Consumer-Driven Contract Testing, the Developer's Perspective

ANDRIY REDKO

OCTOBER 28, 2019

CODE SPACE

About Me

- 20+ years as software developer and still ❤️ it
- Mostly Java and Scala these days
- Open-source contributor
- Socialize at https://www.linkedin.com/in/aredko
- Blog at https://aredko.blogspot.com/
- Code at https://github.com/reta

What we are going to talk about …

- Modern Software Systems
- Why Contracts?
- What Constitutes Contracts?
- Consumer-Driven Contract Testing
- Evolving Contracts
- Going Contract First or Code First?
- Contracts and Implementations

Modern Software Systems …

- Use microservices or service-based architectures
- As such, they are increasingly distributed
- Moreover, polyglot: Go, Java, Scala, JavaScript, …
- Rely on proven architectures, primarily (**but not limited to**) on
  - REST
  - Messaging

REST(ful) APIs: the easy parts

- HTTP protocol
- JSON representation (most of the time)
- Supported in (mostly) any programming language
- Easy to develop and consume

REST(ful) APIs: the hard parts

- The Richardson maturity model
  -> Level 0 – The Swamp of POX
  -> Level 1 – Resources
  -> Level 2 – HTTP Verbs
  -> Level 3 – Hypermedia Controls
- HAL, Hydra, JSON-LD, Siren, Collection+JSON, …

REST(ful) APIs: all parts together

POST /payments

```
{
  "timestamp":“...",
  "amount":102.32,
  "orderId":“...",
  "creditCard":{
    "ccv":"111",
    "expiration":"10/22",
    "number":"2222-1111-2333-2211",
    "holder":"John Smith"
  }
}
```

HTTP/1.1 201

```
{
  "id": “...",
  "status": "ACCEPTED“
}
```

Client

?

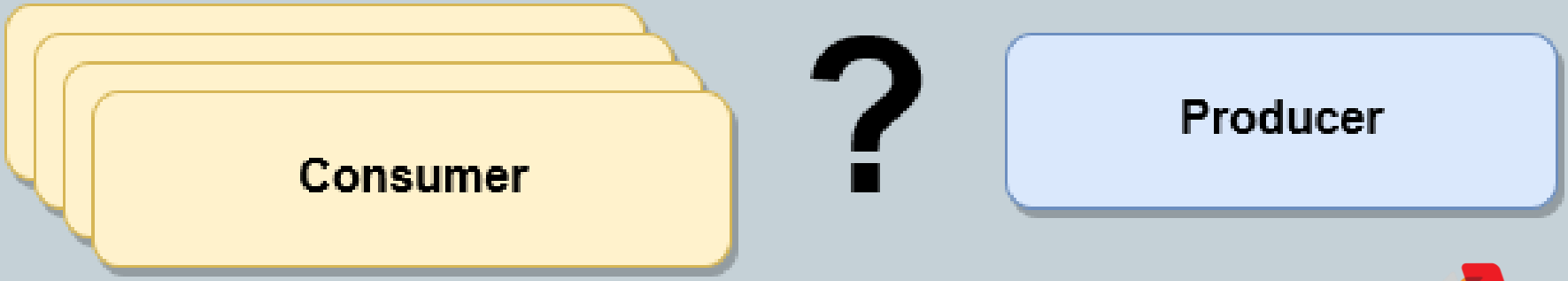
Provider

Messaging / Eventing

- Many architectural styles
  - EDA
  - CQRS
  - Event Sourcing
  - ...

