Contract Testing

Toni Masotti
masotti@bikeleasing.de
Bikeleasing-Service

November 26, 2023

Outline

Let's start

It all started with Bob...

...and it continues with Bob

It all started with Bob…

They do work on my local machine though 😢

Bob

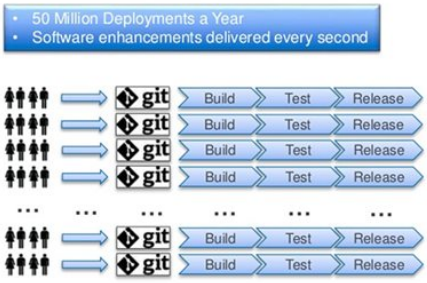
BOB ! the changes you made do not work on the live system 😡

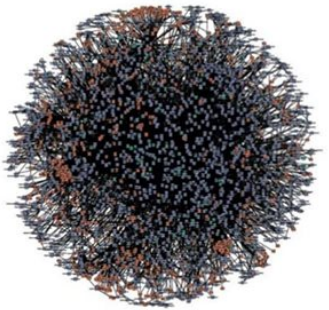
Angry chef

Bob has learnt the lesson...


MICROSERVICES
MICROSERVICES EVERYWHERE

Extreme examples...





Gigantic Web of Micro-services at Amazon

amazon web services

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

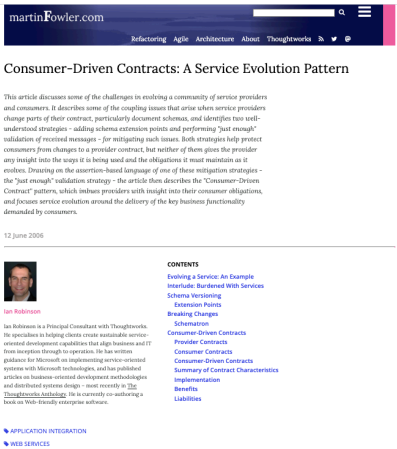
- Big promise: Independent deployability
- Fast feedback loops
- Strong team focus, ideally crossfunctional
- Loosely coupled architecture
- Fast onboarding
- Cost-efficiency on the long run
- Cloud native
- Tech-agnostic approach
- Scalability out of the box
- Improved Security (no SPOF, no SPOC)

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

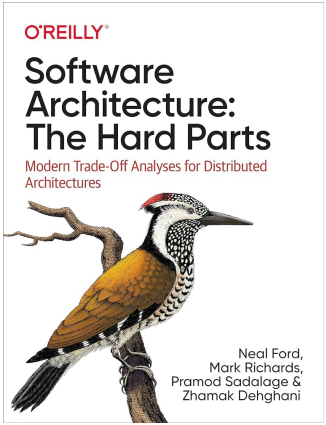
Everybody loves microservices

> *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.*



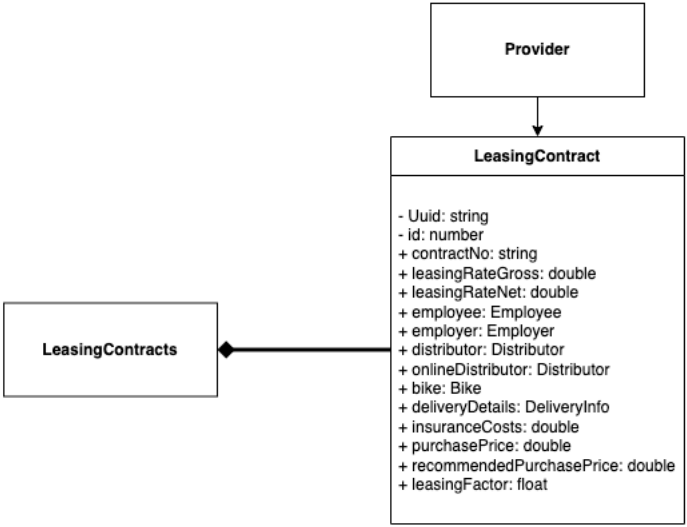
Is always a tradeoff

> *There is only one hard rule in software development:* ***Everything is a tradeoff****.*
>
> *– Neal Ford*

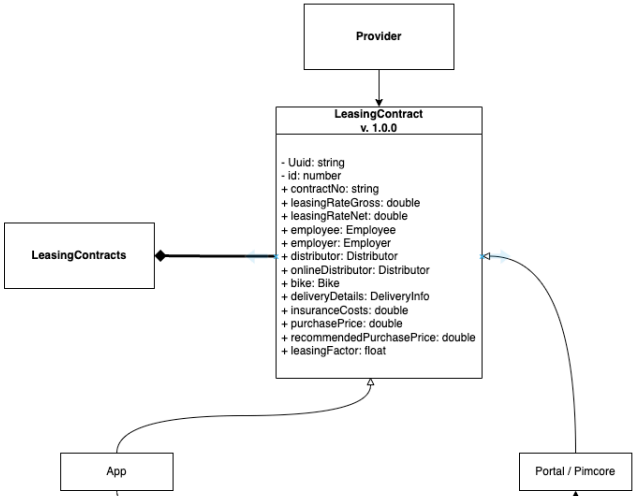


Bob's company is being very successfull...

