

Quantum Web Services: Practical Session

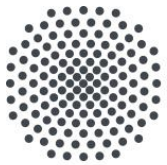
Jose Garcia-Alonso

Juan M. Murillo

jgaralo@unex.es

juanmamu@unex.es

Institute of Architecture of Application Systems



University of Stuttgart



Let's go to the practical part!!



SCAN ME

Material of the examples
and preparation of the
environment

<https://uex.be/24tutorial>

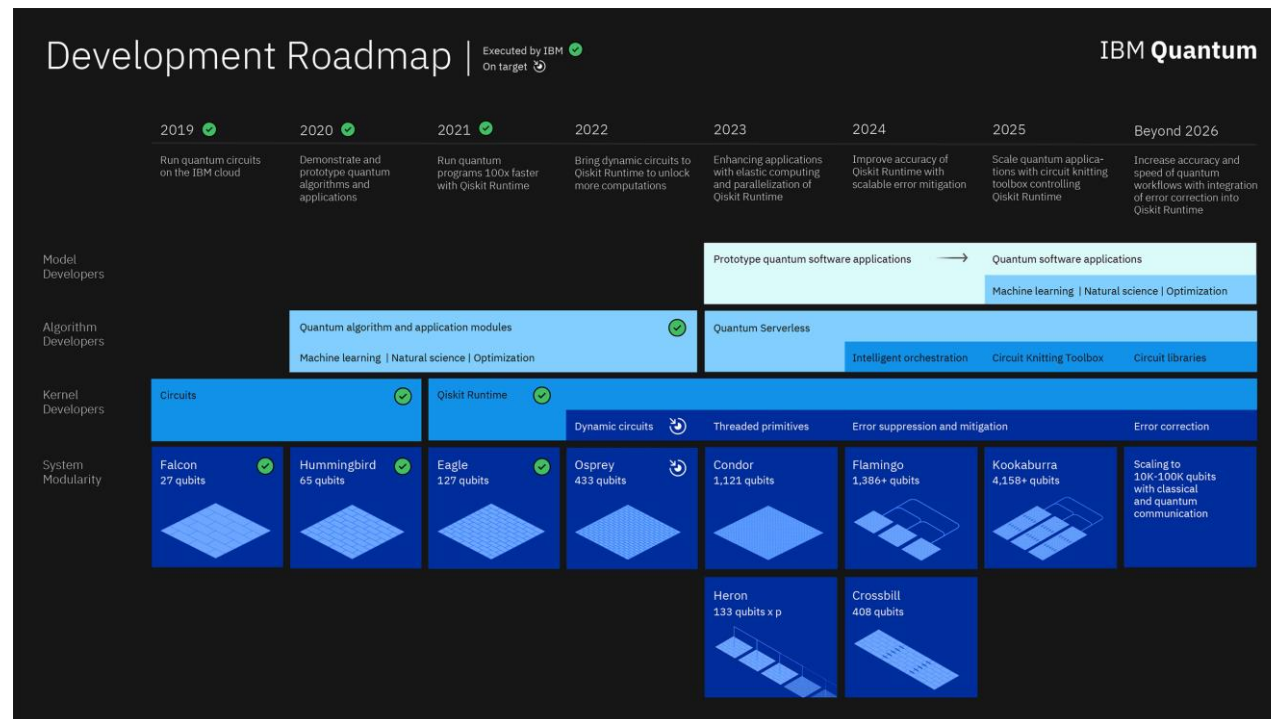
Introduction to quantum services with IBM Quantum

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**)



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 environment

IBM QTM



Quantum Services with IBM Quantum

Preparing the enviroment

1. Create an IBM Quantum account

<https://quantum.ibm.com/>

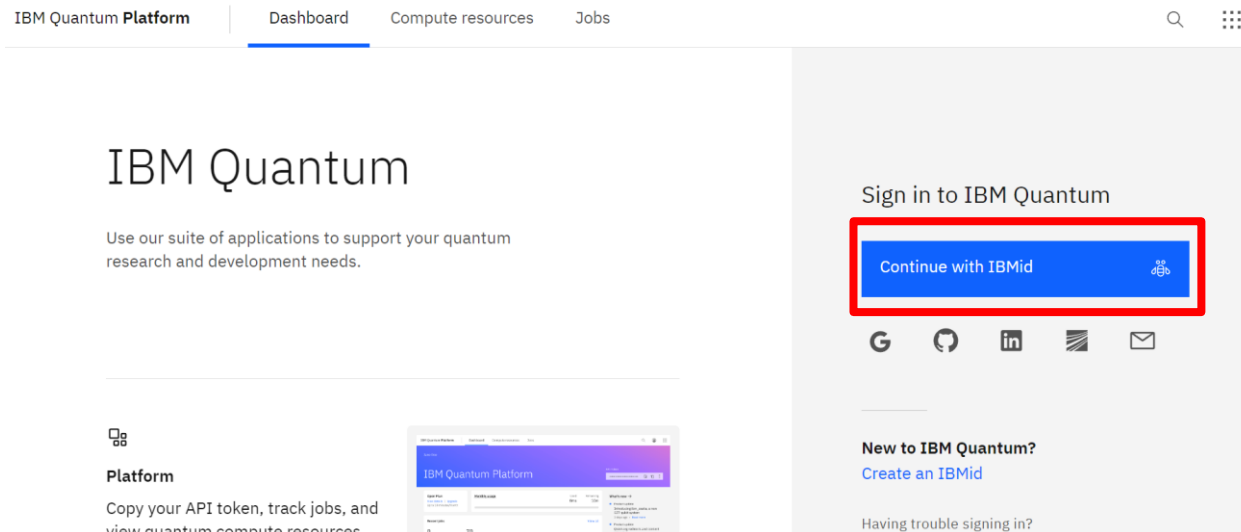


Only if you want to run the services in real quantum machines

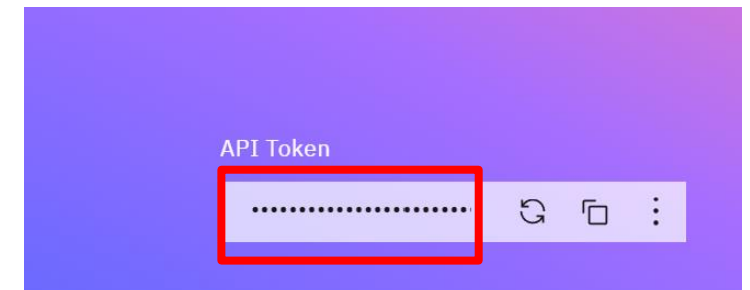
Quantum Services with IBM Quantum

Preparing the environment

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')"
```



<https://uex.be/24utorial>



Quantum Services with IBM Quantum



Quantum Services Examples

Shor's Algorithm reminder

