SPRING IO

#SPRING1018

[Spring Cloud Contract]

Testing every level of your spring microservices application

Jeroen Sterken

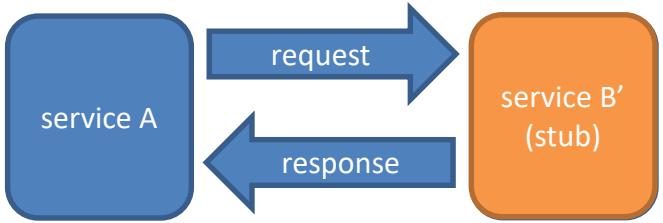
@jeroensterken

Agenda

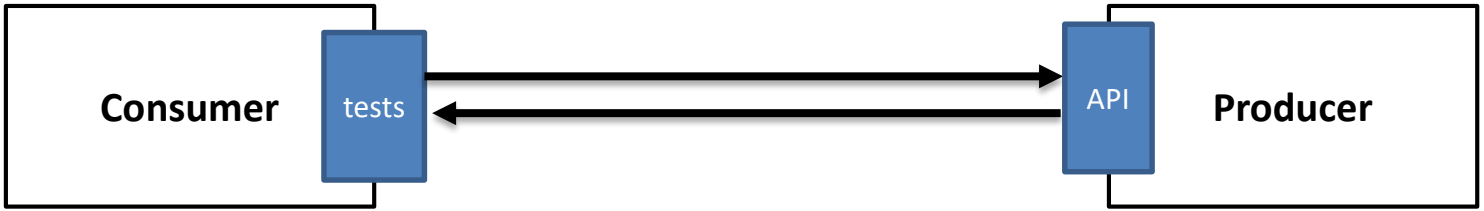
- Introduction
- Consumer driven contract
- Spring Cloud Contract
- Exercise

Introduction

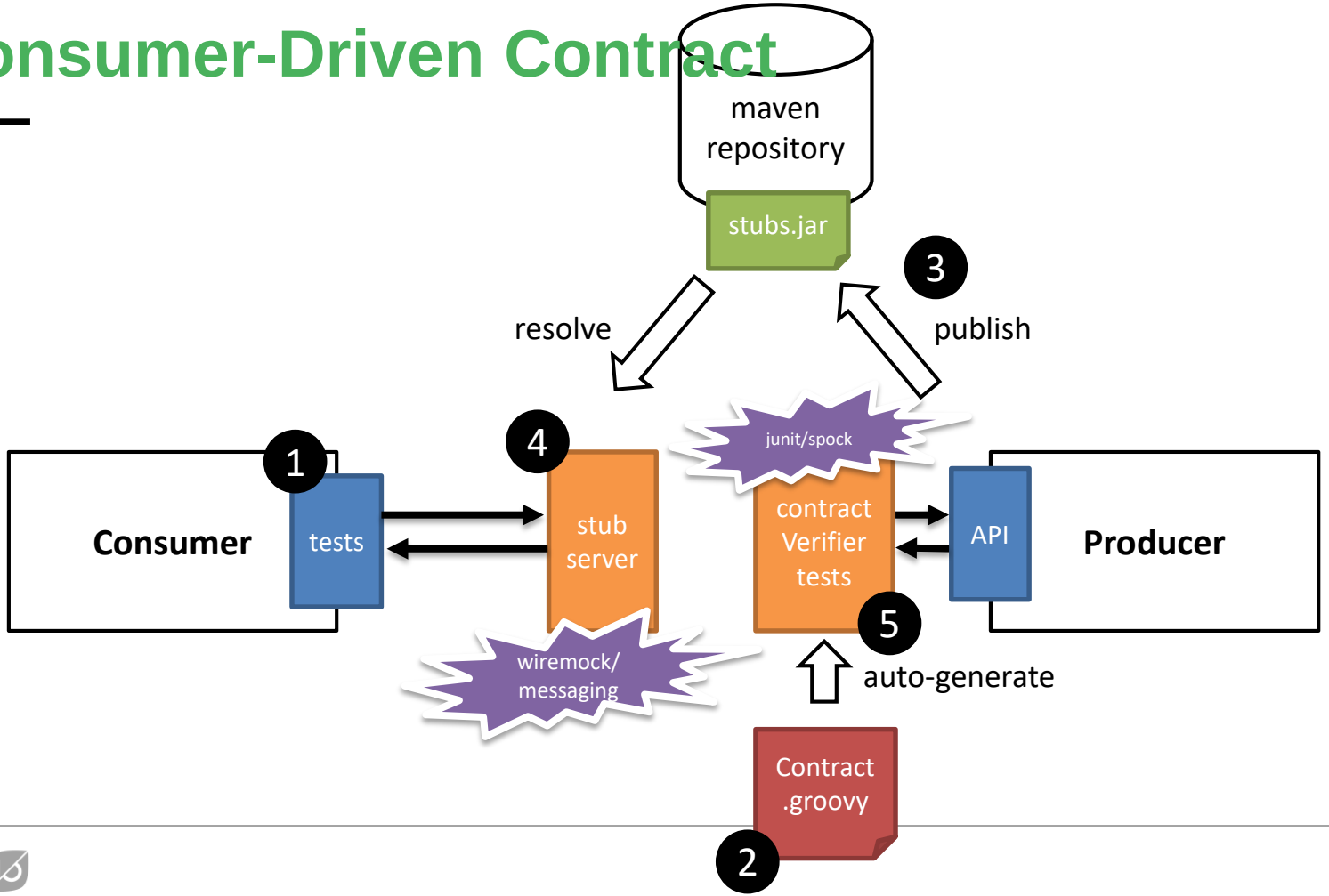
- Testing chained microservices is hard
  - Just want to test the API
- Mocking/service virtualization
  - Tools: Hoverfly or WireMock
  - How to guarantee that service B's stub tracks changes of the actual service?
- Contracts
  - Agreement between 2 services
  - Service A (consumer) creates a contract that service B (producer) will have to abide by
  - = Consumer-Driven Contract



