

# API Workshop

## Contract Driven Development

O'REILLY®

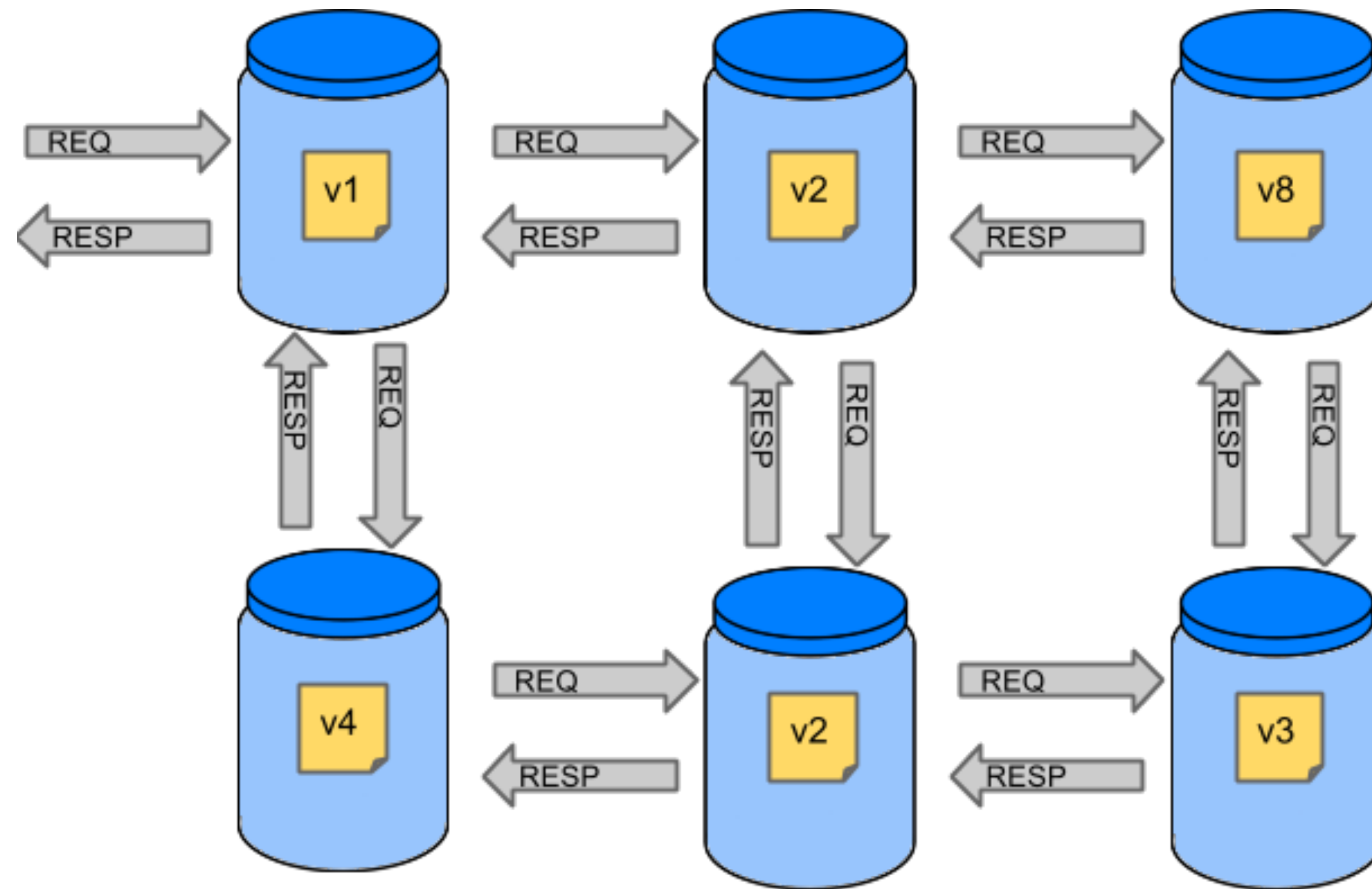
Software  
Architecture

# Agenda

- Testing services
- What are contracts
- Contract development methodologies
- Distributing contracts

# Problems with decoupled services

How do we test an application that calls out to other services?