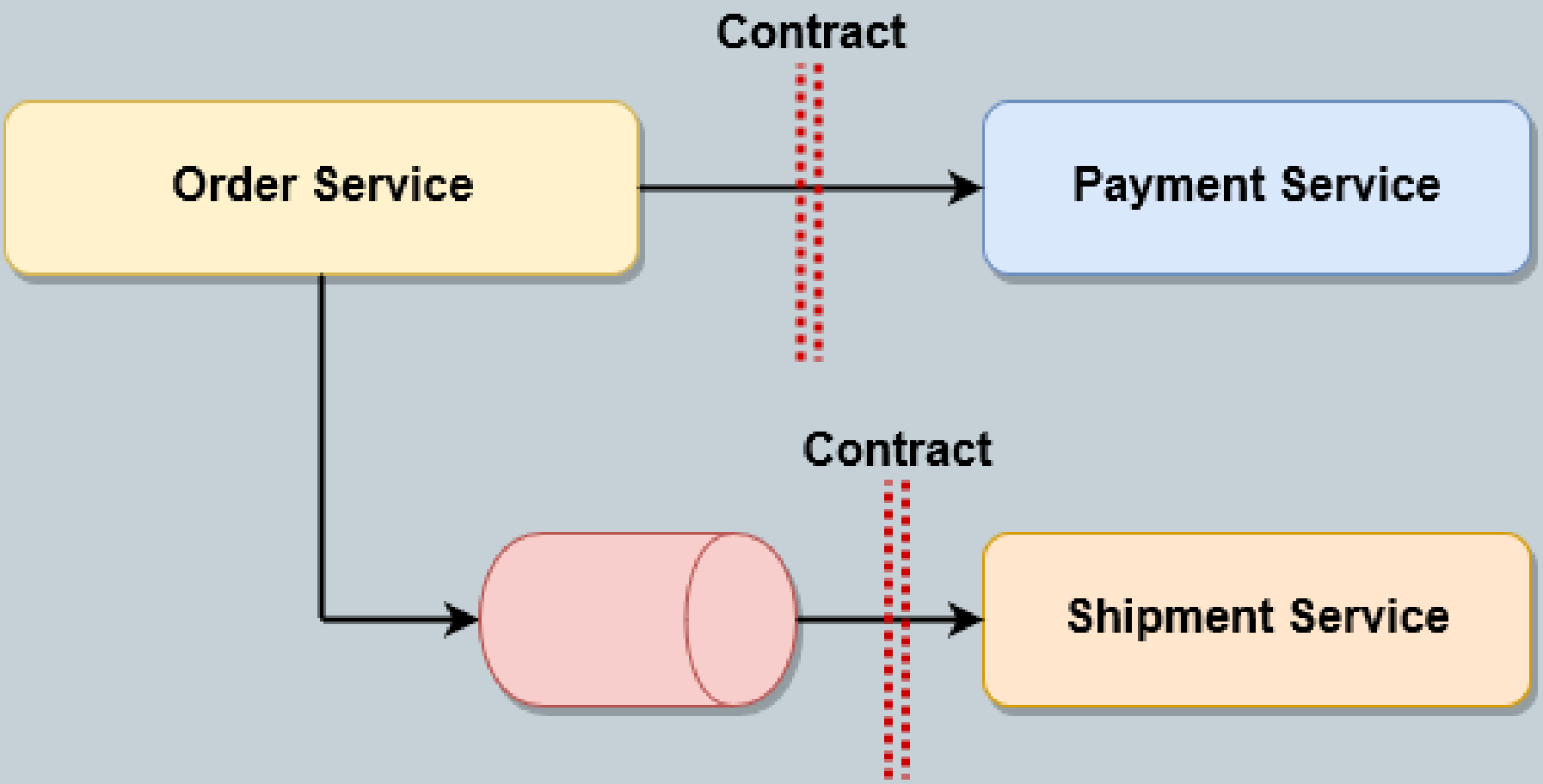
What are messages / events?

```
{
  orderId: "...",
  paymentId: "...",
  amount: 102.32,
  street: "1203 Westmisnter Blvrd",
  city: "Westminster",
  state: "MI",
  zip: "92239",
 country: "USA"
}
```



CODE SPACE

Why Contracts?



What Constitutes Contracts?