Consumer-Driven Contract



Consumer-Driven Contract



Spring Cloud Contract

- Purpose
  - Ensure WireMock/Messaging stubs (client) do exactly what the actual server-side implementation does
  - Promote ATDD method and Microservices architectural style
  - Published changes in contracts are immediately visible on both sides
  - Generate boilerplate test code to be used on the server side

- NOT: writing business features in the contracts

How it works?

Step 1: TDD approach

- Consumer writes test(s) first

How it works?

Step 2: defining the contract

- Consumer writes expectations as a contract
- 2 ways
  - Groovy DSL
  - YAML

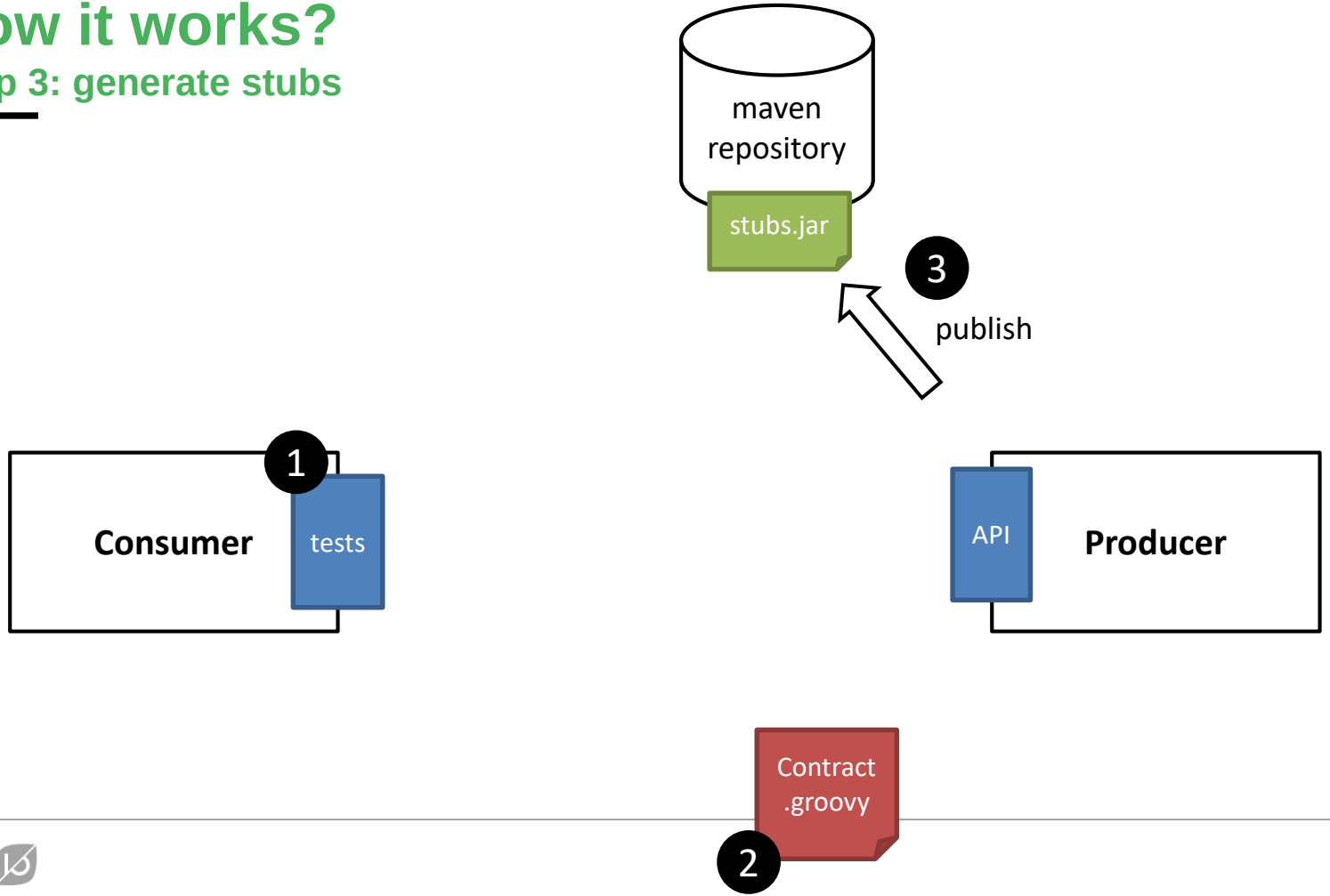
```
package contracts
org.springframework.cloud.contract.spec.Contract.make {
     request {
            method GET()
            url "/tapas"
     }
     response {
            status 200
            headers {
           contentType applicationJson()
            }

            body ([
            name: 'Banderillas',
                  price: 3
            ])
```

```
request:
  method: GET
  url: /tapas
response:
 headers:
    Content-Type: application/json;charset=UTF-8
  status: 200
  body:
    name: "Banderillas"
    price: 3
```

How it works?

Step 3: generate stubs



How it works?

Step 3: generate stubs

- Add the Spring Cloud Contract Verifier dependency + plugin
  - Produce and install stubs
  - Generates and run tests (producer)

