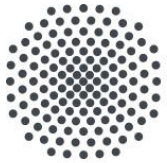
Quantum Web Services: Practical Session



University of Stuttgart

Jose Garcia-Alonso

Juan M. Murillo

jgaralo@unex.es juanmamu@unex.es

Institute of Architecture of Application Systems

UNIVERSIDAD de EXTREMADURA

Let’s go to the practical part!!

SCAN ME

Material of the examples and preparation of the environment

https://uex.be/24utorial

Introduction to quantum services with IBM Quantum

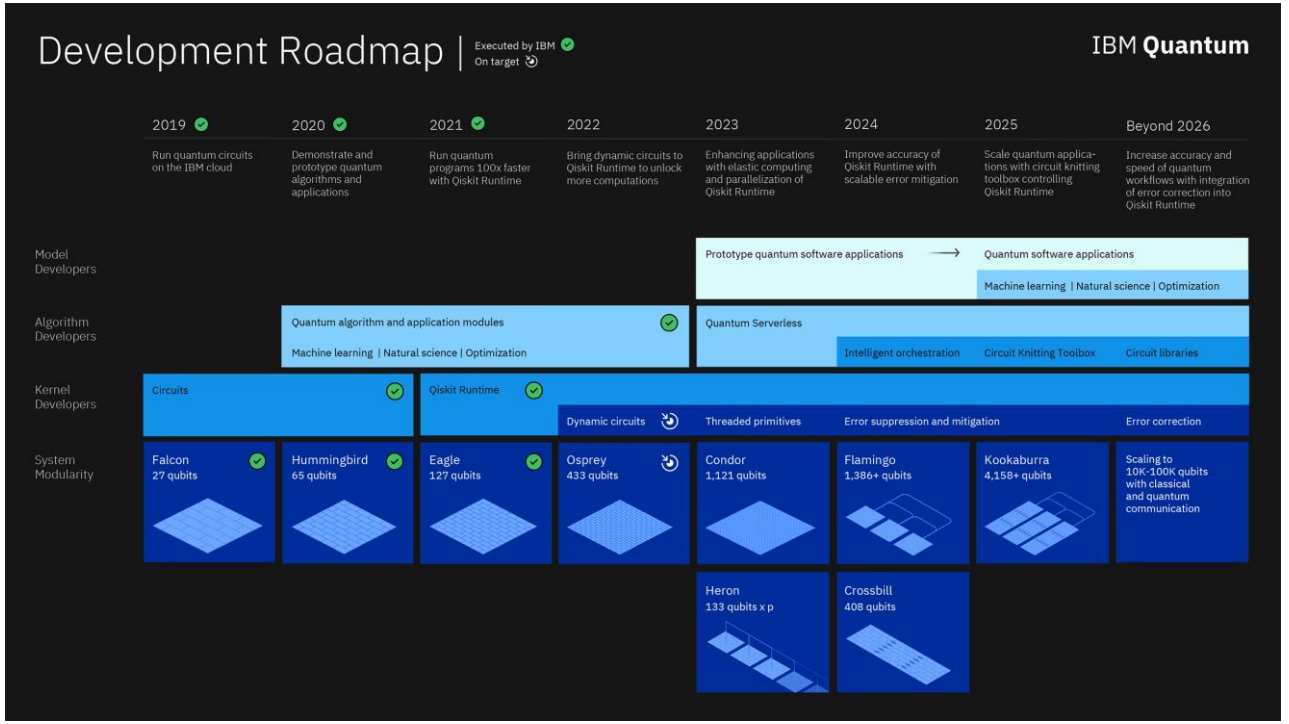
IBM Quantum **Platform**

Quantum Services with IBM Quantum

IBM Quantum

- IBM Quantum Platform
- Accelerating quantum computing research
- Access to various quantum computers and simulators
- Cloud programming using Python (**Qiskit**, **OpenQASM**)

IBM Q™



Quantum Services with IBM Quantum

Flask

- Framework available in Python
- Creation of web applications

- **Most famous alternative to Django** due to its reduced learning curve
- Widely used for the **definition of web services** based on REST APIs

Quantum Services with IBM Quantum

Other tools

Pandas

- Library for data handling and data processing with Python

Matplotlib

- Library for the creation of graphs and visual representations of data with Python.

Quantum Services with IBM Quantum

Preparing the enviroment

Quantum Services with IBM Quantum

Preparing the enviroment

1. Create an IBM Quantum account

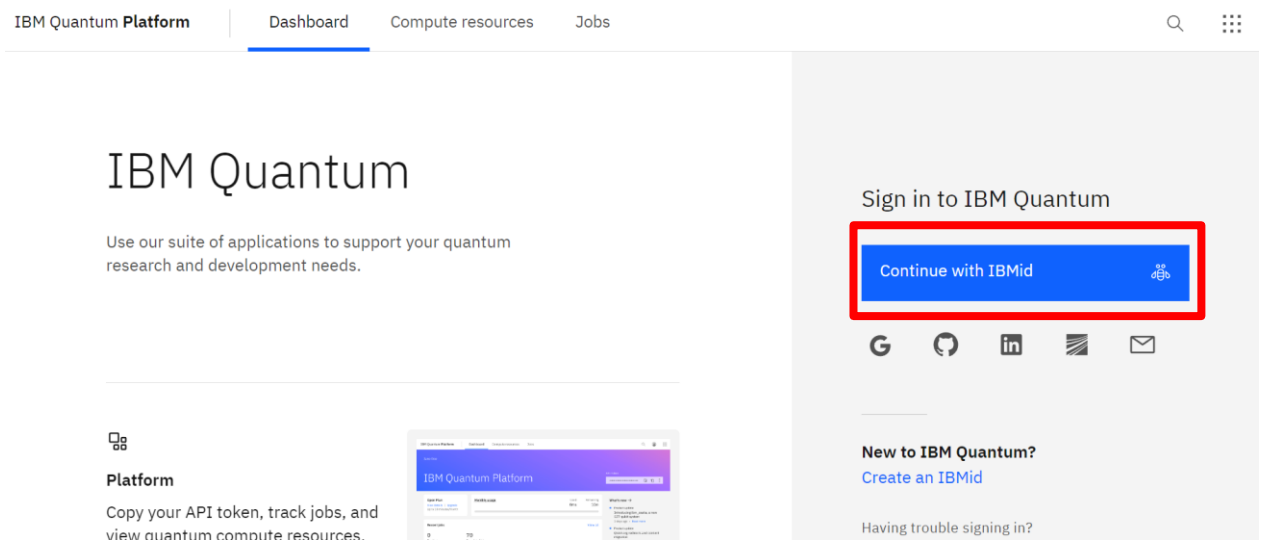
https://quantum.ibm.com/

Only if you want to run the services in real quantum machines

Quantum Services with IBM Quantum

Preparing the enviroment

1. Create an IBM Quantum account



Save API Token

Only if you want to run the services in real quantum machines

Quantum Services with IBM Quantum

Local installation

2. Installing qiskit libraries, Flask and other tools

```
cd Shor

pip install –r requirements.txt

python -c "from qiskit_ibm_provider import IBMProvider; IBMProvider.save_account(token='TOKEN')"
```

SCAN ME

https://uex.be/24utorial

Quantum Services with IBM Quantum

Quantum Services Examples

ICWE Tutorial

Shor's Algorithm reminder

```python
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

def shor_circuit():
    qreg_q = QuantumRegister(8, 'q')
    creg_c = ClassicalRegister(4, 'c')
    shor = QuantumCircuit(qreg_q, creg_c)  # Use 8 qubits and 4 classical bits

    shor.h(range(4))
    shor.x(4)
    shor.cx(0, 5)
    shor.cx(0, 6)
    shor.cx(1, 4)
    shor.cx(1, 6)
    for i in range(4, 8):
        shor.ccx(0, 1, i)


    return shor
```

```python
# Execute circuit
def run(machine, shots):
    circuit = create_cir()
    if machine == "local":
        backend = BasicProvider().get_backend("basic_simulator")
        x = int(shots)
        transpiled_circuit = transpile(circuit, backend)
        job = backend.run(transpiled_circuit, shots=x)
        result = job.result()
        counts = result.get_counts()
        print(counts)
        x, y = factor(counts)
        return [x, y]
    else:
        provider = IBMProvider()
        backend = provider.get_backend(machine)
        x = int(shots)
        job = backend.run(circuit, backend, shots=x)
        result = job.result()
        counts = result.get_counts()
        x, y = factor(counts)
        return [x, y]
```

