OPEN VALUE

Three approaches in API Development

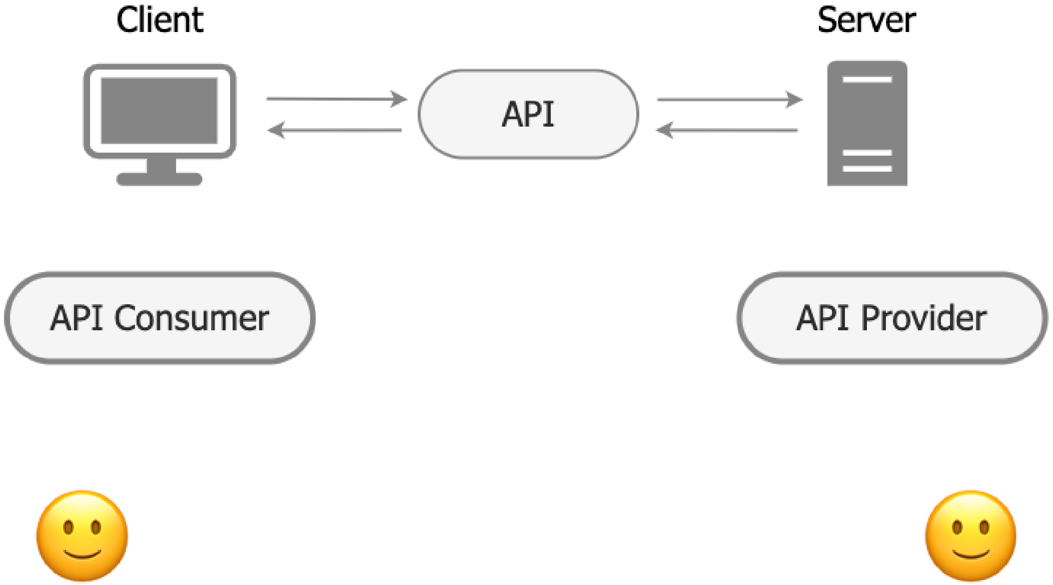
Enis Spahi (@enisspahi)

API

"An application programming interface (API) is a way for two or more computer programs to communicate with each other."

Source: Wikipedia, "API"

Pairs



Client

API

Server

API Consumer

API Provider

🙂 🙂

Boundaries

- Private APIs
  - Provider and consumer are developers in the same team or same organisation

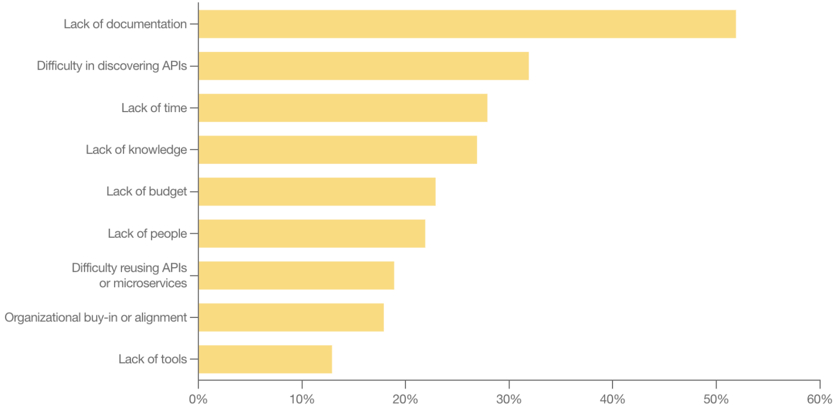
- Partner Facing APIs
  - Serving partners (i.e.: Payment Service Providers)
  - Provider and consumer might not communicate directly

- Public APIs
  - Publicly available (i.e… Geo-Location services)
  - Communication at scale: Many Consumers ↔ 1 Provider

