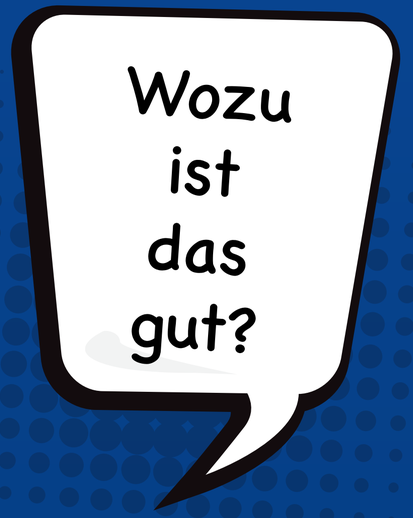
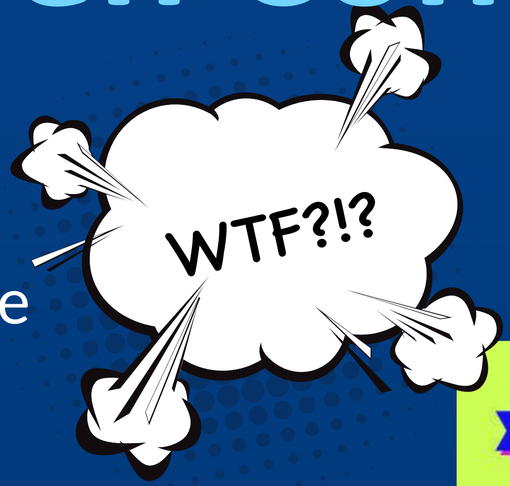
API-Tests

Contract-Tests

Consumer-driven Contract-Tests





Thomas Much, Techniker Krankenkasse

DevLand 2026

Europapark Rust, 13. März 2026

@thmuch

Architektur

Fachlichkeit (Domäne)

INKL. KLEINER LIVE-DEMOS

Contract-Tests

Grundlagen.

Konzepte.

„Warum“ verstehen.

Was wird damit gelöst?

Herausforderungen.

Ausblick.

Agilität
+
Collaborative *

Architektur
+
Continuous *

Technical
Coach

Coden, Testen
+
Craft

Die Techniker

www.tk.de/IT

[ˈtoːmas] 👋

@thmuch

Testen, testen, testen …

Was sind die „Enden“?

Systemintegrationstests bzw. „End-to-end“ (E2E)

Systemtests

(Komponenten-)Integrationstests

Unit-Tests

API

Immer alles „echt“ mittesten?

◀ · · · · · · · · · · · · · · · **Systemintegration**stests bzw. „End-to-end“ (E2E) · · · · · · · · · ▶

„Ansteckung“ bzw. „Infektion“ („Contagion“)

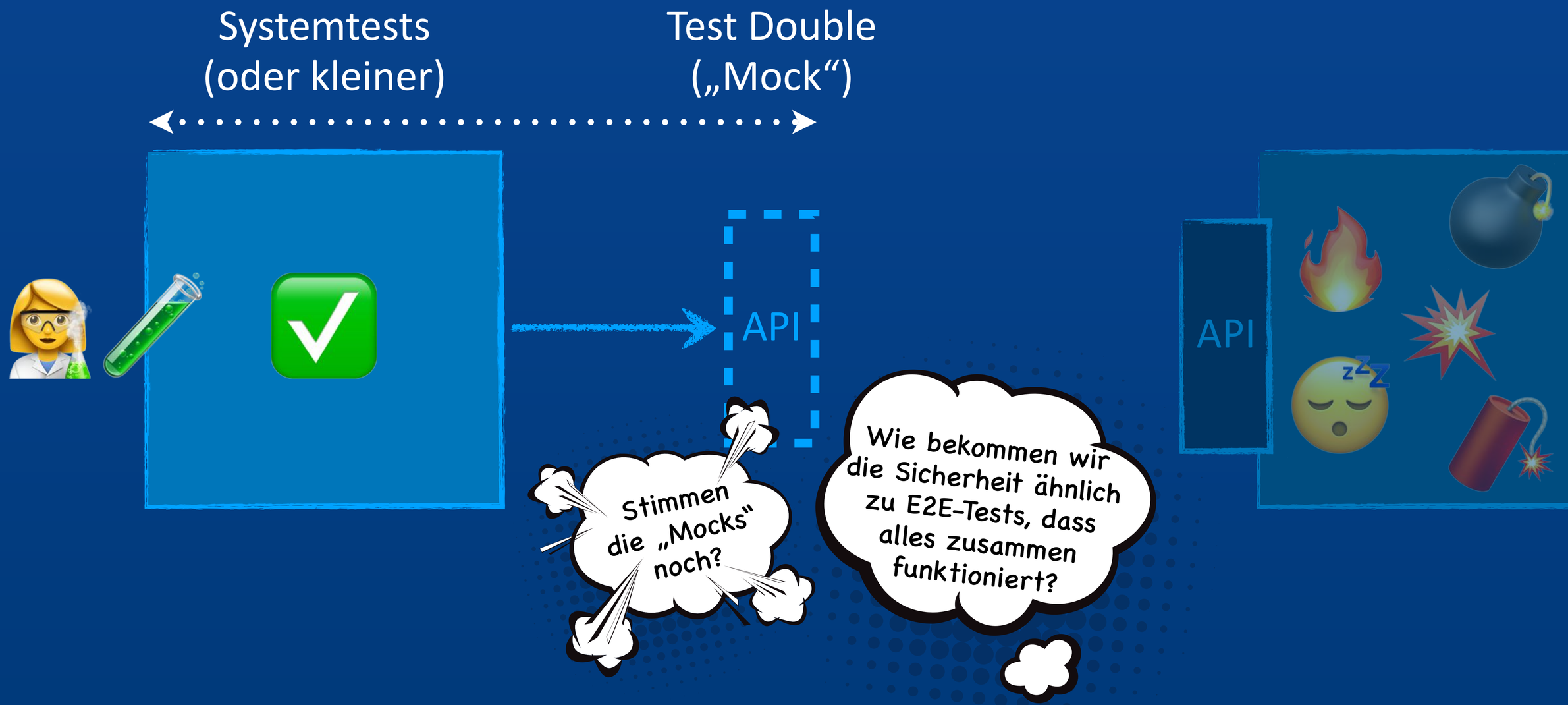
instabil („flaky“)

langsam

kompliziert

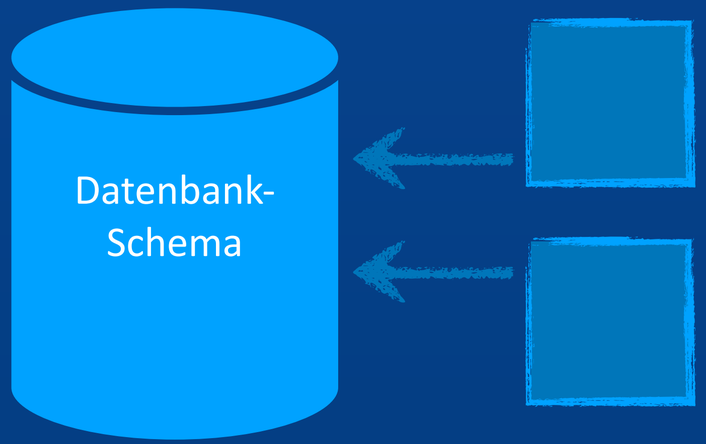
Aber nur so testen wir die Realität!?

