Quantum Web Services

Jose Garcia-Alonso

Juan M. Murillo

jgaralo@unex.es *juanmamu@unex.es*

Institute of Architecture of Application Systems

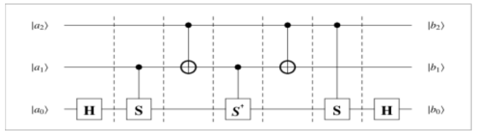
University of Stuttgart

UNIVERSIDAD de EXTREMADURA

Today’s Software Systems…

Connected
Software
Pieces

Supported by…
Service Oriented
Mobile
Cloud
Fog



}

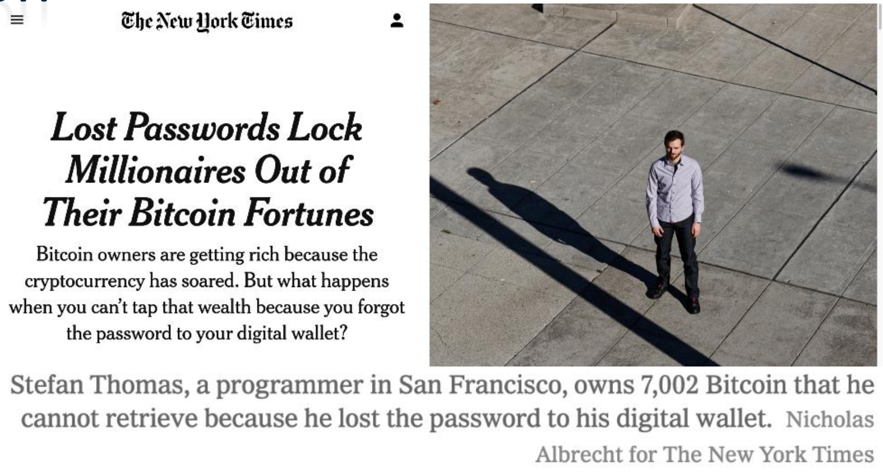
Globaly
Distributed

Future Hybrid Software Systems

Even when we do not know how the future quantum computers will look like, we already know some of the features of the systems that will use them.

1. They will have to co-exist with classical systems
2. Co-existence and interaction, will be supported by service composition
3. The development of quantum services will be governed by the current general criteria of Service Engineering (composability, reusability, maintainability, etc.)

Look at a good classic service implementation

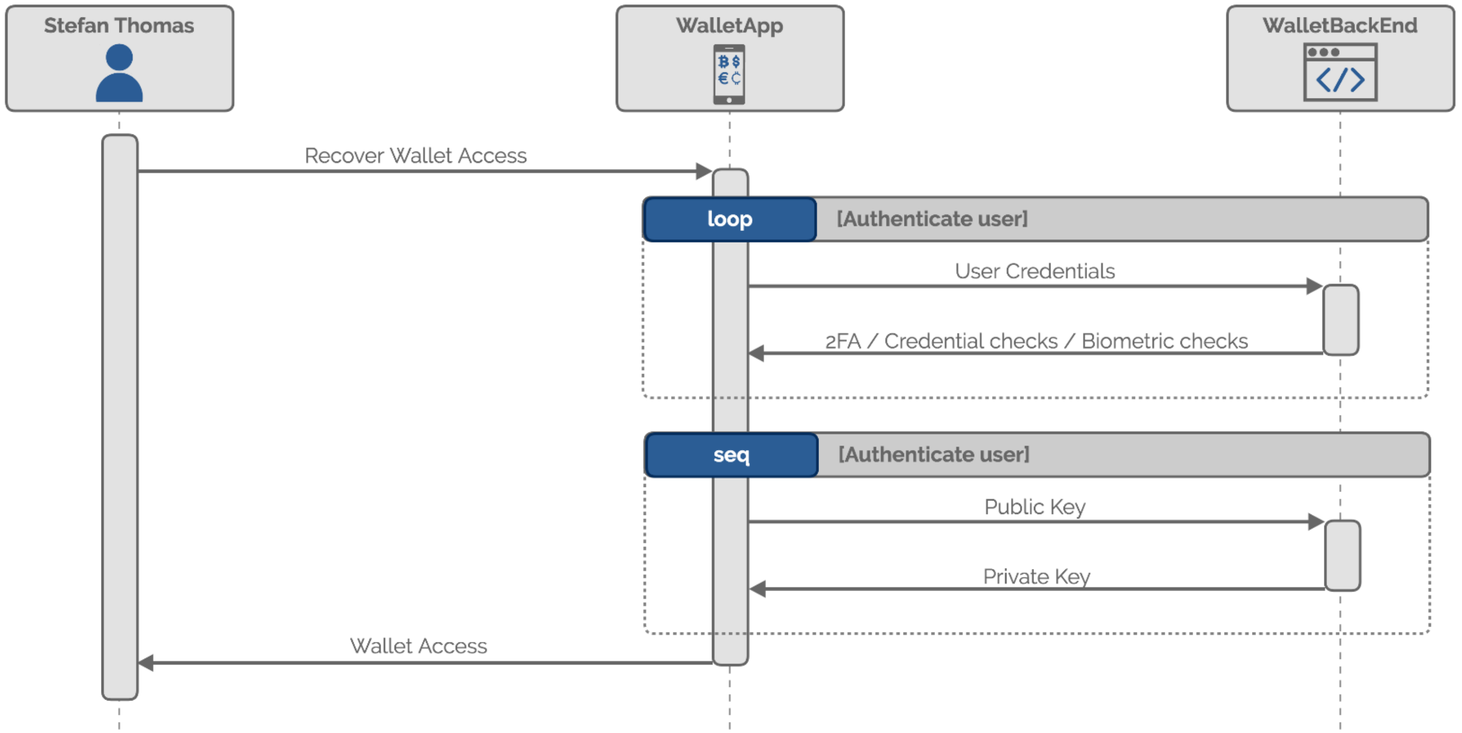


The New York Times

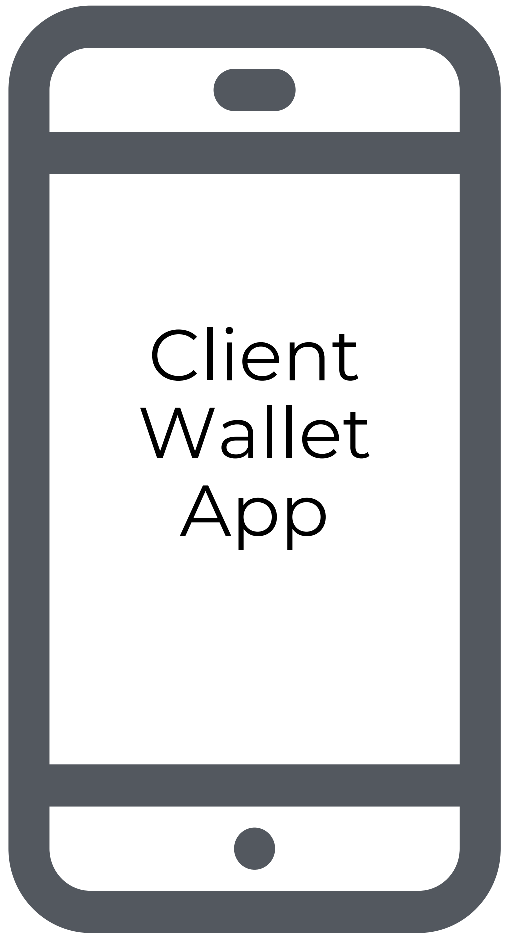
Lost Passwords Lock Millionaires Out of Their Bitcoin Fortunes

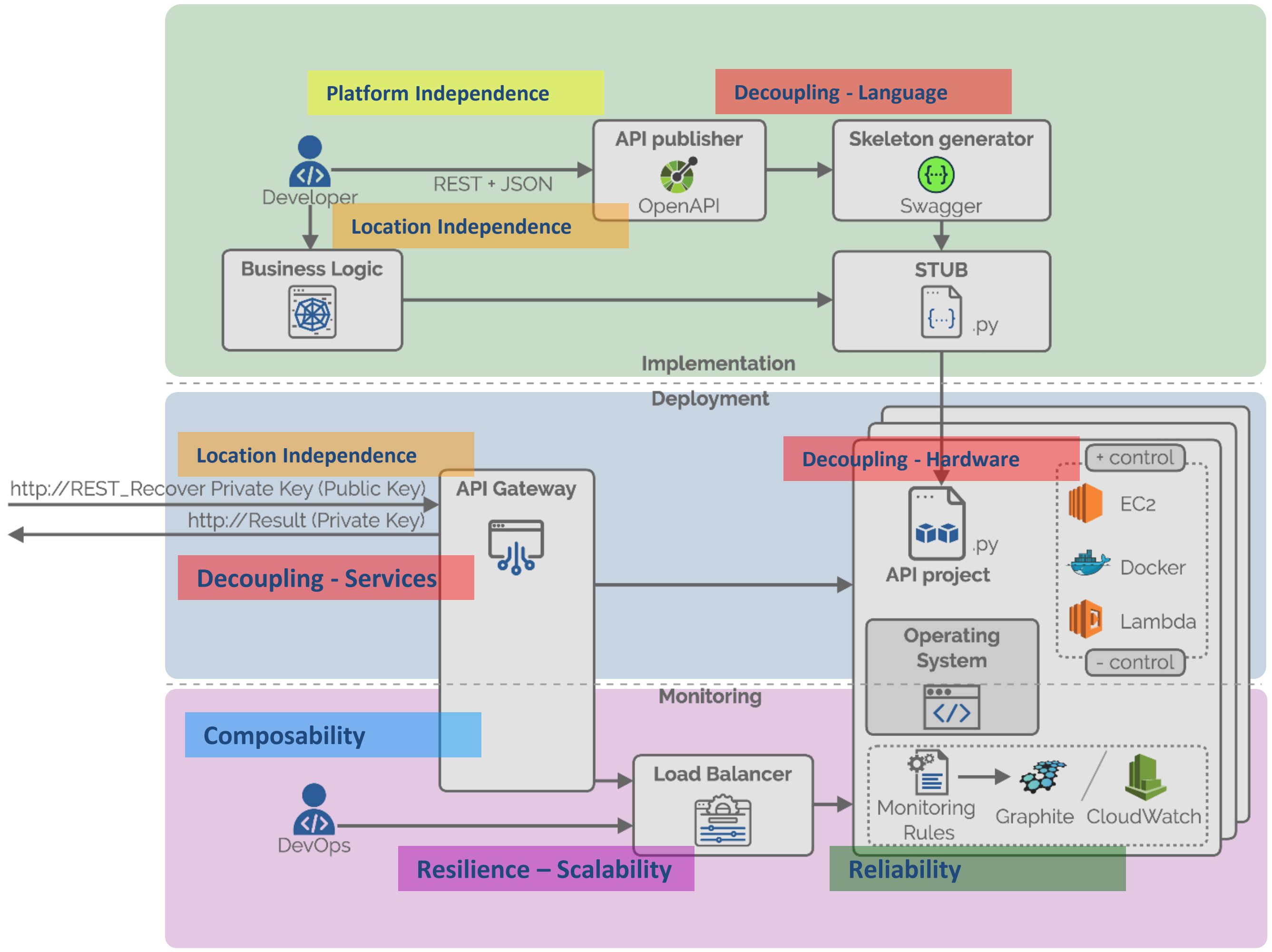
Bitcoin owners are getting rich because the cryptocurrency has soared. But what happens when you can't tap that wealth because you forgot the password to your digital wallet?

Stefan Thomas, a programmer in San Francisco, owns 7,002 Bitcoin that he cannot retrieve because he lost the password to his digital wallet. Nicholas Albrecht for The New York Times