Quantum Services Examples

Shor's Algorithm reminder

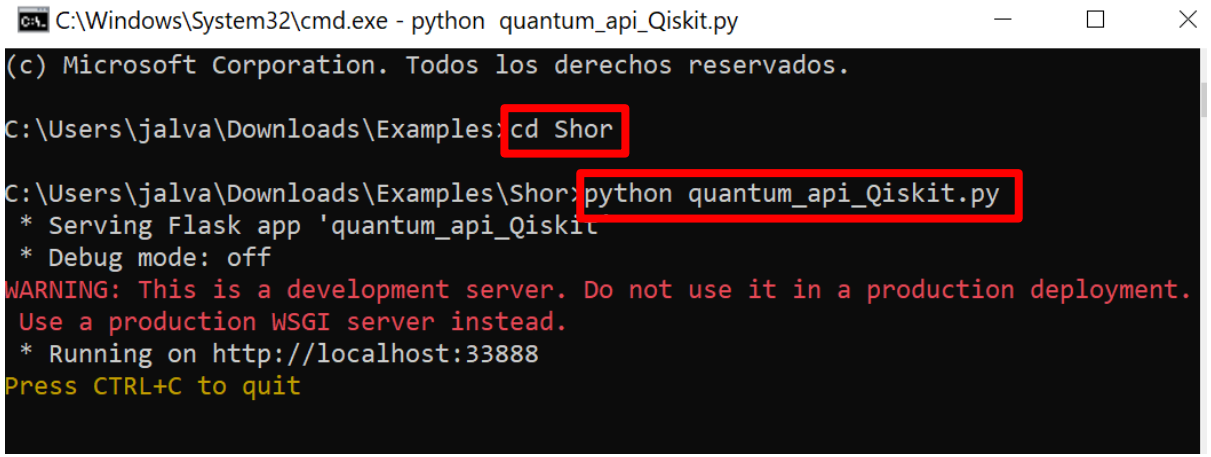
API Deployment

1. Run the follow commands in your system terminal:

- cd Shor
- python quantum_api_Qiskit.py

  or

  python3 quantum_api_Qiskit.py



Quantum Services Examples

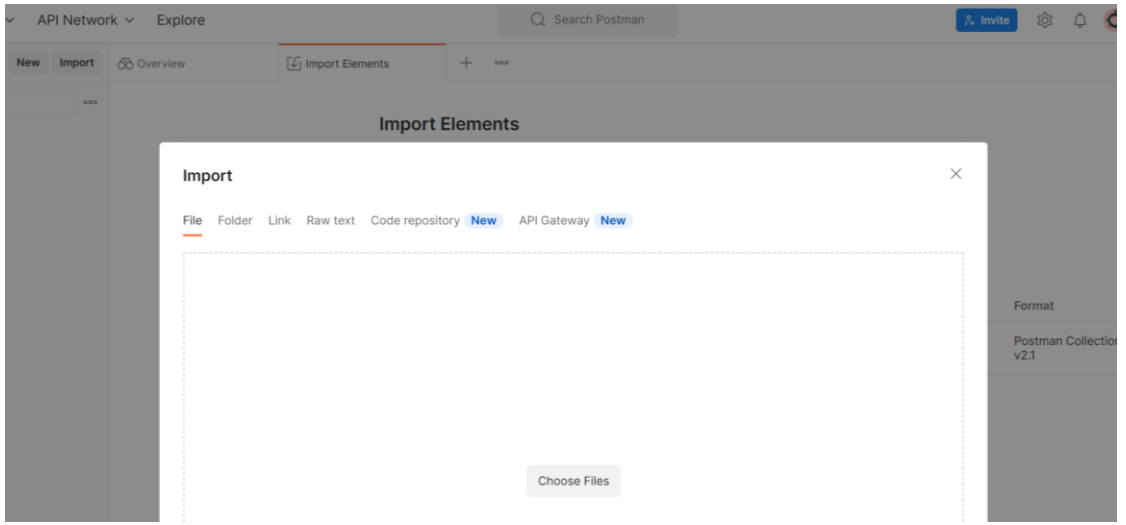
Shor's Algorithm reminder

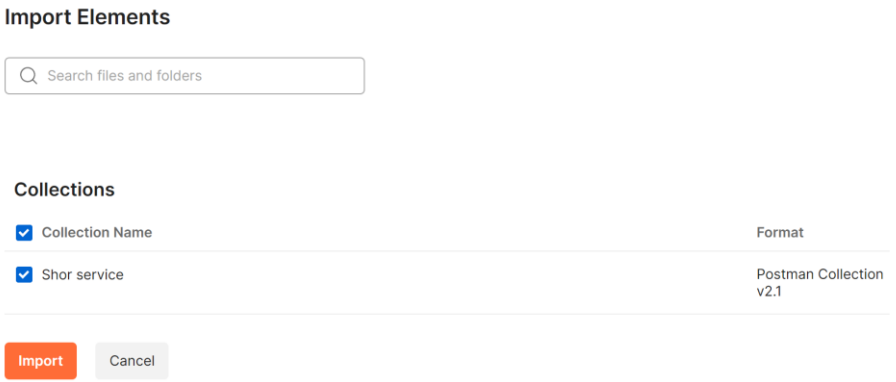
API Deployment

2. Download Postman:

https://www.postman.com/downloads/

3. Import collection file (Shor service ICWE Tutorial.postman_collection) from Shor folder:



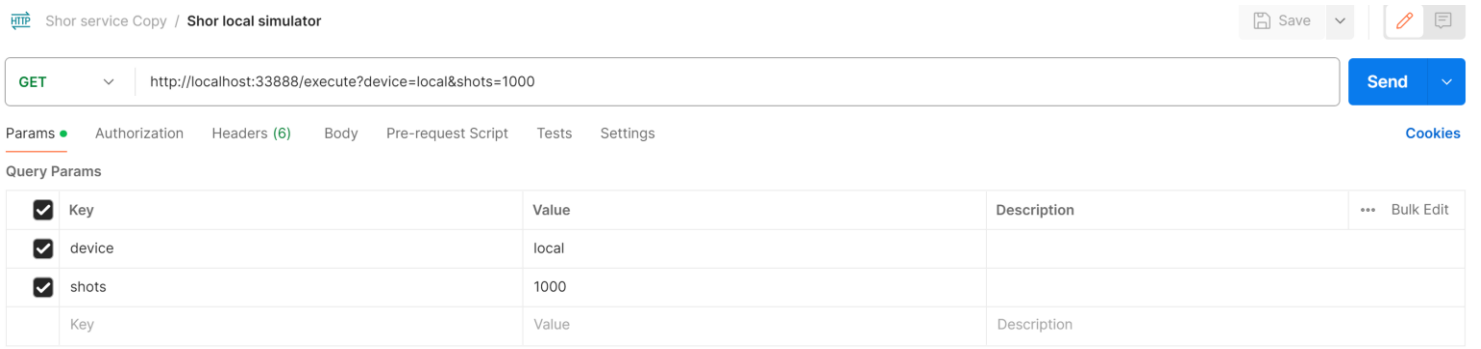


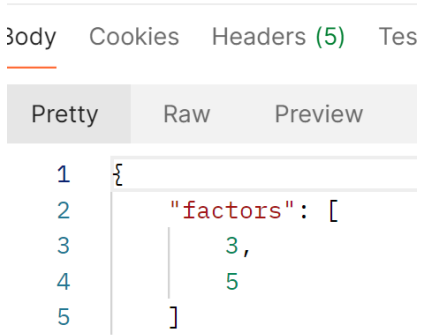
Quantum Services Examples

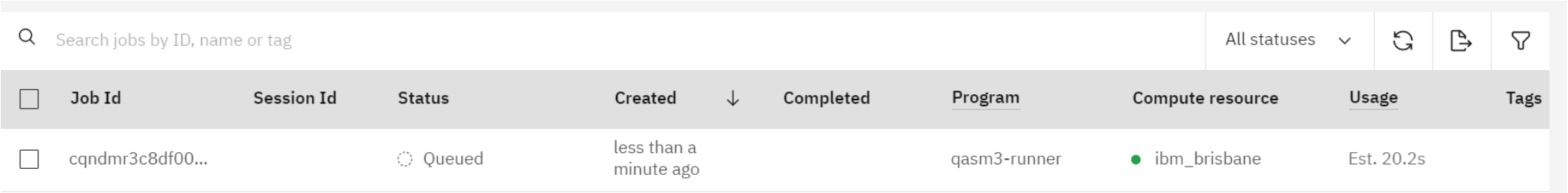
Shor's Algorithm reminder
API Deployment

4. Test the shor service in postman:

- **Shor local simulator:** http://localhost:33888/execute?device=local&shots=1000
- **Shor real device:** http://localhost:33888/execute?device=ibm_brisbane&shots=1000