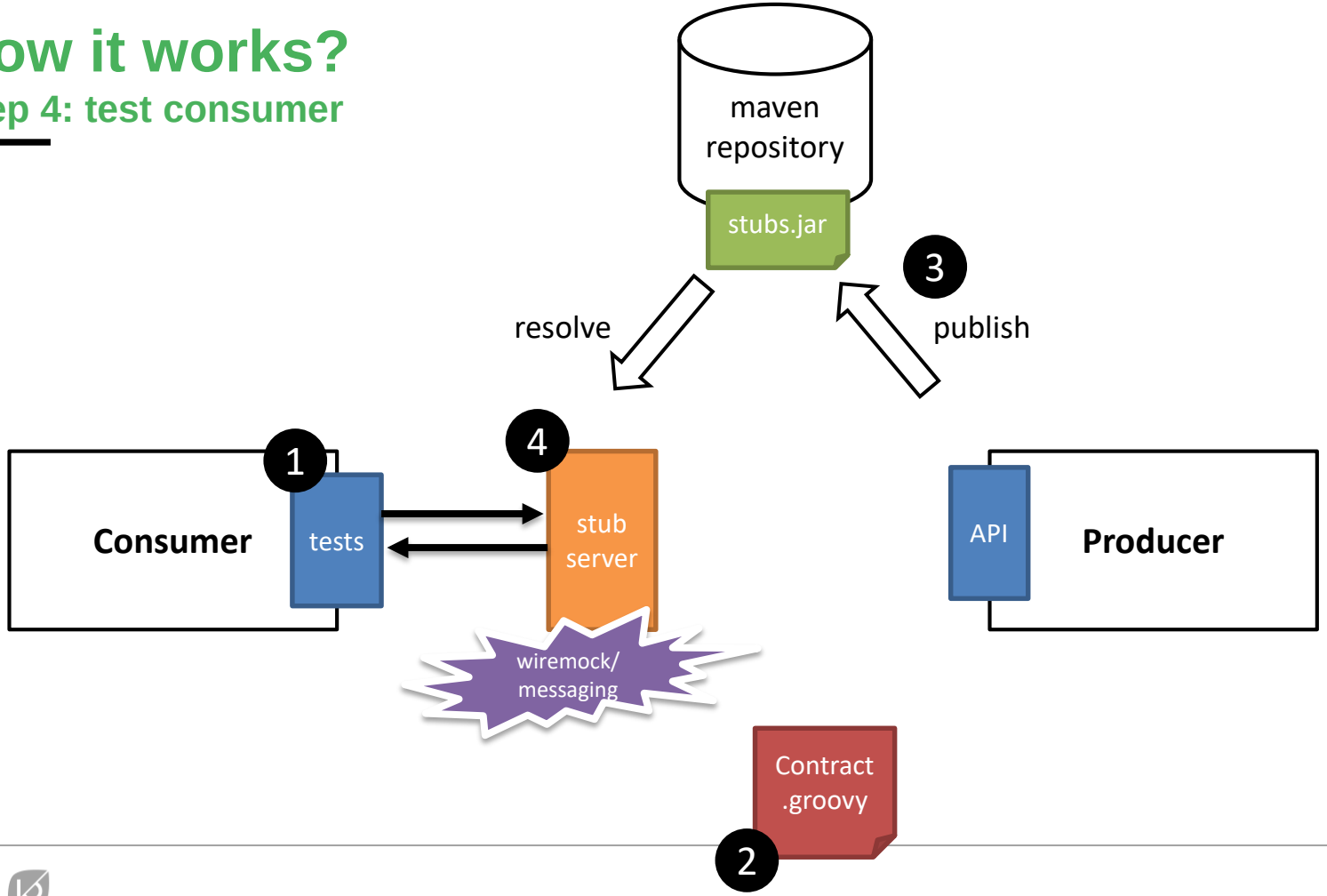
```
<dependency>
<groupId>org.springframework.cloud</groupId>
<artifactId>spring-cloud-starter-contract-verifier</artifactId>
<scope>test</scope>
</dependency>
```

```
<plugin>
      <groupId>org.springframework.cloud</groupId>
      <artifactId>spring-cloud-contract-maven-plugin</artifactId>
      <version>${spring-cloud-contract.version}</version>
      <extensions>true</extensions>
</plugin>
```

- $mvn clean install -DskipTests
  - Stub artifact built + installed in local mvn repo

How it works?

Step 4: test consumer



How it works?

