Contract testing for the past, present and the future with Pact-Go

Women Who Go – London

Feb 28th 2024

Hello WWGL!

Or Wiggle's as you shall affectionally be known

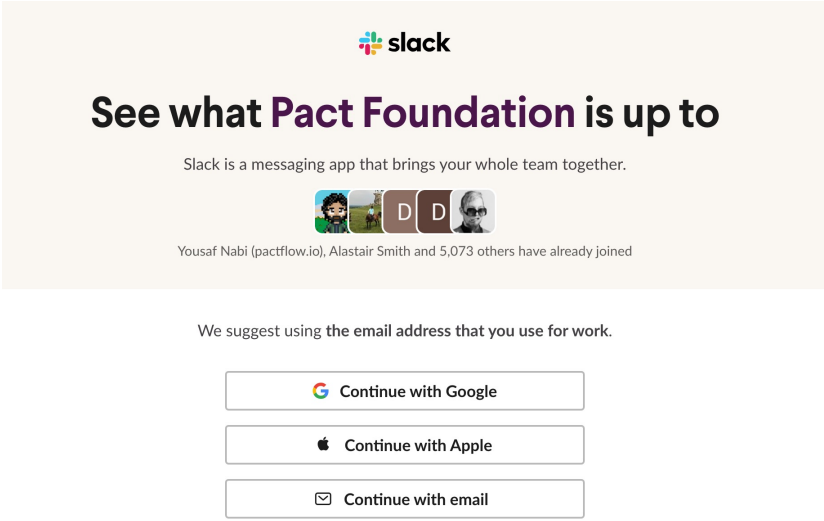
Women Who Go – London

Feb 28th 2024

Thank you Teea!

you54f

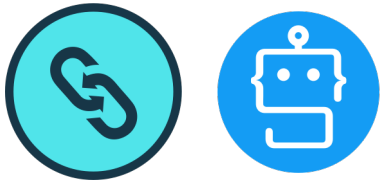
Join us and 5 thousand others: https://slack.pact.io



Frank Kilcommins
API Technical Evangelist
AKA Dev-Rel Dad

Jo Laing
Community Manager

Yousaf Nabi
Developer Advocate
Community Shepherd for Pact

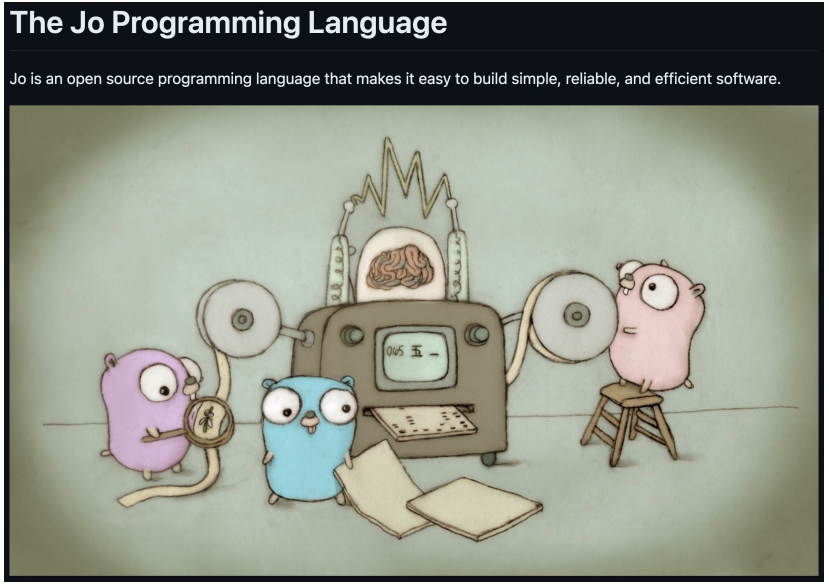


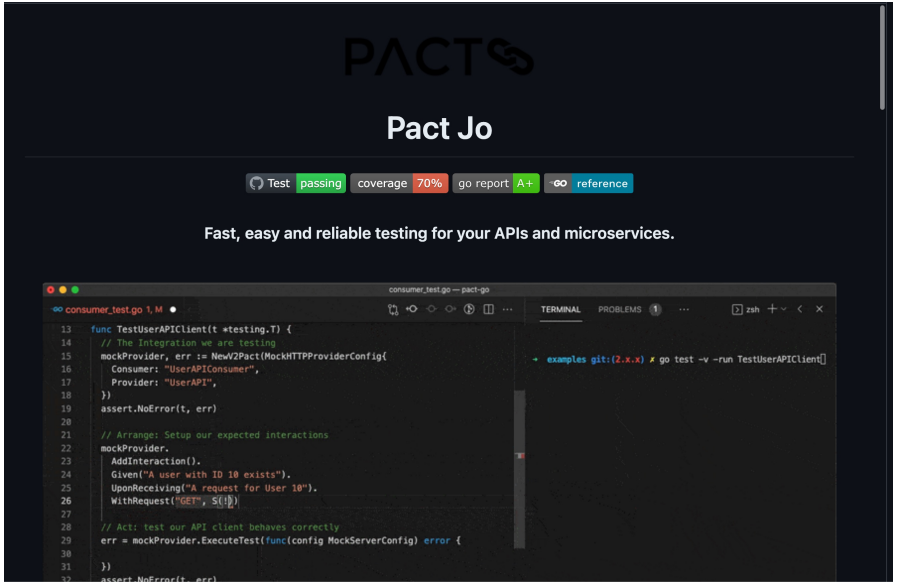
you54f

An Irishman, an Englishman, and an Scottish woman, walk into a bar

you54f

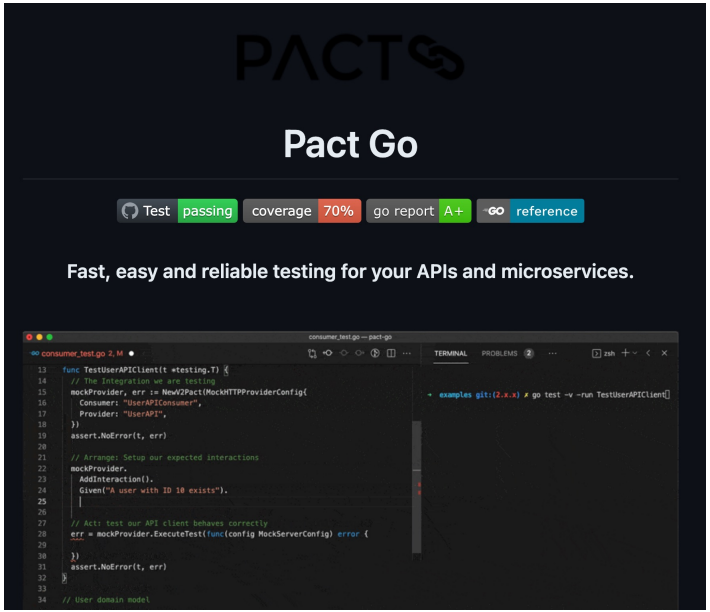
So I'm here to talk about JoLaing





you54f

Oh wait, I'm here to talk about GoLang!



Disclaimer:-

I'm not a Go expert! I'll learn as much from you, as you will from me

Modern architecture

The challenges facing today’s engineering leaders

PACTFLOW
A SMARTBEAR COMPANY

Customer quote

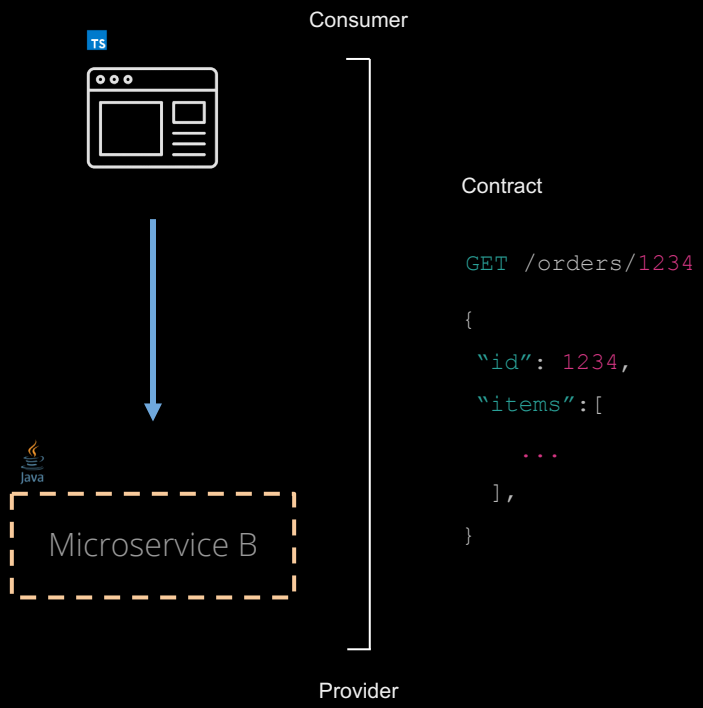
"We have a very large program with many different scrum teams building a wide variety of components all operating in a microservices event based architecture.

