# Consumer Driven Contract Testing

常用术语：

Consumer 消费者

Provider 提供者

Contract 契约，是由consumer端和provider端共同定义的接口规范，包括接口访问的路径，输入和输出数据，是一个json

PACT

基于消费者驱动的框架之一，主要是Ruby，还支持JVM, JavaScript, .NET, GO, Python等。

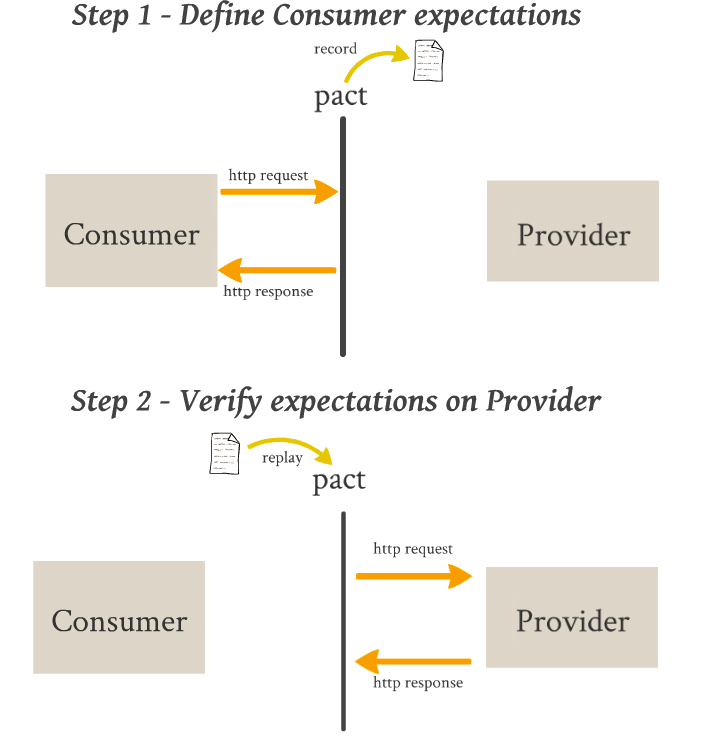
* Enables consumer driven contract testing, providing a mock service and DSL for the consumer project, and interaction playback and verification for the service provider project.

Pact Broker

保存契约文件的服务器，也是Ruby的

* solves the problem of how to share pacts between consumer and provider projects
* allows you to [decouple your service release cycles](http://techblog.realestate.com.au/enter-the-pact-matrix-or-how-to-decouple-the-release-cycles-of-your-microservices/)
* provides API documentation that is guaranteed to be up-to date
* shows you real examples of how your services interact
* allows you to visualise the relationships between your services

## How does it work?



**测试过程：**

Step 1 消费者定义契约

测试环境：

Windows（Failed）：Python、Ruby 在windos环境，通过github上的例子，无法生成文件；

Linux（Pass）：Python在centos上，成功生成json文件

Step 2 验证提供者

未验证，python的github文档没有验证提供者的例子，Ruby有，但是没在centos上装ruby；

如果有provider，可以使用命令行的方式验证

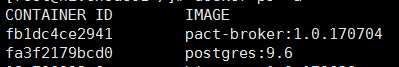
pact-verifier --provider-base-url=http://localhost:8080 --pact-urls=./pacts

Step 3 Pact Broker

安装有两种方式：

一种是linux安装ruby后，用gem install命令去安装 Pact Broker，没有尝试；

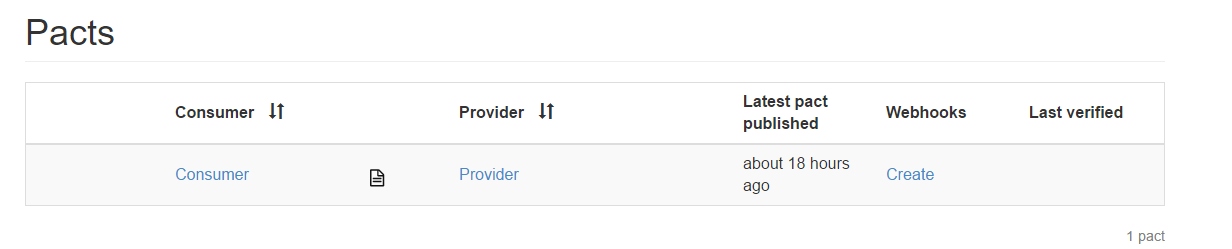
另外一种是docker，需要postgres和pact-broker，已经安装成功，运行OK