Statistics

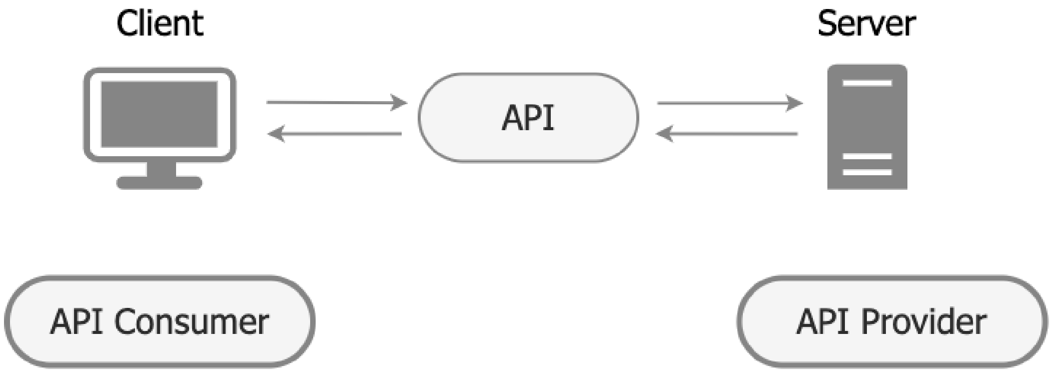
Top 3 Obstacles to consuming APIs

- Lack of documentation
- Difficulty in discovering APIs
- Lack of time



Source: Postman, "2023 State of the API Report"

API Communication



🙂🤦‍♀️😱😡

🤯🤔

Communication Issues between Pairs

Find a common language

OPENAPI INITIATIVE

AsyncAPI

OpenAPI Specification

- Technology agnostic standard to describe Rest APIs
- Formerly Swagger, OpenAPI as of version 3
- Written as JSON or YAML
- Great tooling for code and documentation generation
- https://openapi.tools/

OpenAPI Specification

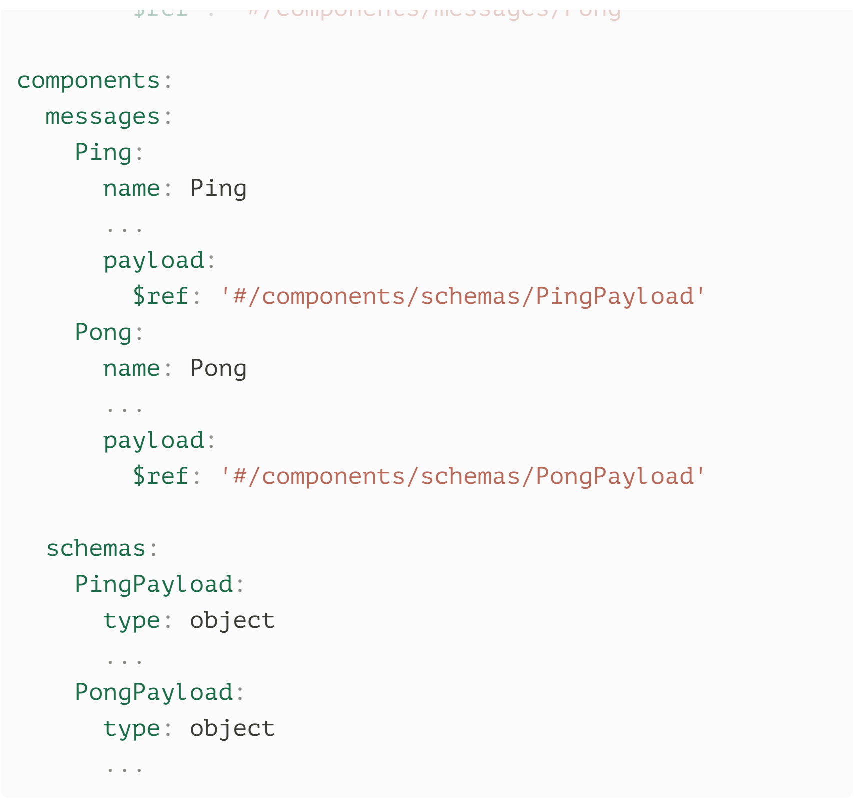
```yaml
openapi: 3.0.3
info:
  title: Recipes API
  ...
servers:
- url: http://localhost:8080
paths:
  /recipes:
    get:
      summary: List all recipes
      ...
      responses:
        ...
        "200":
          description: OK
          content:
            'application/json':
              schema:
                type: array
                items:
                  $ref: '#/components/schemas/Recipe'
```

- **Openapi:** Spec Version
- **Info:** General API information as metadata
- **Servers:** Connectivity information about target servers
- **Paths:** Paths to the endpoints with their expected request, response and errors.
- **Components:** Holds the schemas for the request, response and errors for referencing

