# Spring 2024 Kickoff

*Sam Bieberich*

# Announcements

- Thanks for coming back again
- QHack
  - https://qhack.ai/online-events/#streaming-sessions

# What are we doing now?

We didn’t finish the book last semester, but instead of just going chapter by chapter, we will be targeting the prerequisites for the exam:

https://www.ibm.com/training/certification/ibm-certified-associate-developer-quantum-computation-using-qiskit-v02x-C0010300

On next slide too…

**Number of questions:** 60
**Number of questions to pass:** 44

**Time allowed:** 90 minutes
**Status:** Live

| | |
|---|---|
| Section 1: Perform Operations on Quantum Circuits | 47% |
| Section 2: Executing Experiments | 3% |
| Section 3: Implement BasicAer: Python-based Simulators | 3% |
| Section 4: Implement Qasm | 1% |
| Section 5: Compare and Contrast Quantum Information | 10% |
| Section 6: Return the Experiment Results | 7% |
| Section 7: Use Qiskit Tools | 1% |
| Section 8: Display and Use System Information | 3% |
| Section 9: Construct Visualizations | 19% |
| Section 10: Access Aer Provider | 6% |

# Section 7

*Using Qiskit Tools*

# What does this mean?

On the website, the only thing in this section says:

1. Monitor the status of a job instance

Can be done on the website for IBMQ or via the terminal/python.

# Monitoring Jobs on IBMQ

- Go to the IBMQ website
  - https://quantum.ibm.com/

# IBMQ

**Recent jobs** View all

0
Pending

18
Completed jobs

| Job ID | Status | Created | Completed | Compute resource |
|---|---|---|---|---|
| cmtf8pdhk6gfko899emg | Completed | 11 days ago | 11 days ago | simulator_statevector |
| cmtf06vn6tkp35gkhfvg | Completed | 11 days ago | 11 days ago | ibmq_qasm_simulator |
| cko7etij5hs4e9vpm1h0 | Completed | 4 months ago | 4 months ago | ibmq_qasm_simulator |
| cko6tpn8mm6ij0ssf9mg | Completed | 4 months ago | 4 months ago | ibmq_qasm_simulator |
| cknl4srgl0ct1nq13qdg | Completed | 4 months ago | 4 months ago | ibmq_qasm_simulator |

Jobs /

# civb258b07rkqj8uqv0g

Edit Tags

**Details**

19.2s
Total completion time

simulator_statevector
Compute resource

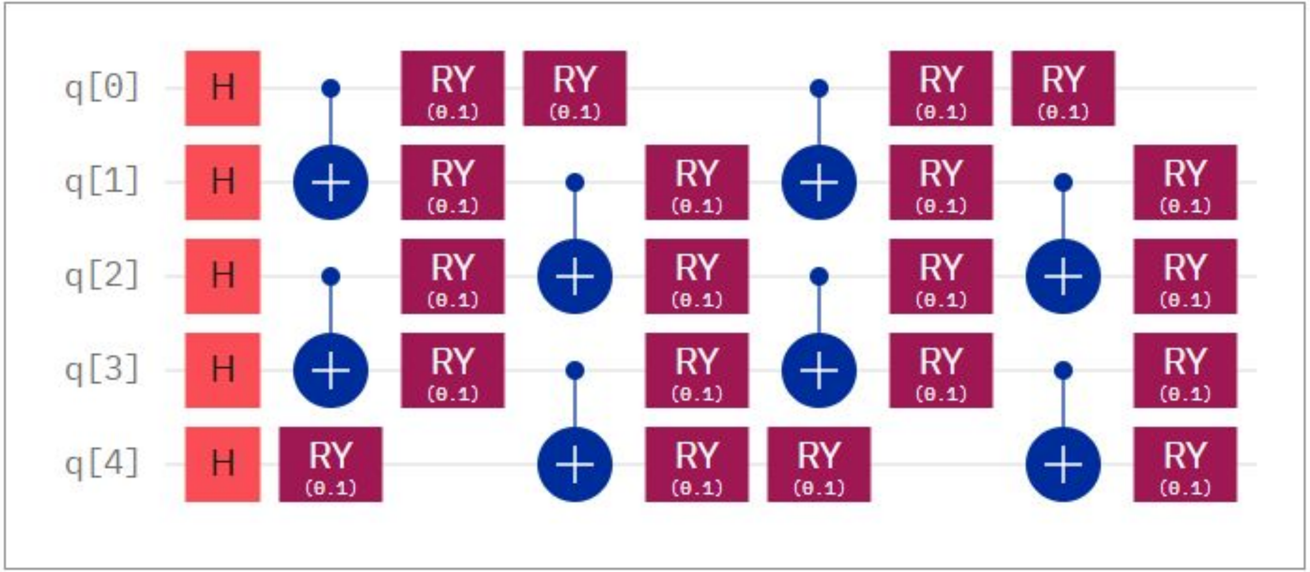
| | |
|---|---|
| Status: | Completed |
| Instance: | ibm-q/open/main |
| Program: | estimator |
| # of shots: | 10000 |
| # of circuits: | 1 |
| Session Id: | civb258b07rkqj8uqv0g |

