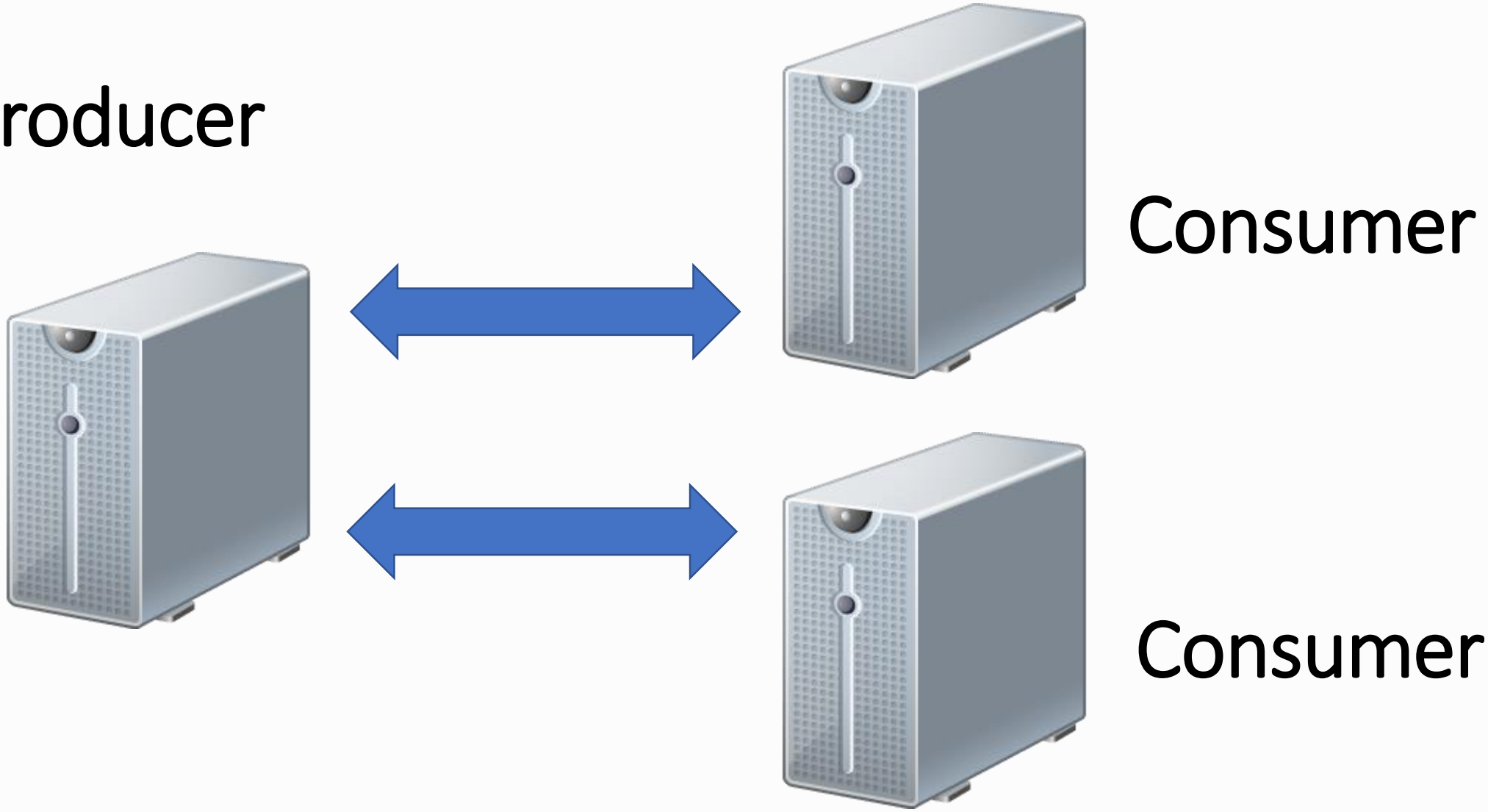
Spring Cloud Contract

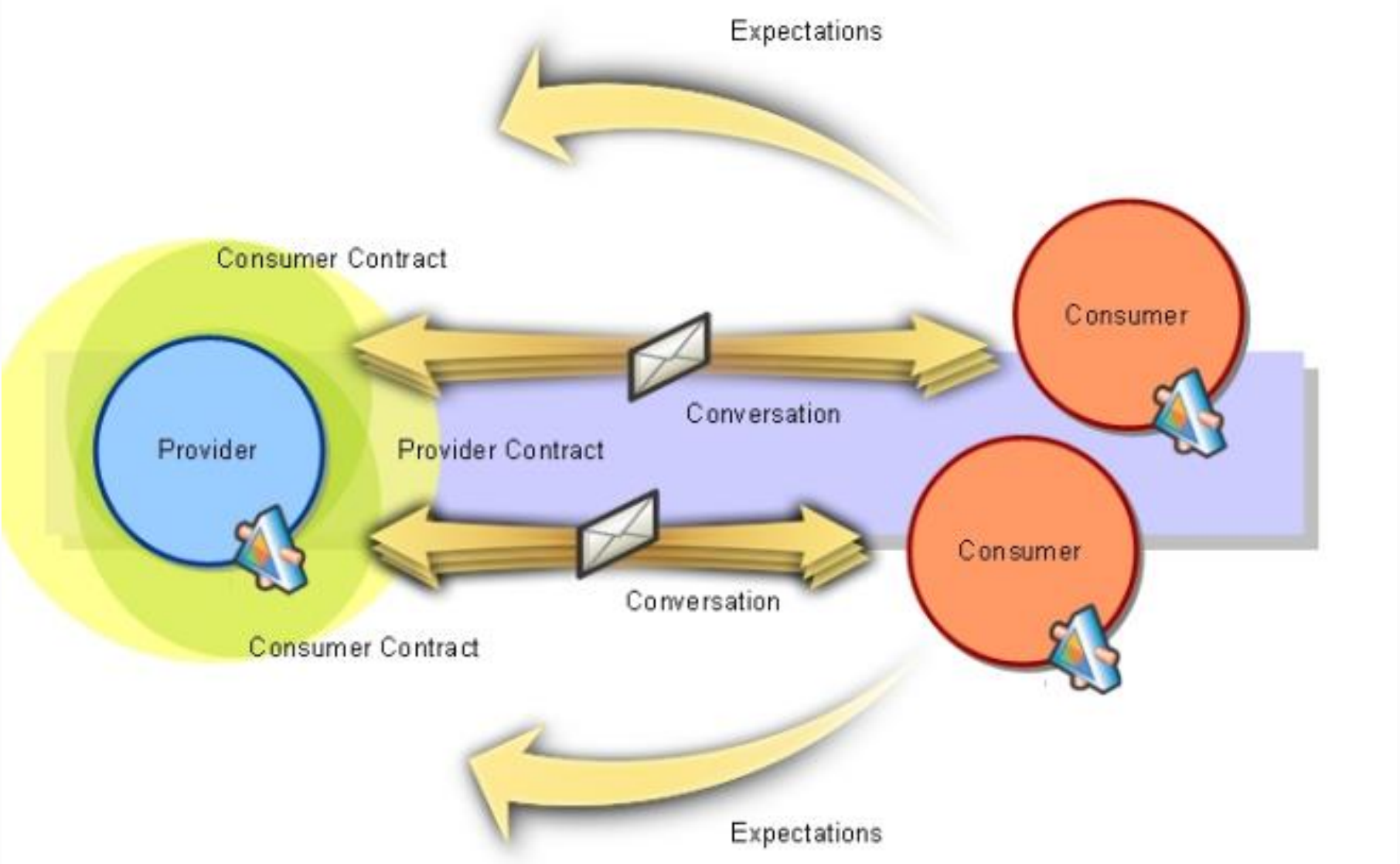
KRZYSZTOF ŁOPUCKI

2018

Consumer-Driven Contracts



Consumer-Driven Contract



TRANSITION TECHNOLOGIES MANAGED SERVICES

O co chodzi???

