# Microservices Matchmaking

Pact Testing between Spring Boot and gRPC services?


## Stelios Gkiokas

Principal Software Engineer in Test, Agile Actors



## Gregory Savvidis

Senior Software Engineer in Test, Agile Actors

# Agenda

## PACT Testing
Exploration of consumer-driven contract testing and its benefits for microservices

01

## gRPC Basics
Basic gRPC concepts, Protocol Buffers, and HTTP/2 transport mechanics

02

## REST vs gRPC
Side-by-side analysis of performance, payload sizes and architectural suitability

03

## Demo
Demonstrate all concepts with actual coding exmaples

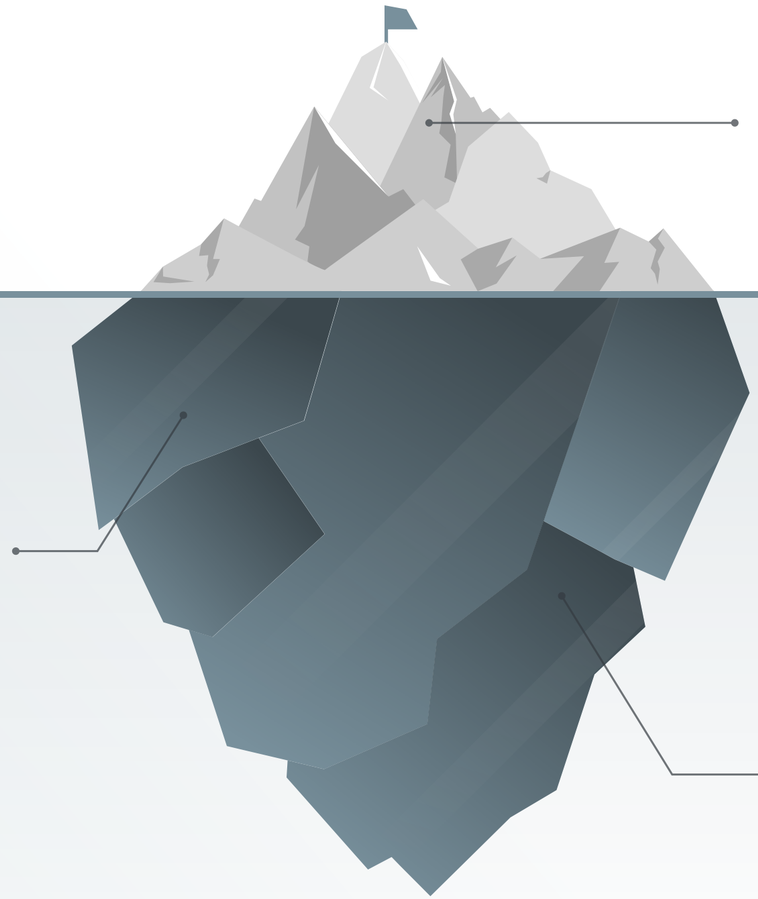
04

## Summary
Concluding recommendations for integrating these tools into a robust development lifecycle

05

# Understanding PACT Contract Testing

# The Concept of the Contract



## The Pact File

The formal agreement that bridges Consumer and Provider applications, ensuring compatibility (**The Connection**).

## Visible Interactions

API endpoints and data structures we see in documentation (**The Tip**).

## Hidden Expectations

The logic and behavioral requirements expected by the Consumer (**The Bulk**).

