OpenAPI development with Python

EuroPython2017@Rimini, 11 July 2017

Takuro Wada

Hi

Takuro Wada

Kabuku Inc.

Software Engineer

- Speaker of EuroPython 2016, PyConJP 2015
- Member of swagger codegen technical committee
  - Python, TypeScript

@taxpon

taxpon

http://takuro.ws

Agenda

1. What is OpenAPI?

Introduction and basics

2. OpenAPI tools

Introduce some tools to increase your productivity

3. Actual case study

Introduce our company’s project

What is OpenAPI?

What is OpenAPI?

OpenAPI is API description language which is focusing on creating, evolving and promoting vendor neutral description format

(Partially cited from https://www.openapis.org/about)

You can write your API spec with OpenAPI

What is OpenAPI?

- Supported spec format
  - YAML and JSON
- Based on JSON schema
  - Vocabulary to annotate and validate JSON
- Originally known as Swagger

```yaml
paths:
  /books:
    get:
      tags: [Books]
      operationId: getBooksList
      responses:
        '200':
          description: success
          schema:
            title: books
            type: array
            items:
              $ref: '#/definitions/Book'
    post:
      tags: [Books]
      operationId: createBook
      parameters:
        - name: book
          in: body
          description: book parameter
          required: true
          schema:
            $ref: '#/definitions/Book'
```

Actual example spec in YAML format

How to use OpenAPI?

- As API documents
  - Generate good looking documents
    - Share API spec in team
      - Frontend and Backend
      - Developers and non-developers
    - Public API Document for Any developers

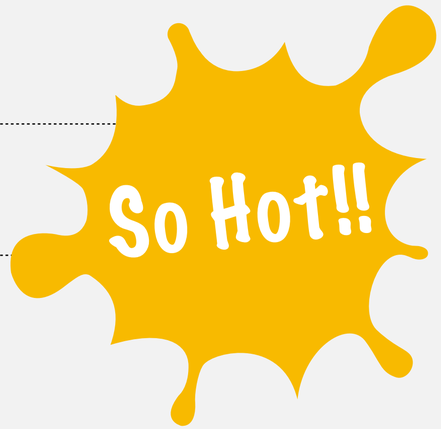
How to use OpenAPI?

- As API tools
  - Code generation
    - Codes for request data validation in server
    - Code for API calling in clients

OpenAPI versions

- OpenAPI is originally known as Swagger
  - Swagger spec was renamed OpenAPI spec in 2016

