Contract testing

-

Verification across service boundaries

Who am I?

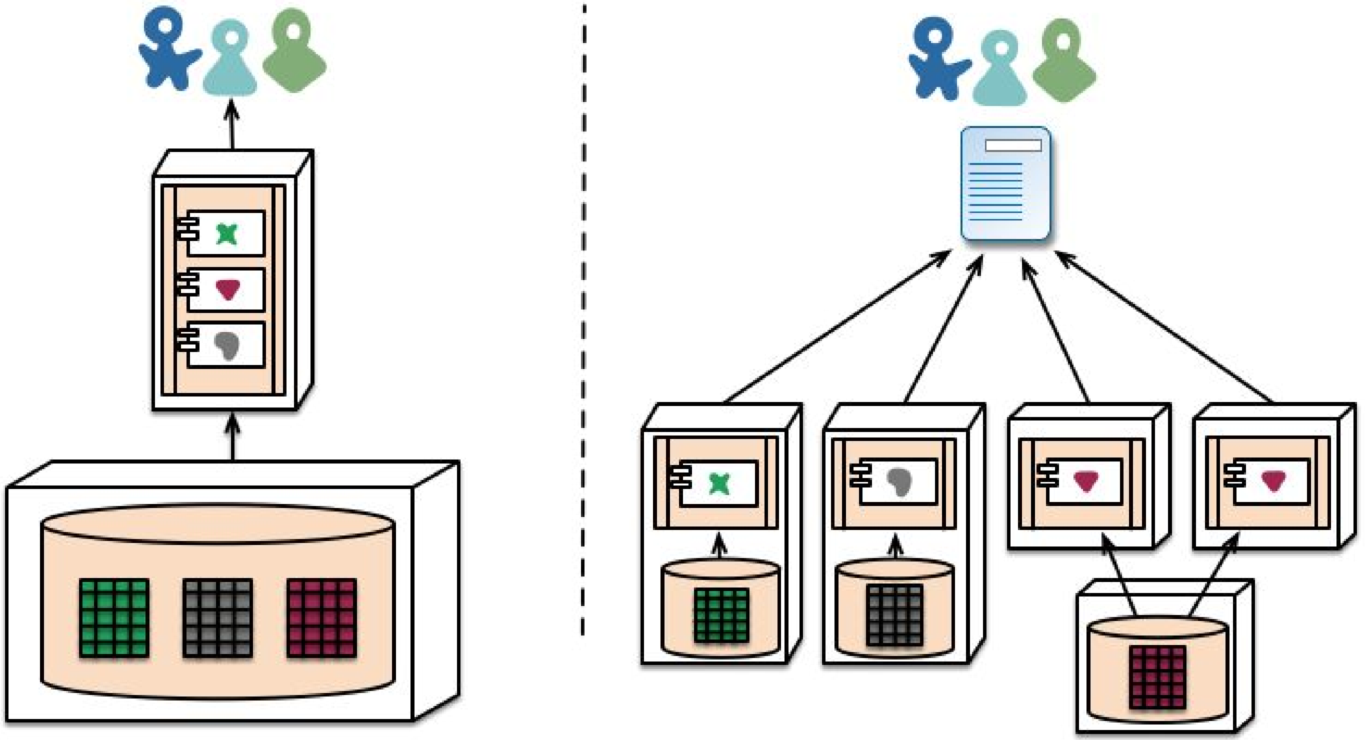
Peter Czibik
@peteyycz
github.com/peteyycz