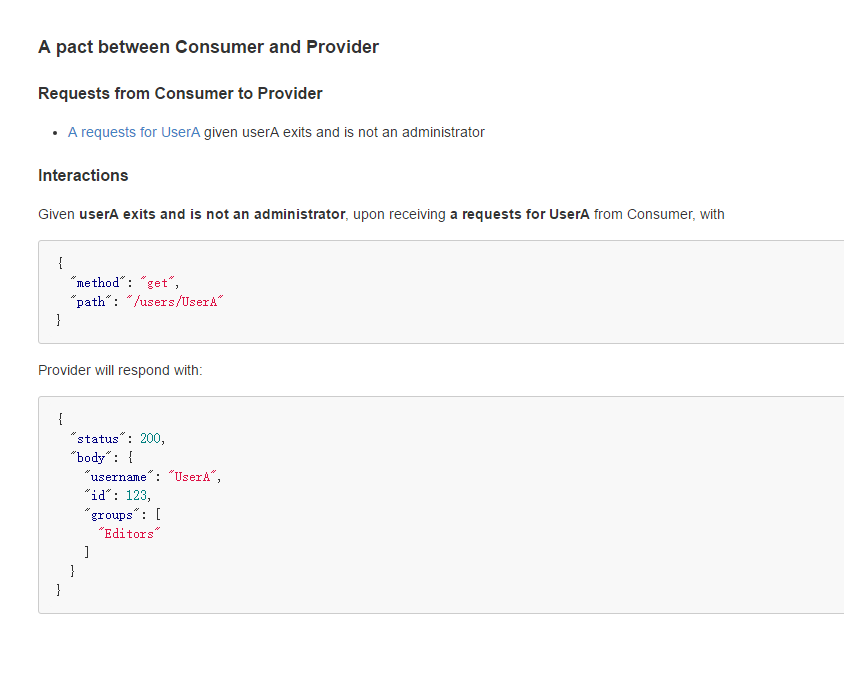


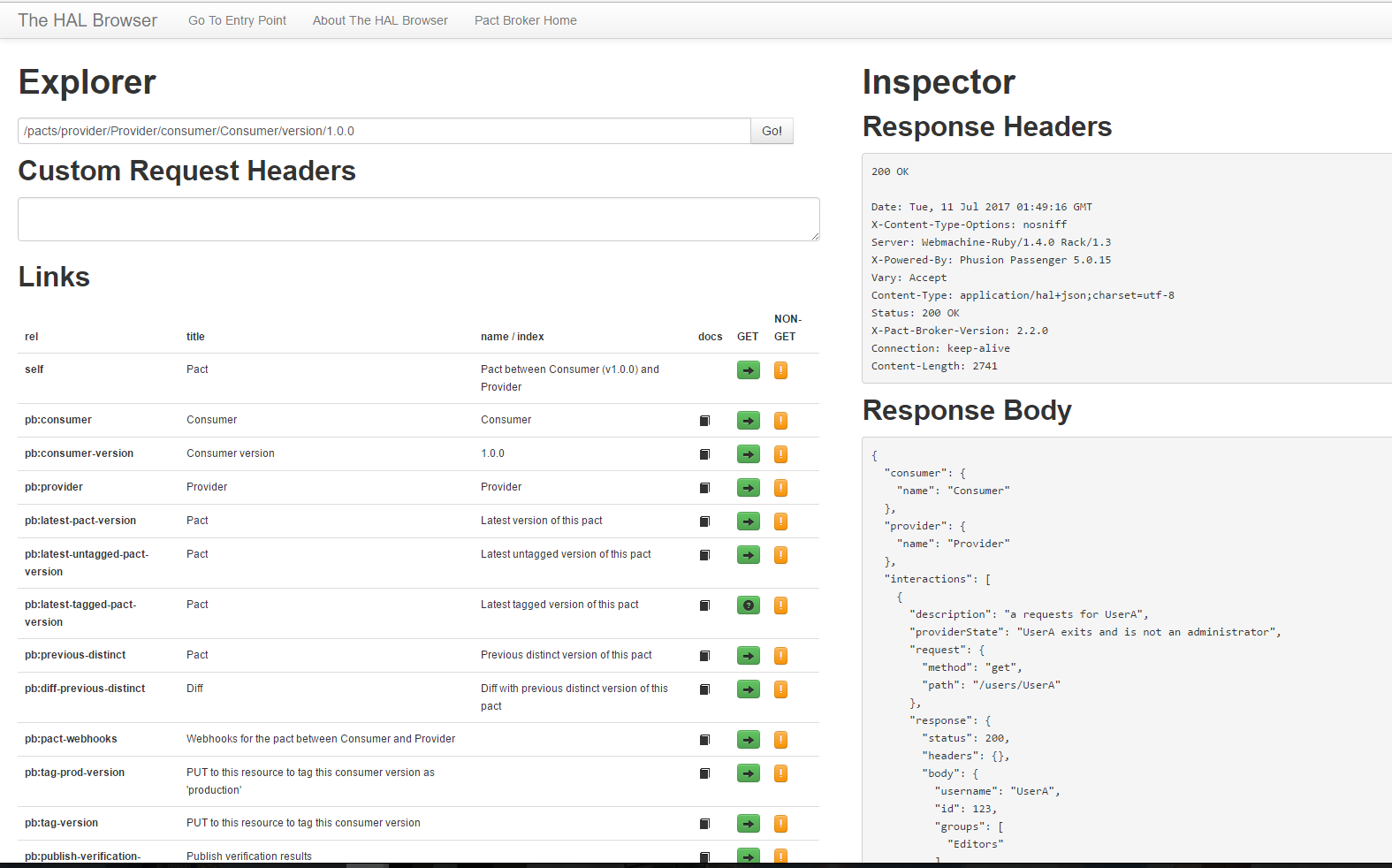
访问方式

<http://172.16.131.23:1222/>

界面，可视化consumer和provider，还提供The HAL Browser方式浏览（还没搞懂），