```
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
```

```
# 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]
```

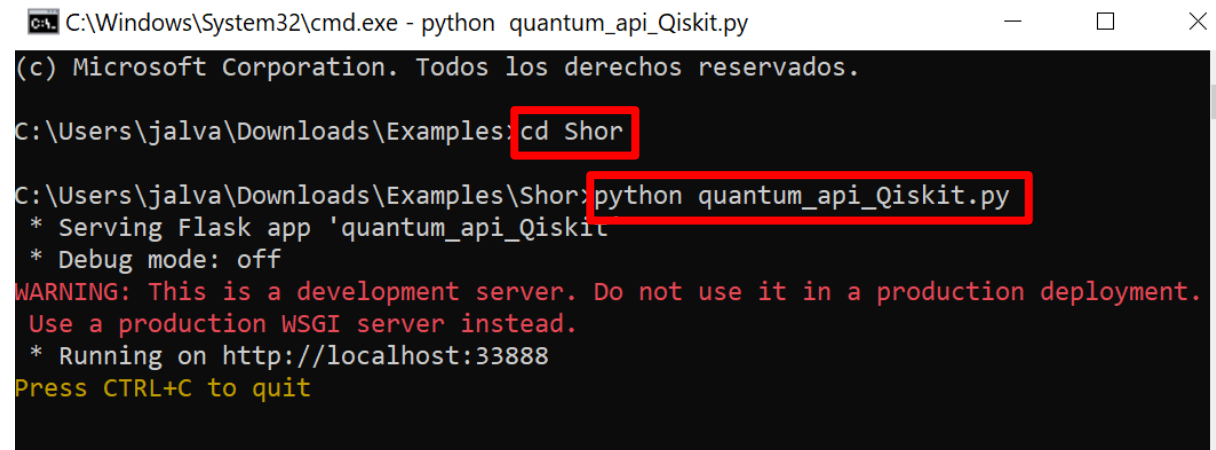
Quantum Services Examples

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`



```
C:\Windows\System32\cmd.exe - python quantum_api_Qiskit.py
(c) Microsoft Corporation. Todos los derechos reservados.
C:\Users\jalva\Downloads\Examples>cd Shor
C:\Users\jalva\Downloads\Examples\Shor>python quantum_api_Qiskit.py
* Serving Flask app 'quantum_api_Qiskit'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Running on http://localhost:33888
Press CTRL+C to quit
```

Quantum Services Examples

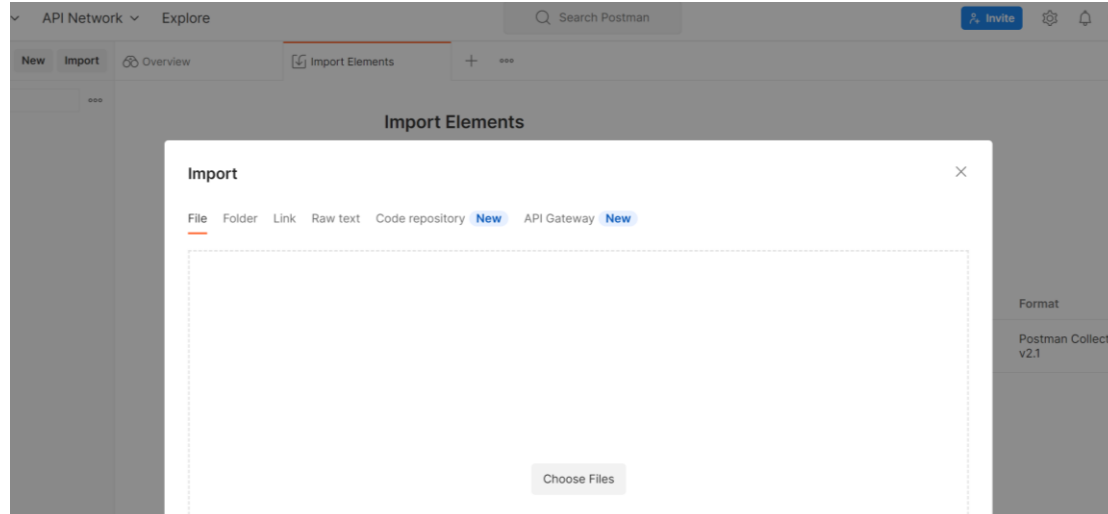
Shor's Algorithm reminder

API Deployment

2. Download Postman:

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

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



Import Elements

Search files and folders

Collections

<input checked="" type="checkbox"/> Collection Name	Format
<input checked="" type="checkbox"/> Shor service	Postman Collection v2.1

Import Cancel

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`



Shor service Copy / Shor local simulator

GET `http://localhost:33888/execute?device=local&shots=1000` Send

Params Authorization Headers (6) Body Pre-request Script Tests Settings

Query Params

<input checked="" type="checkbox"/>	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	device	local			
<input checked="" type="checkbox"/>	shots	1000			
<input type="checkbox"/>	Key	Value	Description		

body Cookies Headers (5) Tests

Pretty Raw Preview

