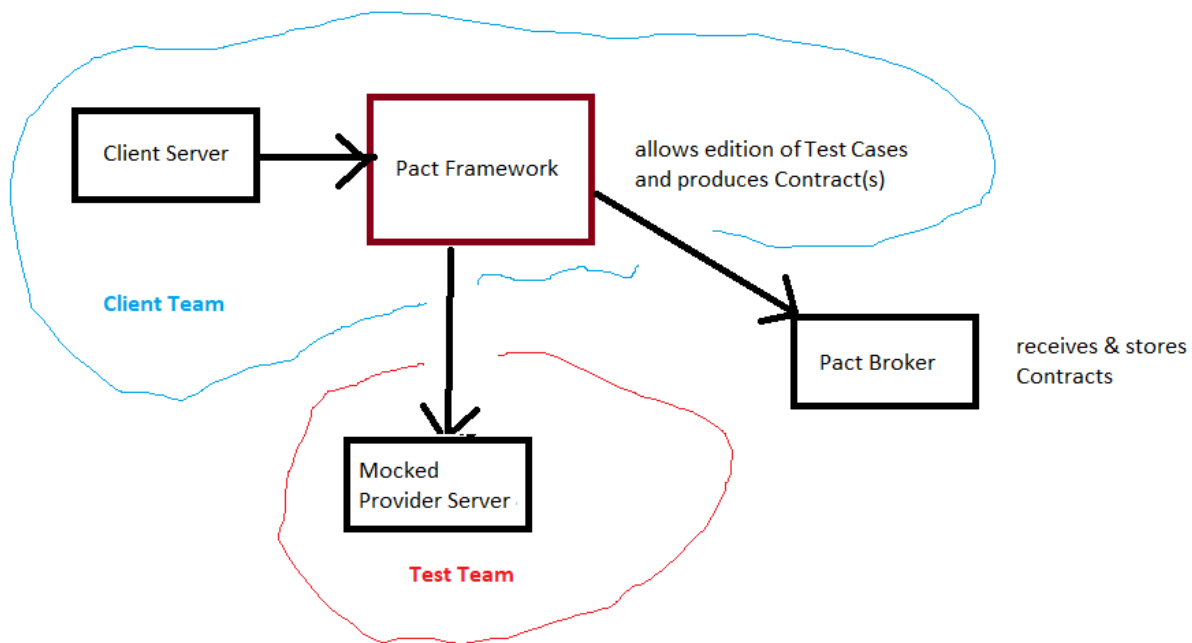


Client Driven Contract Testing with Pact Broker

Consumer-Driven Contract (CDC)

Examples with OpenAI and Copilot



In the [previous articles](#) the focus was given to adhering to Client Driven Contract Testing in the absence of contributions from Client DevOps team. Such approach is classified as Bi-Directional Contract Testing by SmartBear's [Pactflow.io](#). However, the main idea of Client Driven is that the team responsible for the Client server is vividly interested in driving the development.

In the nominal SDLC for Client Driven approach the Client server is operational before the Provider server is. To verify not only the contract itself but the wiring between participants the duty to provide the mocked Provider server, at minimum its stub, or the best its spy falls on the shoulders of the Test Team. Is it within Azure Service Fabric, AWS App Mesh, Google Istio, IBM Cloud App Connect Enterprise, Oracle API Gateway, Alibaba Cloud Service Mesh (ASM), VMware Tanzu Service Mesh, or any other Cloud service where Containers can live the communication between them might be subject to unexpected effects therefore the thorough Contract testing should expand beyond verification of the adherence to the letter of the Contract.

On Azure Cloud [WireMock](#) can be deployed as an Azure App Service or as a Docker container. Wiring to the Client and traffic observability will be impacted by the above decision. Azure App Service provides Application Insights and Log Streaming while Docker option provides Docker Stats and Docker Logs.

Let us concentrate on Contract. In the [previous articles](#) the Test Team was in charge of all activities related to the Client. The Test team relied heavily on specifications provided by the integration focused Business Analyst. In current situation Test team relies on Client Teams Business Analyst and is focused on building mocked Provider that would satisfy the contracts. While mocked Provider is a disposable service its correctness and completeness go in hand with supporting Client team in producing a complete set of Contracts worth enforcing in later stages of testing.

Manually writing or generating a Pact file from something like a Swagger document would be like marking your own exam, and would do nothing to ensure that the code in the consumer and provider are compatible with each other.

Something that could be useful, however, is to generate [skeleton](#) Pact test code from a Swagger document.

Pact-Foundation's Pact itself supports two discern worlds:

1. Pact.NET (C#) and
2. Other languages (C, Node, PHP, Proto, Python, Ruby, Rust)

Support of Pact newest versions, V3 and V4 might give certain indication where the industry is going:

Implementation v1 v1.1 v2 v3 v4

Pact-rust	□ □
Pact-jvm	0 □ □
Pact-ruby	□
Pact-js	□ □ □ *
Pact-net	□ □ □ *
Pact-python	□
Pact-php	□
Pact-go	□ □ □ *
Pact-C++	□ □
scala (scala-pact)	□
scala (pact4s)	□ □ *

Simple implementation diagrams

RUST (2006, a multi-paradigm, general-purpose programming language that emphasizes performance, type safety, and concurrency). Supported by [WireMock](#).