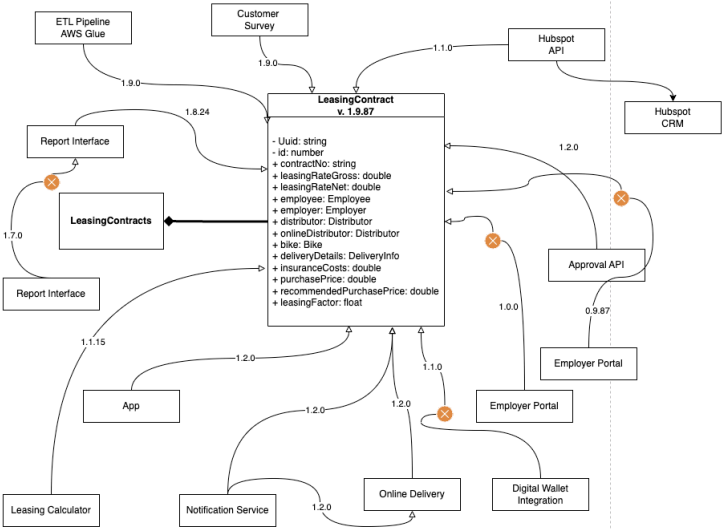
A company growth... so its architecture



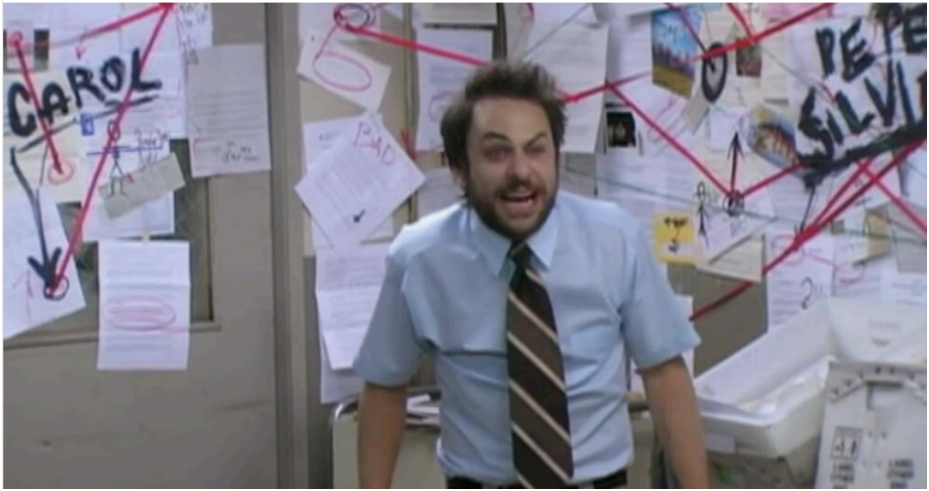
A company growth... so its architecture



A company growth... so its architecture



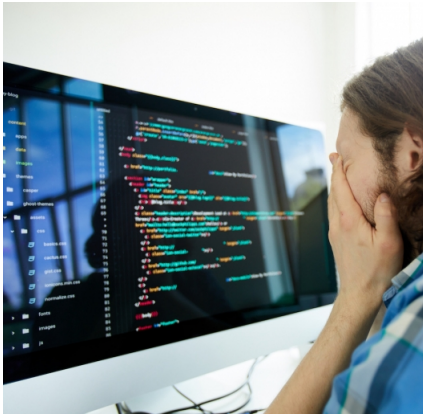
Bob is happy...



