Contract Testing with Pact & Pactflow

By Manassarn Manoonchai (Noom)

What is Contract Testing?

Contract testing

- A technique for testing an integration point by checking each application in isolation to ensure the messages it sends or receives conforms to a shared understanding that is documented in a “contract”.
- A form of testing, where you test the "contract" between the services/systems communicating to each other.

The Consumer & The Provider

Contract will have 2 sides : Consumer & Provider

- Consumer is the one who consume data e.g. API Client, MQ Subscriber
- Provider is the one who provide data e.g. API Server, MQ Publisher/Producer

The "Contract"

- A specification of what data Consumer request and get from the Provider
  - e.g. `` `When consumer calls GET /api/promotions, the Provider would return JSON with array of promotions` ``
- The contract looks just like how we mock HTTP request with `` `Nock` ``, then what's the difference?

```
nock('http://www.example.com')
  .get('/api/promotions')
  .reply(200, { promotions: [{ ... }] })
```

Contract vs Mocking

- When mocking requests, we tend to use it for 3rd party APIs, we don't care how the endpoint will behave if we send data like query strings, since we don't have any control of them.
-
- But since we are building both of the services calling each other (Consumer & Provider) the Contract works on both sides:
  - On consumer side, the contract emulates the provider (i.e. as a HTTP mock server) which receives request and returns data
  - On provider side, the contract emulates the consumer (i.e. cURL, Postman) which make the request and verify the returning data

Who defines the contract?

- Provider -> "Provider-driven Contract Testing"
  - Pros : Good when we have many consumers or consumers don’t understand the server and/or provide meaningful contracts
  - Cons : The workflow will be one-way, and longer or no feedback loop from consumer will happen (https://pactflow.io/blog/the-curious-case-for-the-provider-driven-contract)
- Consumer -> "Consumer-driven Contract Testing"
  - Faster feedback loop
  - More practical
  - https://pact.io

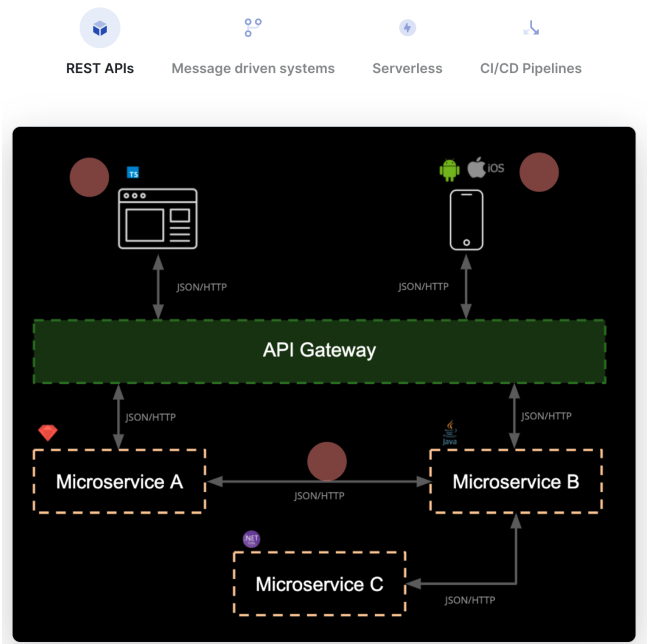
Integration testing done properly

Fast, easy and reliable testing for integrating web apps, APIs and microservices

View on Github ▾

Pact

- Multi-language support (JS, Ruby, Python, Go, etc.)
- Supports most integrations
- Supports CI/CD with Pactflow or self-hosted



Usage

- Start from consumer side (Consumer-driven)
- Provider verifies the contract

Consumer

Write unit/integration tests with Pact mocks

1. Setup Pact (It will create mocked Provider)

```
const url = "http://localhost"
let dogService: DogService

const provider = new Pact({
  // port,
  log: path.resolve(process.cwd(), "logs", "mockserver-integration.log"),
  dir: path.resolve(process.cwd(), "pacts"),
  spec: 2,
  consumer: "Typescript Consumer Example",
  provider: "Typescript Provider Example",
})

const dogExample = { dog: 1 }
const EXPECTED_BODY = eachLike(dogExample)

before(() =>
  provider.setup().then(opts => {
    dogService = new DogService({ url, port: opts.port })
  })
)
```