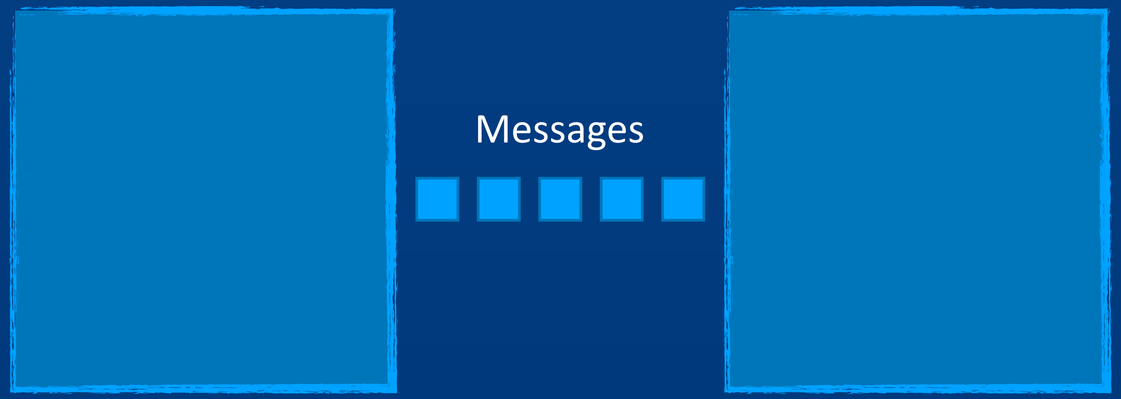
Isolierter testen heißt stabiler testen



Was sind „APIs“?

“Application Programming Interfaces” – Programmierschnittstellen






```
public interface WetterAbfrage {

    WetterInfo wetterIn(Stadt stadt);
}
```

Was sind „Contracts“?

Aussagen über **strukturelle Validität** (inkl. Typen) und **Integrationskompatibilität**.

yaml

```yaml
WetterInfo:
  type: object
  required:
    - stadt
    - temperatur
  properties:
    stadt:
      type: string
      description: lokale
    temperatur:
      type: integer
      description: Temper
      minimum: -90
      maximum: 60
```

JSON

```json
{
  "consumer": { "name": "wetter-app" },
  "provider": { "name": "wetter-service" },
  "interactions": [
    {
      "description": „aktuelles Wetter der Stadt",
      "request": {
        "method": "GET",
        "path": "/wetter",
        "query": "stadt=Hamburg"
      },
      "response": {
        "status": 200,
        "body": {
          "stadt": "Hamburg",
          "temperatur": 21
        },
        "matchingRules": {
          "body": {
            "$.temperatur": {
              "matchers": [
                {
                  "match": „integer"
                },
                …
```

XML

```xml
<definitions xmlns="htt
             xmlns:xs="
             xmlns:soap
             targetName

  <types>
    <xs:schema targetNam
      <xs:element name="G
        <xs:complexType>
          <xs:sequence>
            <xs:element name="stadt" type="xs:string"/>
```

```
public WetterInfo wetterIn(Stadt stadt) {
```

Vorbedingung(en)

Invariante(n)

Nachbedingung(en)

```
}
```

„Design by Contract“ (DbC) – Bertrand Meyer, 1986+

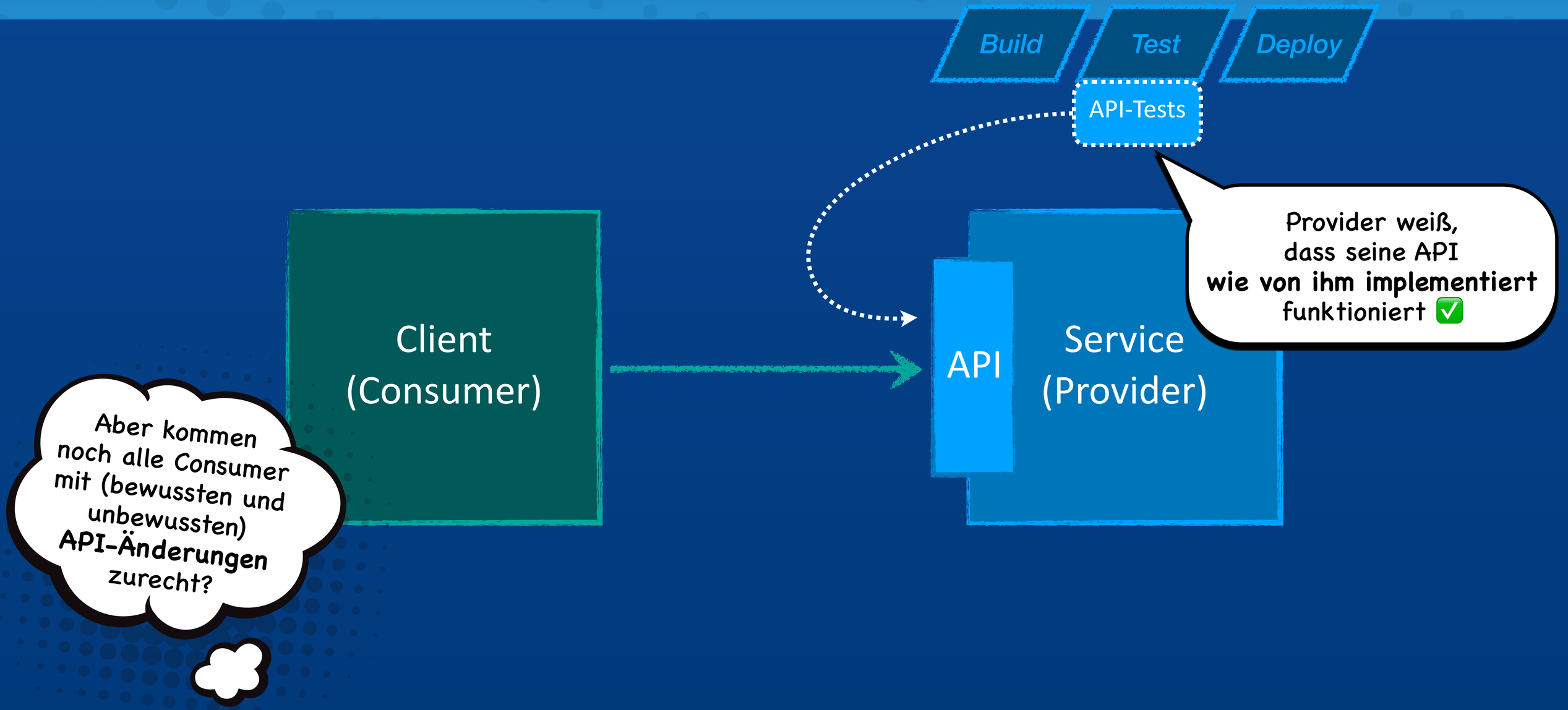
```
public interface WetterAbfrage {

    /**
     * Aktuelle Temperatur in °C.
     * @return niemals null
     * @throws NPE wenn stadt==null
     */
    WetterInfo wetterIn(Stadt stadt);
```

Und was ist an Contracts „consumer-driven“?