```
1 {
2   "factors": [
3     3,
4     5
5   ]
}
```

Search jobs by ID, name or tag									
All statuses									
<input type="checkbox"/>	Job Id	Session Id	Status	Created	↓	Completed	Program	Compute resource	Usage
<input type="checkbox"/>	cqndmr3c8df00...		Queued	less than a minute ago			qasm3-runner	ibm_brisbane	Est. 20.2s



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:

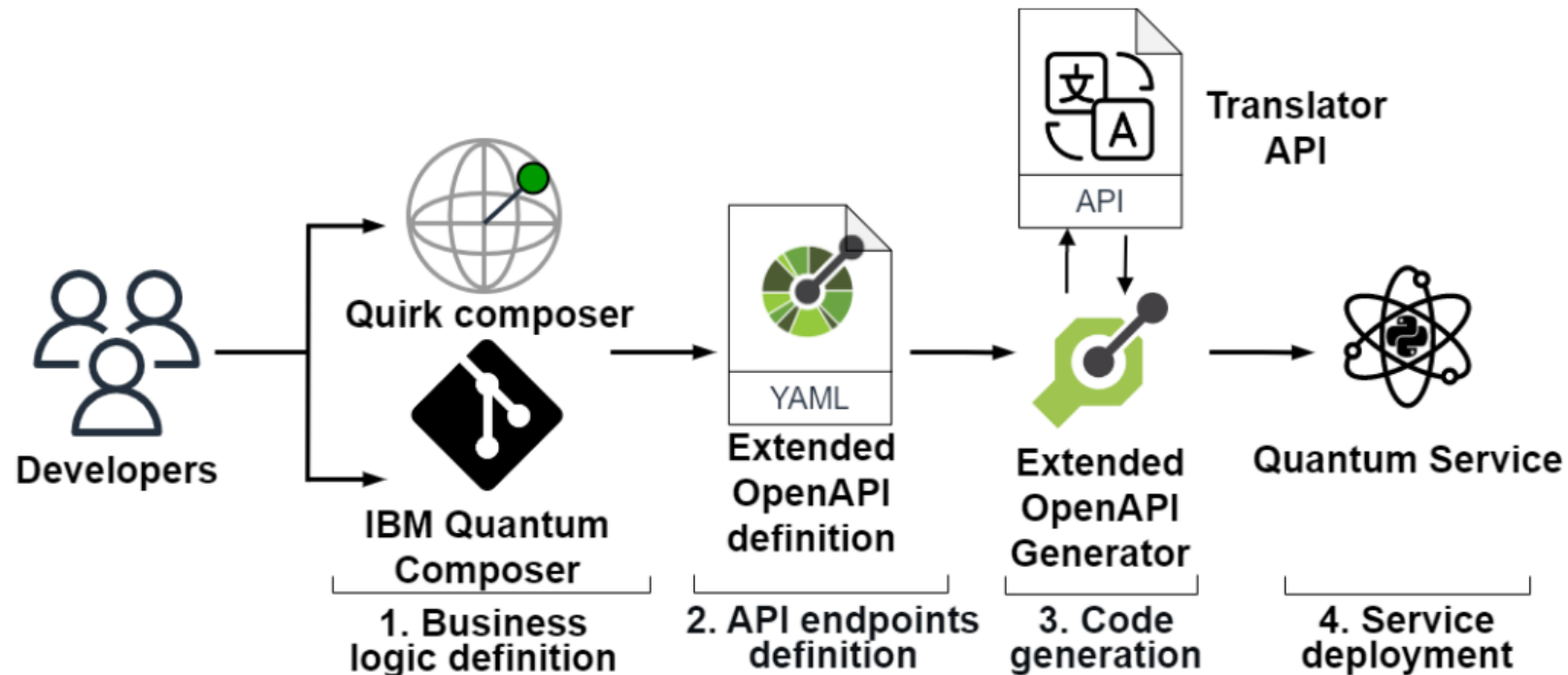
- **The business logic** of the service, which implements the functionality provided by the service.
- **The endpoint** of the service, which determines how an external client can invoke the service.



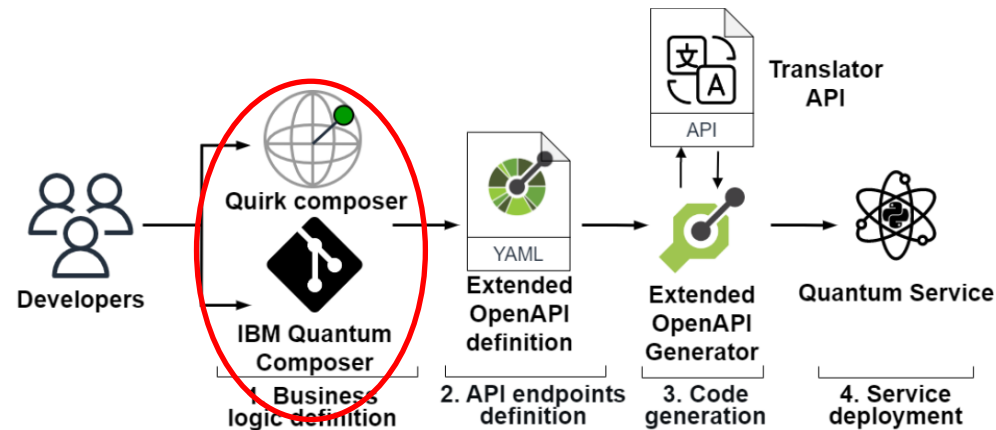
**GitHub –
OpenAPI
Generator
Quantum**

Quantum services generation

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



Quantum services generation

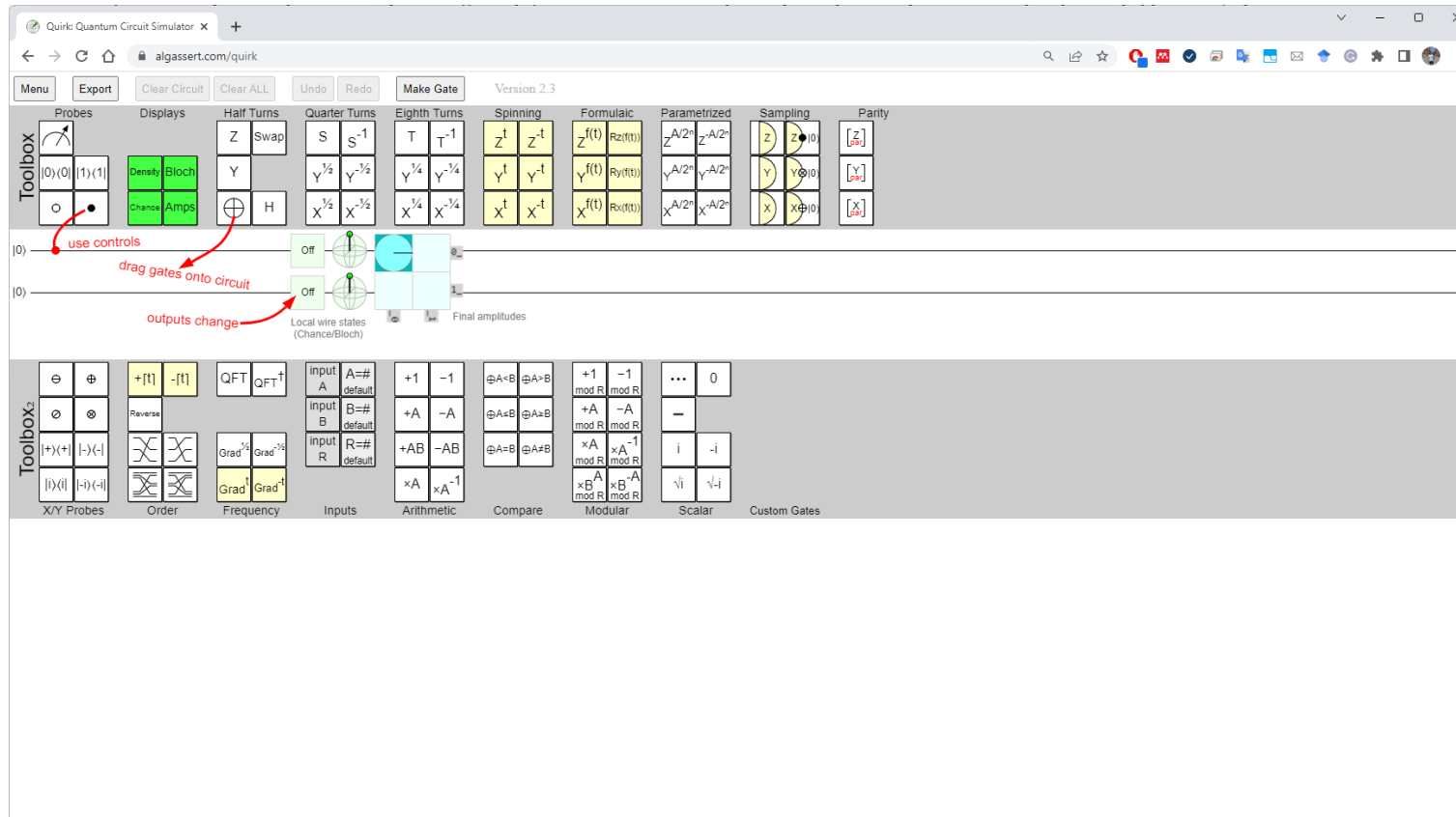


The first step is to define the business logic of the service as a quantum circuit using **Open Quirk**, indicating the Open Quirk link of the created circuit. Or directly indicating the link where is the **source code in Qiskit**

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

The screenshot shows the Quirk quantum circuit editor interface. The browser address bar is highlighted with a red box, displaying the URL: `algassert.com/quirk#circuit={"cols":["H","H","H","H"],["X"],["X","X","X","X"],[1,"•","X"],[1,"X","•"],[1,"•","X"],[1,1,"•","X"],[1,1,"X","•"],[1,1,"•","X"],...`

The interface includes a menu bar with buttons: Menu, Export, Clear Circuit, Clear ALL, Undo, Redo, Make Gate, and Version 2.3.

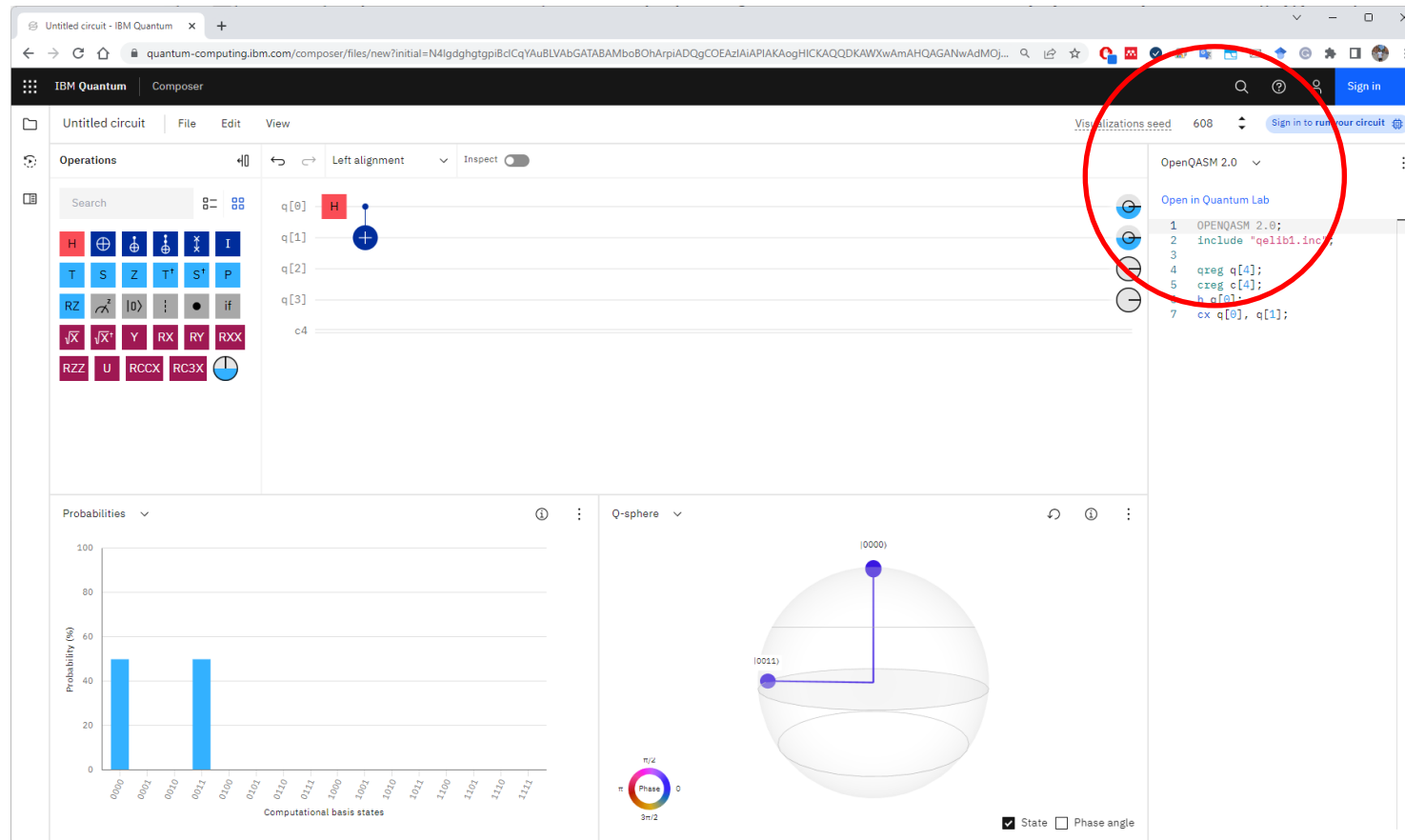
The toolbox is organized into several categories:

- Probes:** $|0\rangle\langle 0|$, $|1\rangle\langle 1|$, \circ , \bullet
- Displays:** Density, Bloch, Chance, Amps
- Half Turns:** Z, Swap, Y, \oplus , H
- Quarter Turns:** S, S^{-1} , $Y^{1/2}$, $Y^{-1/2}$, $X^{1/2}$, $X^{-1/2}$
- Eighth Turns:** T, T^{-1} , $Y^{1/4}$, $Y^{-1/4}$, $X^{1/4}$, $X^{-1/4}$
- Spinning:** Z^t , Z^{-t} , Y^t , Y^{-t} , X^t , X^{-t}
- Formulaic:** $Z^{f(t)}$, $R_z(f(t))$, $Y^{f(t)}$, $R_y(f(t))$, $X^{f(t)}$, $R_x(f(t))$
- Parametrized:** $Z^{A/2^n}$, $Z^{-A/2^n}$, $Y^{A/2^n}$, $Y^{-A/2^n}$, $X^{A/2^n}$, $X^{-A/2^n}$
- Sampling:** Z, $Z \bullet |0\rangle$, Y, $Y \otimes |0\rangle$, X, $X \oplus |0\rangle$
- Parity:** $[Z]_{\text{par}}$, $[Y]_{\text{par}}$, $[X]_{\text{par}}$

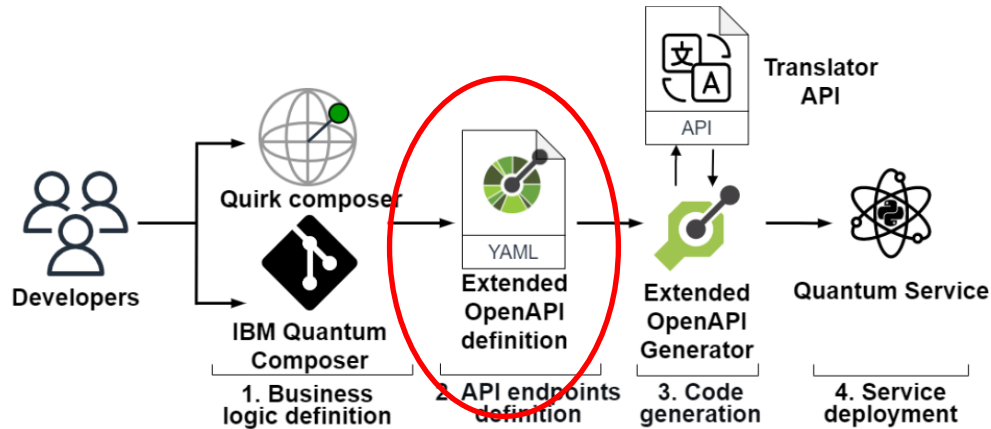
Quantum services generation

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

 Swagger Editor

Supported by SMARTBEAR

File Edit Insert Generate Server Generate Client