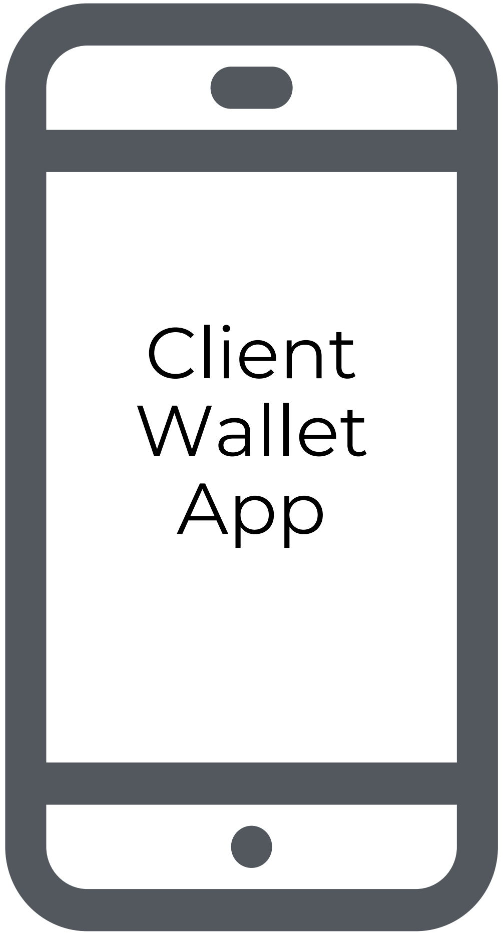
A good classic service implementation

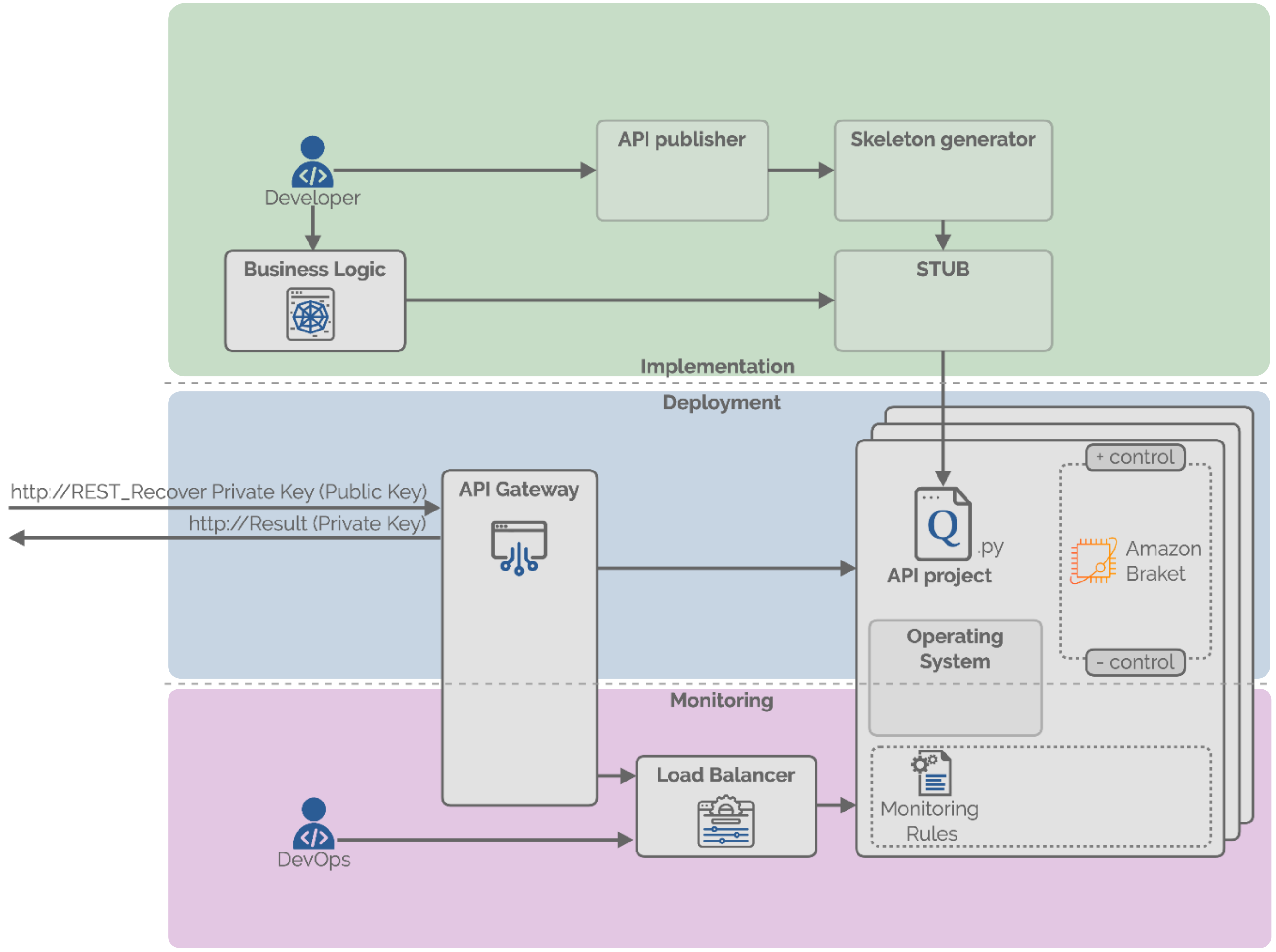




A good quantum service implementation

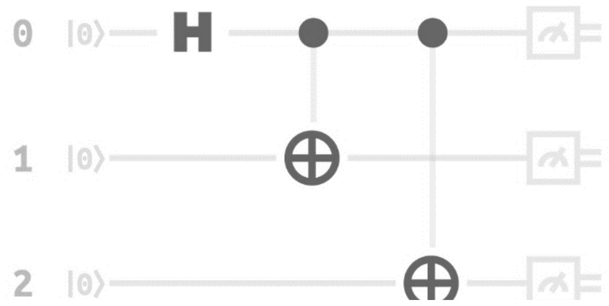
There is no support for real services so X-Abilities are lost



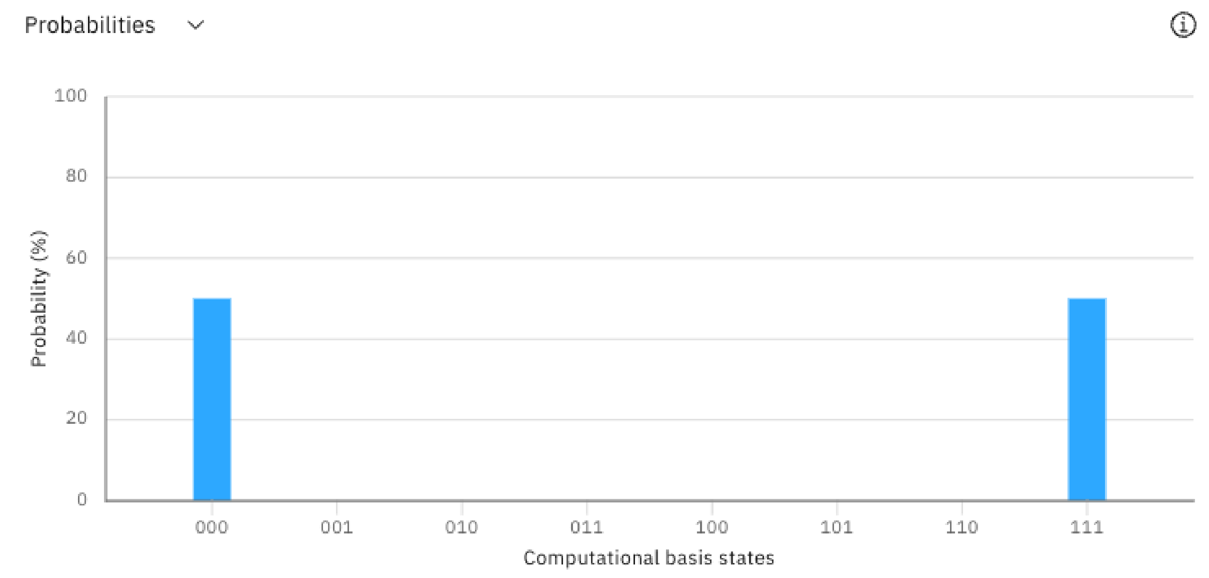


Current posibilities for quantum services

Current posibilities for quantum services



We have a **Quantum Code** providing a functionality and we would like to integrate it in our **app**



CLIENT APP

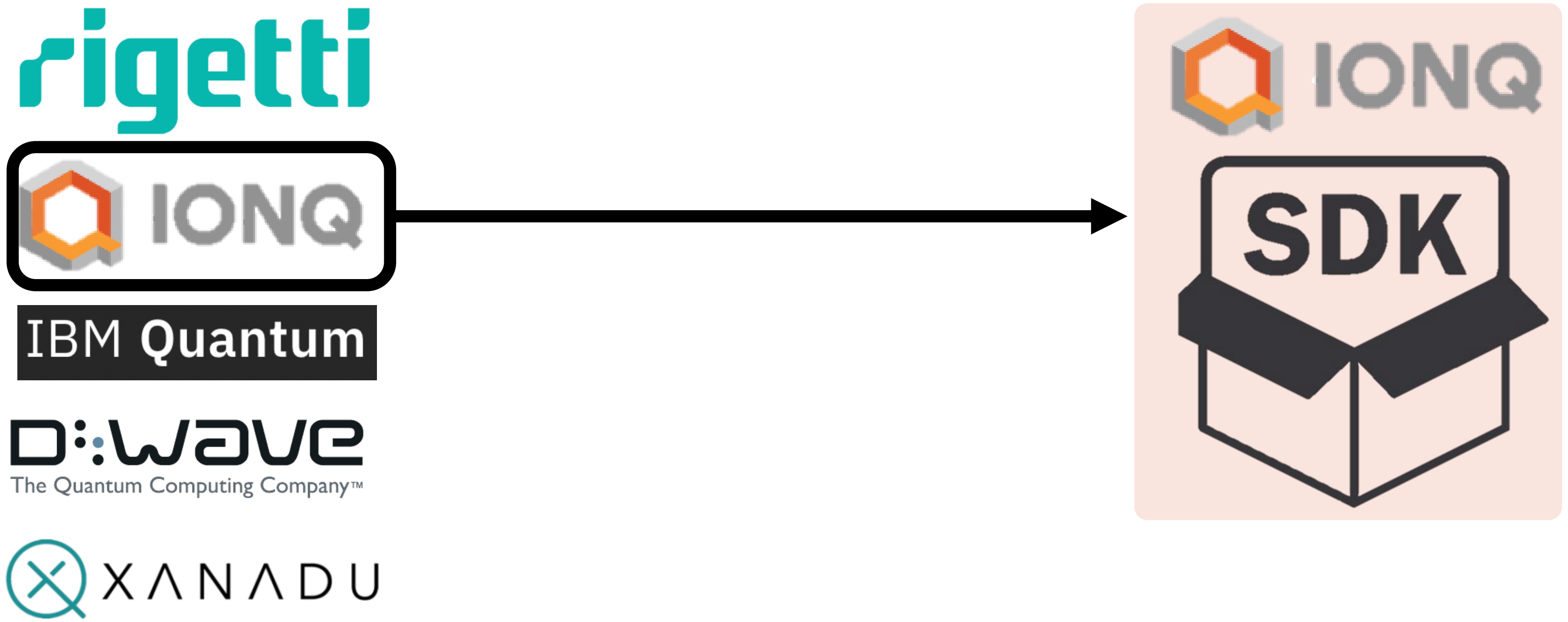
{Quantum

Service Call}

Current posibilities for quantum services

1. Use a hardware provider

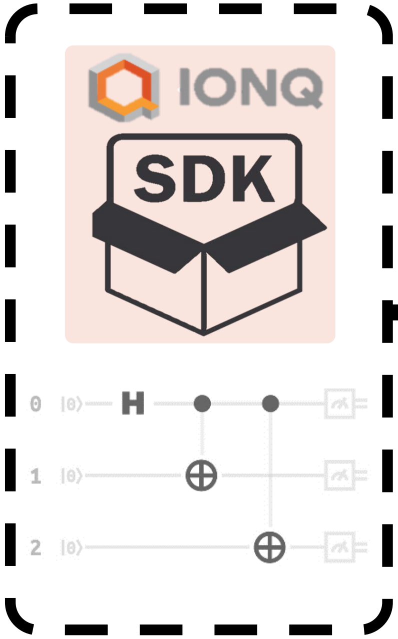
Step 1: choose the provider



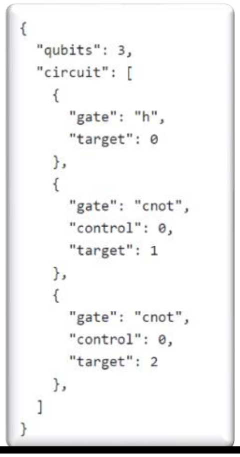
Current posibilities for quantum services

1. Use a hardware provider

Step 2: Use the provider's SDK to develop the quantum program and to integrate it in our app



Circuit Language

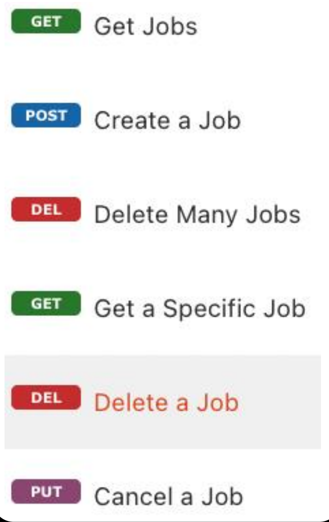


```
{
  "qubits": 3,
  "circuit": [
    {
      "gate": "h",
      "target": 0
    },
    {
      "gate": "cnot",
      "control": 0,
      "target": 1
    },
    {
      "gate": "cnot",
      "control": 0,
      "target": 2
    },
  ]
}
```

Job Language

```
{
  "lang": "json",
  "shots": 1024,
  "name": "My Quantum Alg",
  "target": "qpu.harmony",
  "body": {
    "qubits": 3,
    "circuit": [
      
    ]
  }
}
```

API



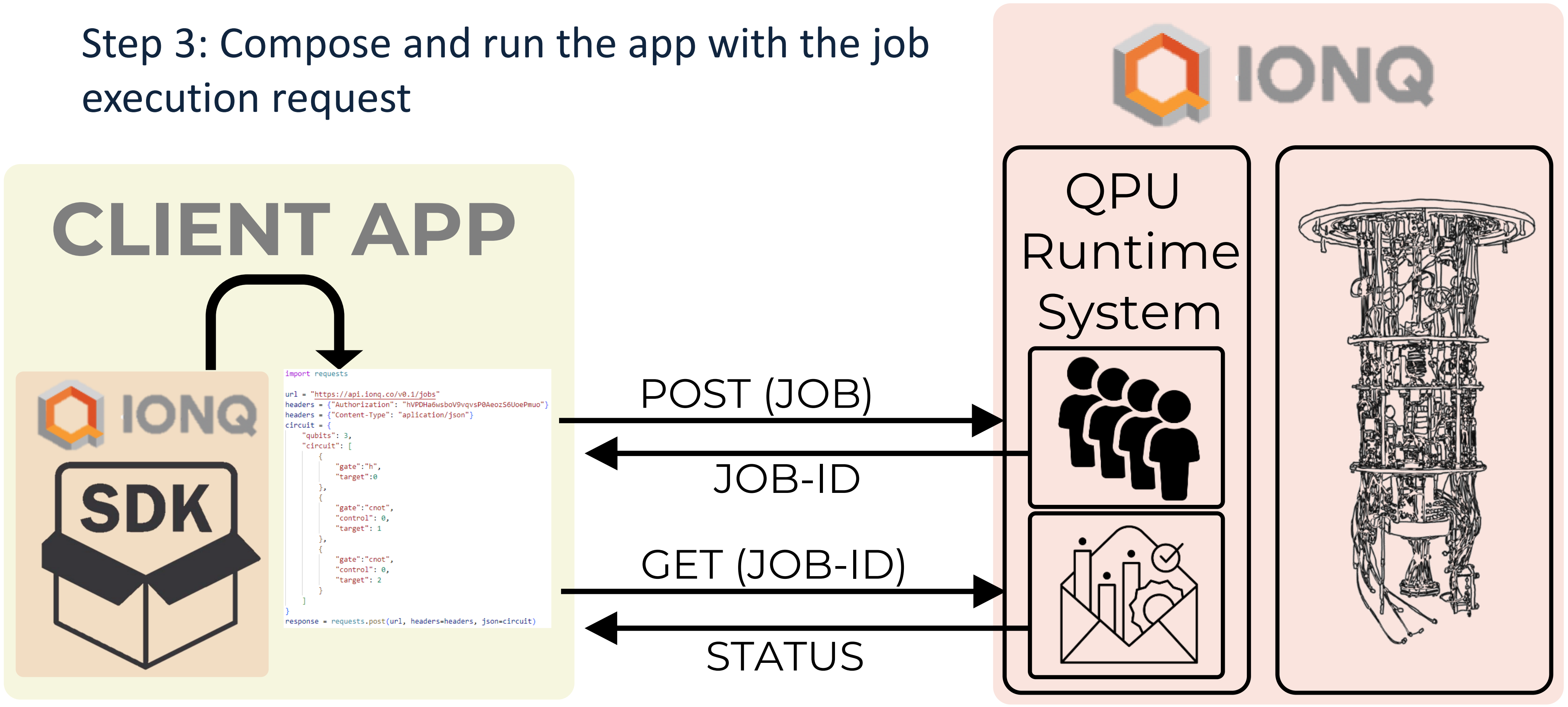
GET Get Jobs
POST Create a Job
DEL Delete Many Jobs
GET Get a Specific Job
DEL Delete a Job
PUT Cancel a Job