Testing inside a highly volatile set of integrated environments is extremely challenging today.

Looking to get better confidence by doing better isolated contract testing...

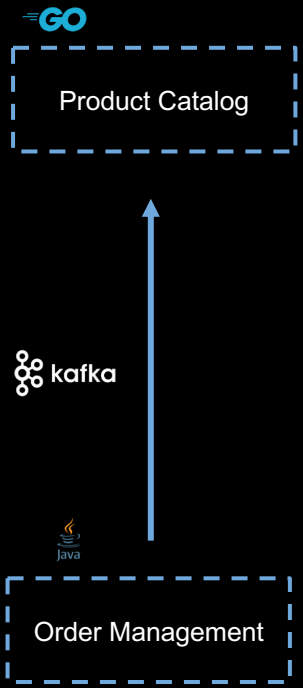
Between direct calls to RESTful or GraphQL APIs, or messages using AWS event bridge or Kafka, and also 3rd party SaaS and partner integrations...it's difficult to manage."

a large banking prospect



Contract

```
GET /orders/1234

{
 "id": 1234,
 "items":[
    ...
  ],
}
```



Consumer

Provider
(Producer)

Contract

```
topic: products
content-type: application/json

{
 "id": 1234,
 "items":[
    ...
  ],
}
```

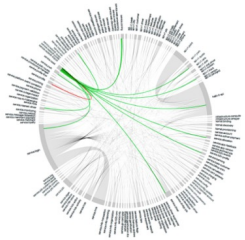
PACTFLOW
A SMARTBEAR COMPANY

Industry insights

More microservices, more protocols

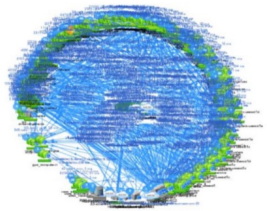
1. 61% say most API growth from microservices
2. 81% of companies operate in a multi-protocol environment
3. 57% use 3 or more protocols

450+ microservices



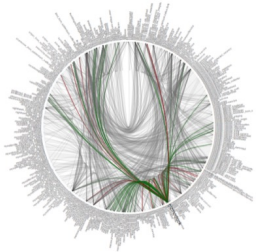
HAIL O

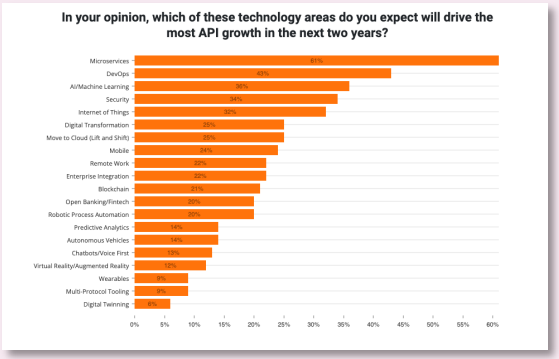
500+ microservices

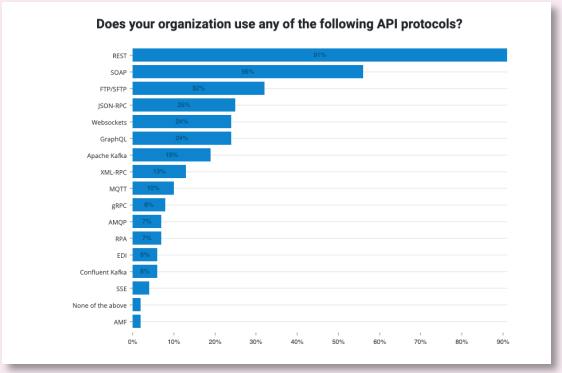


NETFLIX

500+ microservices







https://smartbear.com/state-of-software-quality/api/

PACTFLOW
A SMARTBEAR COMPANY

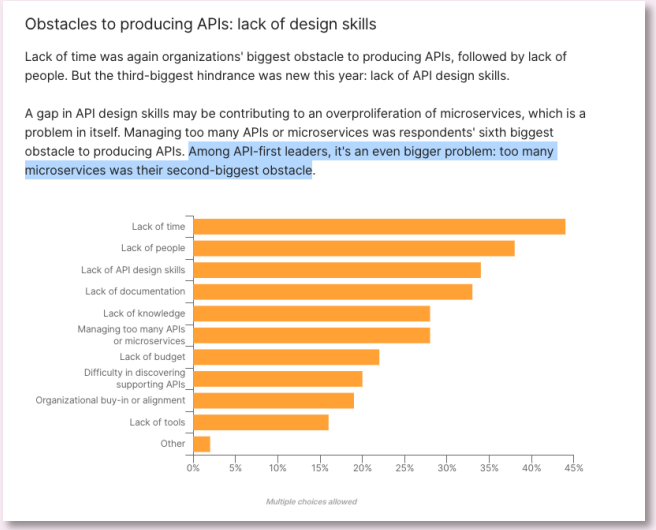
Industry insights

The headwinds of *“Microservices sprawl”*

Barriers to implementing microservices:

1. Experience or skills
2. Complexity of systems
3. Increasing demands on speed of delivery, and
4. Limited time due to workload

Mature organisations feeling the pain



“By 2025, less than 50% of enterprise APIs will be managed, as explosive growth in APIs surpasses the capabilities of API management tools.”