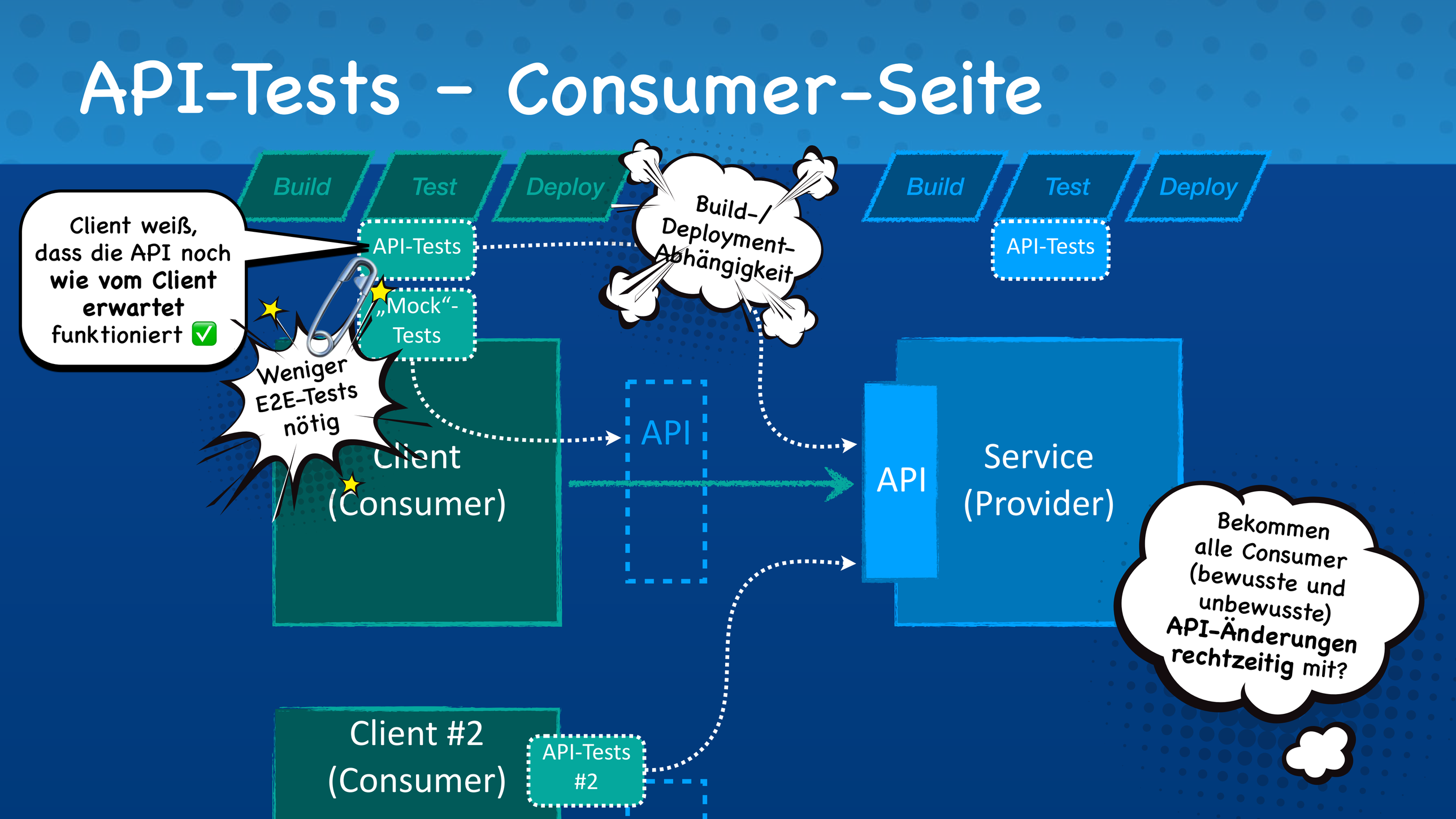
Super Frage!

Fangen wir vorne an.

