Deploy with Confidence!

Making your micro-services pay nicely together with Pact and Consumer Driven Contracts

Ronald Holshausen
rholshausen@dius.com.au
@RHolshausen

DIUS
WHERE IDEAS ARE ENGINEERED

Picture the scene

Monday Morning

9AM

You just got in to work

You haven’t had a coffee

You get pulled into a meeting

The weekends security roll-out failed

It broke everything in production

Due to a API change

Which you did not implement

And when you ask about it

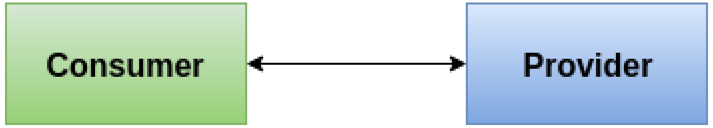
Didn’t you read the yammer post?

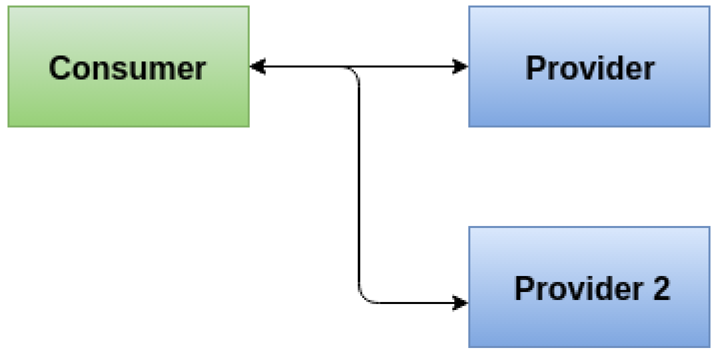
This is a real story

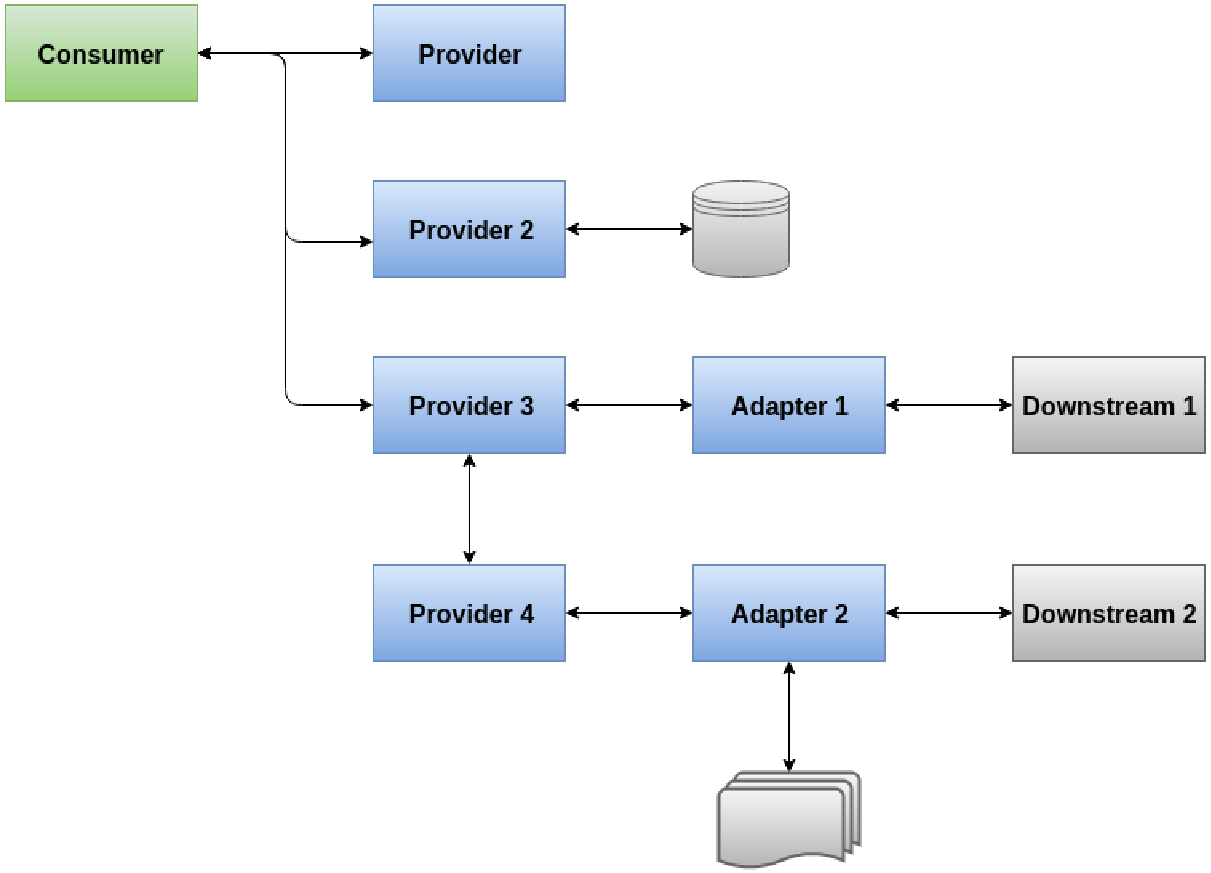
Kind of

`What we got here is a failure to communicate`

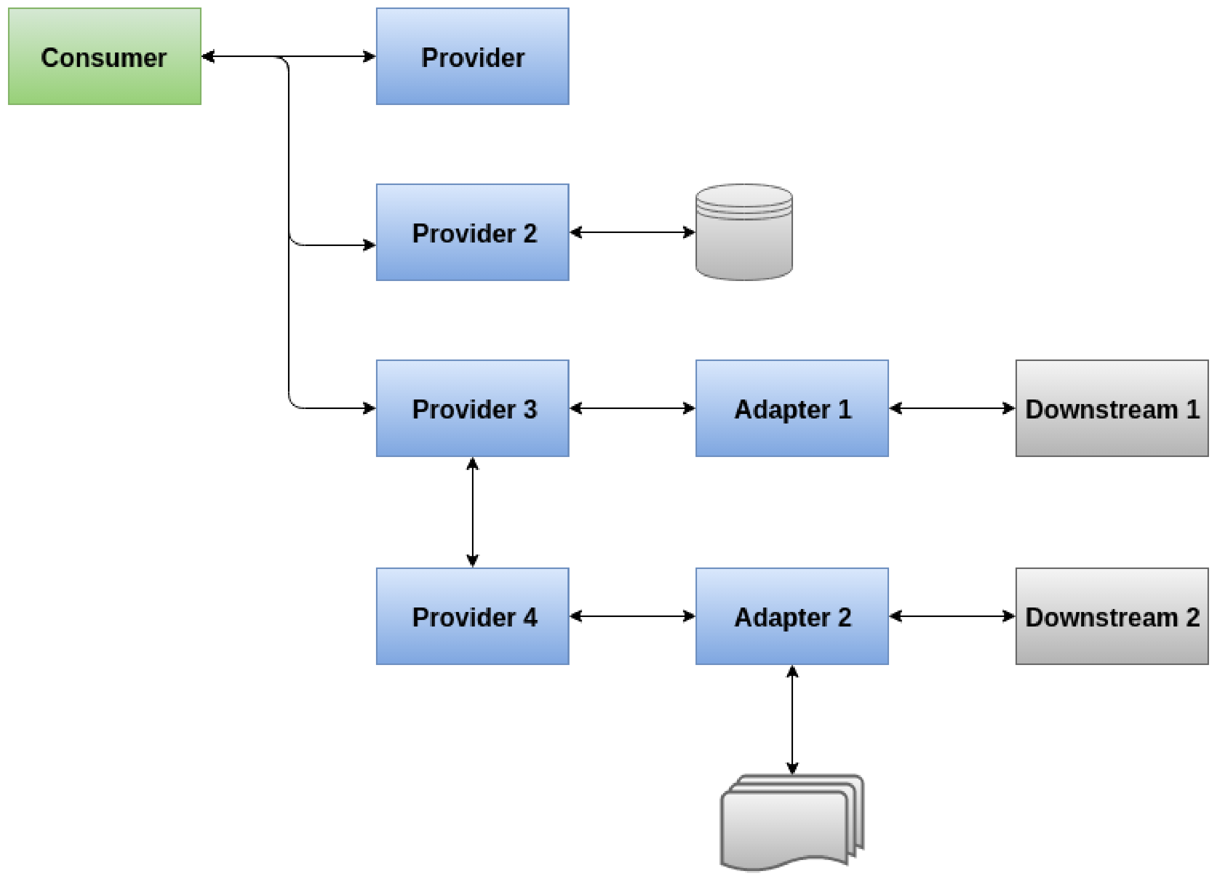
Everyone's moving to micro-services


Consumer
Provider


Consumer
Provider
Provider 2

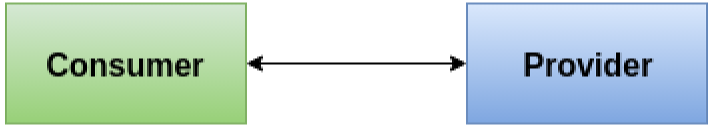

Consumer
Provider
Provider 2
Provider 3
Adapter 1
Downstream 1
Provider 4
Adapter 2
Downstream 2

