MODUS CREATE

HUSTEF
HUNGARIAN SOFTWARE TESTING FORUM

HUSTEF 2022

#hustef22

October 4-6, 2022

MODUS CREATE

HUSTEF
HUNGARIAN SOFTWARE TESTING FORUM

CONTRACT TESTING FOR MOBILE APPLICATION DEVELOPMENT

Motto: no more brittle integration testing!

MESUT GÜNEŞ

Principal Consultant / Quality at Modus Create

in/gunesmes

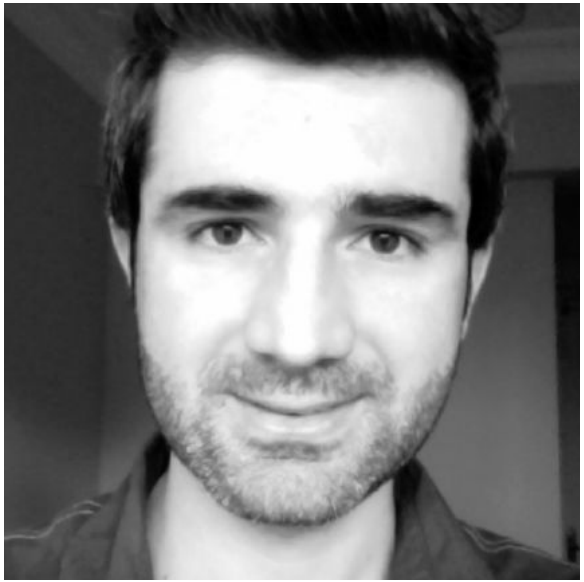
gunesmes

story/gunesmes

gunesmes

testrisk.com

gunesmes@gmail.com



TOPICS

- Contract
- Contract Testing
- Consumer-Provider Definition
- Integration Testing
- Mobile Testing
- Mobile Development Architecture
- Mobile Development Process
- Shift-left Testing

TABLE OF CONTENTS

TESTING FOR MOBILE APP DEVELOPMENT

TEST TYPES AND LEVELS

What tests in what level we should test the apps.

DEBATE
Where should we put the CONTRACT TESTING in the matrix?

NON-FUNCTIONAL
it doesn't validate logic or consumer flows

INTEGRATION
it checks the things that are integrated to others

TEST LEVEL	TEST TYPES	
	FUNCTIONAL	NON-FUNCTIONAL
UNIT	unit tests	load, stress, security, code coverage (metric)
INTEGRATION	unit integration tests, component integration tests, microservices testing	load, stress, security, contract test
SYSTEM	end-to-end test	load, stress, security, reliability, maintainability, scalability
ACCEPTANCE	uat test, alpha - beta testing	load, stress, security, usability, AB test

LET'S FOCUS ON INTEGRATION LEVEL

Integration level has **wider test responsibilities** since the context of the **unit** is very **small** and the **UI** is **user** oriented.

Checking the integration of units/components, services, APIs or even systems

INTEGRATION TEST IS INEFFICIENT

Flakiness

+ Data management for each MSs and APIs
+ Testing different MSs in the same test

Slow

+ Requires network
+ Data creation process
+ Network calls between MSs

Independent from Clients

+ Tests are not created by clients
+ Updates are not made by clients
+ Testing more general instead of clients requirements

LET’S MAKE IT BETTER

How about tests run on the **build time** on its context with written documents that clients create **depends on their specifications.**

CONTRACT TESTING

WHAT IS CONTRACT

Definition of contract

According to dictionary.com

+ An **agreement** between two or more parties for the d**oing or not doing of something** specified.
+ An **agreement** enforceable by **law**.
+ The **written form of** such an agreement.

WHAT IS CONTRACT - TECHNICALLY

Definition of contract

+ An agreement between **provider** and **consumer** for the doing something specified.
+ An agreement **enforceable** by **consumer** or **provider**.
+ Json files to describe expectations

WHAT IS CONTRACT TESTING

Definition of contract

+ **Checking** the contracts
+ **Isolating** the testing efforts
+ **TDD approach** to system
+ Testing the software in **its context** - build time
+ Making the system **shift-left** by early integration testing

KEYWORDS

Contract

+ Specifications of requests and responses

Consumer

+ Clients, one side that consumes the APIs/MCs

Provider

+ One side owns the API

Pact Broker

+ Common place where the contracts live

PROVIDER-DRIVEN CONTRACT TEST

Contracts are created by providers so the provider drives all activities.

