**DevOps Playground: Hands on with Contract Testing using Pact**



# Introduction

On this meetup, we will walk you through what Contract testing. We will explain how PACT works and do a hands-on workshop creating a pact between two micro services and testing it.

Name: Samer Naqvi

Role: Continuous Delivery Consultant

Email: [Samer.Naqvi@ecs-digital.co.uk](mailto:Samer.Naqvi@ecs-digital.co.uk)

LinkedIn: [Let's connect](https://www.linkedin.com/in/samer-naqvi-8b27a3/)

## **Requirements**

* Jdk
* Code editor (e.g. Eclipse, Vi) basic knowledge
* Git
* Maven
* Chrome or any other browser
* Basic Java knowledge
* Basic Linux cli knowledge

## **Setup**

The remote machine will have all the programs you need to follow this session. On a new tab, please open [Remote Machine](https://rdp.devopsplayground.com/#/) to launch the login screen to connect to the test machines. For any technical assistance, please ask around for help.

If you wish to use your own machine on this playground, please make sure you have all the applications and/or programs listed on the requirements and clone or download this repository.

## **Further Reading**

* [Official Pact Documentation](https://docs.pact.io/)
* [Contract Tests vs Functional Tests](https://docs.pact.io/best_practices/consumer/contract_tests_not_functional_tests)
* [Getting started with Pact and Consumer Driven Contract Testing](https://dius.com.au/2016/02/03/microservices-pact/)
* [Pact jvm consumer junit](https://github.com/DiUS/pact-jvm/tree/master/pact-jvm-consumer-junit)
* [Pact Broker](https://github.com/pact-foundation/pact-broker-docker)

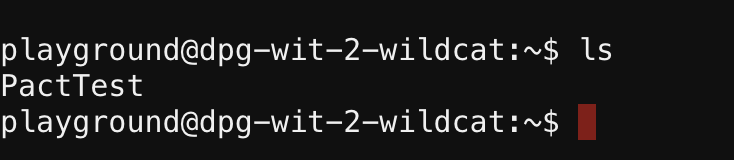
# Survey Link

We would love to hear your feedback and make improvements on the way we deliver our playground session. Please click the [survey](https://qabook.typeform.com/to/Ijm7N1) to participate.