```
1 openapi: 3.0.2
2 info:
3   title: Quantum Services
4   description: This API enables the generation of quantum services.
5   version: '1.0'
6   termsOfService: 'https://quantum.spilab.es/terms'
7   contact:
8     name: Spilab
9     url: 'https://spilab.es'
10    email: info@spilab.es
11 tags:
12 - name: quantum
13   description: Everything about your Quantum Service
14 - name: quantum_code
15   description: Everything about your quantum service code and
    circuits
16 paths:
17   /circuit/grover:
18     get:
19       tags:
20         - quantum_code
21       summary: Get the circuit implementation of Grover Algorithm
22       description: ''
23       operationId: grover_circuit
24       responses:
25         '200':
26           description: successful operation
27         '405':
28           description: Invalid execution
29       x-quantumCode: 'https://algassert.com/quirk#circuit={
"cols":
[[["H","H"],["X","X"],[".", "Z"],["X","X"],["H","H"],["Z","Z"],
[[".", "Z"],["H","H"],["Measure","Measure"]]]}'
```

Quantum Services 1.0 OAS3

This API enables the generation of quantum services.

[Terms of service](#)

[Spilab - Website](#)

[Send email to Spilab](#)

[Find out more about Swagger](#)

quantum

Everything about your Quantum Service

GET

/findByCategory

Finds quantum service by category

quantum_code

Everything about your quantum service code and circuits

GET

/circuit/grover

Get the circuit implementation of Grover Algorithm

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.

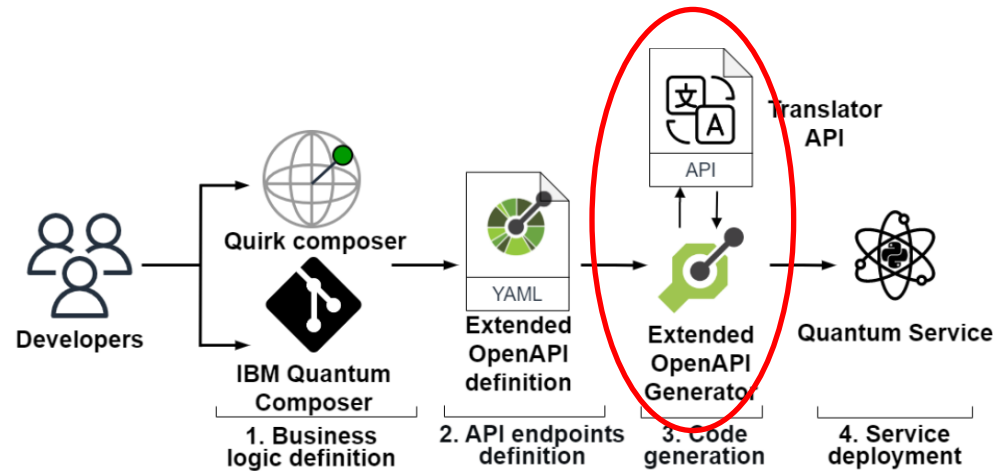
```
36      description: Number of shots
37      required: true
38      style: form
39      explode: false
40      deprecated: true
41      schema:
42        type: number
43      responses:
44        '200':
45          description: successful operation
46        '405':
47          description: Invalid 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'
```

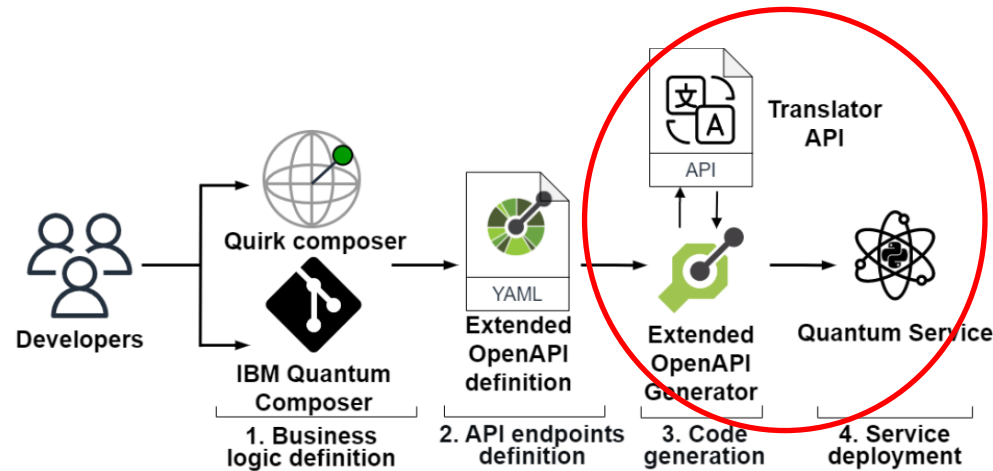
Quantum services generation



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_quantum.yaml",
}
```