Quantum services generation and deployment using OpenAPI Specification


OpenAPI

OpenAPI

OpenAPI Tools

- **Editor**

http://editor.swagger.io

- **API explorer**

http://petstore.swagger.io

- **Validator**

https://online.swagger.io/validator

- **Opensource generator:**
  - skeletons para backends
  - proxies para clientes o front-end

https://openapi-generator.tech/

Quantum services generation

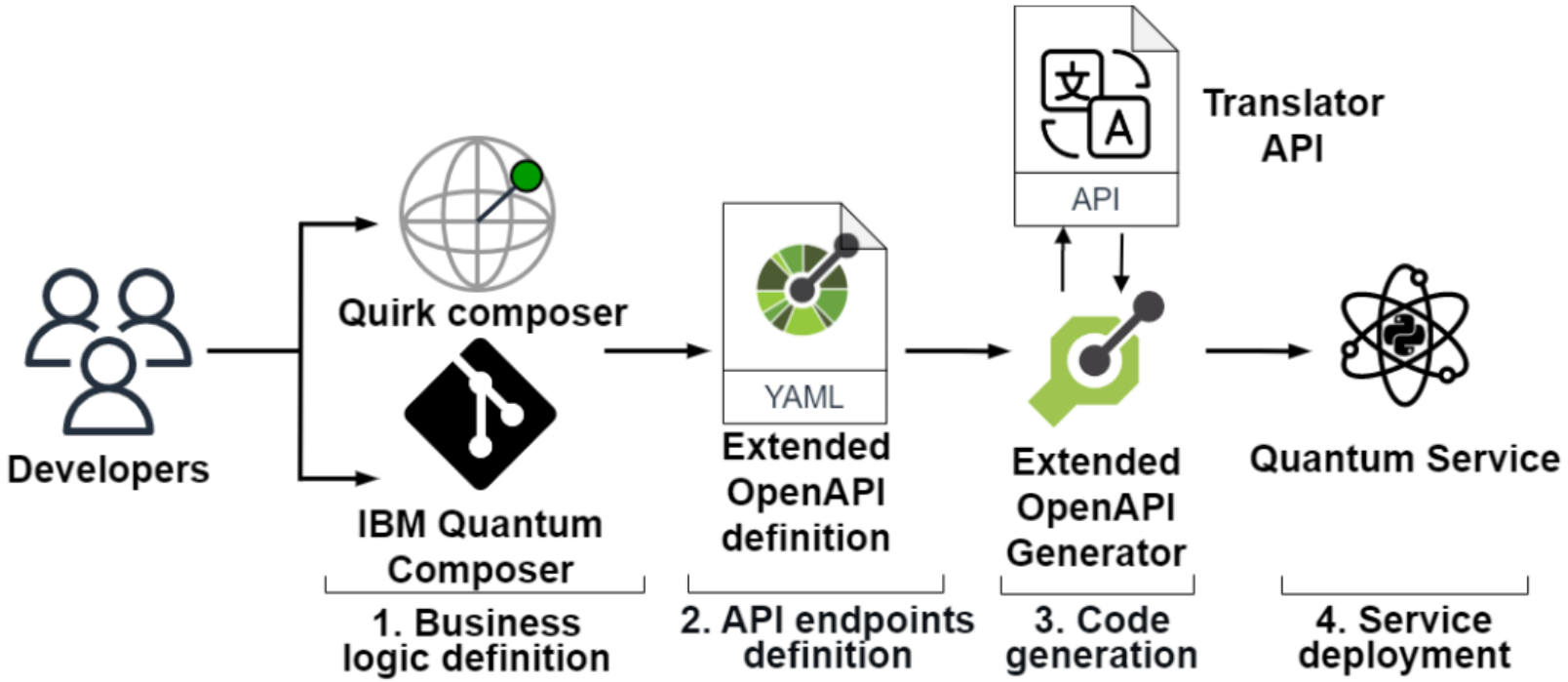
To implement a classic service using OpenAPI, a developer needs to combine two main aspects:

- **The business logic** of the service, which implements the functionality provided by the service.
- **The endpoint** of the service, which determines how an external client can invoke the service.

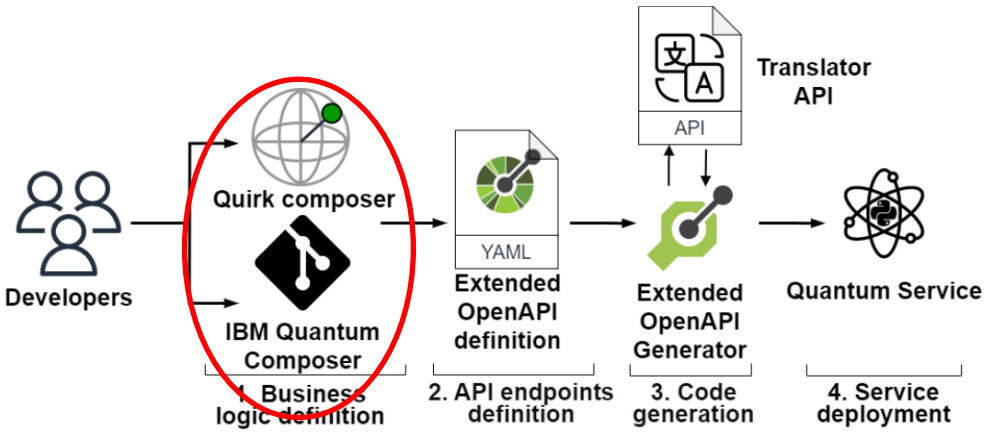
GitHub – OpenAPI Generator Quantum

Quantum services generation

Defining a model for the servitisation of quantum algorithms by extending the OpenAPI specification



Quantum services generation

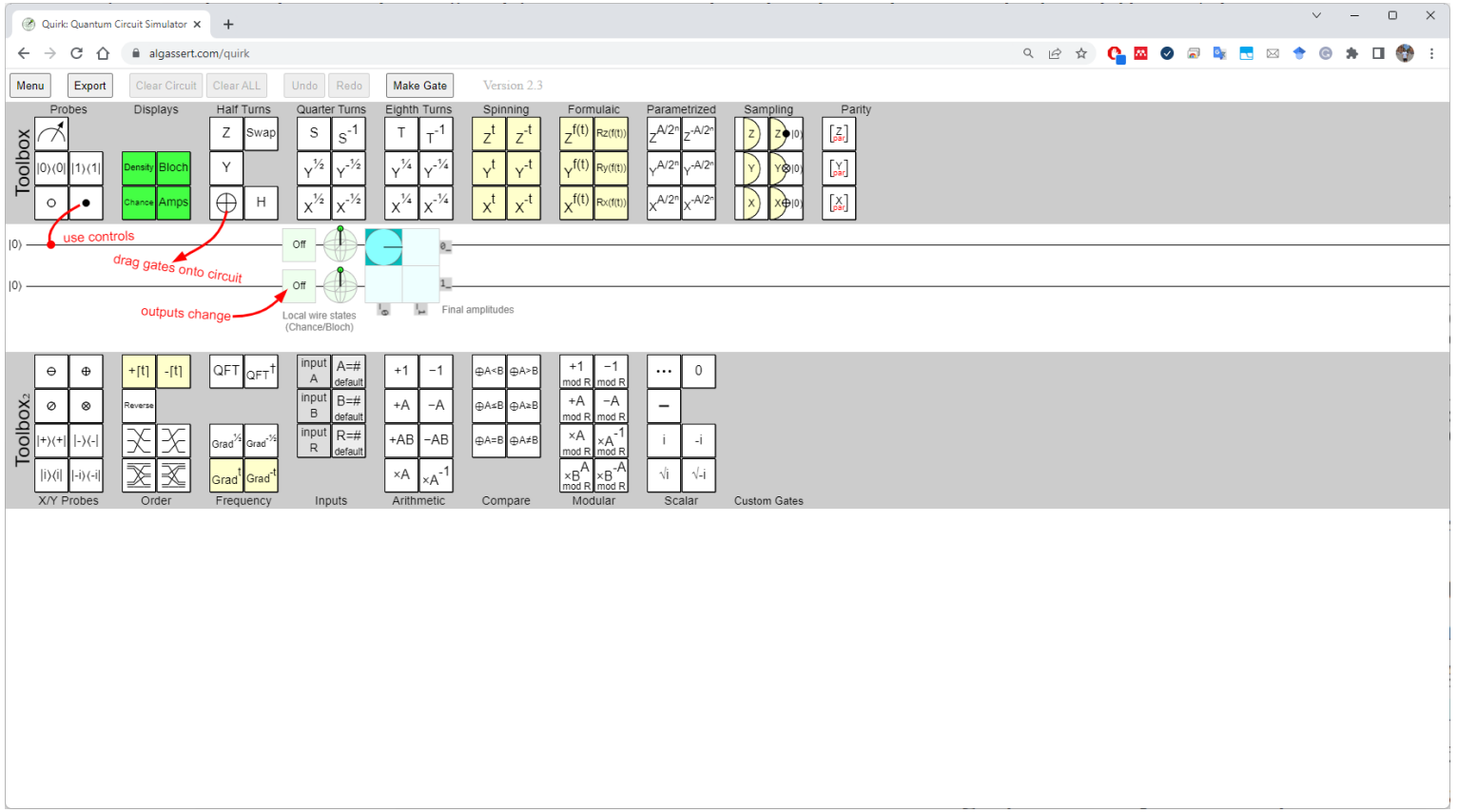


The first step is to define the business logic of the service as a quantum circuit using **Open Quirk**, indicating the Open Quirk link of the created circuit. Or directly indicating the link where is the **source code in Qiskit**

Quantum services generation

OpenAPI for quantum algorithm servitisation

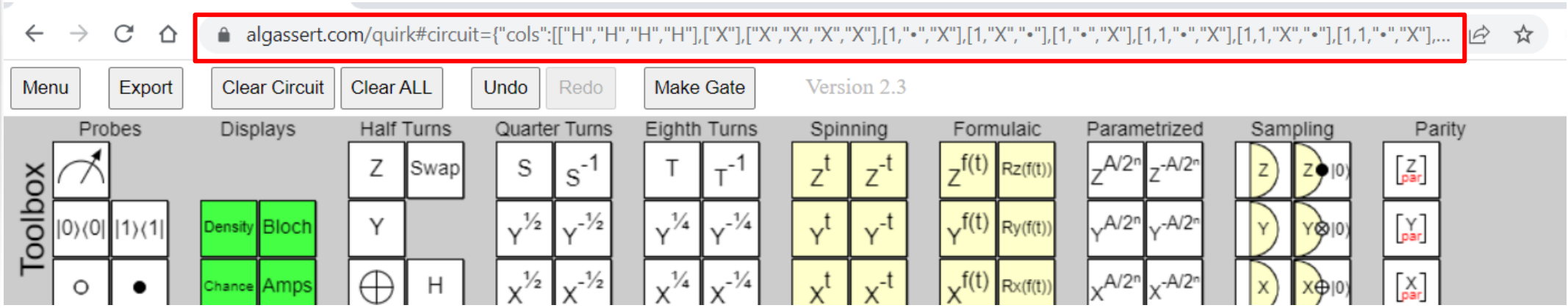
Definition of the circuit corresponding to the Quantum Algorithm with Quirk