# Problems with decoupled services

## End to end testing?

- Real tests like production, but comes at a cost
- May require a few other services, databases
- Difficult to debug if a problem occurs
- Late feedback makes it difficult to discover and fix bugs early

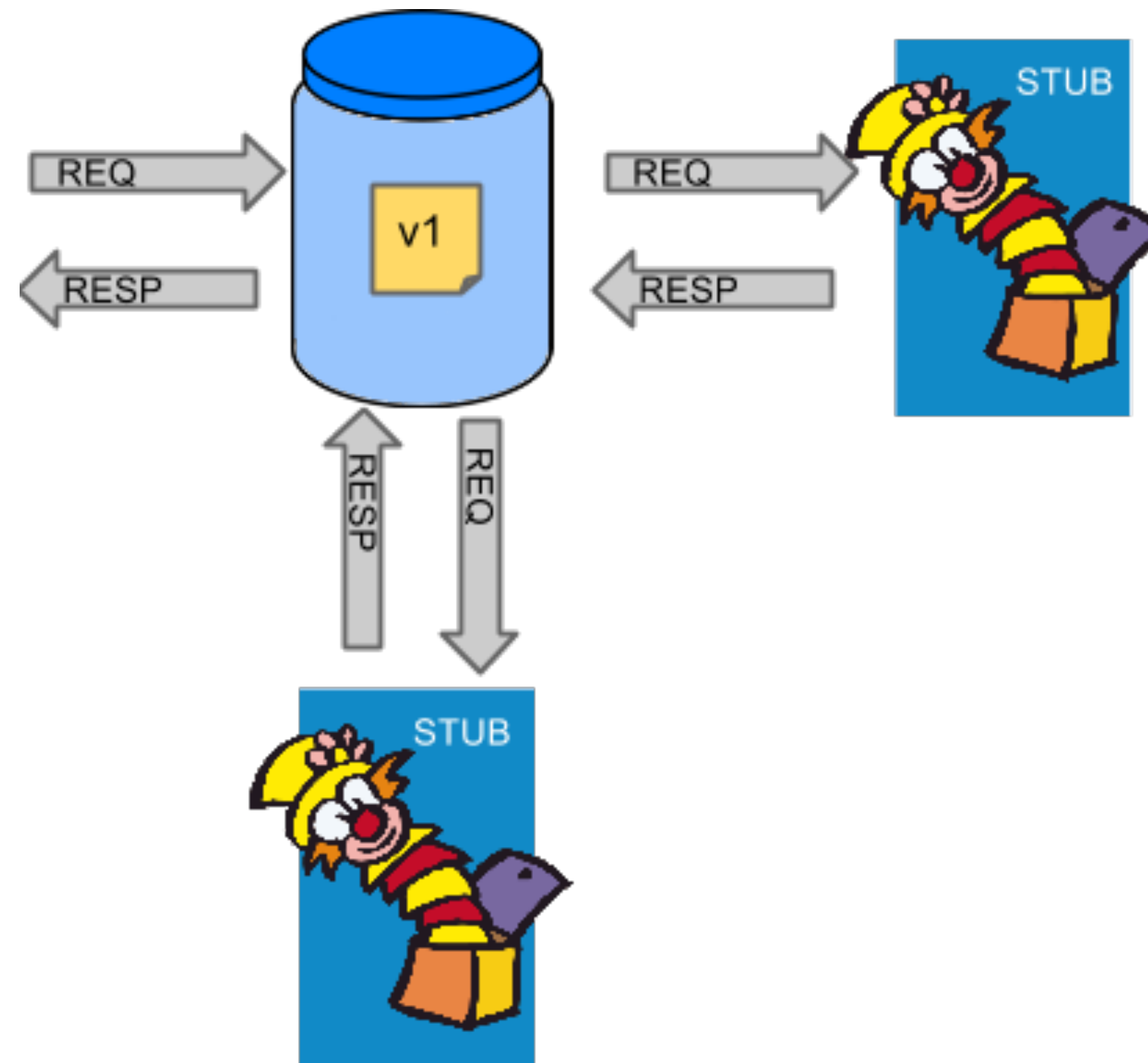
# Problems with decoupled services

## **Mock dependencies in unit and integration tests?**

- Very fast feedback and no infrastructure
- May differ vastly from reality
- Can end up in production with tests that pass, but would fail in production

