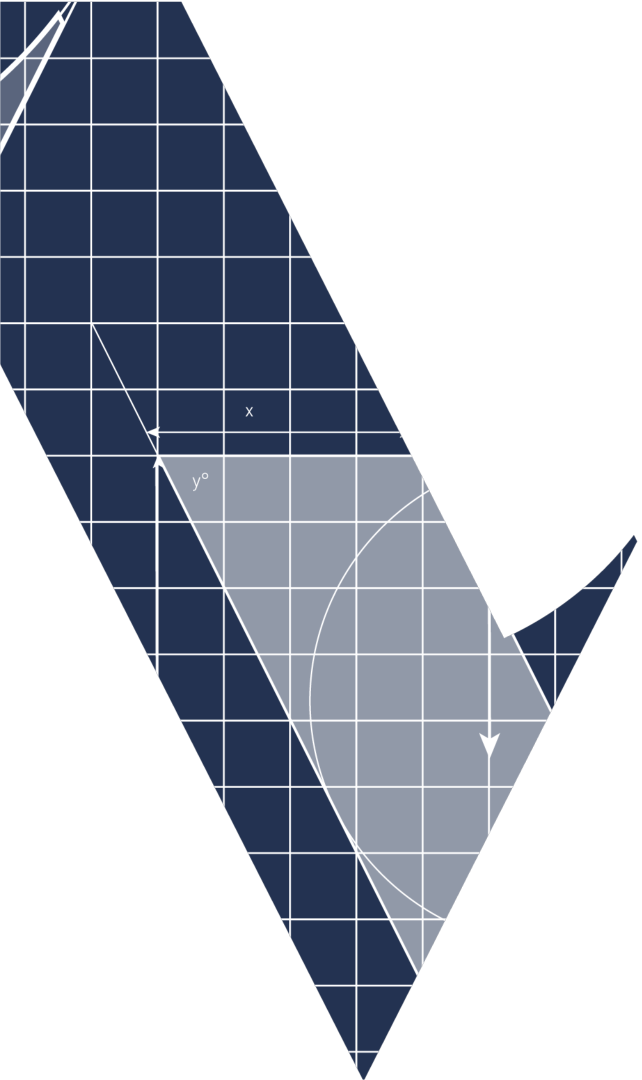


Contract testing with Pact.io

The Workshop solution for integration tests

Who are we?

The Workshop Architecture team

Ignacio Martínez
Senior Architect

Francisco Ramírez
Quality Architect

Agenda

Introduction

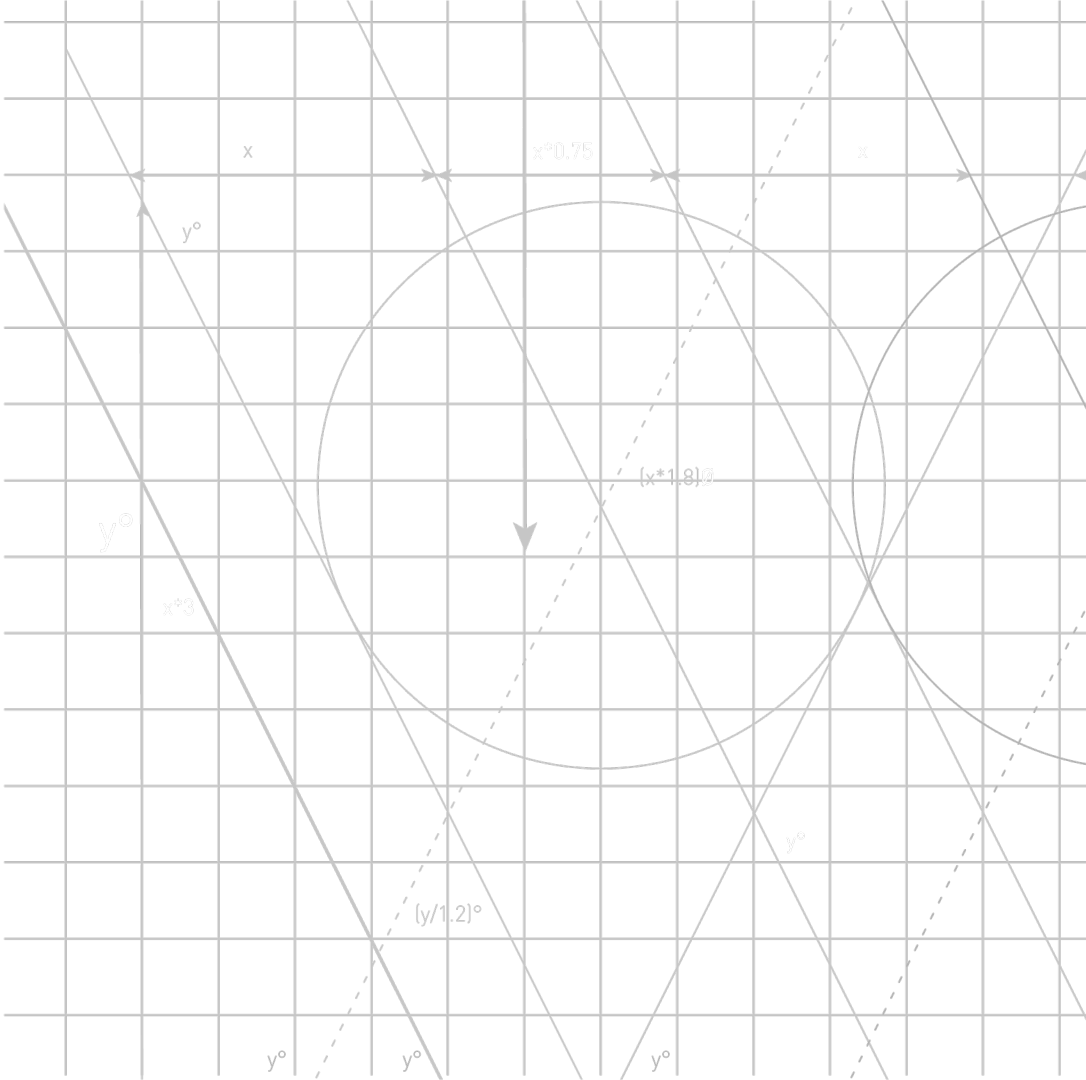
Contract testing

Pact

Exercises

Features in Pact

Takeaways

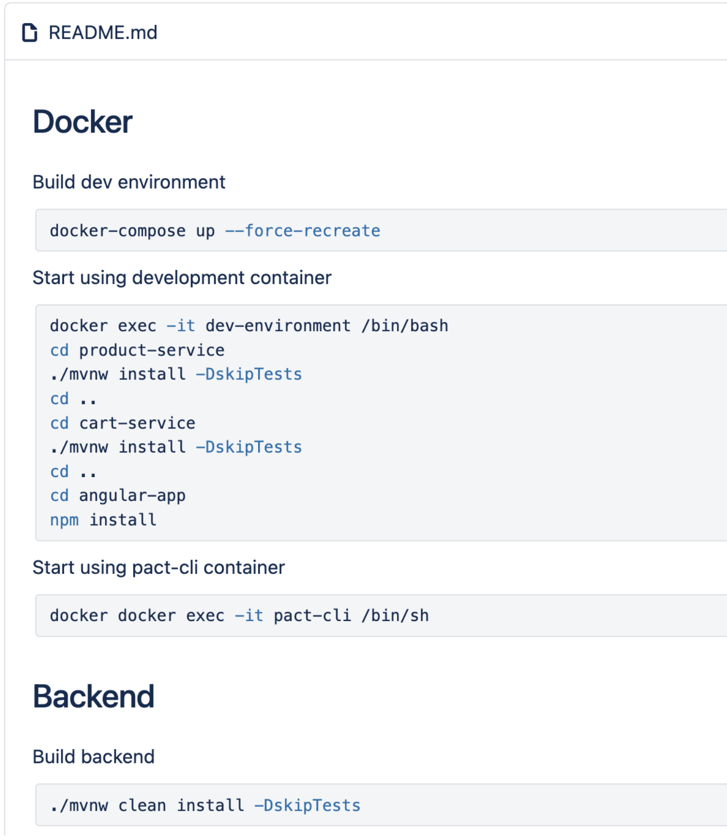


Introduction

Setting the context

Source code

https://github.com/theworkshopcom/pact-workshop



README.md

Docker

Build dev environment

```
docker-compose up --force-recreate
```

Start using development container

```
docker exec -it dev-environment /bin/bash
cd product-service
./mvnw install -DskipTests
cd ..
cd cart-service
./mvnw install -DskipTests
cd ..
cd angular-app
npm install
```