Quantum services generation

OpenAPI for quantum algorithm servitisation

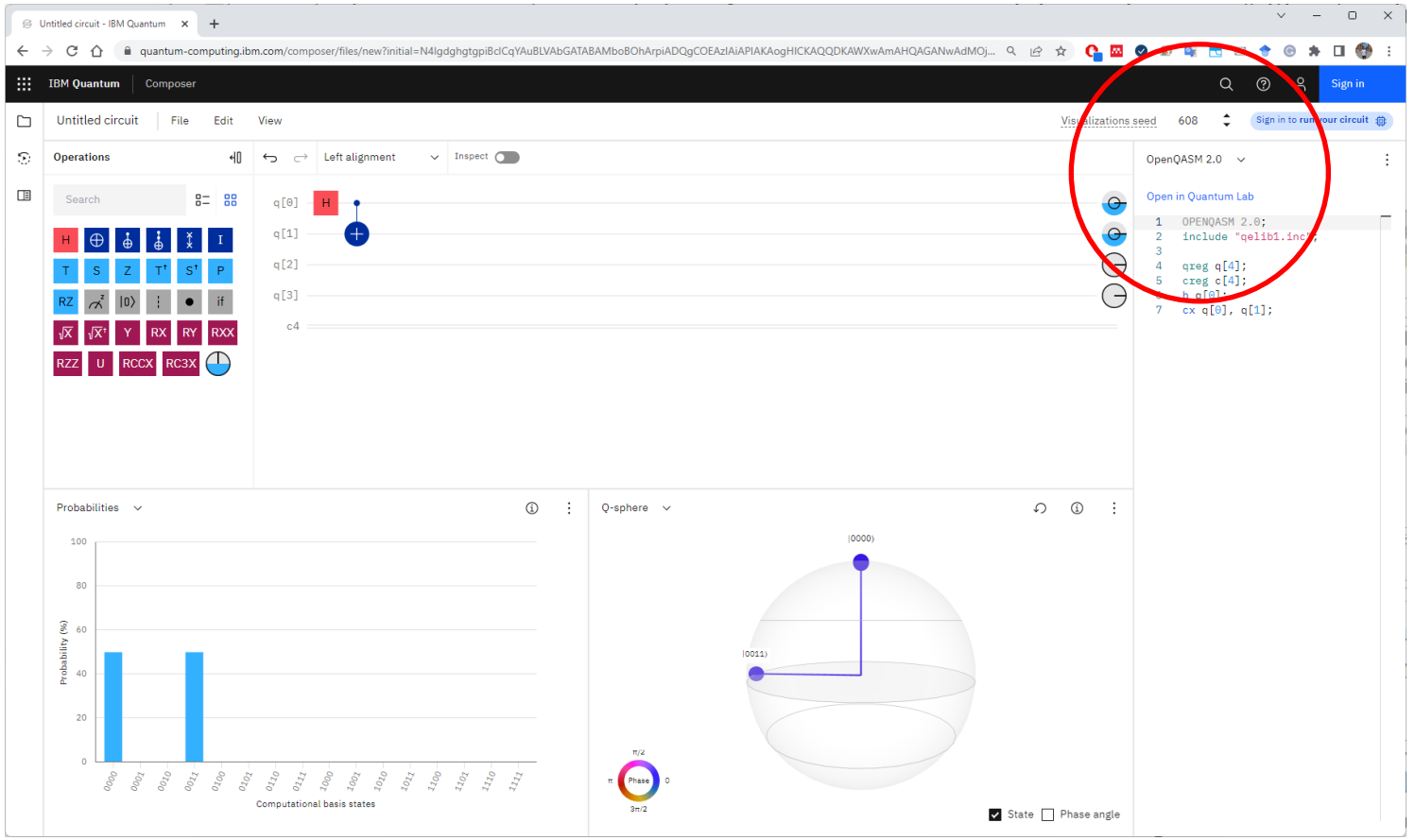
Definition of the circuit corresponding to the Quantum Algorithm with Quirk



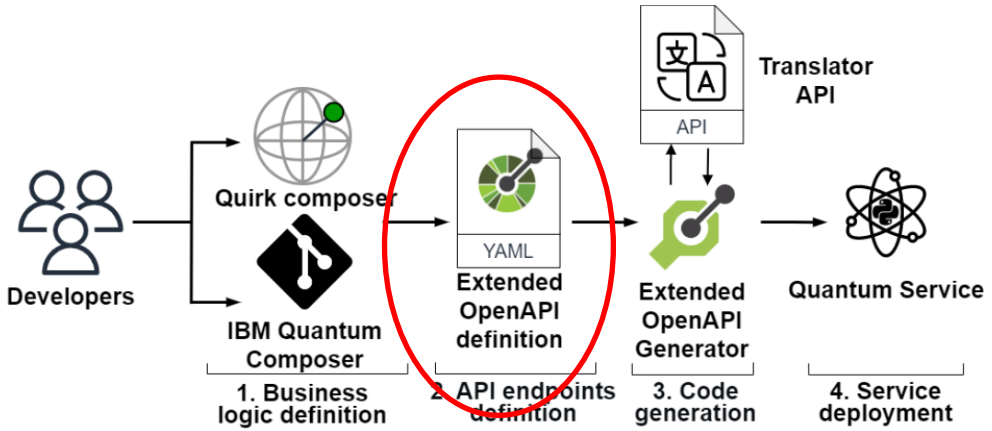
Quantum services generation

OpenAPI for quantum algorithm servitisation

Definition of the circuit corresponding to the Quantum Algorithm with IBM Quantum Composer



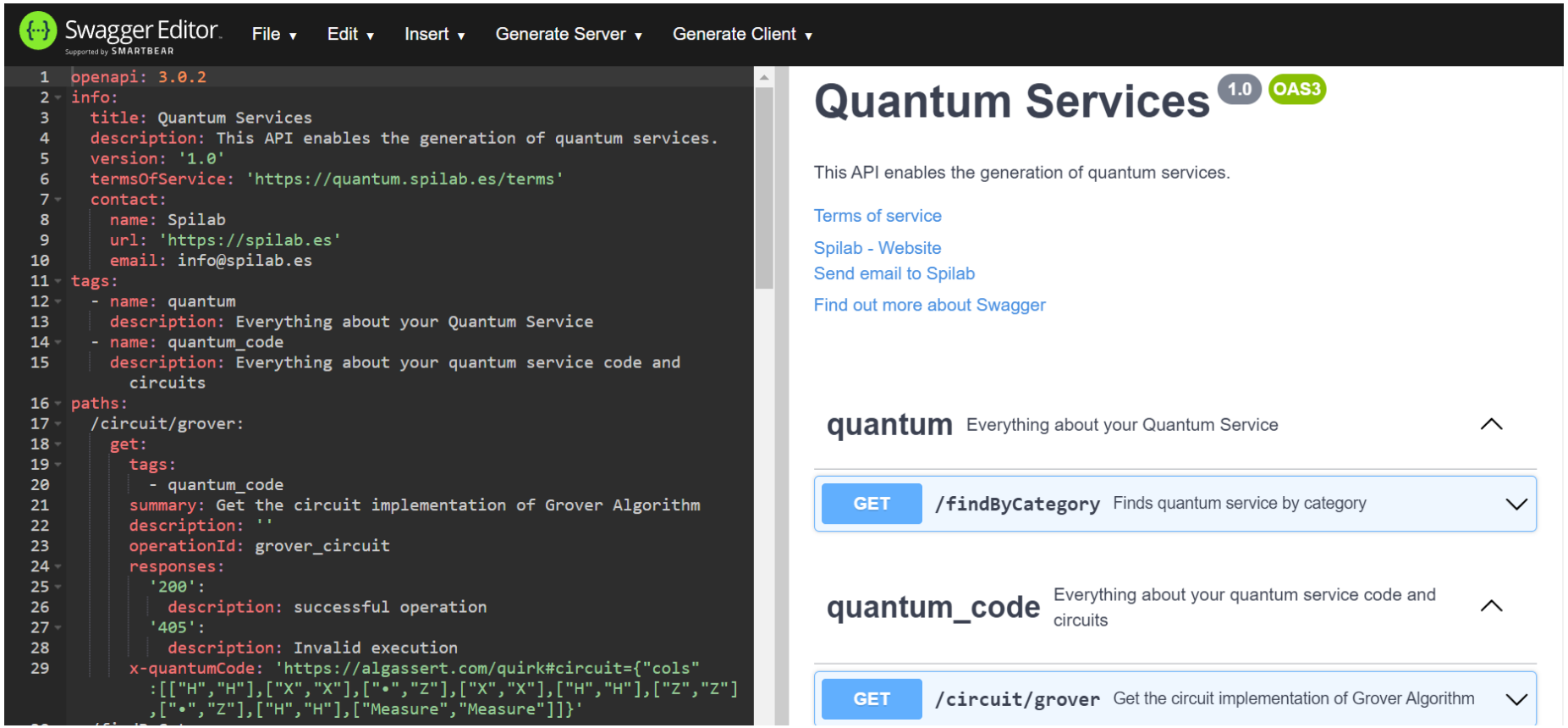
Quantum services generation



In the second step, the quantum API is defined, for which an API contract must be established with OpenAPI.