TDD
ALL CODE IS GUILTY
UNTIL PROVEN INNOCENT

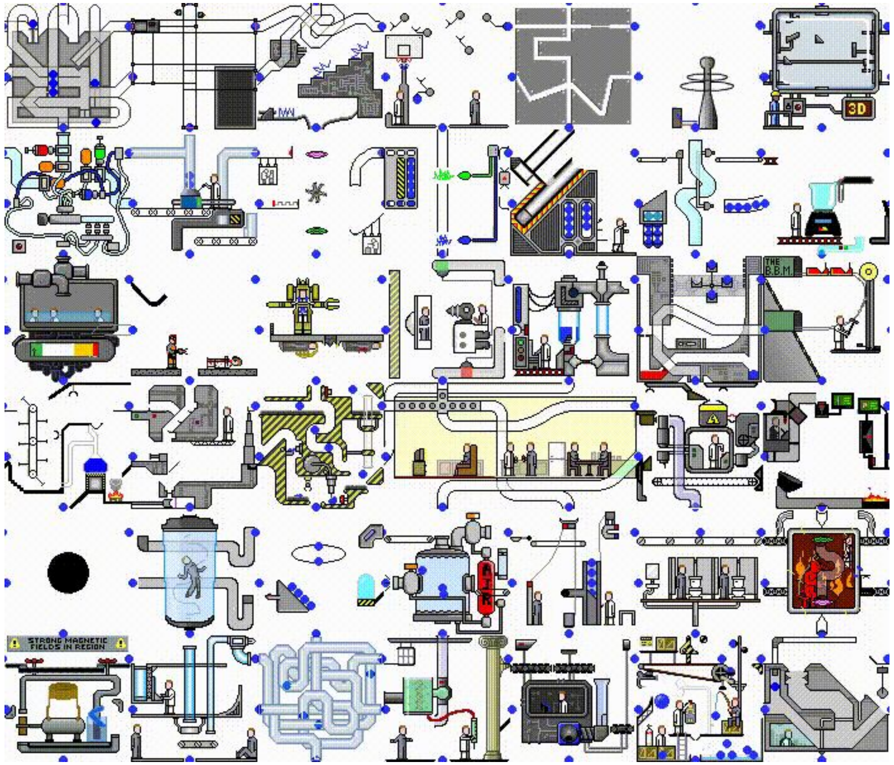
- Nerd
- Senior Node JS developer
- Linux enthusiast
- Lover of programming languages
- Generally a nice guy

RisingStack

We help companies succeed with microservices



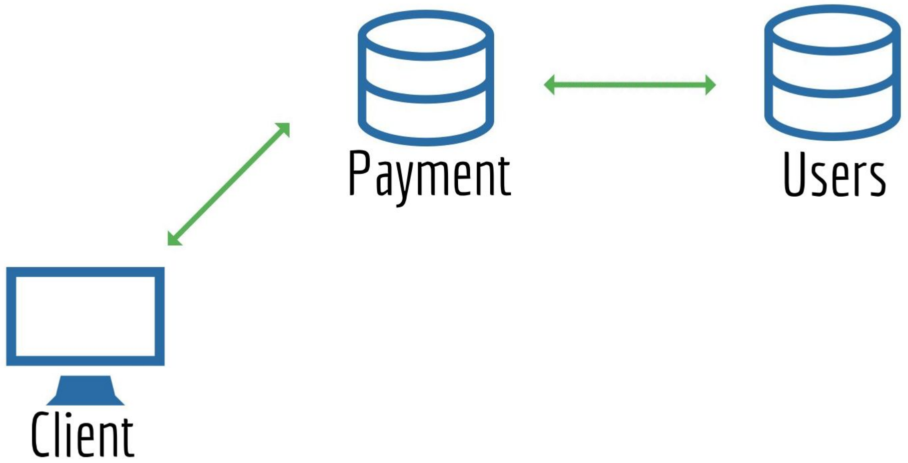
Who here works with microservices?



Everyone loves microservices

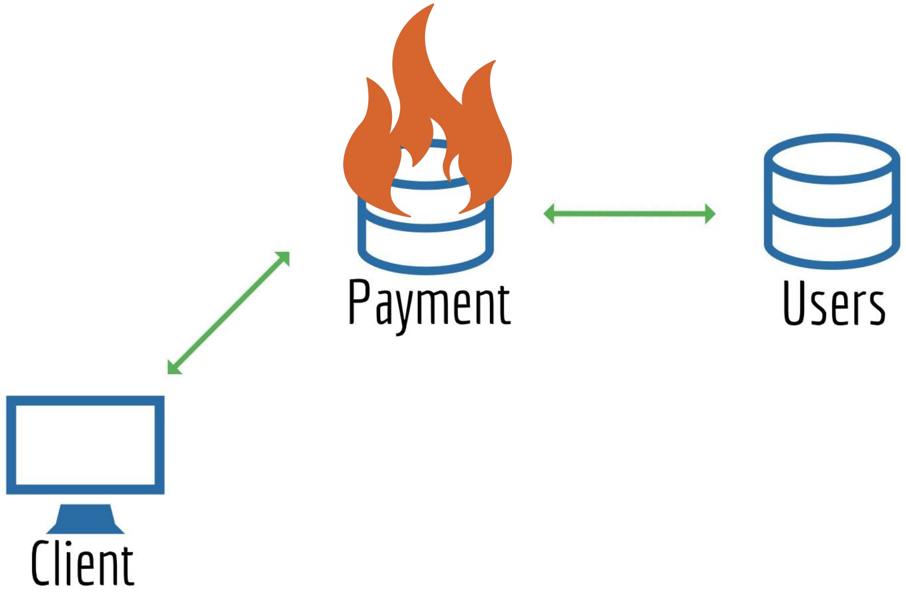
A sad story of Atlassian

- Two independent services: Payment and Users



A sad story of Atlassian

- Two independent services: Payment and Users



A sad story of Atlassian

- Changing a single letter in a json response

```
{
        “user”: [ … ]
}
{
        “users”: [ … ]
}
```

There’s no worse thing than not letting users pay.