JVM (1994, Java 8, 2014 lambda expressions, streams, and default methods). JVM 2.22.2 supports [ed by WireMock](#)

RUBY (1993, Ruby 3.3.0 2023) Note that, because Pact is written in Ruby the support for V3 and V4 are present. Requires gem wrapper to work with WireMock.

Multiple provider states (pact **verification**) and Publish pacts to **Pact Broker** are offered for every language (except for C++). However, [Multiple provider states](#) (pact **creation**) is supported only by JVM, JS, Go and Swift.

NODE (2009, [Node.js](#) 21.4 2023). Supported by [MockWire](#).

.NET (2002, [.NET 8](#), 2023). PactNet is a .NET implementation of the Pact contract testing library. Supported by [WireMock](#). The latest version of WireMock.NET is 1.5.46.

Python (1991, [Python 3.x](#) 2023). Supported by [WireMock](#).

Golang (2007, [Golang](#) 1.21 2023). Suported by [WireMock](#).

Scala (2001, [Scala 2.12.13](#) 2019) Supported by [WireMock](#).

Ready to use Container Images:

[pactfoundation/pact-cli](#)

[pactfoundation/pact-broker](#)

[pactfoundation/pact-ref-verifier](#)

[pactfoundation/pact-ref-mock-server](#)

[pactfoundation/pact-mock-service](#)

[pactflow/demo-consumer-node](#)

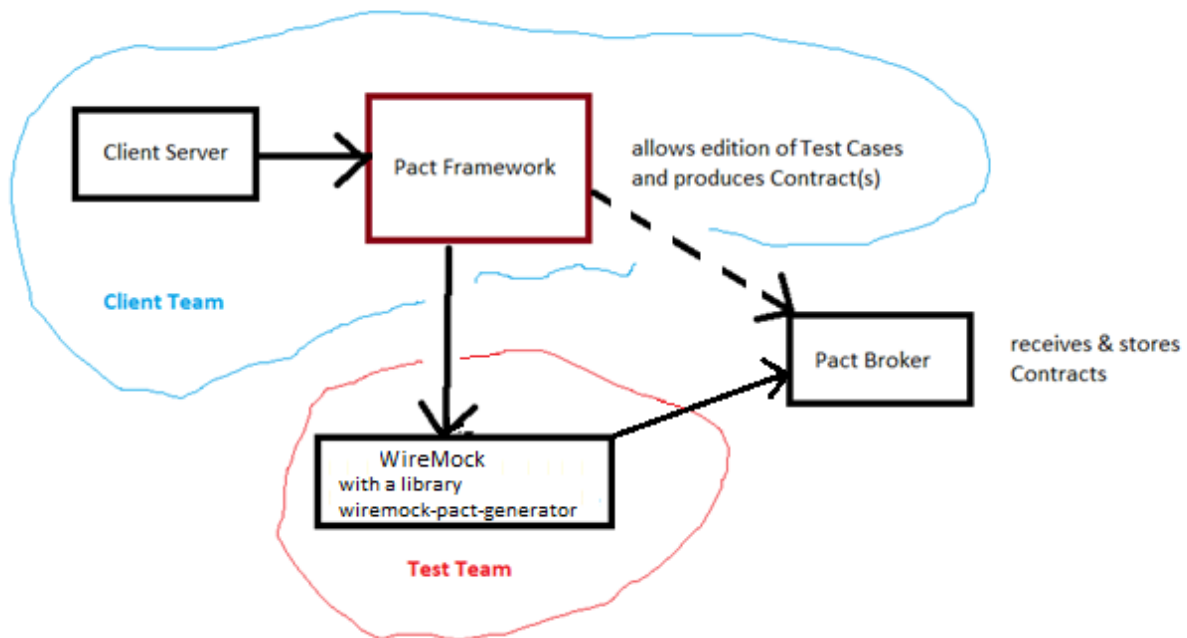
[wiremock/wiremock](#)

[sheyenrath/wiremock.net](#)

[sheyenrath/wiremock.net-nano](#)

[selenium/node-chrome](#)

[mbrzozow/pactflow-swagger-mock-validator](#) [Doc](#)



Note that [Pacts](#) are language neutral. However to run the library you need [Java 1.8](#).

For example:

```
{
  "consumer": {
    "name": "your-consumer-name"
  },
  "provider": {
    "name": "your-provider-name"
  },
  "interactions": [
    {
      "description": "your-description",
      "request": {
        "method": "your-method",
        "path": "your-path",
        "headers": {
          "your-header": "your-header-value"
        },
        "body": {
          "your-body": "your-body-value"
        }
      },
      "response": {
        "status": 200,
        "headers": {
          "your-header": "your-header-value"
        },
        "body": {
          "your-body": "your-body-value"
        }
      }
    }
  ]
}
```

```

    }
  ],
  "metadata": {
    "pactSpecification": {
      "version": "3.0.0"
    }
  }
}

```

Note as well, that .NET [MockServer-Client-net](#) was not updated for four years and never included a functionality to generate Pact Contracts.