Gartner

https://smartbear.com/state-of-software-quality/api/

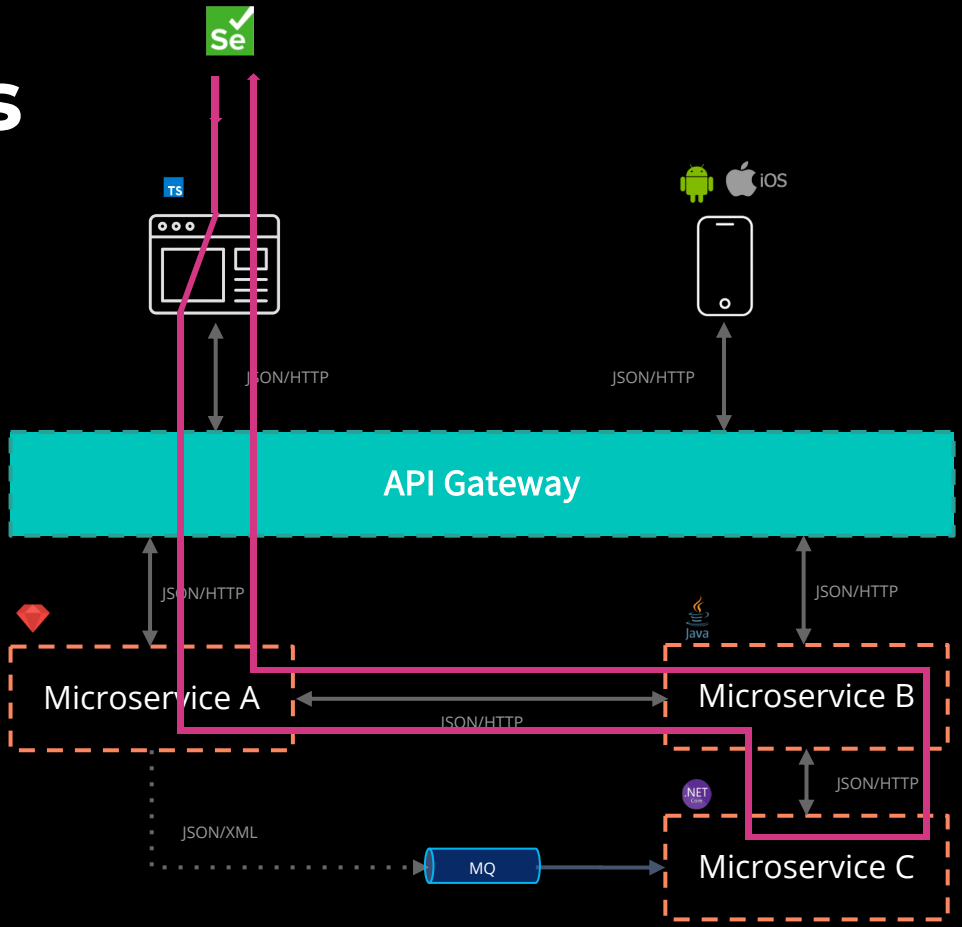
How we test microservices now

And why it doesn’t scale

End-to-end tests

Why this is hard

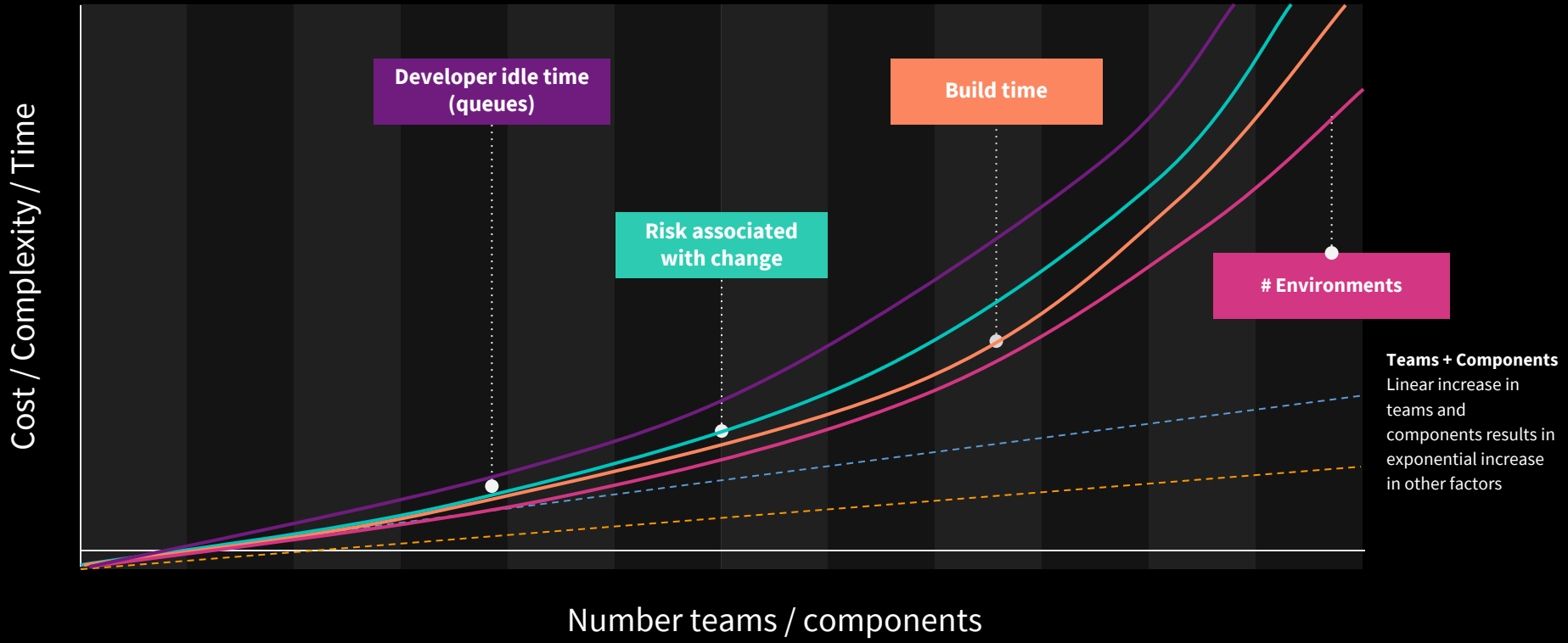
- Slow
- Fragile
- Hard to debug
- All-at-once painful deployments
- Teams wait on build queues



“If you can’t deploy your microservices independently, you have built a distributed monolith”

Beth Skurrie – Creator of the Pact Broker

Scaling Challenges



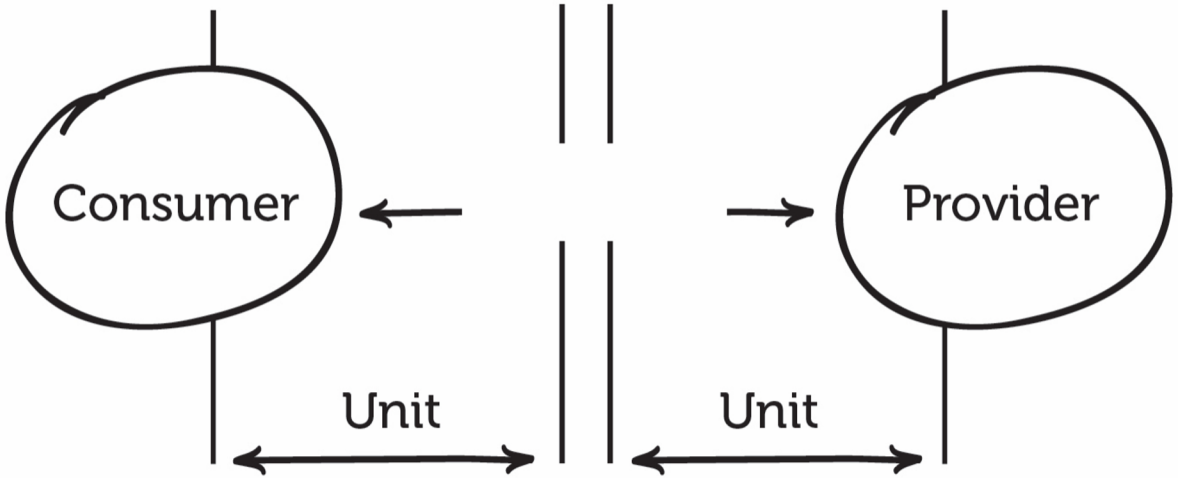
“If we just used *<insert some new tech>*
then we wouldn’t need contract testing”

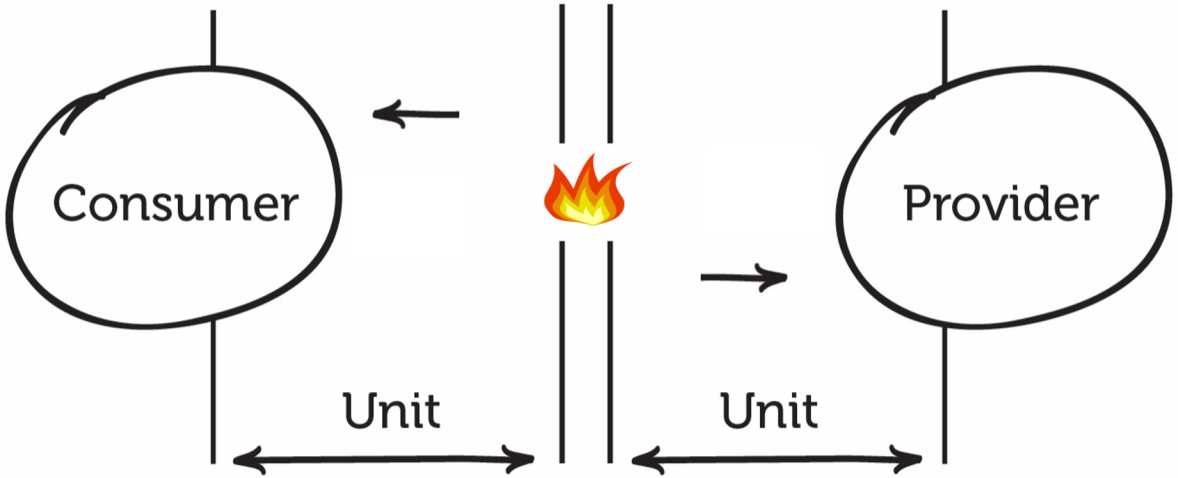
PACTFLOW
A SMARTBEAR COMPANY

If we just used

Mocks

Then, we wouldn’t need contract testing


Consumer
Provider
Unit
Unit


Consumer
Provider
Unit
Unit

Mocks

Solved problems

- Fast feedback
- Few dependencies
- No dedicated environment
- Reliable
- Easy to debug