Vendor specific API Key

Current posibilities for quantum services

1. Use a hardware provider

Step 3: Compose and run the app with the job execution request

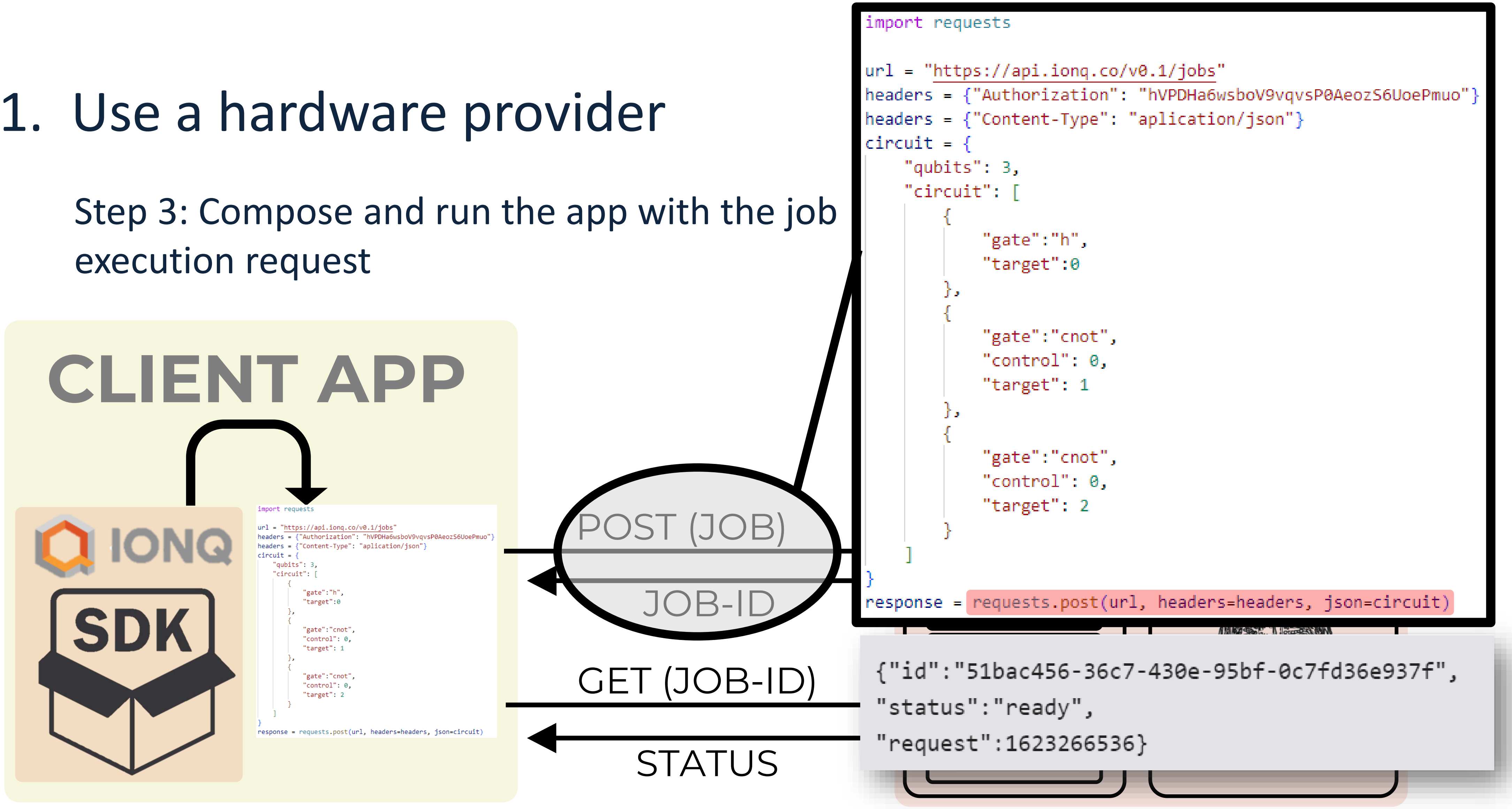


```
import requests

url = "https://api.ionq.co/v0.1/jobs"
headers = {"Authorization": "hVPDHa6wsboV9vqvsP0AeozS6UoePmuo"}
headers = {"Content-Type": "aplication/json"}
circuit = {
    "qubits": 3,
    "circuit": [
        {
            "gate":"h",
            "target":0
        },
        {
            "gate":"cnot",
            "control": 0,
            "target": 1
        },
        {
            "gate":"cnot",
            "control": 0,
            "target": 2
        }
    ]
}
response = requests.post(url, headers=headers, json=circuit)
```

Current posibilities for quantum services

1. Use a hardware provider

Step 3: Compose and run the app with the job execution request



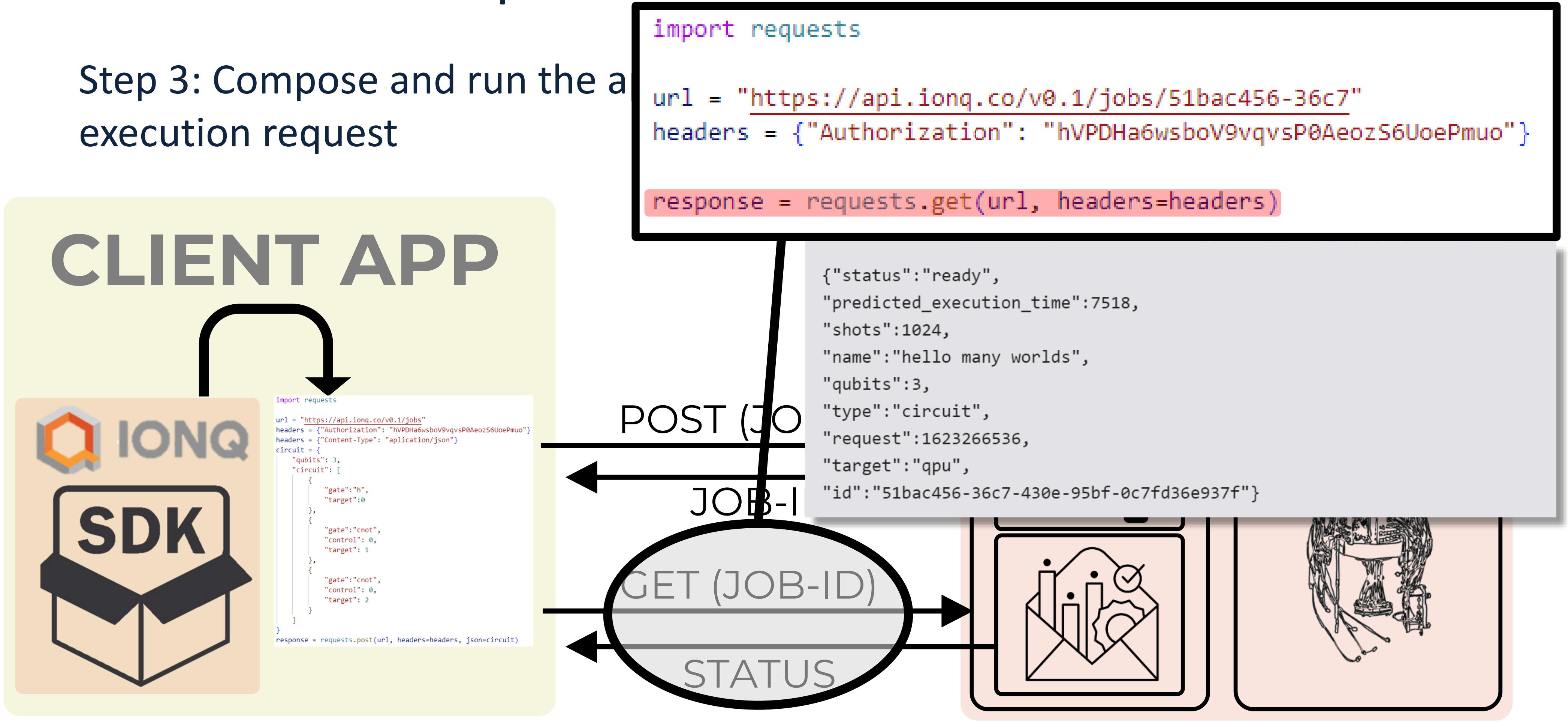
```python
import requests

url = "https://api.ionq.co/v0.1/jobs"
headers = {"Authorization": "hVPDHa6wsboV9vqvsP0AeozS6UoePmuo"}
headers = {"Content-Type": "aplication/json"}
circuit = {
    "qubits": 3,
    "circuit": [
        {
            "gate":"h",
            "target":0
        },
        {
            "gate":"cnot",
            "control": 0,
            "target": 1
        },
        {
            "gate":"cnot",
            "control": 0,
            "target": 2
        }
    ]
}
response = requests.post(url, headers=headers, json=circuit)
```

```
{"id":"51bac456-36c7-430e-95bf-0c7fd36e937f",
"status":"ready",
"request":1623266536}
```

Current posibilities for quantum services

1. Use a hardware provider

Step 3: Compose and run the a execution request



```python
import requests

url = "https://api.ionq.co/v0.1/jobs"
headers = {"Authorization": "hVPDHa6wsboV9vqvsP0AeozS6UoePmuo"}
headers = {"Content-Type": "aplication/json"}
circuit = {
    "qubits": 3,
    "circuit": [
        {
            "gate":"h",
            "target":0
        },
        {
            "gate":"cnot",
            "control": 0,
            "target": 1
        },
        {
            "gate":"cnot",
            "control": 0,
            "target": 2
        }
    ]
}
response = requests.post(url, headers=headers, json=circuit)
```

```python
import requests

url = "https://api.ionq.co/v0.1/jobs/51bac456-36c7"
headers = {"Authorization": "hVPDHa6wsboV9vqvsP0AeozS6UoePmuo"}

response = requests.get(url, headers=headers)
```

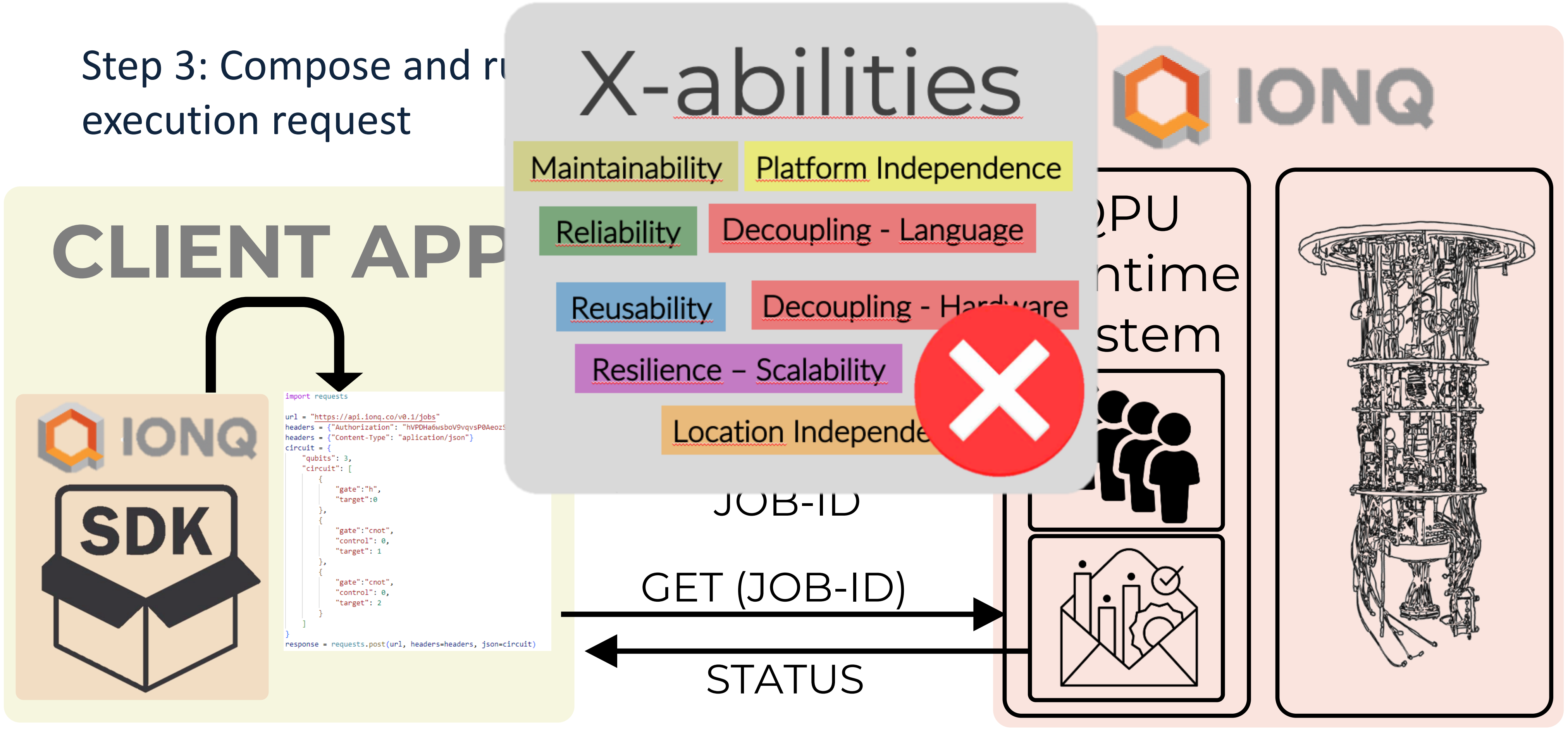
```
{"status":"ready",
"predicted_execution_time":7518,
"shots":1024,
"name":"hello many worlds",
"qubits":3,
"type":"circuit",
"request":1623266536,
"target":"qpu",
"id":"51bac456-36c7-430e-95bf-0c7fd36e937f"}
```