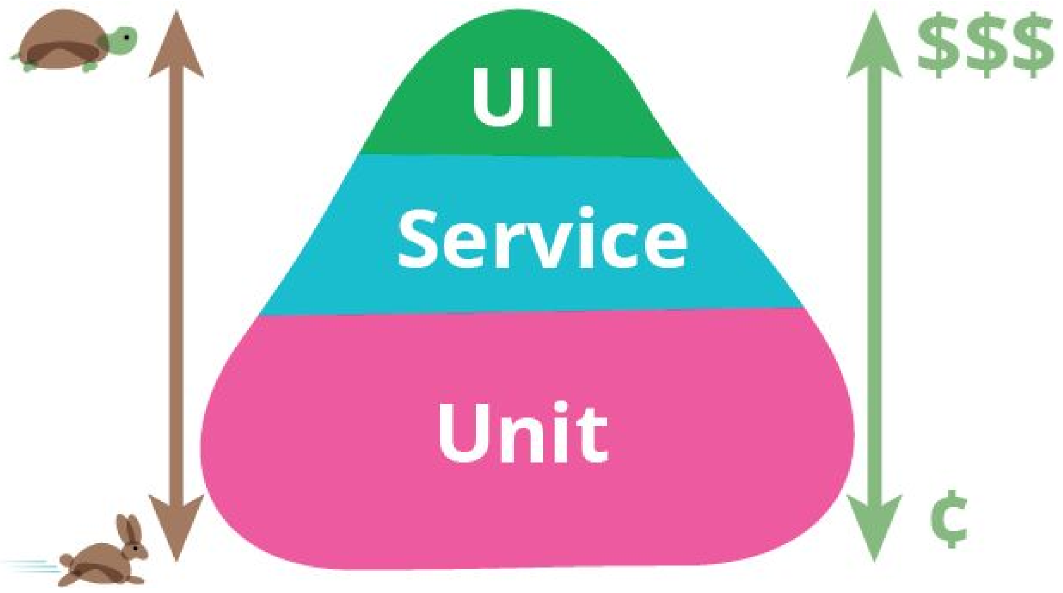
How can we avoid problems like that?

Not by printing test pages, but writing some tests!

What kind of tests?

Testing

- Unit tests
- Integration tests
- End-to-end tests



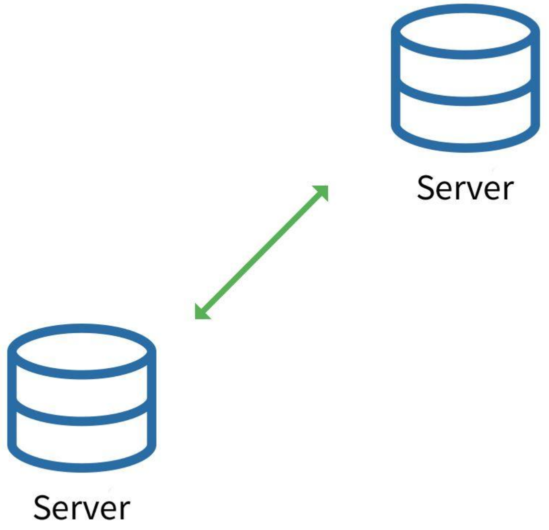
Those are not enough.

Indeed.