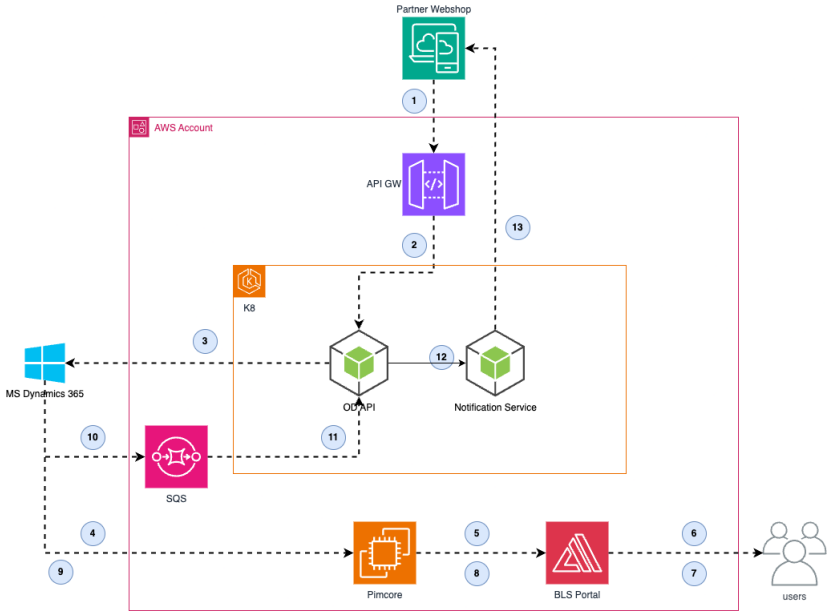
The Problem...

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

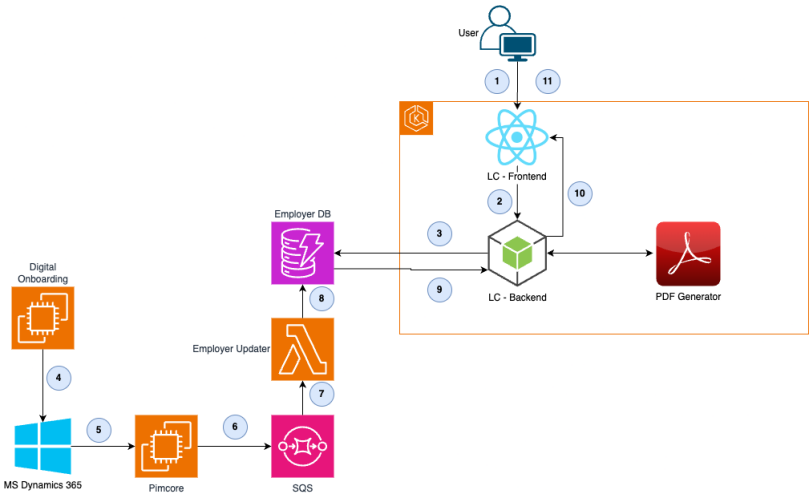
- Dependency hell
- Devs don't have confidence in deploying changes
  - Consequences of changes are hard to track
  - Distributed architectures are hard to test



Architecture example: Online Delivery



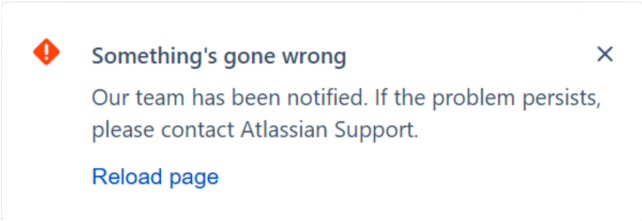
Architecture example: Leasing Calculator



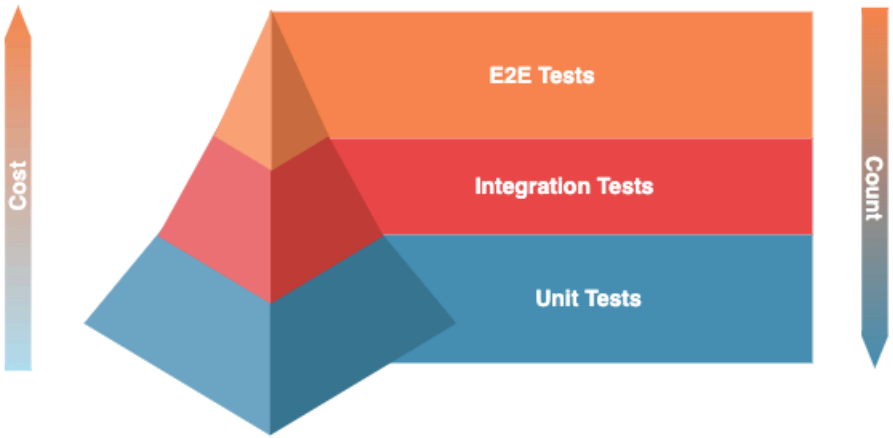
Updating a microservice: the sad Atlassian case