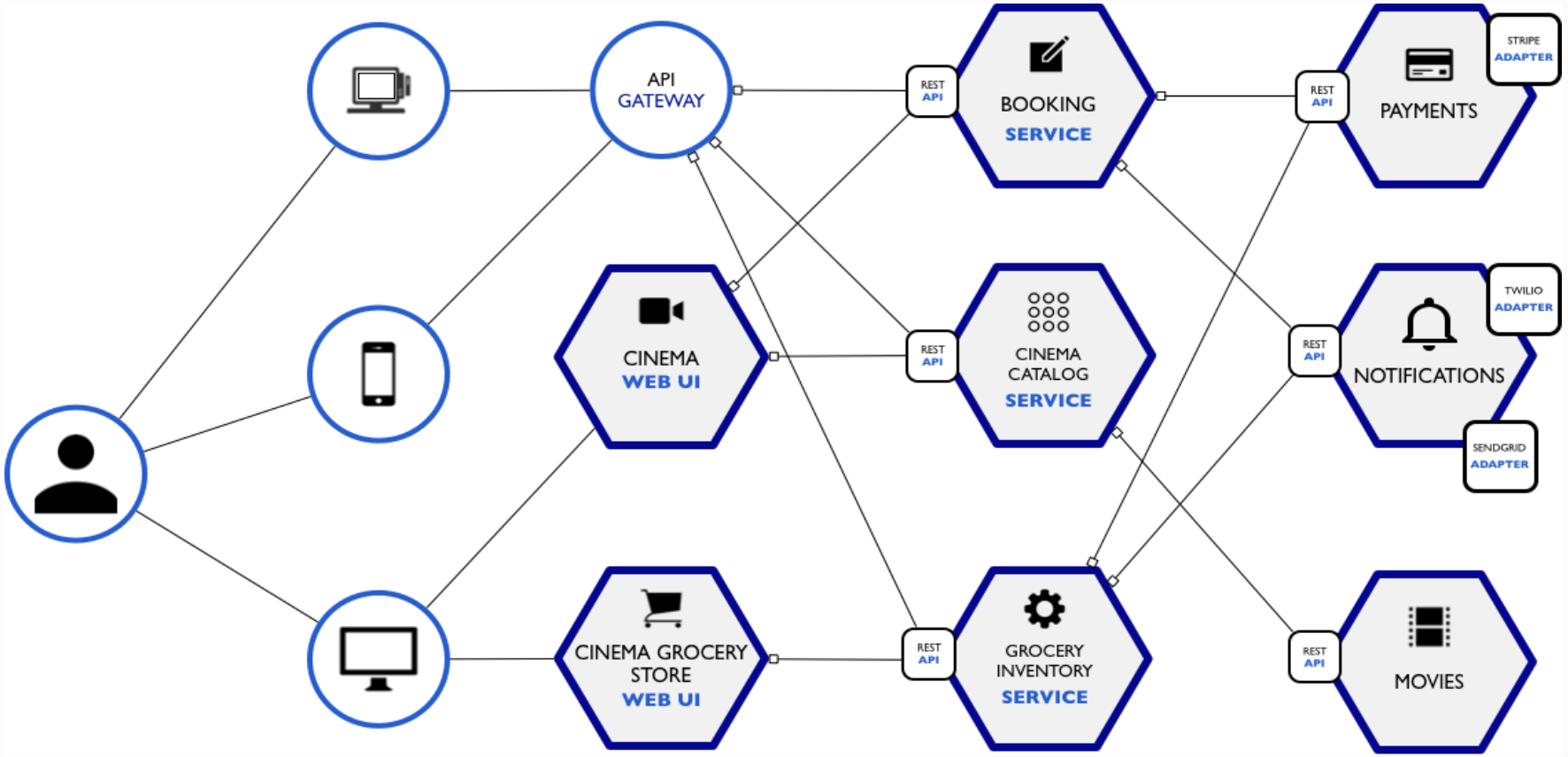
Skomplikowane???