Status Timeline

- Created: Jul 24, 2023 12:12 PM
- In queue: 13.9s
- Running: Jul 24, 2023 12:13 PM
  Qiskit runtime usage: 0ms
- Completed: Jul 24, 2023 12:13 PM

Circuit

Diagram | Qasm | Qiskit



Circuit

Diagram | **Qasm** | Qiskit

```
OPENQASM 2.0;
include "qelib1.inc";
qreg q[5];
h q[0];
h q[1];
h q[2];
h q[3];
h q[4];
cx q[0],q[1];
cx q[2],q[3];
ry(0.1) q[0];
ry(0.1) q[1];
ry(0.1) q[2];
ry(0.1) q[3];
ry(0.1) q[4];
cx q[1],q[2];
```

Open in composer

Circuit

Diagram | Qasm | **Qiskit**

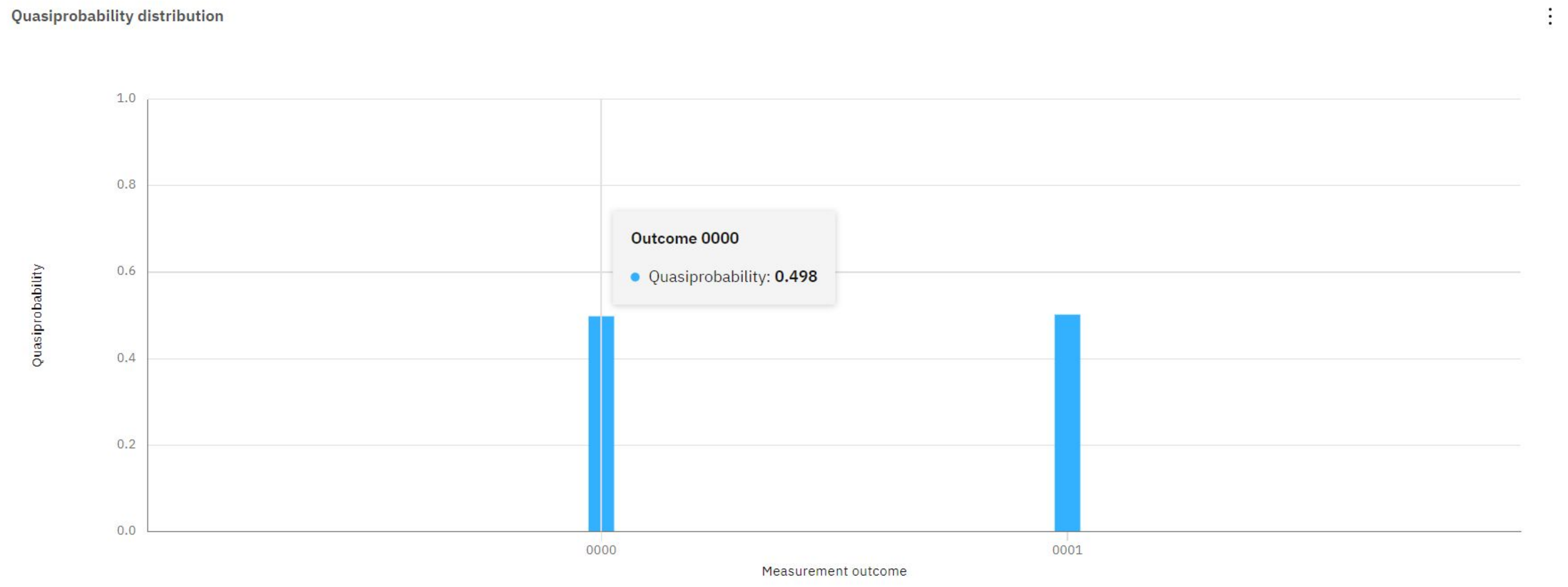
```
from qiskit import QuantumRegister, ClassicalRegister, QuantumCircuit
from numpy import pi

qreg_q = QuantumRegister(5, 'q')

circuit = QuantumCircuit(qreg_q)

circuit.h(qreg_q[0])
circuit.h(qreg_q[1])
circuit.h(qreg_q[2])
circuit.h(qreg_q[3])
circuit.h(qreg_q[4])
circuit.cx(qreg_q[0], qreg_q[1])
circuit.cx(qreg_q[2], qreg_q[3])
circuit.ry(0.1, qreg_q[0])
circuit.ry(0.1, qreg_q[1])
```