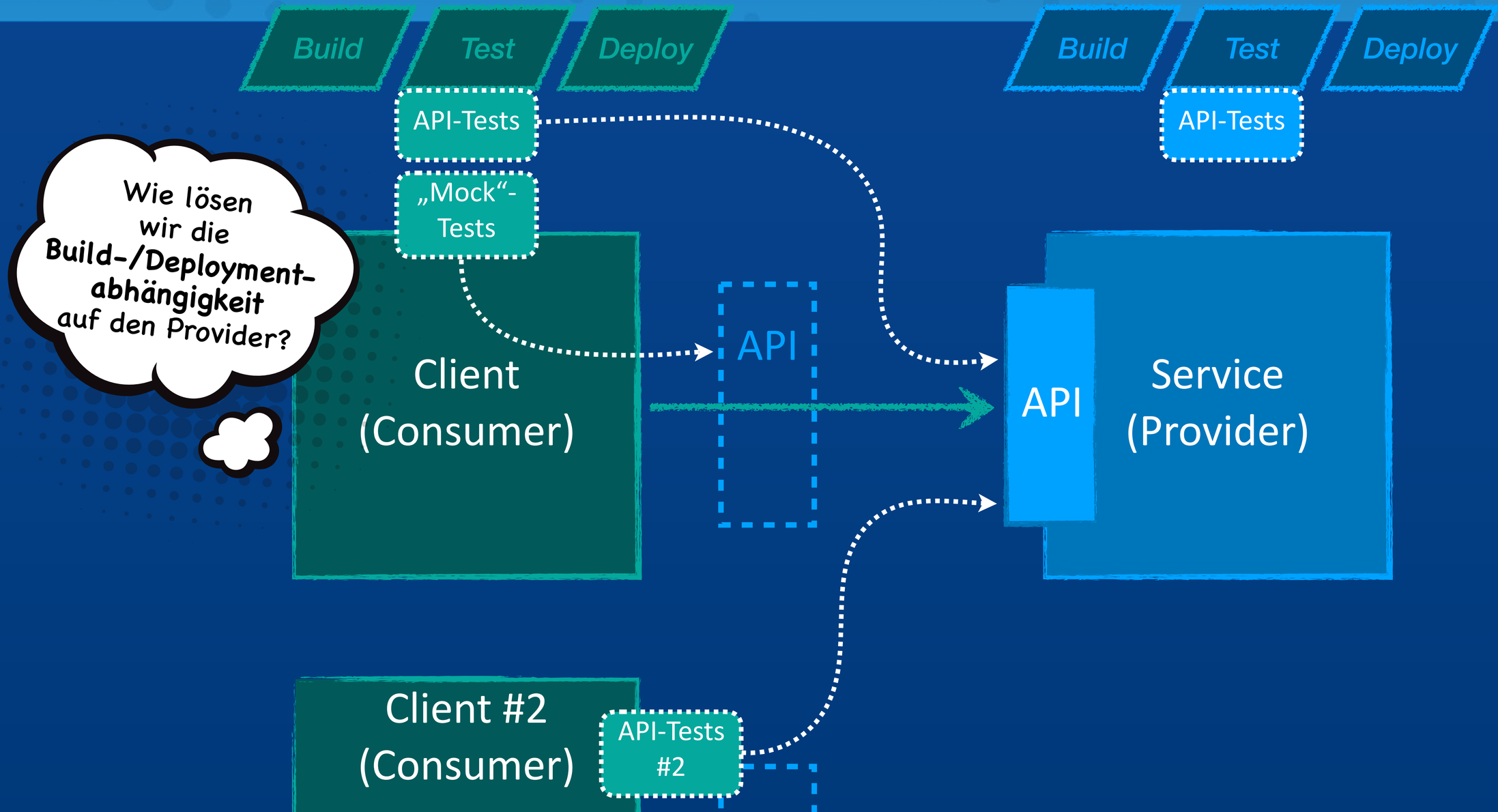
API-Tests – Provider-Seite



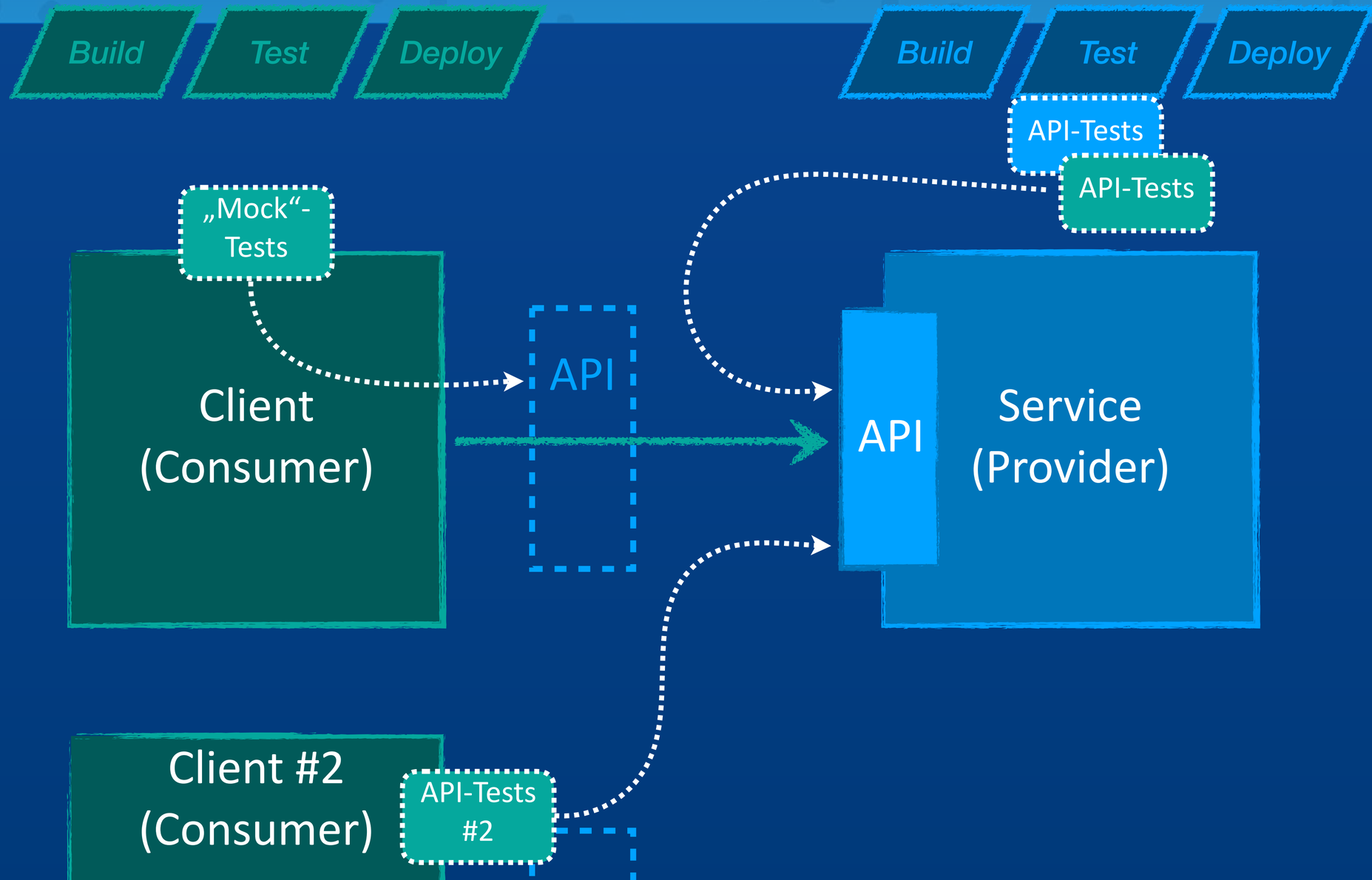
API-Tests – Consumer-Seite



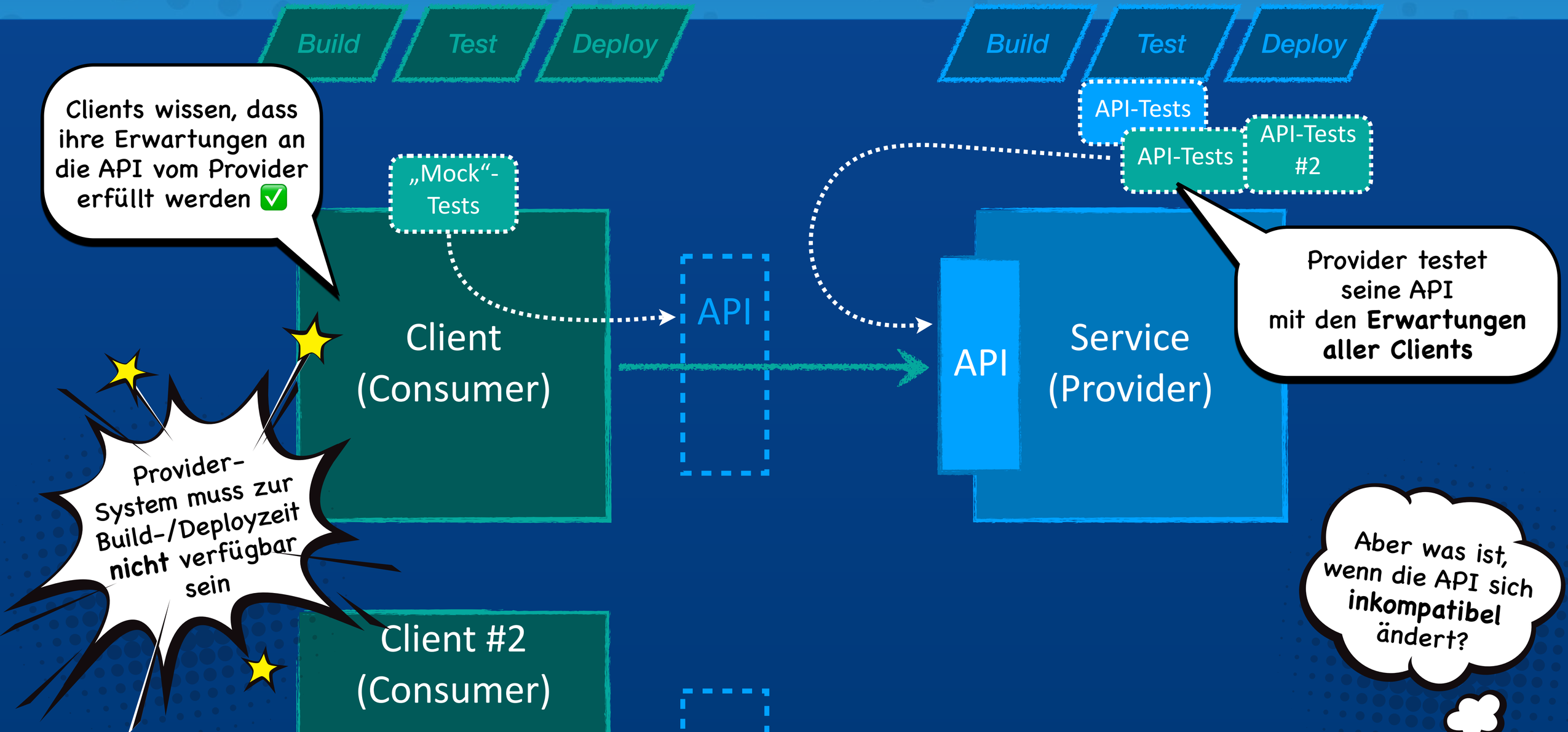
API-Tests – Consumer-Seite



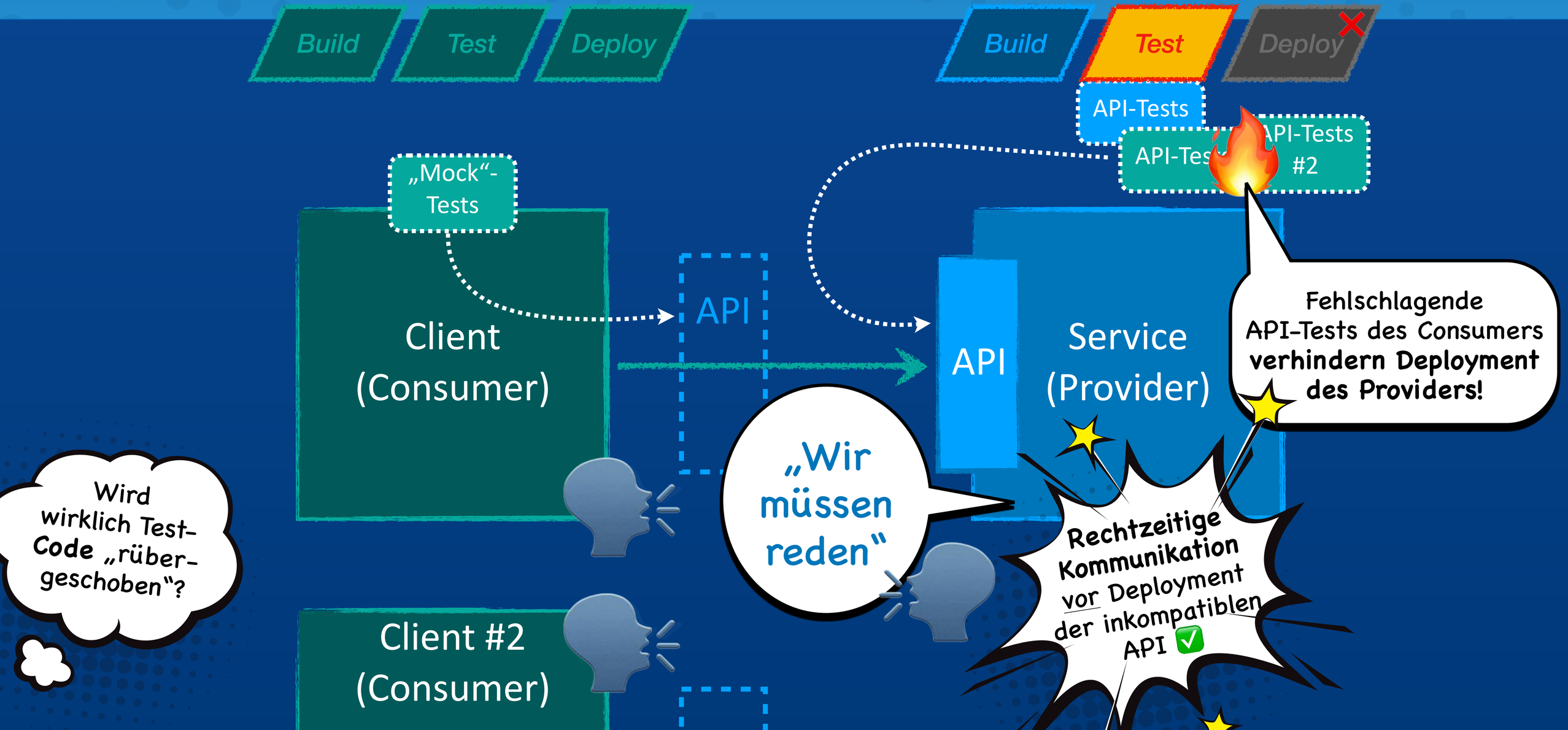
API-Tests – Consumer-driven



API-Tests – Consumer-driven



API-Tests – Consumer-driven



Consumer-driven Contracts

Strikte Versionierung nicht zwingend. **Kompatibilität** zu allen Erwartungen reicht!