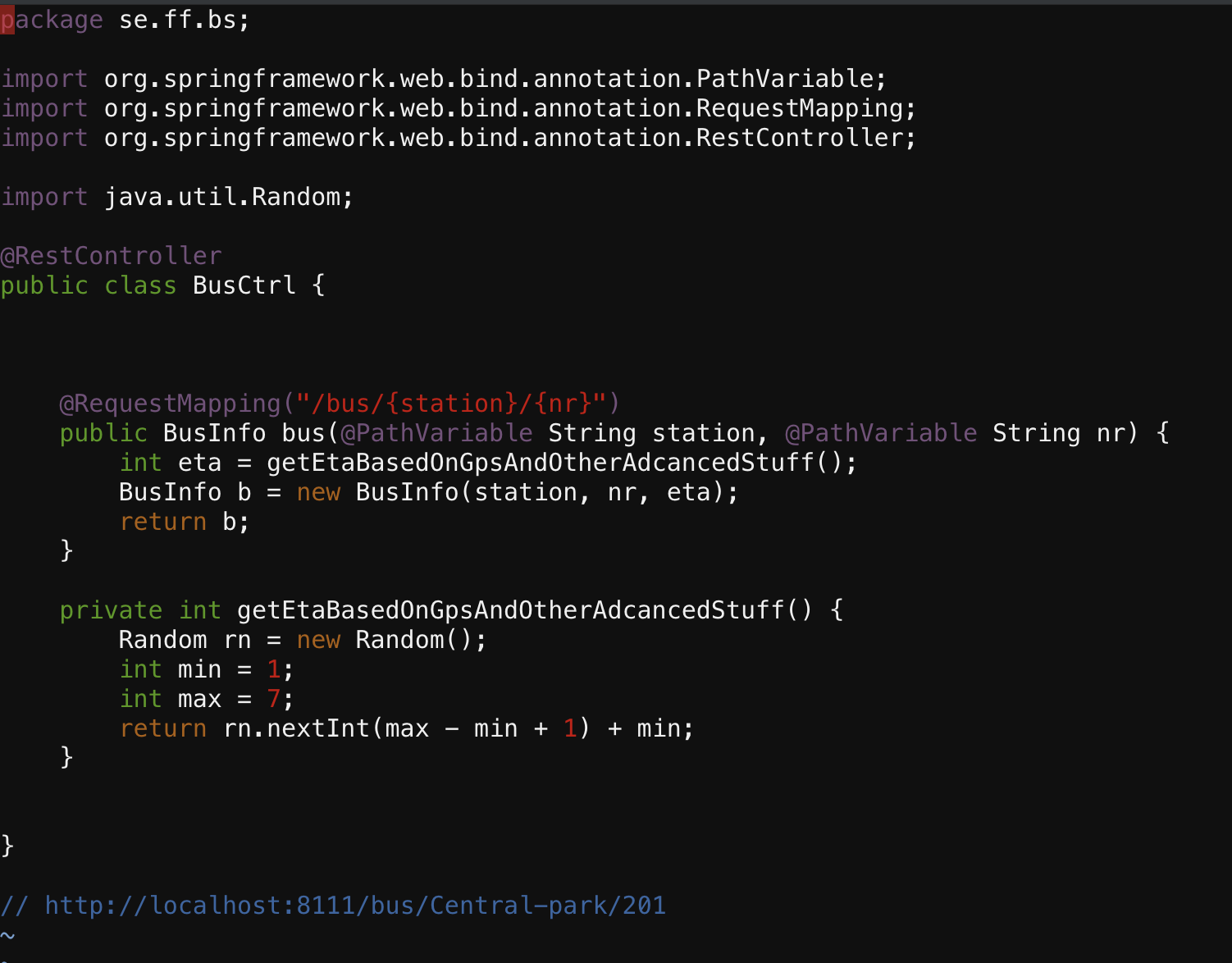
# Pact Project

Once you are successfully logged in to your instance, follow the steps below to open the project.

1. On the terminal write ‘ls’ and press Enter, to make sure your Pact project is present.



1. Let’s first set the paths as variables.
2. BUSAPP=”**PactTest/bs/src/main/java/se/ff/bs”**
3. BUSSPRINGBOOT=”**pact/PactTest/bs”**
4. BUSCOMES=**”PactTest/client/src/main/java/se/ff/bsc”**
5. BUSTEST=” **PactTest/client/src/test/java/se/ff/bsc”**
6. CLIENT=”**PactTest/client”**
7. BROKER=” **PactTest/dockerpactbroker”**
8. VERIFY= **“PactTest/verifyer/[src](https://github.ecs-digital.co.uk/samernaqvi/PactTest/tree/master/verifyer/src)/**[**test**](https://github.ecs-digital.co.uk/samernaqvi/PactTest/tree/master/verifyer/src/test)**/**[**java**](https://github.ecs-digital.co.uk/samernaqvi/PactTest/tree/master/verifyer/src/test/java)**/**[**se**](https://github.ecs-digital.co.uk/samernaqvi/PactTest/tree/master/verifyer/src/test/java/se)**/[ff](https://github.ecs-digital.co.uk/samernaqvi/PactTest/tree/master/verifyer/src/test/java/se/ff)/[bsv](https://github.ecs-digital.co.uk/samernaqvi/PactTest/tree/master/verifyer/src/test/java/se/ff/bsv)”**
9. Type **cd $BUSAPP.** Click **Enter**
10. Type or copy/paste vi BusCtrl.java , Press Enter
11. The above step will open the class BusCtrl.java in vi terminal, which will look like this.



This is the bus application (spring boot) that has a controller, **BusCtrl**. This service takes station and bus number as input variables and then it uses the method **getEtaBasedOnGpsAndOtherAdancedStuff().** This method returns an integer which then creates a **BusInfo** object with station number and the time left.