Pros

+ Provider **decides** naming and versioning
+ Good if APIs are **public**

Cons

+ No info about **requirements**
+ No info the **aims** for consuming

CONSUMER-DRIVEN CONTRACT TEST

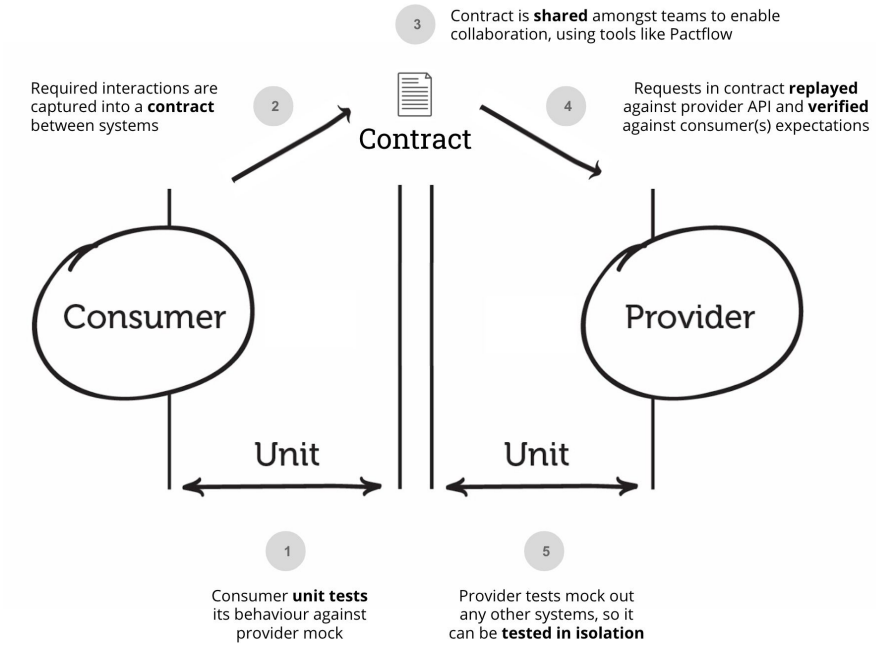
Contracts are created by **consumers** so that the providers use them to check the **specifications** of the consumers

Pros

+ Up-to-date contracts
+ Consumers unit test its behaviors
+ Providers check in isolation

Cons

+ Dependency to follow consumer



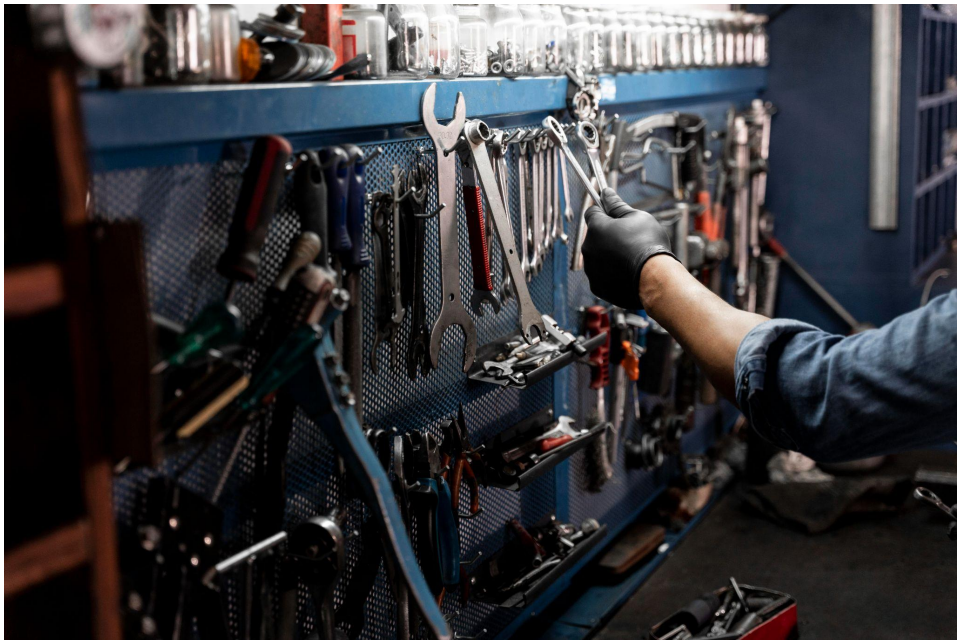
CONTRACT TESTING TOOLS

Pact

+ Open Source - First commit: **Feb 20, 2013**
+ Pact Foundation created
+ Smartbear acquired
+ Supports **many languages**
  + JS
  + Ruby
  + **JVM**
  + **Swift**
  + Python
  + ... more than 11

Spring Cloud Contract

+ Open Source - First commit: **Dec 6, 2014**
+ Mainly for Spring Boot application
+ Supports
  + JVM
  + Non JVM for provider contract