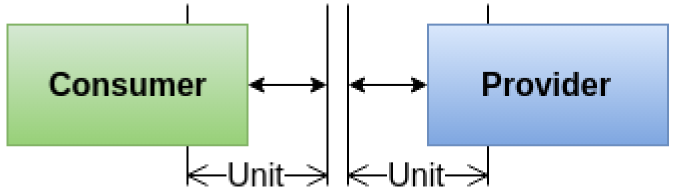
How do you test this?

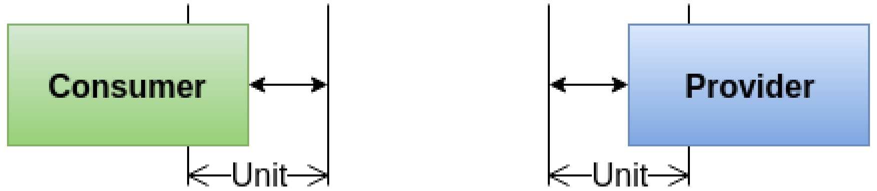

Consumer
Provider
Provider 2
Provider 3
Adapter 1
Downstream 1
Provider 4
Adapter 2
Downstream 2

`Integration tests are a scam`

J.B. Rainsberger

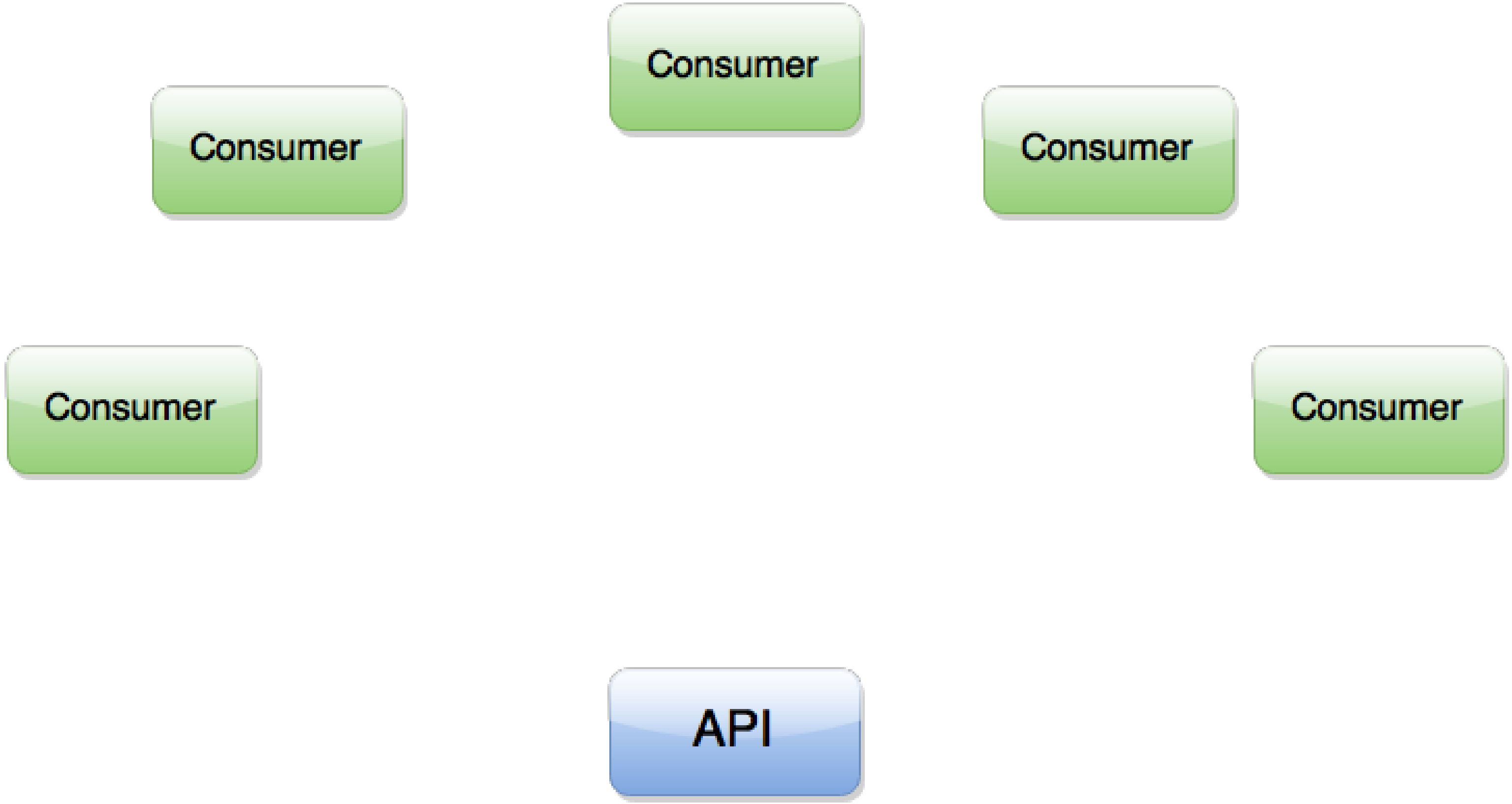

Consumer
Provider

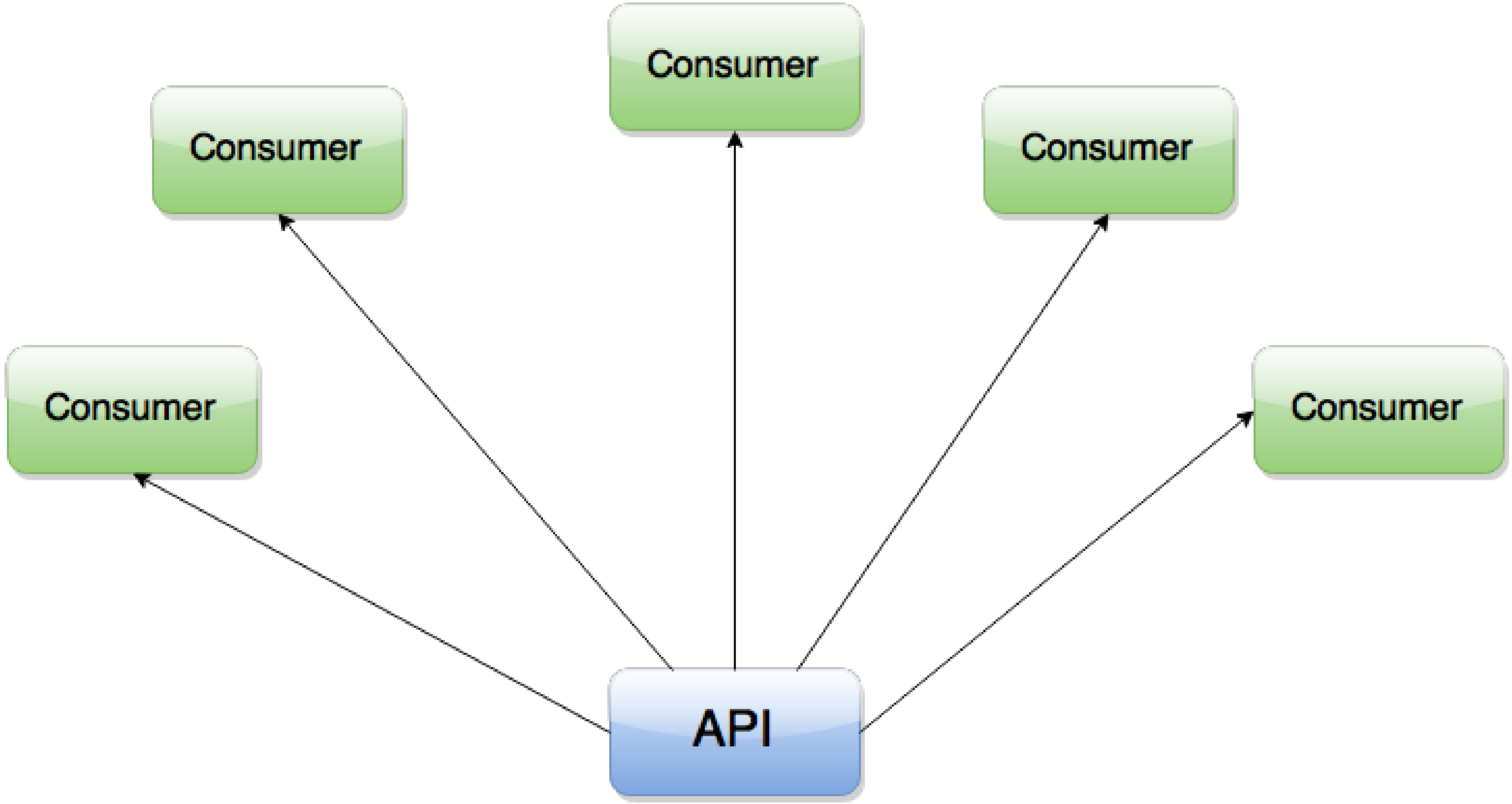

Consumer