# Contract Driven Development

## What about contracts?



# Contract Driven Development

## What about contracts?

- Ensures that HTTP messaging stubs are exactly how the server will behave
- Promotes acceptance test driven development (ATTD)
- Publish contracts that can immediately be used by the consumer and producer
- Generates and guides tests

In reality we want to have a balance of end to end, mocking and contract tests



# Contract Driven Development

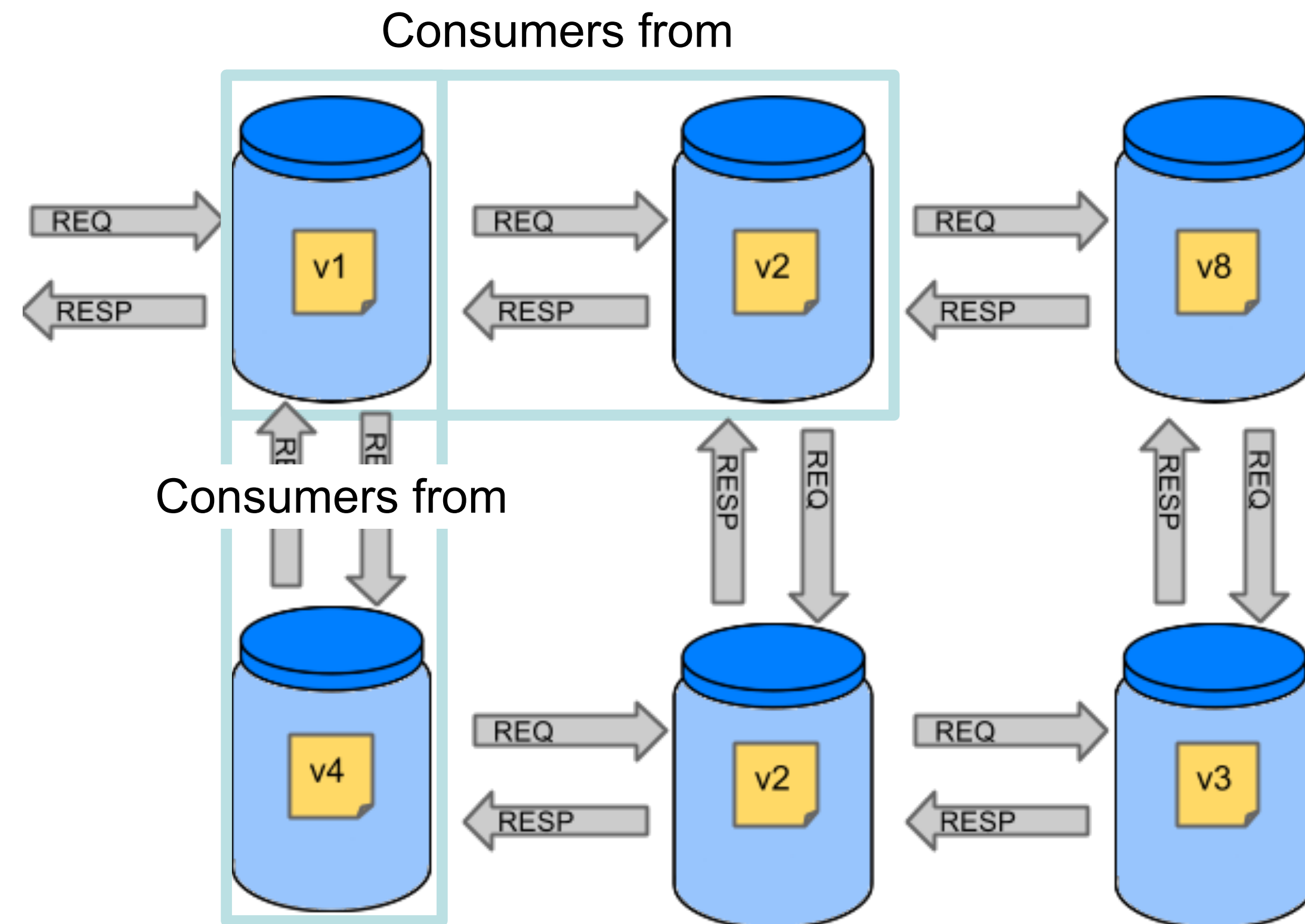
## The two entities

- Producer
  - An application that is producing data, e.g. We are currently writing a Todo application. The app is a producer as it **provides**
- Consumer
  - Any application that **consumes** information from a provider e.g. The webclient that is sending requests to the Todo application, it is consuming the Todo service