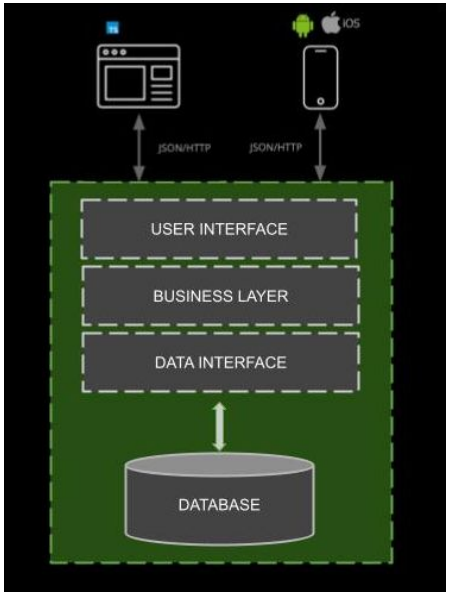


CONTRACT TESTING FOR MOBILE APPS

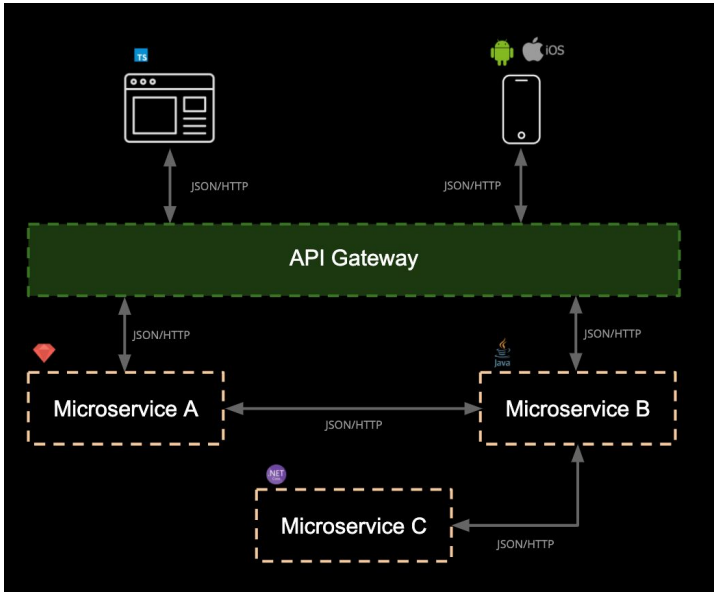
ARCHITECTURE

Contract test should be shaped depends on the architecture that you have for the whole system. Do you have **monolithic** or **microservice architecture?**

Monolithic



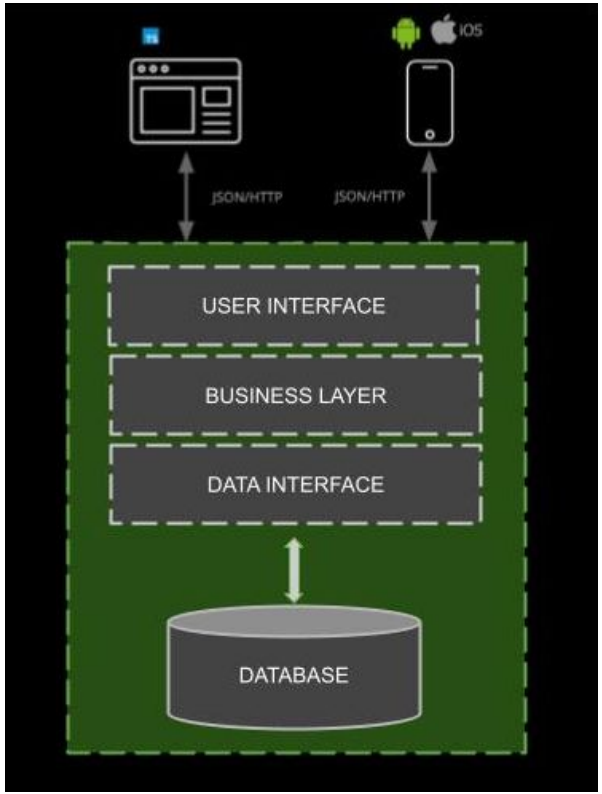
Microservices



ARCHITECTURE - MONOLITHIC

Monolithic architecture has **one code base** but the it has several logical layer which are **strictly coupled**.

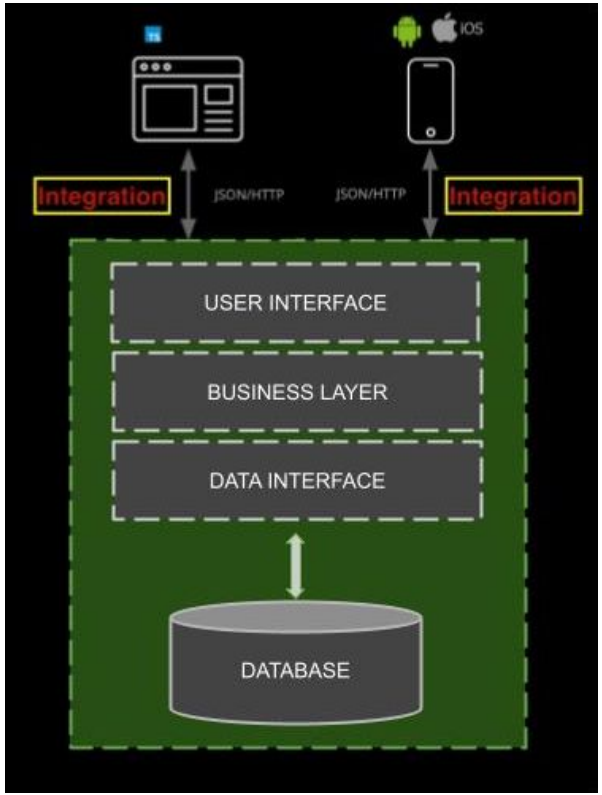
+ One repo
+ One language
+ One pipeline



ARCHITECTURE - MONOLITHIC

All clients connect to the application **instance directly**. If many instances we have for scaling we should have a **load balancer** to distribute the load.

