CHAPTER 4

Enter IBM Quantum: A One-of-a-Kind Platform for Quantum Computing in the Cloud

In this chapter we take a look at quantum computing in the cloud with IBM Q: the first platform of its kind. The chapter starts with an overview of the composer, the web console used to visually create circuits, submit experiments, explore hardware devices, and more. Next, you will learn how to create your first experiment and submit it to the simulator or real quantum device. IBM Quantum features a powerful REST API to control the life cycle of the experiment, and this chapter will show you how with detailed descriptions of the endpoints and request parameters. Finally, the chapter ends with a series of exercises to put your REST API skills to the test. Let's get started.

IBM has certainly taken an early lead in the race for quantum computing in the cloud. They came up with a powerful platform to run experiments remotely called IBM Quantum. Let's see how this platform has the power to transform the status quo.

V. Silva, *Quantum Computing by Practice*, https://doi.org/10.1007/978-1-4842-9991-3_4

Getting Your Feet Wet with IBM Quantum

This is IBM's platform for quantum computing; let's take a look – follow the steps outlined here (All Reprints Courtesy of International Business Machines Corporation, © International Business Machines Corporation):

- Create an account in https://quantum-computing.ibm.com/. You will need an email; wait for the approval and confirm.
- Login to the web console and navigate to the composer tab by clicking the *Learning* menu from the application switcher in the top right of the main menu.

Quantum composer

The composer is the visual tool used to create your quantum circuits. At the top, it shows the experiment histogram with qubits available for use (see Figure 4-1).

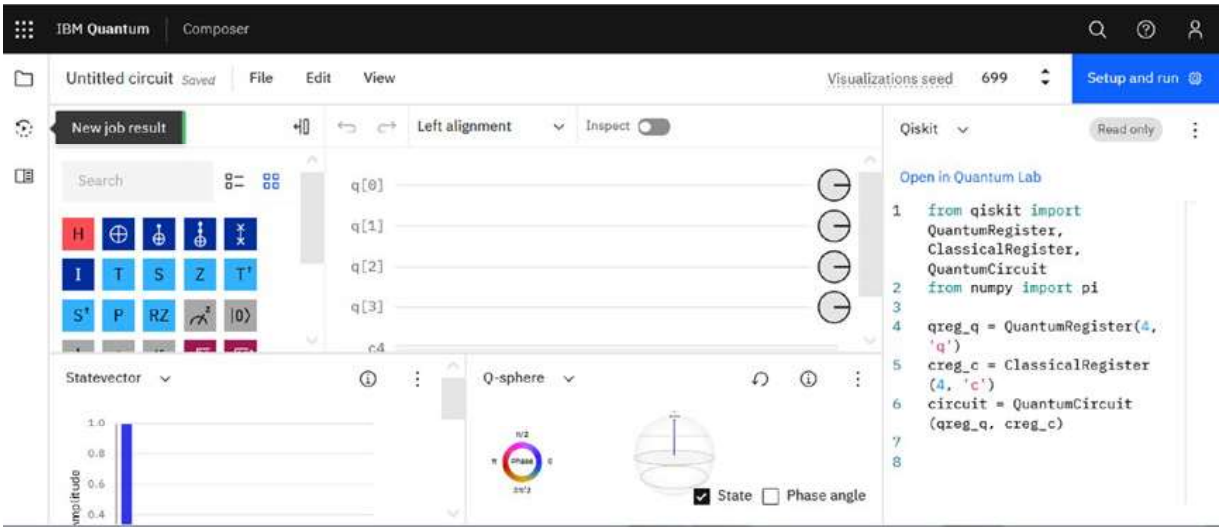


Figure 4-1. *Experiment histogram from the composer*

- On the right side, we see the histogram with 4 qubits available for use. They are all initialized to the ground state |0>. This is where you place your quantum gates.
- On the left side, we have the quantum gates. Drag gates into the histogram location of a specific qubit to start building a circuit.

Let's look at the gates and their meaning.

Quantum Gates

The quantum gates supported by IBM Quantum are described in Table 4-1.