- WADL (the WSDL of REST)
- OpenApi / Swagger
- RAML
- Blueprint
- JSON schema

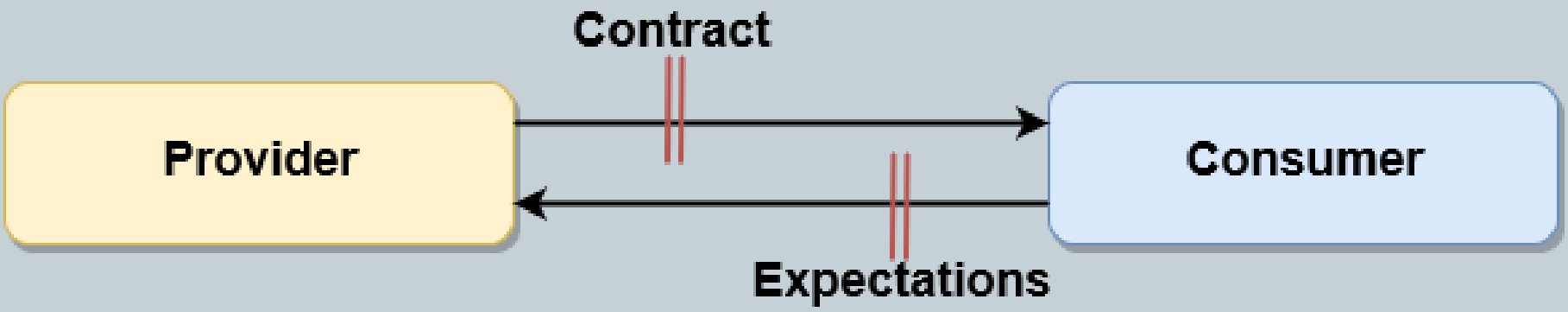
Hypermedia?

Consumer-driven contract testing

- What it means to be “consumer-driven”?
- Not a replacement to existing testing techniques but yet another one
- It is all about collaboration between clients / consumers and providers / producers.
- **Focuses on contracts not implementations (more about that later)**

Consumer-driven contract testing illustrated

- Provider publishes the contract
- Consumers express their expectations in terms of the contract
- Consumers may use only a subset of the available APIs



Consumer-driven contract testing tools

- The tooling is quite mature and flourishing
- Available for many languages, the examples we are going to see are based of JVM
- Pact (JVM)

  - https://github.com/DiUS/pact-jvm
- Spring Cloud Contract

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

Traditional Pact

- Could be used from mostly any programming language
- Notion of pacts between provider and consumer, each pact consists of
  - consumer identity
  - provider identity
  - interactions (request / response)
  - along with matching rules, generators, etc ...

Traditional Pact: example


```
{
  "provider": {
    "name": "Payment Service"
  },
  "consumer": {
    "name": "Order Service"
  },
  "interactions": [
    {
      "description": "POST new payment",
      "request": {
        "method": "POST",
        "path": "/payments",
        "headers": {
          "Content-Type": "application/json"
        },
        "body": {
          "amount": 100,
          "creditCard": {
            "ccv": "111",
            "expiration": "10/22",
            "holder": "string",
            "number": "string"
          },
          "orderId": "e2490de5-5bd3-43d5-b7c4-526e33f71304",
          "timestamp": "2000-01-31T08:00:00.000-05:00"
        }
      },
      "response": {
        "status": 201,
        "headers": {
          "Content-Type": "application/json"
        },
        "body": {
          "id": "e2d548c5-e1bf-407f-aed4-c973dc753e3e",
          "status": "ACCEPTED"
        },
      },
      "providerStates": [
        {
          "name": "default"
        }
      ]
    }
  }
}
```


CODE SPACE

Traditional Pact

DEMO time!

https://github.com/reta/consumer-driven-contract

Monoglot on JVM: Pact and OpenApi

- Providers and consumers are both on JVM (ideally, Java or/and Scala)
- Use Pact JVM along with Swagger Request Validator

https://bitbucket.org/atlassian/swagger-request-validator

new ValidatedPactProviderRule("*/contract/openapi.json*", *PROVIDER_ID*, "localhost", *port*, *this*);