# The PACT Testing Workflow

### Consumer

The application defining the expected response from the Provider

### Publish Pact

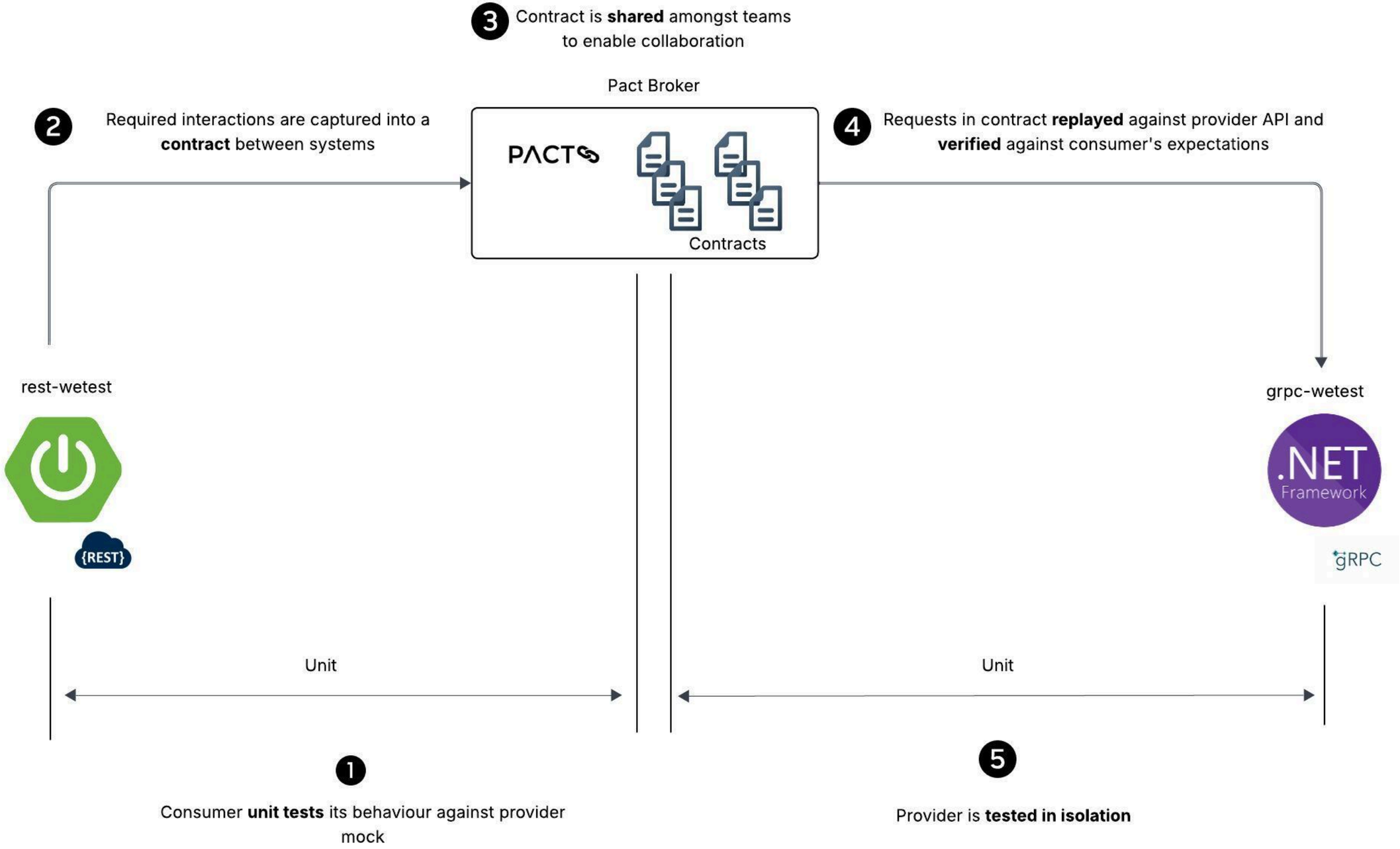
The generated JSON 'contract' file is uploaded to the **Pact Broker from the Consumer**