**测试结论：**

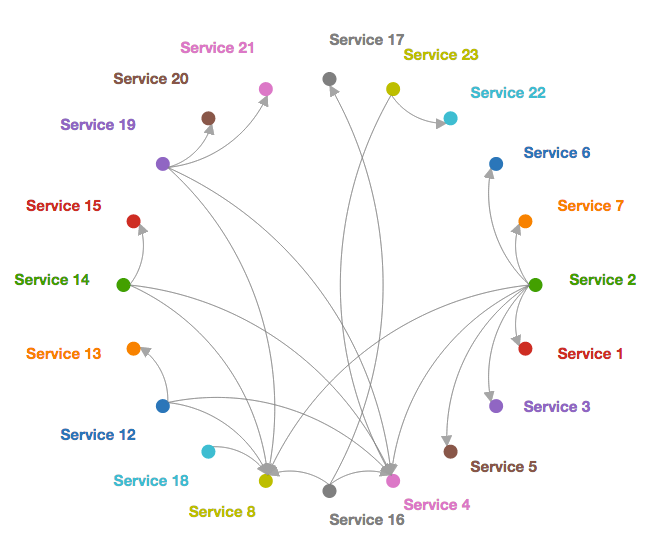
**理解：**

契约测试，更像是微服务的单元测试，提供了mock service。使用场景最好是两个开发人员分别开发一个微服务，相互之间约定接口，生成契约文件，然后并行开发，最后集成测试。适合开发微服务之前引入。