Quantum services generation

Definition of the API contract, in YAML format, using the Swagger Editor



Quantum services generation

Two custom variables, one to indicate the link to the quantum circuit, and the other to indicate the provider where we want to run it.

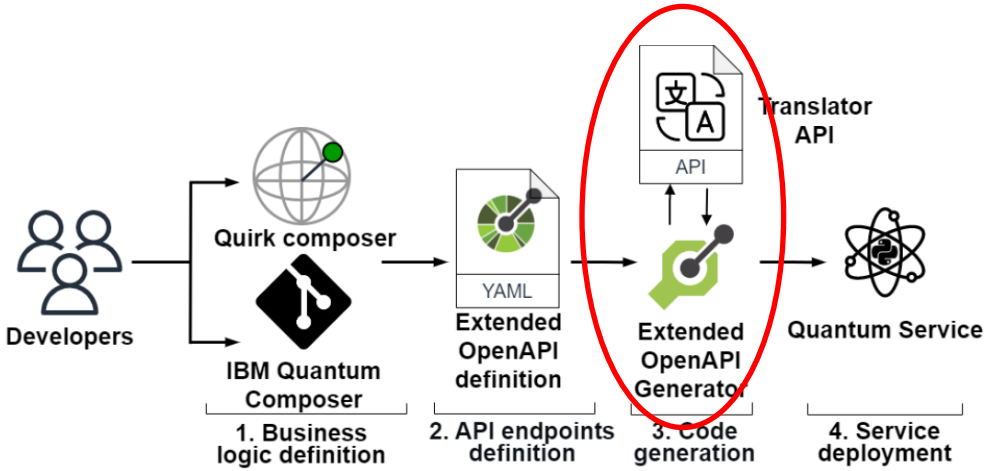
```
37            required: true
38            style: form
39            explode: false
40            deprecated: true
41            schema:
42              type: number
43        responses:
44          '200':
45            description: successful operation
46          '405':
47            description: Invalid execution
48        x-quantumCode: 'QISKIT-URL'
49        x-quantumProvider: 'aws'
50    /circuit/ShorIBMqiskit:
51      get:
52        tags:
```

Quantum services generation

Two custom variables, one to indicate the link to the quantum circuit, and the other to indicate the provider where we want to run it.

```
paths:
  /circuit/randomAWS:
    get:
      tags:
        - quantum_code
      summary: Get the circuit implementation of Grover Algorithm
      description: ''
      operationId: grover_circuitAWS
      parameters:
        - name: machine
        - name: shots
      responses:
        '200':
          description: successful operation
        '405':
          description: Invalid execution
      x-quantumCode: 'QISKIT-URL'
      x-quantumProvider: 'aws'
```

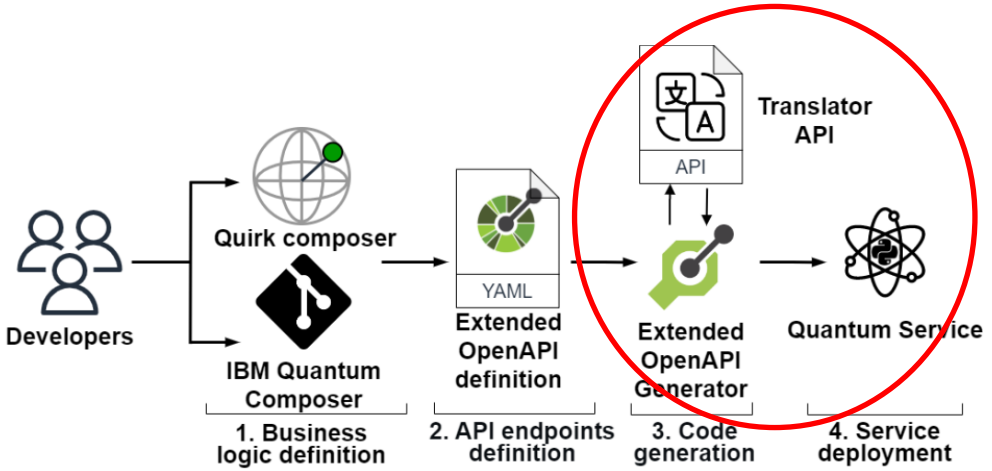
Quantum services generation



In the third step, the extension of the OpenAPI Code Generator is used to generate the source code of the quantum services.

It uses our Translator API to compose the code from the provided quantum circuit for the selected provider.

Quantum services generation



PATH PARAMETERS

*framework string: python-quantum

framework

BODY DATA (required)

parameters

EXAMPLE MODEL application/json

```
{
  "openAPIUrl":
"https://raw.githubusercontent.com\QuantumOpenAPI\main\openapi_quant
um.yaml",


}
```

```python
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit, execute,IBMQ
```

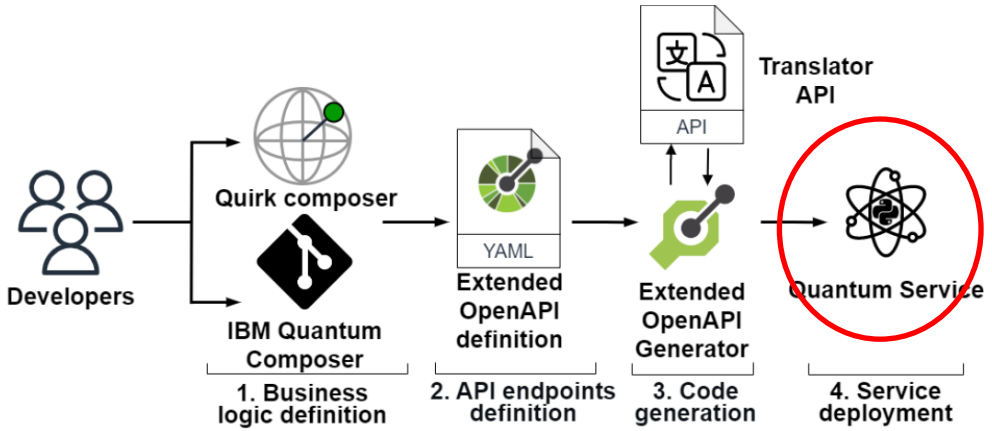
Qiskit libraries for quantum computing

```python
def random_circuit(api_token):  # noqa: E501

    """Get the circuit implementation for random numbers

     # noqa: E501

    :param api_token: API Token
    :type api_token: str

    :rtype: None
    """
    IBMQ.enable_account(api_token)
    provider = IBMQ.get_provider(hub='ibm-q')

    q = QuantumRegister(16,'q')
    c = ClassicalRegister(16,'c')
    circuit = QuantumCircuit(q,c)
    circuit.h(q) # Applies hadamard gate to all qubits
    circuit.measure(q,c) # Measures all qubits

    backend = provider.get_backend('ibmq_qasm_simulator')
    job = execute(circuit, backend, shots=1)

    result = job.result()
    counts = result.get_counts(circuit)

    return counts
```