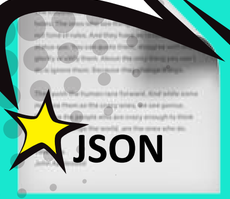
Deklarative Beschreibungen („Verträge“) der Consumer-**Erwartungen** an die API

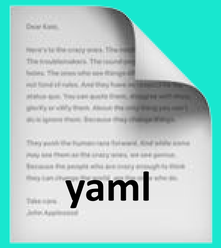
Client (Consumer)

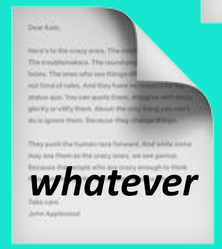
Client #2 (Consumer)

„publish“

„publish“







Broker
Registry
Repository
whereever

„verify“

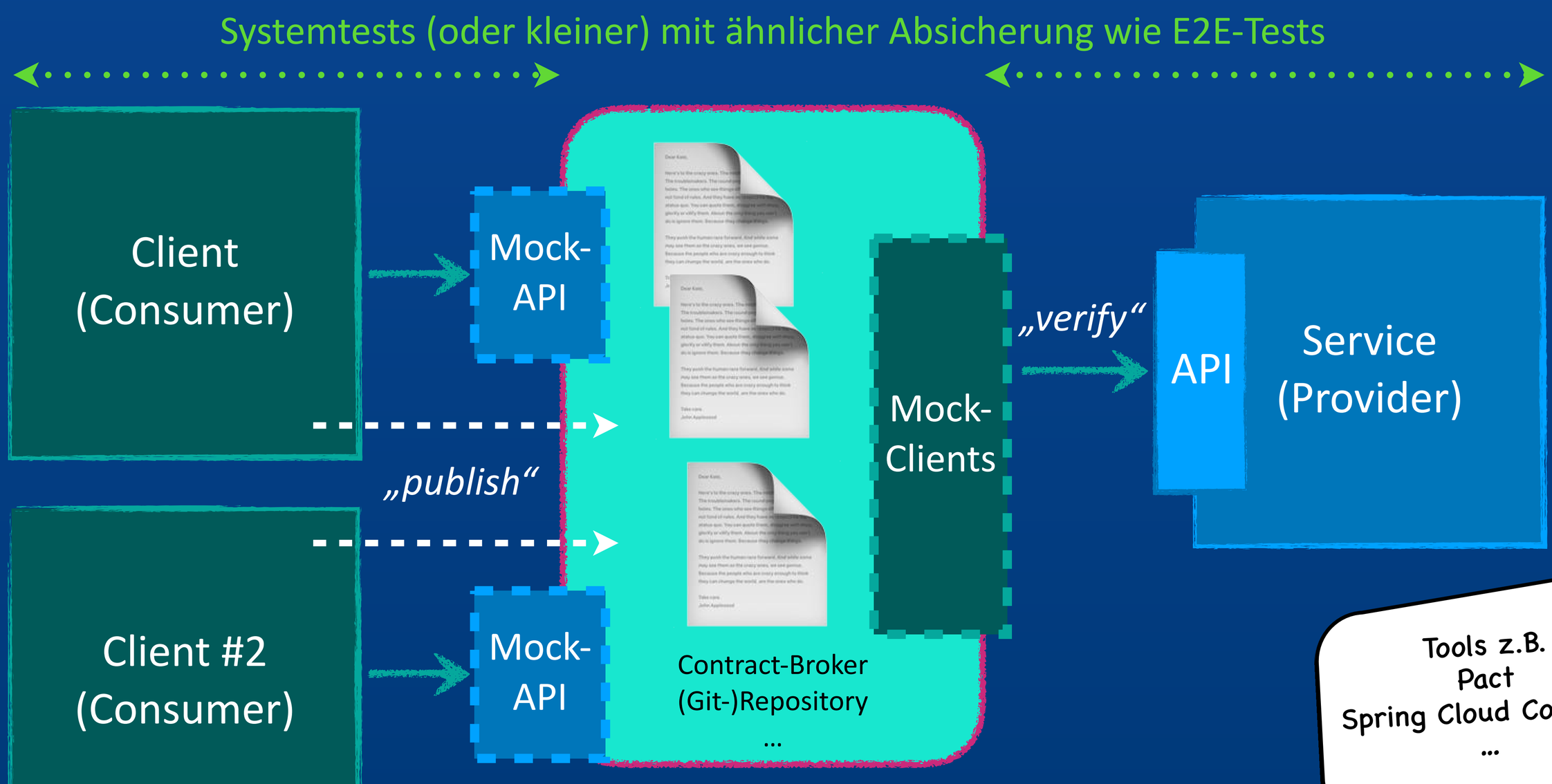
API

Service (Provider)