Dlaczego???

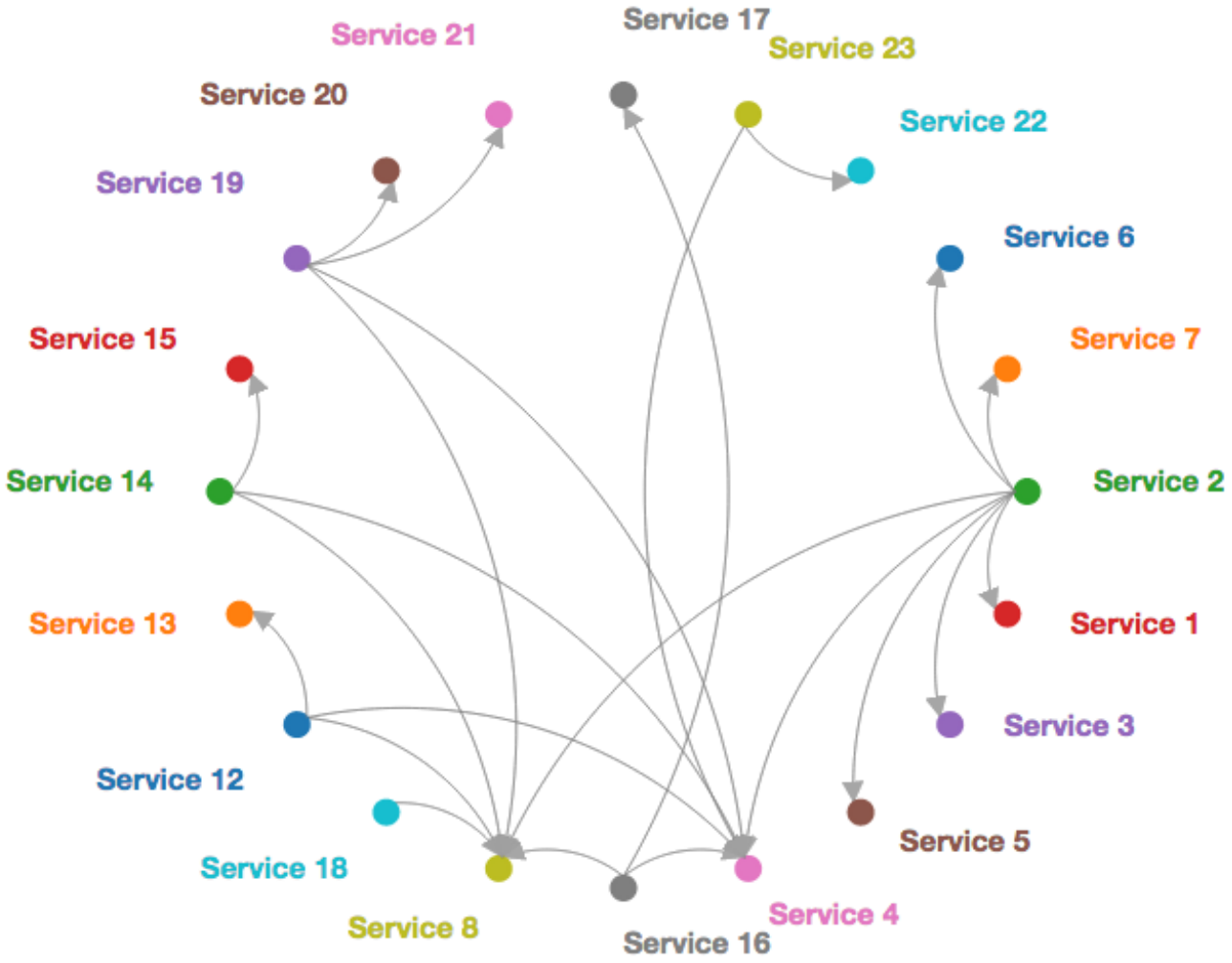
Jak???