New problems

- Hard to keep both sides in sync

PACTFLOW
A SMARTBEAR COMPANY

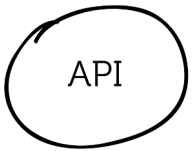
If we just used

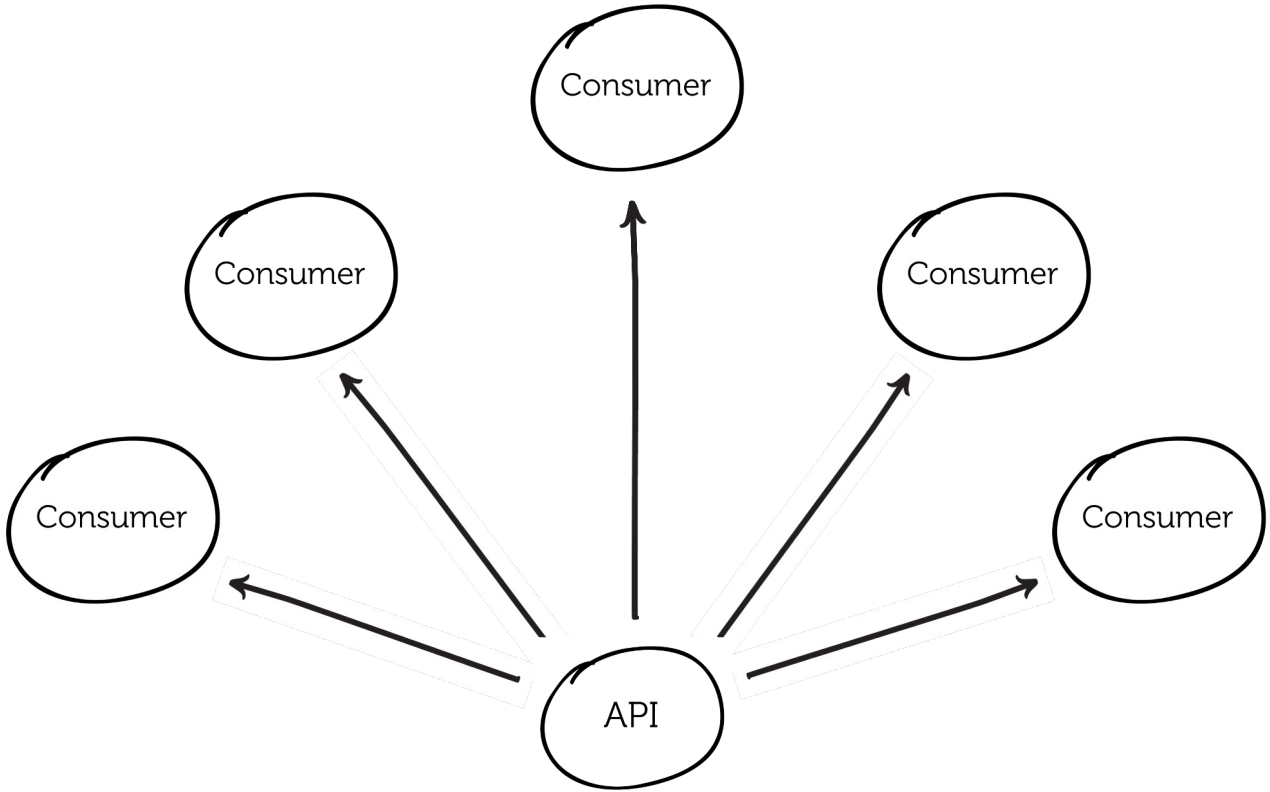
API Specifications such as OpenAPI

Then, we wouldn't need contract testing

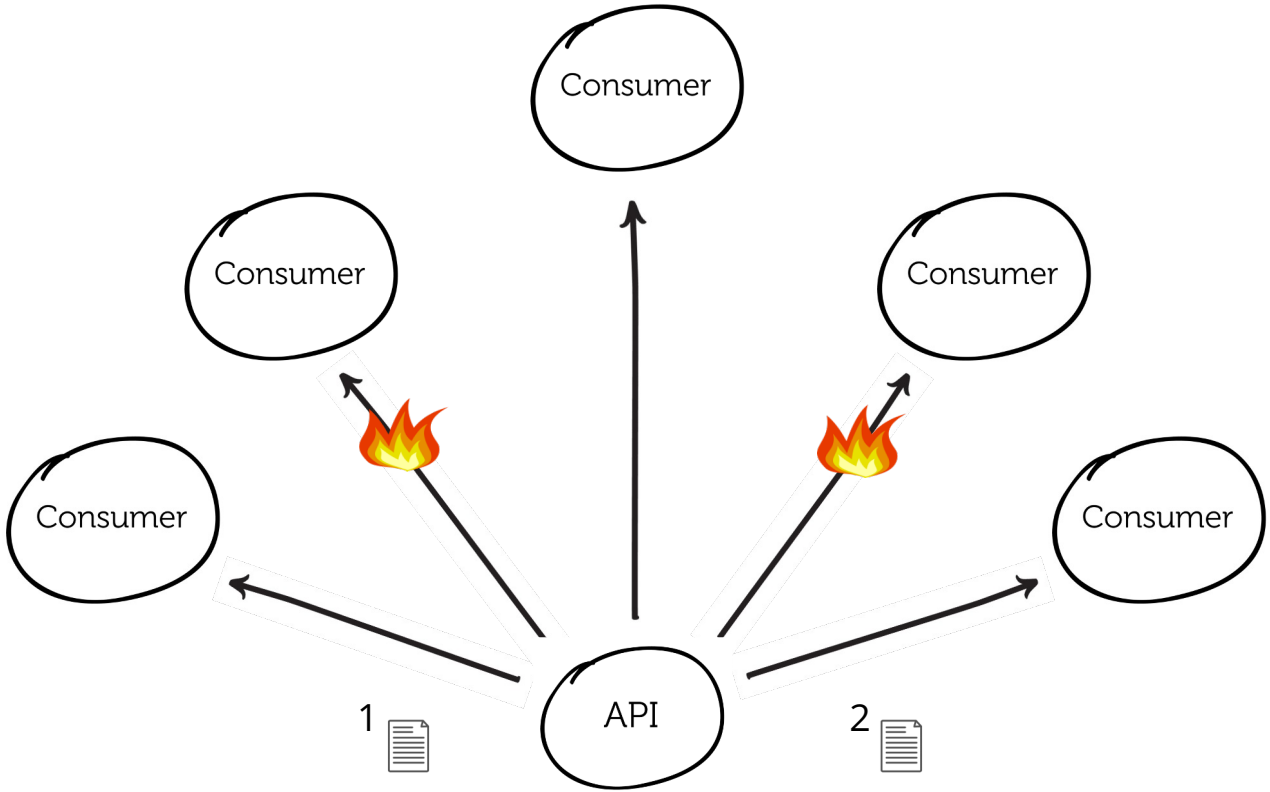
How to: Spec first development

1. Architect independent of teams postulate API requirements
2. Document perfect API (Swagger/OAS/WSDL etc.)
3. Create said API
4. Publish said document to consumers
5. Repeat steps 1-4


API


Consumer
Consumer
Consumer
Consumer
Consumer
API

Specification first design



Specification first design

Solved problems

- Good documentation
- Aides discoverability and communication between teams/organisations
- Clearer expectations on API

New problems

- Who is using my API?
- Requires diligence to ensure backwards compatibility
- Developers hate maintaining versioning
- Limited by expressiveness of specification (vague)
- = Hard to get 100% coverage (can only say “not incompatible”)

PACTFLOW
A SMARTBEAR COMPANY

If we just used

Interface Definition Languages

Then, we wouldn't need contract testing

Protobufs (+ Avro and Thrift)

How it aims to solve the problem

1. Designed with **schema evolution** in mind
2. In built forwards and backwards **compatibility**
3. Supports **codegen** to create server/client SDKs

> *“Protocol buffers provide a language-neutral, platform-neutral, extensible mechanism for serializing structured data in a forward-compatible and backward-compatible way. It’s like JSON, except it's smaller and faster, and it generates native language bindings.”*

Protobufs (+ Avro and Thrift)

Why it doesn’t

Colourless green ideas sleep furiously

Protobufs (+ Avro and Thrift)

Why it doesn’t

The curious case of missing merchant payments

> *I lost count of how many bugs we had at <redacted> because people where unaware of the default value behaviour*
>
> *- Poor soul responsible for finding the bug*

Protobufs (+ Avro and Thrift)

Why it doesn't

1. Message semantics
2. Optionals and defaults: a race to incomprehensible APIs
3. Managing breaking changes (e.g. Field descriptors)
4. Providing transport layer safety
5. Narrow type safety (strict encodings)
6. Loss of visibility into real-world client usage
7. Coordinating changes (forwards compatibility)

> *Forwards and backwards compatibility is not enforced: while forwards and backwards compatibility is a promise of Protobuf, actually maintaining backwards-compatible Protobuf APIs isn't widely practiced, and is hard to enforce.*
>
> https://docs.buf.build

Your Provider Contract

…is only one representation your API

> *With a sufficient number of users of an API, it does not matter what you promise in the contract: all observable behaviors of your system will be depended on by somebody*
>
> *- Hyrum's Law*

https://xkcd.com/1172/

The solution?