Note as well that instead of using Pact Framework or WireMock library the BA team can use simply ChatGPT or Copilot to generate contracts. In these situations OpenAI simply works as syntax formatted statement corrector, and the content needs to come from Client's team BA.

Here is AI example:

Question:

Generate Pact Contract for request that contains only name and response that contains length, width, weight, humidity and price, a distance to North Pole from my current location and taysys exchange rate CHF to CDN.

ChatGPT feeds back Node.js code:

ChatGPT:

```

provider.addInteraction({
  state: 'provider accepts a request with name and returns dimensions, price, distance to North Pole,
and CHF to CDN exchange rate',

  uponReceiving: 'a request for dimensions, price, distance to North Pole, and CHF to CDN exchange
rate',

  withRequest: {
    method: 'POST',
    path: '/data',
    headers: {
      'Content-Type': 'application/json',
    },
    body: {
      name: like('John Doe'), // Example of a 'name' field in the request
    },
  },
  willRespondWith: {

```

```

status: 200,
headers: {
  'Content-Type': 'application/json',
},
body: {
  length: like(10), // Example of 'length' in the response
  width: like(5), // Example of 'width' in the response
  weight: like(15), // Example of 'weight' in the response
  humidity: like('20%'), // Example of 'humidity' in the response
  price: like(100), // Example of 'price' in the response
  distanceToNorthPole: like('5000 km'), // Example of 'distanceToNorthPole' in the response
  CHFToCDNExchangeRate: like(0.9), // Example of 'CHFToCDNExchangeRate' in the response
},
},
});

```

provider.finalize();

But, catches up when prompted:

Question:

Could you please express it as JSON formatted contract?

ChatGPT

```

{
  "consumer": {
    "name": "Consumer"
  },
  "provider": {
    "name": "Provider"
  },
  "interactions": [
    {
      "description": "provider accepts a request with name and returns dimensions, price, distance to North Pole, and CHF to CDN exchange rate",
      "providerState": "provider accepts a request with name and returns dimensions, price, distance to North Pole, and CHF to CDN exchange rate",
      "request": {
        "method": "POST",
        "path": "/data",
        "headers": {
          "Content-Type": "application/json"
        },
        "body": {
          "name": "John Doe"
        }
      }
    }
  ]
}

```

```

    }
  },
  "response": {
    "status": 200,
    "headers": {
      "Content-Type": "application/json"
    },
    "body": {
      "length": 10,
      "width": 5,
      "weight": 15,
      "humidity": "20%",
      "price": 100,
      "distanceToNorthPole": "5000 km",
      "CHFToCDNExchangeRate": 0.9
    }
  }
},
"metadata": {
  "pactSpecification": {
    "version": "3.0.0"
  }
}
}

```

On the other hand, Edge's Copilot generates immediately JSON file formatted Pact Contract. However it singles out a distance to north Pole and exchange rate between Swiss Francs and Canadian Dollars and presents them as separate items not included in the JSON file.

Copilot:

```

{
  "request": {
    "method": "GET",
    "path": "/",
    "query": {
      "name": "string"
    }
  },
  "response": {
    "status": 200,
    "headers": {
      "Content-Type": "application/json"
    },
    "body": {
      "length": "string",
      "width": "string",
      "weight": "string",
      "humidity": "string",
      "price": "string"
    }
  }
}

```

```
}  
}  
}
```

To get the distance from your current location to the North Pole, I need access to your location data. Please enable location access in your browser settings and try again.

The exchange rate of Swiss Franc (CHF) to Canadian Dollar (CAD) is 1 CHF = 1.57 CAD. Please note that exchange rates are subject to change and the rate mentioned here is for informational purposes only.

Therefore the author tweaks the question to Copilot:

Question

Generate Pact Contract for request that contains only name and response that contains length, width, weight, humidity and price, as well as an approximate distance to North Pole from Mississauga, Ontario named FarFarAway and with the value you will find and include in JSON as well today's exchange rate from CHF to CDN.

Copilot, however insists on getting details it does not need and dodges inclusion of the last two items in JSON file:

Copilot:

To get the distance from Mississauga, Ontario to the North Pole, I need access to your location data. Please enable location access in your browser settings and try again.

The exchange rate of Swiss Franc (CHF) to Canadian Dollar (CAD) as of December 31, 2023 is 1 CHF = 1.57 CAD. Please note that exchange rates are subject to change and the rate mentioned here is for informational purposes only.

In the above round ChatGPT demonstrated superiority over Copilot by including 'Provider State' and 'Description' in generated Pact Contracts.

So, on basis of the above real life examples, my conclusion here is: AI is still a few miles before any project could rely on it. However, I would not hesitate to encourage Business Analysts to use ChatGPT to vet their English while writing specifications. The formalism of Gherkin with GIVEN, WHEN, THEN provides some incentives to gather thoughts but does NOT guarantee the clarity of specification. OpenAI is still too immature to rely on it but is more likely to challenge one's English in a constructive way and its usage will grow if AI continues advance to next maturity levels.