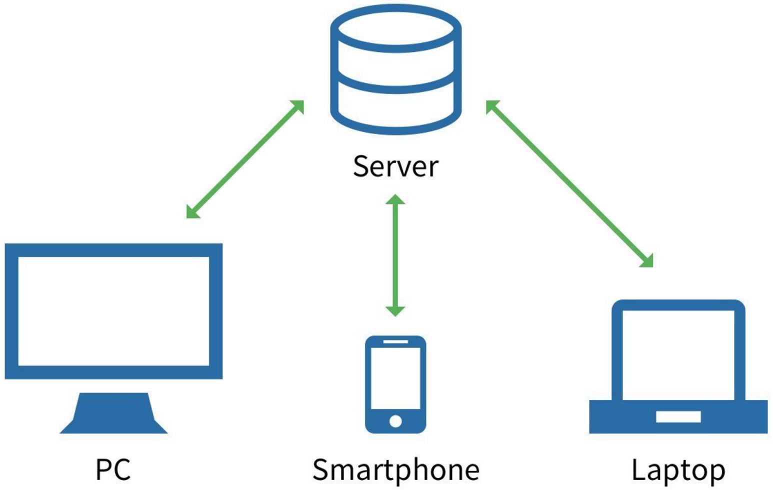
Testing beyond borders

- Separate codebases
- Individual deployments
- Multiple languages with shared protocol

Testing beyond borders



Testing beyond borders



These concepts apply to both worlds.

Solution A

Using an instance of the upstream service

- Pain to spin up other team’s services
- Trustworthy
- Expensive
- Unreliable

Solution B

Mocking

- Light on resources
- No data pollution
- Spares maintenance cost
- Points out issues quickly
- Reliable
- Not trustworthy

Wrap up

Mocking vs. Extra instance

Mocking	Extra instance
- Reliable	- Unreliable
- Light on resources	- Resource heavy
- Cheap ($$$ and code)	- Expensive ($$$ and code)
- CI compatible	- Not CI compatible
- Not trustworthy	- Trustworthy

Wrap up

Mocking vs. Extra instance

Mocking

- Reliable
- Light on resources
- Cheap ($$$ and code)
- CI compatible
- Not trustworthy

Extra instance

- Unreliable
- Resource heavy
- Expensive ($$$ and code)
- Not CI compatible
- Trustworthy

Solution C

C for Contract

Consumer Driven Contract Testing

- Implementation steps:
    a. Client makes expectations
    b. Client uses these expectations to test
    c. Transfer expectations to server
    d. Server uses same expectations to test
- Two roles:
    - Consumer
    - Provider

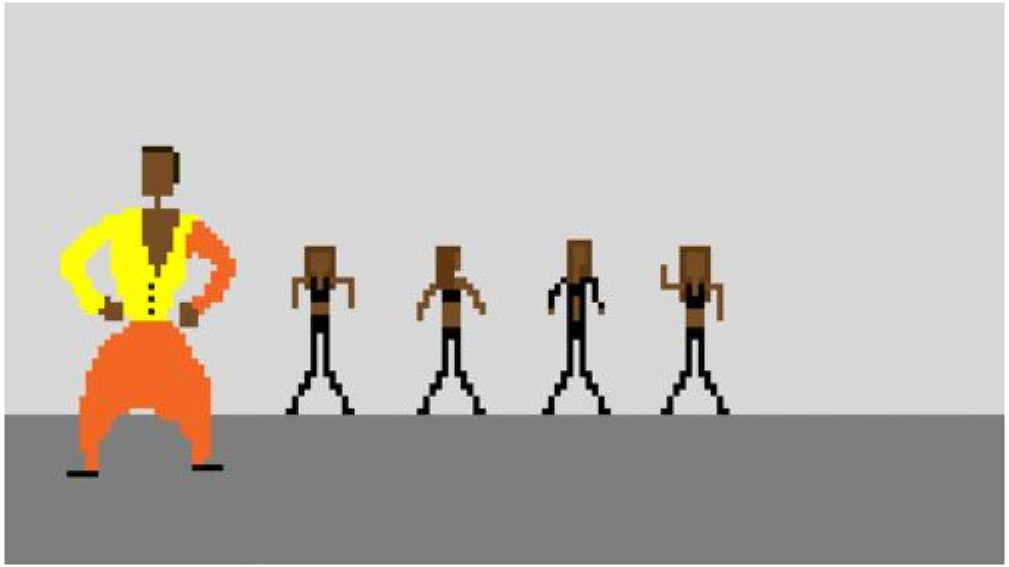
PACT

Meet Pact

Pact

- Implementation of a Consumer Driven Contract Test Suite
- JSON (language independent)
- Libraries already exist in multiple languages
    - .NET
    - JVM (Java & Scala)
    - Javascript

Everyone stop.



It’s demo time!

Contract testing

- Real data
- No extra infrastructure
- Fast (with occasional synchronization)
- Easy to scale
- Low setup
- Stable
- Reliable

Contract testing

- Real data
- No extra infrastructure
- Fast (with occasional synchronization)
- Easy to scale
- Low setup
- Stable
- Reliable

Even if you do, be polite about it.

Thank you for your attention

¿Questions?