Consumer Driven Contract testing

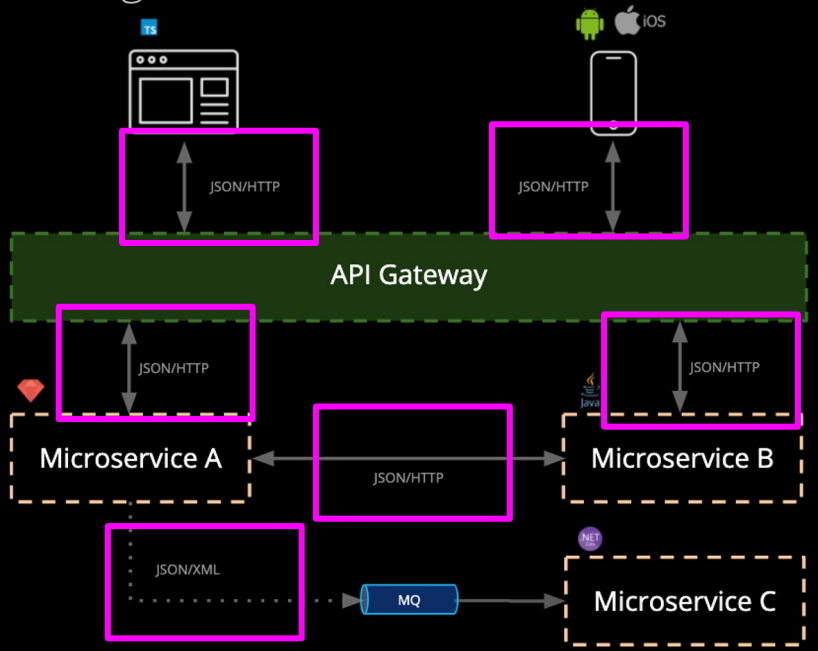
What is Contract Testing?

An alternative approach to API communication testing

Benefits:

- **Simpler** - test a single integration at a time
- No **dedicated test environments** - run on a dev machine
- Get **fast**, reliable feedback
- Tests that scale **linearly**
- **Deploy** services independently

It tracks these over time, enabling **evolution**



Contract Testing

A generalised approach to API communication testing

1. Record / replay	→	Tests the representative examples against the real provider
2. Specification by example	→	Reduces ambiguity, improves API comprehension
3. Service evolution	→	Time travel, by pairing application versions with known supported contracts
4. Transport concerns	→	Are encoded in the contract
5. Typed field matchers	→	Provide advanced narrow type system, including semantics (such as dates)
6. API surface area	→	Is made visible, by the sum of all of the consumer contracts

What is Pact?

Microservice testing made easy

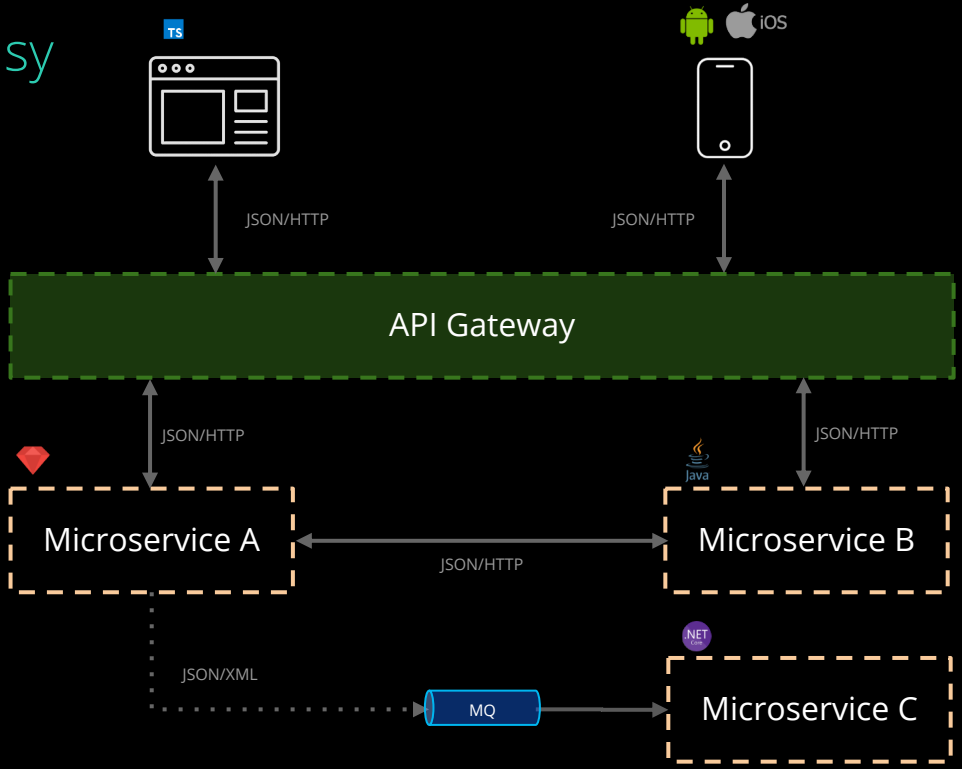
Pact is an Open Source, *consumer driven contract testing* tool that makes it easy to test microservices quickly, independently and release safely.

Use cases:

- Javascript web applications (e.g. React)
- Native mobile applications
- RESTful microservices with JSON and XML
- Asynchronous messaging (e.g. MQ)

Goals:

- Removing end-to-end integrated tests
- Reducing reliance on complex test environments



Open Source

...and in your preferred language

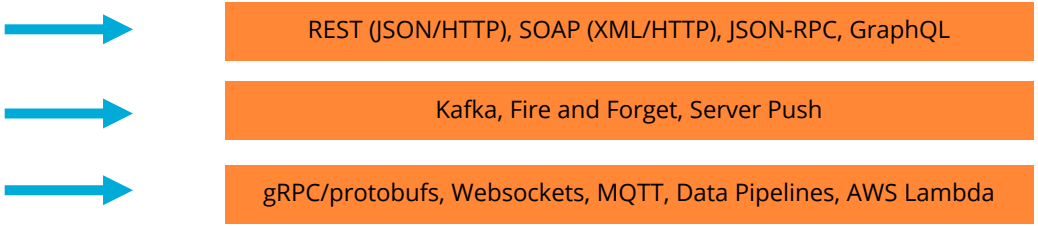
SMARTBEAR
PactFlow

Concepts

Interaction Types

Types of interactions:

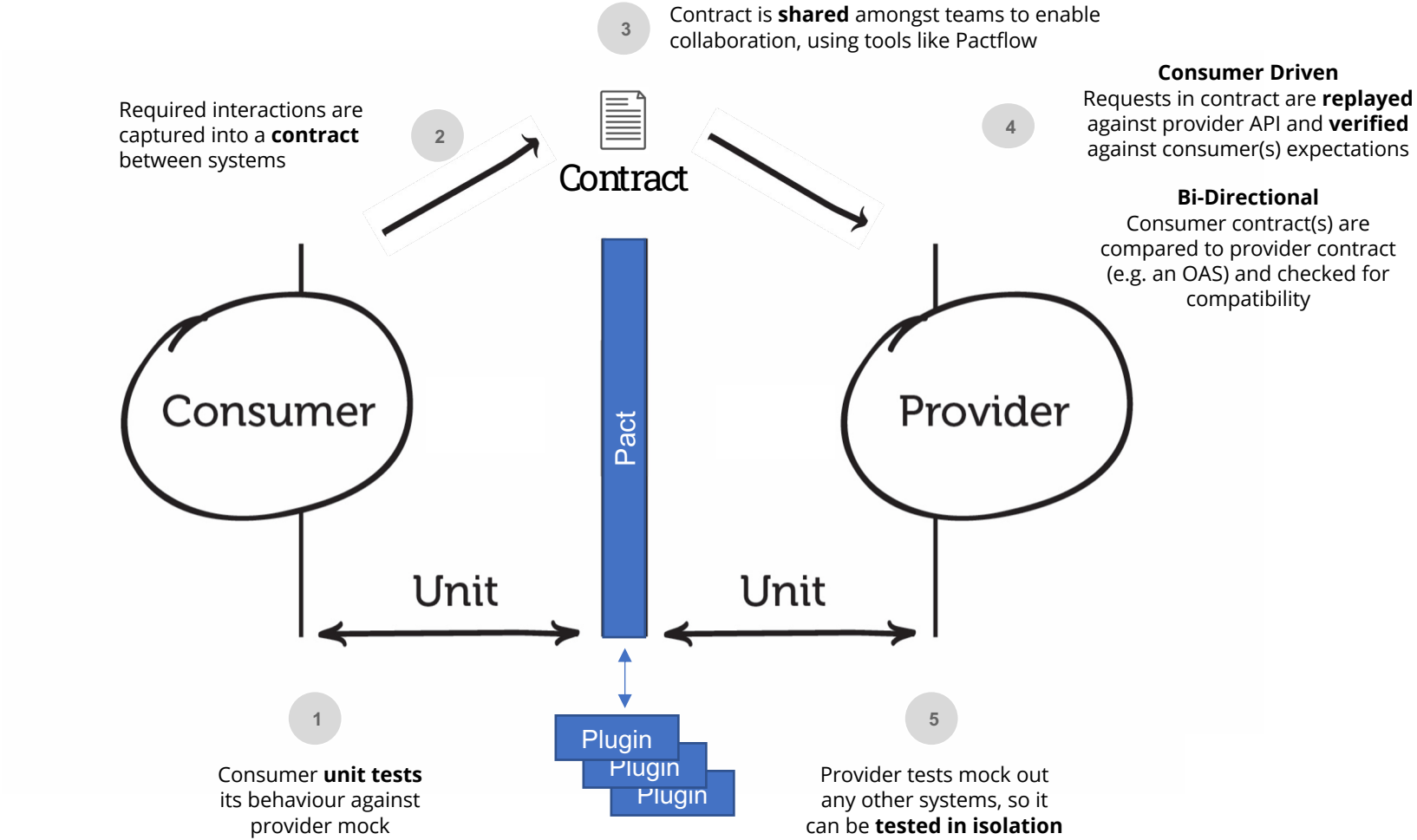
- Synchronous/HTTP
- Asynchronous/Messages
- Synchronous/Messages



