4th UCAAT User Conference on Advanced Automated Testing

ETSI World Class Standards

CONTRACT TESTING IN THE CLOUD IN GEHC

Presented by Andras Naszrai

© All rights reserved

SZÉCHENYI 2020

NATIONAL RESEARCH, DEVELOPMENT AND INNOVATION FUND

HUNGARIAN GOVERNMENT

INVESTING IN YOUR FUTURE

4th UCAAT User Conference on Advanced Automated Testing

ETSI World Class Standards

ABOUT CONTRACTS

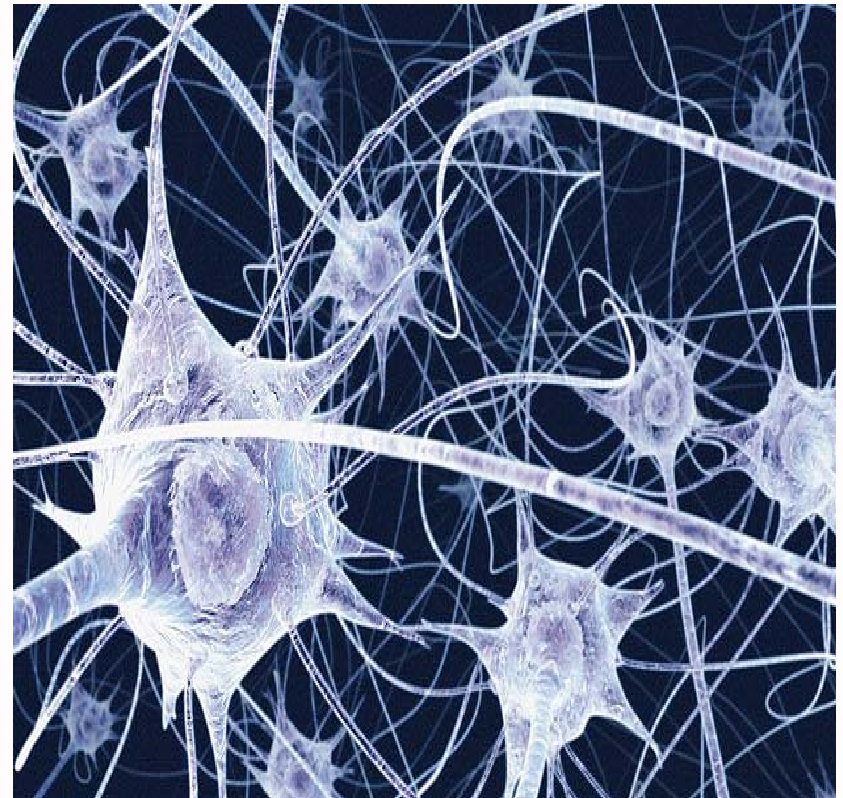
What is a contract? Why do we test it?

© All rights reserved

SZÉCHENYI 2020

Microservice Architecture

- The system is decomposed into small components with narrow scope
- Communication between the components is done over the network
- The individual components have their own independent lifecycle and they are independent deployment units



© All rights reserved

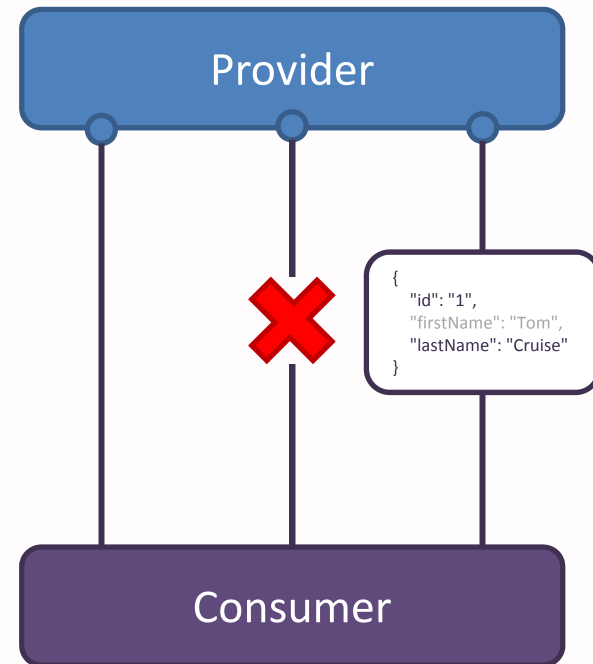
Relationships in Microservice Architecture

- Are asymmetric, they follow the server-client model
- Server is called provider, as it provides a service and the client is called consumer for similar reasons
- Usually they use HTTP (RESTful)
- Or some kind of messaging protocol, like AMQP