AsyncAPI Specification

- Technology agnostic standard to describe message-driven APIs
- An adaptation of the OpenAPI specification
- Written as JSON or YAML
- Protocols: AMQP, HTTP, JMS, Kafka, but not only
- https://www.asyncapi.com/tools

AsyncAPI Specification



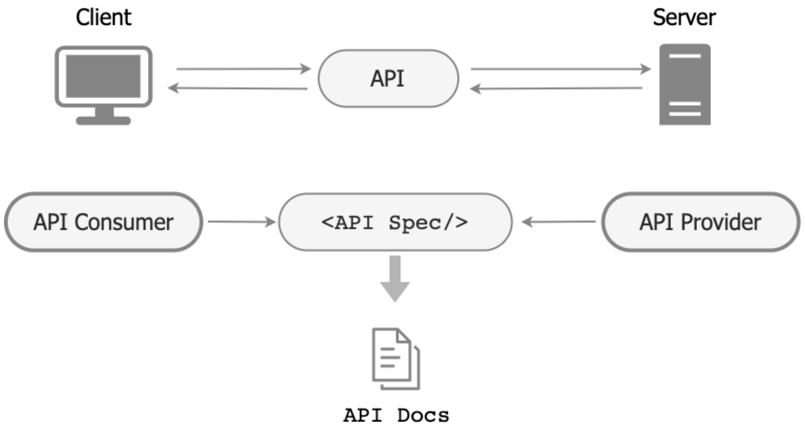
```
        $ref: '#/components/messages/Pong'

components:
  messages:
    Ping:
      name: Ping
      ...
      payload:
        $ref: '#/components/schemas/PingPayload'
    Pong:
      name: Pong
      ...
      payload:
        $ref: '#/components/schemas/PongPayload'

  schemas:
    PingPayload:
      type: object
      ...
    PongPayload:
      type: object
      ...
```

- **Asyncapi:** Spec Version
- **Info:** Metadata information about the API
- **Servers:** Connectivity information about servers (i.e. Kafka brokers)
- **Channels:** Messages exchange between provider and consumer
- **Components:** Defines the reusable objects such as schemas or messages which could be referenced.

Standardized API Communication

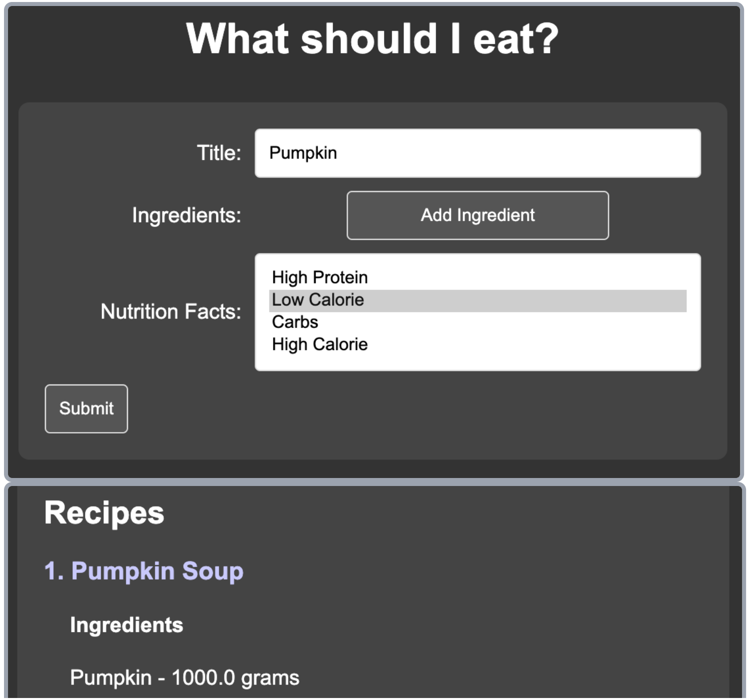


- Common Language for API discovery
  - OpenAPI/Swagger for REST APIs
  - AsyncAPI for message-driven APIs
- Foundation for tooling
  - Code generation
  - Documentation
- Community

Development Process

Let’s build an API - Recipes Demo

Client



What should I eat?