By combining interaction types with the various Plugin capabilities, rich support for various frameworks and protocols emerge.

https://github.com/pact-foundation/pact-specification/tree/version-4#interactions

How Pact works



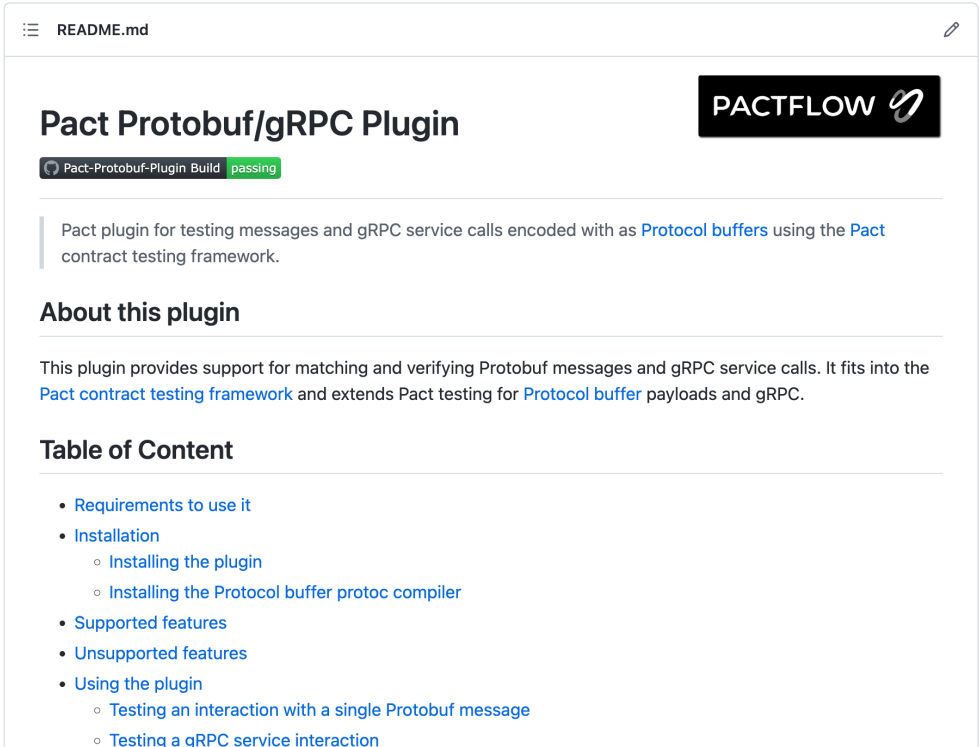
PACTFLOW
A SMARTBEAR COMPANY

Pact

Extend capabilities via Plugins

With plugins, you can create custom:

1. Transports (e.g. gRPC)
2. Protocols (e.g. protobufs)
3. Matching rules (e.g. semver strings)


README.md

PACTFLOW

Pact Protobuf/gRPC Plugin

Pact-Protobuf-Plugin Build passing

> Pact plugin for testing messages and gRPC service calls encoded with as Protocol buffers using the Pact contract testing framework.

About this plugin

This plugin provides support for matching and verifying Protobuf messages and gRPC service calls. It fits into the Pact contract testing framework and extends Pact testing for Protocol buffer payloads and gRPC.

Table of Content

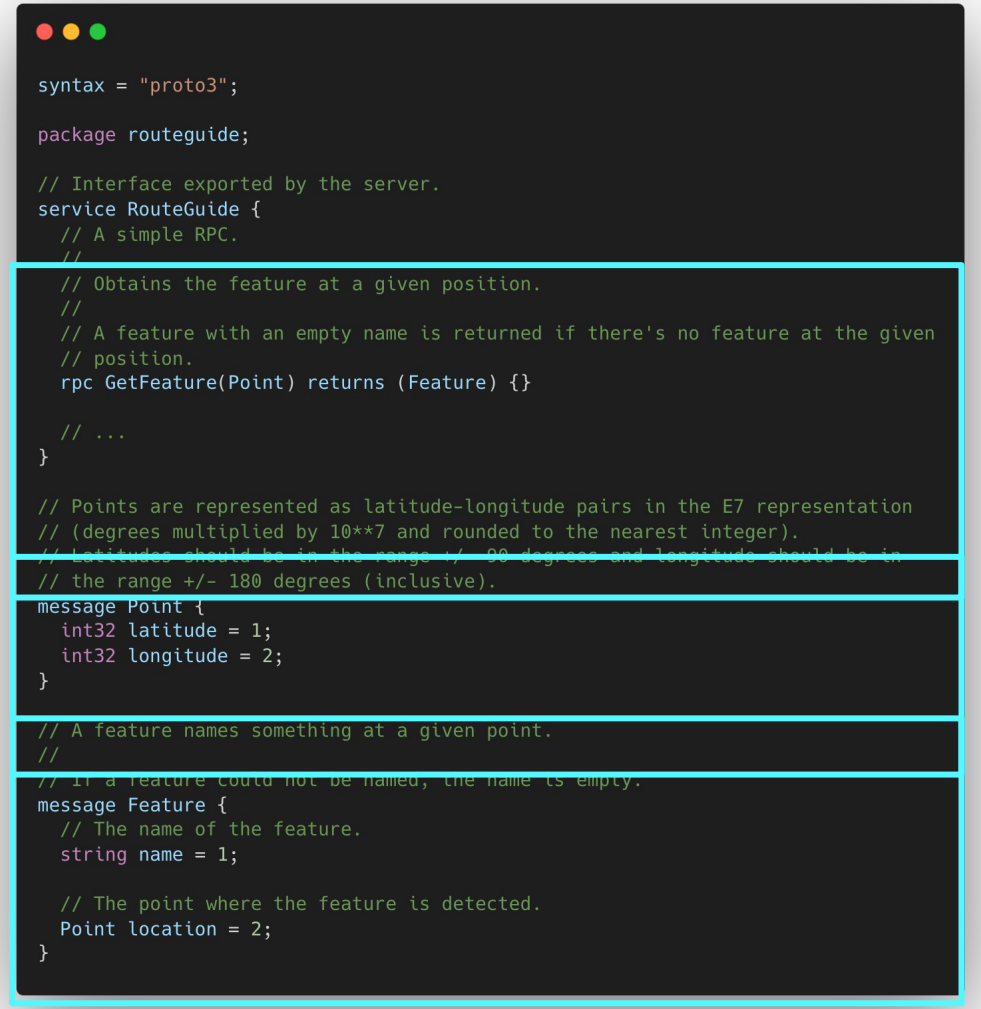
- Requirements to use it
- Installation
  - Installing the plugin
  - Installing the Protocol buffer protoc compiler
- Supported features
- Unsupported features
- Using the plugin
  - Testing an interaction with a single Protobuf message
  - Testing a gRPC service interaction

Demo

gRPC with Pact-Go

Scenario –

Route Guide



```proto
syntax = "proto3";

package routeguide;

// Interface exported by the server.
service RouteGuide {
  // A simple RPC.
  //
  // Obtains the feature at a given position.
  //
  // A feature with an empty name is returned if there's no feature at the given
  // position.
  rpc GetFeature(Point) returns (Feature) {}

  // ...
}

// Points are represented as latitude-longitude pairs in the E7 representation
// (degrees multiplied by 10**7 and rounded to the nearest integer).
// Latitudes should be in the range +/- 90 degrees and longitude should be in
// the range +/- 180 degrees (inclusive).
message Point {
  int32 latitude = 1;
  int32 longitude = 2;
}

// A feature names something at a given point.
//
// If a feature could not be named, the name is empty.
message Feature {
  // The name of the feature.
  string name = 1;

  // The point where the feature is detected.
  Point location = 2;
}
```