Pact + OpenApi: Expectations

```
@Pact(provider = PROVIDER_ID, consumer = CONSUMER_ID)
public RequestResponsePact processPayment(PactDslWithProvider builder) {
  return builder
    .uponReceiving("POST new payment")
    .method("POST")
    .path("/payments")
    .headers(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON)
    .body(
      new PactDslJsonBody()
        .uuid("orderId")
        .decimalType("amount", new BigDecimal(102.33d))
        .stringType("notes")
        .timestamp("timestamp", "yyyy-MM-dd'T'HH:mm:ss.SSSXXX")
        .object("creditCard")
          .stringType("number")
          .stringType("holder")
          .stringMatcher("expiration", "\\d{2}/\\d{2}", "10/22")
          .stringMatcher("ccv", "\\d{3}", "111")
          .closeObject()
    )
    .willRespondWith()
    .status(201)
    .matchHeader(HttpHeaders.CONTENT_TYPE, MediaType.APPLICATION_JSON)
    .body(
      new PactDslJsonBody()
        .uuid("id")
        .stringMatcher("status", "REJECTED|ACCEPTED", "ACCEPTED")
    )
    .toPact();
}
```

Pact + OpenApi: Conversation

```
@Test
@PactVerification(value = PROVIDER_ID, fragment = "processPayment")
public void testProcessPayment() {
  given()
    .baseUri(provider.getConfig().url())
    .contentType(ContentType.JSON)
    .body(Json
      .createObjectBuilder()
      .add("orderId", "e2d548c5-e1bf-407f-aed4-c973dc753e3e")
      .add("amount", new BigDecimal(102.33d))
      .add("timestamp",
OffsetDateTime.now().format(DateTimeFormatter.ISO_OFFSET_DATE_TIME))
      .add("notes","Purchase Order #1")
      .add("creditCard", Json
        .createObjectBuilder()
        .add("number", "2222-1111-2333-2211")
        .add("holder", "John Smith")
        .add("expiration", "10/22")
        .add("ccv", "111"))
      .build(), ObjectMapperType.JOHNZON)
    .post("/payments")
    .then()
    .log()
    .all();
}
```

Monoglot on JVM: Pact and OpenApi

DEMO time!

https://github.com/reta/consumer-driven-contract

Spring Cloud Contract

- Multiple flavors to write tests
- Includes Pact support
- Looks simple at a glance …
- … but could be difficult to grasp
- … and it is driven by plugins (Maven, …)
- Uses **client/stub** and **server/test** notation
- And a lot of Groovy

Spring Cloud Contract, the Pact style (consumer)

- Defines the specification

```
org.springframework.cloud.contract.spec.Contract.make {
  request {
    ....
  }
  response {
    ....
  }
}
```

- Publishes the stubs

```
mvn clean install -DskipTests=true
```

- Creates tests against stubs

Spring Cloud Contract, the Pact style (provider)

- Defines the missing base classes

```
public class PaymentBase {
    @Before
    public void setup() {
        RestAssuredMockMvc.standaloneSetup(new PaymentController());
    }
}
```

- Run generated tests

```
mvn test
```

Spring Cloud Contract example (Groovy)

```
org.springframework.cloud.contract.spec.Contract.make {
  request {
    method 'POST'
    url '/payments'
    body([
        orderId: $(regex('[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}')),
        amount : 102.32,
        timestamp: '2019-09-29T20:43:03.6977944-04:00',
        creditCard: [
          number: $(regex('[0-9]{4}-[0-9]{4}-[0-9]{4}-[0-9]{4}')),
          holder: 'John Smith',
          expiration: $(regex('[0-9]{2}/[0-9]{2}')),
          ccv: $(regex('[0-9]{3}'))
        ]
    ])
    headers {
      contentType('application/json')
    }
  }
  response {
    status CREATED()
    body([
      id: $(regex('[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}')),
      status: 'ACCEPTED'
    ])
    headers {
      contentType('application/json')
    }
  }
}
```

CODE SPACE

Spring Cloud Contract

DEMO time!

https://github.com/reta/consumer-driven-contract

Messaging Contracts

- Communication is not only about REST APIs
- Asynchronous message-driven/event-driven systems are norm
- Events and messages are part of the service contract
- Quite often represented in JSON
- Subject to the consumer-driven contract testing