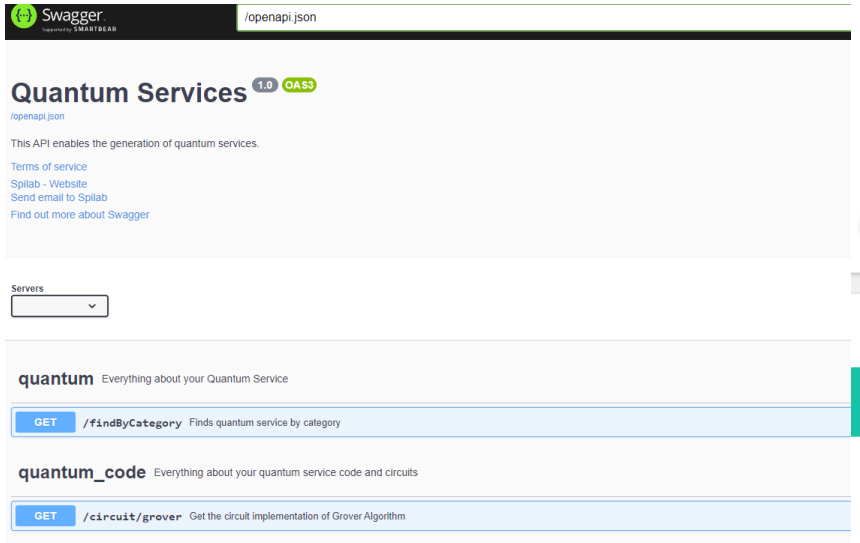
Classical wrapping endpoint service

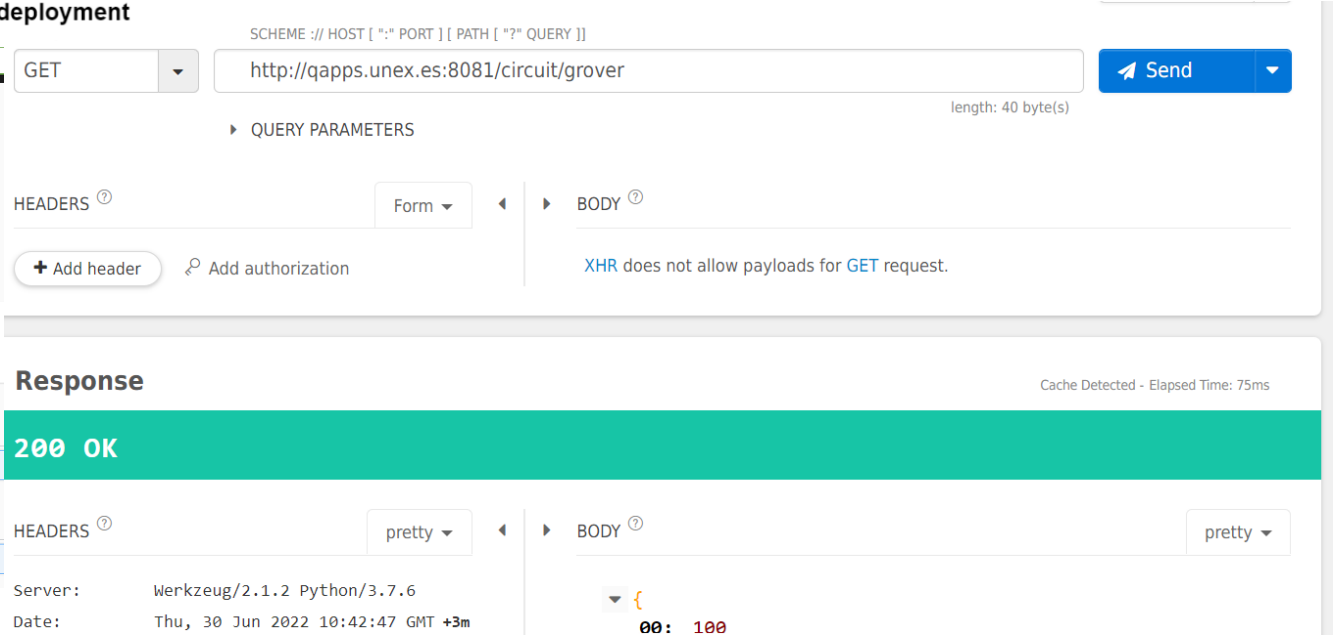
Random Numbers quantum circuit

Quantum services generation



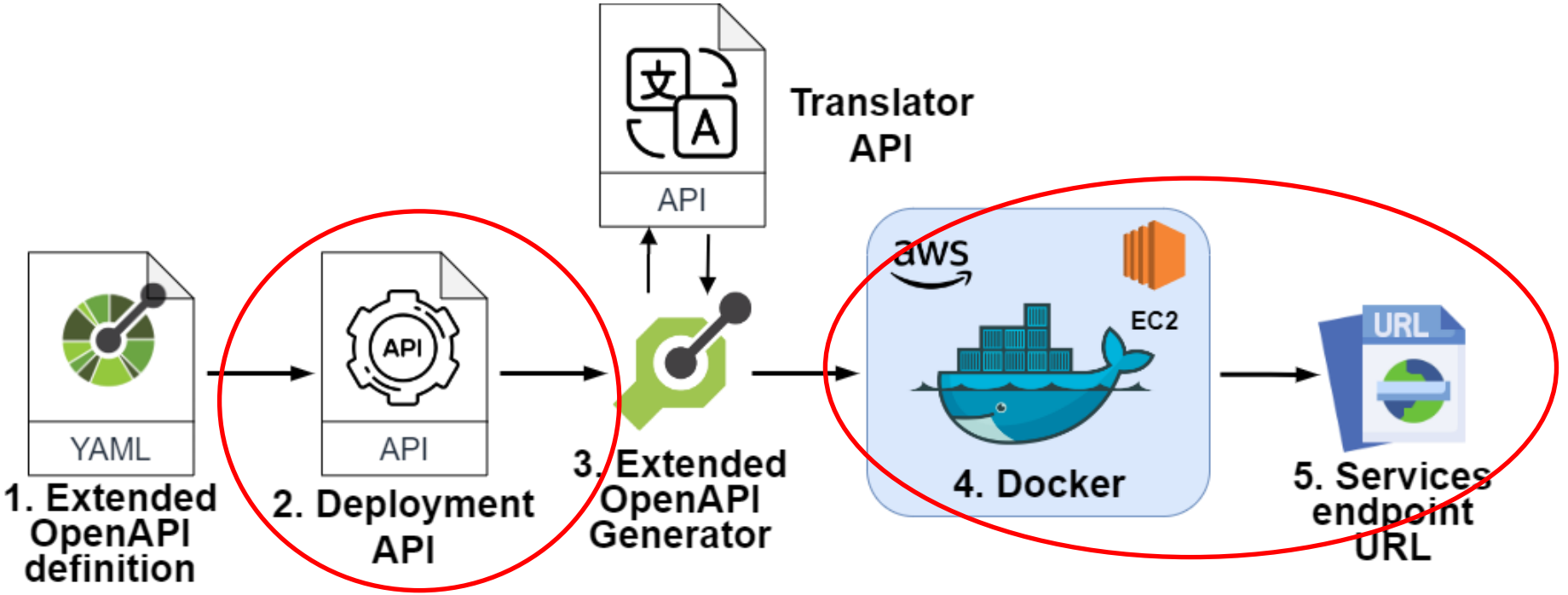
These generated quantum services can be deployed and invoked (fourth step) via RESTful calls to the endpoints encapsulating the quantum algorithm code.



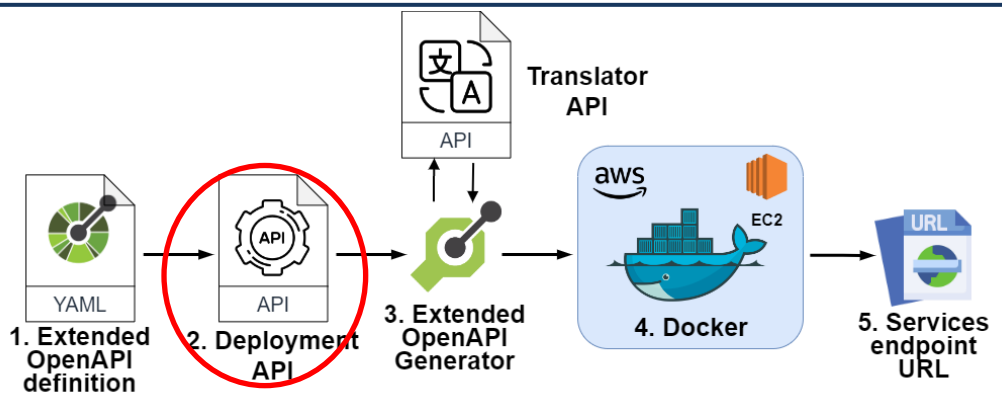


Quantum service Deployment

Once we have generated quantum services, which can be deployed manually, why not do something to automate it?