Po co???

Mikroserwisy



Consumer-Driven Contract



TRANSITION TECHNOLOGIES MANAGED SERVICES

To działa?

- Deploy wszystkich mikroserwisów i uruchomienie testów end-to-end.
- Mock powiązanych mikroserwisów i uruchomienie testów jednostkowych/integracyjnych.

Deploy wszystkich mikroserwisów i uruchomienie testów end-to-end:

Zalety:

- Dokładne odzwierciedlenie tego co dzieje się na produkcji.
- Rzeczywista komunikacja między serwisami.

Wady:

- Postawienie całego środowiska dla testu jednego z mikroserwisów.
- Jedna osoba może wykonać testy w tej samej chwili.
- Długo uruchamiane.
- Późna odpowiedź o powodzeniu lub niepowodzeniu.
- Trudne do debugowania.

Mock powiązanych mikroserwisów i uruchomienie testów jednostkowych/integracyjnych :

Zalety:

- Szybko dostajemy odpowiedź o rezultacie testów.
- Brak wymagań dla infrastruktury.

Wady:

- Właściwie nic nie testujemy.
- Testy zielone a produkcja?


SOLUTION


PACT
FOUNDATION

Consumer-Driven Contract

Spring Cloud Contract

- ✓ Implementuje Customer-Design Contract.
- ✓ Dostarcza inne opcje testowania.
- ✓ Publikuje odpowiednie serwisy dla celów testowych.
- ✓ Uruchamia testy wykonując asercje.
- ✓ Zapewnia nas, że serwer zwraca dokładnie to co powinien zwracać.
- ✓ Wspiera pisanie testów akceptacyjnych przy architekturze mikrokontrolerów.
- ✓ Generuje Testy.

Zależności

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


<dependencyManagement>
        <dependencies>
                <dependency>
                        <groupId>org.springframework.cloud</groupId>
                        <artifactId>spring-cloud-dependencies</artifactId>
                        <version>${spring-cloud.version}</version>
                        <type>pom</type>
                        <scope>import</scope>
                </dependency>
        </dependencies>
</dependencyManagement>
```

Zależności

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

Kontrakt

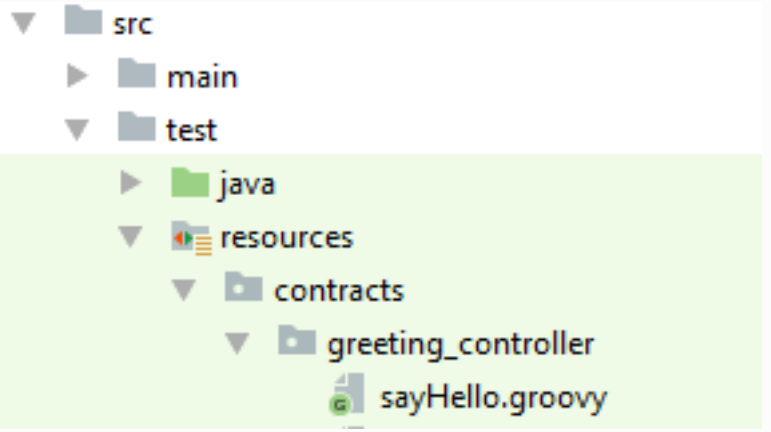
```
package contracts.greeting_controller

import org.springframework.cloud.contract.spec.Contract

Contract.make {

    request {
        method 'GET'
        url("/greeting")
    }

    response {
        status 200
        body("HELLO")
        headers {
            header('Content-Type': value(
                    producer(regex('text/plain;charset=ISO-8859-1')),
                    consumer('text/plain;charset=ISO-8859-1')
            ))
        }
    }
}
```



Implementacja

```
@RequestMapping(method = GET)
public String sayHello() {
    return "HELLO";
}
```

Testy

```
@Mock
LibraryService libraryService;
@InjectMocks
LibraryController libraryController;