Start using pact-cli container

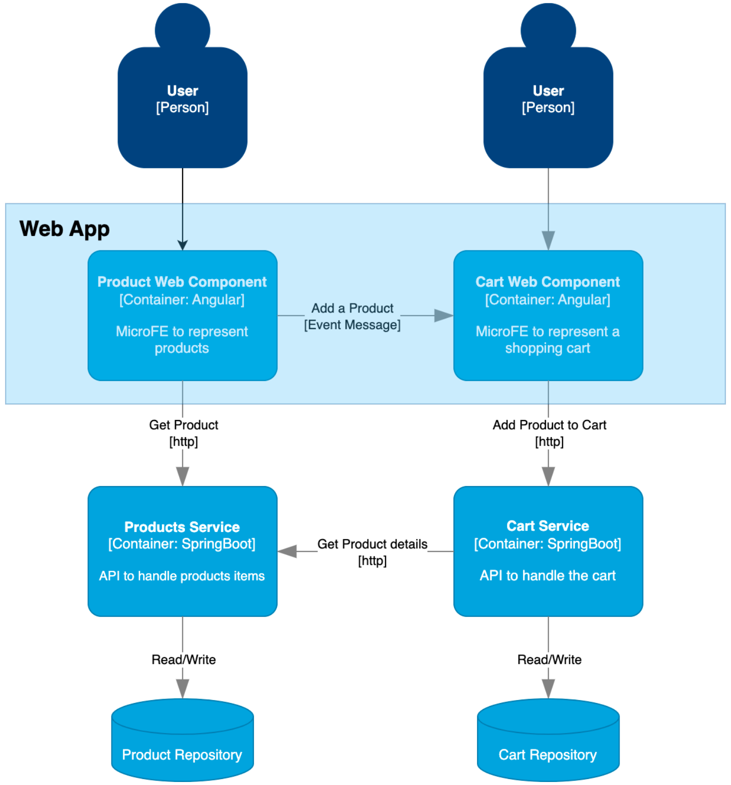
```
docker docker exec -it pact-cli /bin/sh
```

Backend

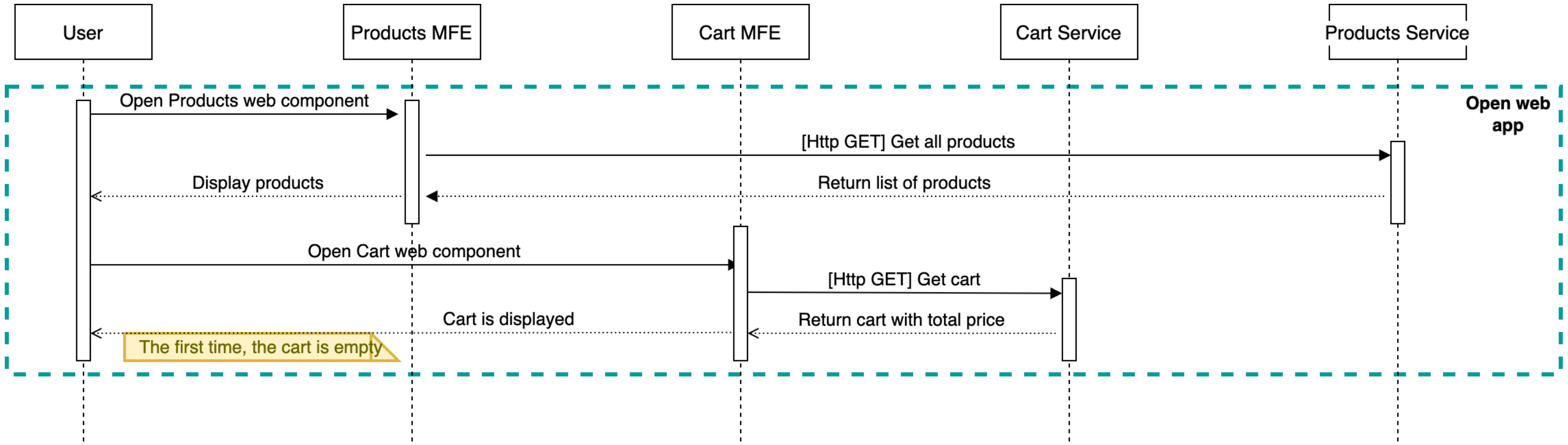
Build backend

```
./mvnw clean install -DskipTests
```