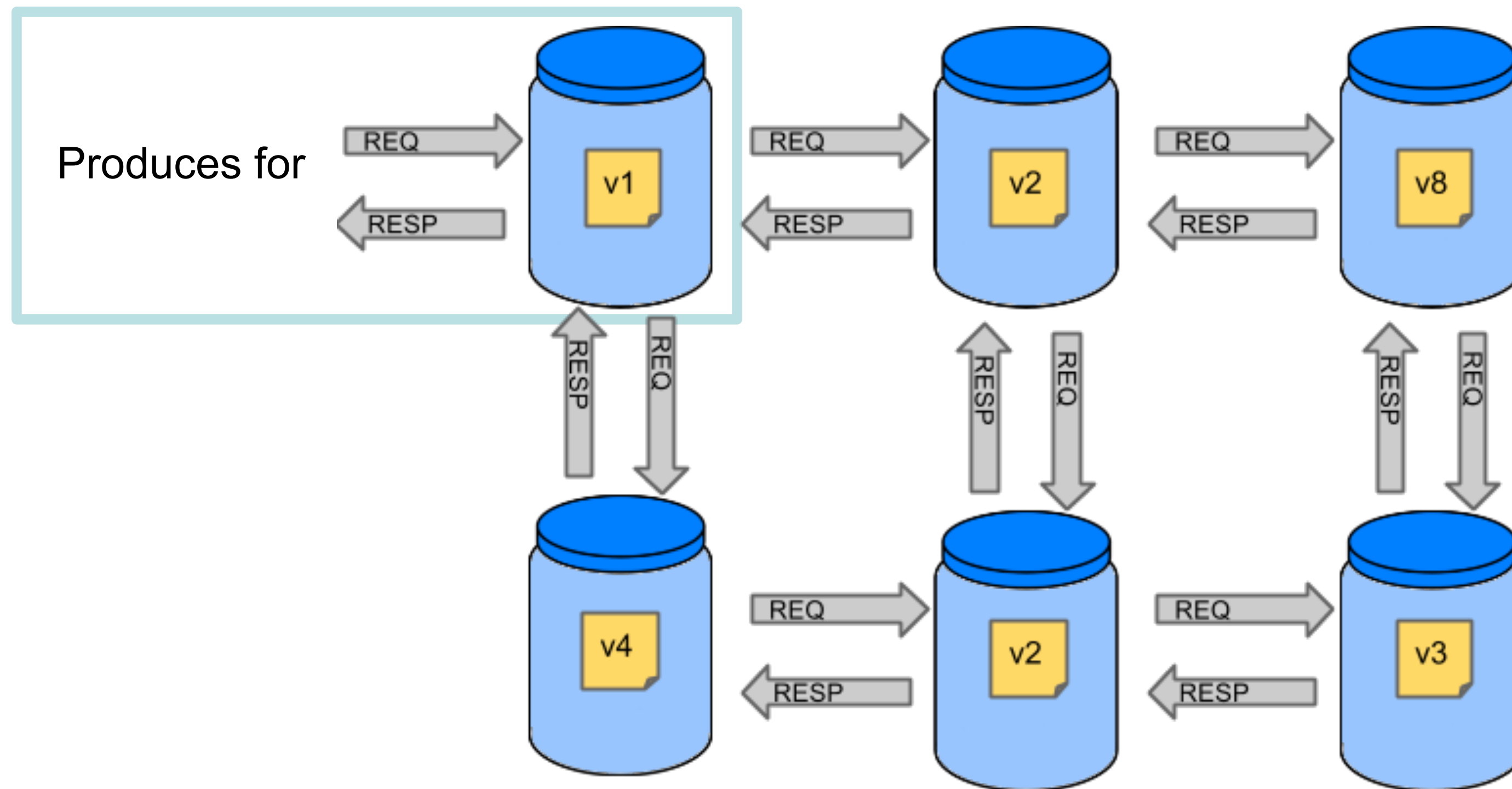
# Contract Driven Development

## Consumer



# Contract Driven Development

## Producer



# Contract Driven Development

What do contracts do?

- Specify how we expect the client to interact with our service
- There may be multiple contracts per endpoint
- The API shape (or structure) does not define full behaviour
- Contracts provide quick feedback on evolving APIs

# Contract Driven Development

Two types of Contract driven development

- Provider Contract Development
- Consumer Driven Contracts

# Producer Contract Development

# Producer Contract Development

## Provider Contract Development

The Producer creates their own contracts.

- External facing, large client base
  - Example of usage: The Todo API we are developing is an external facing API used by millions

# Example Contract

```
org.springframework.cloud.contract.spec.Contract.make {  
    request {  
        method POST()  
        headers {  
            contentType(applicationJson())  
        }  
        url '/todos'  
        body(  
            "message": "A new todo"  
        )  
    }  
    response {  
        status CREATED()  
        headers {  
            header(location(), anyUrl())  
        }  
    }  
}
```



# Example Contract - Generated test for producer

```
@Test
public void validate_task_create_entry() throws Exception {
    // given:
    MockMvcRequestSpecification request = given()
        .header("Content-Type", "application/json")
        .body("{ \"message\": \"A new todo\"}");

    // when:
    ResponseOptions response = given().spec(request)
        .post("/todos");

    // then:
    assertThat(response.statusCode()).isEqualTo(201);
    assertThat(response.header("Location")).matches(anyUrlRegex());
}
```

# Stub generated for Consumer

- The contracts also form stubs
- Stubs can be used to mock out services
- Allows consumers to work on their applications, whilst serverside logic is developed
- Decouples dependencies during the development process