Open in quantum lab



Quasiprobability distribution
Outcome 0000
Quasiprobability: 0.498
1.0
0.8
0.6
0.4
0.2
0.0
Quasiprobability
0000
0001
Measurement outcome

# Using Qiskit Tools in Python

Job.status

- INITIALIZING = 'job is being initialized'
- QUEUED = 'job is queued'
- VALIDATING = 'job is being validated'
- RUNNING = 'job is actively running'
- CANCELLED = 'job has been cancelled'
- DONE = 'job has successfully run'
- ERROR = 'job incurred error'

# Job Monitor

From qiskit.tools import job_monitor

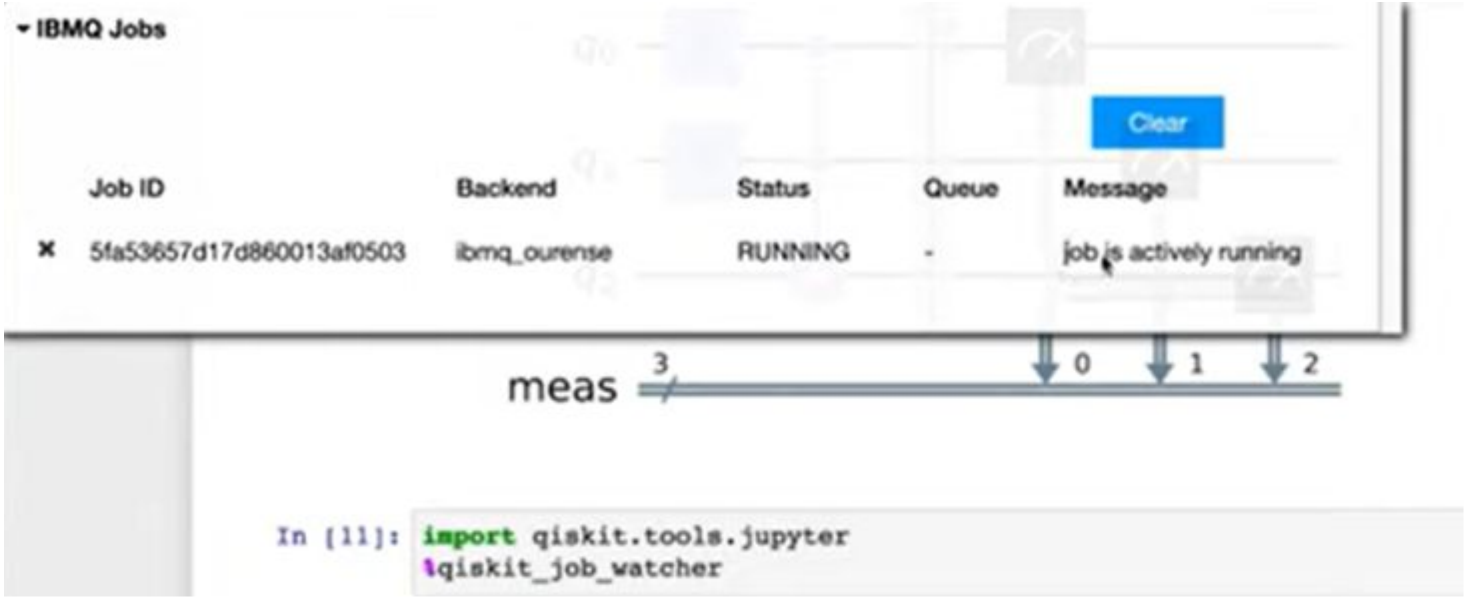
Job = execute(qc, ourense) #fake backend