@Before
public void setup() {
    Book bookToSave = mockGFBook();

    BDDMockito.given(libraryService.findBooks())
                            .willReturn(ImmutableList.of(mockBook()));
    BDDMockito.given(libraryService.save(bookToSave)).willReturn(bookToSave);

    RestAssuredMockMvc.standaloneSetup(GreetingController.class, libraryController);
}
```

Testy

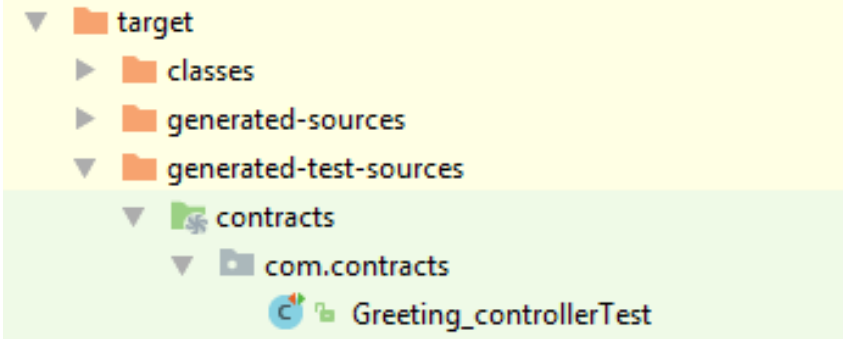
```
mvn install
```

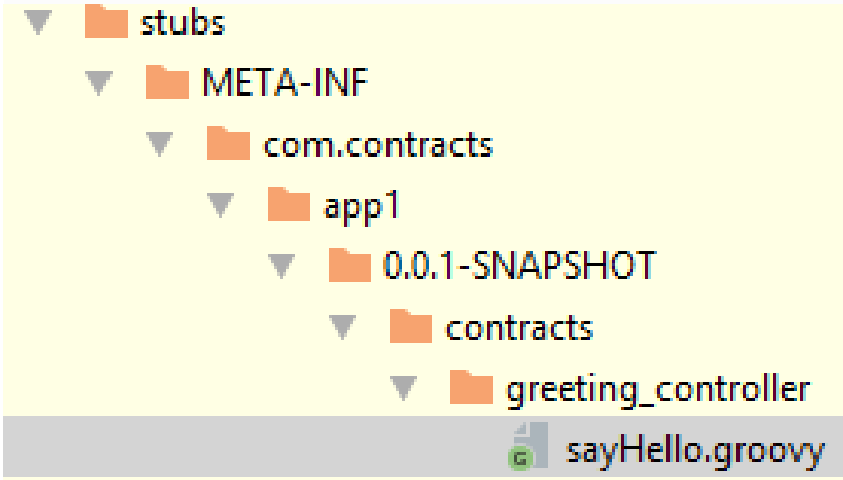
Rezultat

```
@Test
public void validate_sayHello() throws Exception {
    // given:
        MockMvcRequestSpecification request = given();

    // when:
        ResponseOptions response = given().spec(request)
                .get("/greeting");

    // then:
        assertThat(response.statusCode()).isEqualTo(200);
        assertThat(response.header("Content-Type"))
                .matches("text/plain;charset=ISO-8859-1");
    // and:
        String responseBody = response.getBody().asString();
        assertThat(responseBody).isEqualTo("HELLO");
}
```





TT MS TRANSITION TECHNOLOGIES MANAGED SERVICES

Zależności

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

<dependencyManagement>
    <dependencies>
        <dependency>
            <groupId>org.springframework.cloud</groupId>
            <artifactId>spring-cloud-dependencies</artifactId>
            <version>${spring-cloud.version}</version>
            <type>pom</type>
            <scope>import</scope>
        </dependency>
    </dependencies>
</dependencyManagement>
```