Consumer

Write unit/integration tests with Pact mocks

2. Setup mock request & response with `Interaction`

```
describe("get /dogs using builder pattern", () => {
  before(() => {
    const interaction = new Interaction()
      .given("I have a list of dogs")
      .uponReceiving("a request for all dogs with the builder pattern")
      .withRequest({
        method: "GET",
        path: "/dogs",
        headers: {
          Accept: "application/json",
        },
      })
      .willRespondWith({
        status: 200,
        headers: {
          "Content-Type": "application/json",
        },
        body: EXPECTED_BODY,
      })

    return provider.addInteraction(interaction)
  })
```

Consumer

Write unit/integration tests with Pact mocks

3. Write tests as usual

```
it("returns the correct response", done => {
  dogService.getMeDogs().then((response: any) => {
    expect(response.data[0]).to.deep.eq(dogExample)
    done()
  }, done)
})
```

Consumer

Run the contract test

- When the test runs, it will create a contract as JSON file

Consumer

Share the contract

- Then we share the contract with the provider to use
  - Send the file directly
  - Publish to the "Broker" `pact_broker` (Free), Pactflow (Paid)

Provider

Provider

Get the contract

- Setup Pact to fetch the contract from the file, or the broker

Provider

- Start the provider test server, then verify the contract

```
const { Verifier } = require("@pact-foundation/pact")
const { versionFromGitTag } = require("@pact-foundation/absolute-version")
const chai = require("chai")
const chaiAsPromised = require("chai-as-promised")
chai.use(chaiAsPromised)
const { server, importData, animalRepository } = require("../provider.js")
const path = require("path")

server.listen(8081, () => {
  importData()
  console.log("Animal Profile Service listening on http://localhost:8081")
})
```

Provider

- Start the provider test server, then verify the contract

```
return new Verifier(opts).verifyProvider().then(output => {
  console.log("Pact Verification Complete!")
  console.log(output)
})
```

Provider

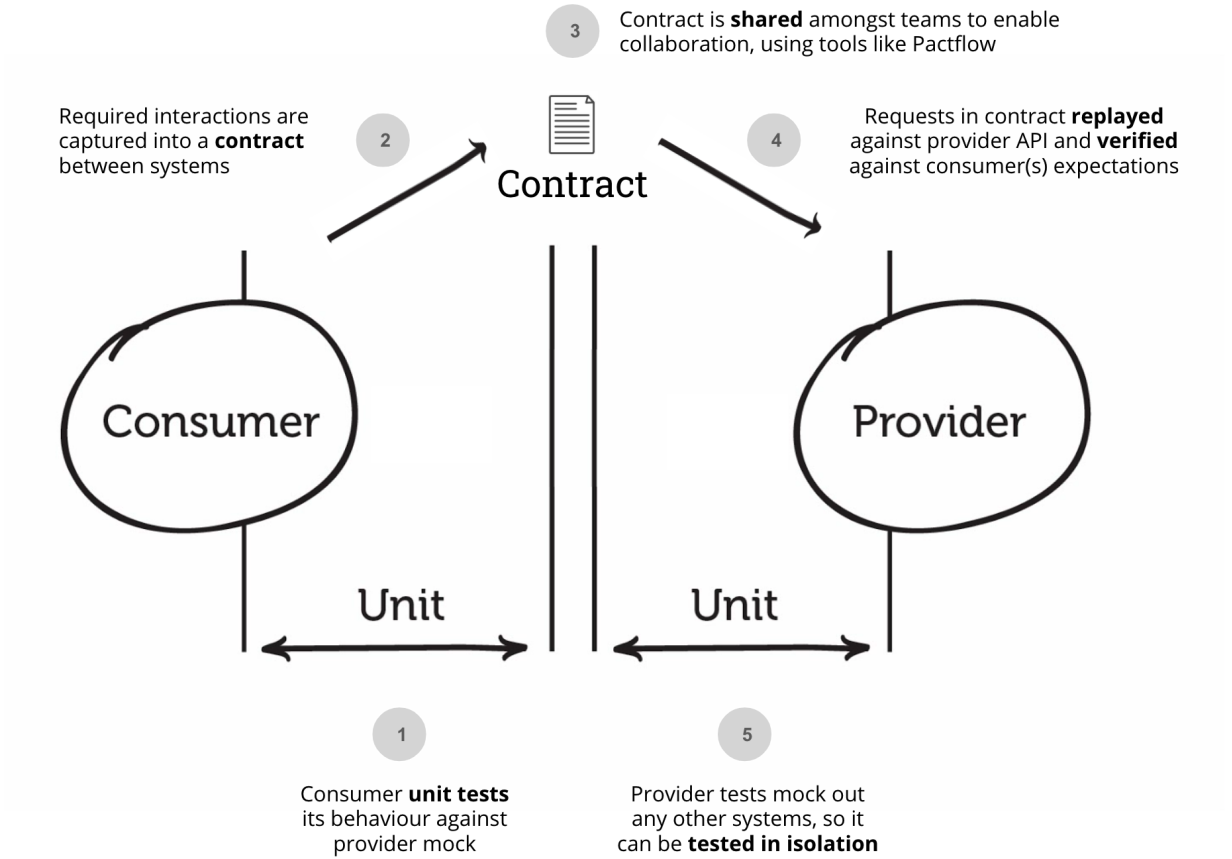
- If using Pactflow, it will support detecting the branch, version and the contract result
  -