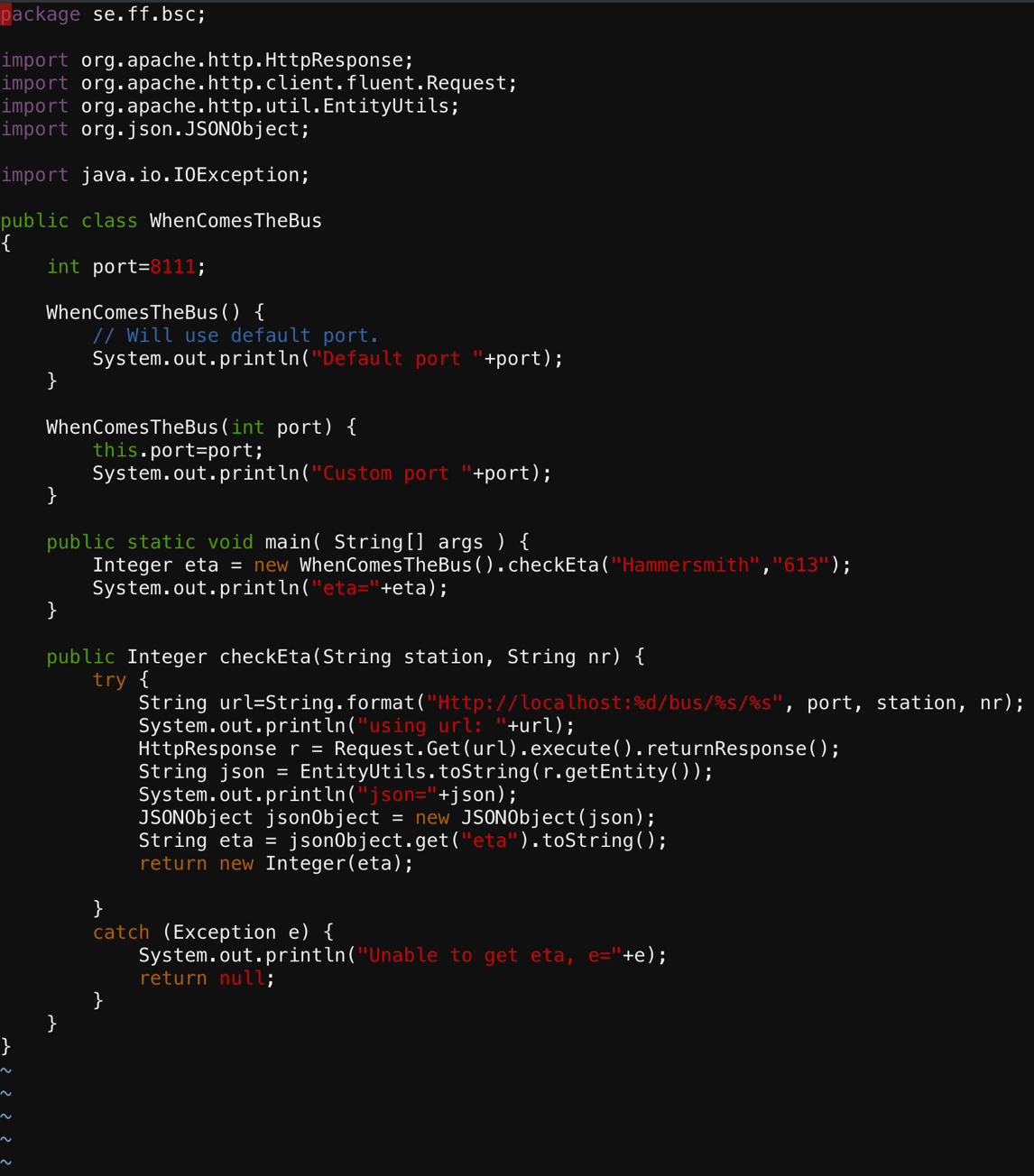
1. Type ‘ **:q** ‘ and press Enter. This will take you back to your present working directory.
2. Type ‘**cd**’ and press Enter.
3. Type **cd** **$BUSSPRINGBOOT**, Press **Enter**
4. Now start the spring boot service by giving the command: mvn spring-boot:run
5. Go to browser/new tab and type