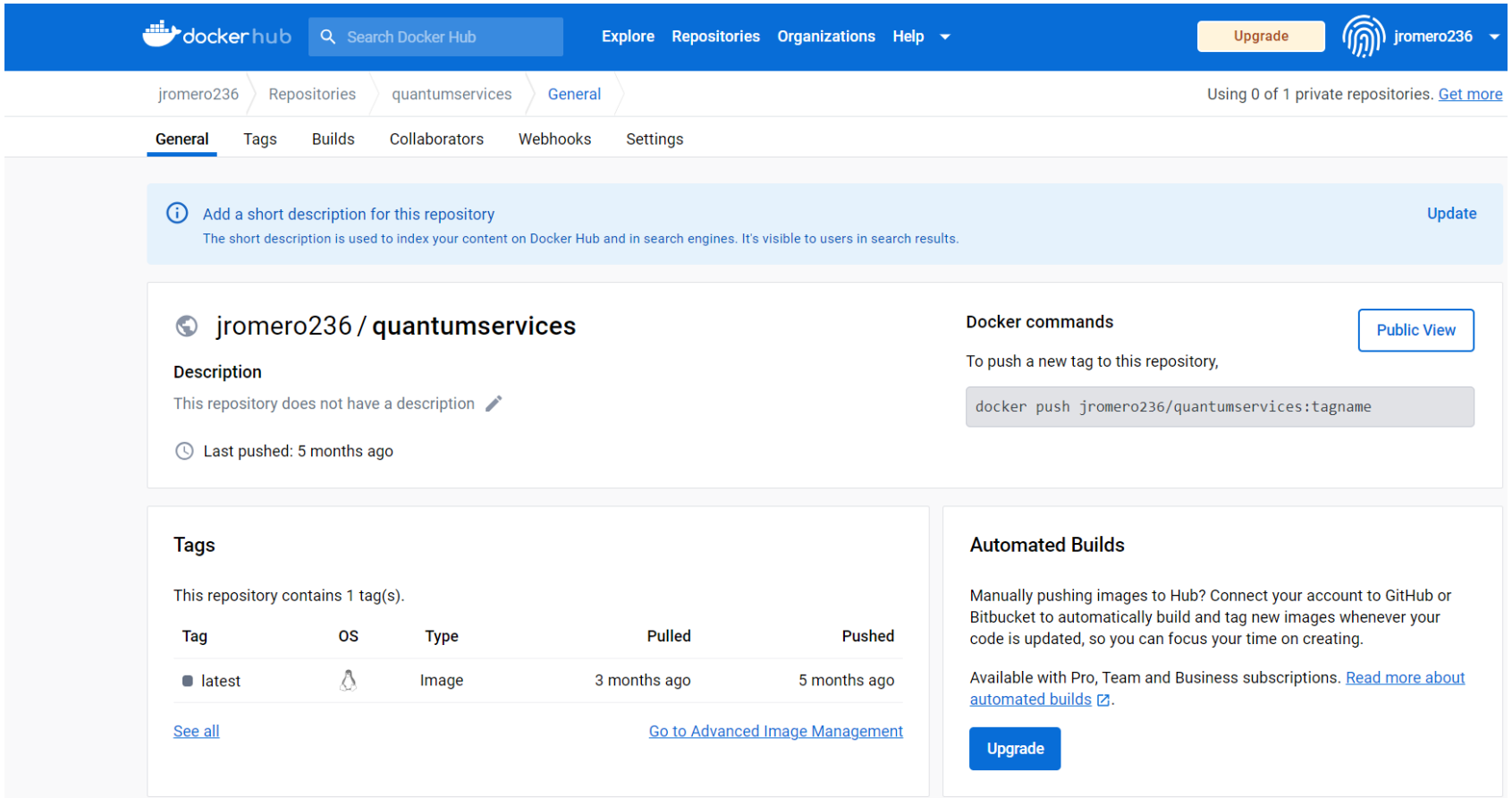


Quantum service Deployment

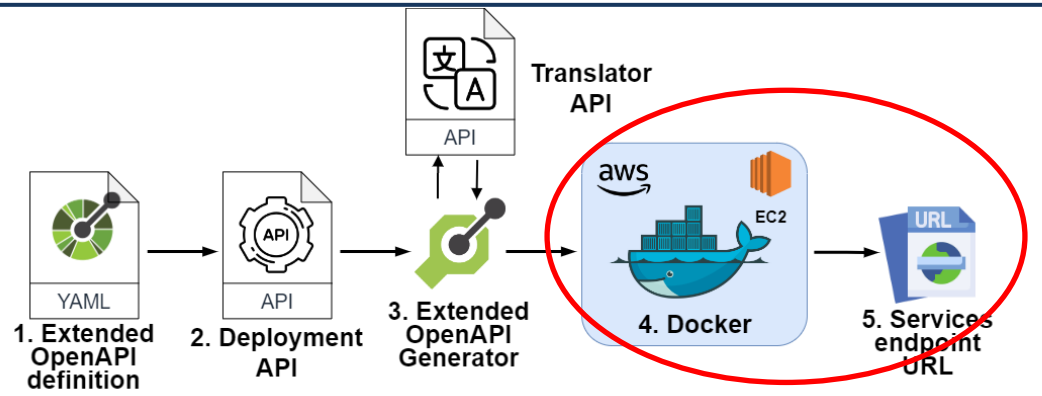


The Deployment API receives the YAML file based on the modification of the previous OpenAPI specification and the necessary parameters for the configuration of the providers (e.g. AWS keys).

Quantum service Deployment



Quantum service Deployment

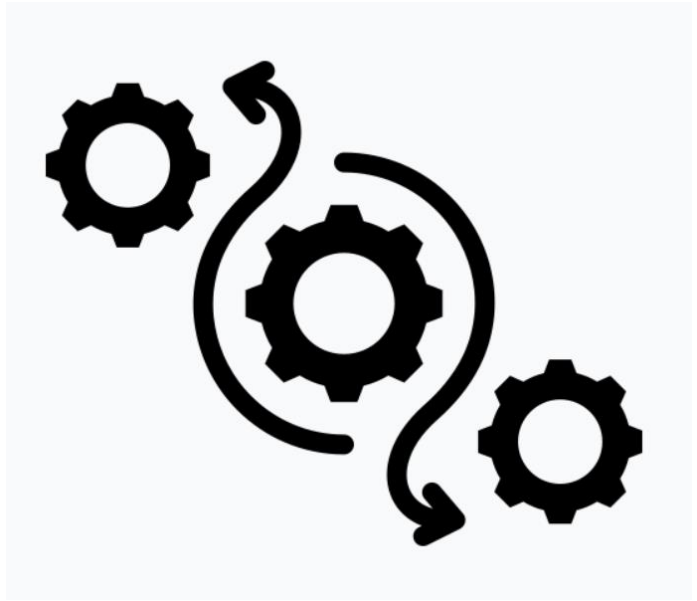


Body Cookies Headers (5) Test Results Status: 200 OK Time: 8.62 s Size: 304 B Save Response

Pretty Raw Preview Visualize HTML

```
1  <html>
2  <h1 href='http://quantumservicesdeployment.spilab.es:8088/ui'>http://quantumservicesdeployment.spilab.es:8088/ui<\h1>
3      <\html>
```

Why not bring **generation** and **deployment** together and **automate** it?



Workflow for the continuous deployment

Workflow for the continuous deployment

GitHub Actions

1.Automation: GitHub Actions allows you to automate workflows in software development projects.

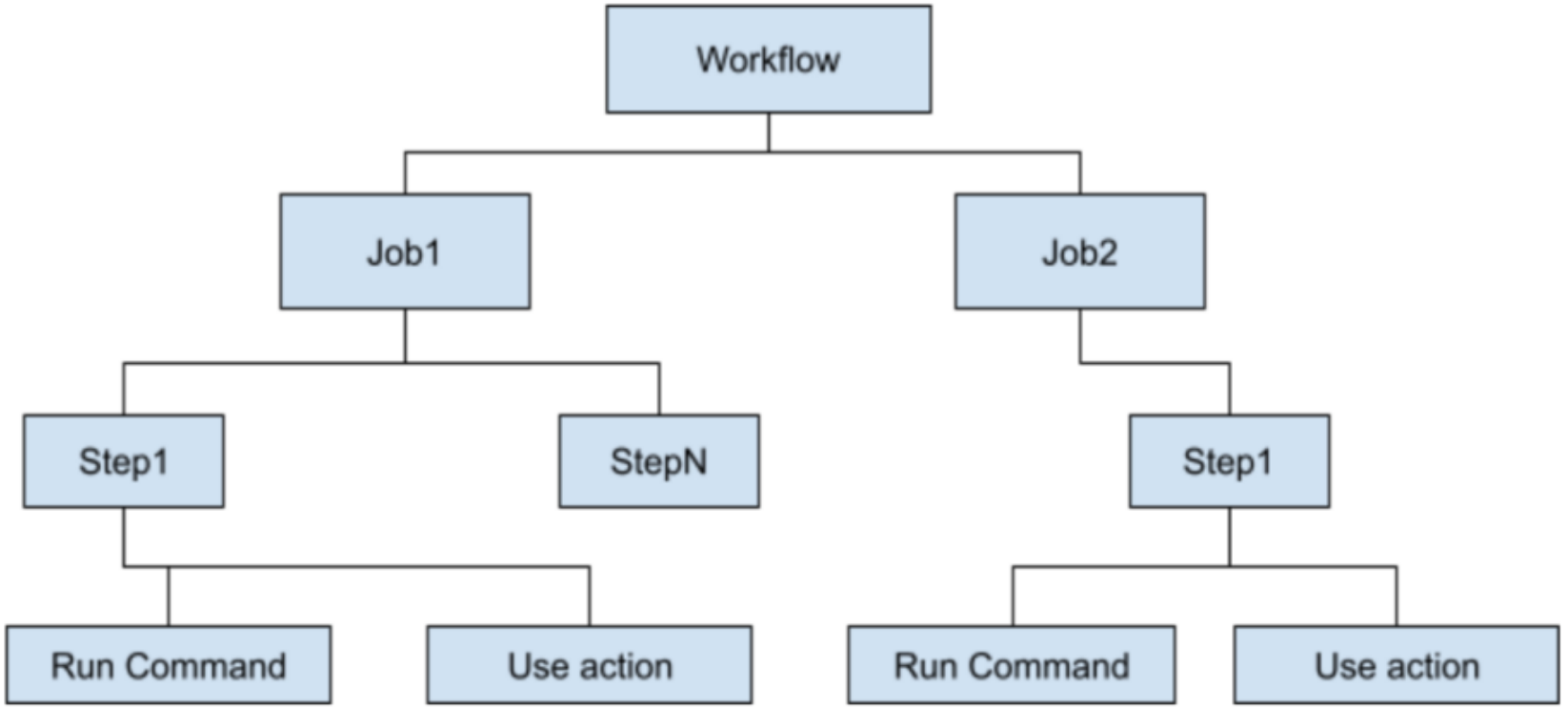
2.YAML-based Configuration: Workflows are defined using YAML configuration files stored in the repository.