# Consumer Driven Contracts Development

# Consumer Driven Contracts Development

## Consumer Driven Contracts

- The Consumer creates that contracts for the Producer to fulfil
  - Teams working closely together
  - Example of usage: The Todo API will be used by a few teams in a company

# Example Interaction

```
@Pact(consumer = "todo-api-consumer-pact")
public RequestResponsePact updateAToDoRequest(PactDslWithProvider builder) {
    return builder
        .given("an todo of id 1 has already been created")
        .uponReceiving("a request to update the todo")
        .matchPath("/todos/1")
        .method("PUT")
        .headers("Content-Type", "application/json")
        .body("{\"message\":\"Make all the beds\"}")
        .willRespondWith()
        .status(204)
        .toPact()
        ;
}
```

# Example Interaction - Generated Contract

```
{
  "provider": {"name": "todo-api-producer-pact"},
  "consumer": {"name": "todo-api-consumer-pact"},
  "interactions": [
    {
      "description": "a request to update the todo",
      "request": {
        "method": "PUT",
        "path": "/todos/1",
        "headers": {"Content-Type": "application/json"},
        "body": {"message": "Make all the beds"},
        "matchingRules": {
          "path": {"": {"matchers": [{
            "match": "regex",
            "regex": "/todos/1"
          }],
            "combine": "AND"
          }},
          "header": {},
          "body": {}
        }
      },
      "response": {"status": 204},
      "providerStates": [
        {"name": "an todo of id 1 has already been created"}
      ]
    }
  ]
}
```

# Test generated for Producer

- The interactions also form tests for the Producer
- Defines the interactions the Consumer wants to see fulfilled
- Allows consumers to drive out the features that they want to see and use
- Decouples dependencies during the development process



# Contract broker

- An application for sharing for contracts (Pacts)
- Pact provide a broker