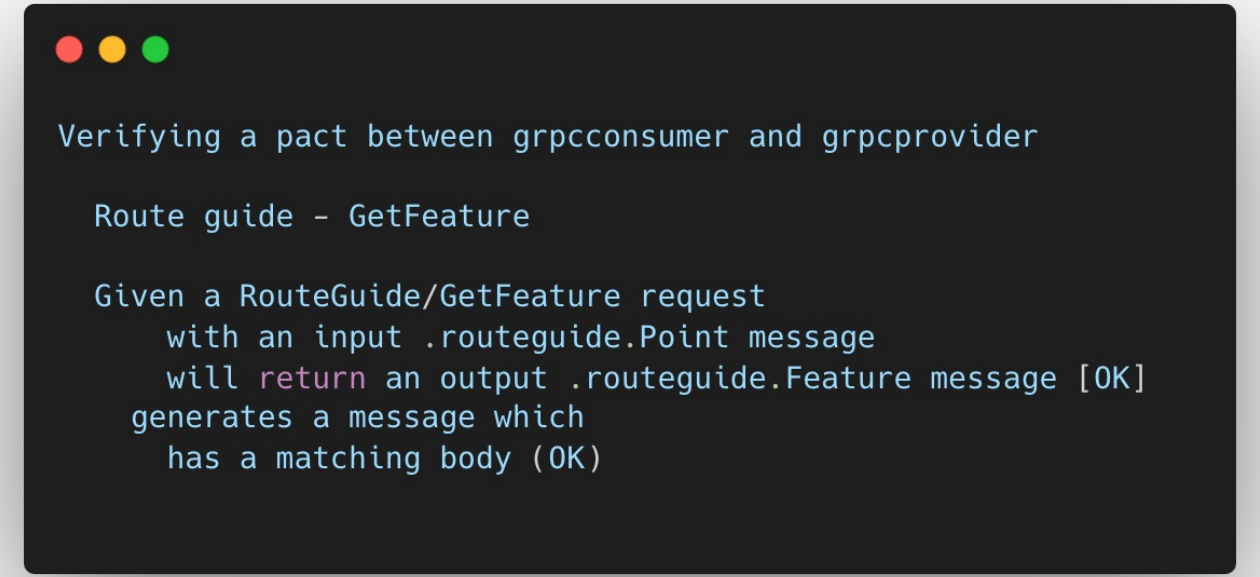
gRPC example - Consumer

```
func TestRouteServiceGetFeature(t *testing.T) {
    p, _ := message.NewSynchronousPact({...})

    grpcInteraction := `{
        "request": {
            "latitude": "matching(number, 180)",
            "longitude": "matching(number, 200)"
        },
        "response": {
            "name": "notEmpty('Big Tree')",
            "location": {
                "latitude": "matching(number, 180)",
                "longitude": "matching(number, 200)"
            }
        }
    }`

    err := p.AddSynchronousMessage("Route guide - GetFeature").
        Given("feature 'Big Tree' exists").
        [illegible]: "protobuf", Version: "0.1.7"}).
        WithContents(grpcInteraction, "application/protobuf").
        StartTransport("grpc", "127.0.0.1", nil).
        ExecuteTest(t, func(transport message.TransportConfig, m message.SynchronousMessage) error {
            feature := getFeature(transport.Port)
            assert.Equal(t, "Big Tree", feature.GetName())
            assert.Equal(t, int32(180), feature.GetLocation().GetLatitude())

            return nil
        })

    assert.NoError(t, err)
}
```

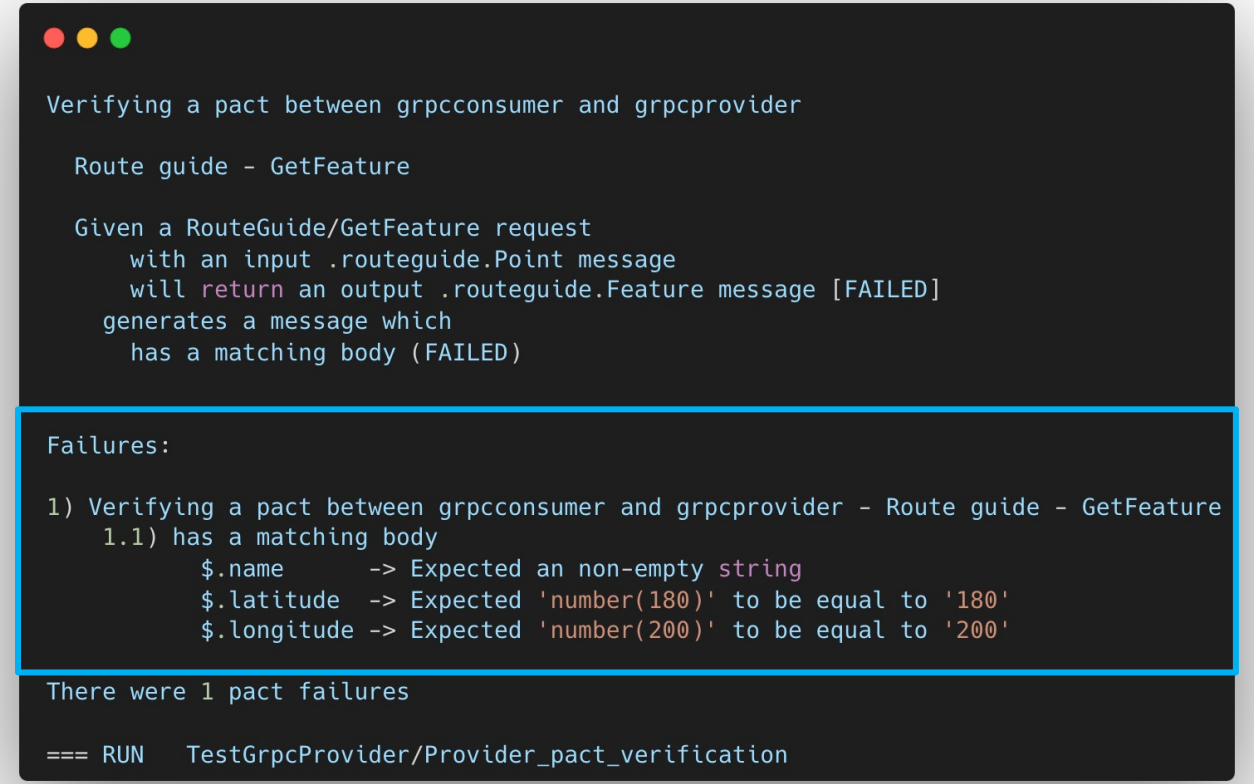
gRPC example - Provider

```
func TestGrpcProvider(t *testing.T) {
    go startProvider()

    verifier := provider.PluginVerifier{}

    err := verifier.VerifyProvider(t, provider.VerifyPluginRequest{
        ProviderAddress: "http://localhost:8222",
        Provider:        "grpcprovider",
        PactFiles: []string{
            filepath.ToSlash(fmt.Sprintf("%s/../pacts/grpcconsumer-grpcprovider.json", dir)),
        },
    })

    assert.NoError(t, err)
}

func startProvider() {
    lis, err := net.Listen("tcp", fmt.Sprintf("localhost:%d", 8222))
    if err != nil {
        log.Fatalf("failed to listen: %v", err)
    }
    var opts []grpc.ServerOption
    grpcServer := grpc.NewServer(opts...)
    pb.RegisterRouteGuideServer(grpcServer, server.NewServer())
    grpcServer.Serve(lis)
}
```

gRPC example – Provider Output



gRPC example – Bad Provider



Consumer	Version	Pact Published	Provider	Version	Pact verified
A	1	1 day ago	B	1	1 day ago
A	1	1 day ago	B	2	1 day ago
A	1	1 day ago	B	4	1 day ago

```
$ pact-broker can-i-deploy
    --app A --version 1
    --app B --version 4
```

```
> cd-test-endpoints@1.0.0 can-i-deploy /app
> ./scripts/can-i-deploy.sh


 -----> Checking if we're safe to deploy!
        have version: 1.0.0-ab976ccd9ccf3a85ec3bec4d808f3cc311306a3a
        running can-i-deploy
Computer says yes \o/

CONSUMER           | C.VERSION                       | PROVIDER           | P.VERSION                       | SUCCESS?
-------------------|---------------------------------|--------------------|---------------------------------|---------
CentreDiscoveryWeb | 0.13.0-ebe9050a61d9f49ad16b... | CentreDiscoveryApi | 1.0.0-ab976ccd9ccf3a85ec3be... | true

All verification results are published and successful


    ┌─────────────────────────────────┐
    │                                 │
    │   Update available 5.6.0 → 6.2.0   │
    │     Run npm i -g npm to update     │
    │                                 │
    └─────────────────────────────────┘

▸ Container exited normally
▸ Checking linked containers
▸ Uploading container logs as artifacts
▸ Cleaning up after docker-compose
▸ Running global pre-exit hook
▸ Stopping ssh-agent 7164
▸ Running local pre-exit hook
```

How can / get involved?

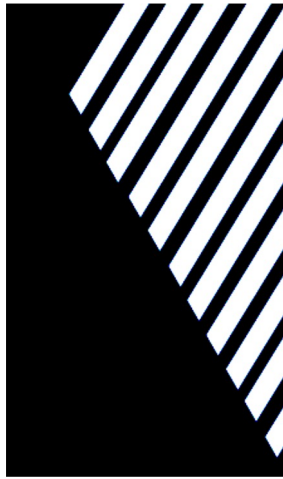
I want in! What can I do?