Current posibilities for quantum services

1. Use a hardware provider

Step 3: Compose and r

execution request

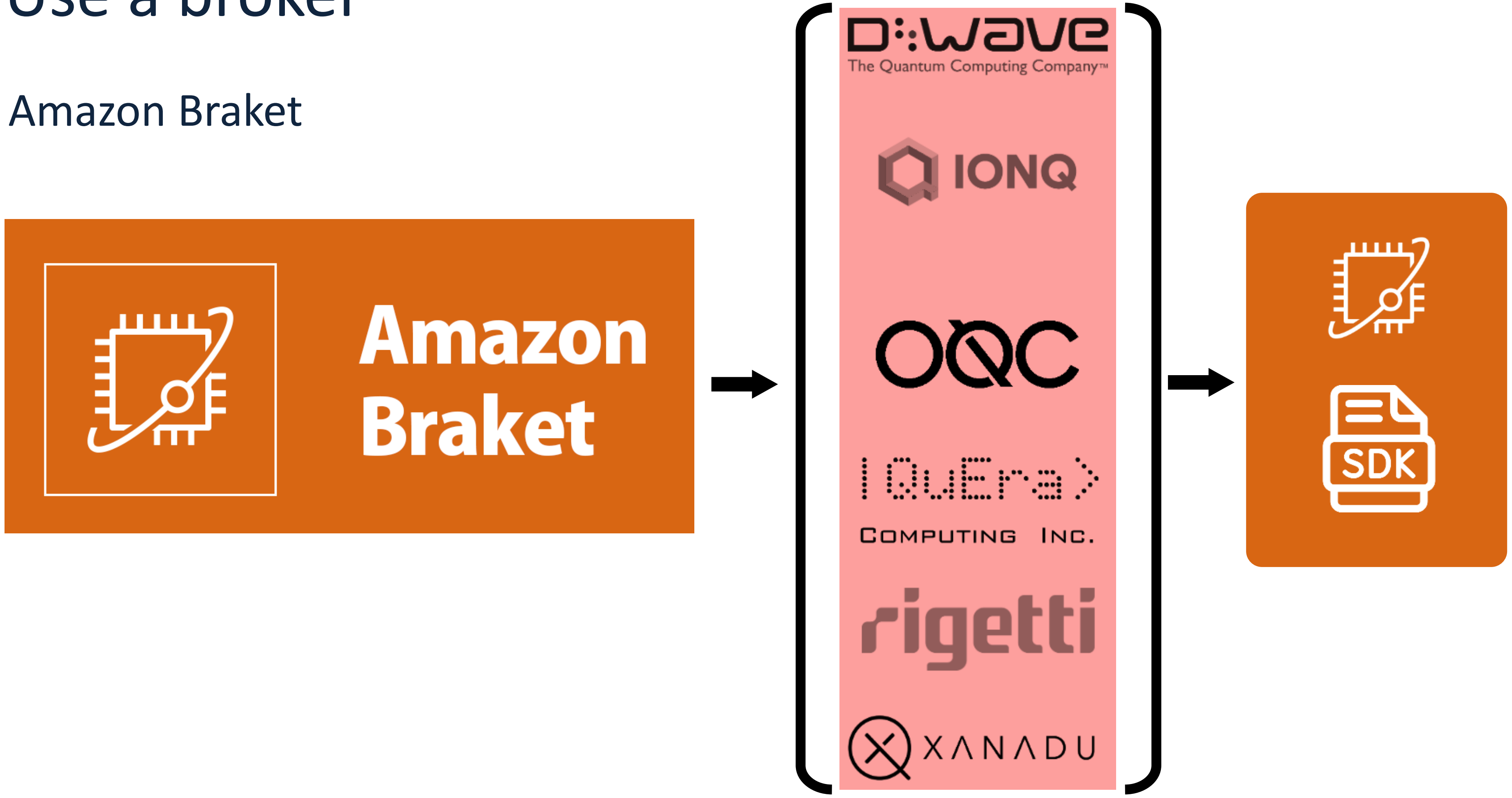


```python
import requests

url = "https://api.ionq.co/v0.1/jobs"
headers = {"Authorization": "hVPDHa6wsboV9vqvsP0AeozS
headers = {"Content-Type": "aplication/json"}
circuit = {
    "qubits": 3,
    "circuit": [
        {
            "gate":"h",
            "target":0
        },
        {
            "gate":"cnot",
            "control": 0,
            "target": 1
        },
        {
            "gate":"cnot",
            "control": 0,
            "target": 2
        }
    ]
}
response = requests.post(url, headers=headers, json=circuit)
```

Current possibilities for quantum services

2. Use a broker

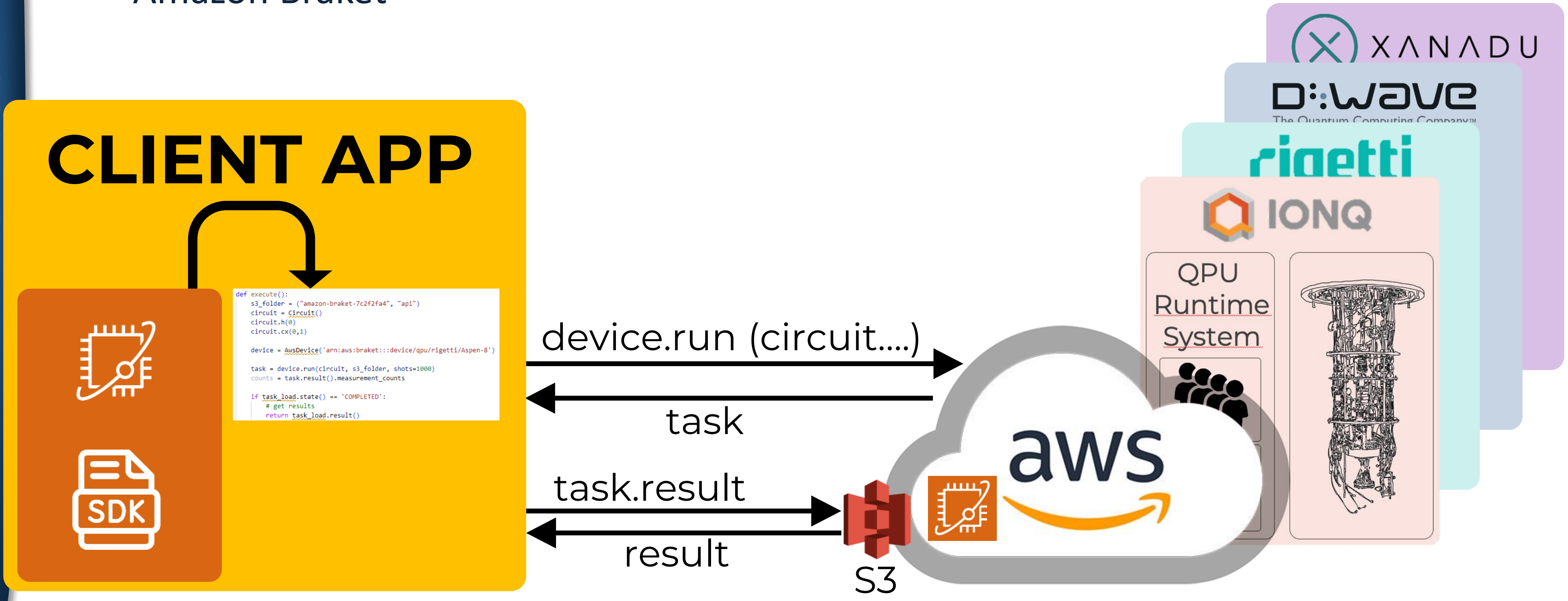
Amazon Braket



Current possibilities for quantum services

2. Use a broker

Amazon Braket

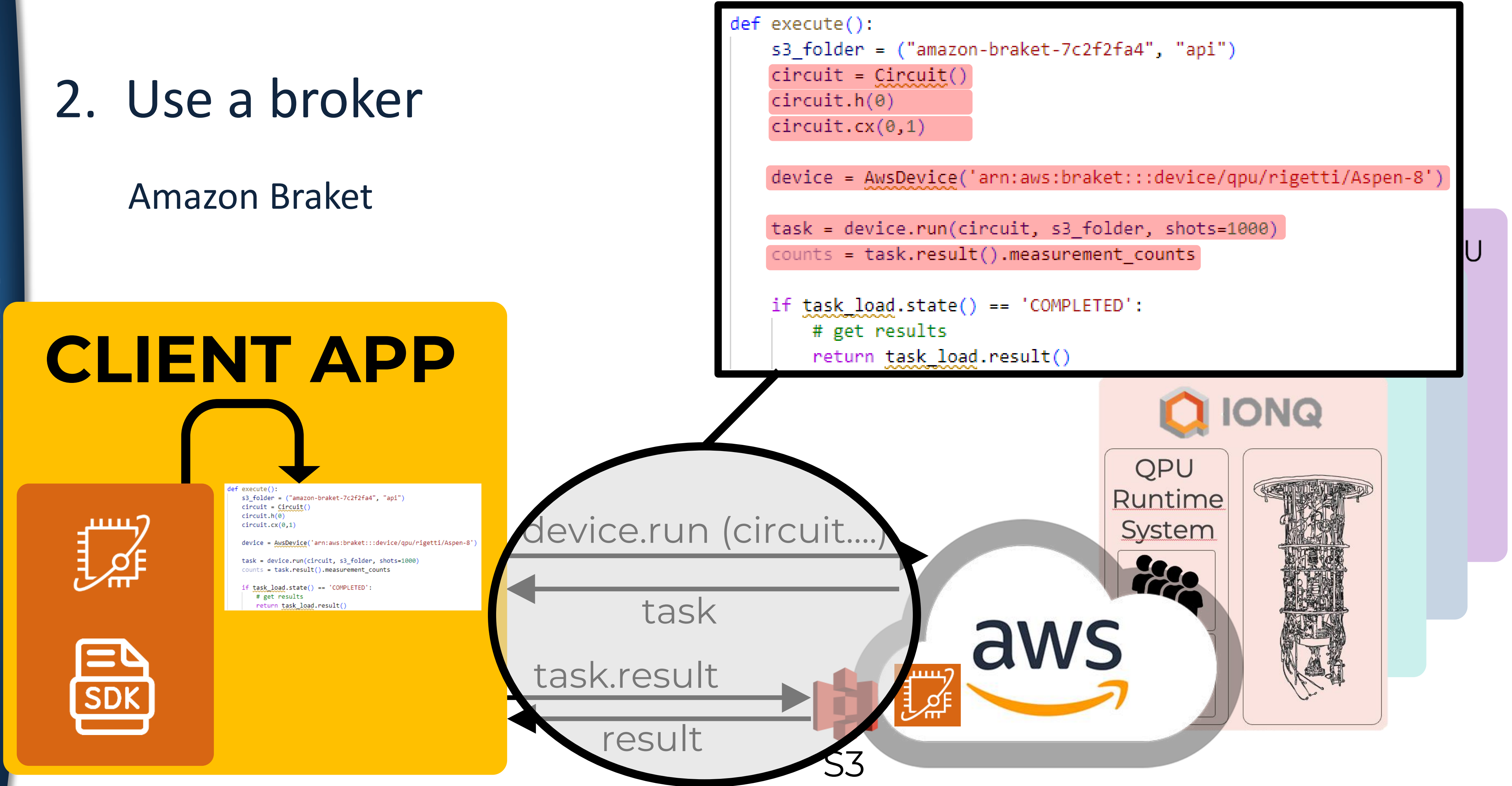


```
def execute():
    s3_folder = ("amazon-braket-7c2f2fa4", "api")
    circuit = Circuit()
    circuit.h(0)
    circuit.cx(0,1)

    device = AwsDevice('arn:aws:braket:::device/qpu/rigetti/Aspen-8')

    task = device.run(circuit, s3_folder, shots=1000)
    counts = task.result().measurement_counts

    if task_load.state() == 'COMPLETED':
        # get results
        return task_load.result()
```

Current possibilities for quantum services

2. Use a broker

   Amazon Braket

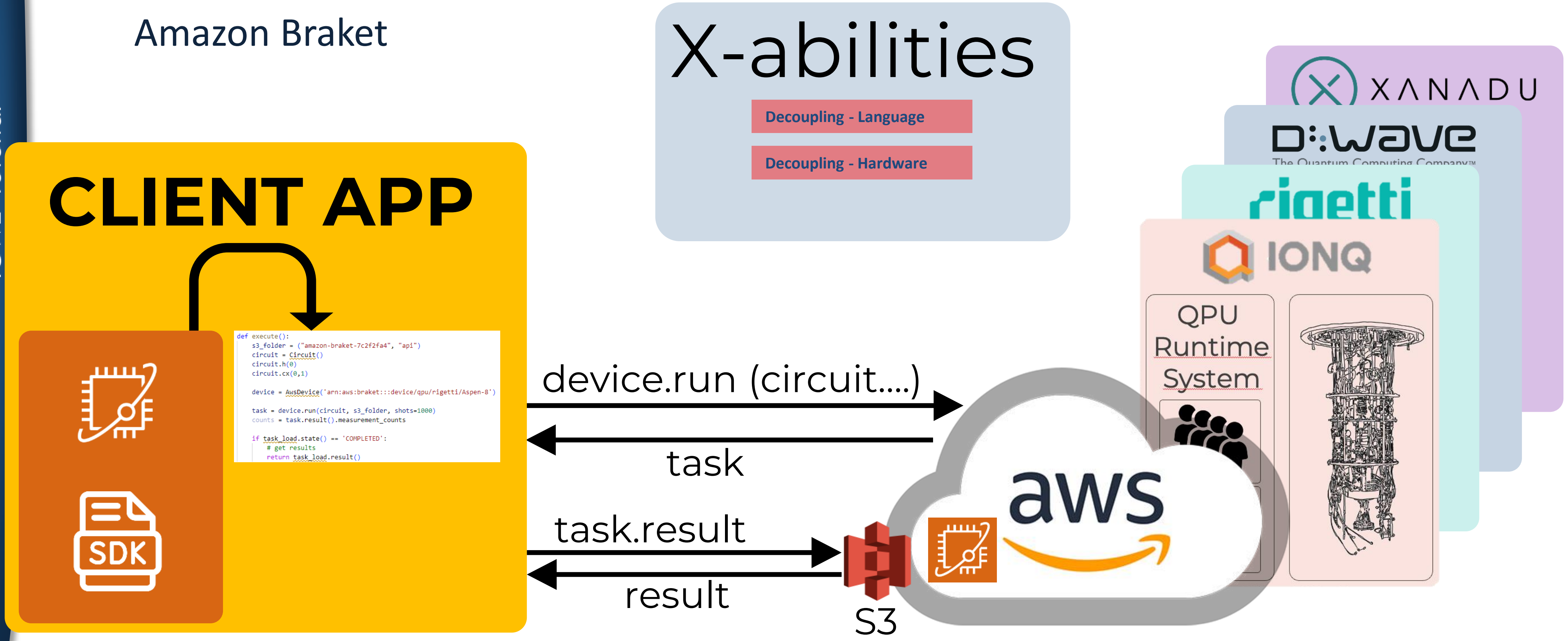


```
def execute():
    s3_folder = ("amazon-braket-7c2f2fa4", "api")
    circuit = Circuit()
    circuit.h(0)
    circuit.cx(0,1)

    device = AwsDevice('arn:aws:braket:::device/qpu/rigetti/Aspen-8')

    task = device.run(circuit, s3_folder, shots=1000)
    counts = task.result().measurement_counts

    if task_load.state() == 'COMPLETED':
        # get results
        return task_load.result()
```

