**PACT**

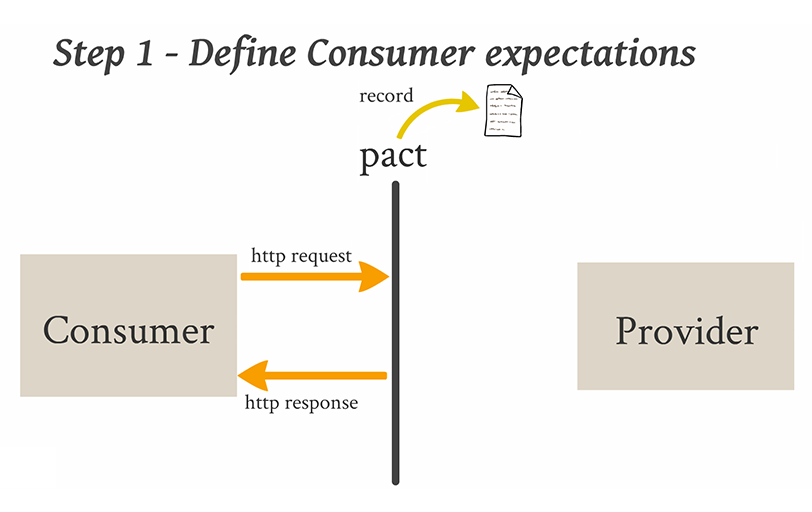
**Pre-requisites:**

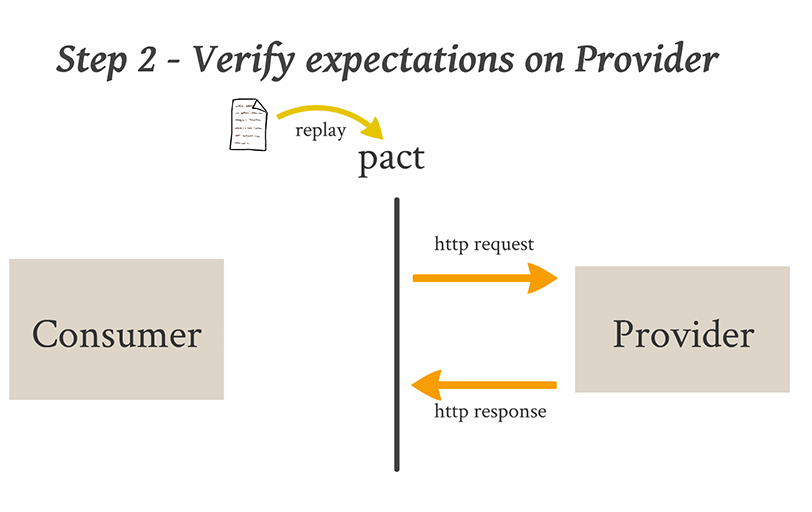
* Java
* JDK 1.7+
* Maven 3+
* Spring Boot

**What is PACT?**

The **Pact** family of testing frameworks provide support for **Consumer Driven Contract Testing** between dependent systems where the integration is based on HTTP (or message queues for some of the implementations). They are particularly useful for µ-services.

**What PACT actually does?**





**Our understanding of PACT:**

Pact works with the concept of Consumer Driven Contracts (CDC).

CDC is the contract developed by a consumer against a provider from a consumer point of view.

Consumer defines the shape of **“request to be sent to the provider and response to be given by the provider”**. The format of requests and responses are defined in JSON file called PACT file.

The pact file is generated with the support of mock provider.

This shaped PACT file is tested against the actual provider.

Two main steps in PACT testing,

1. Generating PACT file through mock provider.
2. Testing the PACT against the original provider.

**Our example**:

We have taken two micro services namely

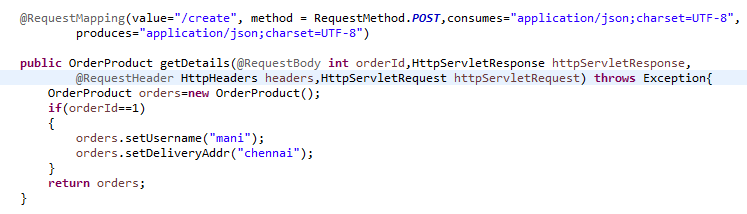
1. Shipping (Consumer)
2. Order (Provider)

Here shipping service acts as a consumer, because it requests for values from Order service and consumes the response provided by Order Service.

Order service as a provider because it responds for values requested by consumer.

In our example,

**Our order service (provider) be like,**



When we request for the details of orderId: 1

URL: http://localhost:8224/orders/create

With the request body: 1

Response is:

{

"orderId": null,

"productId": 0,

"customerId": 0,

"username": "mani",

"paymentMode": null,

"billingAddress": null,

"deliveryAddr": "chennai",

"orderQuantity": 0,

"totalAmt": 0 }

So we have to define our pact with request and responses as seen above.

1. **Generate a PACT file**

* We define a mock server’s host and port to represent the provider.