job_monitor(job)

```
In [8]: from qiskit.tools import job_monitor
        job = execute(qc, ourense)
        job_monitor(job)

        Job Status: job has successfully run
```

# Qiskit Job Watcher



# Providers

A Provider is an entity that **provides access to a group of different backends** (for example, backends available through IBM Q). It interacts with those backends to, for example, find out which ones are available, or retrieve an instance of a particular backend.

A provider inherits from BaseProvider and implements the methods:

backends(): returns all backend objects known to the provider.
get_backend(name): returns the named backend.
Qiskit includes interfaces to two providers: Aer and IBMQ:

**Aer**: provides access to several simulators that are included with Qiskit and run on your local machine.

**IBMQ**: implements access to cloud-based backends — simulators and real quantum devices — hosted on IBM Q.

# Providers

## System providers

Access IBM Quantum and 3rd party systems and simulators to the following providers

### Qiskit IBM Runtime

IBM maintained · Provider · Partner

Run Qiskit primitives on IBM Quantum hardware with built-in error suppression and mitigation.

Go to repository ↗
Go to documentation ↗

### Qiskit IBM Provider

IBM maintained · Provider · Partner

Access IBM Quantum systems.

Go to repository ↗
Go to documentation ↗

### Qiskit IonQ Provider

Provider · Partner

This project contains a provider that allows access to IonQ ion trap quantum systems.

Go to repository ↗

### Qiskit AQT Provider

Provider

Qiskit provider for ion-trap quantum computers from Alpine Quantum Technologies (AQT).

Go to repository ↗
Go to website ↗
Go to documentation ↗

### Qiskit Cold Atom

Provider · Physics

Qiskit-cold-atom builds on core Qiskit functionalities to integrate programmable quantum simulators of trapped cold atoms in a gate- and circuit-based framework. The project includes a provider and simulators for fermionic and spin-based systems.

Go to repository ↗
Go to documentation ↗

### Qiskit Rigetti Provider

Provider

Rigetti Provider for Qiskit.

Go to repository ↗

### Qiskit Qulacs

Circuit simulator · Converter · Provider

Qiskit-Qulacs allows users to execute Qiskit programs using Qulacs backend.