Current possibilities for quantum services

2. Use a broker

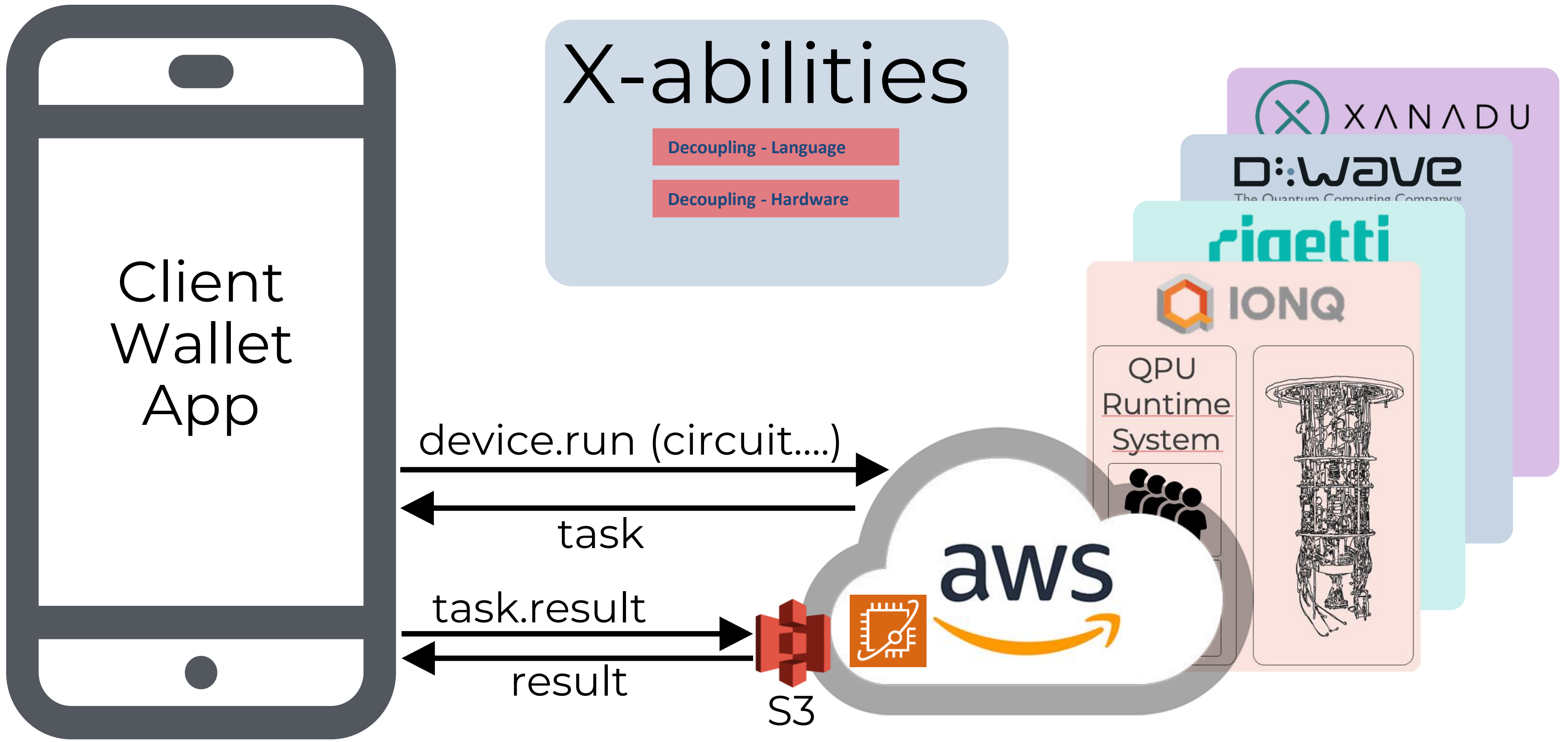
Amazon Braket



```
def execute():
    s3_folder = ("amazon-braket-7c2f2fa4", "api")
    circuit = Circuit()
    circuit.h(0)
    circuit.cx(0,1)

    device = AwsDevice('arn:aws:braket:::device/qpu/rigetti/Aspen-8')

    task = device.run(circuit, s3_folder, shots=1000)
    counts = task.result().measurement_counts

    if task_load.state() == 'COMPLETED':
        # get results
        return task_load.result()
```

Current possibilities for quantum services

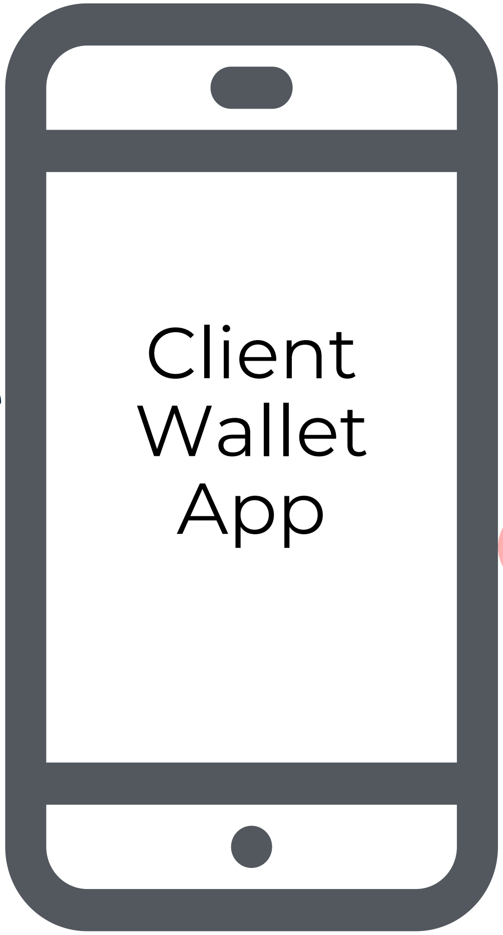
2. Use a broker

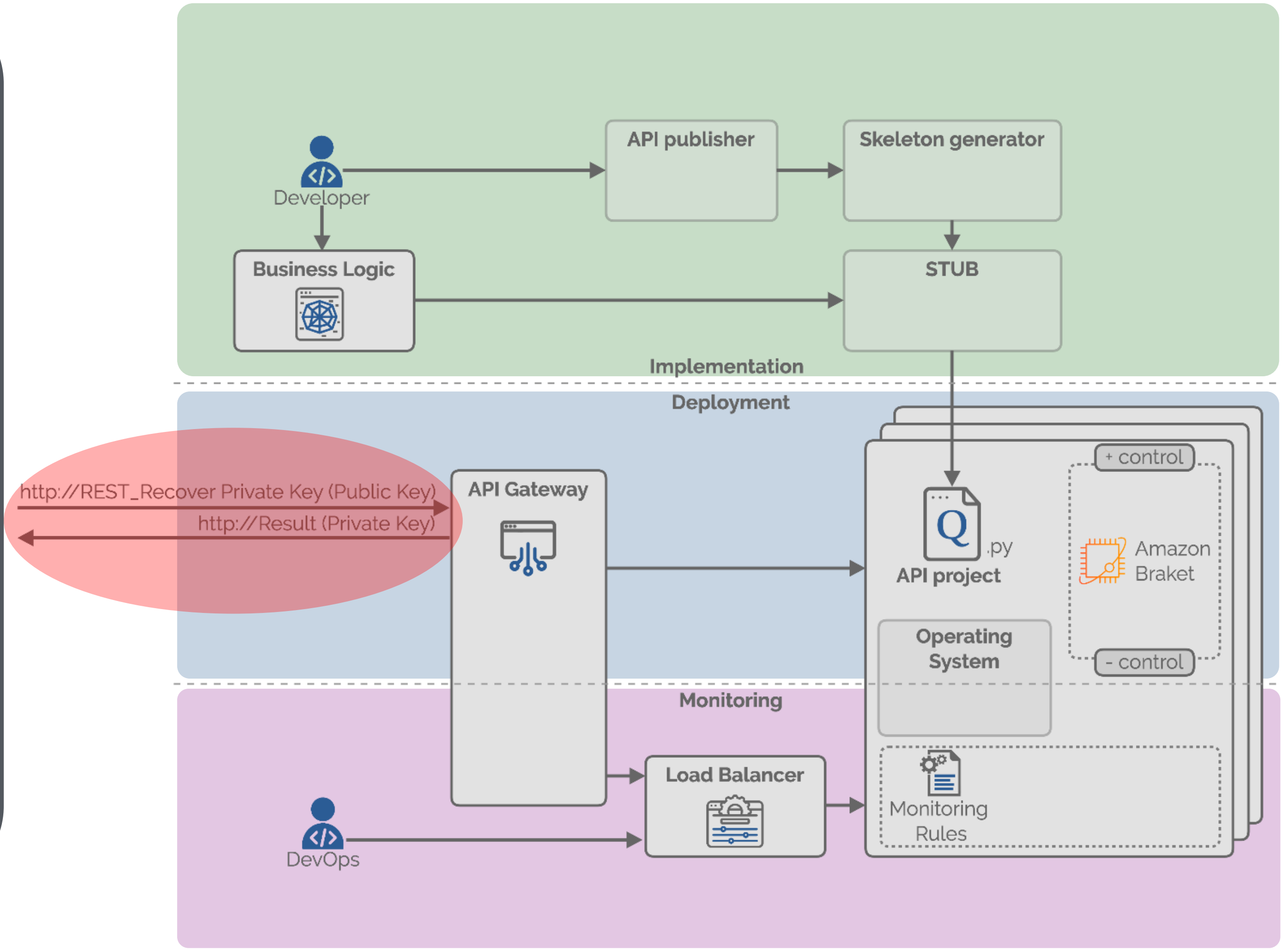


Servitizing quantum circuits

Servitizing quantum circuits

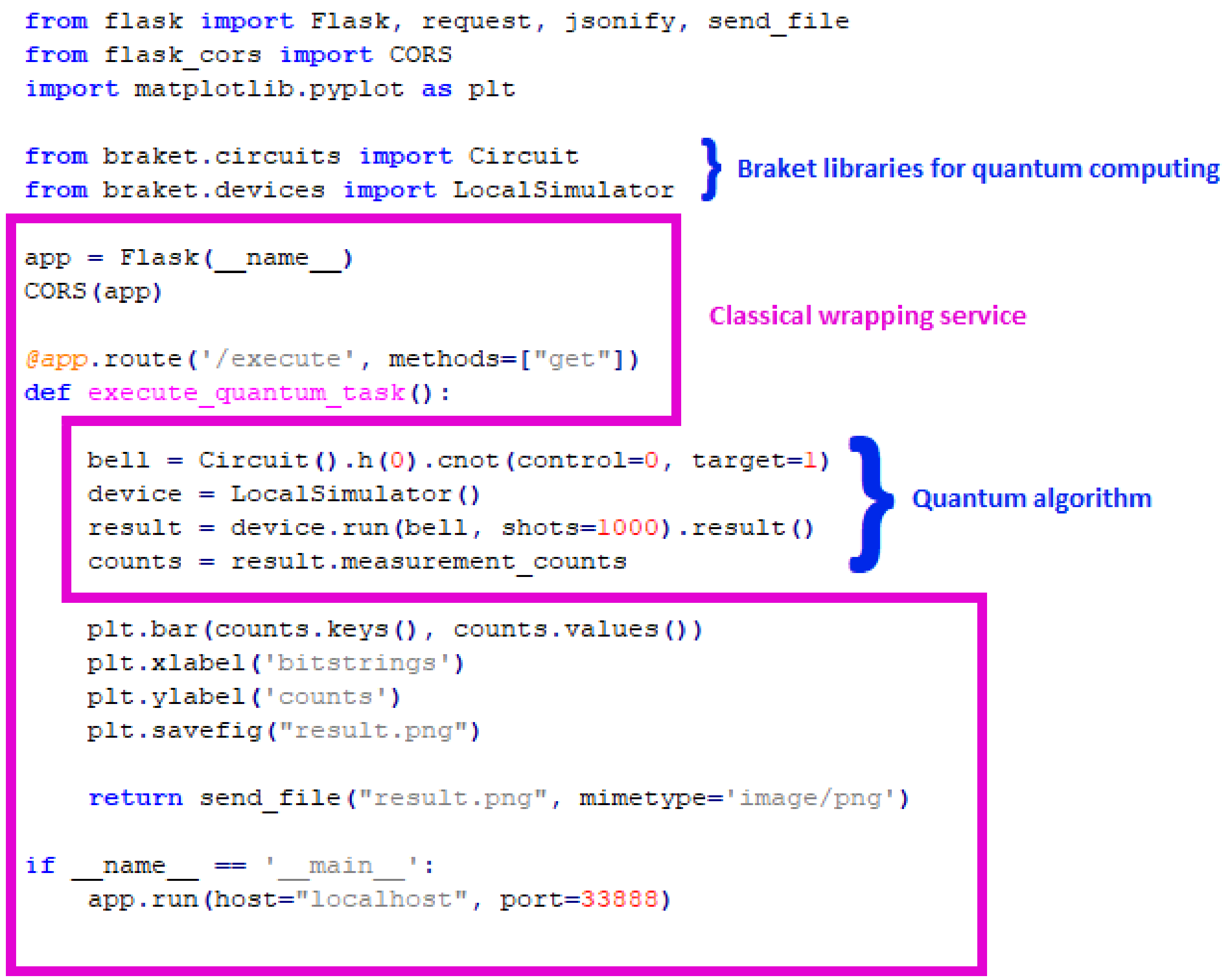
Objective 1
To have service calls instead of program execution requests





Servitizing quantum circuits

We started by encapsulating a quantum circuit inside a classical service acting as a wrapper



```python
from flask import Flask, request, jsonify, send_file
from flask_cors import CORS
import matplotlib.pyplot as plt

from braket.circuits import Circuit
from braket.devices import LocalSimulator

app = Flask(__name__)
CORS(app)

@app.route('/execute', methods=["get"])
def execute_quantum_task():
    bell = Circuit().h(0).cnot(control=0, target=1)
    device = LocalSimulator()
    result = device.run(bell, shots=1000).result()
    counts = result.measurement_counts

    plt.bar(counts.keys(), counts.values())
    plt.xlabel('bitstrings')
    plt.ylabel('counts')
    plt.savefig("result.png")

    return send_file("result.png", mimetype='image/png')

if __name__ == '__main__':
    app.run(host="localhost", port=33888)
```

Braket libraries for quantum computing
Classical wrapping service
Quantum algorithm