+ Integrations happen on client connections
  + Mobile
  + Web
  + API
+ These are where we can apply **contract tests**



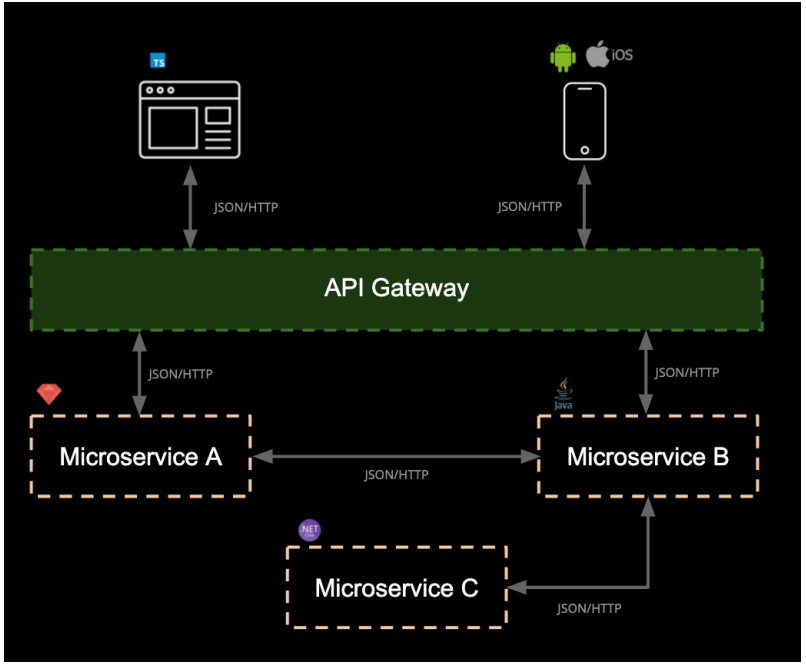
ARCHITECTURE - MICROSERVICE

New thing!

Instead of one big code base, there are many small services.

Nowadays this is **mostly applied architecture**.

This simple microservice architecture has many MS and an API Gateway. The **API Gateway** is the point where **clients** are connected.

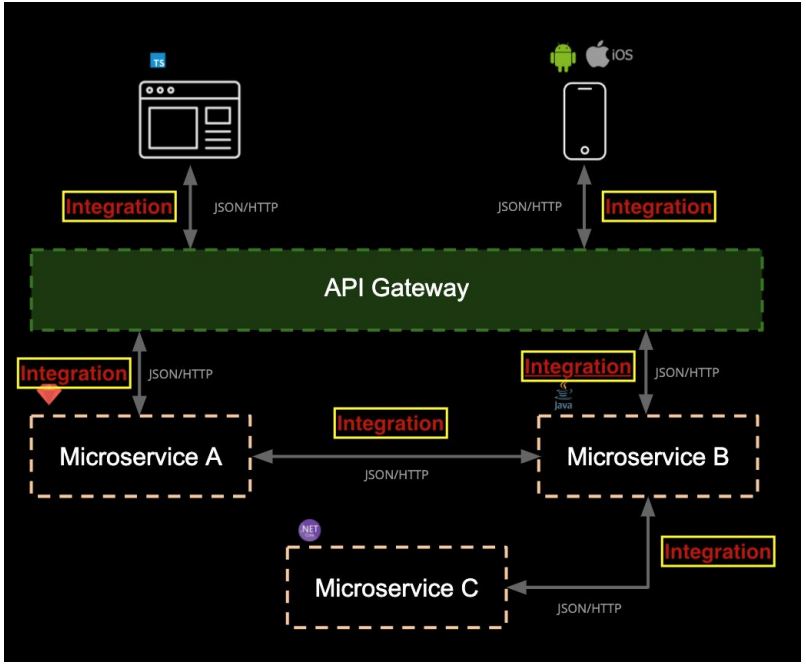


ARCHITECTURE - MICROSERVICE

Bingo!

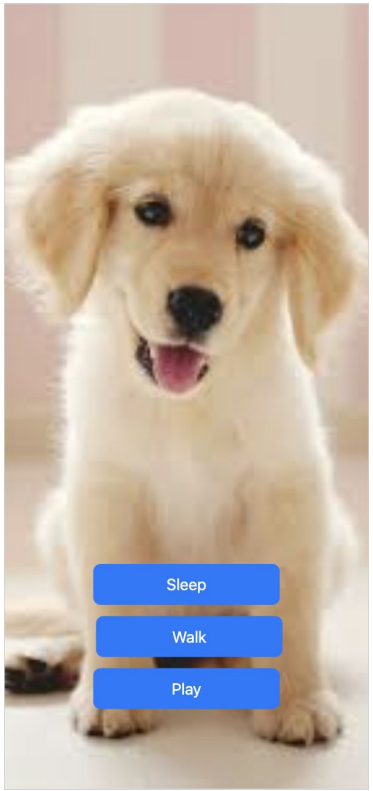
... we have **many integration** points :)

Who are the consumers and the providers?