**http://<address of your instance>:8111/bus/Central\_station/60**

## **Class WhenComesTheBus.java (Client)**

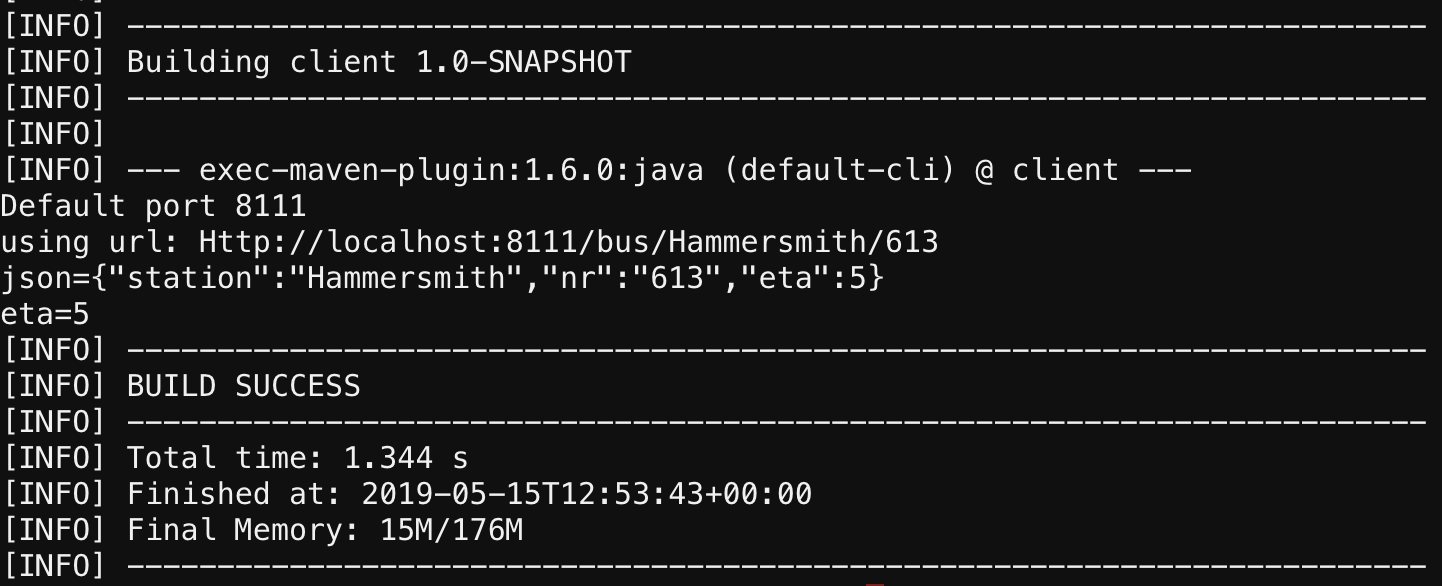
We can now write out client side test.

1. Open a new tab in your browser and enter address of your linux instance again and log in.
2. Copy paste **cd** **$BUSCOMES.** Click **Enter**
3. Type vi WhenComesTheBus.java
4. The class WhenComesTheBus.java will open in the vi editor.



This class uses port 8111 as default port. It has the method **checkEta()** that checks estimated time for the station Hammersmith and bus number 613. It than prints out the results. CheckEta() method looks at local host, port, station and the bus number. It than makes an HTTP request and transform it into json and extract the integer from the result.

1. Type ‘ **:q**  ‘ and press **Enter** to exit the vi editor
2. Go back to the parent working directory by typing ‘**cd**’ and press **Enter**.
3. Type or paste **cd** **$CLIENT**, press **Enter**.
4. Let’s run the class WhenComesTheBus.java by typing the command mvn exec:java -Dexec.mainClass=se.ff.bsc.WhenComesTheBus
5. The class should run and give you the result as below:



## **Pact Test WhenComesTheBusTest.java**

Ok let’s now create the pact file. The test that we are going to write is based on an example I picked from [pact-jvm](https://github.com/DiUS/pact-workshop-jvm).