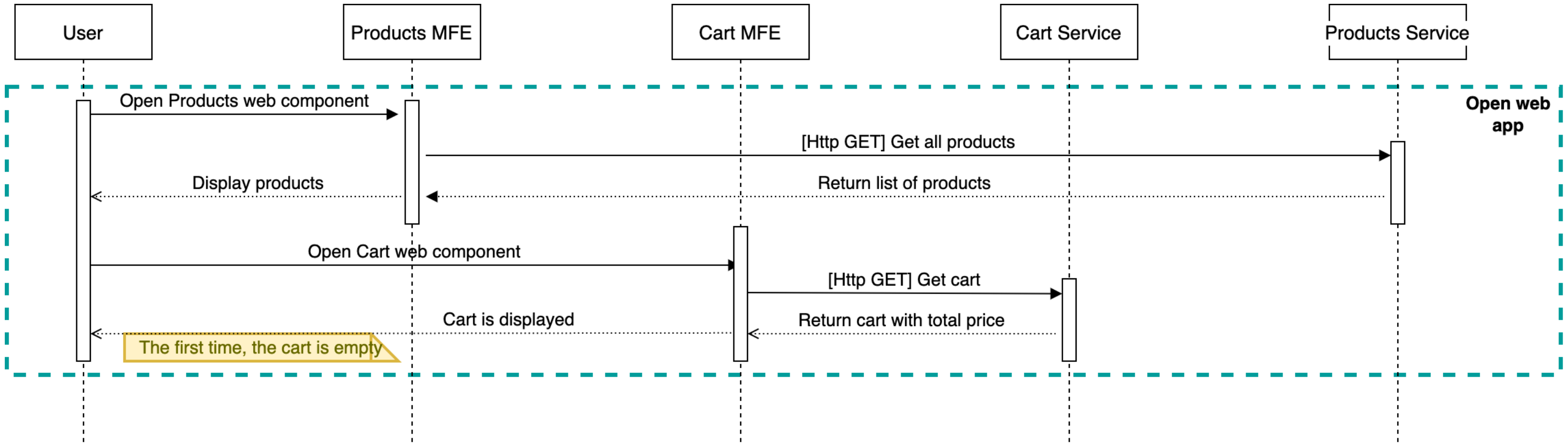
Micro-services architecture



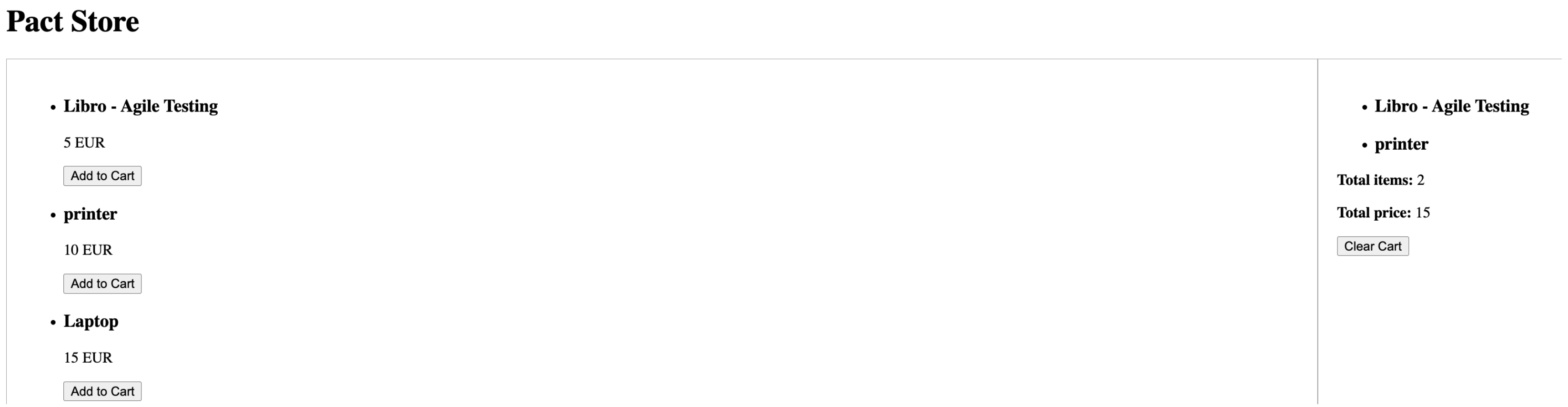
User flows


User
Products MFE
Cart MFE
Cart Service
Products Service
Open web app
Open Products web component
[Http GET] Get all products
Display products
Return list of products
Open Cart web component
[Http GET] Get cart
Cart is displayed
Return cart with total price
The first time, the cart is empty

User flows


User
Products MFE
Cart MFE
Cart Service
Products Service
Open web app
Open Products web component
[Http GET] Get all products
Display products
Return list of products
Open Cart web component
[Http GET] Get cart
Cart is displayed
Return cart with total price
The first time, the cart is empty

User Interface



Pact Store

- **Libro - Agile Testing**

  5 EUR

  Add to Cart

- **printer**

  10 EUR

  Add to Cart

- **Laptop**

  15 EUR

  Add to Cart

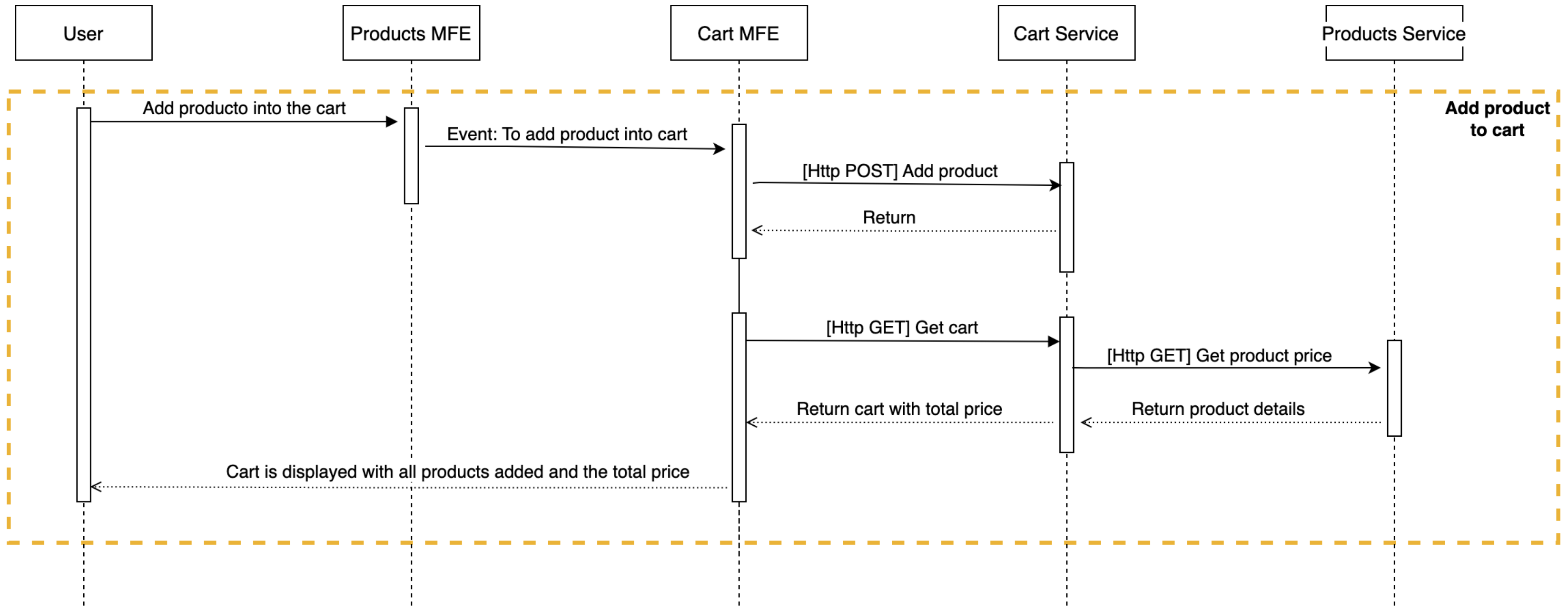
- **Libro - Agile Testing**
- **printer**

Total items: 2

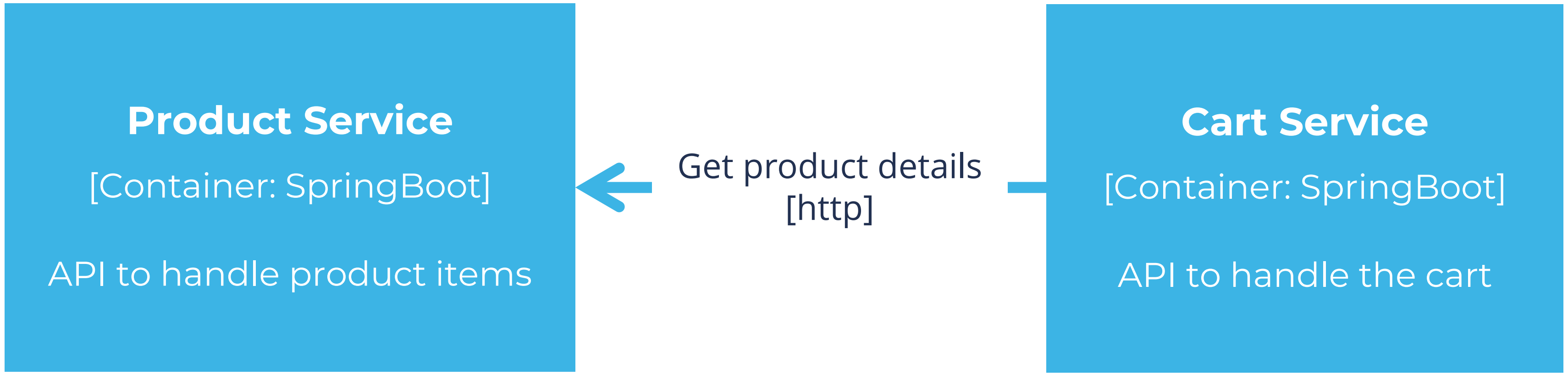
Total price: 15

Clear Cart

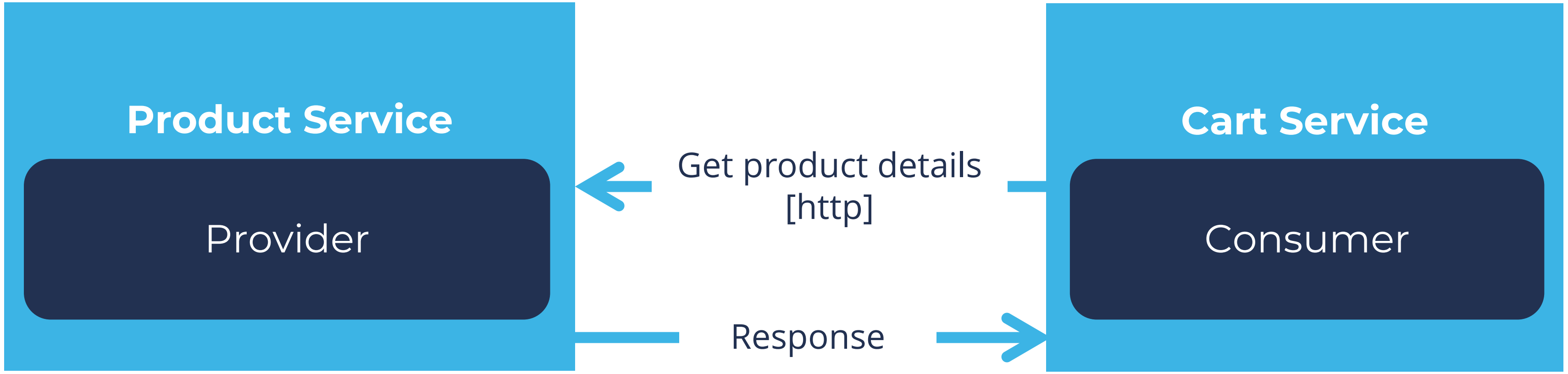
User flows


User
Products MFE
Cart MFE
Cart Service
Products Service
Add product to cart
Add producto into the cart
Event: To add product into cart
[Http POST] Add product
Return
[Http GET] Get cart
[Http GET] Get product price
Return cart with total price
Return product details
Cart is displayed with all products added and the total price