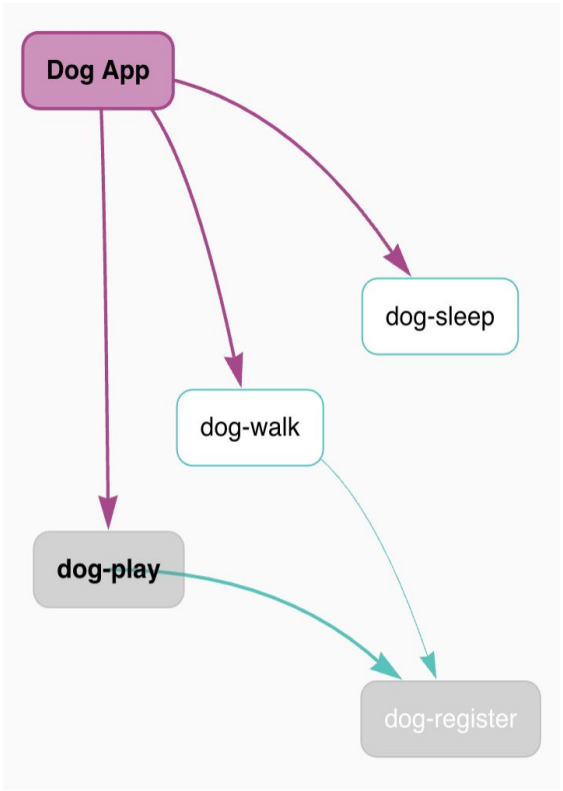


4 IMPLEMENTATION

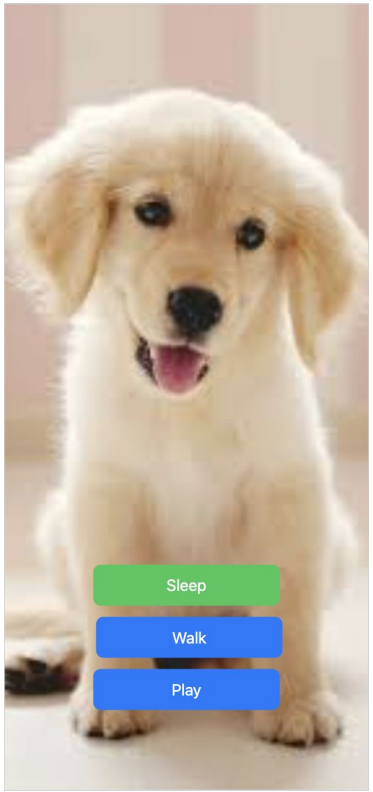
DOG APP



Dog App consumes **Microservices**. Dogs are very **simple animals** so the contract network is not **that complex :)**