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

Qiskit libraries for quantum computing

```
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
```

Classical wrapping
endpoint service

```
: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

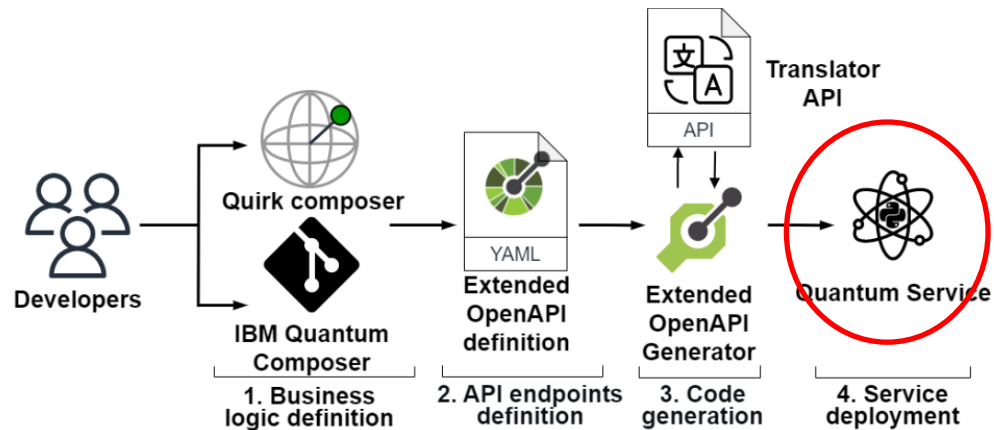
backend = provider.get_backend('ibmq_qasm_simulator')
job = execute(circuit, backend, shots=1)

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

Random Numbers
quantum circuit