Provider
Unit
Unit

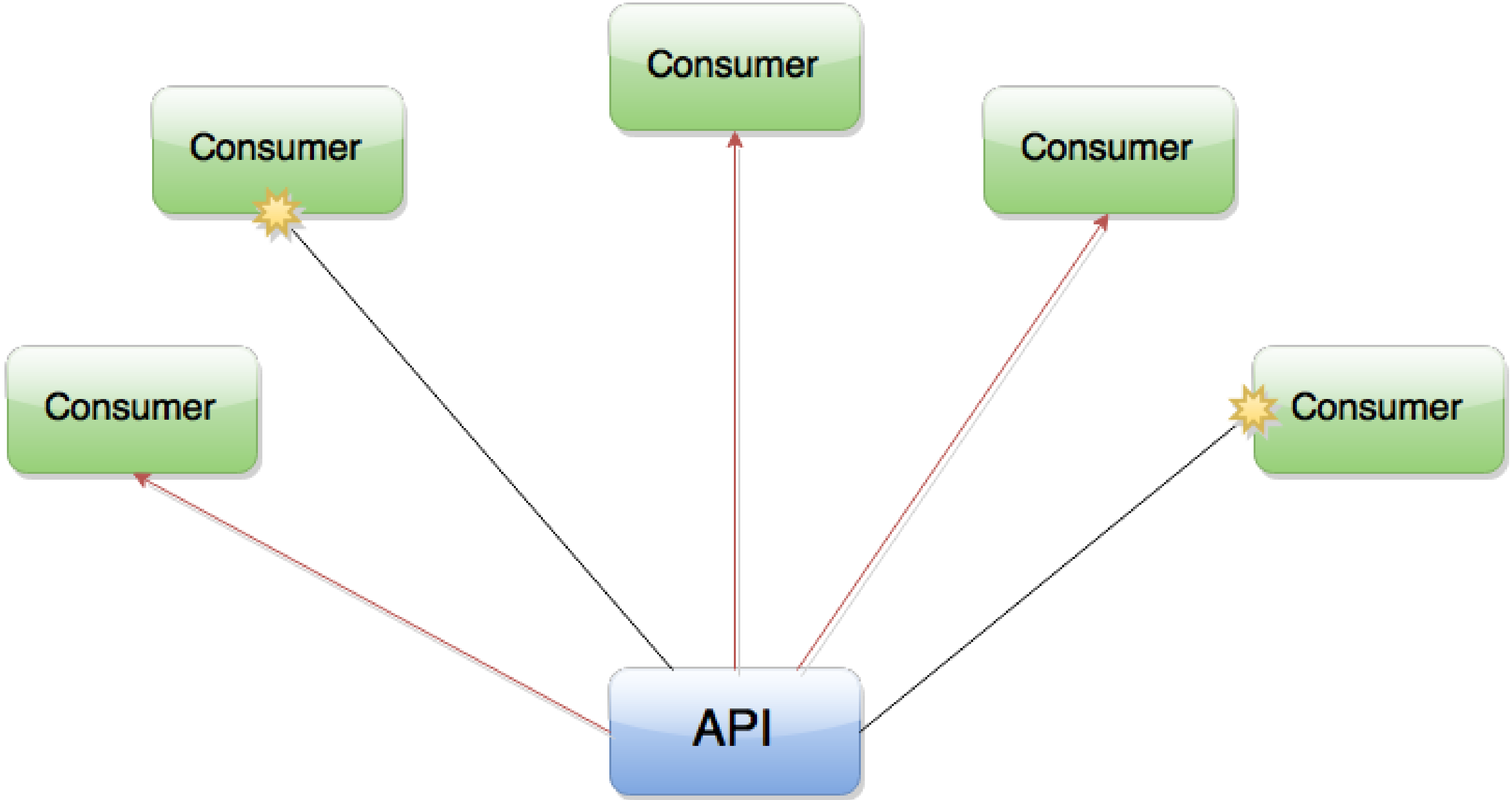

Consumer
Unit
Unit
Provider

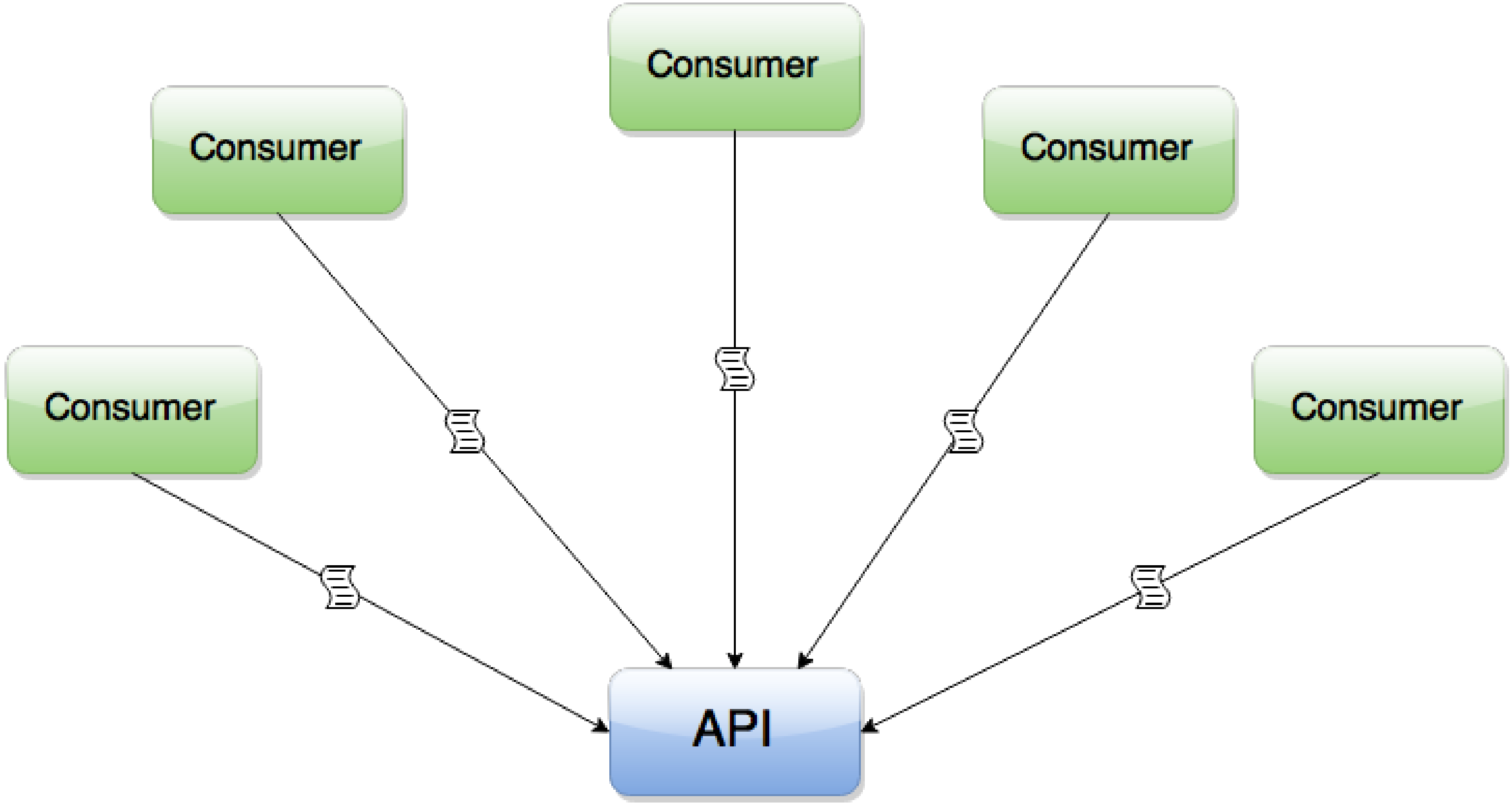
Consumer Driven Contracts

API


Consumer
Consumer
Consumer
Consumer
Consumer
API


Consumer
Consumer
Consumer
Consumer
Consumer
API


Consumer
Consumer
Consumer
Consumer
Consumer
API


Consumer
Consumer
Consumer
Consumer
Consumer
API