Go to repository ↗
Go to documentation ↗

### qiskit-superstaq

Converter · Provider

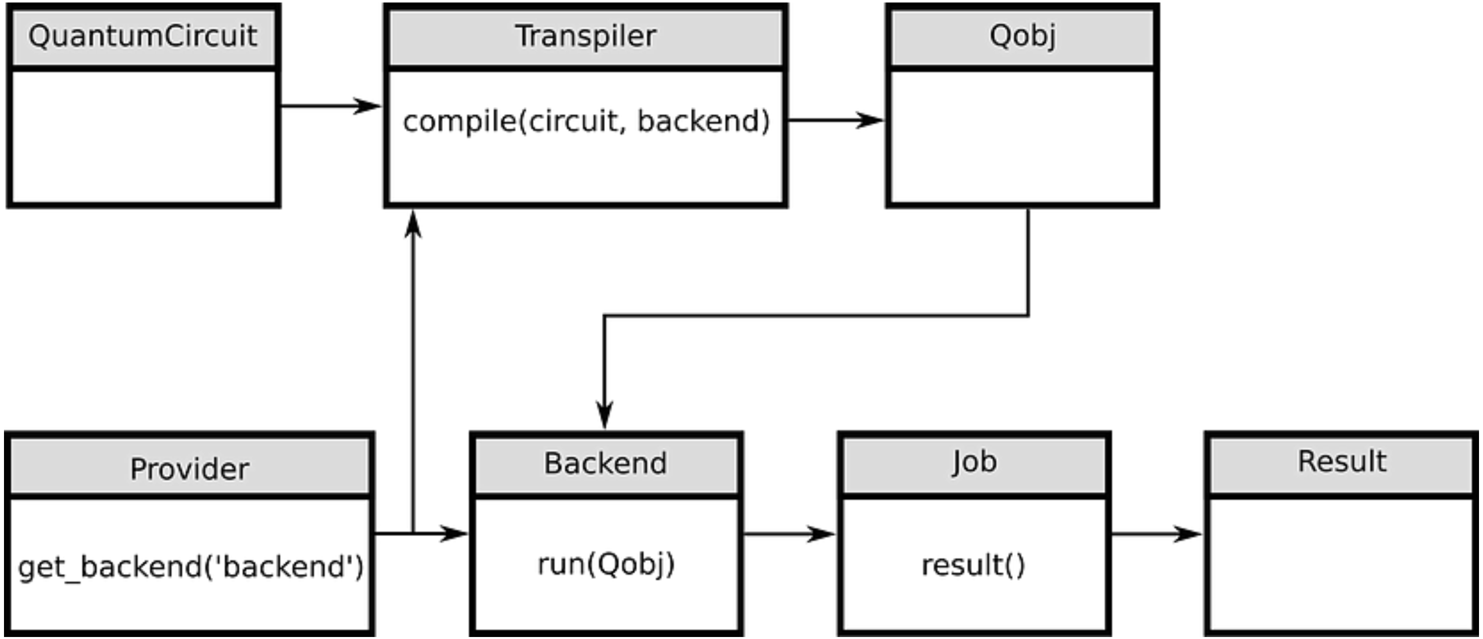
This package is used to access SuperstaQ via a Web API through Qiskit. Qiskit programmers can take advantage of the applications, pulse level optimizations, and write-once-target-all features of SuperstaQ with this package.

Go to repository ↗

# Backends

- Dependent on your group (main, internal, etc), you can access different backends (quantum computers)
- Backends represent either a simulator or a real quantum computer, and are responsible for running quantum circuits and returning results.
  - Take in a quantum objects as input
  - return a BaseJob object, allowing asynchronous running of jobs for retrieving results from a backend when the job is completed

# Backends Review



# Compute resources

Access IBM Quantum systems and simulators via our available access plans.
Resources you don't have access to will be denoted by a lock.

**Instance resources** | All systems | All simulators

You have access to the following resources with instance **ibm-q/open/main.**

Card | Table

Search by system or simulator name

Your systems & simulators (8)

ibm_**brisbane**

| | |
|---|---|
| System status | Online |
| Processor type | Eagle r3 |

| Qubits | EPLG | CLOPS |
|---|---|---|
| 127 | 1.9% | 5K |

ibm_**osaka**

| | |
|---|---|
| System status | Online |
| Processor type | Eagle r3 |

| Qubits | EPLG | CLOPS |
|---|---|---|
| 127 | 2.8% | 5K |

ibm_**kyoto**

| | |
|---|---|
| System status | Online |
| Processor type | Eagle r3 |

| Qubits | EPLG | CLOPS |
|---|---|---|
| 127 | 3.6% | 5K |

**simulator_stabilizer**

| | |
|---|---|
| Simulator status | Online |
| Simulator type | Clifford simulator |

Qubits
5000

**simulator_mps**

| | |
|---|---|
| Simulator status | Online |
| Simulator type | Matrix Product State |

Qubits
100

**simulator_extended_stabilizer**

| | |
|---|---|
| Simulator status | Online |
| Simulator type | Extended Clifford (e.g. Clifford+T) |

