

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 shows the IBM Quantum Lab interface with a Jupyter Notebook titled 'Teleport_Databand.ipynb'. The notebook contains the following code sections:

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

install the databand package

```
[16] import numpy as np
# Importing classical circuit elements
from qiskit import QuantumCircuit, transpile, Aer, IBMQ
from qiskit.visualization import *
from joblib import Parallel, delayed
from qiskit.providers.aer import QasmSimulator
from qiskit.quantum_info import Statevector, partial_trace, Statevector
from qiskit.extensions import Initialize
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from qiskit.circuit.library import State
from numpy import pi
```

add the databand credentials

```
[17] from dbnd import dbnd_tracking, log_duration, log_metric
with dbnd_tracking(
    "test": {
        "databand_url": "http://databand-ml-databand.172.16.17.100:8080-00000000-0000-0000-0000-000000000000",
        "databand_project_name": "qiskit-ml-databand",
        "databand_credentials": "qiskit-ml-databand-credentials",
        "databand_credentials_file": "qiskit-ml-databand-credentials.json",
        "databand_credentials_file_path": "qiskit-ml-databand-credentials.json"
    }
):
    job_name="Quantum Teleport",
    job_number="Quantum Job",
    project_name="Quantum project"
)
pass
```

log the metrics you want to display

```
[18] from qiskit import QuantumRegister, ClassicalRegister, Aer, IBMQ
from qiskit import QuantumCircuit, transpile, Aer, IBMQ
from qiskit.providers.aer import QasmSimulator
from qiskit.quantum_info import Statevector, partial_trace, Statevector
from qiskit.extensions import Initialize
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from qiskit.circuit.library import State
from numpy import pi

# Define our metrics to log in Databand
log_metric('Quantum State Vector 0 Real', np.real(psi[0]))
log_metric('Quantum State Vector 0 Imag', np.imag(psi[0]))
log_metric('Quantum State Vector 1 Real', np.real(psi[1]))
log_metric('Quantum State Vector 1 Imag', np.imag(psi[1]))

# Initialize the circuit
init_gate = Initialize(psi)
circuit.append(init_gate, [0])
circuit.measure([0], [0])
circuit.barrier()
out_vector = Statevector.from_instruction(circuit)
plot_state(out_vector, out_vector)
```

log the performance of important parts of the algorithm

```
[19] from qiskit import QuantumRegister, ClassicalRegister, Aer, IBMQ
from qiskit import QuantumCircuit, transpile, Aer, IBMQ
from qiskit.providers.aer import QasmSimulator
from qiskit.quantum_info import Statevector, partial_trace, Statevector
from qiskit.extensions import Initialize
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from qiskit.circuit.library import State
from numpy import pi

# Define our metrics to log in Databand
log_metric('Quantum State Vector 0 Real', np.real(psi[0]))
log_metric('Quantum State Vector 0 Imag', np.imag(psi[0]))
log_metric('Quantum State Vector 1 Real', np.real(psi[1]))
log_metric('Quantum State Vector 1 Imag', np.imag(psi[1]))

# Initialize the circuit
init_gate = Initialize(psi)
circuit.append(init_gate, [0])
circuit.measure([0], [0])
circuit.barrier()
out_vector = Statevector.from_instruction(circuit)
plot_state(out_vector, out_vector)
```

The notebook also displays three Bloch sphere visualizations for qubit 0, qubit 1, and qubit 2, showing their state evolution. The qubit 0 sphere shows a state vector pointing towards the top pole (|0>). The qubit 1 and qubit 2 spheres show state vectors pointing towards the bottom pole (|1>).

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](#)