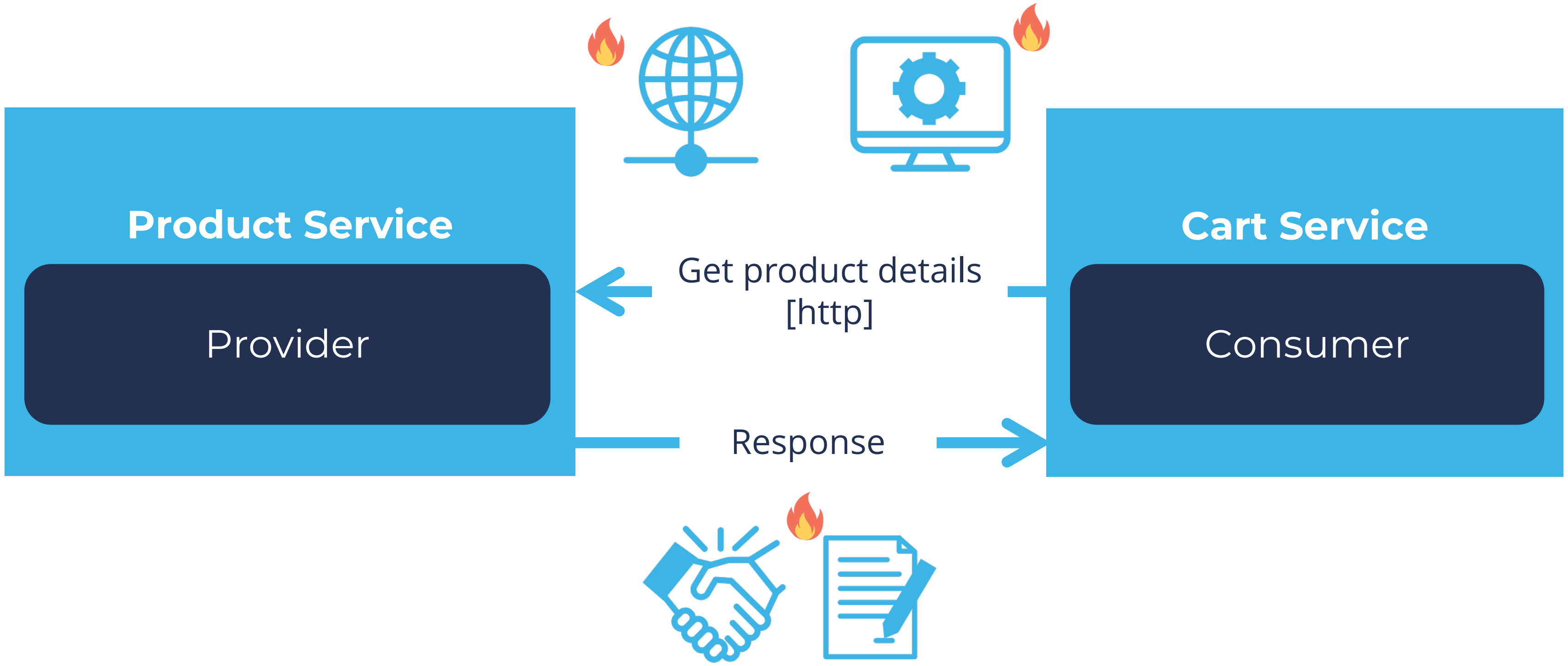
Interaction



Interaction



Interaction



Contracts

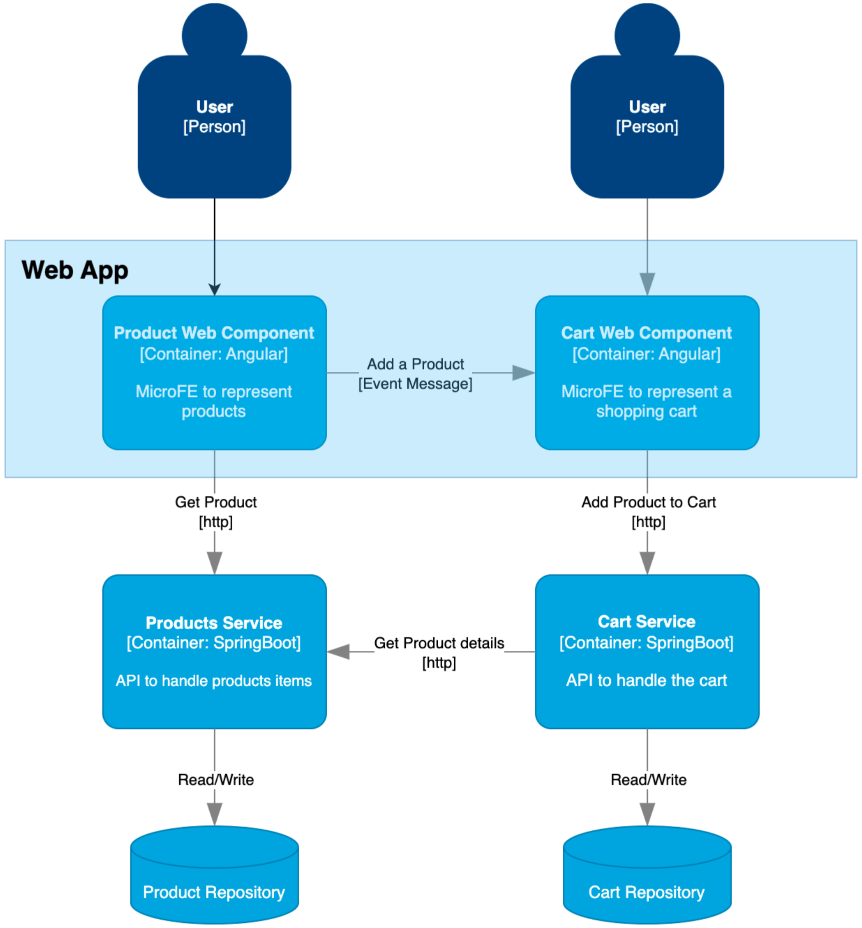
Consumer: *Cart Service*

```
"request": {
  "method": "GET",
  "path": "/api/products/product{id}"
  },
```