Implementacja

```
final String uri = "http://localhost:8080/library";

RestTemplate restTemplate = new RestTemplate();
HttpEntity<String> response =
            restTemplate.exchange(uri, HttpMethod.GET, null, String.class);

String resultString = response.getBody();

return objectMapper.readValue(resultString, Book.class);
```

Testy

```
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureStubRunner(
        ids = "com.contracts:producer:+:stubs:8080", workOffline = true
)
```

```
Book book = remoteRepository.getAllBook();

assertThat(book).isNotNull();
assertThat(book.getAuthor()).isEqualTo("J. K. Rowling");
assertThat(book.getTitle()).isEqualTo("Harry Potter");
assertThat(book.getISBN()).isEqualTo("XXX");
```

Rezultat

Tests run: 1, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 9.374 s - in

com.contracts.App2ApplicationTests

Spring Cloud Contract - Podsumowanie

- Implementuje Consumer-Driven Contract.
- Łatwy w użyciu.
- Daje możliwość skopiowania JSON’a z danymi i delikatną zmianę jego elementów.
- Generuje testy.
- Pobiera potrzebne zależności i uruchamia odpowiednie serwisy jako STUB’y.
- Integruje się ze Spring Cloud – nie potrzeba serwisów discovery.

Spring Cloud Contract – Zdefuniujmy kilka kontraktów

```
Contract.make {
    request {
        method 'POST'
        url '/check'
        body(
                age: $(oldEnough()),
                name: $(anyAlphaUnicode())
        )
        headers {
            contentType(applicationJson())
        }
    }
    response {
        status 200
        body("""
         {
            "status": "${value(ok())}",
            "surname": "${fromRequest().body('$.name')}"
         }
         """)
        headers {
            contentType(applicationJson())
        }
    }
}
```

TRANSITION TECHNOLOGIES MANAGED SERVICES

Spring Cloud Contract – Zdefuniujmy kilka kontraktów

```
Contract.make {
    request {
        method 'POST'
        url '/check'
        body(
                age: $(ConsumerUtils.oldEnough())
        )
        headers {
            contentType(applicationJson())
        }
    }
    response {
        status 200
        body(file('response.json'))
        headers {
            contentType(applicationJson())
        }
    }
}
```

Spring Cloud Contract – Zdefuniujmy kilka kontraktów

```
Contract.make {
    description("""
Represents a grumpy waiter that will sell alcohol only to Starbuxman.
""")
    request {
        method POST()
        url '/buy'
        body(
                name: "starbuxman",
                age: 25
        )
        stubMatchers {
            jsonPath('$.age', byRegex('[2-9][0-9]'))
        }
        headers {
            contentType(applicationJson())
        }
    }
    response {
        status 200
        body(
                message: "There you go Josh!",
                status: "OK"
        )
        headers {
            contentType(applicationJson())
        }
        async()
    }
    priority 10
}
```

Spring Cloud Contract – Zdefuniujmy kilka kontraktów

```
request:
  method: POST
  url: /stats
  body:
    name: foo
  headers:
    Content-Type: application/json
  matchers:
    body:
      - path: "$.name"
        type: by_regex
        value: "[\\p{L}]*"
response:
  status: 200
  body:
    text: Dear {{{jsonpath this '$.name'}}} thanks for your interested in drinking beer
    quantity: 5
  headers:
    Content-Type: application/json;charset=UTF-8
  matchers:
    body:
      - path: $.quantity
        type: by_regex
        value: -?(\d*\.\d+|\d+)
```

https://github.com/klopucki/spring-cloud-contract

Koncentrując swoją uwagę na przetestowaniu rozwiązania i wykorzystaniu zdobytej wiedzy podczas wykładu, zaimplementuj funkcjonalność w dwóch mikroserwisach :

1. Mikroserwis odpowiedzialny za konta bankowe umożliwia proste operacje CRUD.
2. Mikroserwis klienta, który wykorzystuje wszystkie interfejsy serwisu bankowego.

Podczas wykonywania zadania wykrzystaj wiedzę zdobytą podczas zajęć oraz dostępne zasoby w internecie.