

## Customized user metrics

### *Exemplified with Quantum Computing algorithms*

This chapter will show one example of monitoring the performance data of any python program, with independence of it was originally designed as a data pipeline or perhaps for quite different purpose that might need to be analyzed in the context of a data preparation project. It is not necessary to install or setup any of the technical resources we will see below to understand the contents of this chapter. Consequently, this section will not necessarily be a hands-on exercise.

The use case that we will inspect is one of the most basic algorithms in quantum computing: The Quantum Teleportation. For this workshop, the only thing we need to know about it is that working with quantum computing implies, in the vast majority of the current use cases, to code an algorithm in Python and, optionally for educational purposes, in a Jupyter environment like the IBM Quantum Lab.

### **1. Enable the monitoring of an algorithm**

The only thing that we need to do is to add some lines of code in the quantum algorithm as described in the Databand documentation and more specifically [here](#) and [here](#). The following screenshots are taken in the IBM Quantum Lab environment and indicate the sections in the code that we want to track and the databand api calls that we need to embed.

The screenshot displays the IBM Quantum Lab environment with a Jupyter notebook titled 'Teleport\_Databand.ipynb'. The notebook contains the following code sections:

- Installation and Imports:**

```
[1] pip install dbnd
```

```
[16] import numpy as np
# Importing classical Qiskit elements
from qiskit import QuantumCircuit, transpile, Aer, IBMQ
from qiskit.visualization import *
from qiskit_aer import *
from qiskit.quantum_info import Statevector
from qiskit.providers.aer import QasmSimulator
from qiskit.quantum_info import partial_trace, Statevector
from qiskit.providers.aer import InstructionSet
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from qiskit.circuit.library import State
from numpy import pi
```
- dbnd Configuration:**

```
[17] from dbnd import dbnd_tracking, log_duration, log_metric
with dbnd_tracking(
    "test": {
        "databand_url": "http://databand-ml-databand.172.17.0.1:8080",
        "databand_project": "qiskit-ml-databand",
        "databand_token": "qiskit-ml-databand-token"
    },
    job_name="Quantum Teleport",
    run_name="Quantum Sim",
    project_name="Quantum project"
):
    pass
```
- Quantum Circuit Setup and Metrics Logging:**

```
[18] from qiskit import QuantumRegister, ClassicalRegister, Aer, IBMQ
cirq_qr = QuantumRegister(2, 'qr')
cirq_cr = ClassicalRegister(1, 'cr')
circuit = QuantumCircuit(cirq_qr, cirq_cr)
psi = random_statevector(2)

# Log the metrics to be used in dbnd
log_metric('Quantum State Vector 0 Seed', np.real(psi[0]))
log_metric('Quantum State Vector 1 Seed', np.real(psi[1]))
log_metric('Quantum State Vector 0 Imag', np.imag(psi[0]))
log_metric('Quantum State Vector 1 Imag', np.imag(psi[1]))

# Initial gate = Initialize(psi)
init_gate = Initialize(psi)
circuit.append(init_gate, [0])
circuit.barrier()
out_vector = Statevector.from_instruction(circuit)
plot_state(out_vector, out_vector)
```
- Quantum Circuit Diagram:** A diagram showing three qubits (qubit 0, qubit 1, qubit 2) initialized to  $|0\rangle$ . Qubit 0 and 1 are entangled via a CNOT gate. Qubit 1 and 2 are entangled via a CNOT gate. The circuit ends with a measurement on qubit 1 and a classical control line from qubit 1 to qubit 0.
- Performance Logging:**

```
[20] sim = Aer.get_backend('aer_simulator')
circuit.save_statevector()

# Log the duration of the execution
with log_duration('Quantum Execution'):
    out_vector = sim.run(circuit).result().get_statevector()
plot_state(out_vector, out_vector)
```
- Final State Visualization:** Three Bloch spheres for qubit 0, qubit 1, and qubit 2, showing the final state of each qubit after the circuit execution.

As you can see, we have simply used the functions:

- `dbnd_tracking` to enable the tracking

- `log_metric` to track variable parameters of the algorithm
- `log_duration` to track the elapsed time of the algorithm execution

Don't forget to add your credentials in the pipeline definition header, like we did in the chapter 9 Python pipelines

## 2. Observe the data collected

The metrics of our algorithm will be collected and displayed in the Metrics section of each run. Remember to select user metric types (default choice) and click on the purple icons of the metrics you want to be added to the graph.

The variability of the parameters in the algorithm execution will be displayed this way:



Analogously, the following graph is an indication of the execution time of the algorithm. Note that it is not the full execution time of the python program, but only the elapsed time that the quantum computer (actually, the simulator) was occupied executing the quantum circuit (the materialization of our algorithm)



[Previous Section: Alerts and exceptions](#)

[Return to main](#)