Benefits?

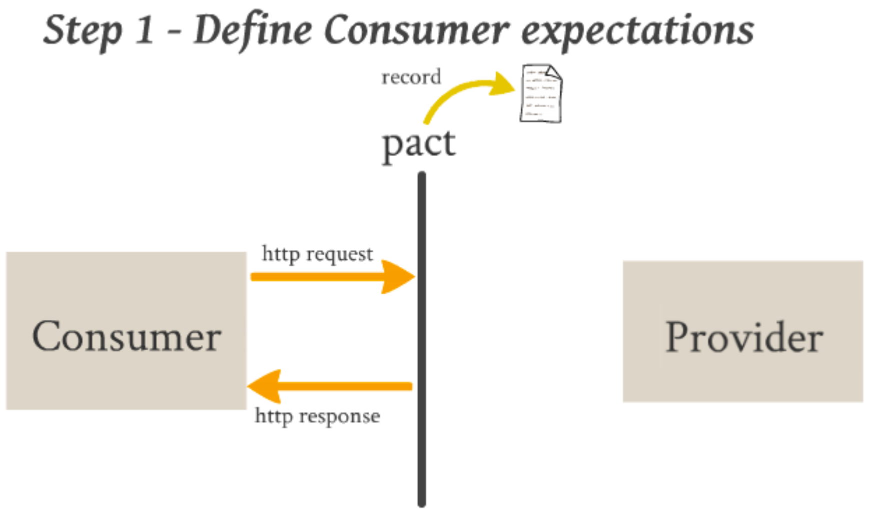
You’ll know when you break a consumer

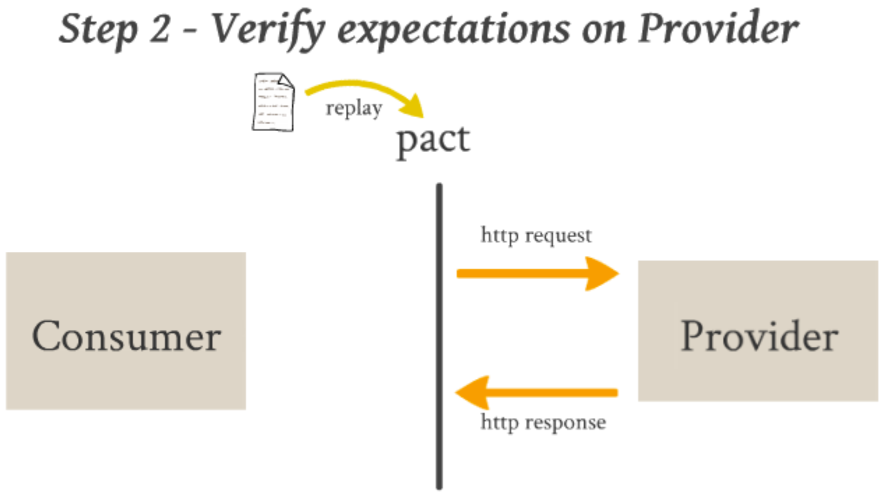
You have a form of documentation

Pact

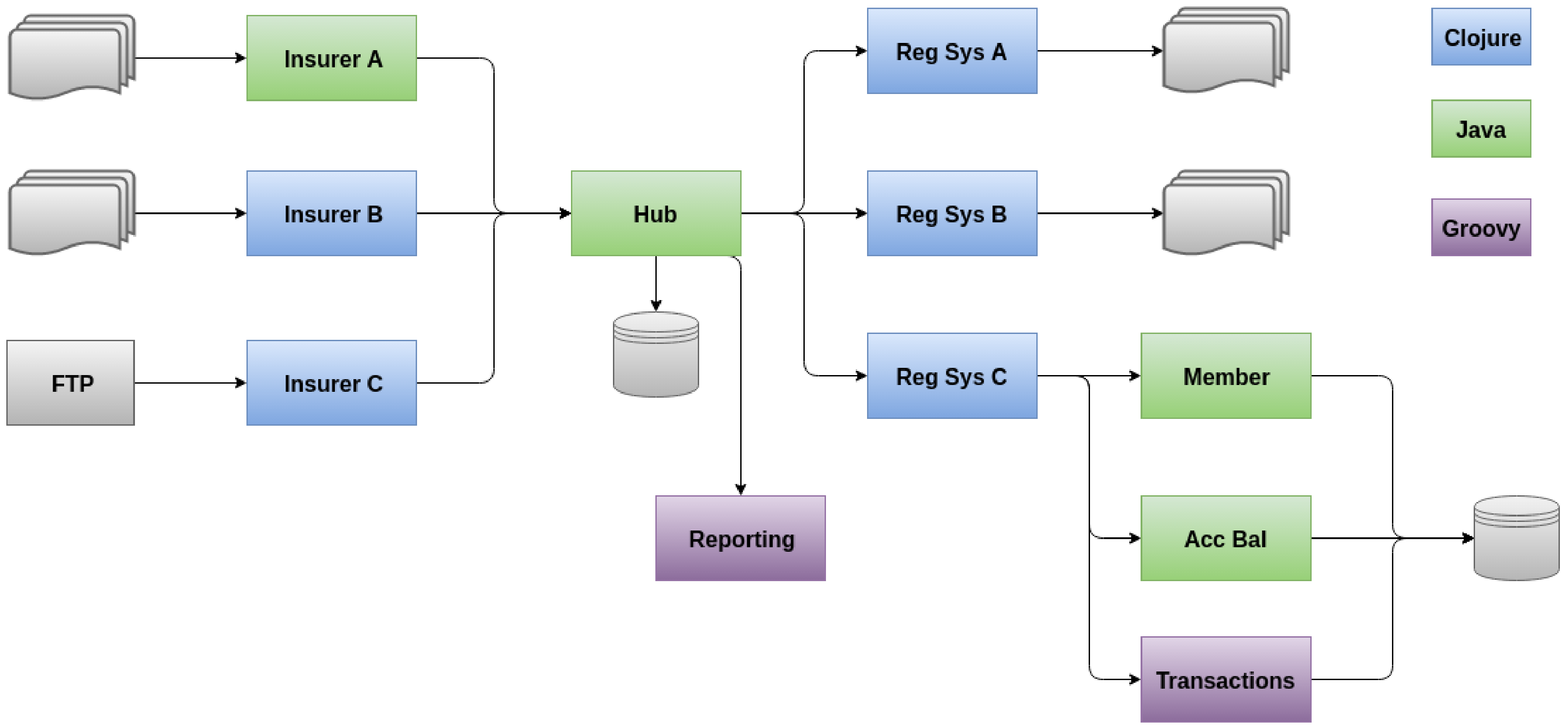
www.pact.io

Evolved from combining these two principals


Step 1 - Define Consumer expectations
record
pact
http request
Consumer
http response
Provider