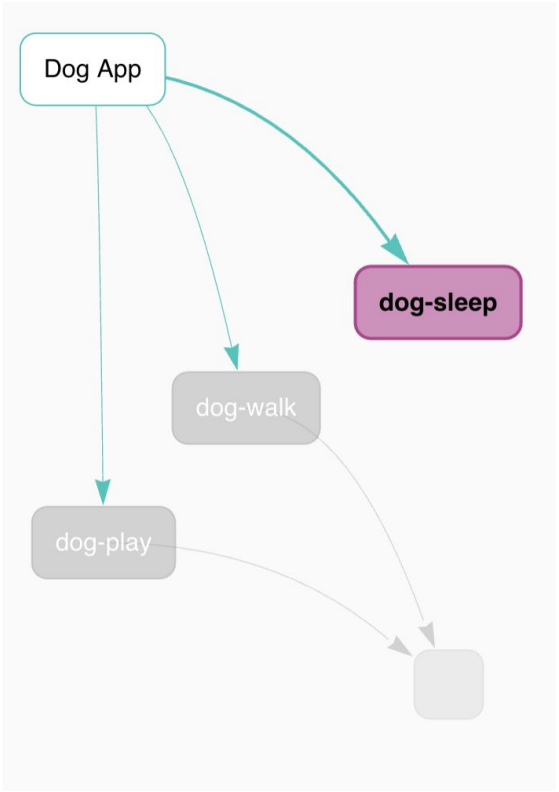


DOGS WANT SLEEPING

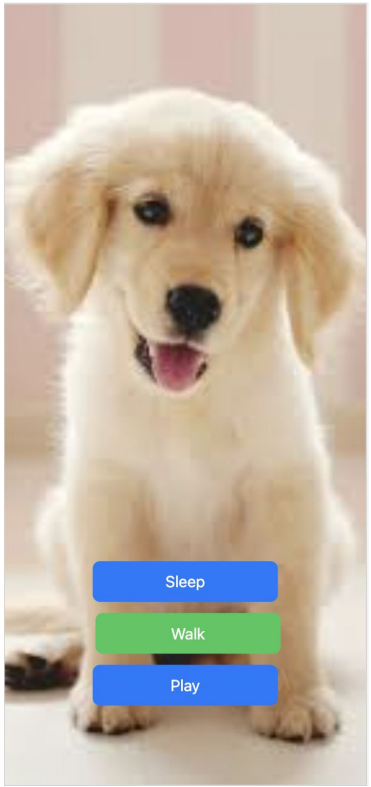


GET localhost:8992/sleep

⇒ `[{ dog: 1 }, { dog: 2 }]`

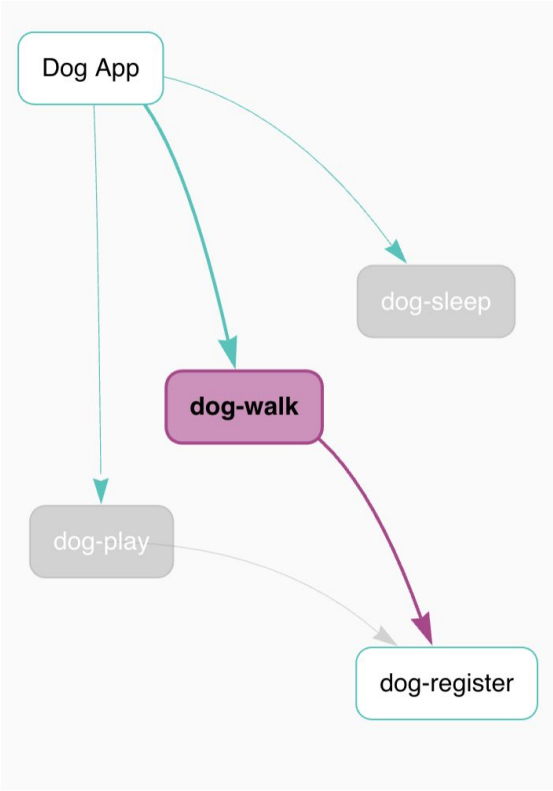


DOGS WANT WALKING

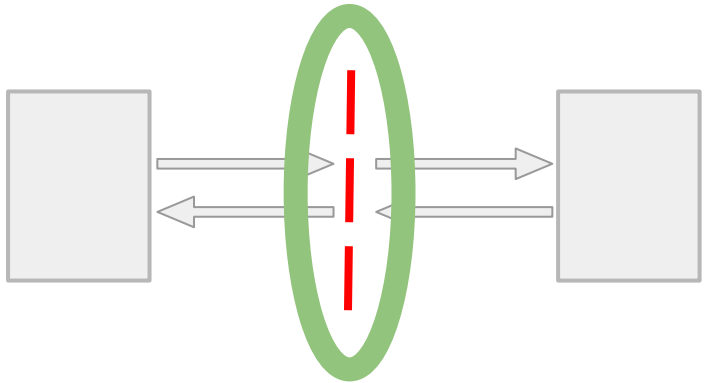


GET localhost:8993/walk

⇒ `[{ dog: 1 }, { dog: 2 }]`

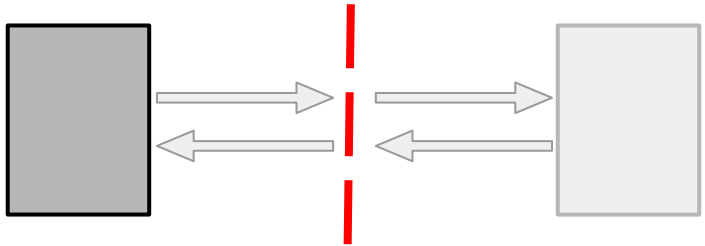


DOGS APP CONTRACT TEST - Mock Service



```
class DogAppClientSpec: QuickSpec {
  override func spec() {
    var dogAppService: MockService?
    var dogAppClient: dogAppClient?

    describe("tests of against to SleepService") {
      beforeEach {
        dogAppService = MockService(
            provider: "DogApp Sleep Service",
            consumer: "DogApp iOS App")
      }

      it("it returns dogs want sleeping") {
        dogAppService!.uponReceiving("a request for sleeping")
                      .withRequest(
                       method: .GET,
                         path: "/sleep",
                         headers: ["Accept": "application/json"])
                      .willRespondWith(
                         status: 200,
                         headers: ["Content-Type": "application/json"],
                         body: "[{dog: 1}, {dog: 2}]")


        dogAppService!.run { (testComplete) -> Void in //Run tests
          dogAppClient!.sleep { (message, status) -> Void in //Test client
            expect(status).to(equal(200))
            expect(body).to(equal("[{dog: 1}, {dog: 2}]"))
            testComplete()
          }
        }
      }
```

DOGS APP CONTRACT TEST - A Specification

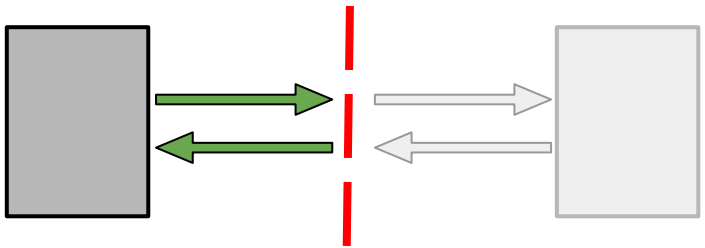


```
class DogAppClientSpec: QuickSpec {
  override func spec() {
    var dogAppService: MockService?
    var dogAppClient: dogAppClient?

    describe("tests of against to SleepService") {
      beforeEach {
        dogAppService = MockService(
            provider: "DogApp Sleep Service",
            consumer: "DogApp iOS App")

        dogAppClient = dogAppClient(baseUrl: dogAppService!.baseUrl)
      }
```

```
dogAppService!.uponReceiving("a request for sleeping")
              .withRequest(
               method: .GET,
                 path: "/sleep",
                 headers: ["Accept": "application/json"])
              .willRespondWith(
                 status: 200,
                 headers: ["Content-Type": "application/json"]
                 body: "[{dog: 1}, {dog: 2}]")
```