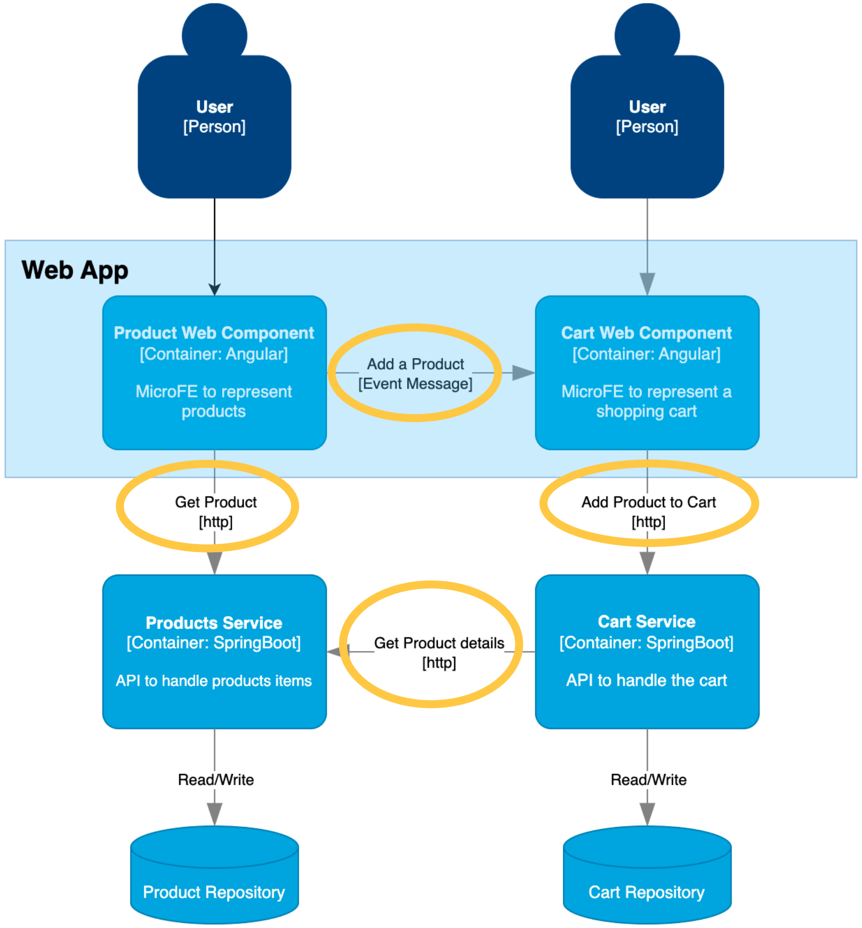
Provider: *Products Service*

```
"response": {
  "status": 200,
  "body": {
    "name": "book",
    "price": 10
  }
```

Contracts

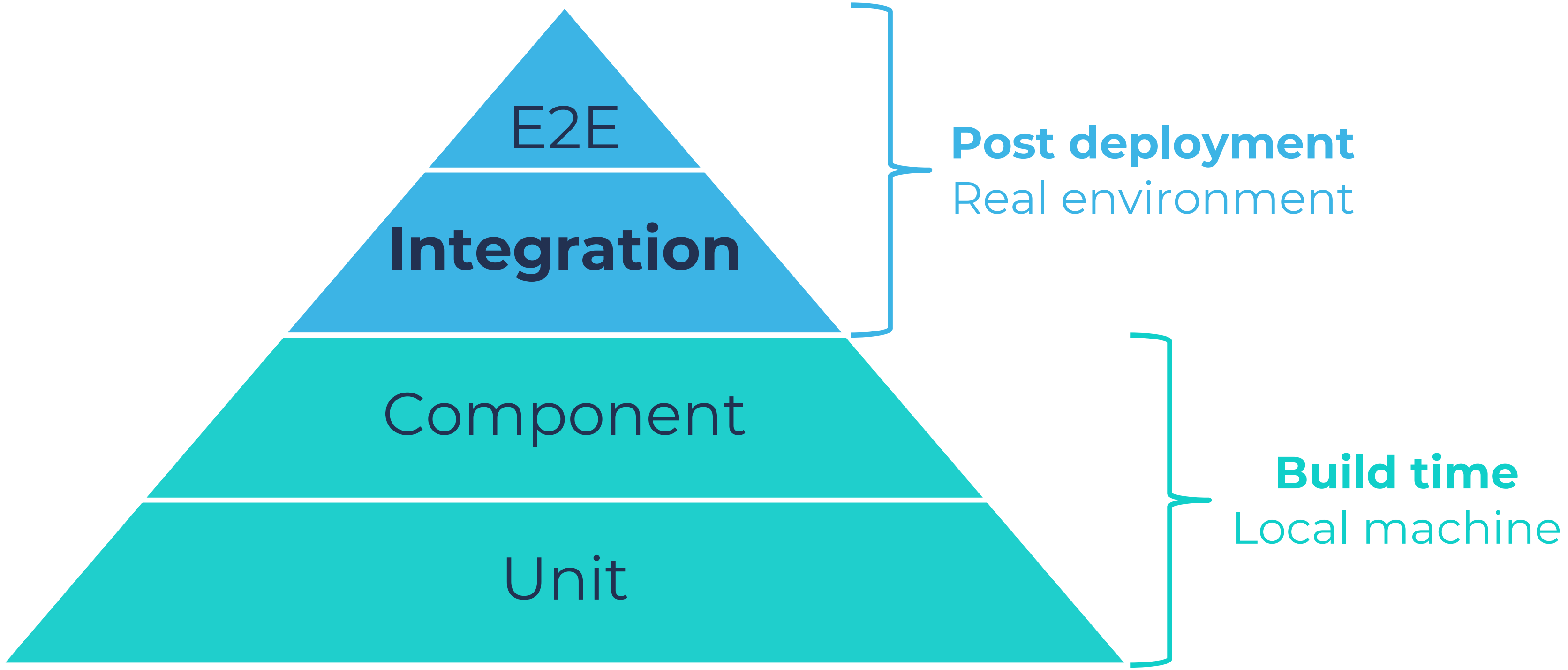

User
[Person]
User
[Person]
Web App
Product Web Component
[Container: Angular]
MicroFE to represent products
Add a Product
[Event Message]
Cart Web Component
[Container: Angular]
MicroFE to represent a shopping cart
Get Product
[http]
Add Product to Cart
[http]
Products Service
[Container: SpringBoot]
API to handle products items
Get Product details
[http]
Cart Service
[Container: SpringBoot]
API to handle the cart
Read/Write
Read/Write
Product Repository
Cart Repository

Contracts

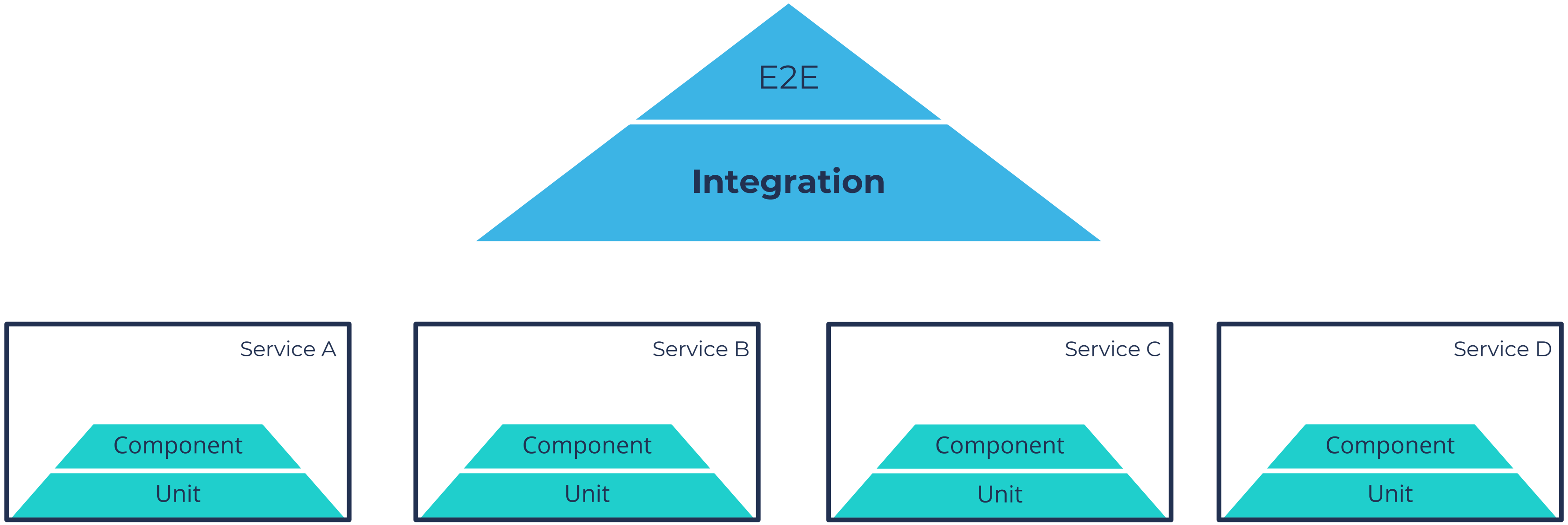

User
[Person]
User
[Person]
Web App
Product Web Component
[Container: Angular]
MicroFE to represent products
Add a Product
[Event Message]
Cart Web Component
[Container: Angular]
MicroFE to represent a shopping cart
Get Product
[http]
Add Product to Cart
[http]
Products Service
[Container: SpringBoot]
API to handle products items
Get Product details
[http]
Cart Service
[Container: SpringBoot]
API to handle the cart
Read/Write
Read/Write
Product Repository
Cart Repository