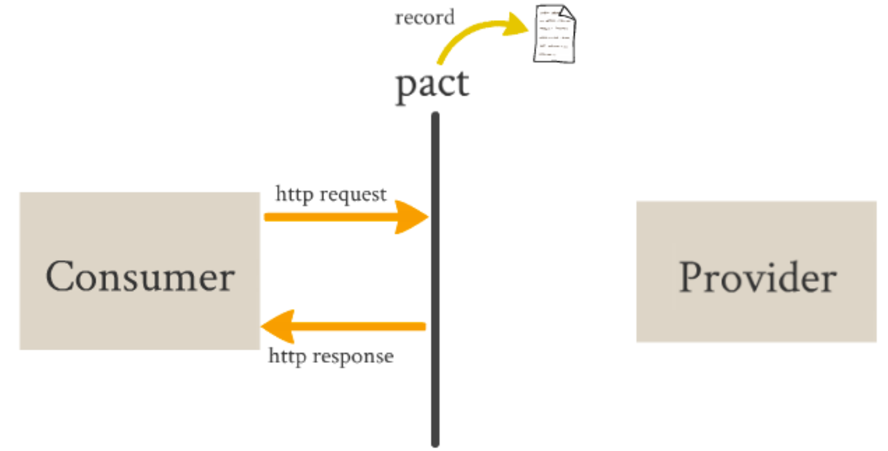

Step 2 - Verify expectations on Provider
replay
pact
http request
Consumer
Provider
http response

NO

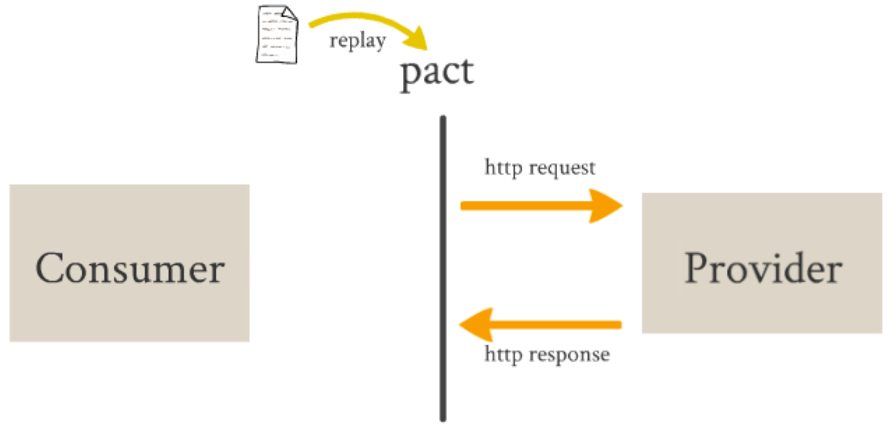

Insurer A
Insurer B
FTP
Insurer C
Hub
Reporting
Reg Sys A
Reg Sys B
Reg Sys C
Member
Acc Bal
Transactions
Clojure
Java
Groovy

So how does it work?

Step 1 - Define Consumer expectations



Step 2 - Verify expectations on Provider



Start with a consumer test

Given “some state exists”

When I Receive “a GET request for data”

With “these headers and query”

Respond with “200 OK”

And “this data in the body”

Given **“User A exists”**

When I Receive **“a GET request for user A”**

Respond with **“200 OK”**

And **“User A’s details in the body”**

Given “User A does not exist”

When I Receive “a GET request for user A”

Respond with “404 Not Found”

Ruby

JVM (Java, Groovy, Scala, Clojure)

.Net (1.1)

JavaScript/Node.js (Mocha, Jasmine)

Rust/Go/Python/PHP

Java Example

```java
@Rule
public PactProviderRule provider = new PactProviderRule("test_provider", "localhost", 8080, this);

@Pact(provider="test_provider", consumer="test_consumer")
public PactFragment createFragment(PactDslWithProvider builder) {
    Map<String, String> headers = new HashMap<String, String>();
    headers.put("testreqheader", "testreqheadervalue");

    return builder
        .given("test state")
        .uponReceiving("ExampleJavaConsumerPactRuleTest test interaction")
            .path("/")
            .method("GET")
            .headers(headers)
        .willRespondWith()
            .status(200)
            .headers(headers)
            .body("{\"responsetest\": true, \"name\": \"harry\"}")
        .given("test state")
        .uponReceiving("ExampleJavaConsumerPactRuleTest second test interaction")
            .method("OPTIONS")
            .headers(headers)
            .path("/second")
            .body("")
        .willRespondWith()
            .status(200)
            .headers(headers)
            .body("")
        .toFragment();
}

@Test
@PactVerification("test_provider")
public void runTest() throws IOException {
    Assert.assertEquals(new ConsumerClient("http://localhost:8080").options("/second"), 200);
    Map expectedResponse = new HashMap();
    expectedResponse.put("responsetest", true);
    expectedResponse.put("name", "harry");
    assertEquals(new ConsumerClient("http://localhost:8080").getAsMap("/", ""), expectedResponse);
}
```

```java
@Rule
public PactProviderRule provider = new PactProviderRule("test_provider", "localhost", 8080, this);

@Pact(provider="test_provider", consumer="test_consumer")
public PactFragment createFragment(PactDslWithProvider builder) {
    Map<String, String> headers = new HashMap<String, String>();
    headers.put("testreqheader", "testreqheadervalue");

    return builder
        .given("test state")
        .uponReceiving("ExampleJavaConsumerPactRuleTest test interaction")
            .path("/")
            .method("GET")
            .headers(headers)
        .willRespondWith()
            .status(200)
            .headers(headers)
            .body("{\"responsetest\": true, \"name\": \"harry\"}")
        .given("test state")
        .uponReceiving("ExampleJavaConsumerPactRuleTest second test interaction")
            .method("OPTIONS")
            .headers(headers)
            .path("/second")
            .body("")
        .willRespondWith()
            .status(200)
            .headers(headers)
            .body("")
        .toFragment();
}

@Test
@PactVerification("test_provider")
public void runTest() throws IOException {
    Assert.assertEquals(new ConsumerClient("http://localhost:8080").options("/second"), 200);
    Map expectedResponse = new HashMap();
    expectedResponse.put("responsetest", true);
    expectedResponse.put("name", "harry");
    assertEquals(new ConsumerClient("http://localhost:8080").getAsMap("/", ""), expectedResponse);
}
```

```java
@Rule
public PactProviderRule provider = new PactProviderRule("test_provider", "localhost", 8080, this);

@Pact(provider="test_provider", consumer="test_consumer")
public PactFragment createFragment(PactDslWithProvider builder) {
    Map<String, String> headers = new HashMap<String, String>();
    headers.put("testreqheader", "testreqheadervalue");

    return builder
        .given("test state")
        .uponReceiving("ExampleJavaConsumerPactRuleTest test interaction")
            .path("/")
            .method("GET")
            .headers(headers)
        .willRespondWith()
            .status(200)
            .headers(headers)
            .body("{\"responsetest\": true, \"name\": \"harry\"}")
        .given("test state")
        .uponReceiving("ExampleJavaConsumerPactRuleTest second test interaction")
            .method("OPTIONS")
            .headers(headers)
            .path("/second")
            .body("")
        .willRespondWith()
            .status(200)
            .headers(headers)
            .body("")
        .toFragment();
}

@Test
@PactVerification("test_provider")
public void runTest() throws IOException {
    Assert.assertEquals(new ConsumerClient("http://localhost:8080").options("/second"), 200);
    Map expectedResponse = new HashMap();
    expectedResponse.put("responsetest", true);
    expectedResponse.put("name", "harry");
    assertEquals(new ConsumerClient("http://localhost:8080").getAsMap("/", ""), expectedResponse);
}
```

```java
@Rule
public PactProviderRule provider = new PactProviderRule("test_provider", "localhost", 8080, this);

@Pact(provider="test_provider", consumer="test_consumer")
public PactFragment createFragment(PactDslWithProvider builder) {
    Map<String, String> headers = new HashMap<String, String>();
    headers.put("testreqheader", "testreqheadervalue");

    return builder
        .given("test state")
        .uponReceiving("ExampleJavaConsumerPactRuleTest test interaction")
            .path("/")
            .method("GET")
            .headers(headers)
        .willRespondWith()
            .status(200)
            .headers(headers)
            .body("{\"responsetest\": true, \"name\": \"harry\"}")
        .given("test state")
        .uponReceiving("ExampleJavaConsumerPactRuleTest second test interaction")
            .method("OPTIONS")
            .headers(headers)
            .path("/second")
            .body("")
        .willRespondWith()
            .status(200)
            .headers(headers)
            .body("")
        .toFragment();
}

@Test
@PactVerification("test_provider")
public void runTest() throws IOException {
    Assert.assertEquals(new ConsumerClient("http://localhost:8080").options("/second"), 200);
    Map expectedResponse = new HashMap();
    expectedResponse.put("responsetest", true);
    expectedResponse.put("name", "harry");
    assertEquals(new ConsumerClient("http://localhost:8080").getAsMap("/", ""), expectedResponse);
}
```