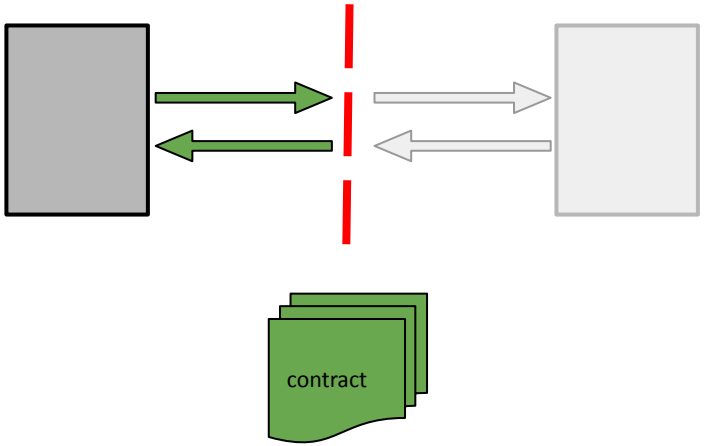
```
        dogAppService!.run { (testComplete) -> Void in //Run tests
          dogAppClient!.sleep { (message, status) -> Void in //Test client
            expect(status).to(equal(200))
            expect(body).to(equal("[{dog: 1}, {dog: 2}]"))
            testComplete()
          }
        }
      }
```

DOGS APP CONTRACT TEST - Run Test



```
class DogAppClientSpec: QuickSpec {
  override func spec() {
    var dogAppService: MockService?
    var dogAppClient: dogAppClient?

    describe("tests of against to SleepService") {
      beforeEach {
        dogAppService = MockService(
            provider: "DogApp Sleep Service",
            consumer: "DogApp iOS App")

        dogAppClient = dogAppClient(baseUrl: dogAppService!.baseUrl)
      }

      it("it returns dogs want sleeping") {
        dogAppService!.uponReceiving("a request for sleeping")
                      .withRequest(
                       method: .GET,
                         path: "/sleep",
                         headers: ["Accept": "application/json"])
                      .willRespondWith(
                         status: 200,
                         headers: ["Content-Type": "application/json"],
                         body: "[{dog: 1}, {dog: 2}]")
dogAppService!.run { (testComplete) -> Void in //Run test
  dogAppClient!.sleep { (message, status) -> Void in //Te
    expect(status).to(equal(200))
    expect(body).to(equal("[{dog: 1}, {dog: 2}]"))
          testComplete()
        }
      }
    }
```

DOGS APP CONTRACT TEST - Complete




```
class DogAppClientSpec: QuickSpec {
  override func spec() {
    var dogAppService: MockService?
    var dogAppClient: dogAppClient?

    describe("tests of against to SleepService") {
      beforeEach {
        dogAppService = MockService(
            provider: "DogApp Sleep Service",
            consumer: "DogApp iOS App")

        dogAppClient = dogAppClient(baseUrl: dogAppService!.baseUrl)
      }

      it("it returns dogs want sleeping") {
        dogAppService!.uponReceiving("a request for sleeping")
                      .withRequest(
                       method: .GET,
                         path: "/sleep",
                         headers: ["Accept": "application/json"])
                      .willRespondWith(
                         status: 200,
                         headers: ["Content-Type": "application/json"],
                         body: "[{dog: 1}, {dog: 2}]")


        dogAppService!.run { (testComplete) -> Void in //Run tests
          dogAppClient!.sleep { (message, status) -> Void in //Test client
            expect(status).to(equal(200))
            testComplete()
          }
        }
      }
```