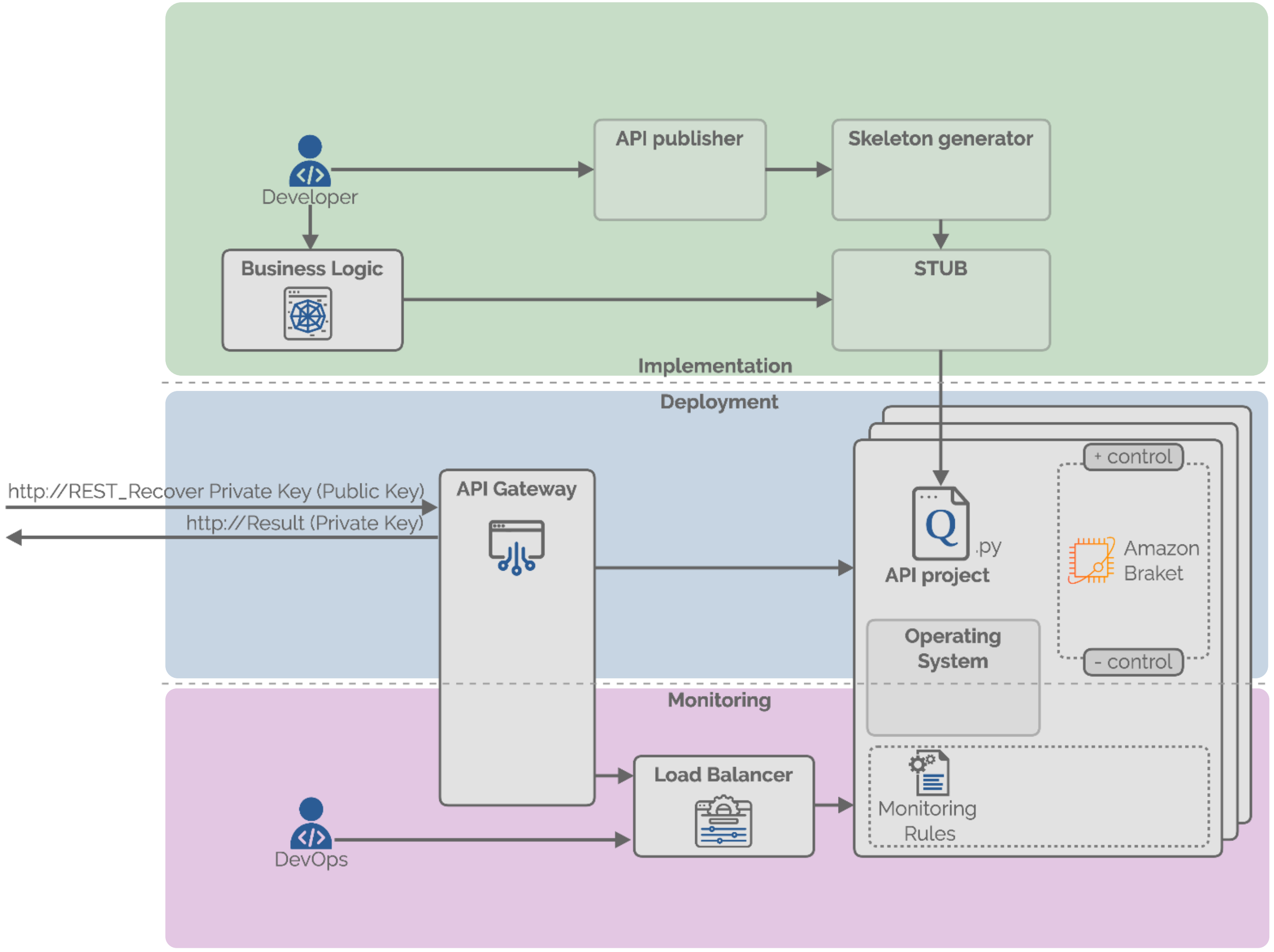
Servitizing quantum circuits

Objective 1

To have service calls instead of program execution requests

Client Wallet App

```
response =
requests.get
(https://aws.amazon
.com/braket/Recove
rPrivateKey/execute)
;
```



Servitizing quantum circuits

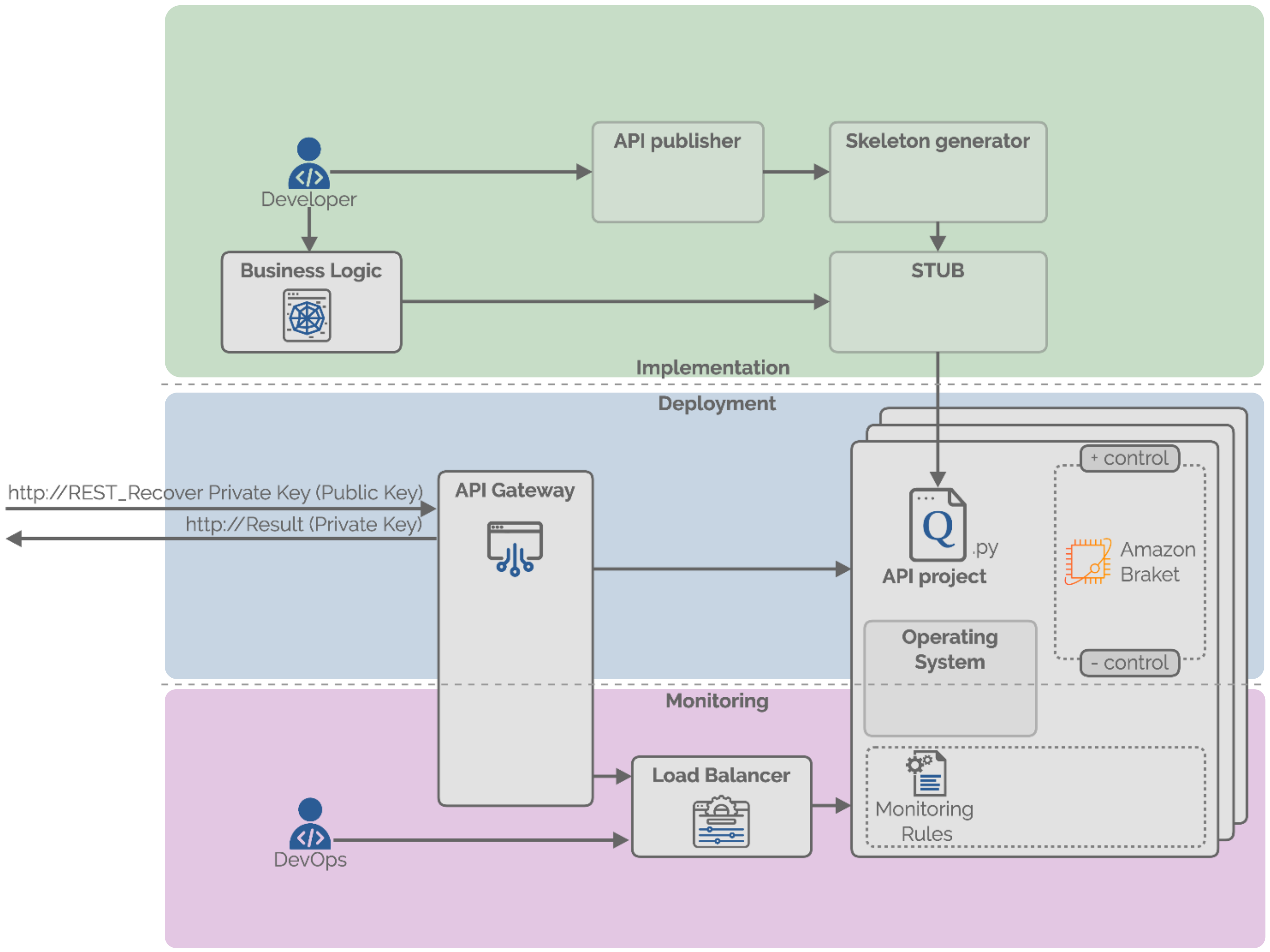
Objective 1

X-abilities

- Decoupling - Language
- Decoupling - Hardware
- Location Independence
- Manteinability
- Reusability

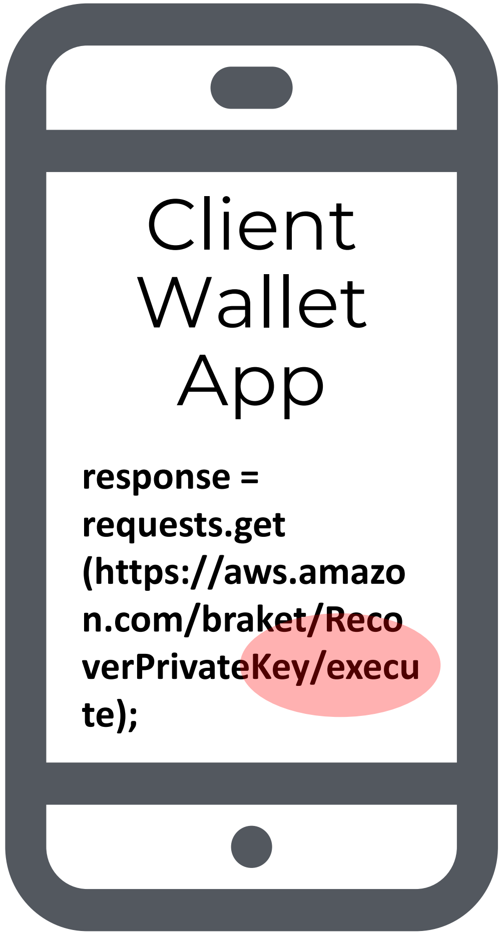
Client Wallet App

```
response =
requests.get
(https://aws.amazon
.com/braket/Recove
rPrivateKey/execute)
;
```

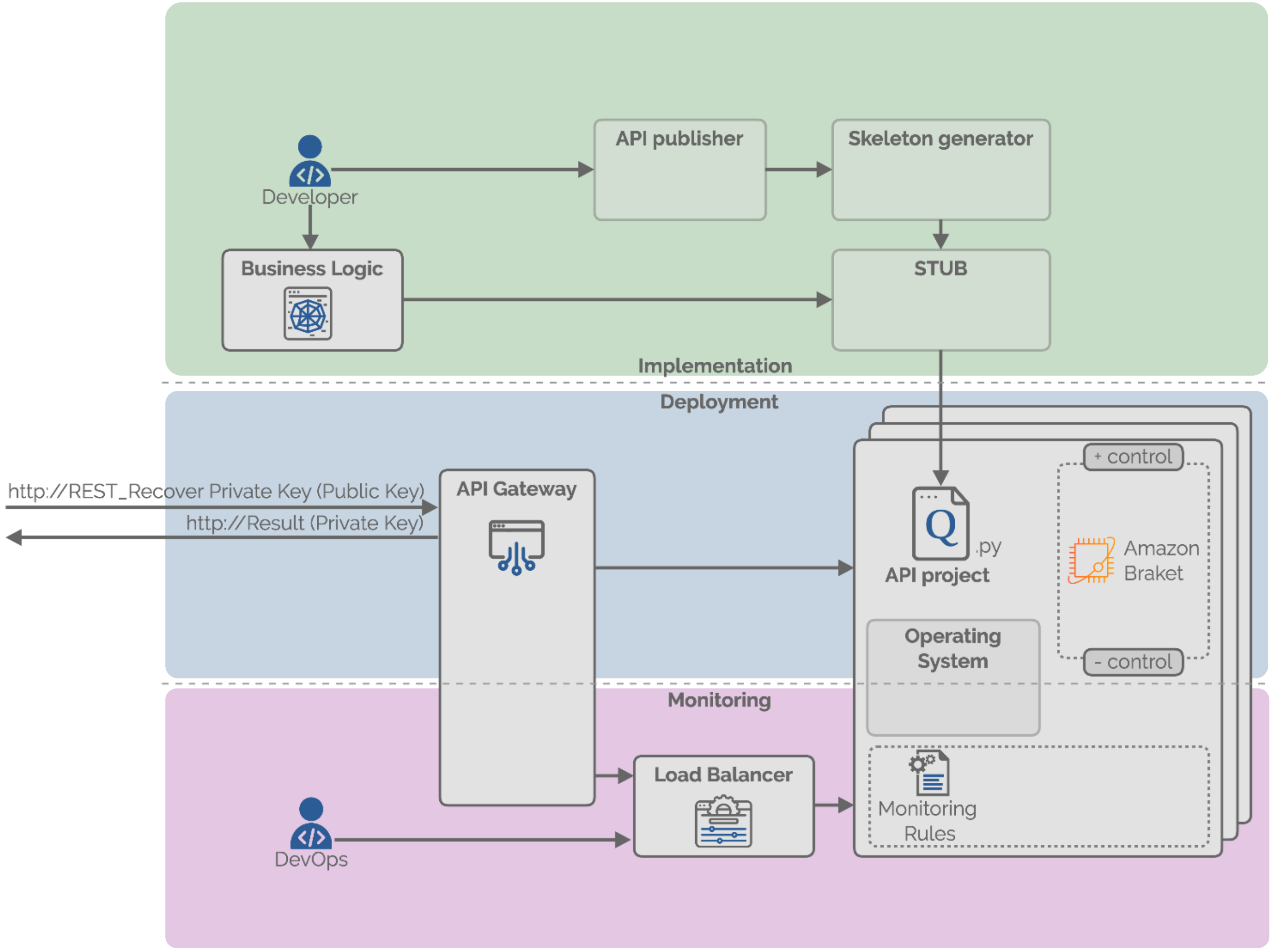


Servitizing quantum circuits

Objective 2
To parameterize the service call

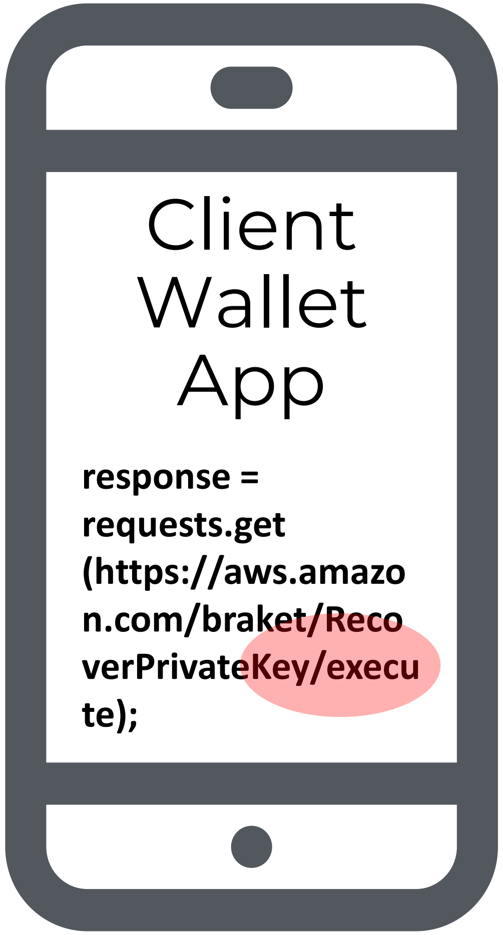


```
response =
requests.get
(https://aws.amazo
n.com/braket/Reco
verPrivateKey/execu
te);
```



Servitizing quantum circuits

Objective 2
To parameterize the service call



```
response =
requests.get
(https://aws.amazo
n.com/braket/Reco
verPrivateKey/execu
te);
```

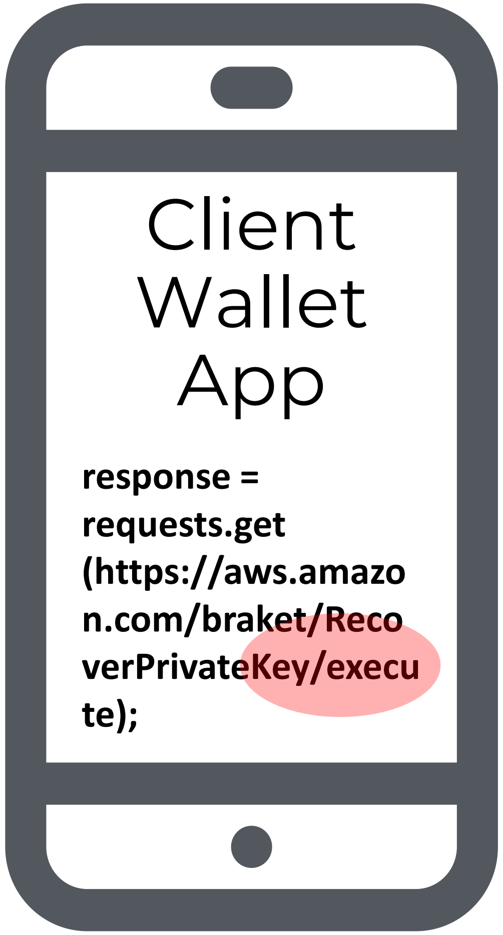
What kind of parameters?