Step 4: test consumer

- Add ‘Spring Cloud Contract Stub Runner’ dependency

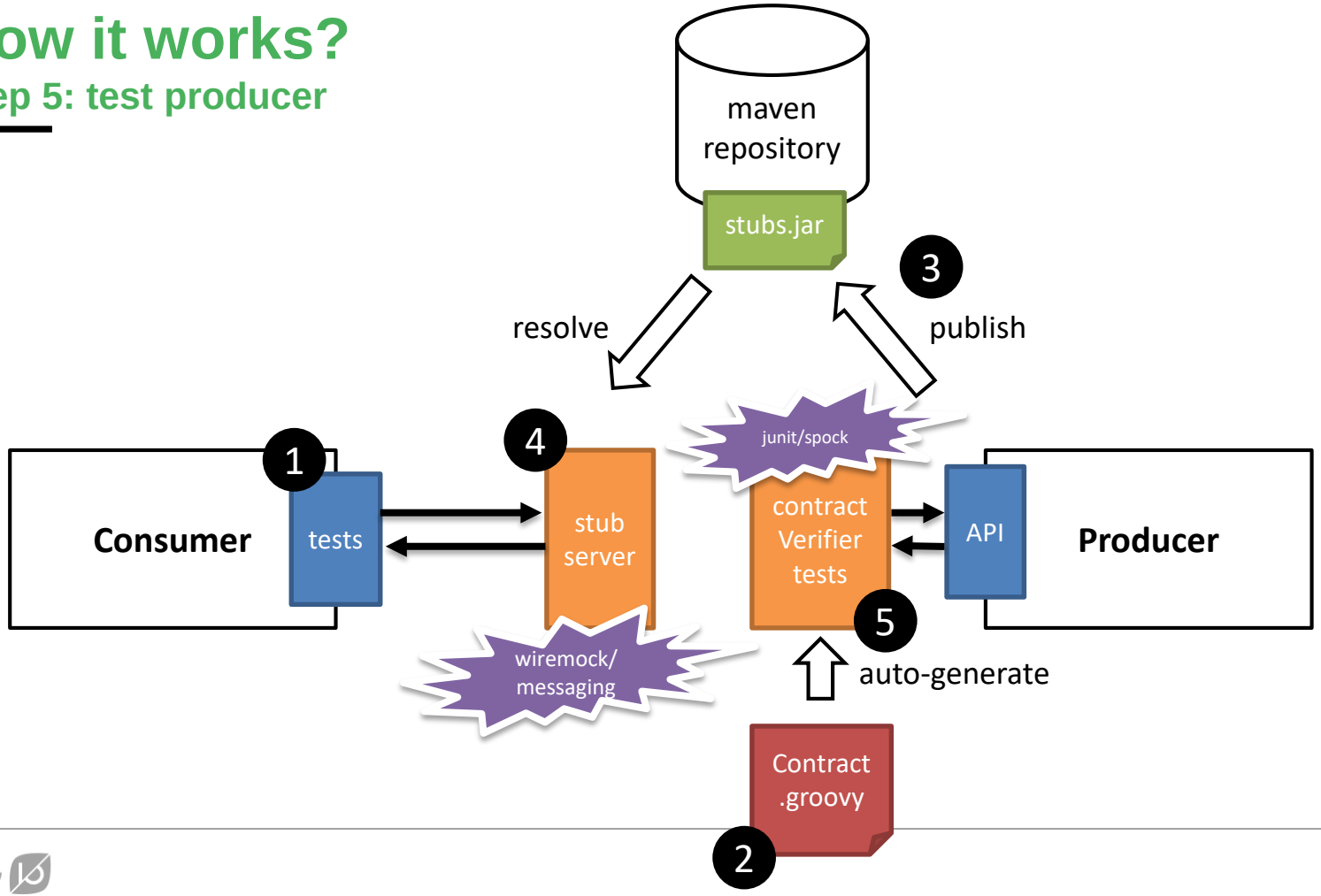
```
<dependency>
   <groupId>org.springframework.cloud</groupId>
   <artifactId>spring-cloud-starter-contract-stub-runner</artifactId>
   <scope>test</scope>
</dependency>
```

- Get Producer-side stubs
  - Annotate test class

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment=WebEnvironment.NONE)
@AutoConfigureStubRunner(ids = {"com.example:http-server-dsl:+:stubs:6565"},
stubsMode = StubRunnerProperties.StubsMode.LOCAL)
@DirtiesContext
public class LoanApplicationServiceTests {
```

How it works?

Step 5: test producer



How it works?

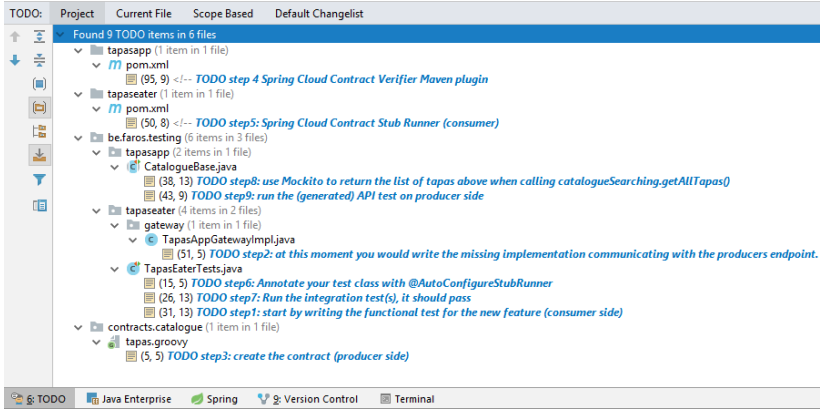
Step 5: test producer

- Spring Cloud Contract Verifier dependency + plugin
  - Produce and install stubs
  - Generates and run tests (producer)
- Create base test class
  - Extended by all the auto-generated tests, and contains all the setup necessary to run them
- $mvn clean install
  - Generates test + test passes

EXERCISE

Exercise

spring-cloud-contract-exercise.pdf



FAROS

CONSUMER

Step1: write consumer test

- We use a TDD approach on the consumer side (tapaseater service), writing the functional test for the new feature (retrieving the list of all available tapas) before implementing it.
- Run the test, and it will obviously fail because we didn't implement the functionality yet.