- **Please consider using schemas-based mechanisms for serializing structured data (Avro, Thrift, Protocol Buffers, FlatBuffers, …)**

Messaging Contracts and Pact

- Supported starting from Pact specification v3.0
- Very similar approach to the traditional REST service contracts
- Instead of interactions, the concept of **messages** has been introduced
- Other elements (consumer and provider identities, matching rules, generators, ...) stay the same

Pact Messaging Example


```
{
 "consumer": {
  "name": "Shipment Service"
 },
 "provider": {
  "name": "Order Service"
 },
 "messages": [
  {
   "description": "an Order confirmation message",
   "metaData": {
    "contentType": "application/json"
   },
   "contents": {
    "zip": "string",
    "country": "string",
    "amount": 100,
    "orderId": "e2490de5-5bd3-43d5-b7c4-526e33f71304",
    "city": "string",
    "paymentId": "e2490de5-5bd3-43d5-b7c4-526e33f71304",
    "street": "string",
    "state": "string"
   },
   "providerStates": [
    {
     "name": "default"
    }
   ],
  }
 ]
}
```


CODE SPACE

Messaging Contracts and Pact

DEMO time!

https://github.com/reta/consumer-driven-contract

Messaging and Spring Cloud Contract

- Very similar approach to the REST service contracts
- However a few things are inverted
- Clear line between producers and consumers
- Not generic in a sense, has a number of concrete integrations (AMQP, Kafka, JMS, Spring Integration, Spring Cloud Stream, ...)

Messaging and Spring Cloud Contract (producer)

- Defines the specification

```
org.springframework.cloud.contract.spec.Contract.make {
   input {
     ...
   }
   outputMessage {
    ...
   }
}
```

- Publishes stubs

```
mvn clean install -DskipTests=true
```

Messaging and Spring Cloud Contract (producer)

- Defines missing base classes

```
@RunWith(SpringRunner.class)
@SpringBootTest
@AutoConfigureMessageVerifier
public class OrderBase {
    ...
}
```

- Run tests

```
mvn test
```

Messaging and Spring Cloud Contract (consumer)

- Imports stubs
- Creates tests against stubs
- Picks the messaging integration (AMQP, Kafka, ...)
- Uses **StubFinder** to trigger a particular message

```
@Autowired private StubFinder stubFinder;
```

- Uses **MessageVerifier** to capture and assert against received messages

```
@Autowired private MessageVerifier<Message<?>> verifier;
```

- Run tests

```
mvn test
```

Spring Cloud Contract example (Groovy)

```
org.springframework.cloud.contract.spec.Contract.make {
  name "OrderConfirmed Event"
  label 'order'

  input {
    triggeredBy('createOrderTriggered()')
  }

  outputMessage {
    sentTo 'orders'

    body([
      orderId: $(regex('[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}')),
      paymentId: $(regex('[0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12}')),
      amount: 102.32,
      street: '1203 Westmisnter Blvrd',
      city: 'Westminster',
      state: 'MI',
      zip: '92239',
      country: 'USA'
    ])
    headers {
      header('Content-Type', 'application/json')
    }
  }
}
```

Messaging Contracts and Spring Cloud Contract

DEMO time!

https://github.com/reta/consumer-driven-contract

Evolving RESTful APIs

- Consumer-driven contract testing helps to keep integrations on both sides (consumers and providers) healthy
- You may not get feedback from all consumers
- Every more or less successful system evolves over time
- It means contracts have to evolve as well, the changes are the only constant
- How providers should approach that without risking to break promises?

Evolving RESTful APIs: backward compatibility

- What kind of changes are backward compatible?

  - adding a new **non-required** property to request
  - adding a new property to response
  - adding a new **non-required** query parameter
  - adding a new API endpoint

Evolving RESTful APIs: backward compatibility

- What kind of changes are **NOT** backward compatible?