Provider

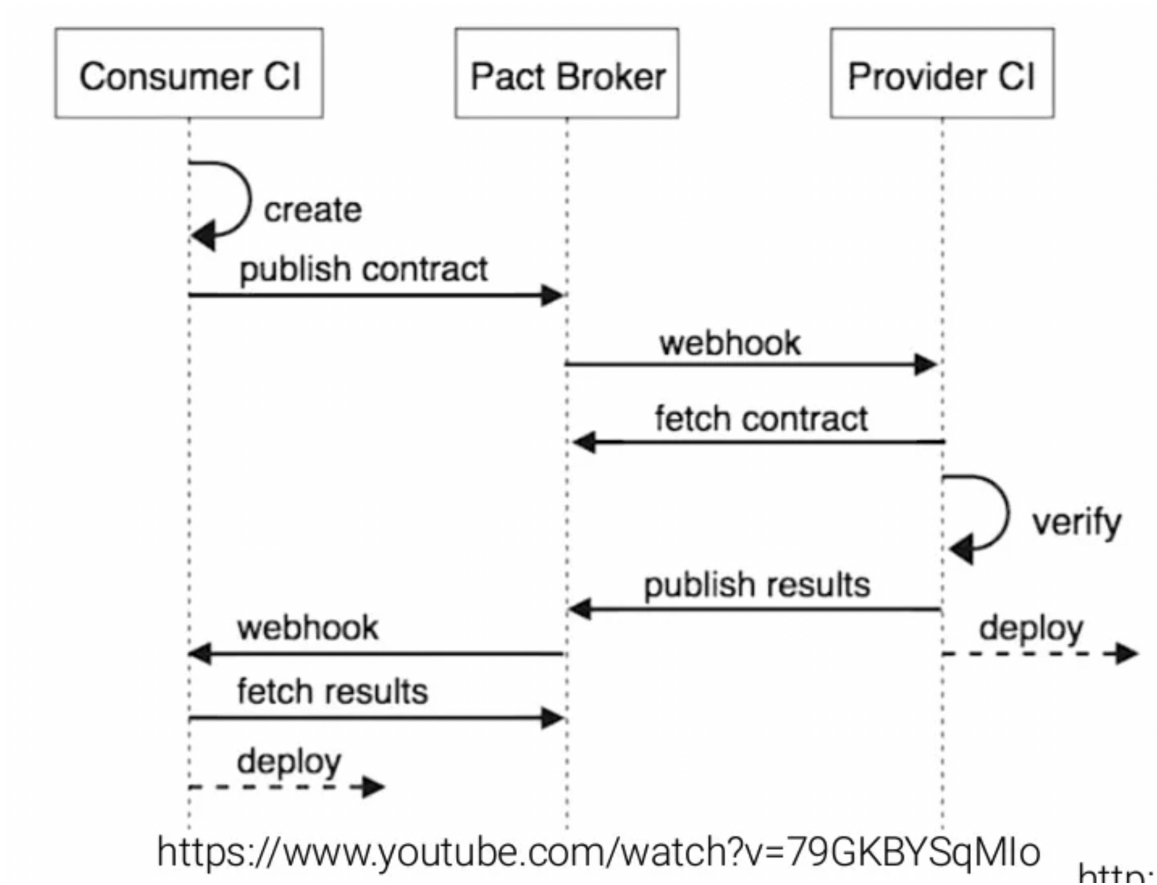
- `Success` means that the contract test on Consumer version X and Provider version Y is passed and is safe to be deployed
- Use `can-i-deploy` CLI tool on CI to check the status against consumer & provider version
  - Here is how we would check to see if we were safe to deploy Foo version 23 to production:

```
$ pact-broker can-i-deploy --pacticipant Foo --version 23 --to-environment production

Computer says yes \o/

CONSUMER | C.VERSION  | PROVIDER | P.VERSION  | SUCCESS? | RESULT#
---------|------------|----------|------------|----------|--------
Foo      | 23         | Bar      | 56         | true     | 1


VERIFICATION RESULTS
--------------------
1. https://pact-broker/pacts/provider/Bar/consumer/Foo/pact-version/cc4e5ae3c12482c6ffd87c4018090a7a1524c634/


All required verification results are published and successful
```