1. Let’s clear the path by typing **cd** and then **Enter**.
2. Now let’s write our test class WhenComesTheBusTest.java. I have already created a shell class for that. Type in **cd $BUSTEST**, press **Enter**.
3. Type in **vi WhenComesTheBusTest.java** and then press **Enter**.
4. You should now see an empty class WhenComesTheBusTest.java. Copy the code below and paste it in the vi editor. (Press **i** to make vi in edit mode)

In the code below we add the pact rule to represent our provider. The hostname and port are optional. If left out, it will default to 127.0.0.1 and a random available port. You can get the URL and port from the pact provider rule.

We are using Port 8112 which will create a mock service. And when we run the pact verification method, **doTest(),** the mock service will get populated with the information that we setup in the Pact annotated method.

@Rule

**public** PactProviderRuleMk2 provider = **new** PactProviderRuleMk2("BusService", "localhost", 8112, **this**);

@Pact(consumer = "BusServiceClient")

**public** RequestResponsePact createPact(PactDslWithProvider builder) {

Map<String, String> headers = **new** HashMap();

headers.put("Content-Type", "application/json");

DslPart etaResults = **new** PactDslJsonBody()

.stringType("station","Hammersmith")

.stringType("nr","613")

.integerType("eta",4)

.asBody();

RequestResponsePact result = builder

.given("There is a bus with number 613 arriving to Hammersmith bus station")

.uponReceiving("A request for eta for bus number 613 to Hammersmith bus station")

.path("/bus/Hammersmith/613")

.method("GET")

.willRespondWith()

.status(200)

.headers(headers)

.body(etaResults).toPact();

**return** result;

}

@Test

@PactVerification()

**public** **void** doTest() {

System.*setProperty*("pact.rootDir","../pacts"); // Change output dir for generated pact-files

Integer eta = **new** WhenComesTheBus(provider.getPort()).checkEta("Hammersmith", "613");

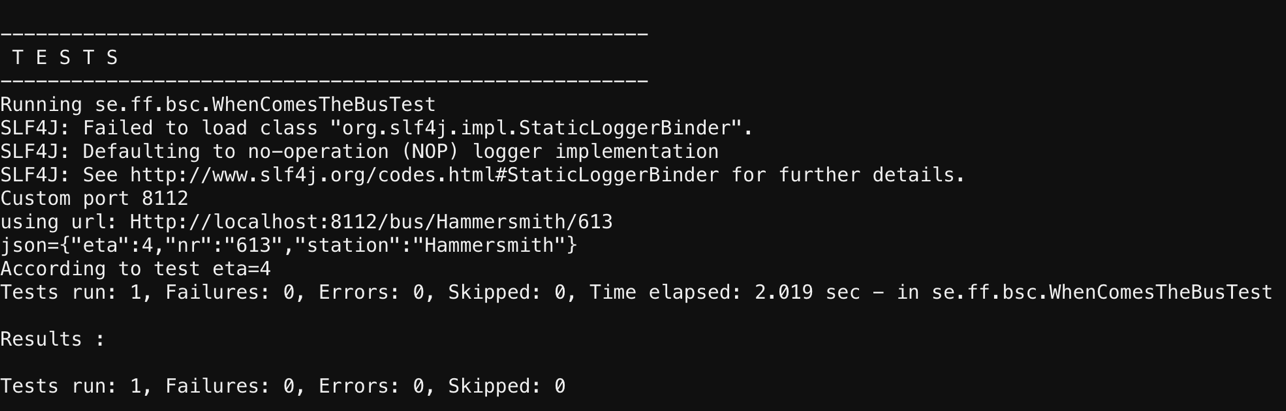
System.***out***.println("According to test eta="+eta);

*assertTrue*(eta >= 0);

}

In the above code, once the **@PactVerification()** method is run, the mock service will get populated with the information that is setup in the **@Pact** annotated method. In that method we declare the **PactDslJsonBody()**, where we say that we are depending on a string type named ‘**station’**, string type named ‘**nr**’ and a integer type named ‘**eta**’. These are the things that we need from the consumer. We also say that the status should be 200 and the header should be what we have given in **headers.put("Content-Type", "application/json")**.  
In the **@PactVerification()** method, we first set the root directory for the pact file to be created in. In the next step than we use the class **WhenComesTheBus** at the provider port, which means that it won’t use the live application data but the mock/virtual data. Than the method **checkEta** is called with station name and bus number.