Provider muss Erwartungen **aller** Consumer kennen

Wie funktionieren typische **Tools?**

Consumer-driven Contract-Tests

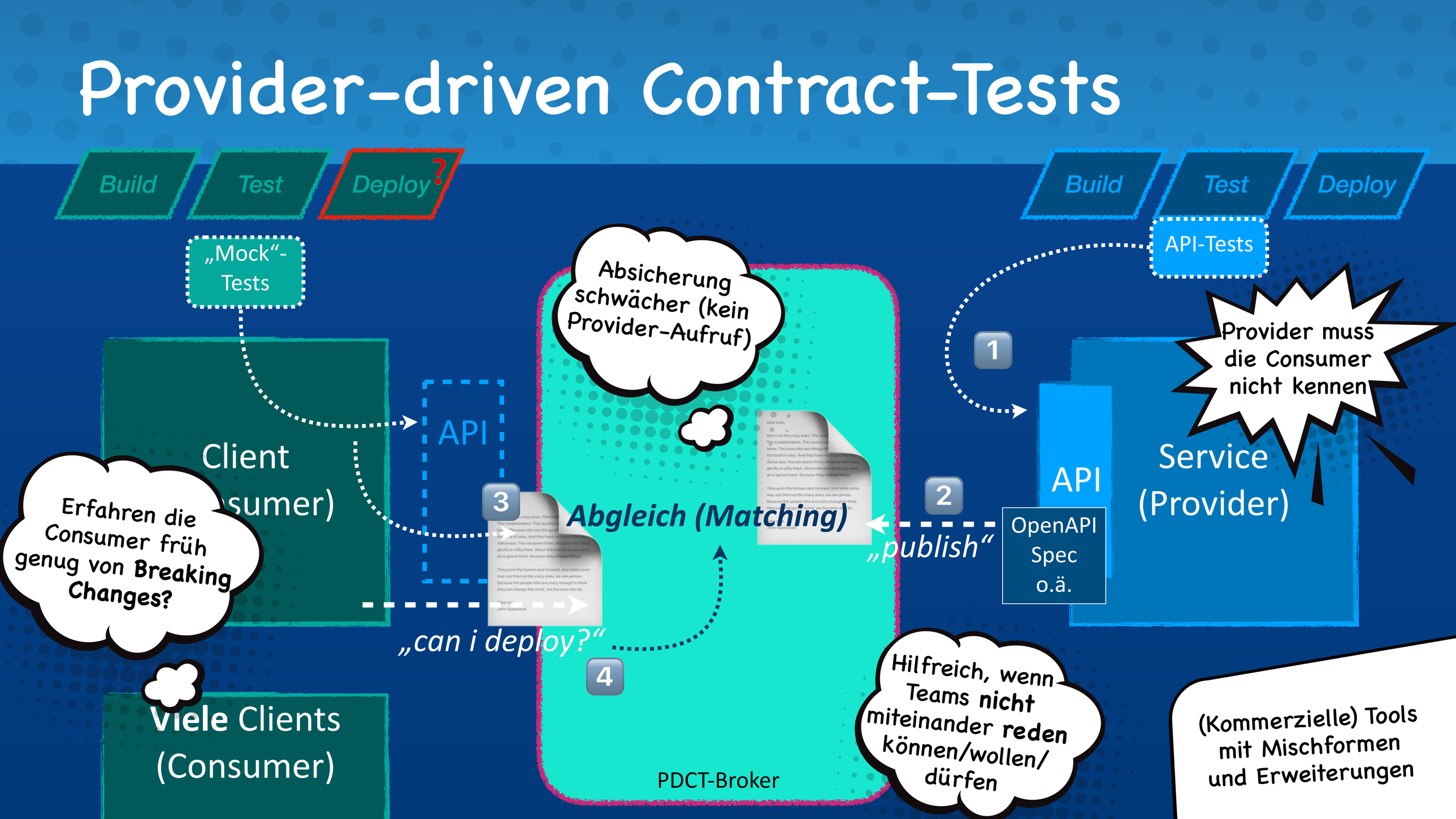



DEMO-
TIME

Wenn es „consumer-driven“ gibt …

… gibt es dann auch „provider-driven“?

Provider-driven Contract-Tests



Provider-driven Contracts ohne Broker

Build Test Deploy?

Build Test Deploy

Passen die generierten Artefakte wirklich zur **fachlichen** Erwartung?

Client (Consumer)

API

Erfahren die Consumer früh genug von **Breaking Changes?**

„generate“

3

dependency update (PR)

2

Renovate Dependabot

OpenAPI Spec o.ä.

Nexus o.ä.

1

„publish“

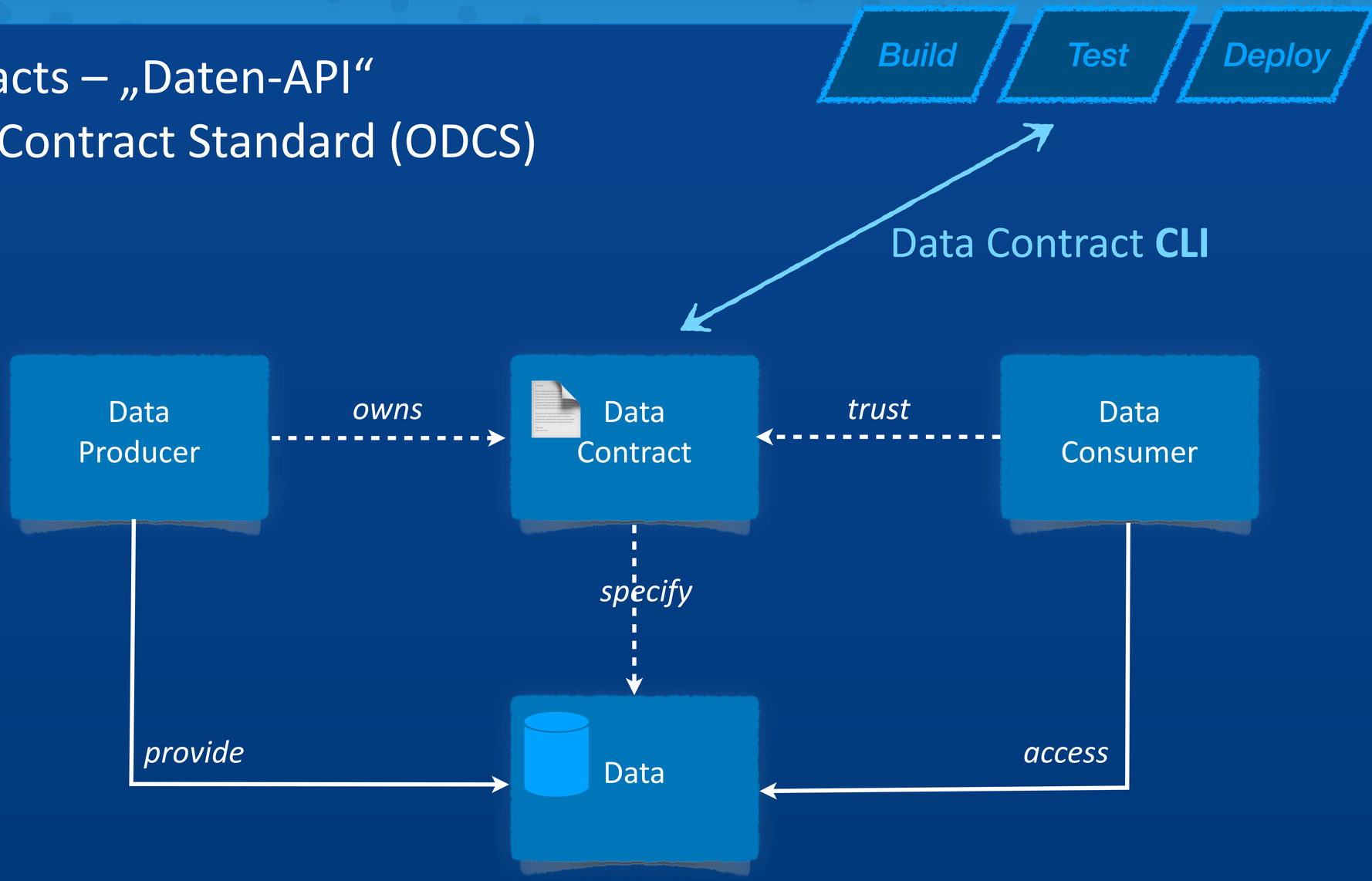
API

Service (Provider)