```
return counts
```

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.

Swagger **Quantum Services** 1.0 OAS3

/openapi.json

This API enables the generation of quantum services.

[Terms of service](#)
[Spiralab - Website](#)
[Send email to Spiralab](#)
[Find out more about Swagger](#)

Servers

quantum Everything about your Quantum Service

GET /findByCategory Finds quantum service by category

quantum_code Everything about your quantum service code and circuits

GET /circuit/grover Get the circuit Implementation of Grover Algorithm

SCHEME :// HOST [":" PORT] [PATH ["?" QUERY]]

GET http://qapps.unex.es:8081/circuit/grover length: 40 byte(s)

QUERY PARAMETERS

HEADERS ②

+ Add header Add authorization

BODY ②

XHR does not allow payloads for GET request.

Response Cache Detected - Elapsed Time: 75ms

200 OK

HEADERS ②

pretty

Server: Werkzeug/2.1.2 Python/3.7.6
Date: Thu, 30 Jun 2022 10:42:47 GMT +3m

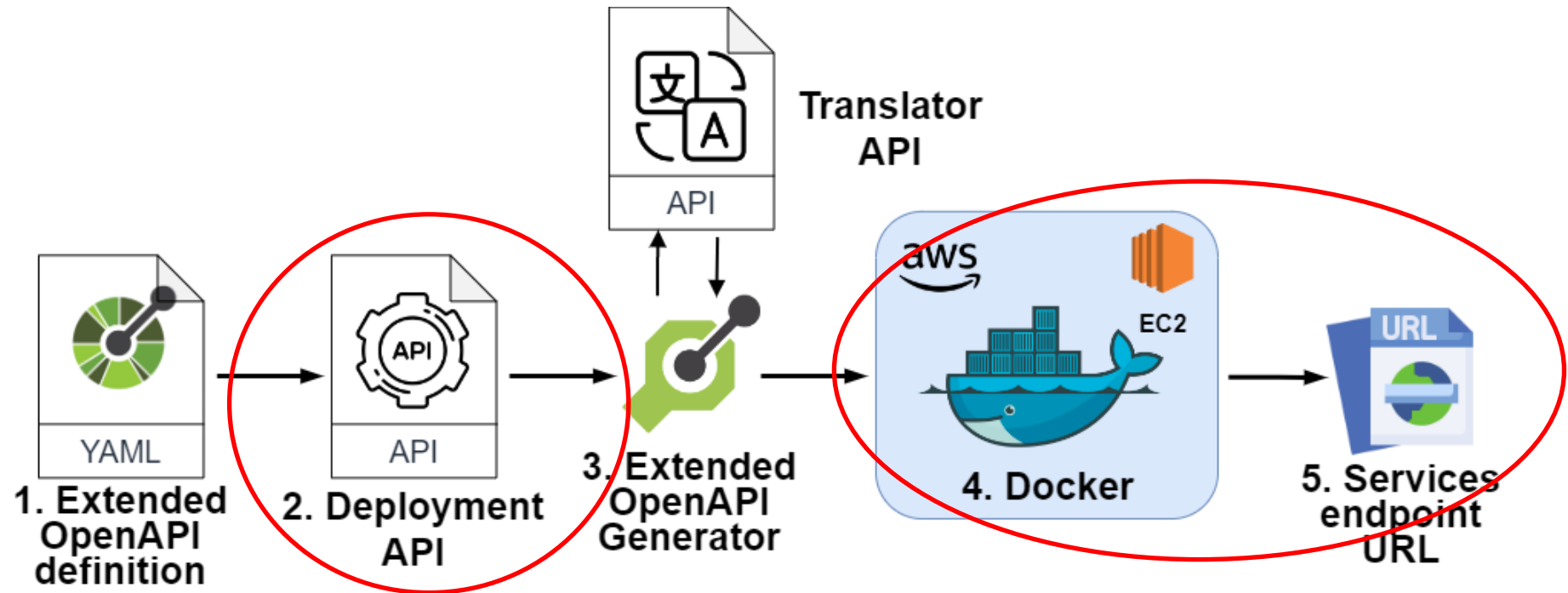
BODY ②

pretty

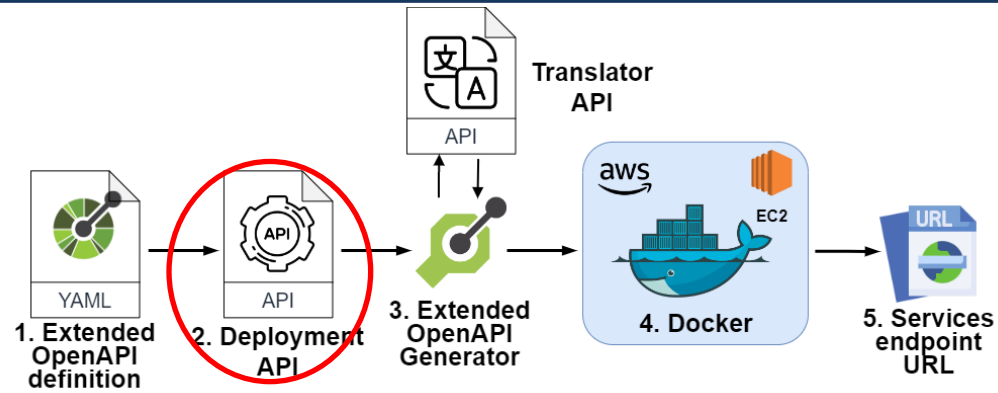
00: 100

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

 **dockerhub**


Explore Repositories Organizations Help ▾

Upgrade  jromero236 ▾


jromero236 > Repositories > quantumservices > General



Using 0 of 1 private repositories. [Get more](#)

General Tags Builds Collaborators Webhooks Settings

 Add a short description for this repository
The short description is used to index your content on Docker Hub and in search engines. It's visible to users in search results.

Update



 **jromero236 / quantumservices**

Description
This repository does not have a description 
 Last pushed: 5 months ago


Docker commands [Public View](#)
To push a new tag to this repository,