**优点：**

使用契约测试可并行开发，将所有契约文件Put上传到Pact Broker，

可以自动生成Network diagram，可视化微服务之间的关系，看起来高大上，如下图

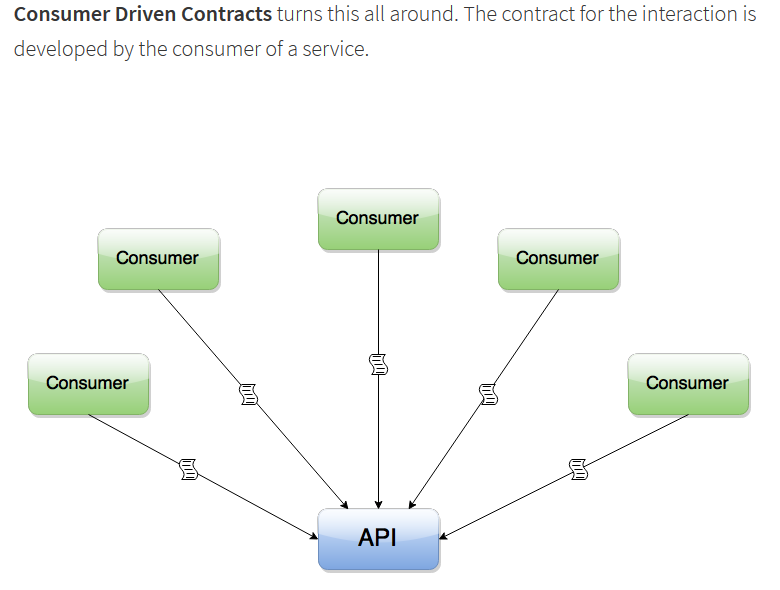


**个人观点：**

站在开发的角度（当然我不是），我认为最有用的是这个：

对于一个API，可以晓得和哪些consumer有关系，一旦我的API改变，我可以知道影响 范围，如果一个API对应几十个consumer，这应该很有用，对于重构。更多优点如下：

The benefits to this approach are that you know who all the consumers of your service are, you’ll know when you make breaking changes so it is easier to make changes and the contract is also a form of documentation. Secondly, you’ll know exactly what functionality needs to be provided so you can implement the right services needed by your consumers and can follow the [YAGNI](https://en.wikipedia.org/wiki/You_aren't_gonna_need_it) principle.



**五大缺点及What is it not good for：**

1. 契约测试，更像是微服务的单元测试，建议由开发人员来生成契约文件，但是这样会为开发带来工作量（首先，学习成本，其次，维护成本，接口的改变都需要新生成契约文件）
2. 资料全英文，中文资料极少
3. 目前只有linux可用，windows不可用，mac不知道，一般是在开发环境生成契约文件，由于我们团队的开发是windows和mac环境，所以不适用。
4. PACT是Ruby开发的东西，github上 的ruby版4年前就有了，只有34个贡献者。

就技术上来说，

Ruby：应该是可用的，不然这个没法出名，几乎是CDC的代表了，但是我们团队没人会ruby；

Java版：支持JVM，没用过

JS版：有一篇中文博客是用js做契约测试，比较详细，nodejs的，应该可用。

Python版：发展缓慢，只有7个贡献者，只是能生成pact文件

.net版：支持，没用过

1. 每次API变更，都需要重新生成pact，不方便

## What is it not good for?

* Testing new or existing providers where the functionality is not being driven by the needs of the consumer (eg. public APIs)
* Testing providers where the consumer and provider teams do not have good communication channels.
* Performance and load testing.
* Functional testing of the provider - that is what the provider's own tests should do. Pact is about checking the contents and format of requests and responses.
* Situations where you cannot load data into the provider without using the API that you're actually testing (eg. public APIs). [Why?](https://github.com/realestate-com-au/pact/wiki/Why-Pact-may-not-be-the-best-tool-for-testing-public-APIs)
* Testing "pass through" APIs, where the provider merely passes on the request contents to a downstream service without validating them. [Why?](https://github.com/realestate-com-au/pact/wiki/Why-Pact-may-not-be-the-best-tool-for-testing-pass-through-APIs)