Testing approach

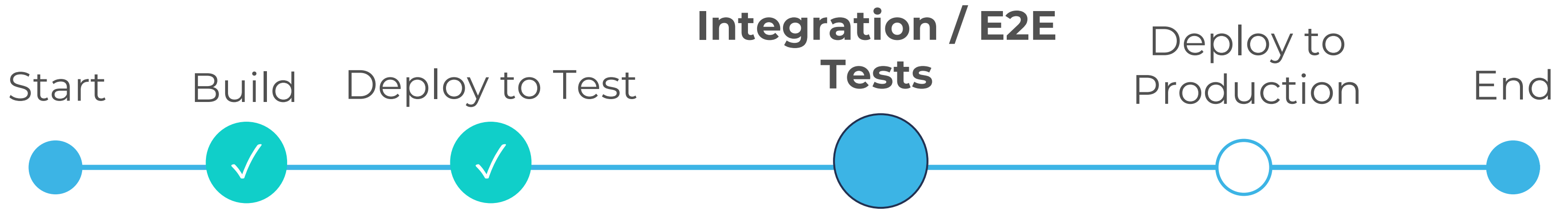
Testing pyramid



Testing pyramid



Testing post-deployment



Integration test

Demo

Integration test

1- System is running

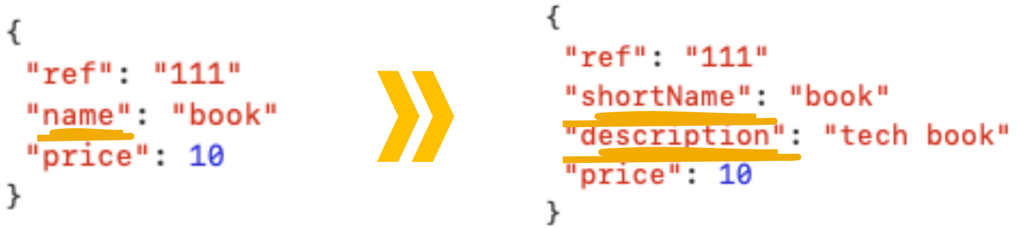
- Cart Service, Products Service, Web app (Products midro-FE, Cart micro-FE)

2- Run integration test

- Test passes (happy path + error cases)

3- Change the response structure in the provider

- Products Service: GET /product/{ref}



```
{
  "ref": "111"
  "name": "book"
  "price": 10
}
```

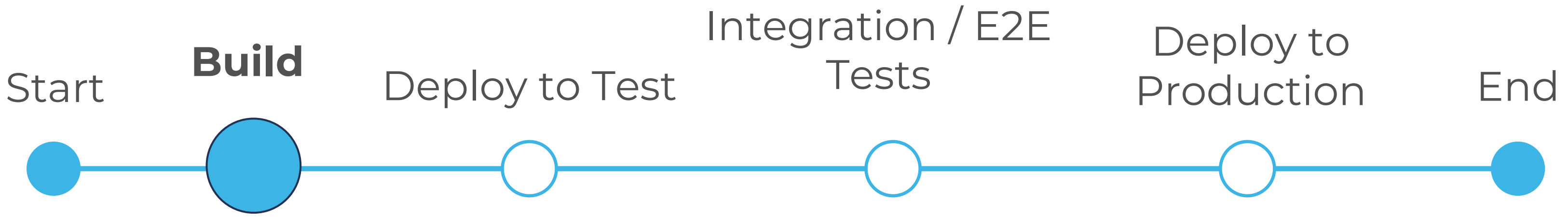
```
{
  "ref": "111"
  "shortName": "book"
  "description": "tech book"
  "price": 10
}
```