Qubits
63

ibmq_**qasm_simulator**

**simulator_statevector**

# Extra Info: Qiskit Ecosystem

https://qiskit.github.io/ecosystem/

# Section 8

*Display and use system information*

# Goals

Section 8: Display and Use System Information 3%

1. Perform operations around the Qiskit version
2. Use information gained from %*quiskit_backend_overview*

# Installing Qiskit

- Installing Qiskit takes many forms, but I will be using the most supported method from IBMQ themselves
- The first step is to install python. IBMQ recommends Jupyter, so that is what we will talk about later
  - Also, Qiskit doesn't require, but works best with a python virtual environment (see next page)

# Setting up the Virtual Environment

```
python3 -m venv c:\path\to\virtual\environment

c:\path\to\virtual\environment\Scripts\Activate.ps1
```

# Install packages

1. Install pip first to make things easier

```
py -m ensurepip --upgrade
```

2. Install other random things

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

# Determining Version

In [23]:

```
import qiskit

qiskit.__qiskit_version__
```

Out[23]:

```
{'qiskit-terra': '0.23.2', 'qiskit-aer': '0.1
1.2', 'qiskit-ignis': None, 'qiskit-ibmq-prov
ider': '0.20.1', 'qiskit': '0.41.1', 'qiskit-
nature': None, 'qiskit-finance': None, 'qiski
t-optimization': None, 'qiskit-machine-learni
ng': None}
```

In [24]:

```
from qiskit import __version__

__version__
```

Out[24]:

```
'0.23.2'
```

# Other Methods

```
import qiskit.tools.jupyter
%qiskit_version_table
%qiskit_copyright
```

Or

```
conda create -n qiskit_virtualenv python=3.8
conda activate qiskit_virtualenv
pip install qiskit
import qiskit
__qiskit_version__
```

Or

```
pip show qiskit
```

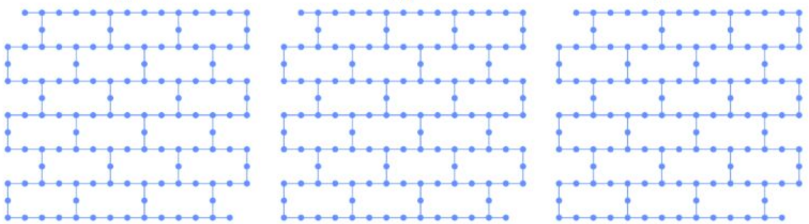
# %qiskit_backend_overview

```
[2]: from qiskit.providers.ibmq import IBMQ
     import qiskit.tools.jupyter
     %matplotlib inline

     IBMQ.load_account()

     %qiskit_backend_overview
```

**Backend Overview**



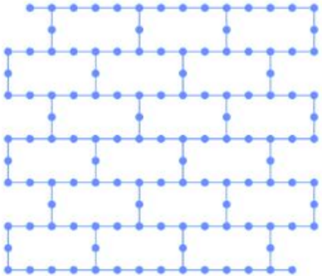
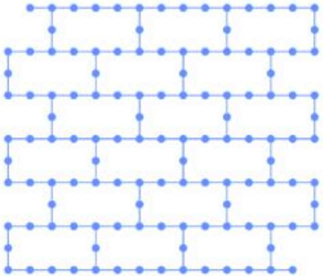
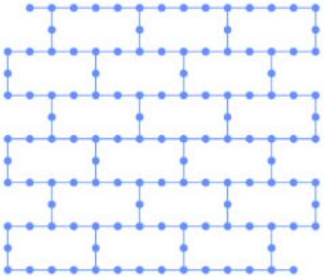
| | ibm_osaka | ibm_kyoto | ibm_brisbane |
|---|---|---|---|
| Num. Qubits | **127** | **127** | **127** |
| Quantum Vol. | - | - | - |
| Pending Jobs | 1 50 | 1109 1119 | 1050 1060 |
| Operational | True | True | True |
| Least Busy | True | False | False |
| Avg. T1 / T2 | 281.7 / 161.9 us | 215.9 / 125.4 us | 222.4 / 148.2 us |
| Avg. CX Err. | NA | NA | NA |
| Avg. Meas. Err. | 0.0564 | 0.0352 | 0.0242 |

```
[2]: from qiskit.providers.ibmq import IBMQ
import qiskit.tools.jupyter
%matplotlib inline

IBMQ.load_account()

%qiskit_backend_overview
```