3.Event-Driven: Workflows can be triggered by events like code pushes, pull requests, or scheduled intervals.

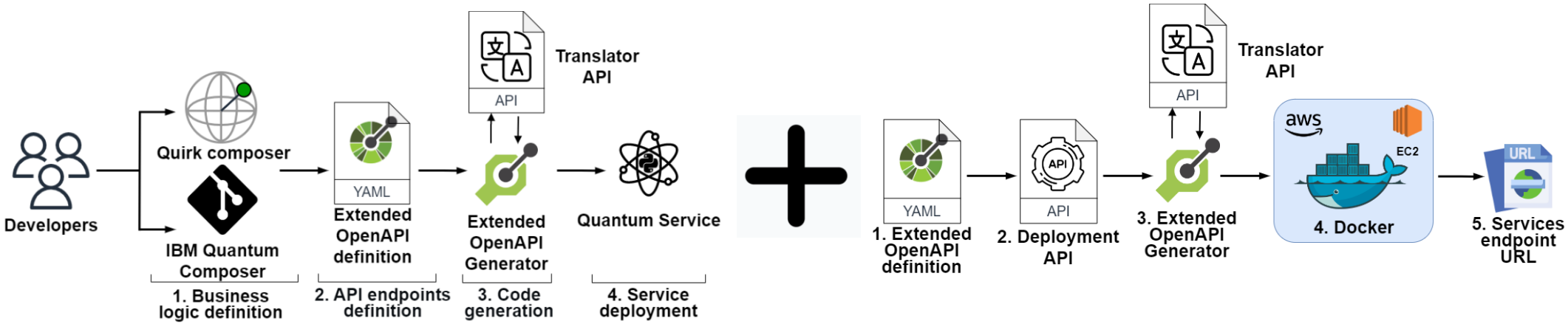
4.Integrations: GitHub Actions seamlessly integrates with other GitHub features, enabling collaboration and streamlining the development process.

Workflow for the continuous deployment

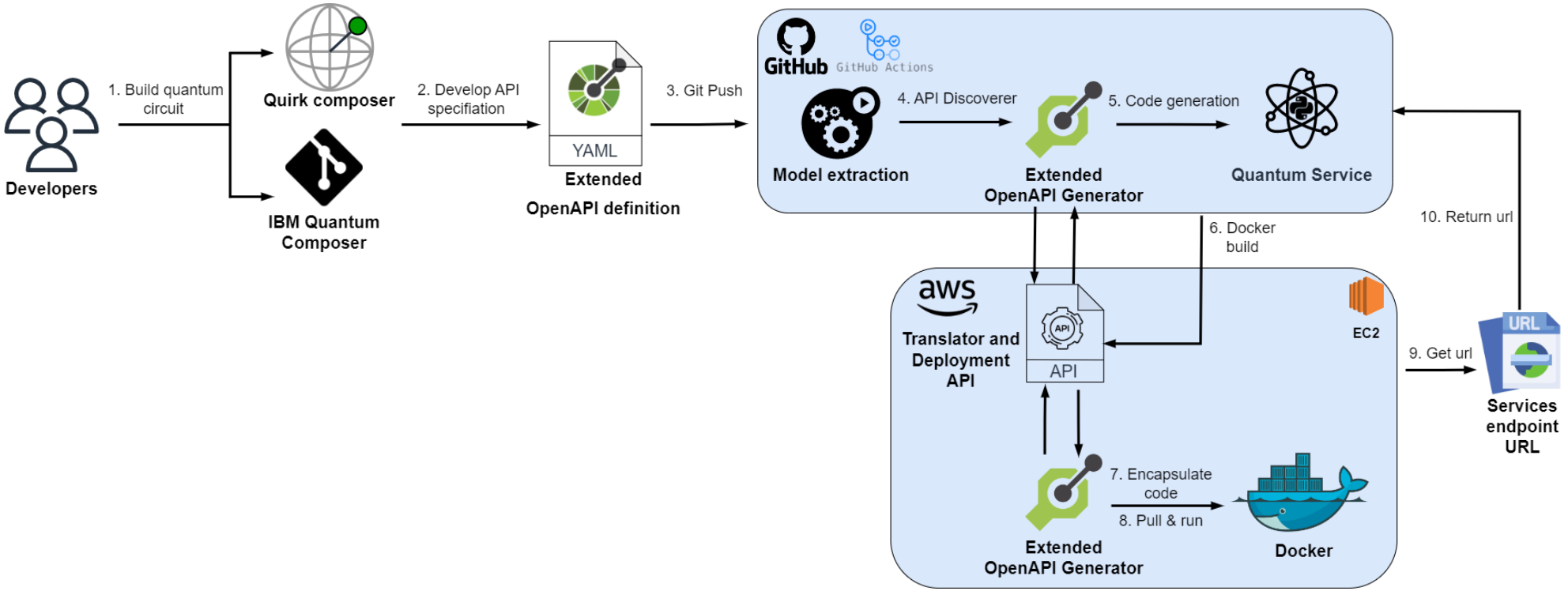
GitHub Actions



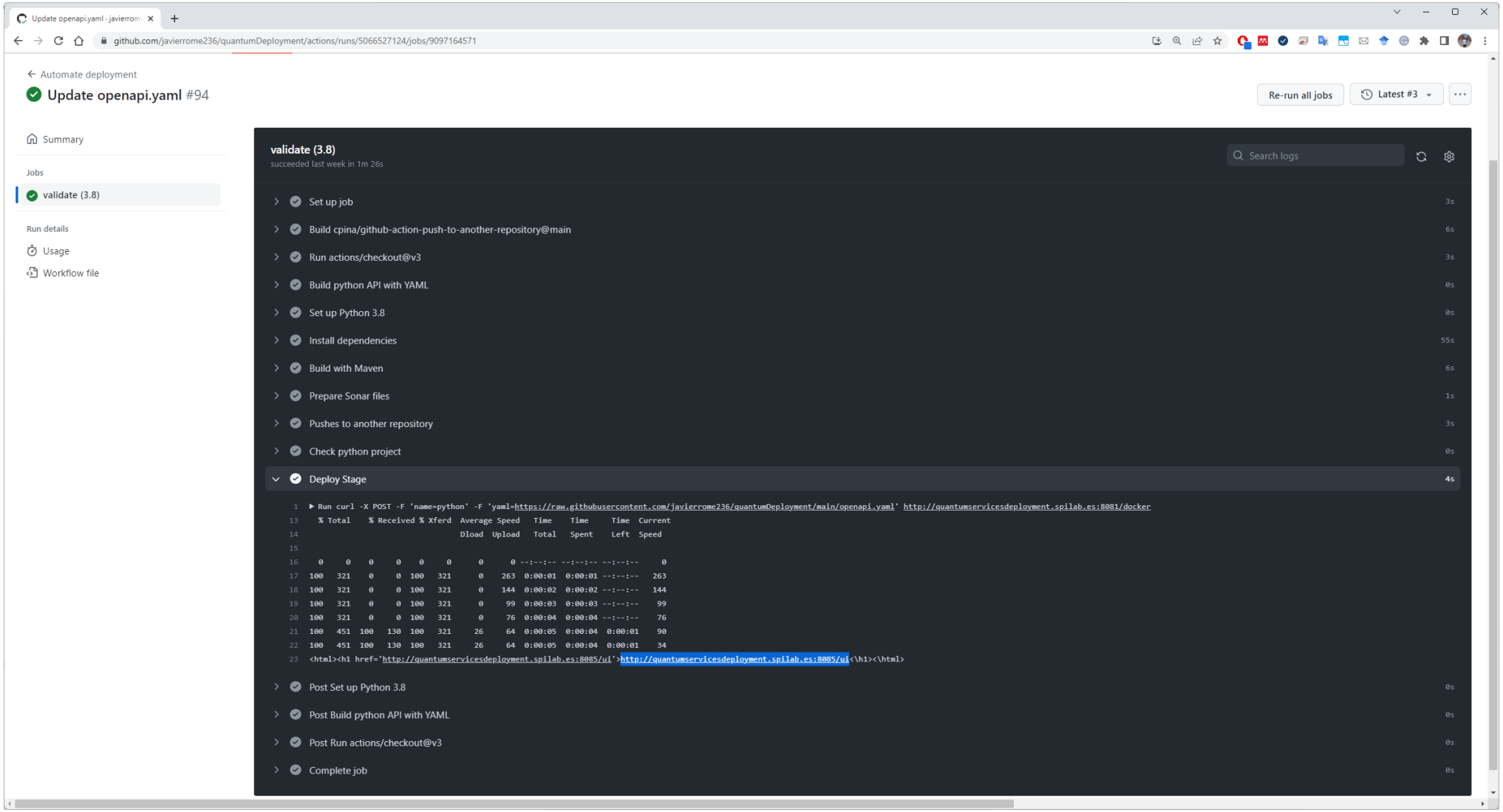
Workflow for the continuous deployment



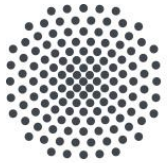
Workflow for the continuous deployment



Workflow for the continuous deployment



Quantum Web Services: Practical Session



University of Stuttgart

Jose Garcia-Alonso

Juan M. Murillo

jgaralo@unex.es juanmamu@unex.es

Institute of Architecture of Application Systems

UNIVERSIDAD de EXTREMADURA