© All rights reserved

Contract

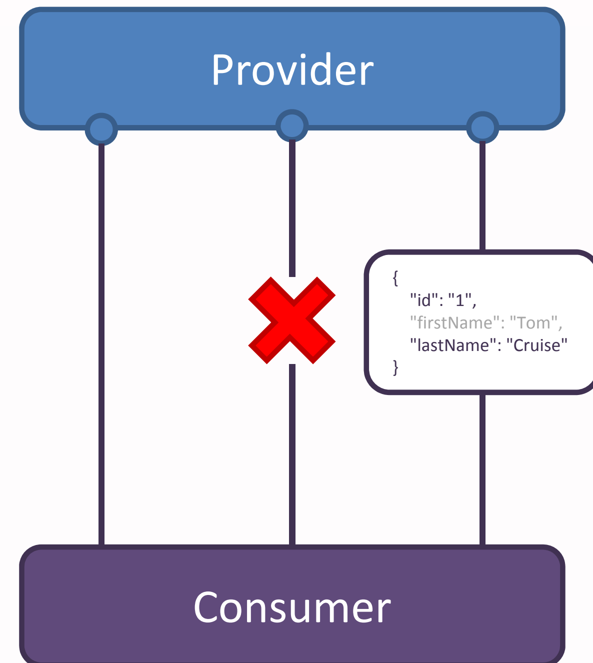
- A contract is formed when a component becomes the consumer of another
- It describes how the service is consumed by the consumer
- It contains the consumer's expectations about
  - Input-output data structures
  - Performance
  - Concurrency characteristics



© All rights reserved

Contract Testing

- Contract testing aims to ensure that a consumer's dependency on a provider is continuously fulfilled
- Therefore
  - It is done on the provider
  - from the consumer's point of view
  - Focus is on the behaviour on the interface (not on the domain logic)