1. Regular parameters for services (eg. Public_key to regenerate private_key)

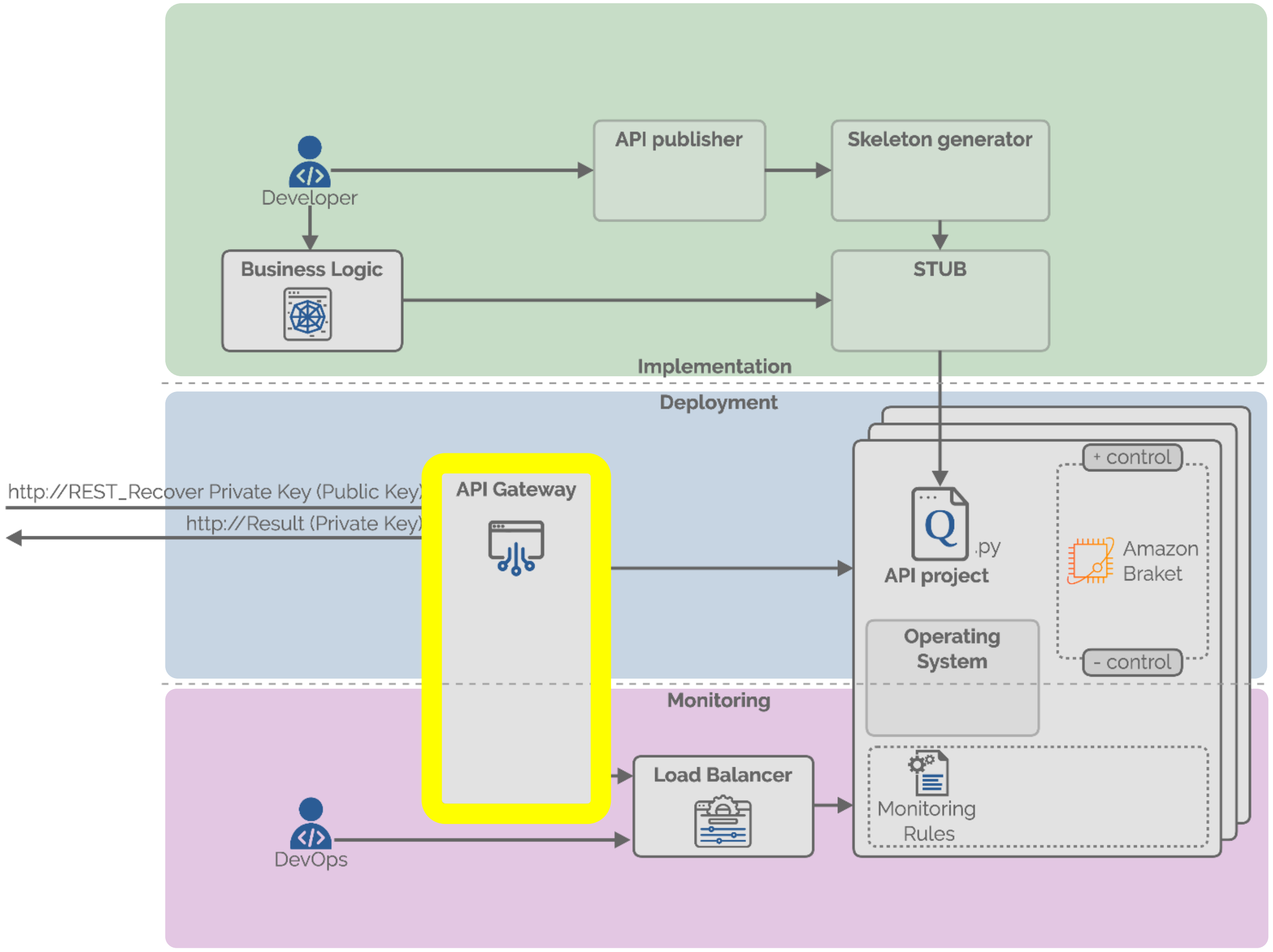
2. Specific parameters for the quantum execution
   a. Number of shoots
   b. Time (maximum)
   c. Cost (maximum)
   d. The QCaaS provider

Servitizing quantum circuits

Objective 2
To parameterize the service call

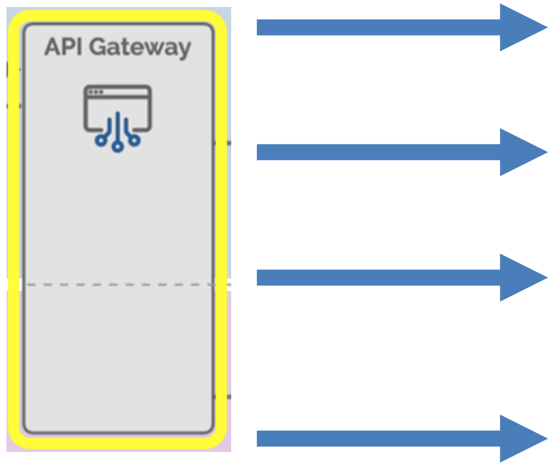


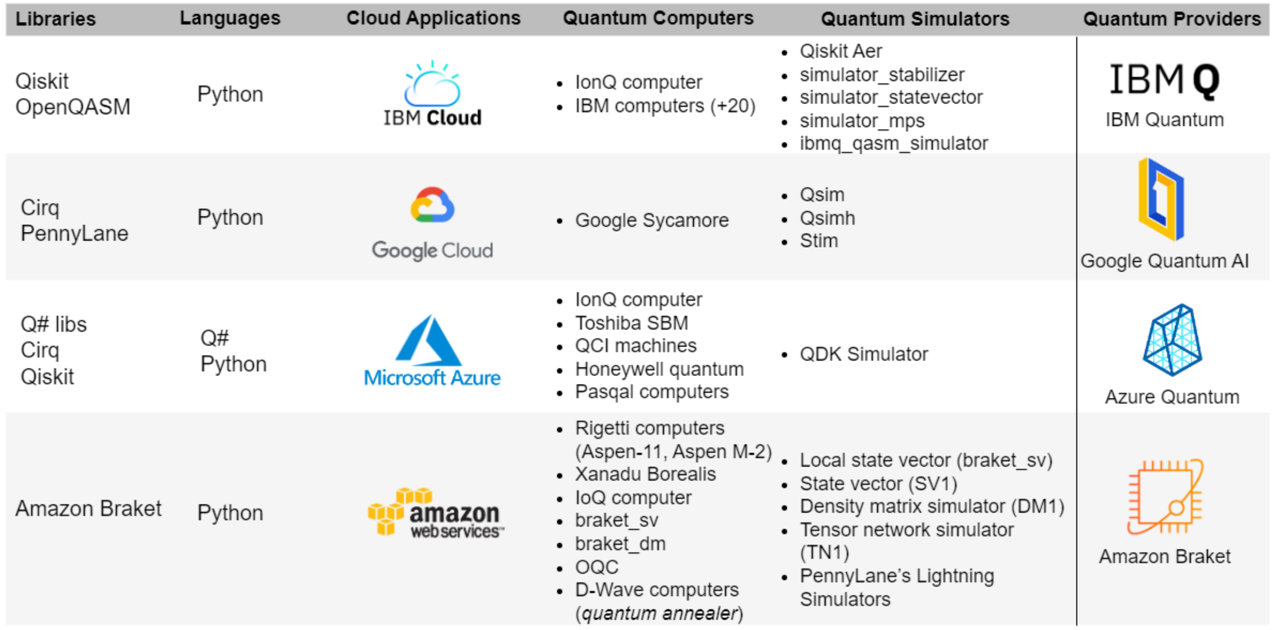
```
response =
requests.get
(https://aws.amazo
n.com/braket/Reco
verPrivateKey/execu
te);
```



Servitizing quantum circuits

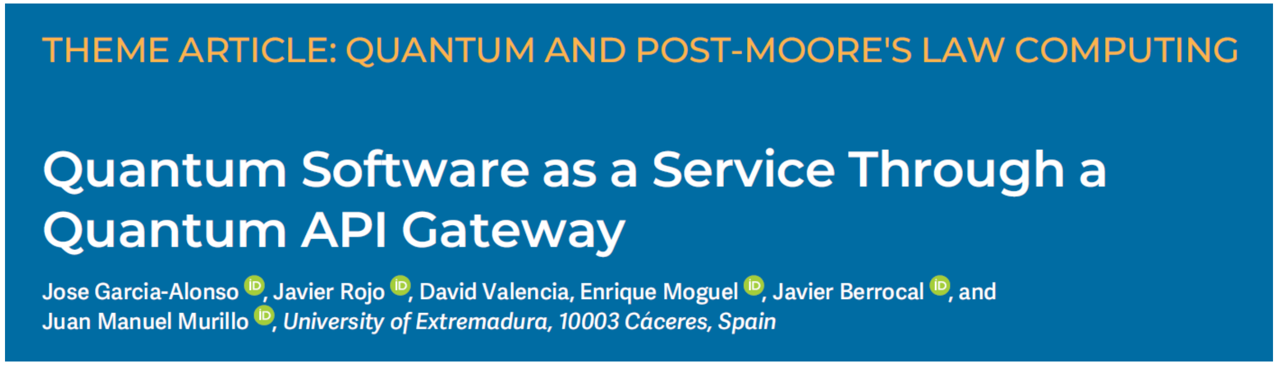
Empower the API Gateway providing it with Quantum awareness: the Quantum API Gateway





Libraries	Languages	Cloud Applications	Quantum Computers	Quantum Simulators	Quantum Providers
Qiskit OpenQASM	Python	IBM Cloud	• IonQ computer • IBM computers (+20)	• Qiskit Aer • simulator_stabilizer • simulator_statevector • simulator_mps • ibmq_qasm_simulator	IBM Q IBM Quantum
Cirq PennyLane	Python	Google Cloud	• Google Sycamore	• Qsim • Qsimh • Stim	Google Quantum AI
Q# libs Cirq Qiskit	Q# Python	Microsoft Azure	• IonQ computer • Toshiba SBM • QCI machines • Honeywell quantum • Pasqal computers	• QDK Simulator	Azure Quantum
Amazon Braket	Python	amazon web services	• Rigetti computers (Aspen-11, Aspen M-2) • Xanadu Borealis • IoQ computer • braket_sv • braket_dm • OQC • D-Wave computers (*quantum annealer*)	• Local state vector (braket_sv) • State vector (SV1) • Density matrix simulator (DM1) • Tensor network simulator (TN1) • PennyLane's Lightning Simulators	Amazon Braket