Groovy Example

```groovy
    def 'example V3 spec test'() {
      given:
      def matcherService = new PactBuilder()
      matcherService {
        serviceConsumer 'MatcherConsumer'
        hasPactWith 'MatcherService'
        port 1234
      }

      matcherService {
        uponReceiving('a request')
        withAttributes(method: 'put', path: '/')
        withBody(mimeType: JSON.toString()) {
          name(~/\w+/, 'harry')
          surname regexp(~/\w+/, 'larry')
          position regexp(~/staff|contractor/, 'staff')
          id identifier('1234567890')
          age(100)

          role {
            name('admin')
            id(uuid)
            kind {
              id(100)
            }
            dob date('MM/dd/yyyy')
          }
        }
        willRespondWith(status: 200)
        withBody(mimeType: JSON.toString()) {
          name(~/\w+/, 'harry')
        }
      }
    when:
    VerificationResult result = matcherService.run(specificationVersion: PactSpecVersion.V3) {
      def client = new RESTClient('http://localhost:1234/')
      def response = client.put(requestContentType: JSON, body: [
          'name': 'harry',
          'surname': 'larry',
          'position': 'staff',
          'id': 6444667731,
          'age': 32,
          'role': [
            'name': 'admin',
            'id': '7a97e929-c5b1-43cf-9b2c-295e9d4fa3cd',
            'kind': [
              'id': 100
            ],
            'dob': '12/05/2015'
          ]
        ]
      )

      assert response.status == 200
      assert response.data == [name: 'harry']
    }

    then:
    result == PactVerified$.MODULE$
  }
```

Next publish your pacts

```
{
    "provider": {
        "name": "Alice Service"
    },
    "consumer": {
        "name": "Consumer"
    },
    "interactions": [
        {
            "providerState": "",
            "description": "a retrieve Mallory request",
            "request": {
                "method": "GET",
                "path": "/mallory",
                "query": "name=ron&status=good",
                "body": null
            },
            "response": {
                "status": 200,
                "headers": {
                    "Content-Type": "text/html"
                },
                "body": "\"That is some good Mallory.\""
            }
        }
    ],
    "metadata": {
        "pact-specification": {
            "version": "2.0.0"
        },
        "pact-jvm": {
            "version": "3.2.7"
        }
    }
}
```

PactBroker

Build Artifacts

S3

Version Control

Then verify your provider

Gradle

Maven

Leiningen

SBT

JUnit

Ruby (Rake)

.Net (xUnit)

JS (NPM, Grunt, …)

Go

```
Verifying a pact between sampleconsumer and Activity Service
 [Using file /home/ronald/Development/Projects/Pact/pact-gradle-test/src/test/resources/sample-pact.json]
 Given many activities exist
        WARNING: State Change ignored as there is no stateChange URL
 a request for activities
   returns a response which
     has status code 200 (OK)
     includes headers
       "Content-Type" with value "application/json" (OK)
     has a matching body (OK)

Verifying a pact between sampleconsumer2 and Activity Service
 [Using file /home/ronald/Development/Projects/Pact/pact-gradle-test/src/test/resources/sample-pact2.json]
 Given many activities exist
        WARNING: State Change ignored as there is no stateChange URL
 a request for activities
   returns a response which
     has status code 200 (OK)
     includes headers
       "Content-Type" with value "application/json" (OK)
     has a matching body (OK)

Verifying a pact between sampleconsumer3 and Activity Service
 [Using file /home/ronald/Development/Projects/Pact/pact-gradle-test/src/test/resources/sample-pact3.json]
 add a broker
   returns a response which
     has status code 200 (OK)
     includes headers
       "Content-Type" with value "text/plain;charset=UTF-8" (OK)
     has a matching body (OK)
:pactVerify

BUILD SUCCESSFUL
```

```
Verifying a pact between sampleconsumer and Activity Service
 [Using file /home/ronald/Development/Projects/Pact/pact-gradle-test/src/test/resources/sample-pact.json]
 Given many activities exist
        WARNING: State Change ignored as there is no stateChange URL
 a request for activities
   returns a response which
     has status code 200 (OK)
     includes headers
       "Content-Type" with value "application/json" (OK)
     has a matching body (FAILED)

Failures:

0) Verifying a pact between sampleconsumer and Activity Service - a request for activities Given many activities exist returns a
response which has a matching body
     $.body.activities -> Expected List(Map(description -> 100, id -> 100, name -> Bob)) to have minimum 2
     $.body.activities.0.description -> Expected 100 to be the same type as 'f_UXcxIXYhgqtxjiPumRiCo9C5JNDX'

     Diff:

     @3
             {
     -            "description": "f_UXcxIXYhgqtxjiPumRiCo9C5JNDX",
     -            "name": "hx55sbvMPk1kF-9"
     -        },
     -        {
     -            "description": "f_UXcxIXYhgqtxjiPumRiCo9C5JNDX",
     -            "name": "hx55sbvMPk1kF-9"
     +            "description": 100,
     +            "id": 100,
     +            "name": "Bob"
             }


:pactVerify_Activity Service FAILED

FAILURE: Build failed with an exception.

* What went wrong:
There were 1 pact failures for provider Activity Service
```

Well, that’s interesting ...