```
docker push jromero236/quantumservices:tagname
```

Tags
This repository contains 1 tag(s).

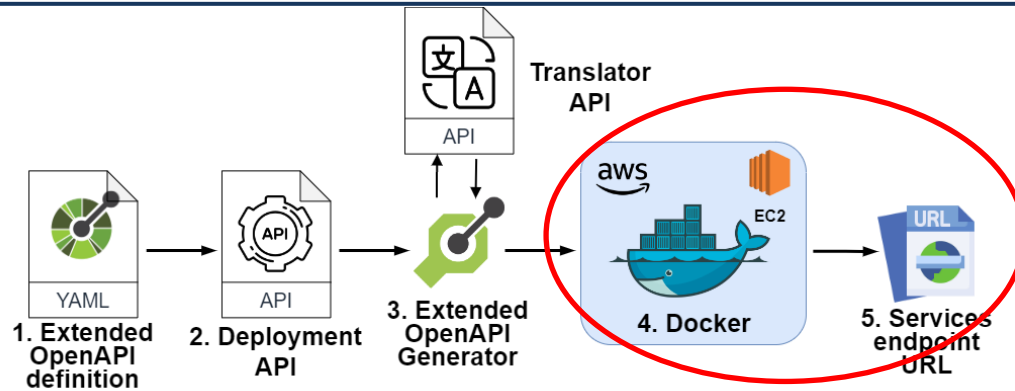
Tag	OS	Type	Pulled	Pushed
 latest		Image	3 months ago	5 months ago

[See all](#) [Go to Advanced Image Management](#)

Automated Builds
Manually pushing images to Hub? Connect your account to GitHub or Bitbucket to automatically build and tag new images whenever your code is updated, so you can focus your time on creating.
Available with Pro, Team and Business subscriptions. [Read more about automated builds](#) 
[Upgrade](#)



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

⌵

📄

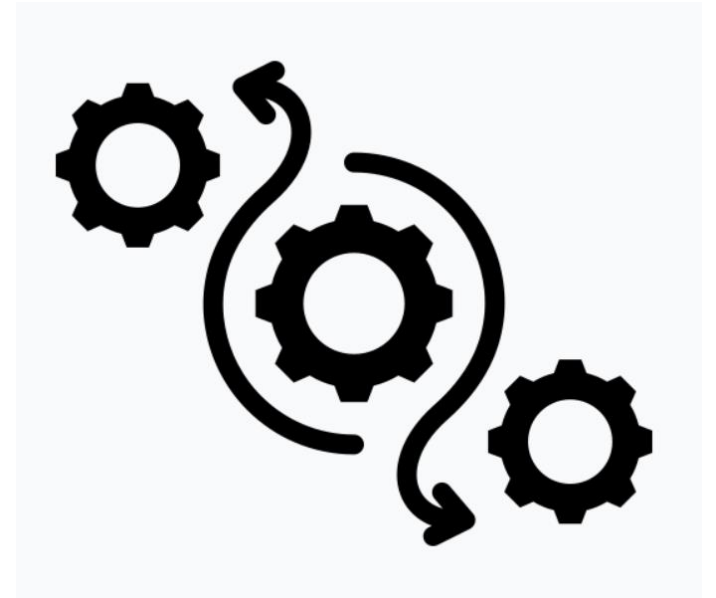
🔍

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



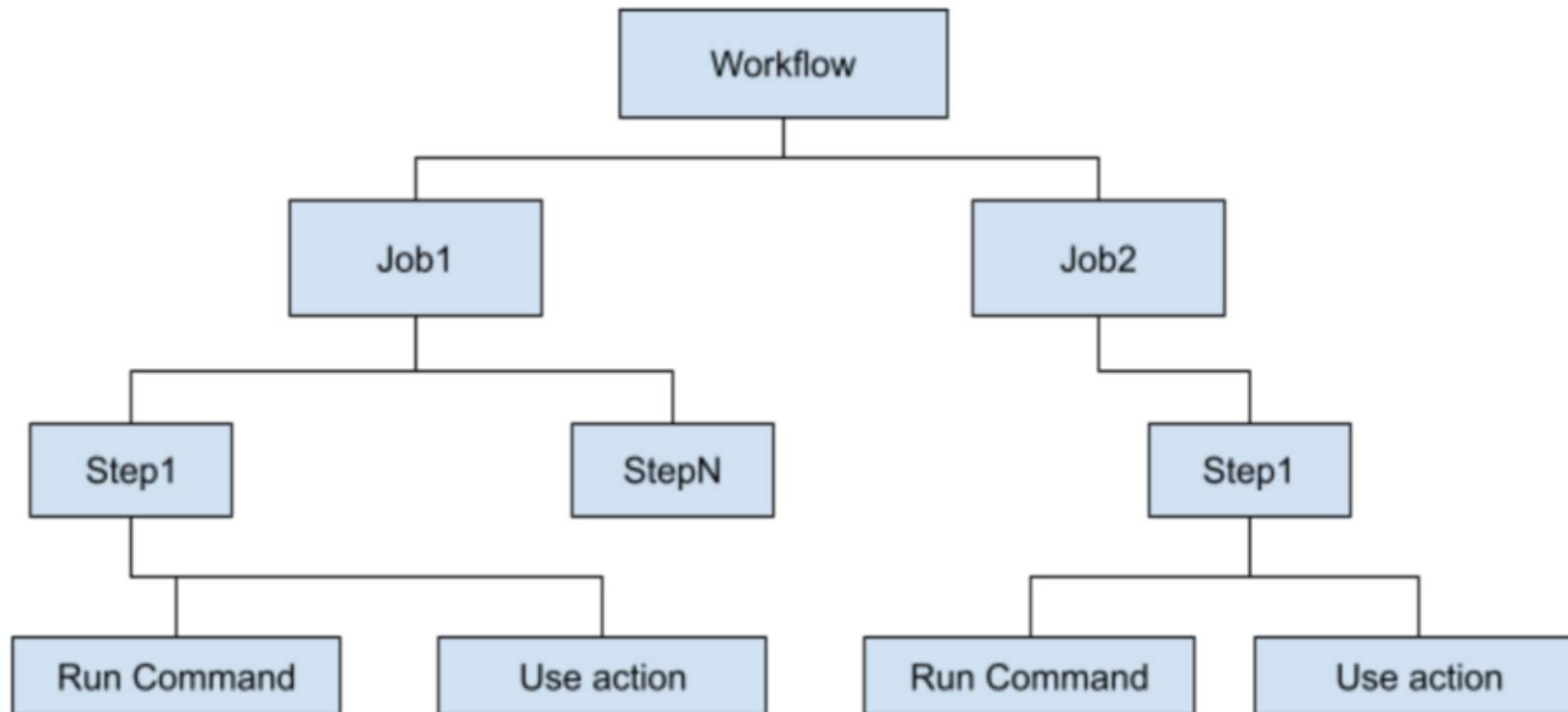
GitHub Actions

- 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.

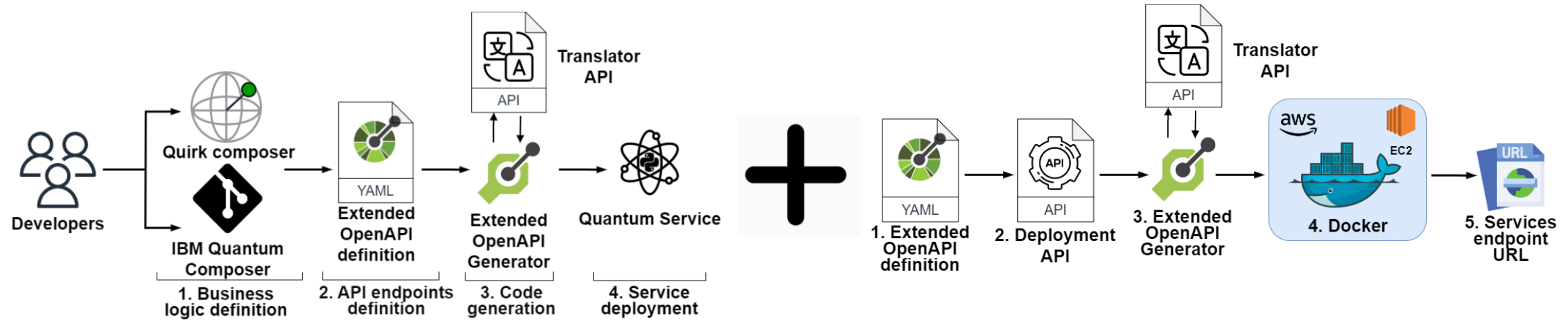
Workflow for the continuous deployment



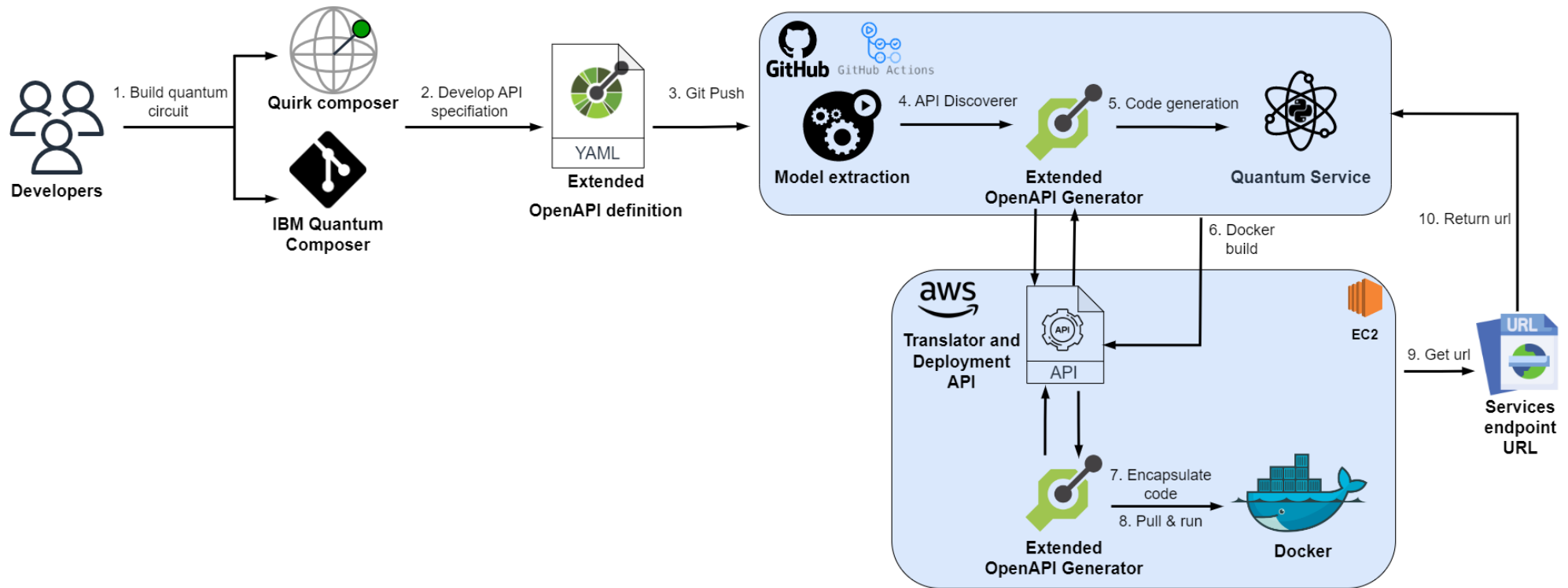
GitHub Actions



Workflow for the continuous deployment



Workflow for the continuous deployment



Workflow for the continuous deployment

The screenshot shows a GitHub Actions workflow run for the repository 'javierrome236/quantumDeployment'. The workflow is named 'Automate deployment' and the specific run is 'Update openapi.yaml #94'. The workflow is currently in a 'Completed' state, indicated by a green checkmark. The left sidebar shows the 'Summary' tab selected, with links to 'Jobs', 'Run details', 'Usage', and 'Workflow file'. The 'Jobs' section shows a single job named 'validate (3.8)' which is also completed. The main area displays the job's steps and logs. The steps are: 'Set up job', 'Build cpina/github-action-push-to-another-repository@main', 'Run actions/checkout@v3', 'Build python API with YAML', 'Set up Python 3.8', 'Install dependencies', 'Build with Maven', 'Prepare Sonar tests', 'Pushes to another repository', 'Check python project', 'Deploy Stage', 'Post Set up Python 3.8', 'Post Build python API with YAML', 'Post Run actions/checkout@v3', and 'Complete job'. The 'Deploy Stage' step is expanded, showing a terminal log with a curl command and its output. The log shows a successful POST request to the 'quantumservicesdeployment.spilab.es' API, returning a 200 status code and a JSON response.

Update openapi.yaml - javierrome236

github.com/javierrome236/quantumDeployment/actions/runs/5066527124/jobs/9097164571

← Automate deployment

Update openapi.yaml #94

Re-run all jobs Latest #3

Summary

Jobs

validate (3.8)

Run details

Usage

Workflow file

validate (3.8)
succeeded last week in 1m 26s

Search logs

- Set up job 3s
- Build cpina/github-action-push-to-another-repository@main 6s
- Run actions/checkout@v3 3s
- Build python API with YAML 0s
- Set up Python 3.8 0s
- Install dependencies 55s
- Build with Maven 6s
- Prepare Sonar tests 1s
- Pushes to another repository 3s
- Check python project 0s
- Deploy Stage 4s
- Post Set up Python 3.8 0s
- Post Build python API with YAML 0s
- Post Run actions/checkout@v3 0s
- Complete job 0s