CONTRACT

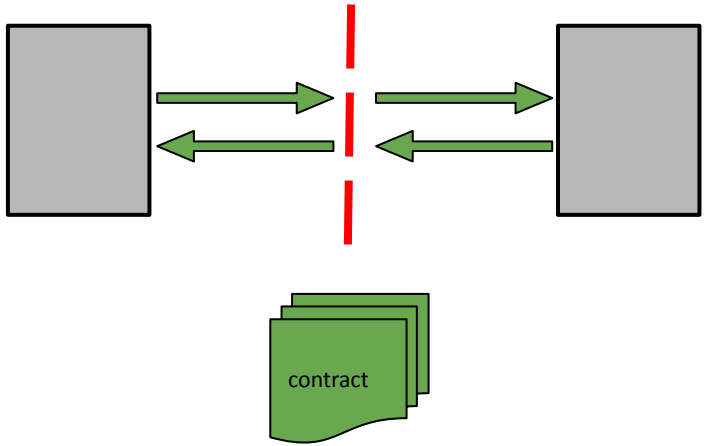
GET localhost:8992/sleep

⇒ `[{ dog: 1 }, { dog: 2 }]`

pacts > {} dog_app-dog-sleep.json

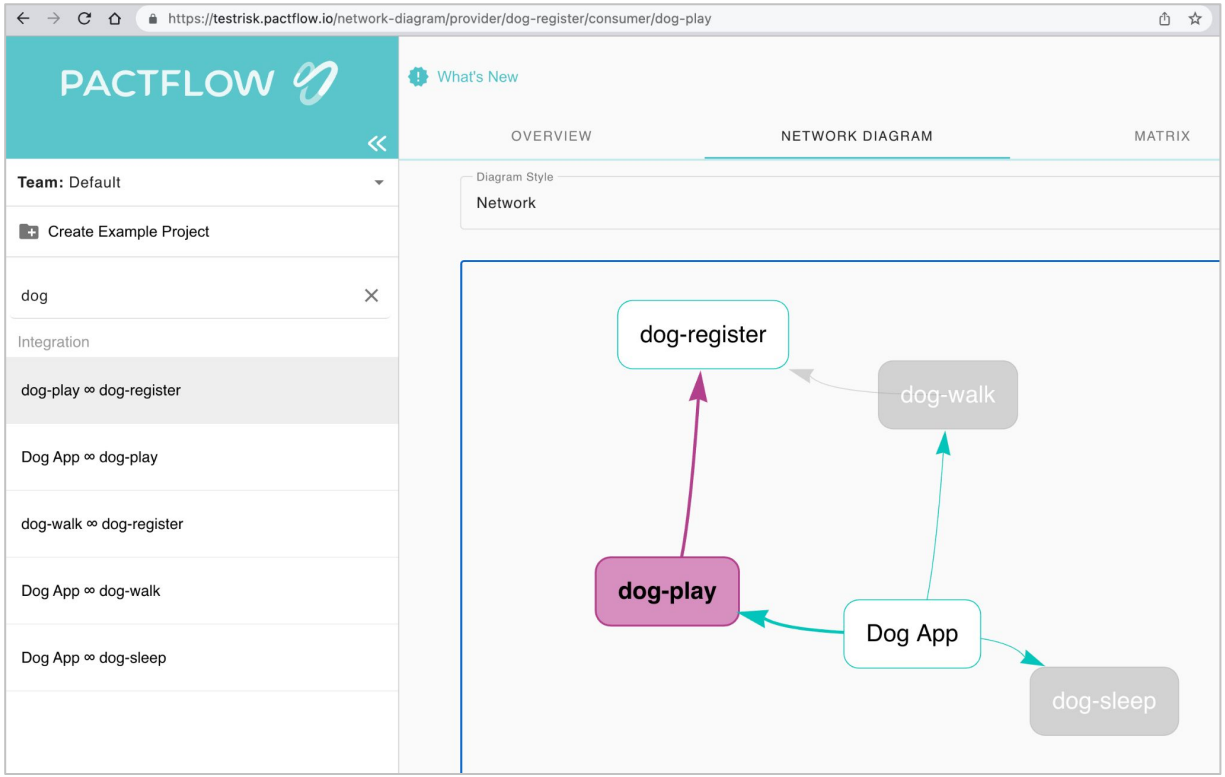

```
{
  "consumer": {
    "name": "DogApp iOS App"
  },
  "provider": {
    "name": "DogApp Sleep Service"
  },
  "interactions": [
    {
      "description": "tests against to SleepService",
      "providerState": "a request for sleeping",
      "request": {
        "method": "GET",
        "path": "/sleep",
        "headers": {
          "Accept": "application/json"
        }
      },
      "response": {
        "status": 200,
        "headers": {
          "Content-Type": "application/json"
        },
        "body": [
          {
            "dog": 1
          },
          {
            "dog": 2
          }
        ]
      }
    },
```

DOGS APP CONTRACT TEST - Verify Provider




```
72
73  const opts: VerifierOptions = {
74    stateHandlers: {…
131   },
132   requestFilter: (req, res, next) => {…
147   },
148   provider: process.env.PACT_PROVIDER_NAME,
149   providerBaseUrl: process.env.PACT_PROVIDER_URL,
150   pactBrokerUrl: process.env.PACT_BROKER_BASE_URL,
151   pactBrokerToken: process.env.PACT_BROKER_TOKEN,
152
153   publishVerificationResult: publishResultsFlag || false,
154   validateSSL: true,
155   changeOrigin: true,
156   providerVersion,
157   tags: tagsArray,
158   logLevel: "error"
159 };
160
161 new Verifier(opts)
162   .verifyProvider()
163   .then(() => {
164     // tslint:disable-next-line: no-console
165     console.log("successfully verified pacts");
166     process.exit(0);
167   })
168   .catch((error: any) => {
169     // tslint:disable-next-line: no-console
170     console.log(error);
171     process.exit(1);
172   });
173
```

BROKER CONTRACTS



Q&A

References

+ https://blog.crisp.se/2013/06/20/alexandertarlinder/continuous-delivery-the-simplest-possible-build-pipeline-for-an-integration-scenario
+ https://github.com/andrewspinks/PactSwiftExample
+ https://vuestorefront.io/microservices

THANK YOU!

Let’s make something great together.

moduscreate.com