1. Save the above code by typing ‘:**wq** ’ and press **Enter**.
2. Run the testcase WhenComesTheBusTest.java by going back to root directory. Type ‘ **cd** ‘ press **Enter**. Type or paste **cd $CLIENT**, press **Enter**.
3. Type ‘ **mvn test** ‘ and press **Enter**. You should see the below result in the terminal.



1. By running the above pact test will create a json pact file in the location /PactTest/client/target/pacts
2. Once you’re in that directory, type the command vi BusServiceClient-BusService.json

## **Pact file**

1. By entering the command in the above step, you will see the json pact file in vi editor



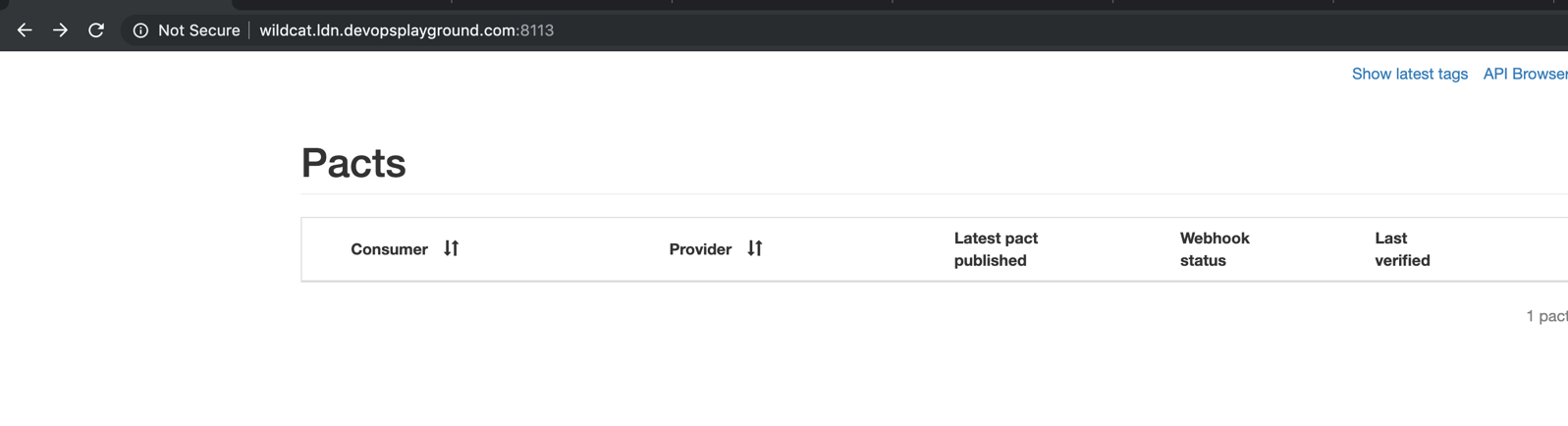
You can see in the above json file that it is a pact between provider **BusService** and a consumer **BusServiceClient**. It contains the description, request method, path, the response and the body. A bit further down there are the **matchingRules**, which verifies that the station and nr matches type and the eta should be of type integer. We have also **providerStates,** which can be used for a test on the provider side.