- removing properties from requests, responses, query parameters

- renaming properties in the requests, responses, query parameters

- removing or relocating API endpoints

- changing data types (f.e. string -> integer)

- changing representations (f.e. dates)

Evolving RESTful APIs: tooling

- **swagger-diff (OAS v2.0)**: standalone Ruby tool
- **openapi-diff (OAS v3.0)**: standalone Java tool
- **assertj-swagger (OAS v2.0)**: JUnit scaffolding for JVM / Java projects

Evolving RESTful APIs: swagger-diff

- **swagger-diff** - is an utility for comparing two different Swagger / OpenAPI v2.0 specifications

$ swagger-diff -i <old> <new>

Prints a list of any backwards-incompatibilities new has when compared to old.

https://github.com/civisanalytics/swagger-diff

swagger-diff: sample output

$ swagger-diff -i old.json new.json

- incompatible request params
  - post /orders
    - new required request param: amount
    - new required request param: id

Evolving RESTful APIs: openapi-diff

- **openapi-diff** - is an utility to compare two OpenAPI v3.0 specifications

$ openapi-diff <old> <new>

Prints a list of any changes the new has when compared to old.

https://github.com/quen2404/openapi-diff

openapi-diff: sample output

```
$ openapi-diff old.yaml new.yaml

=================================================
==                  API CHANGE LOG             ==
=================================================
                  Payment Service
-----------------------------------------------------------------
--                  What's Changed                             --
-----------------------------------------------------------------
- POST   /payments
  Request:
      - Changed application/json
        Schema: Broken compatibility
-----------------------------------------------------------------
--                  Result                     --
-----------------------------------------------------------------
            API changes broke backward compatibility
-----------------------------------------------------------------
```

CODE SPACE

Evolving RESTful APIs: assertj-swagger

- **assertj-swagger** - a Swagger assertj test library which compares a contract-first Swagger YAML/JSON file with a code-first Swagger JSON

```
@Test
public void validate() {
    SwaggerAssertions
        .assertThat(actual)
        .isEqualTo(published);
}
```

https://github.com/RobWin/assertj-swagger

assertj-swagger

DEMO time!

https://github.com/reta/consumer-driven-contract

Contract First or Code First?

- Contract-first vs Code-first: never ending debate
- Not in scope of our discussion but …
- … how to get contracts from code?
- … how to get code from contracts?

Code-First: Contract From Code

- Generate contract from code

```
<plugin>
  <groupId>io.swagger.core.v3</groupId>
    <artifactId>swagger-maven-plugin</artifactId>
    <version>2.0.9</version>
    <configuration>
      <outputFileName>openapi</outputFileName>
     <outputFormat>JSONANDYAML</outputFormat>
      ....
    </configuration>
</plugin>
```

Generating Contract

DEMO time!

https://github.com/reta/consumer-driven-contract

Contract-First: Code From Contract

- Ability to generate service skeletons
- Ability to generate service clients

```
<plugin>
    <groupId>io.swagger.codegen.v3</groupId>
    <artifactId>swagger-codegen-maven-plugin</artifactId>
    <version>3.0.13</version>
    <configuration>
        ...
    </ configuration >
</plugin>
```

Generating Service Client

DEMO time!

https://github.com/reta/consumer-driven-contract

What about gRPC, GraphQL, …?

- Today REST dominates HTTP web APIs but innovations never stops
- gRPC, a contract-first (Protobuf, …) universal RPC framework, mostly used with HTTP/2
- GraphQL, a schema-driven query language for APIs, runs over HTTP in case of web APIs
- Introspected REST, an alternative to REST and GraphQL
- More to come …

Contract and Implementation

- Another area where consumers may struggle a lot is slight changes in the representation or serialization format. Those are very difficult to catch since it is mostly the implementation driven change.

https://github.com/twitter/diffy

Conclusions

- Providers / Producers publish their contracts
- Clients / Consumers express their expectations with respect to published contracts
- Consumer-driven contract testing encourages collaboration between clients / consumers and providers / producers
- Clients / Consumers stay assured they will not be impacted by contract changes
- Providers / Producers are able to validate every single contract change before publishing it

THANK YOU!

Questions?

CODE SPACE