### Provider

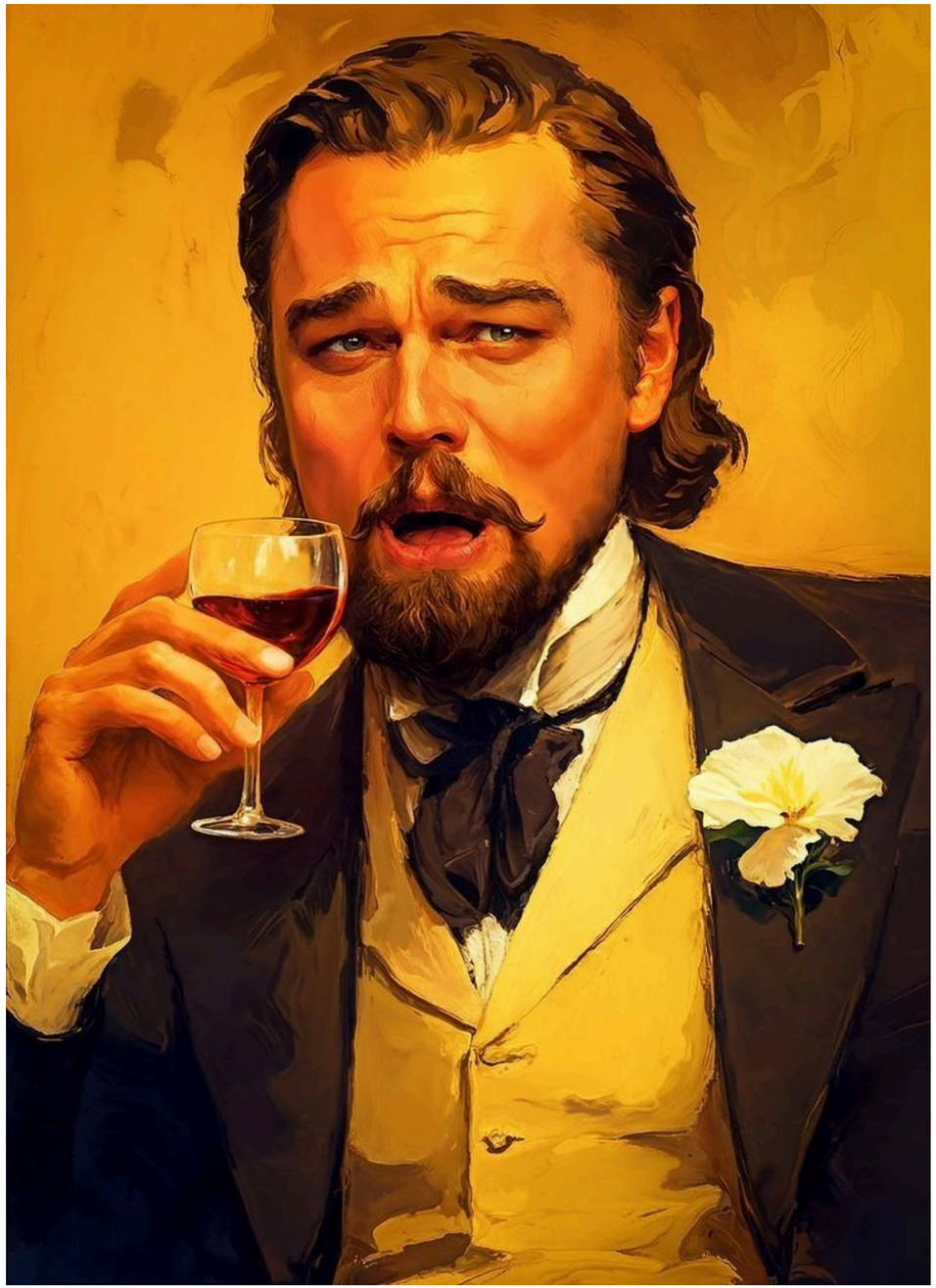
Application adhering to PACT Broker contracts defined by a Consumer