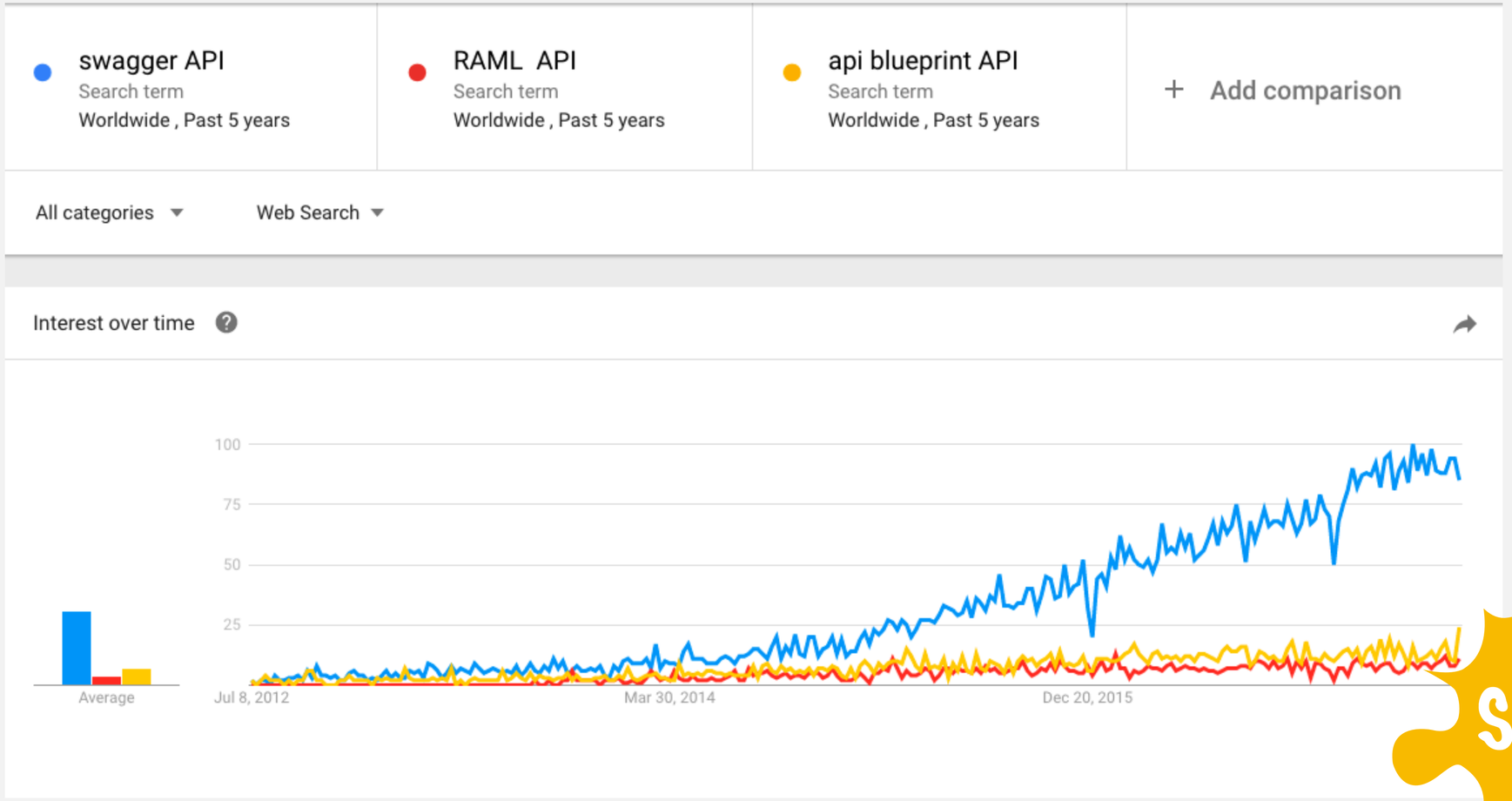
Version	Release Date
1.2	14 Mar 2014
2.0	8 Sep 2014
3.0	Will be released in July 2017 https://www.openapis.org/specification/repo



OpenAPI's competitors

- RESTful API DL (Description Language)
  - RAML (RESTful API Modeling Language)
    - YAML base
  - API Blueprint
    - Markdown base
    - Oracle acquired Apiary in Jan 2017

Google Trends



So Hot!!

OpenAPI (Swagger) is gathering more attention than others!

OpenAPI tools

OpenAPI tools

- Core tools
  - Developed by OpenAPI (Swagger) team

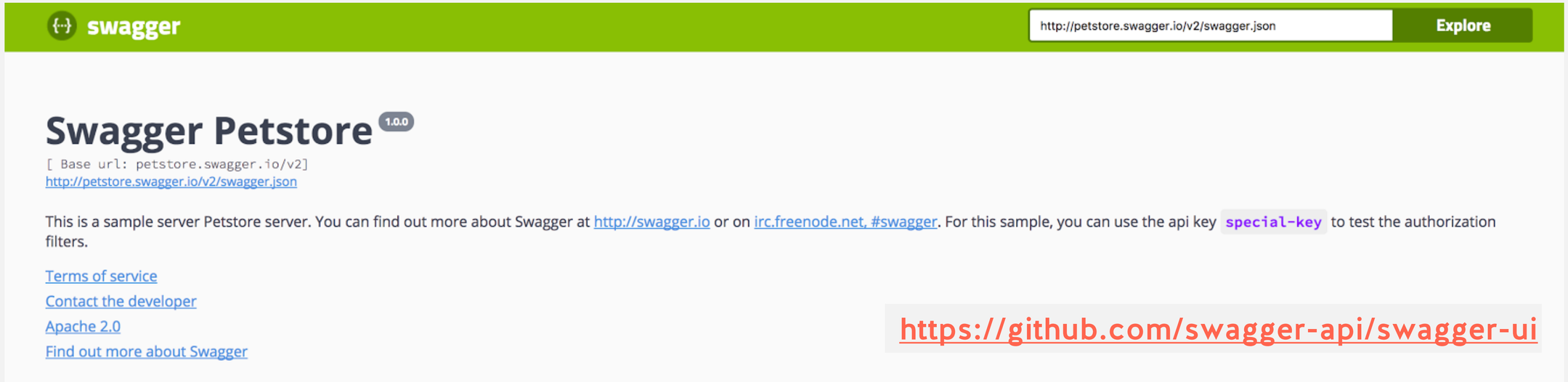
- Community tools
  - Developed by Community
  - Introduce Python tools in this session