Versionierung (Releases) und Umgang mit Breaking Changes wird **deutlich wichtiger** ⚠️

Data Contract Tests

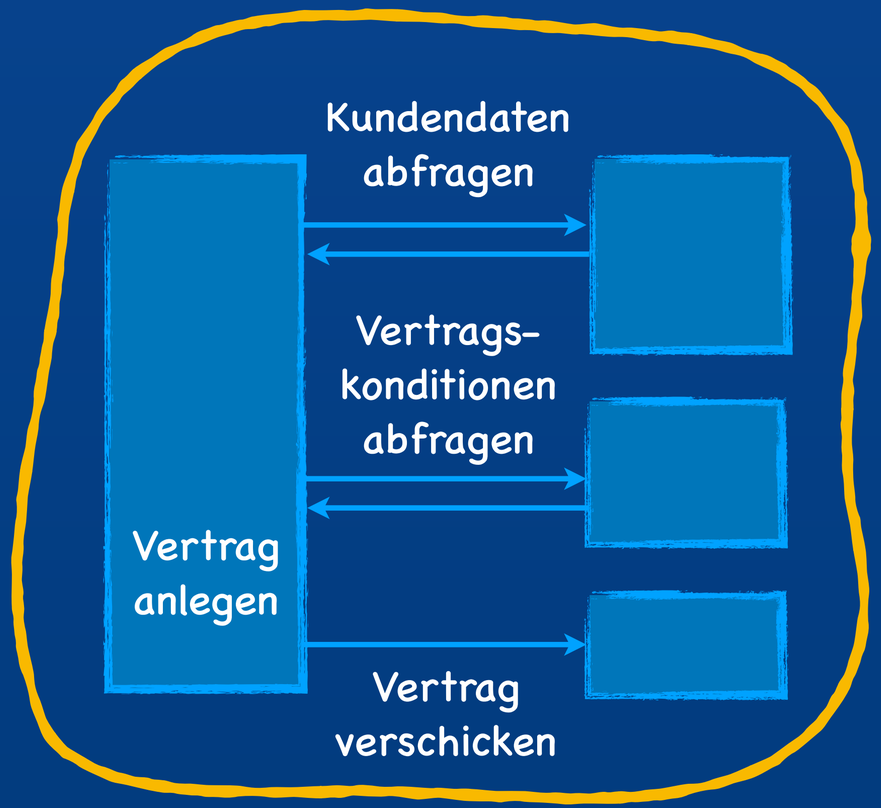
Data Contracts – „Daten-API“
Open Data Contract Standard (ODCS)



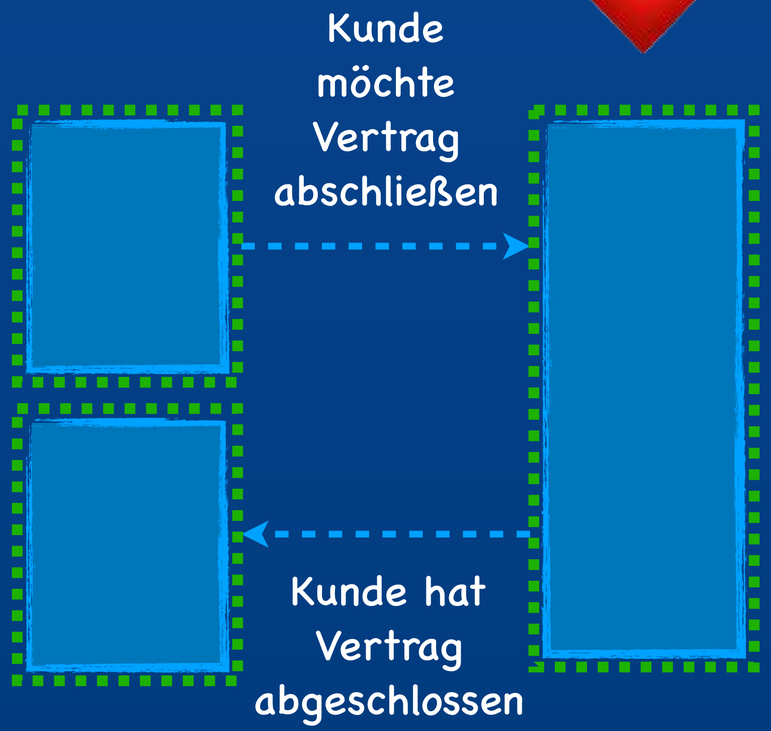
https://datacontract.com/
https://cli.datacontract.com/

Ein paar abschließende Gedanken

Technische vs. fachliche APIs



evtl. Anzeichen für (Distributed) „Big Ball of Mud (BBoM)“



„Self-contained Systems (SCS)“ Fachlich abgeschlossene (Teil-)Systeme

Testen Contract-Tests Fachlichkeit?

✅ **Struktur**elle Validität

✅ Statische Typsicherheit

✅ Beobachtbares Verhalten an einer Schnittstelle

✅ Integrations**kompatibilität**

❌ Geschäftslogik

❌ Datenrichtigkeit

❌ Fachliche Korrektheit

Fachliche Tests (Unit-Tests, Integrationstests)
Nur, wenn nicht über andere Testarten testbar: API-Tests

https://docs.pact.io/consumer/contract_tests_not_functional_tests

APIs und „Postel's Law"

"be conservative in what you do,
be liberal in what you accept from others"

Sei präzise beim Senden und tolerant beim Empfangen.

– RFC 671 (TCP, 1974), „Robustheitsprinzip", Jon Postel zugeschrieben

✅ Vorwärtskompatibilität: Unbekannte Felder ignorieren
✅ Optionale Felder nicht zwingend machen
✅ Toleranz bei Reihenfolge und Whitespace

❌ Falsche Datentypen akzeptieren
❌ Ungültige Werte stillschweigend korrigieren

etc.

Contract Testing

Sei präzise beim Senden
*und **überlege gut,***
wo du bei Empfangen tolerant sein kannst/solltest.

https://datatracker.ietf.org/doc/html/rfc761#section-2.10
https://www.rfc-editor.org/rfc/rfc9413.html

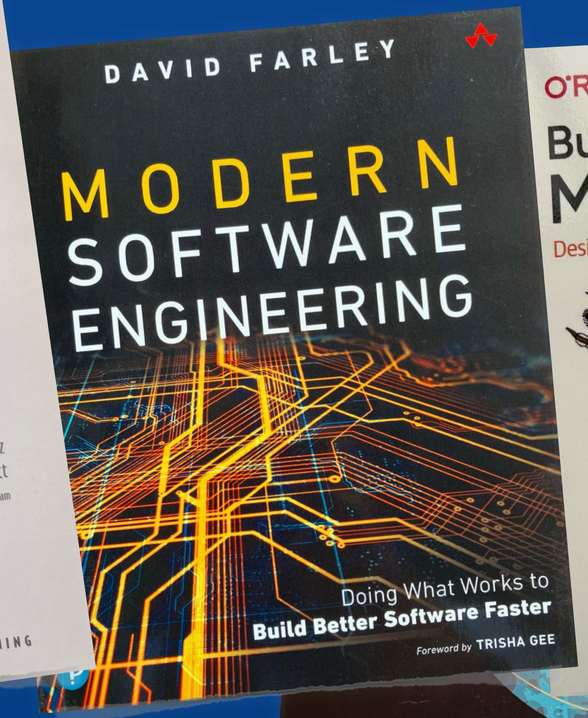
Fazit: API-Tests, Contract-Tests etc.