© All rights reserved

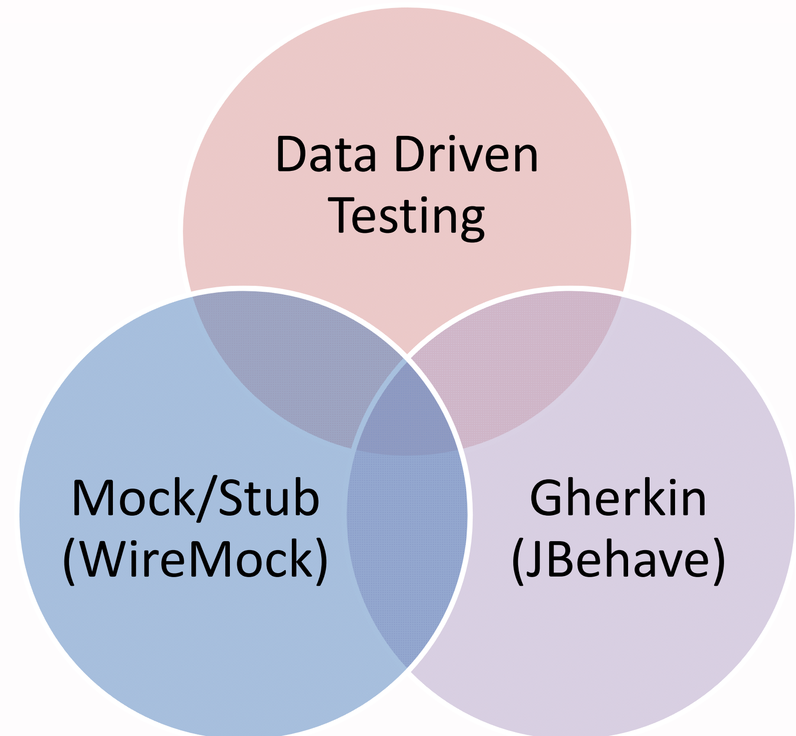
CONCEPTS USED

The building blocks

© All rights reserved

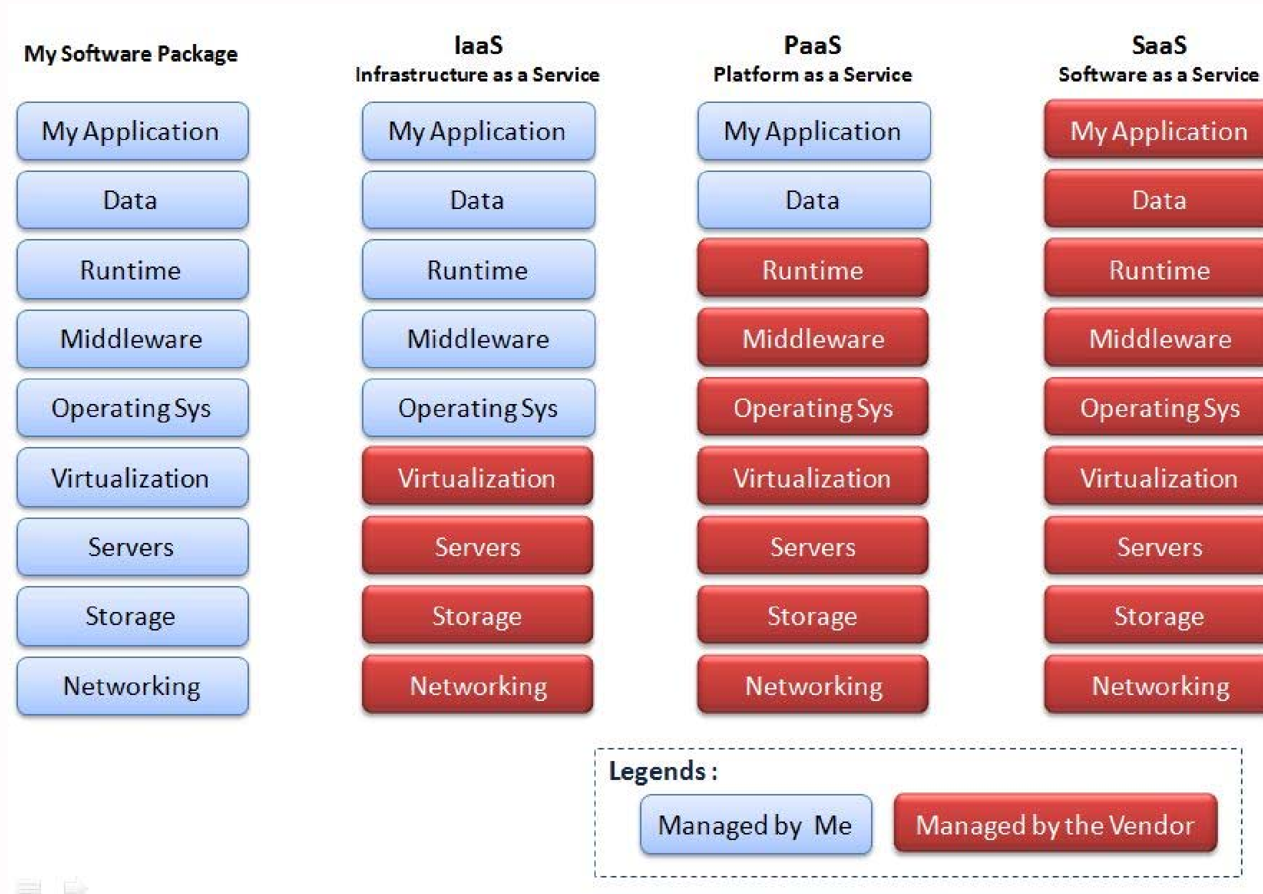
Concepts used

- Data Driven Testing
  - only the data is needed from the user, everything else is there
- Gherkin (JBehave)
  - Self-documenting
  - Automatic translation of data into test
- Mock/Stub (WireMock)
  - Helps the isolation of the tested microservice



© All rights reserved

Cloud services



© All rights reserved

Testing as a Software as a Service

- MyApplication
  - Is a data driven test tool
  - Sitting in the cloud
  - Listening on HTTP for test execution requests

© All rights reserved

WHAT WE DID

Contract Test Executor Service

© All rights reserved

SZÉCHENYI 2020

We defined our MVP

- Support HTTP contracts
- 100% data driven
- Software as a Service
- Support Oauth 2.0
- Automatic mocking for HTTP based dependencies
- Load binary/JSON data from file

© All rights reserved

First we needed a data format

1. Credentials
2. DataRecords
   - Operation
   - HTTP method
   - Parameters
   - Response

```yaml
---
credentials:
  clientId: some-client-id
  clientSecret: the-client-secret
  username: some.user@some.domain
  password: some password
dataRecords:
- operation: https://some.domain/another/path
  httpMethod: POST
  parameters:
  - in: body
    name: content
    type: octet-stream
    reference: /temp/some/file.bin
  - in: header
    name: content-type
    type: string
    value: application/octet-stream
  response:
    code: 200
    type: string
    reference: Successful operation!
    headers:
    - name: content-type
      value: text/plain
```


© All rights reserved

Then we needed a way to convert data to test

```
Scenario: 0b865200-f87b-4329-9a87-00abd8b44390
Given the following HTTP request
| httpMethod | httpAddress                       |
| POST       | https://some.domain/another/path |
And the following parameters for the request
| in     | name         | type         | value                    | reference          |
| body   | content      | octet-stream | null                     | /temp/some/file.bin |
| header | content-type | string       | application/octet-stream | null               |
And the following credentials
| client_id      | client_secret     | username              | password      |
| some-client-id | the-client-secret | some.user@some.domain | some_password |
When I send the request
Then I receive a response with status code 200
And the response body is
| type | value | reference                  |
| json | null  | /temp/some/json-file.json |
And the response headers are
| name         | value            |
| content-type | application/json |
```