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

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



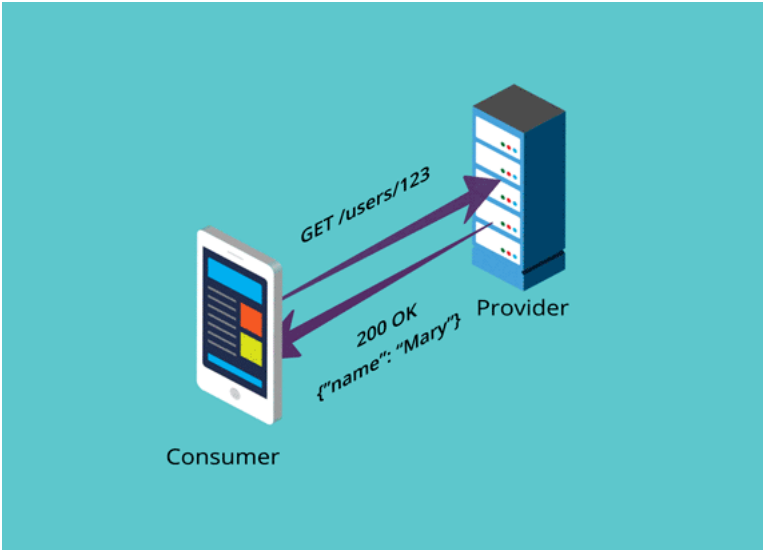
Test Typology



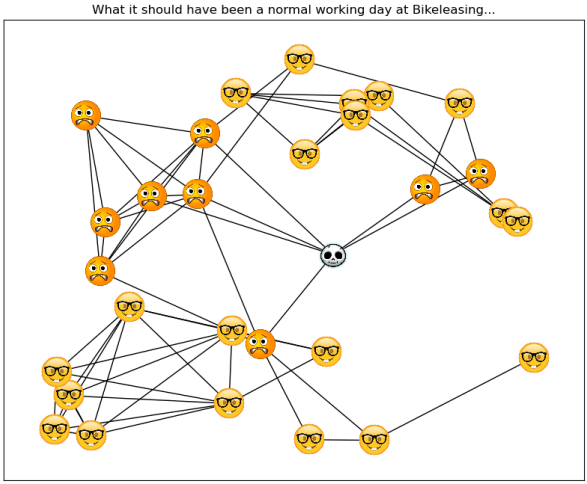
The problem with Unit Tests alone

- 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

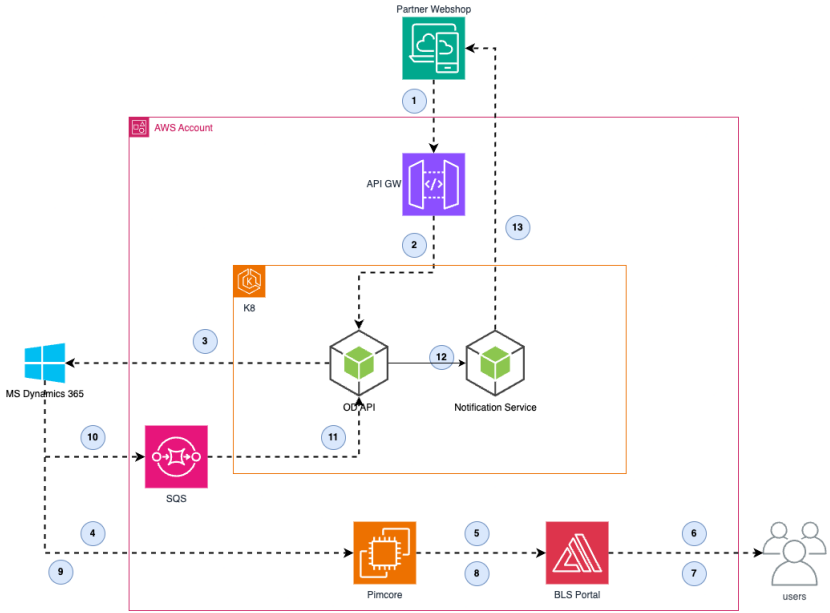


Problem with E2E Tests

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



Distributed $\neq$ 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?