```
org.springframework.web.client.ResourceAccessException: I/O error on GET
request for "http://localhost:8080/tapas ": Connection refused.
```

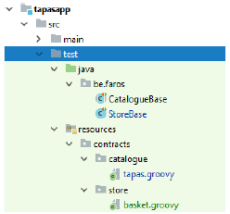
Step2: write gateway implementation

- We already wrote the gateway implementation for you, sending a REST (GET) request to /tapas endpoint using Spring's RestTemplate.

Step3.a: create a new contract

The producer (tapasapp service) owns the contract(s), so physically, all the contract are in the producer's repository. But it's the consumer who writes the contract based on his needs. These contracts defines how the consumer expects the producer to behave.

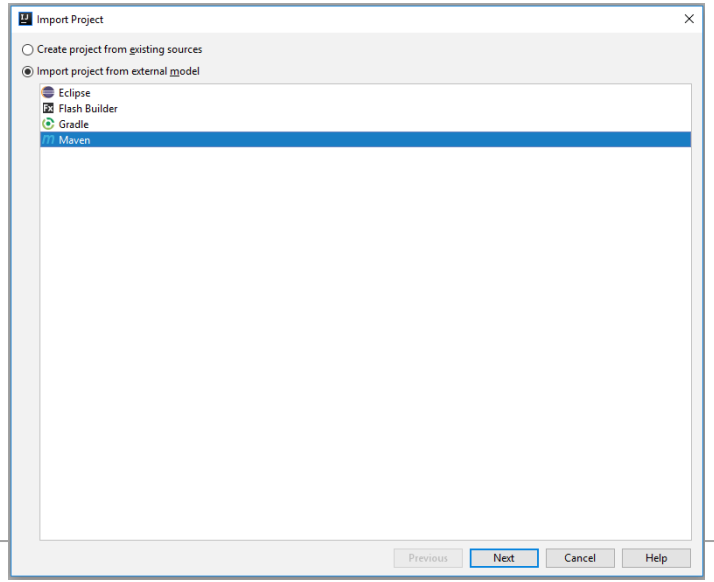
- Switch to the producer side (tapasapp service) for writing the contract for this new feature