Late detection

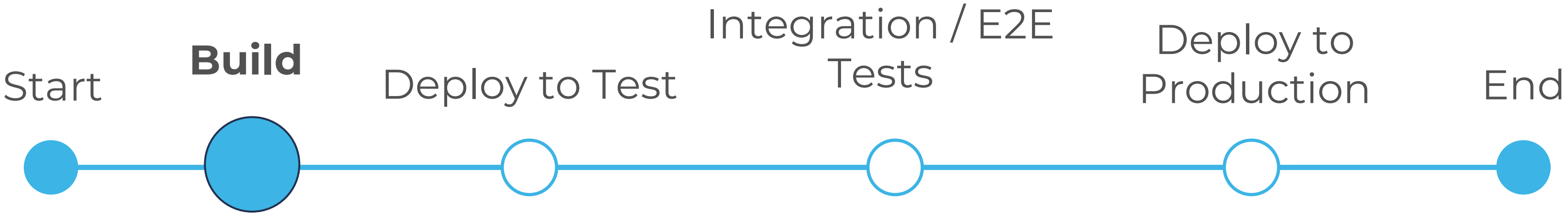
Contract testing

A kind of integration test

Testing in build time



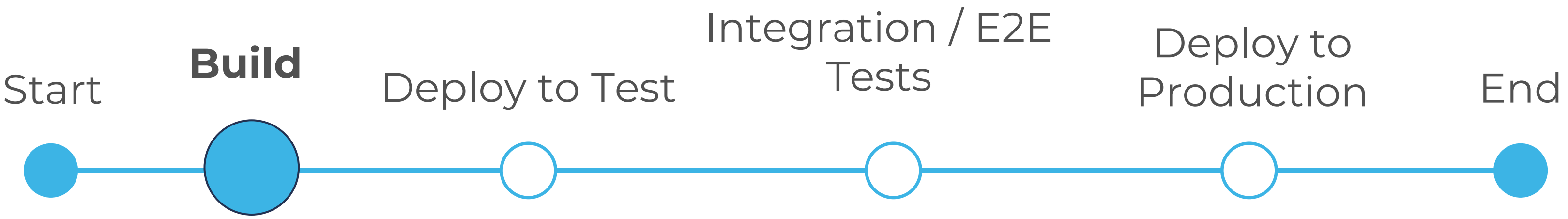
Testing in build time



Mocking

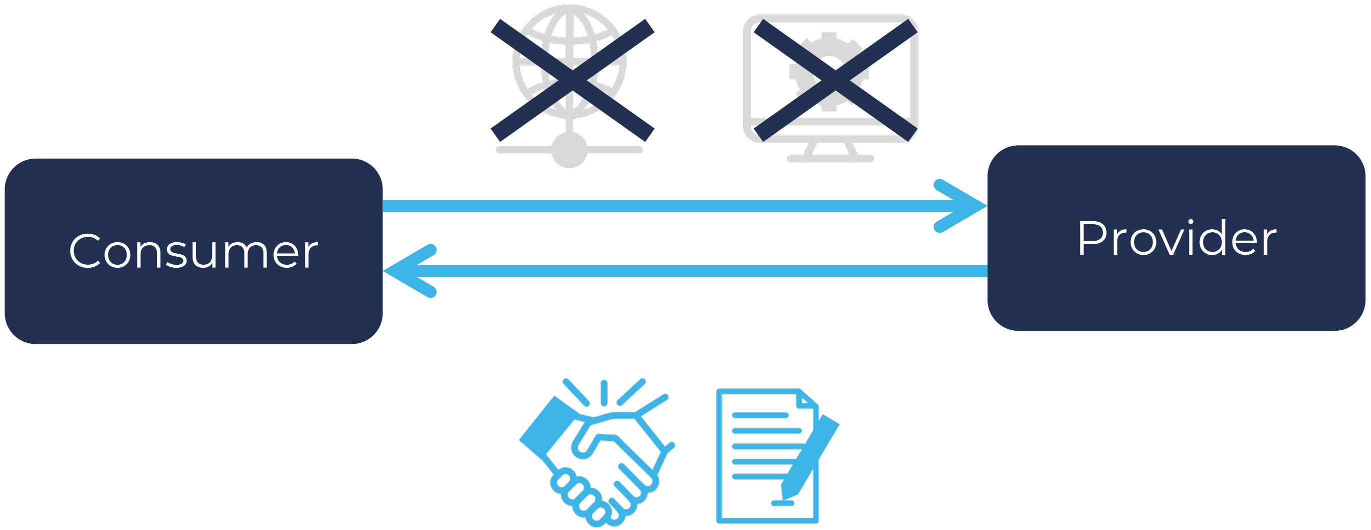
Out-of-date mocks

Testing in build time

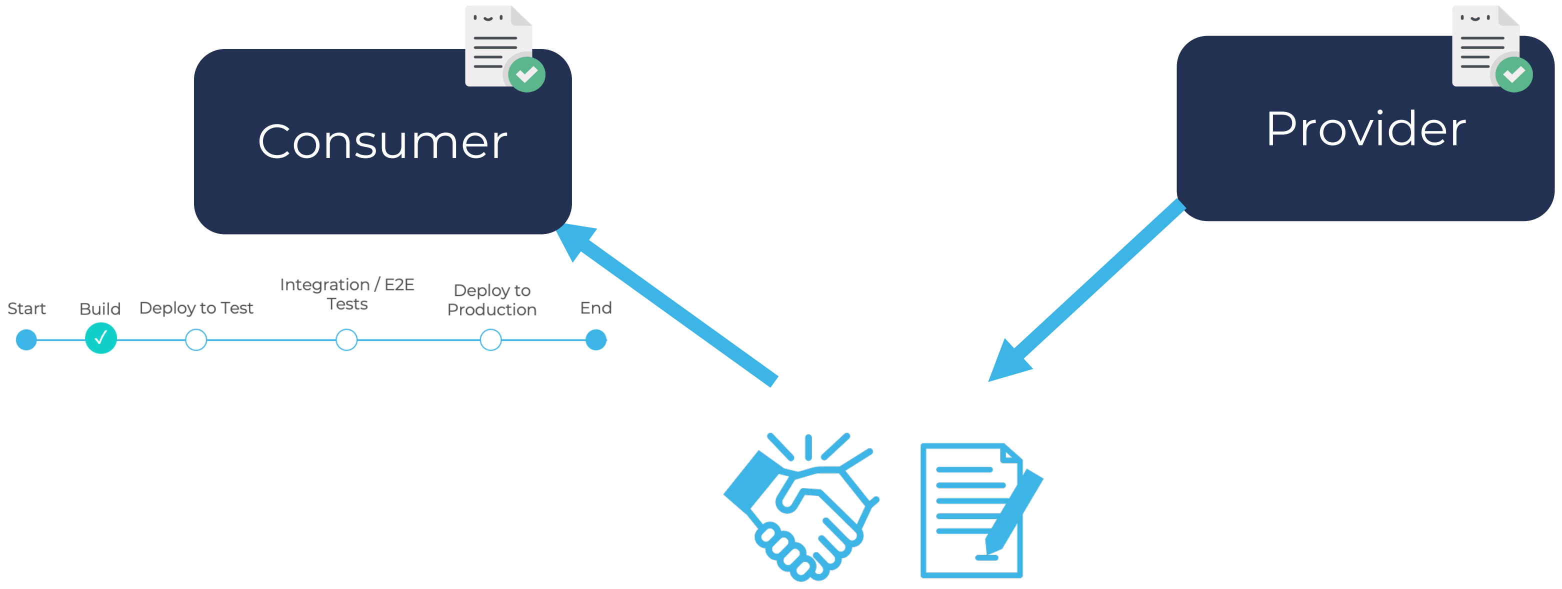


Mocking	Contract testing
Out-of-date mocks	Up-to-date mocks

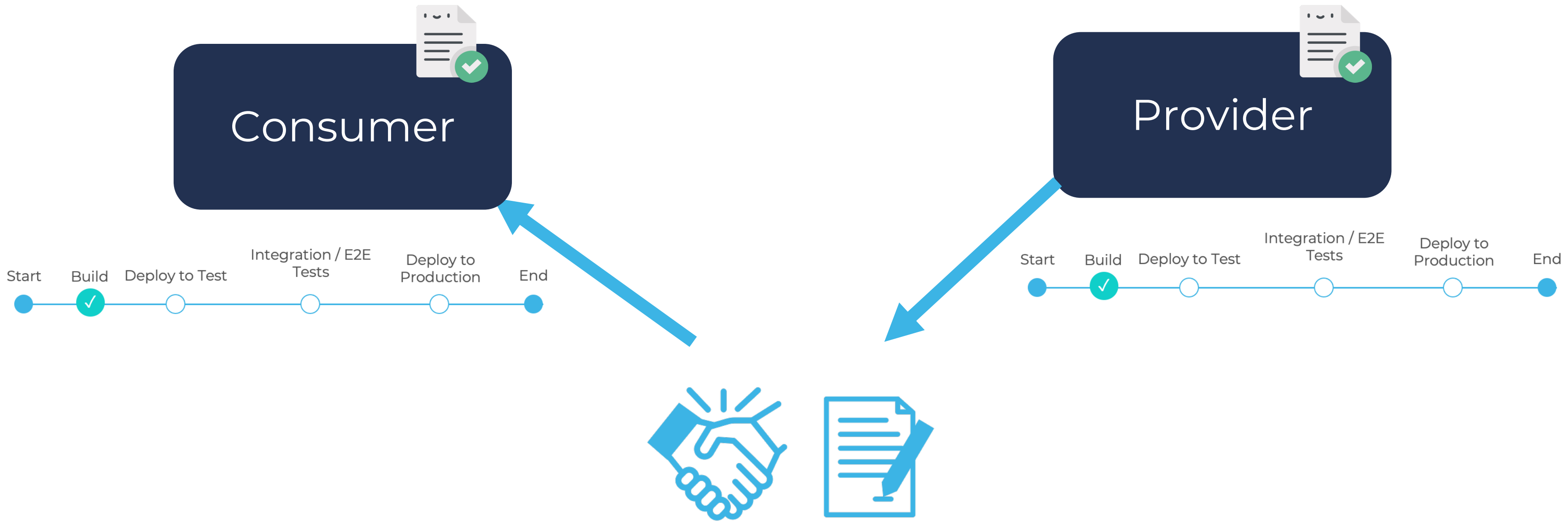
Contract testing



Contract testing



Contract testing



Contract testing benefits

- Providers: Test the integration with the consumer in the way they have defined
- Consumers: Reliable mocks verified by provider
- Healthy environments: Build is broken instead of the environment
- Pre-commit test in local
- Reduce the number of E2E and API tests