Contracts

Contracts

A formal agreement between parties or individual. Synonyms: pact, agreement, protocol, deal
— Oxford English Dictionary

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)

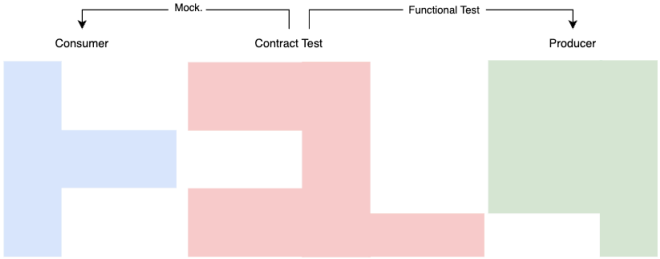
A pact between who?

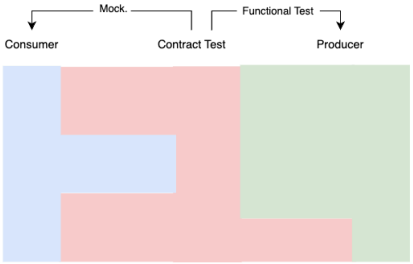
- **Consumer**: the service that consumes the information (typically closer to the user)
- **Provider**: the service that provides the information (typically closer to the data)

Where do we find this architecture?

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

Contract Test Role

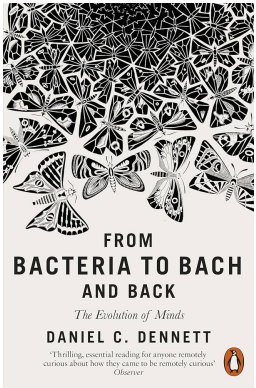


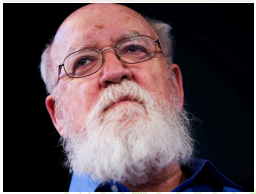


Pact - Consumer Driven Contracts

"'*A strange inversion of reasoning...*"'

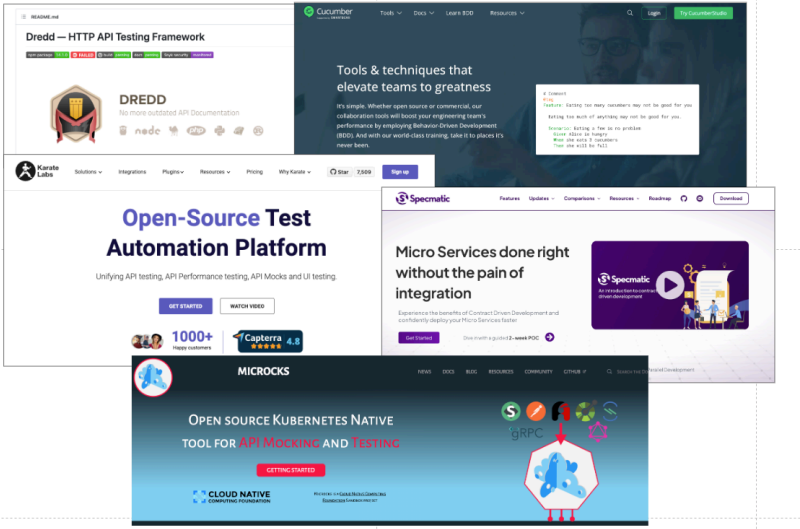
- Consumer Driven Contracts
- Pact is a contract testing tool
- Pact is a specification for consumer driven contracts
- Pact is a set of frameworks that support consumer driven contracts





Hintergrund

Other tools


README.md
Dredd — HTTP API Testing Framework
DREDD
No more outdated API Documentation
Cucumber
Tools
Docs
Learn BDD
Resources
Login
Try CucumberStudio
Tools & techniques that elevate teams to greatness
It's simple. Whether open source or commercial, our collaboration tools will boost your engineering team's performance by employing Behavior-Driven Development (BDD). And with our world-class training, take it to places it's never been.
Karate Labs
Solutions
Integrations
Plugins
Resources
Pricing
Why Karate
Star 7,509
Sign up
Open-Source Test Automation Platform
Unifying API testing, API Performance testing, API Mocks and UI testing.
GET STARTED
WATCH VIDEO
1000+
Happy customers
Capterra 4.8
Specmatic
Features
Updates
Comparisons
Resources
Roadmap
Download
Micro Services done right without the pain of integration
Experience the benefits of Contract Driven Development and confidently deploy your Micro Services faster
Get Started
MICROCKS
NEWS
DOCS
BLOG
RESOURCES
COMMUNITY
GITHUB
OPEN SOURCE KUBERNETES NATIVE TOOL FOR API MOCKING AND TESTING
GETTING STARTED
gRPC
CLOUD NATIVE COMPUTING FOUNDATION

Conclusions

- ja, something

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


THANK
YOU