Title: Pumpkin

Ingredients:

Nutrition Facts:
- High Protein
- Low Calorie
- Carbs
- High Calorie

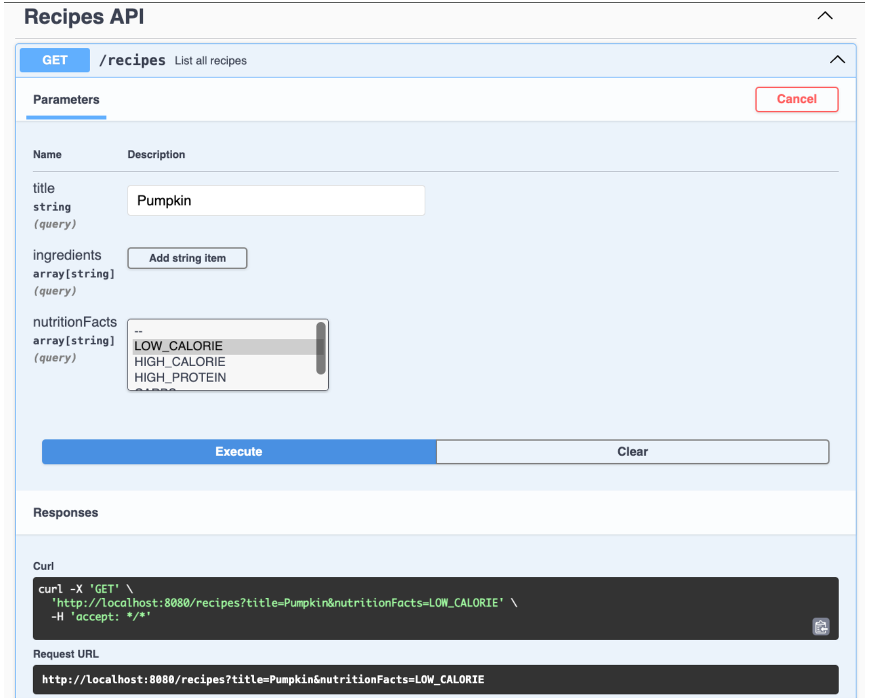
Recipes

1. Pumpkin Soup

Ingredients

Pumpkin - 1000.0 grams

Server



Recipes API

GET /recipes List all recipes

Name	Description
title string *(query)*	Pumpkin
ingredients array[string] *(query)*	
nutritionFacts array[string] *(query)*	-- LOW_CALORIE HIGH_CALORIE HIGH_PROTEIN

Responses

Curl

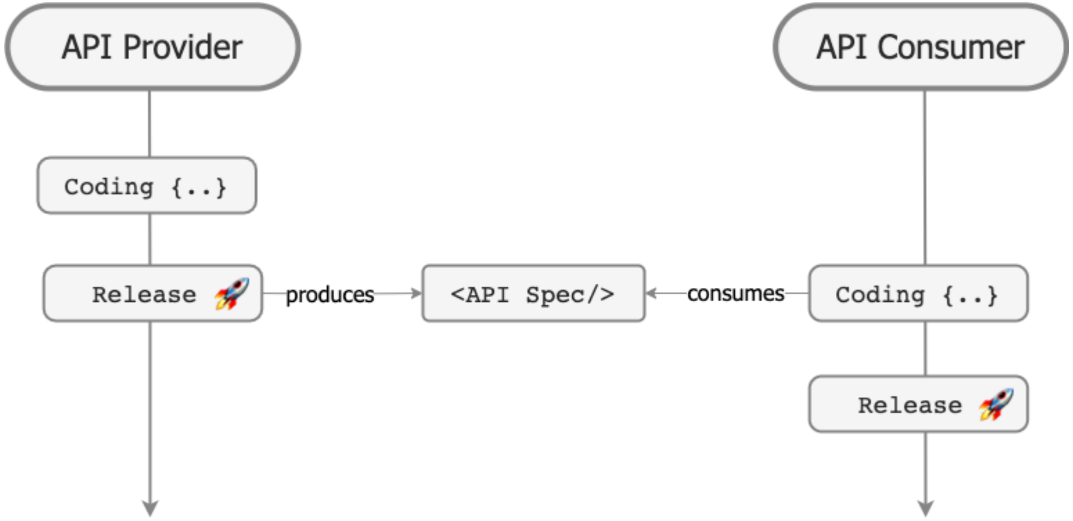
```
curl -X 'GET' \
  'http://localhost:8080/recipes?title=Pumpkin&nutritionFacts=LOW_CALORIE' \
  -H 'accept: */*'
```

Request URL

```
http://localhost:8080/recipes?title=Pumpkin&nutritionFacts=LOW_CALORIE
```

