1 SETTING UP

In this chapter we describe the environment we use to do our test, the tool

1.1 PYTHON ENVIRONMENT

The language used to interface with quantum computer is usually python, in this section we create a virtual environment in python in order to communicate with the IBM quantum Computer and the D-Wave quantum computer.

For our test we manage pyton environments with `conda`, let's start creating the virtual env named `quantum` and activate it with:

```
conda create --name quantum
conda activate quantum
```

For our test and to follow the various example presented both by IBM and D-Wave is also useful to been able of running a Jupyter notebook. We can install Jupyter with:

```
pip install jupyter
```

1.2 IBM QISKIT

To program an architecture gate based and to access IBM quantum computer we use the *Qiskit* software stack. The name Qiskit is a general term referring to a collection of software for executing programs on quantum computers.

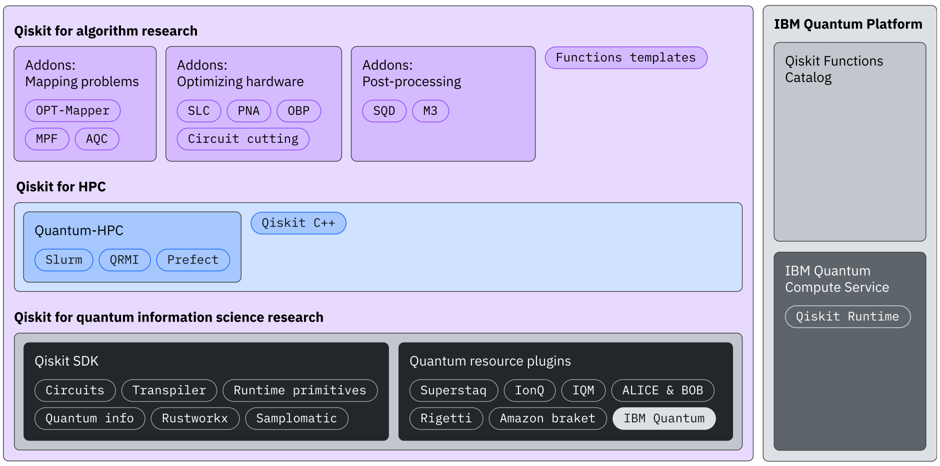

Figure 1.1: Qiskit software stack

The core components are *Qiskit SDK* and *Qiskit Runtime*, the first one is completely open source and allows the developer to define his circuit; the second one is a cloud-based service for executing quantum computations on IBM quantum computer.

1.2.1 Hello World

Following the IBM documentation[1] we can install the SDK and the Runtime with:

```
pip install qiskit matplotlib qiskit[visualization]
pip install qiskit-ibm-runtime
```

If the setup had success we are now able to run a small test to build a Bell state (two entangled qubits). The following code assemble the gates, show the final circuit and use a sampler to simulate on the CPU the result of 1024 runs of the program.

```python
from qiskit import QuantumCircuit
from qiskit.primitives import StatevectorSampler

qc = QuantumCircuit(2)
qc.h(0)
qc.cx(0, 1)
qc.measure_all()

sampler = StatevectorSampler()
result = sampler.run([qc], shots=1024).result()
print(result[0].data.meas.get_counts())
qc.draw("mpl")
```

1.2.2 Transpilation

Each Quantum Processing Unit (QPU) has a specific topology, we need to rewrite our quantum circuit in order to match the topology of the selected device on witch we want to run our program. This phase of rewriting, followed by an optimization, is called transpilation.

Considering, for now, a fake hardware (so we don't need an API key) we can transpile the quantum circuit `qc`, from the code above, to match the topology of a precise QPU:

```python
from qiskit_ibm_runtime.fake_provider import FakeWashingtonV2
from qiskit.transpiler import generate_preset_pass_manager

backend = FakeWashingtonV2()
pass_manager = generate_preset_pass_manager(backend=backend)

transpiled = pass_manager.run(qc)
transpiled.draw("mpl")
```

1.2.3 Execution

1.2.4 A complete example on real hardware

1 https://quantum.cloud.ibm.com/docs/en/guides/install-qiskit