### *can-i-deploy*

A check to ensure the specific versions are compatible before release


3
Contract is **shared** amongst teams to enable collaboration
Pact Broker
2
Required interactions are captured into a **contract** between systems
PACT
Contracts
4
Requests in contract **replayed** against provider API and **verified** against consumer's expectations
rest-wetest
{REST}
grpc-wetest
.NET Framework
gRPC
Unit
Unit
1
Consumer **unit tests** its behaviour against provider mock
5
Provider is **tested in isolation**

# This is easy, meh...



# The problem

"The other service will send me what I expect”

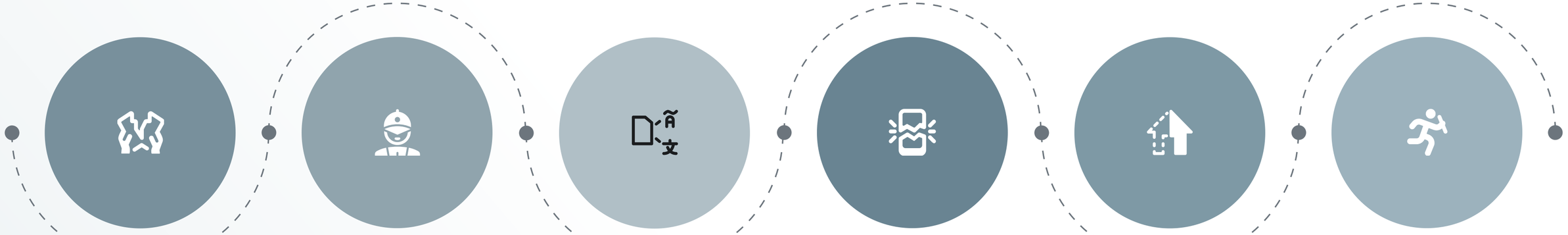
No shared schema enforcement

Different tech stacks and protocols

Unreliable integrations

Async evolvement of services

Speed of delivery



**81%**

of organizations operate in a multi-protocol environment

**57%**

use more than 3 protocols

# Why PACT?

## Shift Left Approach

Enable testing earlier in the SDLC to detect issues before deployment

01

## Independent Evolution

Allows services to evolve and deploy independently without impacting other services

02

## PACT Consumer Contracts

Define explicit PACT Consumer requirements to ensure robust service contracts

03

## Faster Feedback Loops

Accelerate development cycles with immediate validation through automated testing

04

# gRPC in 1'

## RPC

A protocol that lets a program execute a function on another machine as if it were local