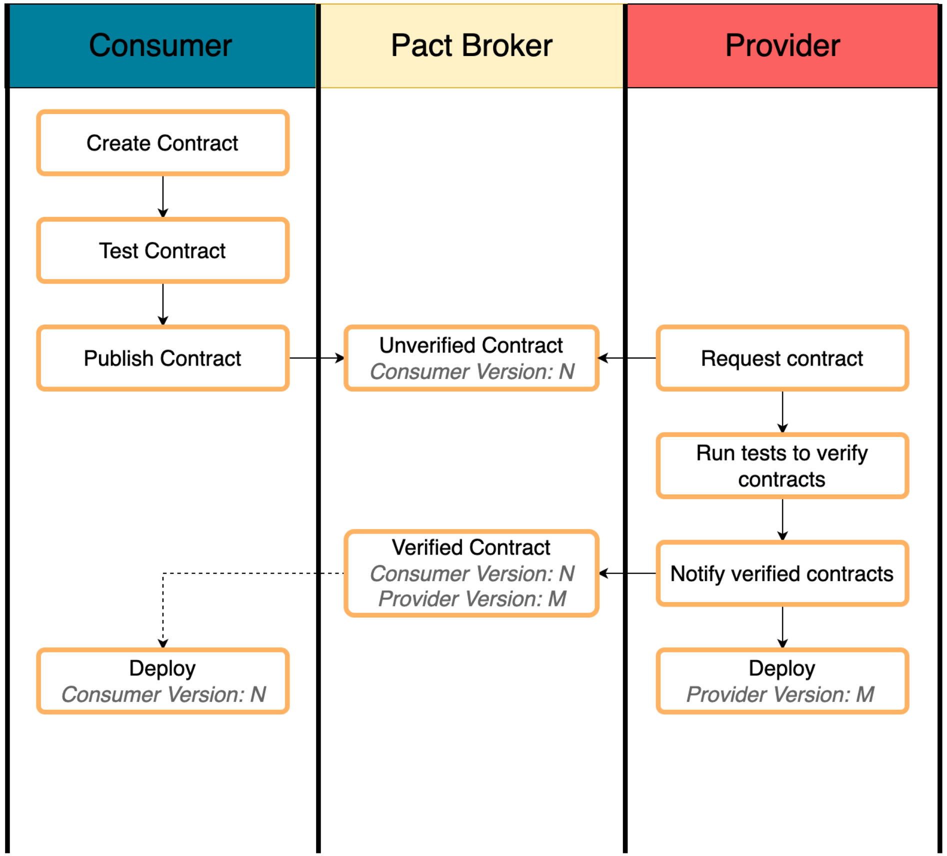
Pact

Framework for contract testing

Pact.io

Pact workflow

Consumer-Driven

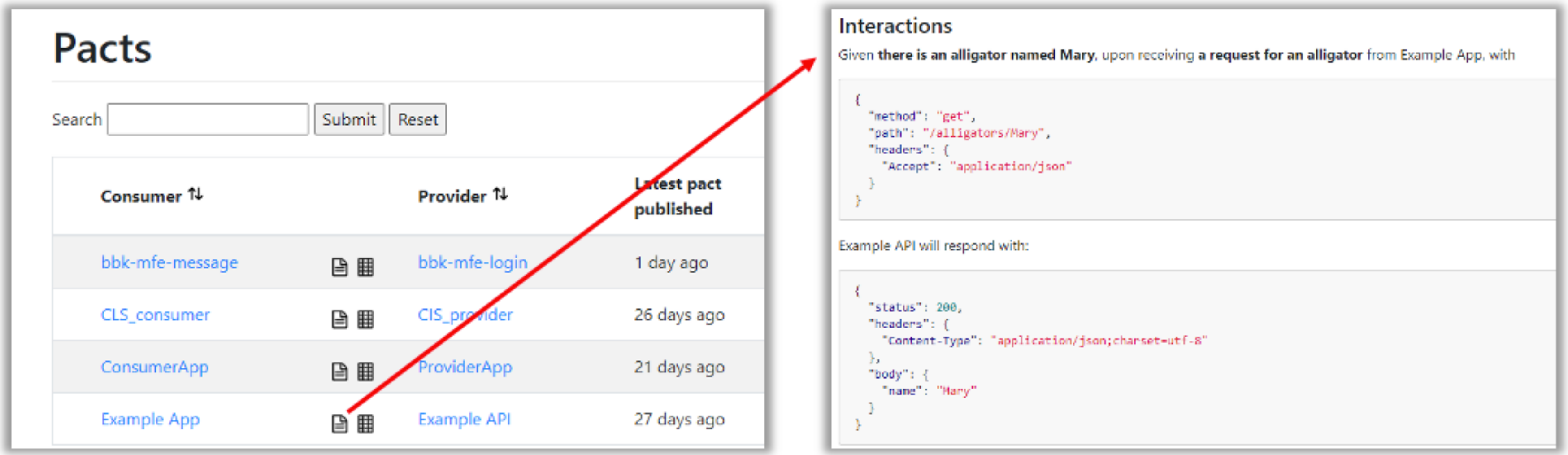


Exercises

Using Pact

Run Pact broker

Example



Pacts

Consumer ⇅		Provider ⇅	Latest pact published
bbk-mfe-message	🗎 ▦	bbk-mfe-login	1 day ago
CLS_consumer	🗎 ▦	CIS_provider	26 days ago
ConsumerApp	🗎 ▦	ProviderApp	21 days ago
Example App	🗎 ▦	Example API	27 days ago

Interactions

Given **there is an alligator named Mary**, upon receiving **a request for an alligator** from Example App, with

```
{
  "method": "get",
  "path": "/alligators/Mary",
  "headers": {
    "Accept": "application/json"
  }
}
```

Example API will respond with:

```
{
  "status": 200,
  "headers": {
    "Content-Type": "application/json;charset=utf-8"
  },
  "body": {
    "name": "Mary"
  }
}
```

Exercise 1

Backend to backend contract

Backend to backend

1- Create & run a consumer test with a contract

Contract is created

2- Publish contract to Pact broker

A new entry in Pact broker for the contract

3- Create & run a provider test

Provider response is validated against all affected contracts in Pact broker

The verification result is updated in Pact Broker

Backend to backend

1- Build a new version in the provider changing the response format

A new verification is added in Pact Broker

2- Breaking a successful verification in the provider

A new response format in the provider

The provider build fails

3- Change contract in the consumer side

We need a new version for the consumer with a new contract

Contract test passes in consumer but validation is failed in the provider

Exercise 2

Frontend to backend contract

Frontend to backend

1- Create & run a consumer test with a contract

Contract is created.

2- Publish contract to Pact broker

A new entry in Pact broker for the contract.

3- Create & run a provider test

Provider response is validated against all affected contracts in Pact broker.

Exercise 3

Frontend to frontend contract

Frontend to frontend

1- Create and run a consumer test with a contract

Contract is created.

2- Publish contract to Pact broker

A new entry in Pact broker for the contract.

3- Create and run a provider test

Provider response is validated against all affected contracts in Pact broker.

Features in Pact

Can-I-deploy

Service	Version	Code
cart-service **Consumer**	v1	Request GET /product/{111} to expect {"name": "book"}
products-service **Provider**	v1	Return {"name": "book"}
cart-service **Consumer**	v2	Request GET /product/{111} to expect {"shortName": "book", "description": "tech book"}
products-service **Provider**	v2	Return {"shortName": "book", "description": "tech book"}

Service	Build	Test Env	Test Prod
▼ **Consumer** cart-service	v2	v2	v1 → v2 Fail
▼ **Provider** products-service	v2	v2	v1

Can-I-deploy

Demo

Can-I-deploy

Consumer version	Provider version	Verified?	Comment
v1 (production)	v1 (production)	yes	The command ‘record-deployment’ is run to set version deployed in the environment
v2	v1	no	Consumer updated the contract
v2	v2	yes	Provider is updated according to the contract

1- Run can-I-deploy provider v2 in production env → NO

- provider v2 is not verified for consumer v1 that is in production
- provider v2 is verified for consumer v2 that is not deployed in production yet

2- Run can-I-deploy consumer v2 in production env → NO

- consumer v2 is not verified for provider v1 that is in production
- consumer v2 is verified for provider v2 that is not deployed in production yet

Can-I-deploy

Consumer version	Provider version	Verified?	Comment
v1 (production)	v1 (production)	yes	The command 'record-deployment' is run to set version deployed in the environment
v2	v1	no	Consumer updated the contract
v2	v2	yes	Provider is updated according to the contract
v1	v3	yes	Provider is updated to be compatible with v1 and v2
v2	v3	yes	Provider is updated to be compatible with v1 and v2

4- Run can-I-deploy provider v3 in production env → YES

- provider v3 is verified for consumer v1 that is in production

5- Run can-I-deploy consumer v2 in production env → YES

- consumer v2 is verified for provider v3 that is in production

Webhook

Automated trigger of provider builds

To ensure all contracts are verified by the provider

Used in CI/CD

Takeaways

Takeaways

Early testing

- Bug prevention
- Less testing after deployment

All types of integrations

- Synchronous & asynchronous
- Web apps, APIs

CI/CD oriented

- Deploy faster, safer & more often

Communication

- Between consumer team and provider team

Documentation

- Who uses what?

Thanks!