Core tools

- Swagger UI
- Swagger Editor
- Swagger Codegen

Core tools (1/3)

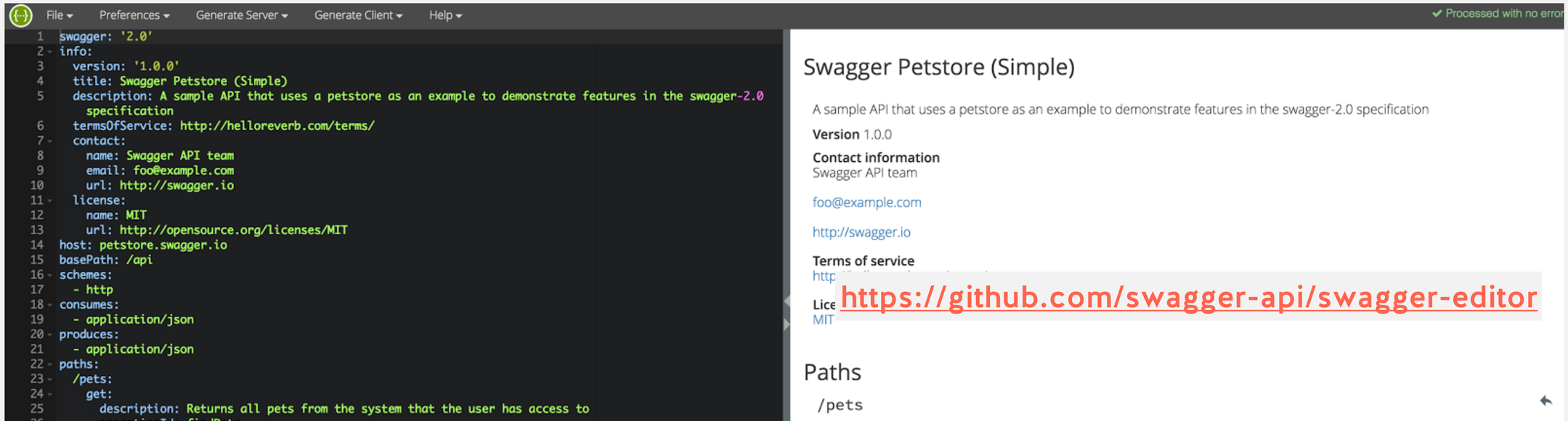
- Swagger UI
  - Show spec with beautiful format
  - Directly API calling from browser
  - http://petstore.swagger.io/



https://github.com/swagger-api/swagger-ui

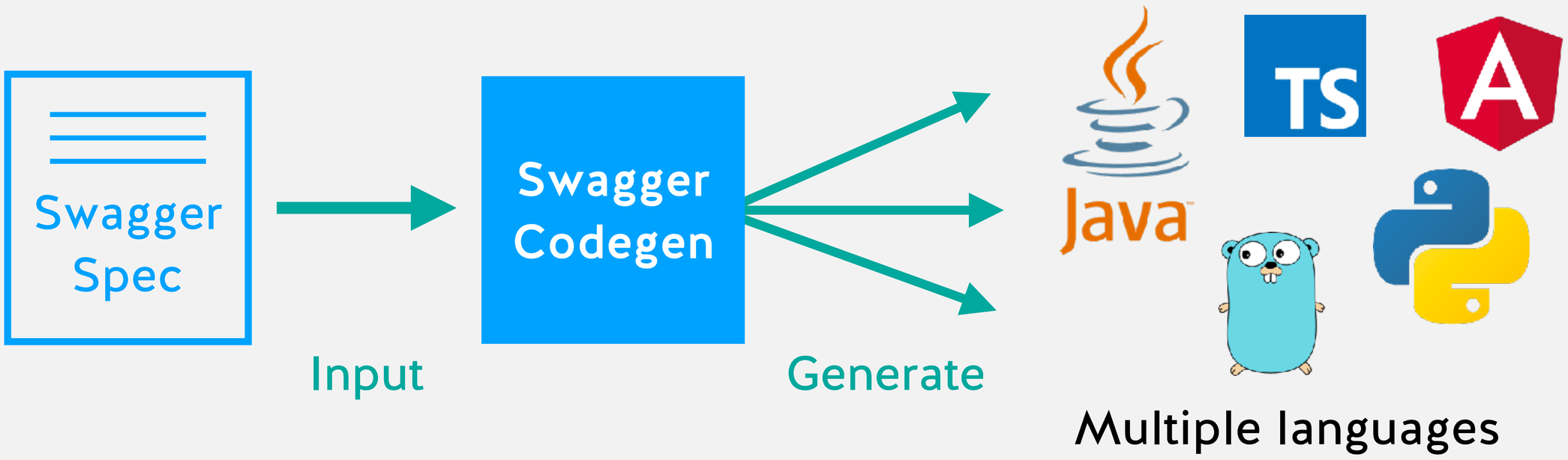
Core tools (2/3)