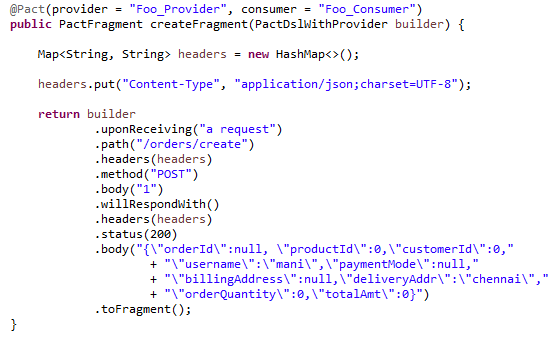


This mock provider runs on any host and port to verify the request & response

We can specify the host and the port if needed

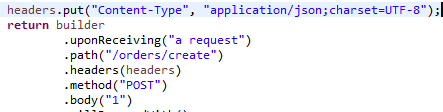


* We define a PACT fragment which defines the contract of requests and responses.

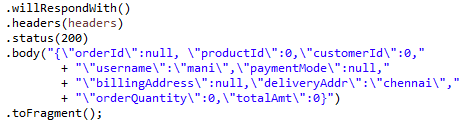


In the above code, we have defined our requests and responses.

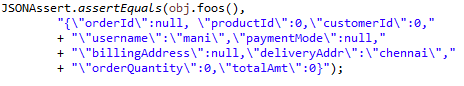
i.e.., The provider (Order) upon receiving a request with the path /orders/create, content type as application/json, POST method with body 1



should respond with the corresponding body as defined below WillRespondWith() in the code.



We ensure that the constructed fragment matches the response from the mock server.

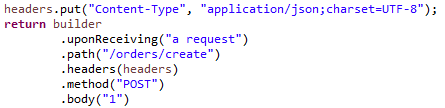


* assertEquals compares the expected response with the actual response.
* Obj.foos() takes the mock server(defined in @rule) and executes the request

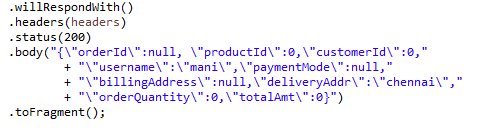
defined in ConsumerPort.java🡪foos() to produce the response as defined in

PactDslWithProvider (createFragment() which builds the PACT file).

* The request in obj.foos() should be identical to the request defined in createFragment() .



This request will respond with the body taken from the createFragment() function. (Since the provider is mock, it takes the response body from createFragment())



Once assertEquals gets success, PACT file will be generated under Target /Pact folder.

{

"provider": {

"name": "Foo\_Provider"

},

"consumer": {

"name": "Foo\_Consumer"

},

"interactions": [

{

"providerState": null,

"description": "a request",

"request": {

"method": "POST",

"path": "/orders/create",

"headers": {

"Content-Type": "application/json;charset=UTF-8"

},

"body": 1

},

"response": {

"status": 200,

"headers": {

"Content-Type": "application/json;charset=UTF-8"

},

"body": {

"billingAddress": null,

"customerId": 0,

"deliveryAddr": "chennai",

"orderId": null,

"orderQuantity": 0,

"paymentMode": null,

"productId": 0,

"totalAmt": 0,

"username": "mani"

}

}

}

],

"metadata": {

"pact-specification": {

"version": "2.0.0"

},

"pact-jvm": {

"version": "3.2.2"

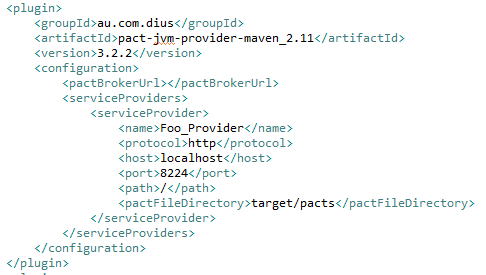
}

}

}

1. **Verify PACT against actual provider**

For this specify the plugin for provider in pom.xml



The actual provider’s port should be given in this plugin (Ensure the service running in that port should be Up).

And also mention the PACT file directory. <pactFileDirectory>target/pacts</pactFileDirectory>

**How it runs**: The actual provider will take the request from the pact file and verifies the actual response of the provider against the response in the contract (pact file)

Once the verification is successful the contract is achieved.

**Order Service:** 

**Download project here:** 

**How to run this microservices-pact-cosnumer project?**

This project contains a **very simple** demo of using **consumer-driven contracts** to verify the interactions between micro services. It leverages Spring Boot for both the provider and consumer services.

**Note**: Please ensure that the provider ‘Order Service’ is up in the port 8824 while verifying the PACT.

1. Run the build with tests using Maven

**mvn clean install**

1. This will result in the creation of a pact file called **Foo\_Consumer-Foo\_Provider.json** at microservices-pact-maven/microservices-pact-consumer/target/pacts
2. To verify against actual provider run this command:

**mvn au.com.dius:pact-jvm-provider-maven\_2.11:verify**