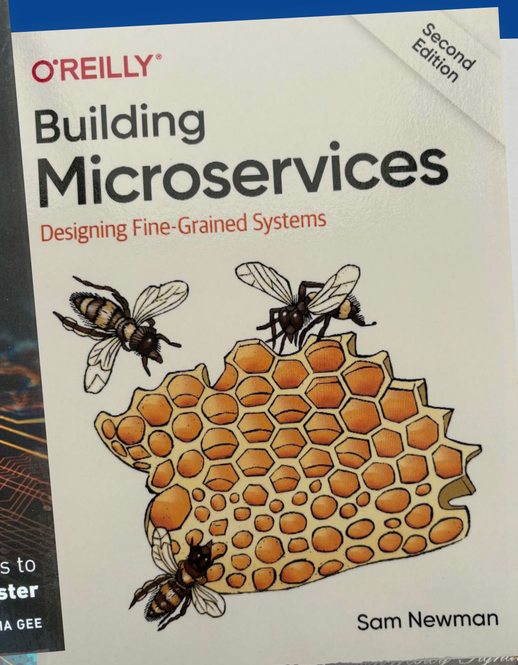
Unterschiedliche Sichtweisen:
Wer möchte was sicherstellen?

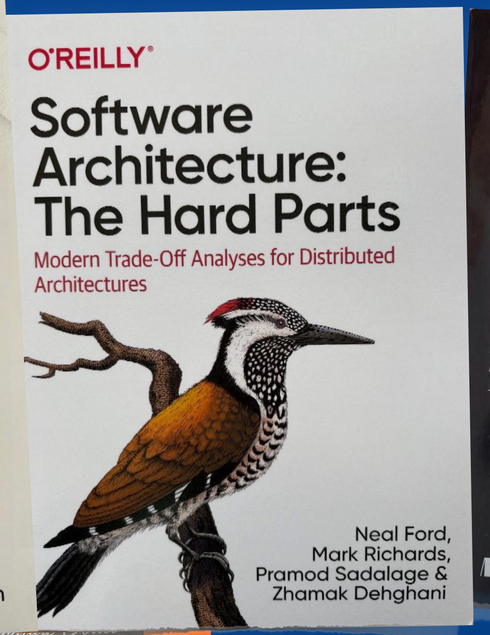
(Consumer-driven) Contract-Testing:
Sicherheitsnetz für Integrationskompatibilität.
Weniger E2E-Tests nötig, aber kein kompletter Ersatz!

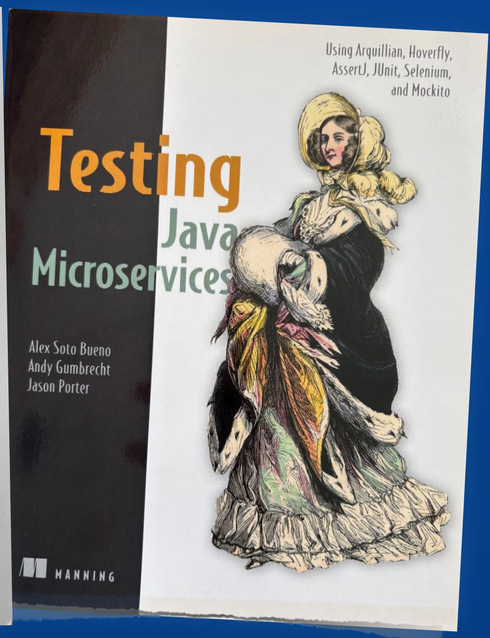
Wie gut sind eure APIs **testbar**?
Wie sind die Systeme ge-/entkoppelt?
HTTP/REST oder Events oder … ?
Das sind **Architekturentscheidungen**!

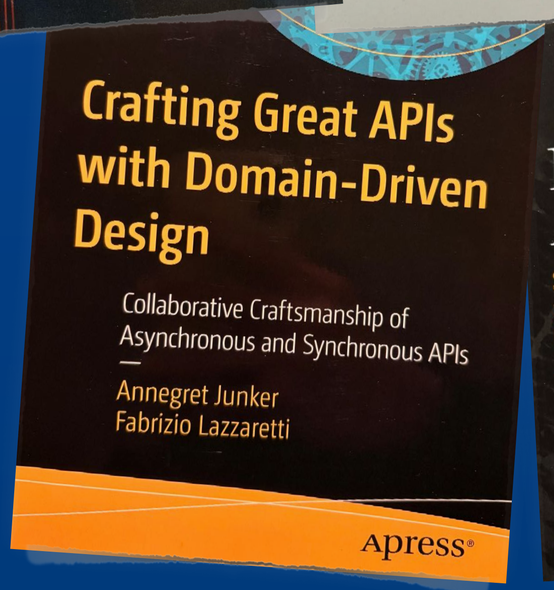

Contract Testing
IN ACTION
With Pact, PactFlow, and GitHub Actions
Marie Cruz
Lewis Prescott
Foreword by Mark Winteringham
MANNING

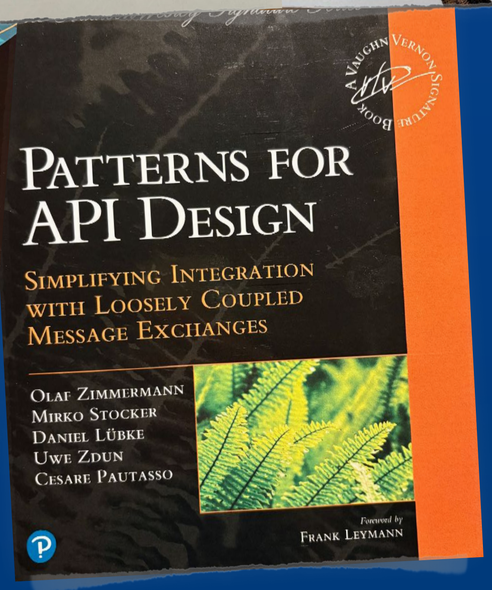

DAVID FARLEY
MODERN
SOFTWARE
ENGINEERING
Doing What Works to
Build Better Software Faster
Foreword by TRISHA GEE


O'REILLY
Second Edition
Building
Microservices
Designing Fine-Grained Systems
Sam Newman


O'REILLY
Software
Architecture:
The Hard Parts
Modern Trade-Off Analyses for Distributed
Architectures
Neal Ford,
Mark Richards,
Pramod Sadalage &
Zhamak Dehghani


Using Arquillian, Hoverfly,
AssertJ, JUnit, Selenium,
and Mockito
Testing
Java
Microservices
Alex Soto Bueno
Andy Gumbrecht
Jason Porter
MANNING


Crafting Great APIs
with Domain-Driven
Design
Collaborative Craftsmanship of
Asynchronous and Synchronous APIs
—
Annegret Junker
Fabrizio Lazzaretti
Apress®


A VAUGHN VERNON SIGNATURE BOOK
PATTERNS FOR
API DESIGN
SIMPLIFYING INTEGRATION
WITH LOOSELY COUPLED
MESSAGE EXCHANGES
OLAF ZIMMERMANN
MIRKO STOCKER
DANIEL LÜBKE
UWE ZDUN
CESARE PAUTASSO
Foreword by
FRANK LEYMANN

Die Techniker

www.tk.de/IT

›DEV LAND‹

DANKE!

[ˈtoːmas] 👋

@thmuch

innoq.com/de/articles/2016/09/consumer-driven-contracts/

microsoft.github.io/code-with-engineering-playbook/automated-testing/cdc-testing/

martinfowler.com/articles/consumerDrivenContracts.html

pact.io

spring.io/projects/spring-cloud-contract