- Swagger Editor
  - WYSIWYG Spec editor in web browser
    - Syntax highlighting, Autocompletion, Real time spec validation
    - http://editor.swagger.io/#/



Core tools (3/3)

- Swagger Codegen
  - Generate server's and client's code from spec



Community tools

- There are many Python tools for OpenAPI
  - Validator
  - Code generator
  - Spec parser
  - Some tools are for the specific framework

Python

Name	Description
falsy	with FALS.Y, you can use falcon, swagger-ui, yml together, which makes writting api easy!
pyramid-swagger	Convenient tools for using Swagger to define and validate your interfaces in a Pyramid webapp.
flask-restplus	Helpers, syntaxic sugar and Swagger documentation for Flask-Restful
pyswagger	A type-safe, dynamic, spec-compliant Swagger client.
flex	Swagger 2.0 schema validation, and tooling for validating arbitrary request/response objects.
Flasgger	Flask Extension to provide Swagger 2.0 to any view using docstrings (embeds swagger UI)
flask-swagger	A Swagger 2.0 extractor for Flask via YAML in docstrings
bravado	Swagger 2.0 client with support for both synchronous and asynchronous http.
bravado-core	Library for Swagger 2.0 schema ingestion, validation, request/response validation, etc.
swagger-spec-validator	Library for validating Swagger 1.2 and 2.0 schemas.
swagger-py-codegen	Generate Flask-RESTful application code from a Swagger Specification doc.
Connexion	Swagger-first REST framework on top of Flask with validation and OAuth 2 support.
pecan-swagger	Partial swagger extractor for pecan.
swagger-parser	Give useful informations about your swagger files.
swagger-tester	Automatic swagger API tester.
swagger-aggregator	Aggregate several swagger APIs in one.
swagger-stub	Generate a stub from a swagger file.
bottle-swagger	Swagger integration for the Bottle web framework
prance	Swagger parser that resolves JSON references.

https://swagger.io/open-source-integrations/#python-19

bravado-core

- Python library that adds client-side and server-side support for OpenAPI (2.7, 3.4+, developed by Yelp, https://github.com/Yelp/bravado-core)
- Not dedicated to any specific framework (You can use it in you own project today)
- Very simple to use
- Features
  - Validation
  - Marshaling and Unmarshaling
    - Custom formats for type conversion

bravado-core

- Spec example

```
Book:
  type: object
  required: [id]
  properties:
    id:
      type: integer
    title:
      type: string
    author:
      type: string
```

bravado-core: (1)Validate

- Validation execution

```
import yaml
from bravado_core.spec import Spec

# 1
with open('openapi.yaml', 'r') as f:
    raw_spec = yaml.load(f)
# 2
spec = Spec.from_dict(raw_spec)
# 3
book = raw_spec[‘definitions']['Book']
# 4
validate_schema_object(spec, book, target)
```

1. Load YAML file with OpenAPI spec (JSON is also OK)
2. Create Spec object
3. Retrieve “Book” definition
4. Validate (target is dict object which is dumped from client’s request)

bravado-core: (1)Validate

- if required property “id” is not defined in dict:

Code

```
validate_schema_object(spec, book, {})
```

Result

```
jsonschema.exceptions.ValidationError: 'id' is a required property

Failed validating 'required' in schema:
    {'properties': {'author': {'type': 'string'},
                    'id': {'type': 'integer'},
                    'release_date': {'format': 'date', 'type': 'string'},
                    'title': {'type': 'string'}},
     'required': ['id'],
     'type': 'object',
     'x-model': 'Book'}

On instance:
    {}
```

bravado-core: (1)Validation

- If a property has invalid type value:

Code

```
validate_schema_object(spec, book, {"id": 1, "title": 1})
```

Result

```
jsonschema.exceptions.ValidationError: 1 is not of type 'string'

Failed validating 'type' in schema['properties']['title']:
    {'type': 'string'}

On instance['title']:
    1
```

bravado-core: (2)Unmarshal

- Unmarshal (dict to object)

Code

```
from bravado_core.unmarshal import unmarshal_schema_object

book_obj = unmarshal_schema_object(
    spec, book,
    {"id": 1,
     "title": "Merchant of Venice”,
     “author": "William Shakespeare"})
print(book_obj)
```

Dict need to be converted

Result

```
Book(author='William Shakespeare', id=1, title='Merchant of Venice')
```

Model object is created !!

bravado-core: (2)Unmarshal

- Formatting in Unmarshal

```
Book:
  type: object
  required: [id]
  properties:
    id:
      type: integer
    title:
      type: string
    author:
      type: string
    release_date:
      type: string
      format: date
```

This property is added and
expected to be string with “date" format

bravado-core: (2)Unmarshal

- Formatting in Unmarshal

Code

```
book_obj = unmarshal_schema_object(
    spec, book,
    {"id": 1,
     "title": "Merchant of Venice”,
     “author": "William Shakespeare”,
     “release_date”: “2017-07-11”})
print(book_obj)
```

Dict need to be converted

Result

```
Book(author='William Shakespeare', id=1,
release_date=datetime.date(2017, 7, 11), title='Merchant of Venice')
```

String with date format is successfully converted to a date object!!

bravado-core: (2)Unmarshal

- Defined formatter
  - Default defined formatter:
    - byte, date, double, date-time, float, int32, int64
    - formatter.py (https://github.com/Yelp/bravado-core/blob/master/bravado_core/formatter.py)
  - Custom formatter by yourself
    - https://github.com/Yelp/bravado-core/blob/master/docs/source/formats.rst

bravado-core: (3)Marshal

- Marshal (object to dict)

Code

```
Book = spec.definitions['Book']
book_obj = Book(id=1, title="Merchant of Venice",
                author="William Shakespeare”,
                release_date=date(2017, 7, 11))
book_dict = marshal_schema_object(spec, book, book_obj)
print(book_dict)
```

“Book” object

Result

```
{'release_date': '2017-07-11', 'title': 'Merchant of Venice', 'id': 1,
'author': 'William Shakespeare'}
```

Date object is successfully converted to string with date format!!

bravado-core

- And many useful features
  - Document
    - http://bravado-core.readthedocs.io/en/latest/
  - Examples
    - https://github.com/taxpon/bravado-core-example

Actual case study

Project overview

- Kabuku Connect
  - Manufacturing cloud platform
  - Connect people who want to make something and the factories selected by AI