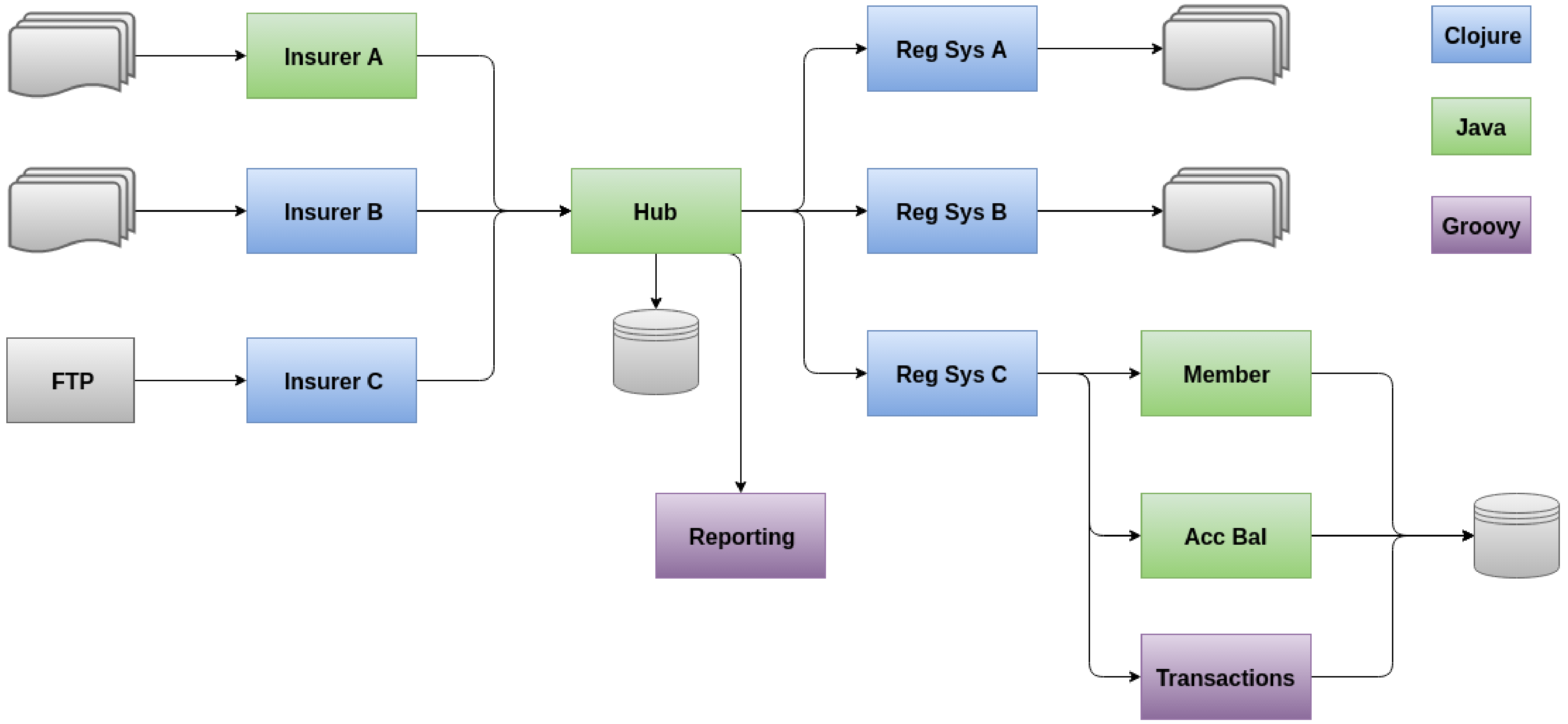
Specification by Example

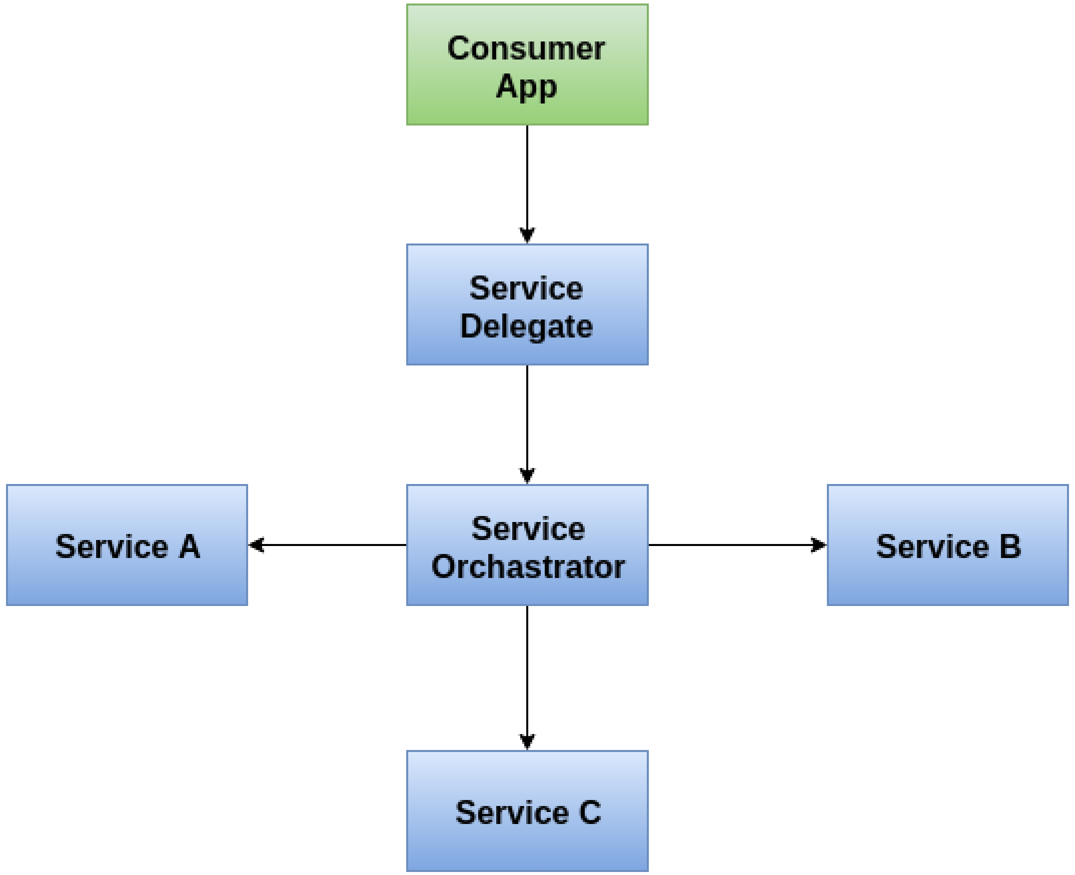
TDD for services

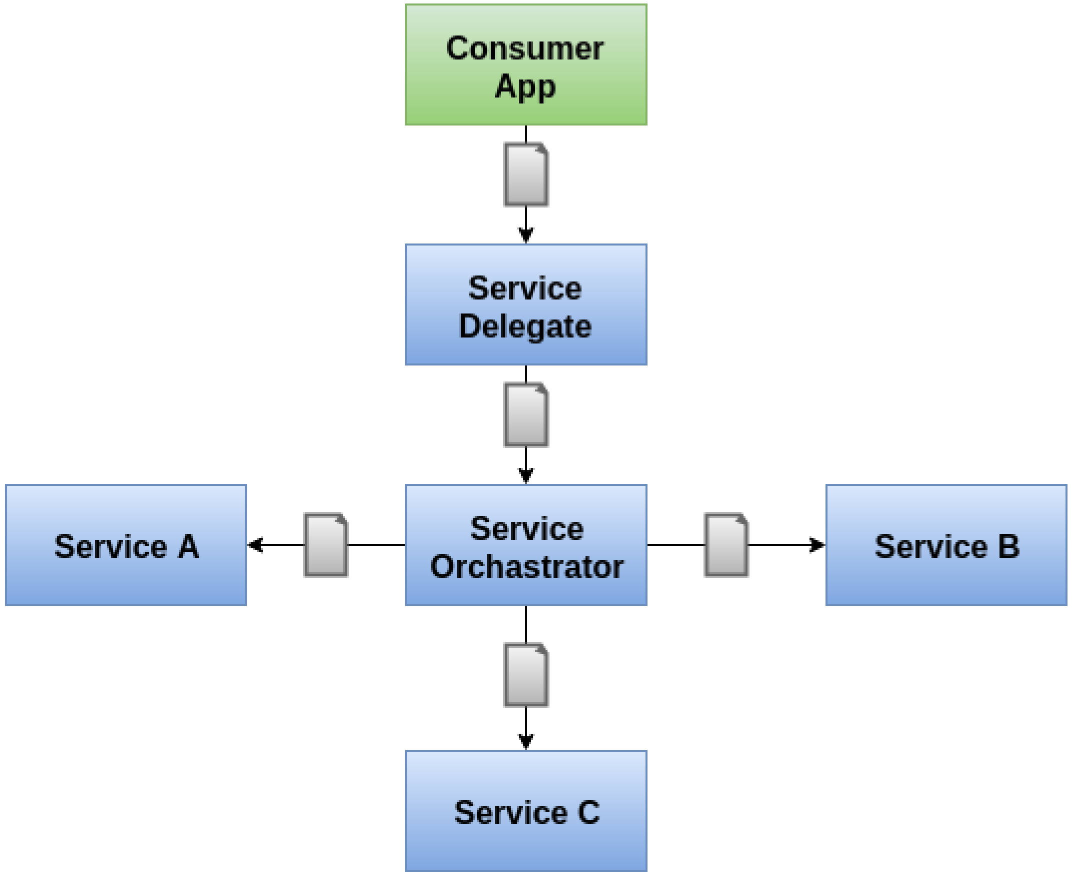
But ...