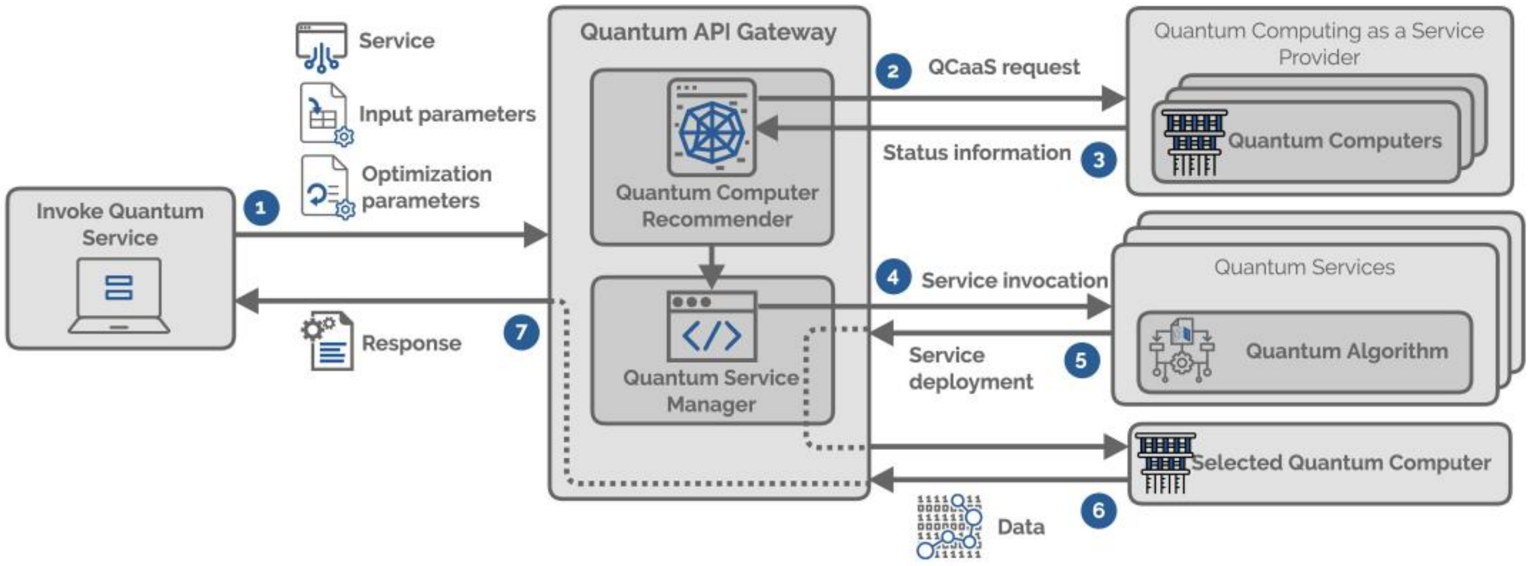
Servitizing quantum circuits



THEME ARTICLE: QUANTUM AND POST-MOORE'S LAW COMPUTING

Quantum Software as a Service Through a Quantum API Gateway

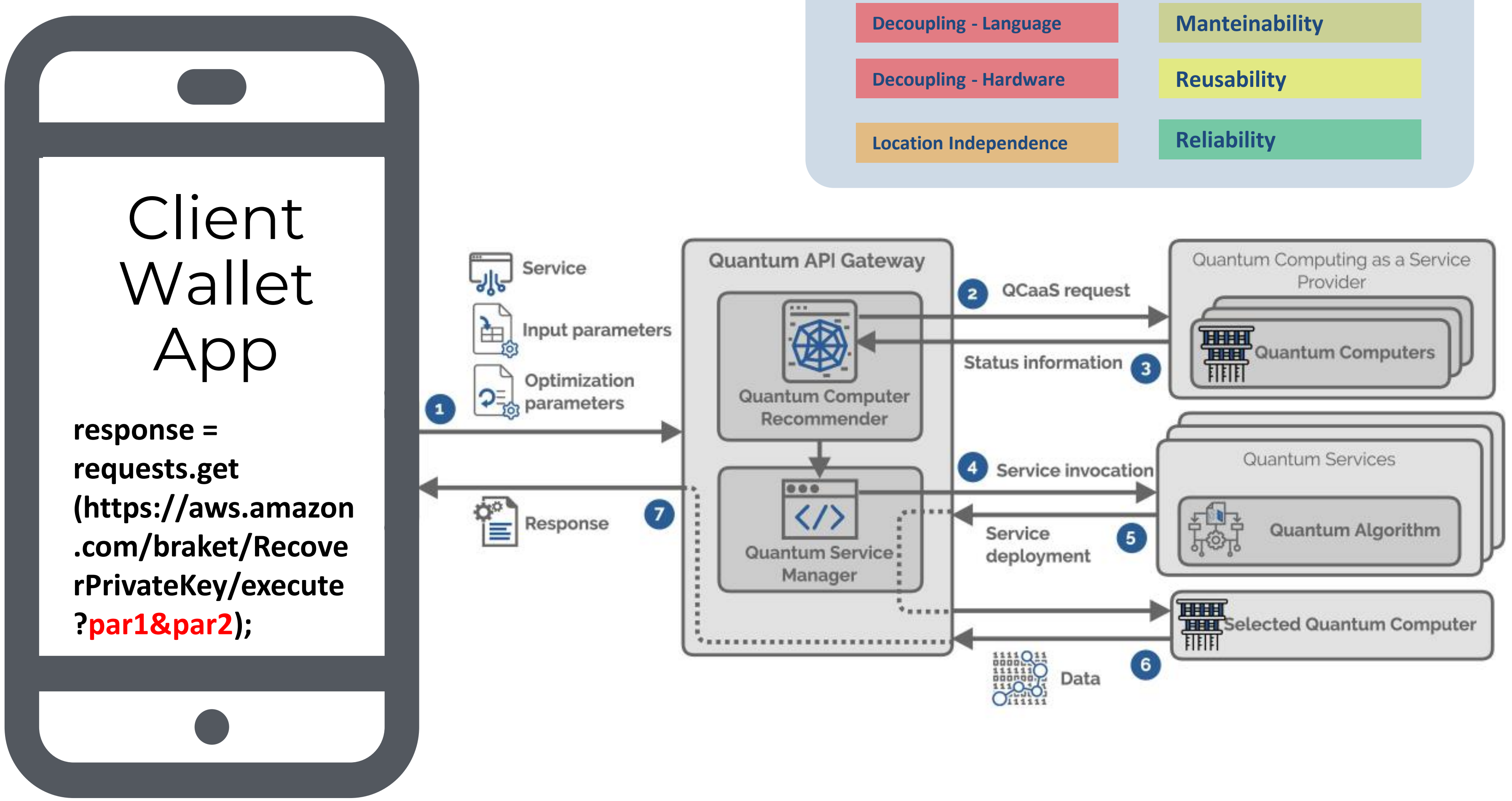
Jose Garcia-Alonso, Javier Rojo, David Valencia, Enrique Moguel, Javier Berrocal, and Juan Manuel Murillo, *University of Extremadura, 10003 Cáceres, Spain*



Servitizing quantum circuits

X-abilities

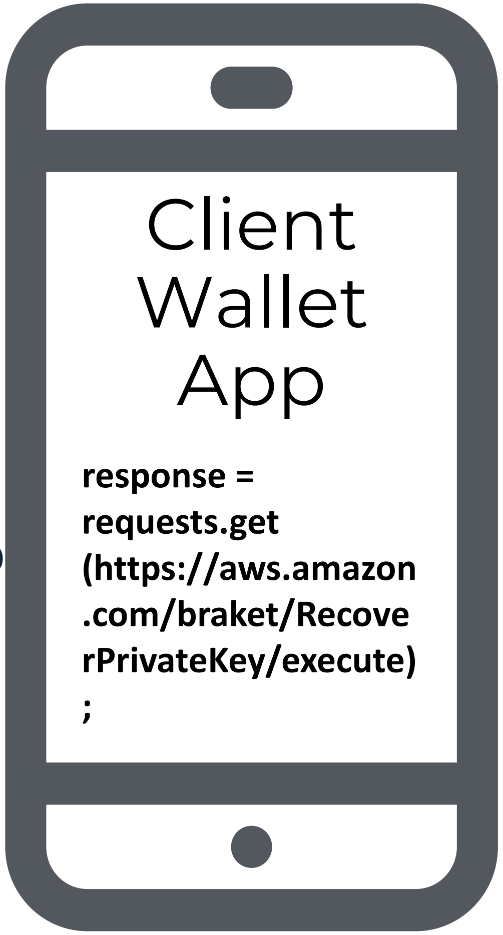
Decoupling - Language	Manteinability
Decoupling - Hardware	Reusability
Location Independence	Reliability



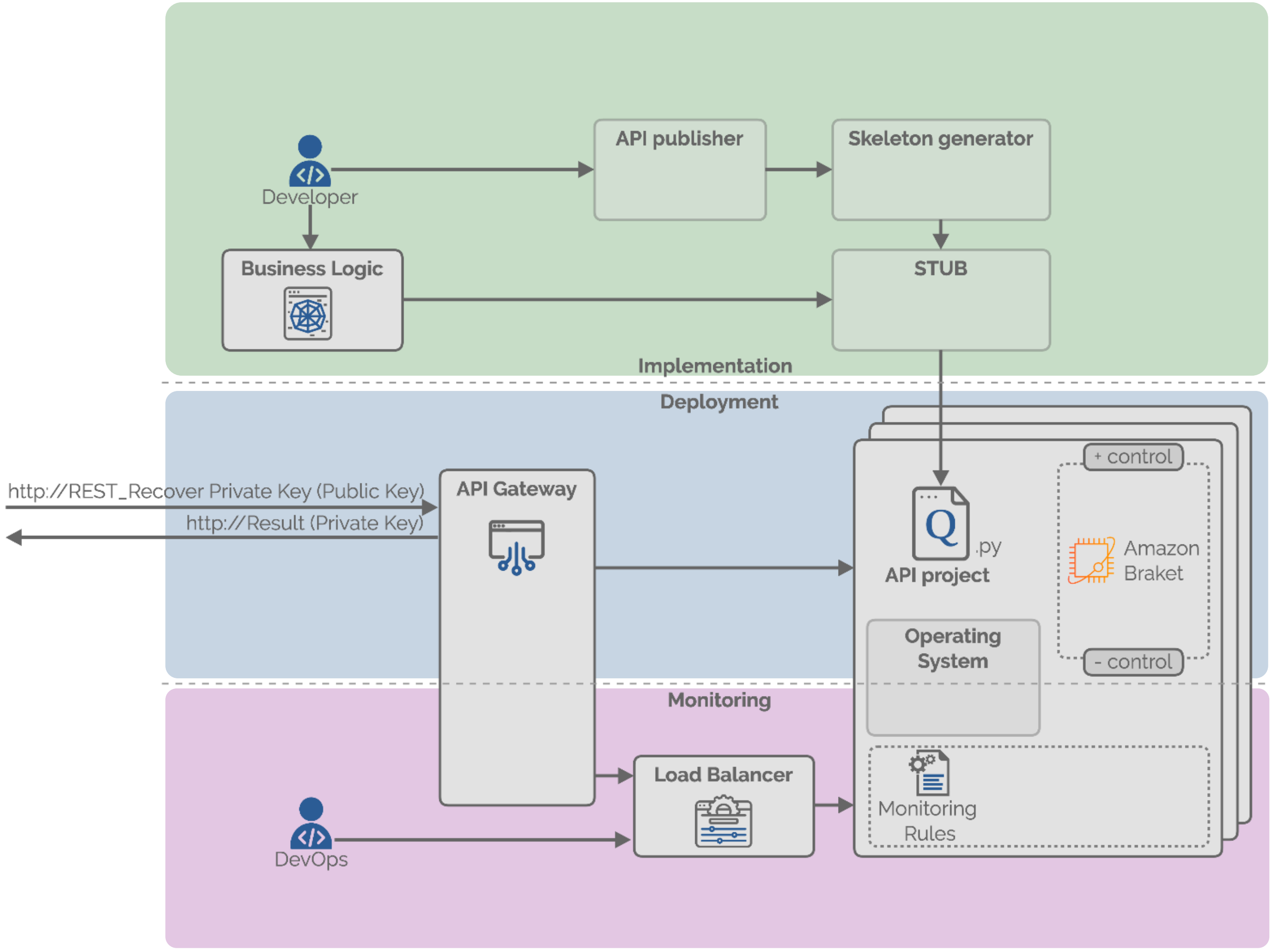
```
response =
requests.get
(https://aws.amazon
.com/braket/Recove
rPrivateKey/execute
?par1&par2);
```

Servitizing quantum circuits

Objective 3
Provide developers with tools like the ones they typically use to build and deploy services

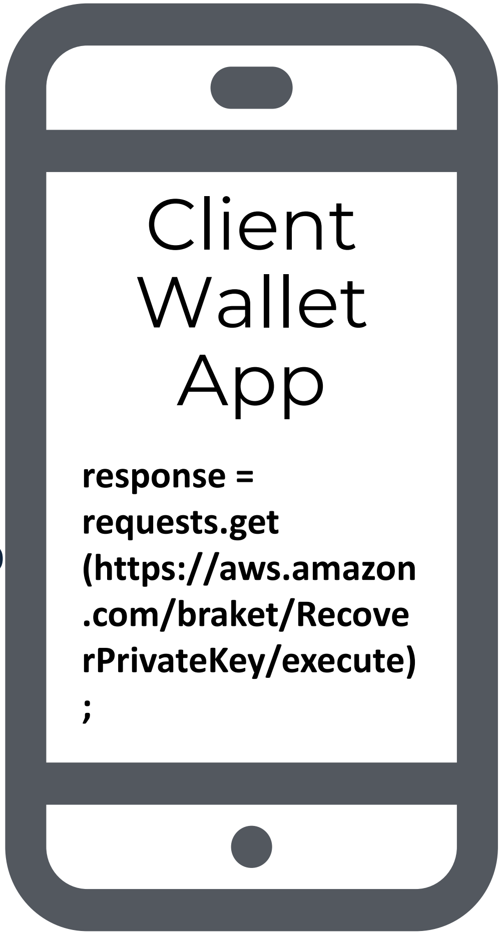


```
response =
requests.get
(https://aws.amazon
.com/braket/Recove
rPrivateKey/execute)
;
```

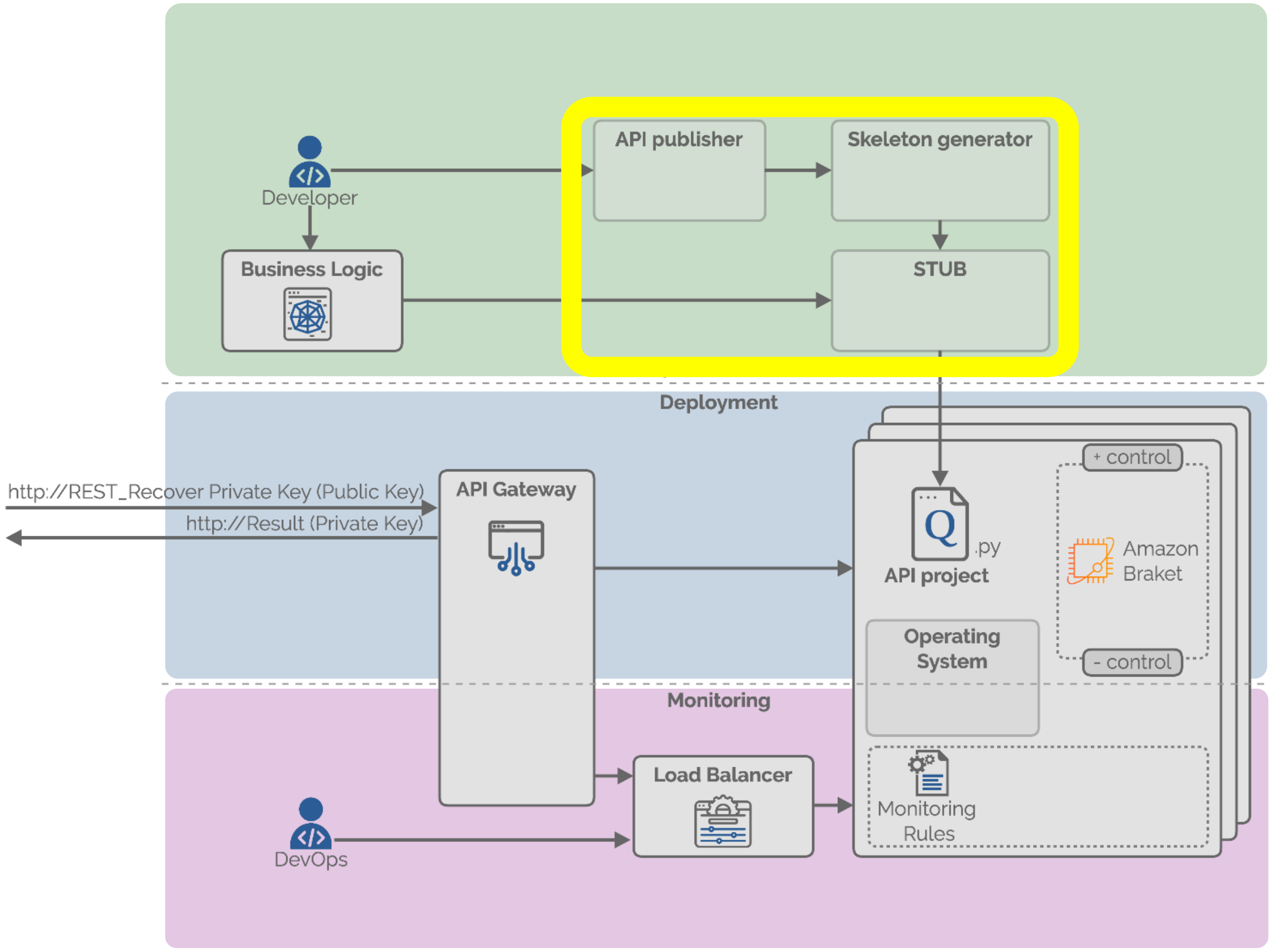


Servitizing quantum circuits

Objective 3
Provide developers with tools like the ones they typically use to build and deploy services



```
response =
requests.get
(https://aws.amazon
.com/braket/Recove
rPrivateKey/execute)
;
```



Quantum Web Services

Jose Garcia-Alonso

Juan M. Murillo

jgaralo@unex.es *juanmamu@unex.es*

Institute of Architecture of Application Systems

University of Stuttgart

UNIVERSIDAD de EXTREMADURA