1. Type ‘ **:q**  ‘ and press **Enter** to exit the vi editor

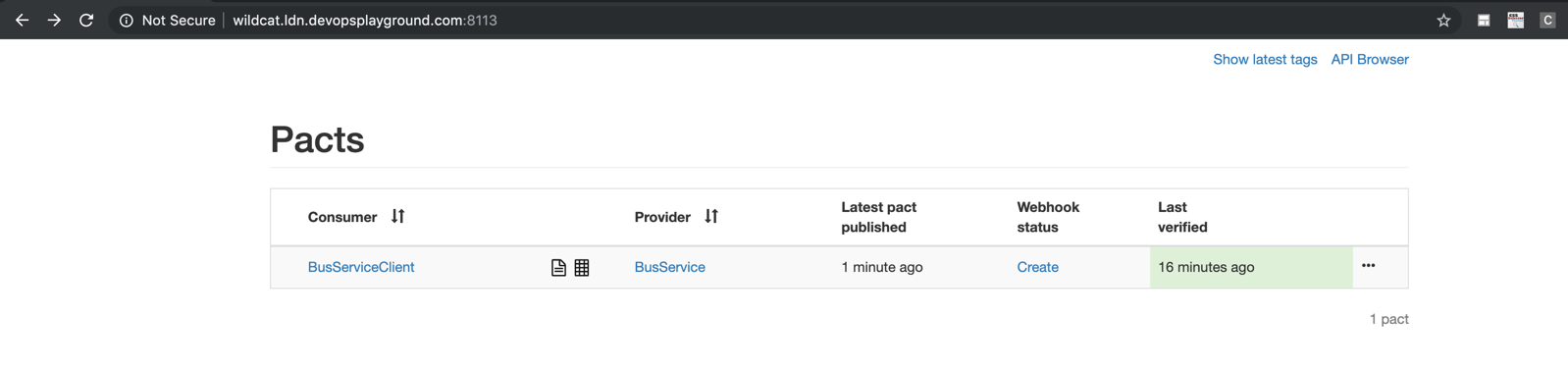
## **Pact Broker**

We will be using a docker pact broker. I have used an existing pack broker. Lets start the pack broker which is on a docker compose

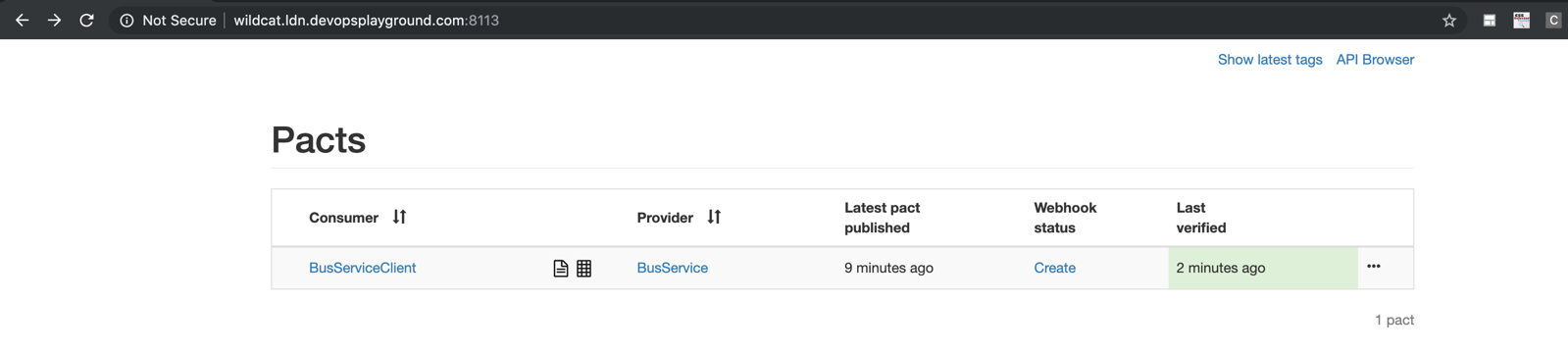
1. Type ‘ **cd** ‘ and **Enter** to go to root directory.
2. Type **cd $BROKER**
3. This folder contains the file docker-compose.yml file. Type in docker-compose up
4. Open a separate tab on your browser and copy paste your linux instance address **<Linus instance address>:8113** which should open up your docker compose like below:

****

1. Currently your pack broker does not have your pact file. So let’s publish the pact file onto the pact broker. For this we need to go to the client folder. Type’ **cd** ‘ and **Enter**, then type **cd $CLIENT**, press **Enter**.
2. Let’s publish the pact file by giving the command mvn pact:publish
3. Go to the tab that has your docker compose running and refresh the page. You should now see your pact file on it as below.