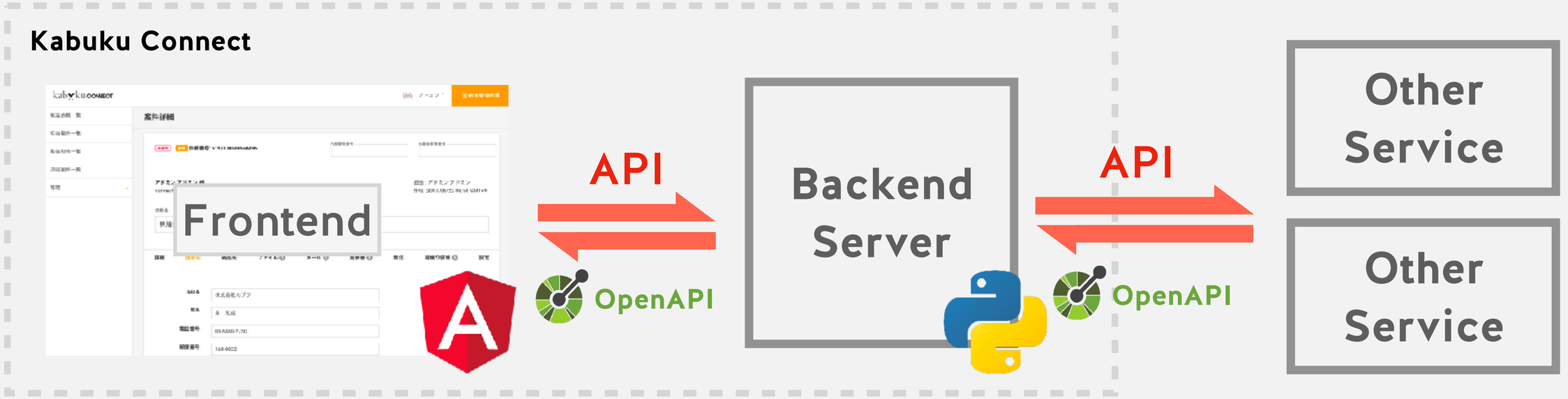
Architecture

Kabuku Connect

- Frontend: Angular (TypeScript)
- Backend: Python

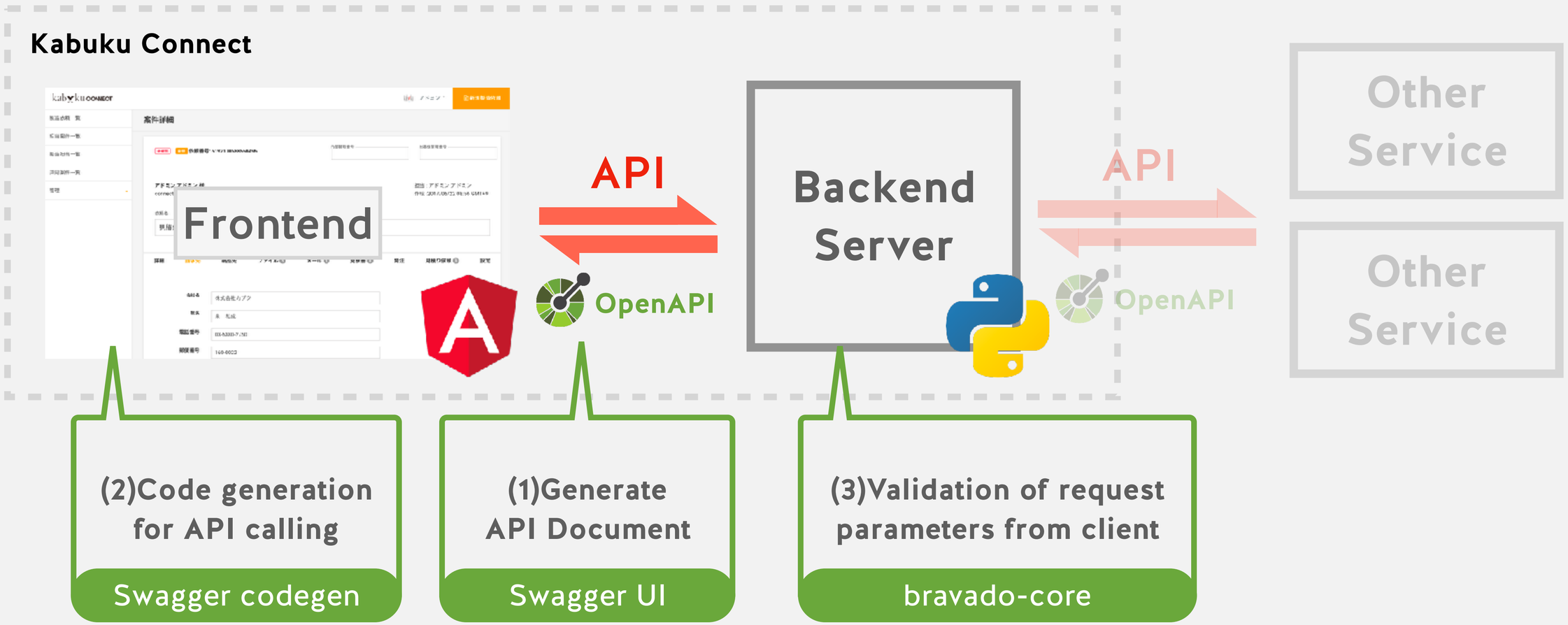
Other services

- Manufacturing management service
- Data analyzing service



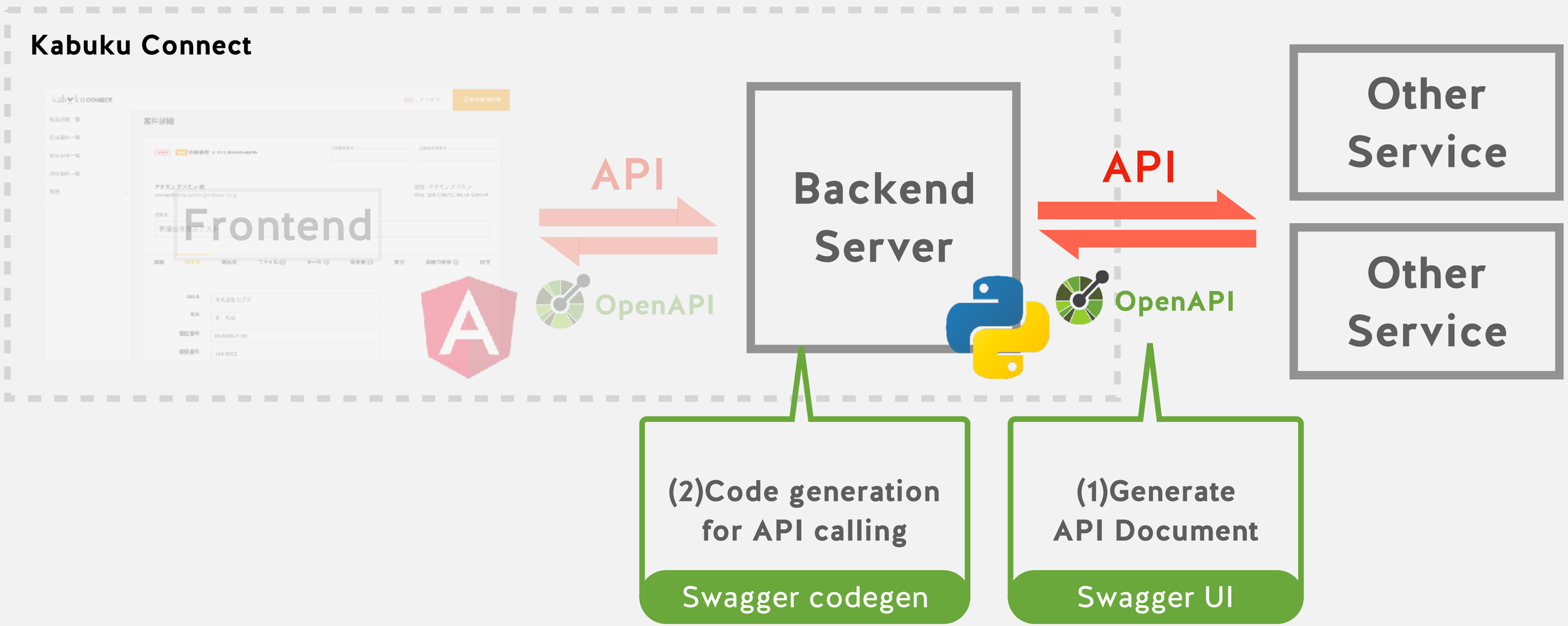
How OpenAPI is used

- In Kabuku Connect



How OpenAPI is used

- With other services



Implementation workflow

1. Design
   - API structure and write OpenAPI spec
2. Implementation
   - Frontend (Angular, Typescript)
     - Using generated clients code and mock server
   - Backend (Python)
   - Frontend and Backend can be implemented in parallel

Impression

- Using OpenAPI tools decrease your tasks so much
  - Document generation
  - Code generation
  - Frontend and Backend, API provider and A consumer can be implemented in parallel

Recap

Recap

- OpenAPI is a hot technology to describe API specification
- There are many tools to increase your productivity with OpenAPI
- You learned actual case with Open

Required more contributors!

- New Open API Spec (Version 3.0) will be released in July 2017
  - There are many added good features
  - Tools need to support OAS v3

We are hiring!

- Python Developer
- C++ Developer
- Frontend Developer
  - Angular/Three.js
- You can use 3D printers all you want
- International members
- 3 Google Developer Experts

https://www.kabuku.co.jp/en

Some 3D printed members