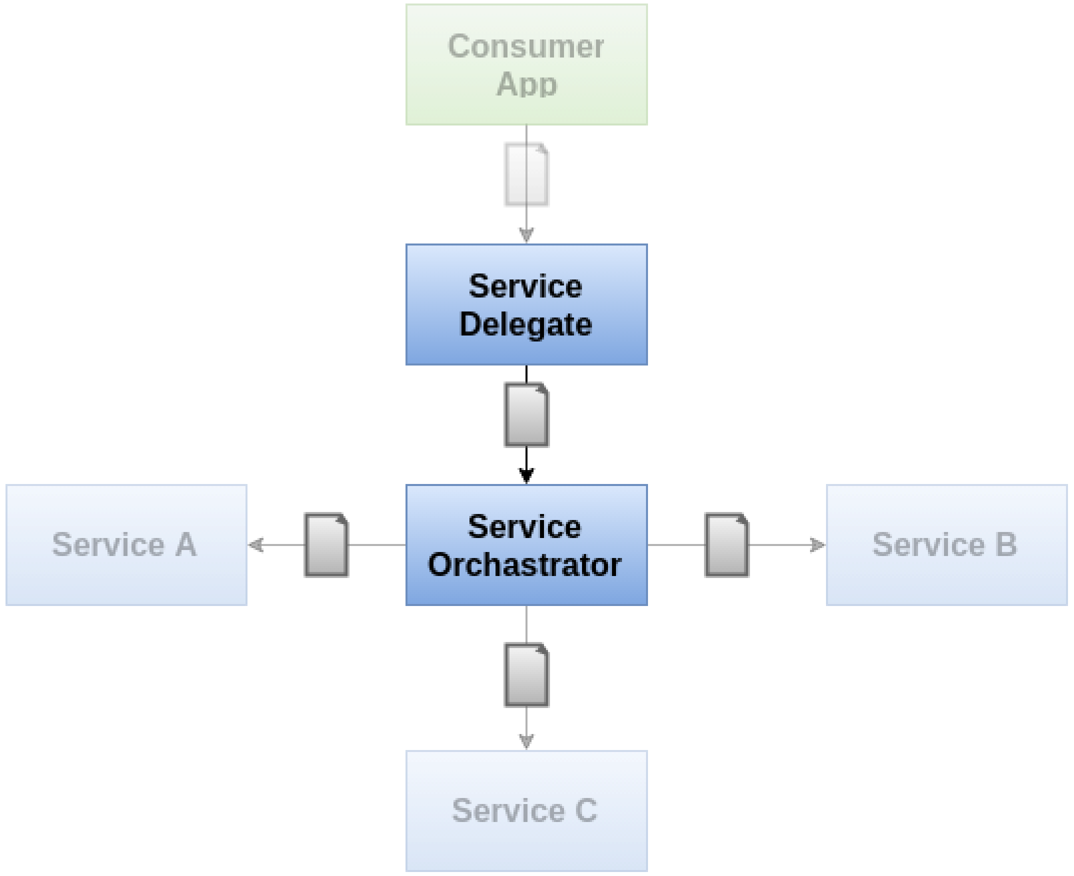
A pitfall for every person

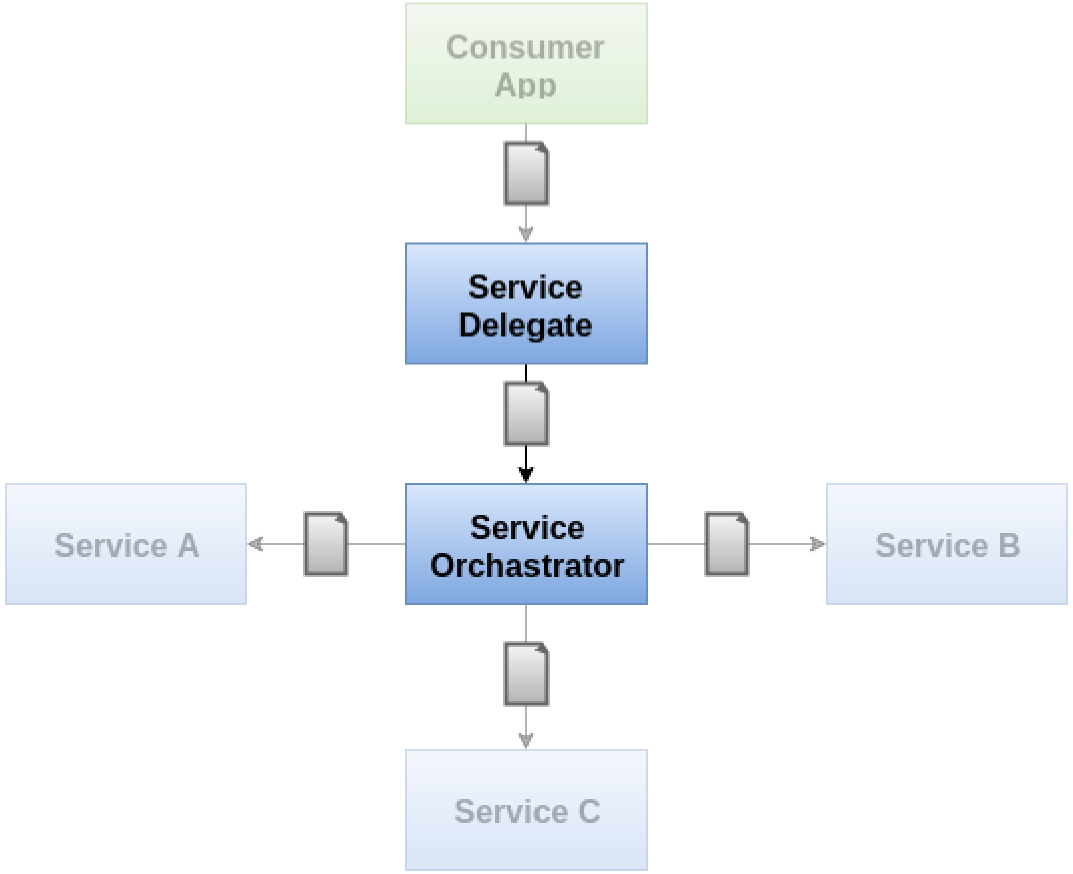
Isolated build data dependency


Insurer A
Insurer B
FTP
Insurer C
Hub
Reporting
Reg Sys A
Reg Sys B
Reg Sys C
Member
Acc Bal
Transactions
Clojure
Java
Groovy


Consumer App
Service Delegate
Service Orchastrator
Service A
Service B
Service C


Consumer App
Service Delegate
Service A
Service Orchestrator
Service B
Service C


Consumer App
Service Delegate
Service A
Service Orchastrator
Service B
Service C


Consumer App
Service Delegate
Service Orchastrator
Service A
Service B
Service C

True Mock Server
JUnit Provider Test

```java
@RunWith(PactRunner.class) // Say JUnit to run tests with custom Runner
@Provider("Hello Provider") // Set up name of tested provider
//@PactUrl(urls = {"file:///home/ronald/Development/Projects/Pact/pact-gradle-test/src/test/resources/hello_consumer-hello_provider.json"})
@PactFolder("pacts")
public class ContractTrialTest {
  @TestTarget // Annotation denotes Target that will be used for tests
  public final Target target = new HttpTarget(5050);

  @TargetRequestFilter
  public void exampleRequestFilter(HttpRequest request) {
    request.addHeader("Authorization", "Basic " + base64StringOf("admin", "admin"));
  }

  private String base64StringOf(String username, String password) {
    return Base64.getEncoder().encodeToString((username + ":" + password).getBytes());
  }
}
```

More Info

- **Gitbook:** docs.pact.io
- **Github:** realestate-com-au/pact **and** DiUS/pact-jvm
- **Google users group:** https://groups.google.com/forum/#!forum/pact-support
- **Gitter: Join the chat at** https://gitter.im/realestate-com-au/pact **and** https://gitter.im/DiUS/pact-jvm
- **Twitter:** @pact_up

Given “The presentation is over”

Upon Receiving “A request for an answer”

With “A valid question”

Respond With “A valid answer”

DIUS