## Backend Overview

| | ibm_osaka | ibm_kyoto | ibm_brisbane |
|---|---|---|---|
| |  |  |  |
| Num. Qubits | 127 | 127 | 127 |
| Quantum Vol. | - | - | - |
| Pending Jobs | 1 50 | 1109 1119 | 1050 1060 |
| Operational | True | True | True |
| Least Busy | True | False | False |
| Avg. T1 / T2 | 281.7 / 161.9 us | 215.9 / 125.4 us | 222.4 / 148.2 us |
| Avg. CX Err. | NA | NA | NA |
| Avg. Meas. Err. | 0.0564 | 0.0352 | 0.0242 |

# Section 10

*Access Aer Provider*

# The End

# Goals

1. Access a statevector_simulator backend
2. Access a qasm_simulator backend
3. Access a unitary_simulator backend

# Simulators

## Provider

| | |
|---|---|
| `BasicAerProvider` () | Provider for Basic Aer backends. |

## Job Class

| | |
|---|---|
| `BasicAerJob` (backend, job_id, result) | BasicAerJob class. |

## Exceptions

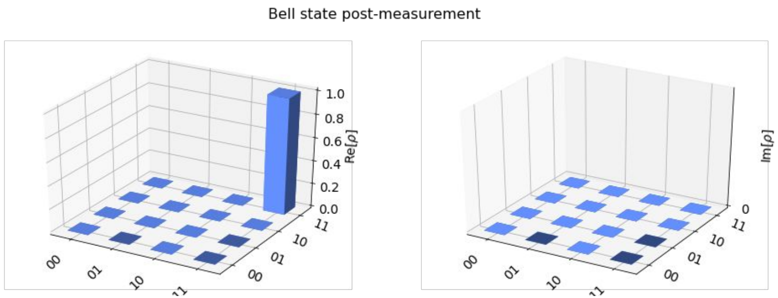
| | |
|---|---|
| `BasicAerError` (*message) | Base class for errors raised by Basic Aer. |

# Returning the state vector of an experiment

```
# Construct quantum circuit with measure
circ = QuantumCircuit(2, 2)
circ.h(0)
circ.cx(0, 1)
circ.measure([0,1], [0,1])

# Select the StatevectorSimulator from the Aer provider
simulator = Aer.get_backend('statevector_simulator')

# Execute and get counts
result = execute(circ, simulator).result()
statevector = result.get_statevector(circ)
plot_state_city(statevector, title='Bell state post-measurement')
```

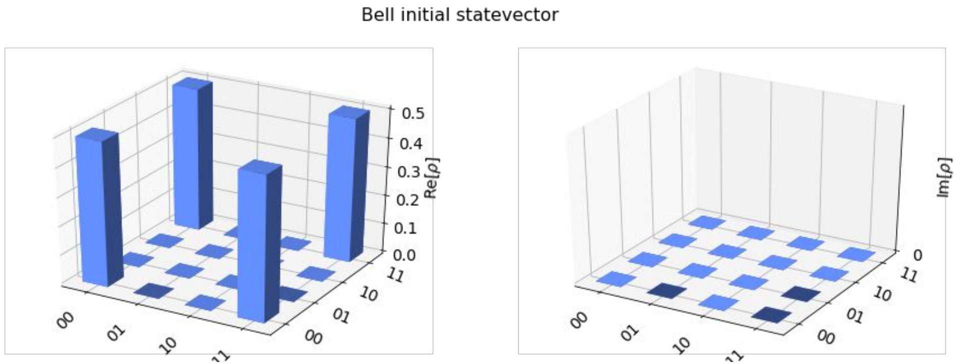


# Returning the state vector of an experiment part 2

```
# Construct a quantum circuit that initialises qubits to a custom state
circ = QuantumCircuit(2)
circ.initialize([1, 0, 0, 1] / np.sqrt(2), [0, 1])

# Select the StatevectorSimulator from the Aer provider
simulator = Aer.get_backend('statevector_simulator')

# Execute and get counts
result = execute(circ, simulator).result()
statevector = result.get_statevector(circ)
plot_state_city(statevector, title="Bell initial statevector")
```



# Returning the unitary of an experiment

The UnitarySimulator constructs the unitary matrix for a Qiskit Quantum Circuit by applying each gate matrix to an identity matrix. The circuit may only contain gates, if it contains resets or measure operations an exception will be raised.