****

1. You might have noticed that the pact file is not verified on your broker. In order to verify we need to go to be in the **bs** folder. Lets go up one folder in our current directory and then type ‘ **cd bs** ‘
2. Type in mvn pact:verify
3. The above command should verify your pact file on your pact broker. Refresh the tab on which your pack broker is running and it should display the last verified time.

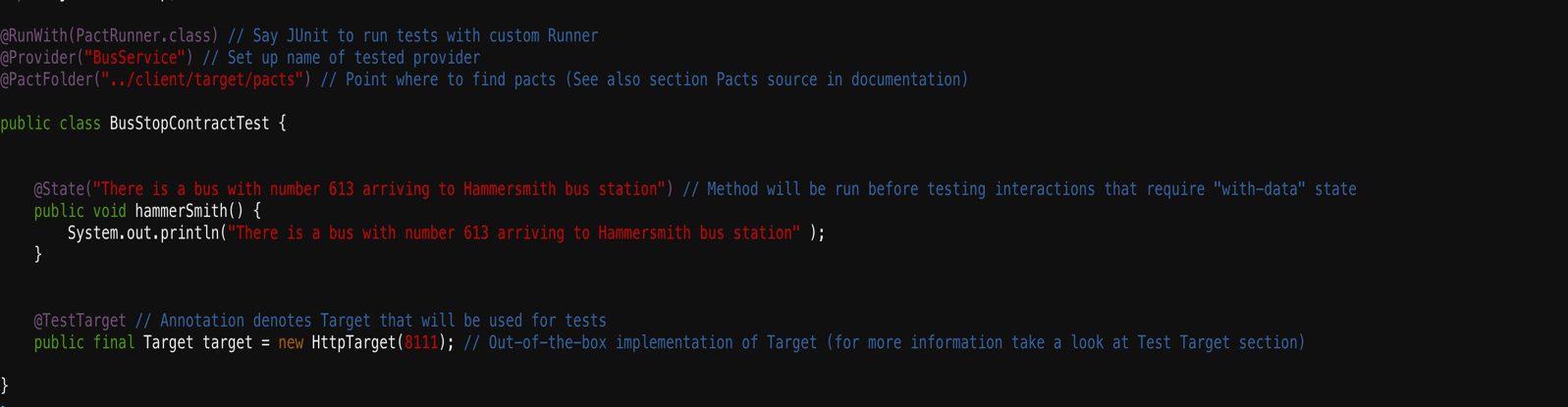


If you click the link BusServiceClient, you will be displayed a graphical image of the BusServiceClient relationship with any other service.

## **BusStopContractTest (Provider)**

We run BusStroContractTest.java class to verify that our live system works according to the client specification/contract that is created.

1. Go back to the parent working directory by typing ‘**cd**’ and press **Enter**.
2. Type **cd $VERIFY** and press **Enter**.
3. Type vi BusStopContractTest.java



It contains a **@State**("There is a bus with number 613 arriving to Hammersmith bus station"), which is given in the WhenComesTheBus.java class. This state was also given in the pact file as **Provider state**. The @state annotation is given in test to ensure for example to insert something in the database or other things to be in place for this test to work. I have kept this test simple hence nothing is being inserted in this test and the state is just given as a statement. But if this state is removed entirely from this class, it will fail since in the contract we have given the provider to be in the state of “There is a bus with number 613 arriving to Hammersmith bus station”.

**@TestTarget** annotation targets the system to be tested. In the above example we are pointing it to port that the live system is running on. This test also needs to have access to the pact file in order for it to verify in the annotation @PactFolder("../client/target/pacts"). It is also necessary for it to be given the same name in @Provider("BusService") that is in the pact file for the provider.

Let run this test now.

1. Let’s go to main directory verify i.e. **PactTest/verifier**
2. Type **mvn test** and press **Enter**

The result should look like below.

