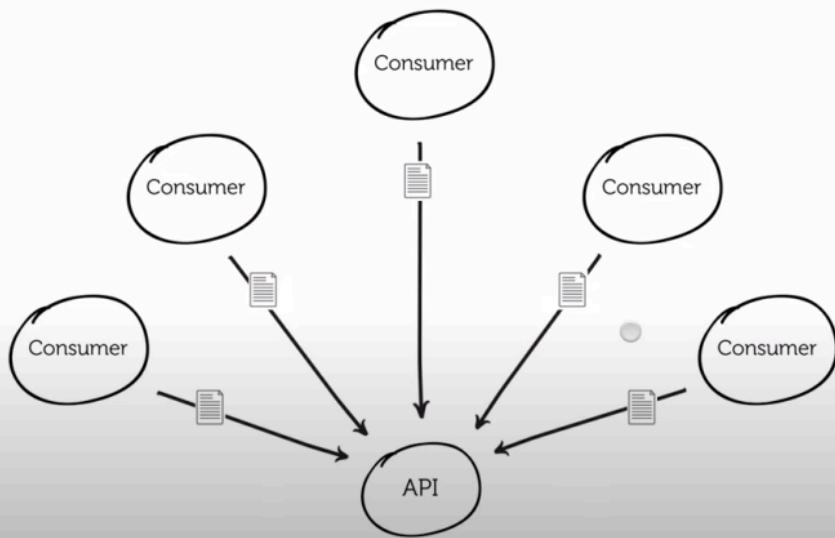
”



Consumer Driven Contracts

“ A strange inversion of reasoning... – D. Dennet ”

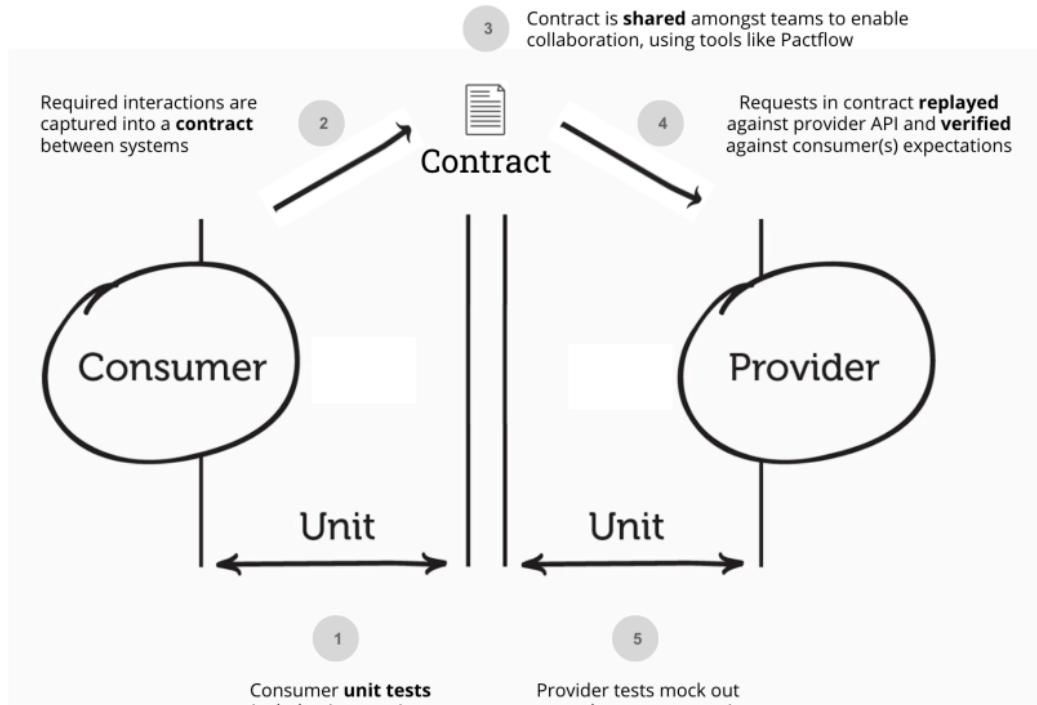
Consumer driven contracts



Pact

- Pact is a contract testing tool
- It is a JVM tool, but it can be used with any language
- It is a **consumer driven** tool
- It is a **mocking** tool
- It is a **verification** tool
- It is a **documentation** tool
- It is a **collaboration** tool

Pact - Overview



1 Problem statement

2 Option C : Contract Testing

3 Demo Time

4 Concluding thoughts

Hintergrund



1 Problem statement

2 Option C : Contract Testing

3 Demo Time

4 Concluding thoughts

Other tools

The image displays five separate screenshots of different open-source testing platforms, each with its own unique interface and branding:

- Dredd — HTTP API Testing Framework:** Shows a landing page with a shield logo and the text "DREDD No more outdated API Documentation".
- Cucumber:** Features a dark header with navigation links like "Tools", "DSLs", "Learn BDD", and "Resources". It includes a search bar and a "My Cucumber Profile" button.
- Karate Labs:** A landing page for an "Open-Source Test Automation Platform". It highlights "Unifying API testing, API Performance testing, API Mocks and UI testing". It features a "GET STARTED" button, social proof from "1000+ Happy customers" and "Capterra 4.8", and a "WATCH VIDEO" button.
- Specmatic:** A platform for "Micro Services done right without the pain of integration". It features a "GET Started" button, a "Watch Video" button, and a "Download" button.
- MICROCKS:** An "OPEN SOURCE KUBERNETES NATIVE TOOL FOR API MOCKING AND TESTING". It features a "GETTING STARTED" button, a "Cloud Native Computing Foundation" badge, and a "Watch Video" button.

Benefits of CDC

- Customer centric
- Fast feedback loops while still keeping test isolation

Where not to use CDC

- To replace functional tests or unit tests
- Error handling

Contract testing and Interface Segregation Principle

- Contract testing ensure that services can communicate with each other effectively

Contract testing and Interface Segregation Principle

- Contract testing ensure that services can communicate with each other effectively
- Interfaces are what services use to communicate with each other