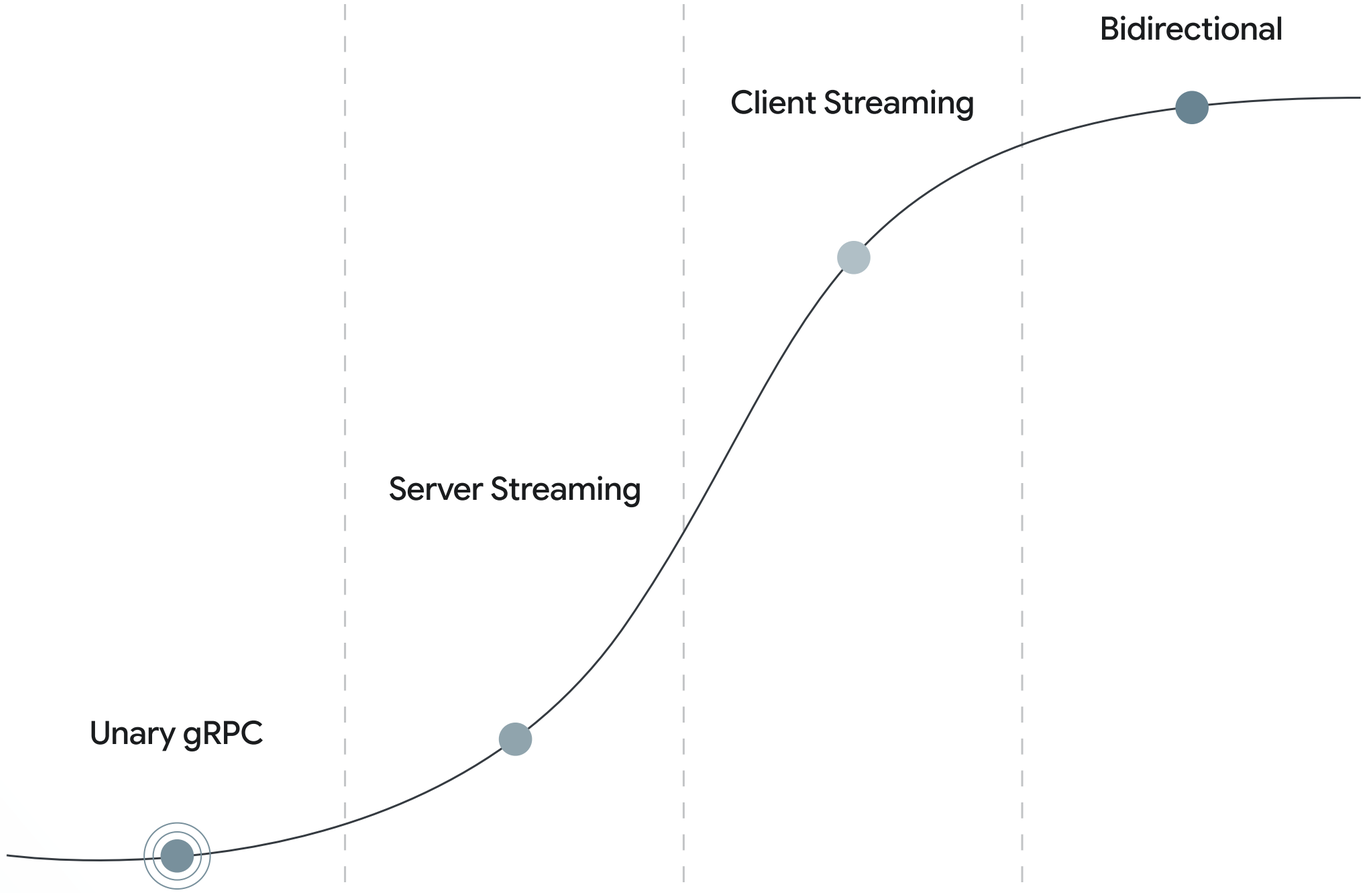
## gRPC

A high-performance RPC framework using HTTP/2 and Protocol Buffers for efficient, strongly-typed communication between services


## Descriptor set

A compiled bundle of Protocol Buffer (.proto) definitions describing services, methods, and message types—used for reflection, tooling or dynamic clients