🔨 **Try a workshop** – Learn about Pact either on your own or with a friend/client

🔨 **Create a plugin** - either on your own or with a friend

🧪 **Try Pact out in your project** - blog/vlog about it, provide feedback (e.g. GH issue)

📣 **Amplify the message** to your networks - we'd love you to share on your social channels



WORKSHOP

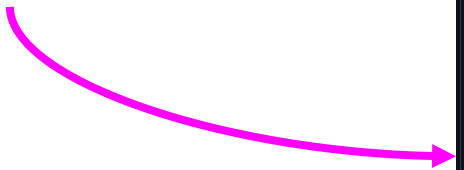
(https://github.com/pact-foundation/pact-workshop-go)

Participating

Getting the most out of the workshop

- Workshop arranged as a series of steps, each in a separate branch
- We will progress each step as a group, but you are encouraged to explore as we go
- Q&A will be available at the end of each step
- Each step has specific learning objectives

✨ **Follow the README!**



Pact Go workshop

Introduction

This workshop is aimed at demonstrating core features and benefits of contract testing with Pact.

Whilst contract testing can be applied retrospectively to systems, we will follow the consumer driven contracts approach in this workshop - where a new consumer and provider are created in parallel to evolve a service over time, especially where there is some uncertainty with what is to be built.

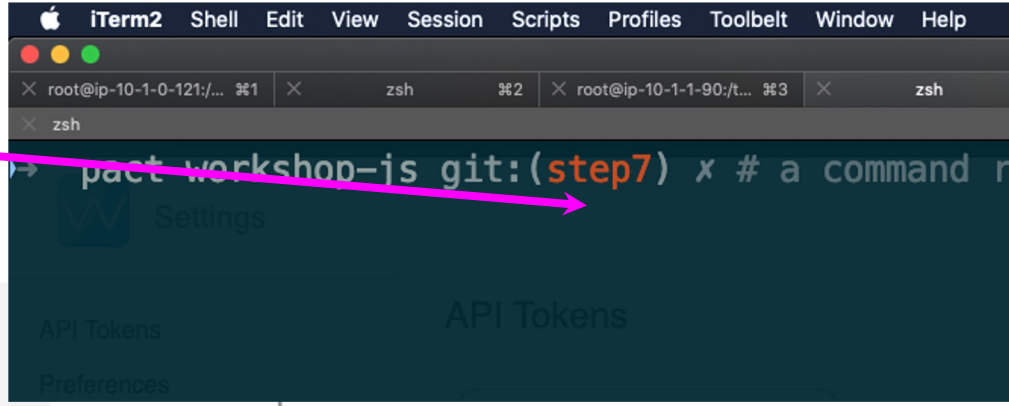
This workshop should take from 1 to 2 hours, depending on how deep you want to go into each topic.

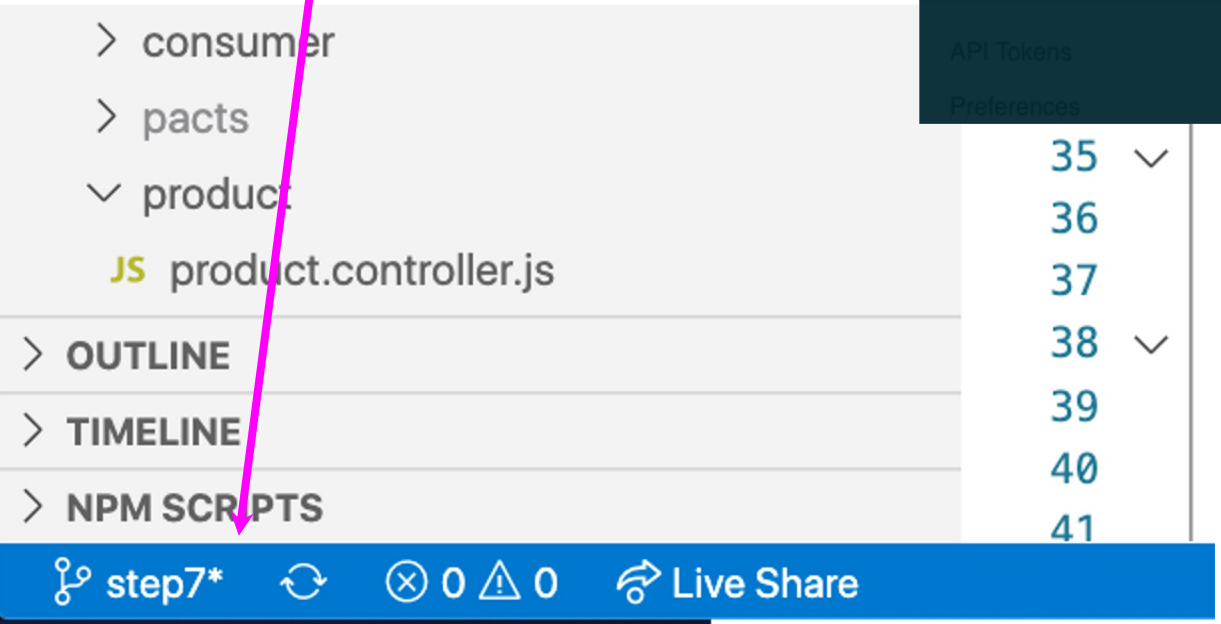
Workshop outline:

- step 1: **create consumer**: Create our consumer before the Provider API even exists
- step 2: **unit test**: Write a unit test for our consumer
- step 3: **pact test**: Write a Pact test for our consumer
- step 4: **pact verification**: Verify the consumer pact with the Provider API
- step 5: **fix consumer**: Fix the consumer's bad assumptions about the Provider
- step 6: **pact test**: Write a pact test for `404` (missing User) in consumer
- step 7: **provider states**: Update API to handle `404` case
- step 8: **pact test**: Write a pact test for the `401` case
- step 9: **pact test**: Update API to handle `401` case
- step 10: **request filters**: Fix the provider to support the `401` case
- step 11: **pact broker**: Implement a broker workflow for integration with CI/CD

Participating

Where am I?





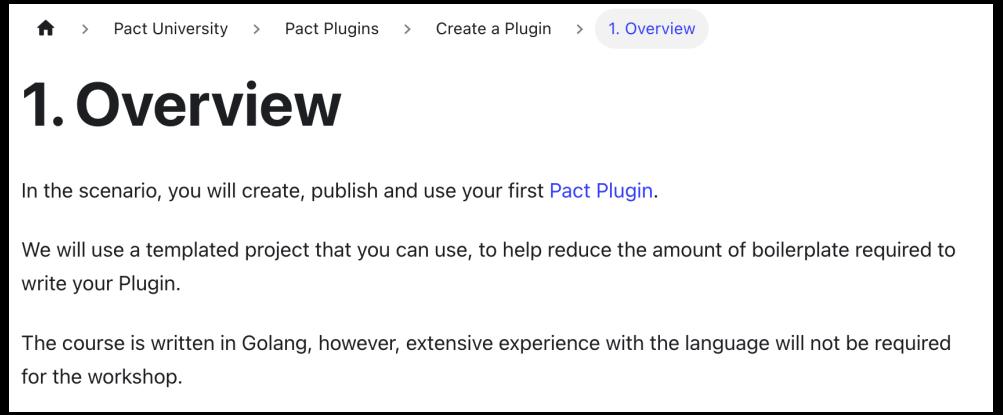
PACTFLOW
A SMARTBEAR COMPANY

Think you can build a better Interface description language?

Designed your own protocol?

Why not build your own Pact plugin?

Our workshop is written in GoLang



1. Understand the basics of the Pact Plugin framework
2. Learn the basics of plugins by creating a simple Content Matcher plugin
3. Publish the plugin to GitHub so that it can be used by others
4. Use the plugin in a project

https://docs.pact.io/plugins/workshops/create-a-plugin/intro

Try Pact-Go in your project

Installation

```
# install pact-go as a dev dependency
go get github.com/pact-foundation/pact-go/v2@2.x.x

# NOTE: If using Go 1.19 or later, you need to run go install instead
# go install github.com/pact-foundation/pact-go/v2@2.x.x

# download and install the required libraries. The pact-go will be install
pact-go -l DEBUG install

# 🚀 now write some tests!
```

https://github.com/pact-foundation/pact-go

What's in it for me?

Why should *I* get involved?

🔥 Be on the bleeding edge

❤️ Benefits of participating in an Open Source project

⚡ Contributing to the future state of the leading contract testing framework

🎉 Building your own profile + shout out via our blog and social media platforms

😎 Get some swag! You'll find stickers on the table

THANK YOU

Get in touch

Join our community 👉 **slack.pact.io**

Say hi 👋 in the **#pact-go** channel