```
1 ▶ Run curl -X POST -F 'name=python' -F 'yaml=https://raw.githubusercontent.com/javierrome236/quantumDeployment/main/openapi.yaml' http://quantumservicesdeployment.spilab.es:8081/docker
13 % Total % Received % Xferd Average Speed Time Time Time Current
14 Dload Upload Total Spent Left Speed
15
16 0 0 0 0 0 0 0 0 --:--:-- --:--:-- --:--:-- 0
17 100 321 0 0 100 321 0 263 0:00:01 0:00:01 --:--:-- 263
18 100 321 0 0 100 321 0 144 0:00:02 0:00:02 --:--:-- 144
19 100 321 0 0 100 321 0 99 0:00:03 0:00:03 --:--:-- 99
20 100 321 0 0 100 321 0 76 0:00:04 0:00:04 --:--:-- 76
21 100 451 100 130 100 321 26 64 0:00:05 0:00:04 0:00:01 90
22 100 451 100 130 100 321 26 64 0:00:05 0:00:04 0:00:01 34
23 <html><h1 href="http://quantumservicesdeployment.spilab.es:8085/ui">http://quantumservicesdeployment.spilab.es:8085/ui</h1></html>
```

Quantum Web Services: Practical Session

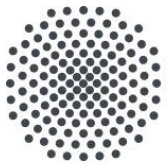
Jose Garcia-Alonso

Juan M. Murillo

jgaralo@unex.es

juanmamu@unex.es

Institute of Architecture of Application Systems



University of Stuttgart