© All rights reserved

And a way to convert data to mock


```
{
  "request": {
    "method": "POST",
    "url": "/another/path",
    "headers": {
      "Content-Type": "application/octet-stream"
    }
  },
  "response": {
    "status": 200,
    "body": "The content of /temp/some/json-file.json",
    "headers": {
      "Content-Type": "application/json"
    }
  }
}
```


© All rights reserved

The rest was just implementation detail

We used 3rd party libs for

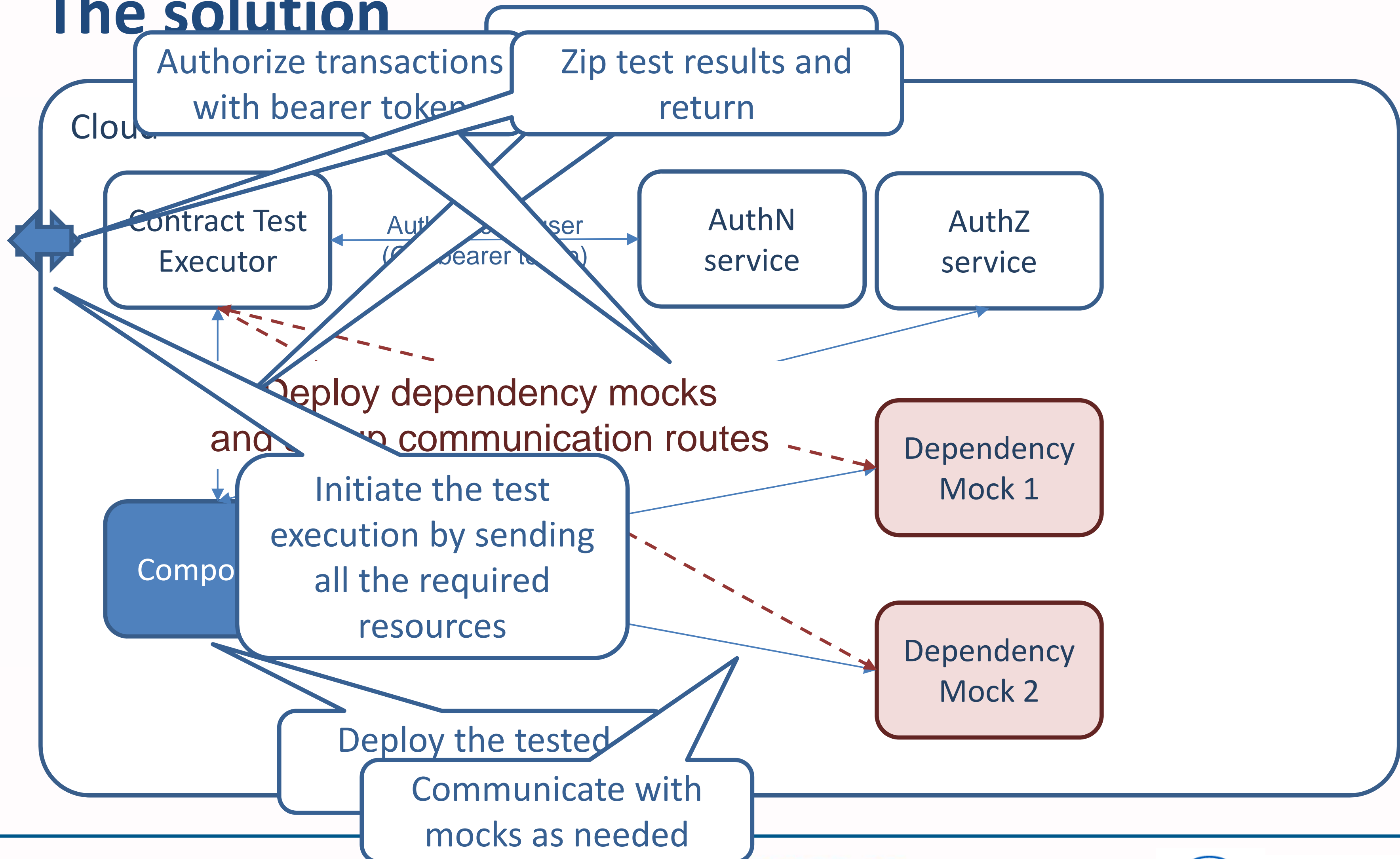
- Oauth 2.0 support
- Binary data conversion
- JSON data conversion

Relied on

- Spring
- jackson

© All rights reserved

The solution



© All rights reserved

Highlights, achievements

- Uses the same data structure for test input and mock data
- Isolates the tested component automatically
- It is self contained
- Hides the implementation details of the test environment completely (you don't have to write code to do automated testing)
- It is really a service in the cloud, that anyone can access

Thank You for your attention!

© All rights reserved