  - In this simplified exercise both services (consumer & producer) are in the same GIT repo but in a real environment this would mean checking out the producer's GIT repository or a separate repo containing only the contracts.
  - We already created a file 'tapas.groovy' for you. In this file we'll define the contract using the Spring Cloud Contract Groovy DSL.
    - The file should be located in the src/test/resources/contracts/ folder for the spring-cloud-contract-plugin to find it (default behavior, look at tapasapp>pom.xml - 'contractsDirectory' if you need to change this)

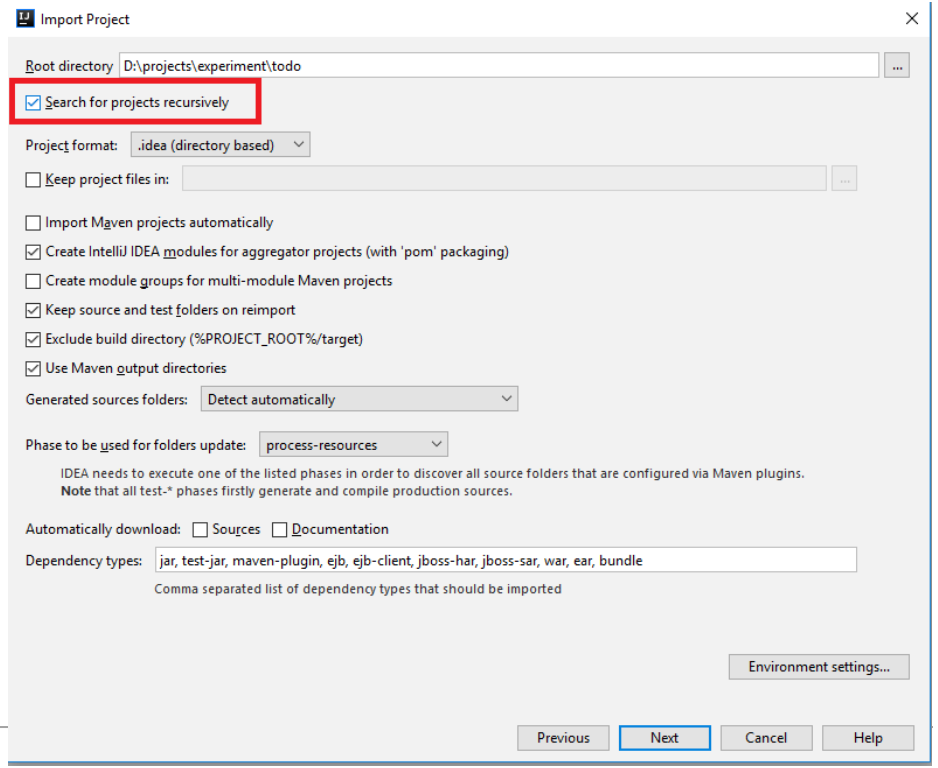


www.faros.be

Exercise

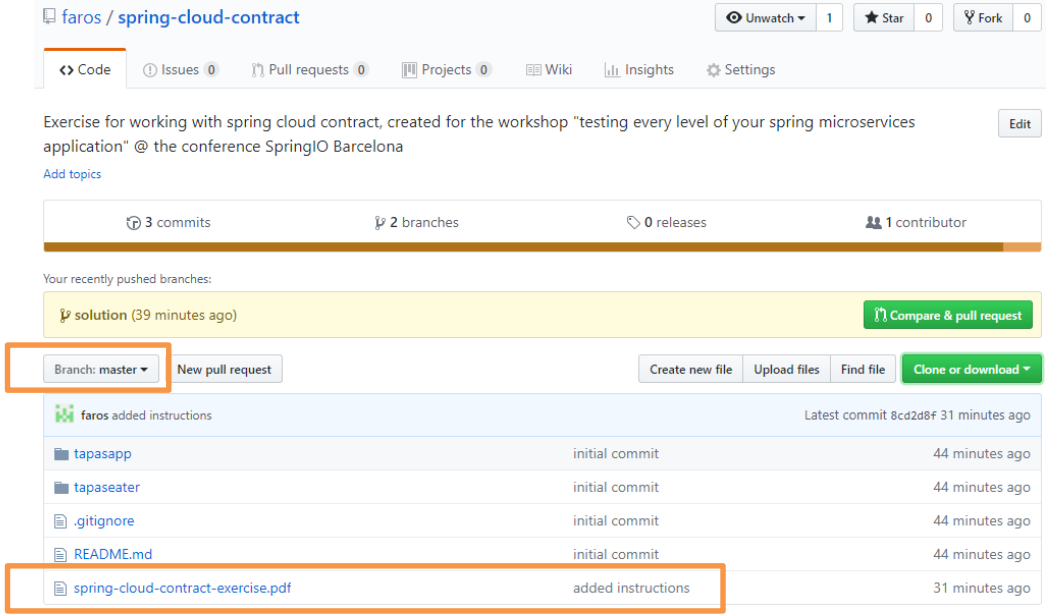
- Import project (IDE)
  - Maven project





Exercise

- https://github.com/faros/spring-cloud-contract.git



Lab instructions

SPRING IO

#SPRING1018

THANKS!
Q & A

Jeroen Sterken
@jeroensterken