API Development - Code First

Communicate API specification once coding has been done



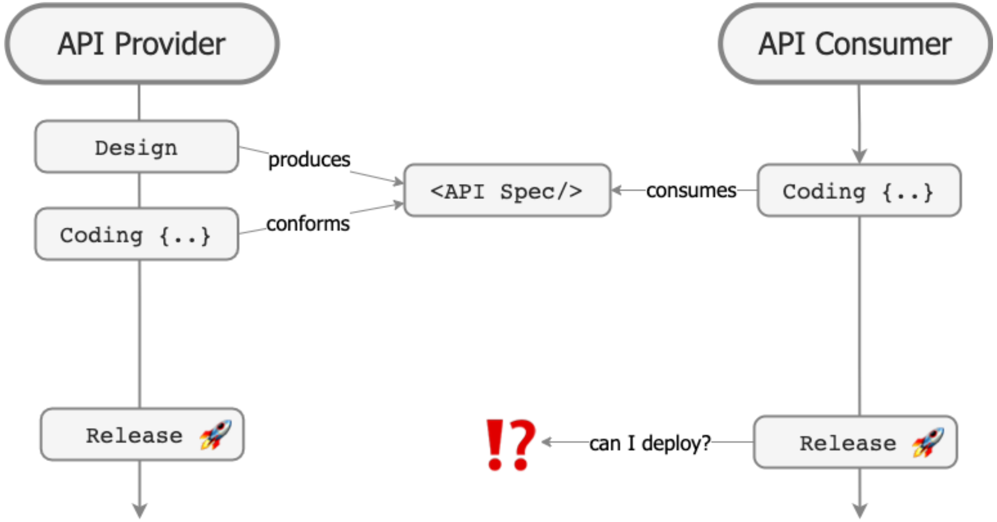
API Development - Code First

Communicate API specification once coding has been done

Advantages	Disadvantages
Focus on coding	Late communication with the consumer
Flexibility to change the API design	Does not enable development in parallel
	Annotations

API Development - API First

Communicate API specification before coding. Prioritizes API design over implementation.



API Development - API First

Communicate API specification before coding. Prioritizes API design over implementation.

Advantages	Disadvantages
Early communication with the consumer	Less flexibility to change the API design
Documentation thought ahead	Sometimes bureaucratic for providers
Enables development in parallel	

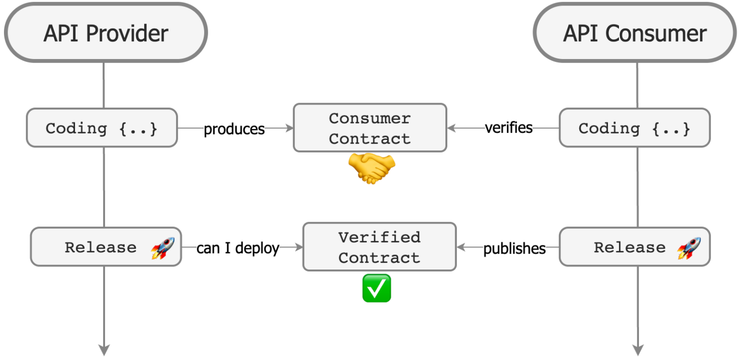
API Development - Consumer First

Consumer dictates the expected API behavior to the provider

Pact: A Code-first consumer contract testing tool that enables consumer driven API development.

Process:

- Consumer produces a pact
- Provider verifies it’s API implementation
- Server / Client deployments synced



Recap

When to use Code first?	Provider initially focuses on coding speed Flexibility to change the design
When to use API first?	API design over implementation Early communication with the consumer → Documentation Utilize code generation Large number of consumers
When to use Consumer first?	Provider should conform to consumer needs API consumer and provider test their applications independently To sync provider and consumer deployments Small number of consumers
When to mix & match?	When API first alone is not sufficient to match consumer needs

Are same approaches applicable to AsyncAPI?

- Inspired by OpenAPI
- Code First and API First
- Pact Messaging support
- https://www.asyncapi.com/tools

OpenValue Blog

https://openvalue.blog/posts/2023/11/25/communicating_our_apis_part1/

https://openvalue.blog/posts/2023/11/26/communicating_our_apis_part2/

Code Samples

https://github.com/enisspahi/code-first-api-example

https://github.com/enisspahi/contract-first-api-example

https://github.com/enisspahi/consumer-first-api-example

Q&A