# Returning the unitary of an experiment part 2

```
# Construct an empty quantum circuit
circ = QuantumCircuit(2)
circ.h(0)
circ.cx(0, 1)

# Select the UnitarySimulator from the Aer provider
simulator = Aer.get_backend('unitary_simulator')

# Execute and get counts
result = execute(circ, simulator).result()
unitary = result.get_unitary(circ)
print("Circuit unitary:\n", unitary)
```

```
Circuit unitary:
 [[ 0.70710678+0.00000000e+00j  0.70710678-8.65956056e-17j
   0.        +0.00000000e+00j  0.        +0.00000000e+00j]
 [ 0.        +0.00000000e+00j  0.        +0.00000000e+00j
   0.70710678+0.00000000e+00j -0.70710678+8.65956056e-17j]
 [ 0.        +0.00000000e+00j  0.        +0.00000000e+00j
   0.70710678+0.00000000e+00j  0.70710678-8.65956056e-17j]
 [ 0.70710678+0.00000000e+00j -0.70710678+8.65956056e-17j
   0.        +0.00000000e+00j  0.        +0.00000000e+00j]]
```

# Returning the unitary of an experiment part 3

```
# Construct an empty quantum circuit
circ = QuantumCircuit(2)
circ.id([0,1])

# Set the initial unitary
unitary1 = np.array([[ 1,  1,  0,  0],
                     [ 0,  0,  1, -1],
                     [ 0,  0,  1,  1],
                     [ 1, -1,  0,  0]] / np.sqrt(2))

# Select the UnitarySimulator from the Aer provider
simulator = Aer.get_backend('unitary_simulator')

# Execute and get counts
result = execute(circ, simulator, initial_unitary=unitary1).result()
unitary2 = result.get_unitary(circ)
print("Initial Unitary:\n", unitary2)

Initial Unitary:
 [[ 0.70710678+0.j  0.70710678+0.j  0.        +0.j  0.        +0.j]
 [ 0.        +0.j  0.        +0.j  0.70710678+0.j -0.70710678+0.j]
 [ 0.        +0.j  0.        +0.j  0.70710678+0.j  0.70710678+0.j]
 [ 0.70710678+0.j -0.70710678+0.j  0.        +0.j  0.        +0.j]]
```

# Available simulators

```
# List Aer backends
Aer.backends()

[QasmSimulator(
 backend_name='qasm_simulator', provider=AerProvider()),
 StatevectorSimulator(
 backend_name='statevector_simulator', provider=AerProvider()),
 UnitarySimulator(
 backend_name='unitary_simulator', provider=AerProvider()),
 PulseSimulator(
 backend_name='pulse_simulator', provider=AerProvider())]
```

Accessing a statevector_simulator backend

```
# Select the StatevectorSimulator from the Aer provider
simulator = Aer.get_backend('statevector_simulator')
```

Note: Remember to use

```
from qiskit import Aer
```

# Accessing a qasm_simulator backend

```
simulator = Aer.get_backend('qasm_simulator')
```

Accessing a unitary_simulator backend

```
simulator = Aer.get_backend('unitary_simulator')
```

Who’d have guessed

# Simulators

1. The **QASM Simulator** is the **main** Qiskit Aer **backend**. This backend **emulates execution of a quantum circuits on a real device and returns measurement counts**. It includes highly configurable noise models and can even be loaded with automatically generated approximate noise models based on the calibration parameters of actual hardware devices.

2. The Statevector Simulator is an auxiliary backend for Qiskit Aer. **It simulates the ideal execution of a quantum circuit and returns the final quantum state vector of the device at the end of simulation**. This is useful for education, as well as the **theoretical study** and debugging of algorithms.

3. The Unitary Simulator is another auxiliary backend for Qiskit Aer. It allows simulation of the final unitary matrix implemented by an **ideal quantum circuit**. This is also useful for education and algorithm studies.