# Communication Patterns in gRPC




The classic client-request, server-response pattern, similar to REST

Client sends one request, server sends a stream of many messages

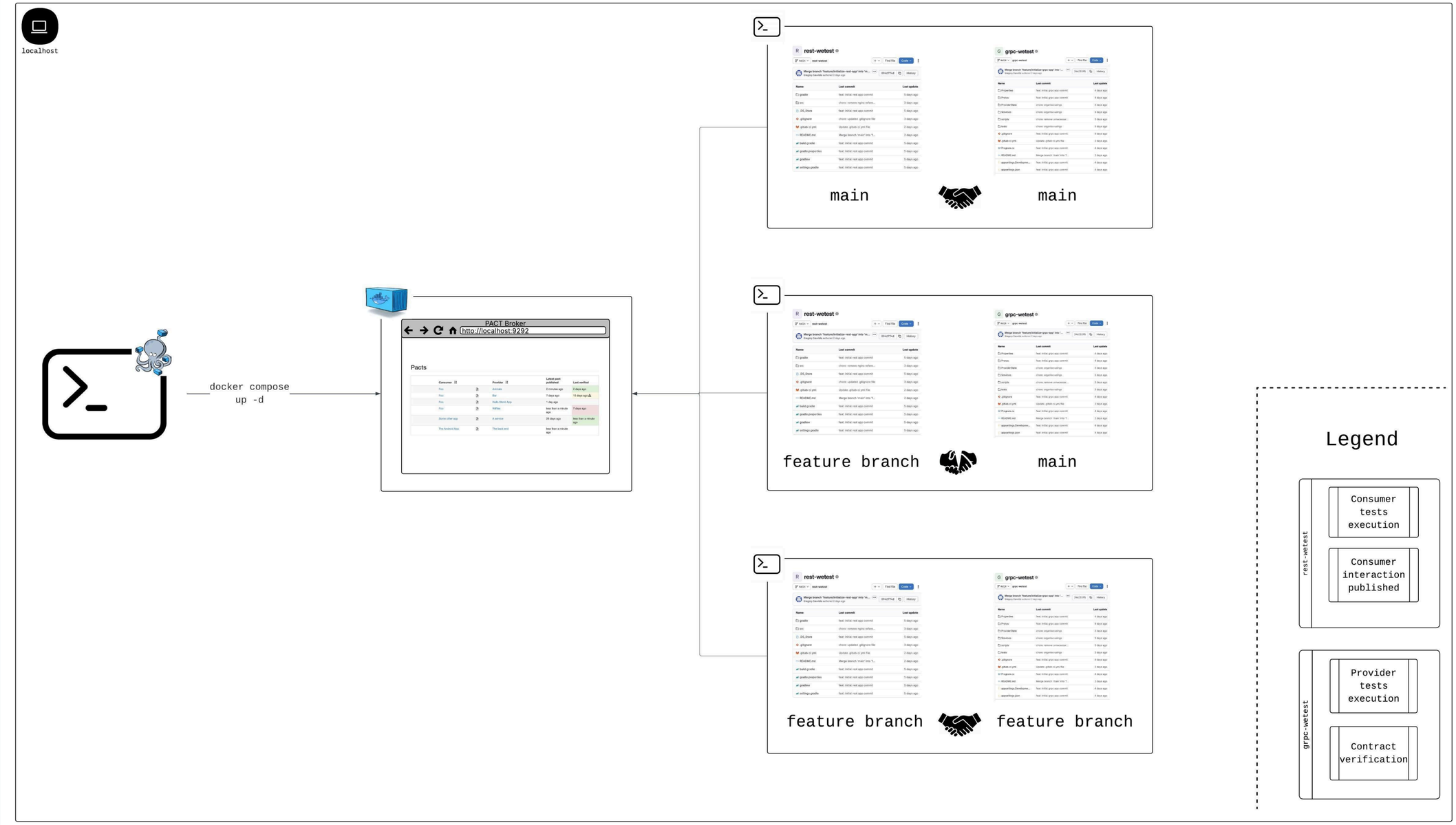
Client sends a stream of messages, server replies with one response

Both parties send a stream of messages simultaneously over one connection

# gRPC vs. REST

# The real problem

| Feature | REST -> REST | REST -> gRPC |
|---|---|---|
| Protocol Complexity | HTTP/1.1, JSON | HTTP/2, Protobuf |
| Pact Support | Native JSON DSL | Strict (.proto files) |
| State Handling | Simple JSON fixtures | **/_pact/state** + in-memory state |
| Schema Enforcement | JSON keys validated | Descriptor sets + exact proto messages |
| Numeric and OneOf Fields | Optional checks | Explicit checks required |
| **Setup/Debugging Effort** | **Low** | **High** |



# Demo

1

Always keep your descriptor sets in sync with the consumer .proto files

---

2

Provider state endpoint is essential for dynamic responses

---

3

Protobuf payloads must match PACT expectations, including encoding and content type

---

4

PACT Plugin version matters

# Find us on LinkedIn


## Stelios Gkiokas

Principal Software Engineer in Test, Agile Actors

## Gregory Savvidis

Senior Software Engineer in Test, Agile Actors

# Thanks !