Workflow


3
Contract is **shared** amongst teams to enable collaboration, using tools like Pactflow
Required interactions are captured into a **contract** between systems
2
Contract
4
Requests in contract **replayed** against provider API and **verified** against consumer(s) expectations
Consumer
Provider
Unit
Unit
1
5
Consumer **unit tests** its behaviour against provider mock
Provider tests mock out any other systems, so it can be **tested in isolation**

Workflow


Consumer CI
Pact Broker
Provider CI
create
publish contract
webhook
fetch contract
verify
publish results
webhook
deploy
fetch results
deploy
https://www.youtube.com/watch?v=79GKBYSqMIo

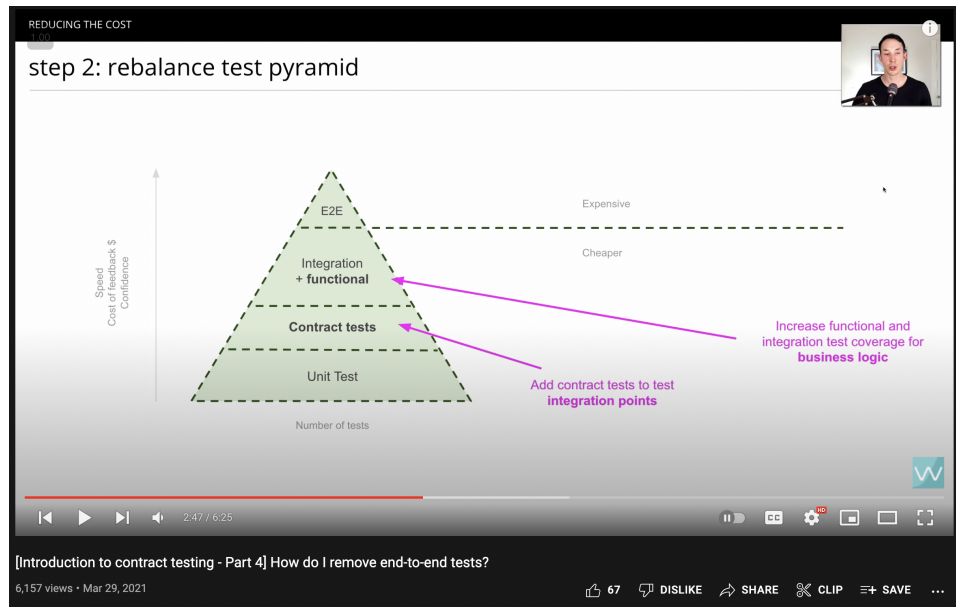
Demo Time!

Summary

- Contract testing
  - Increase confidence
  - Fast
  - Run independently
- Caveats
  - Not "replacing" e2e tests, nor integration tests

References

- https://pactflow.io/how-pact-works
- https://www.slideshare.net/sebrose/contract-testing-and-pact
- https://slides.com/mariedrake/contract-testing-101
- https://www.youtube.com/playlist?list=PLwy9Bnco-IpfZ72VQ7hce8GicVZs7nm0i



Thank you!

Slides made with Slidev