Table 4-1. *Quantum gates for IBM Quantum*

Gate	Description
Pauli X 	It rotates the qubit 180 degrees in the X-axis. Maps |0> to |1> and |1> to |0>. Also known as the bit flip or NOT gate. It is represented by the matrix: $$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}$$
Pauli Y 	It rotates around the Y-axis of the Bloch sphere by π radians It is represented by the Pauli matrix: $$Y = \begin{bmatrix} 0 & -i \\ -i & 0 \end{bmatrix}$$ where $i = \sqrt{-1}$ is known as the imaginary unit
Pauli Z 	It rotates around the Z-axis of the Bloch sphere by π radians It is represented by the Pauli matrix: $$Z = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$
Hadamard 	It represents a rotation of π on the axis $(X + Z)/\sqrt{2}$. In other words, it maps the states: • |0> to $(\|0> + \|1>)/\sqrt{2}$ • |1> to $(\|0> - \|1>)/\sqrt{2}$ This gate is required to make superpositions
Phase $\sqrt{Z}$ 	It has the property that it maps X→Y and Z→Z. This gate extends H to make complex superpositions
Transposed Conjugate of S 	It maps X→-Y, and Z→Z

(*continued*)

Table 4-1. *(continued)*

Gate	Description
Controlled NOT (CNOT) 	This is a two qubit gate that flips the target qubit (applies Pauli X) if the control is in state 1. This gate is required to generate entanglement
Phase $\sqrt{S}$ 	The $\sqrt{S}$ gate performs a halfway rotation of a two-qubit swap. It is universal such that any gate can be constructed from only sqrt(swap) and single qubit gates. It is represented by the matrix: $\sqrt{S} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/2(1+i) & 1/2(1-i) & 0 \\ 0 & 1/2(1-i) & 1/2(1+i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Transposed Conjugate of T or Dagger-T 	Represented by the matrix: $\sqrt{S} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1/2(1-i) & 1/2(1+i) & 0 \\ 0 & 1/2(1+i) & 1/2(1-i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$
Barrier 	It prevents transformations across its source line
Measurement 	The measurement gate takes a qubit in a superposition of states as input and spits either a 0 or 1. There is a probability of a 0 or 1 as output depending on the original state of the qubit. Always remember that measurement should be the last thing done in the circuit
Conditional 	Conditionally apply a quantum operation
Identity	The identity gate performs an idle operation on the qubit for a time equal to one unit of time

You can drag gates from the left side of the composer to create a circuit, or if you prefer to write code, you can switch to the editor mode as shown in Figure 4-2.

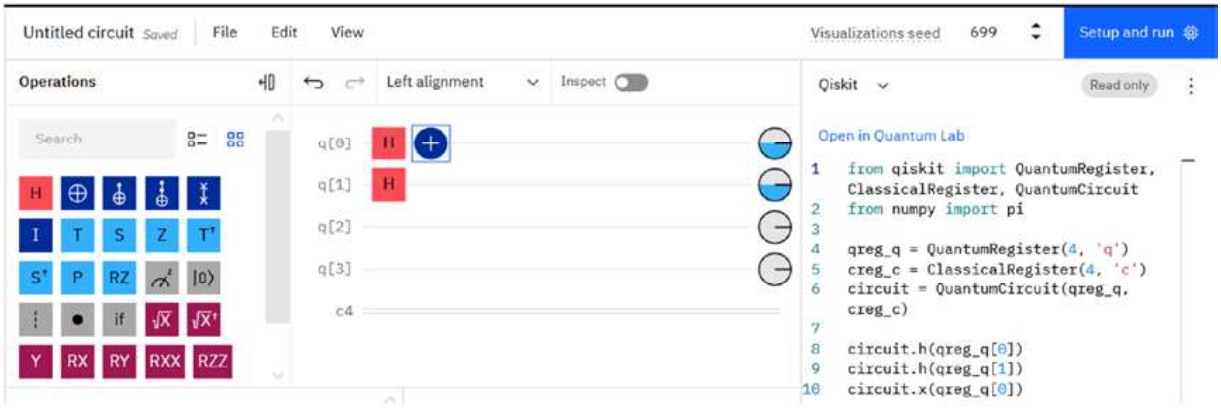


Figure 4-2. *Experiment editor in QASM editor mode*

Tip The editor gives you two language choices: Qiskit or QASM (Quantum Assembly). At the end, your circuits or Python code will be translated into QASM for submission into a real device.

Now let's take a look at the various quantum processors available for use.

Quantum Backends Available for Use

There are many quantum processors available for experimentation under the open (free) plan (from the main menu click Compute Resources). Table 4-2 shows a partial list ranked by the number of qubits according to the IBM Quantum backend information site.[1]

[1] IBM Quantum backend information available at https://quantum-computing.ibm.com/lab/docs/iql/manage/account/ibmq

Table 4-2. *Partial list of quantum backends available for IBM Quantum users under the open plan*

Name	Details
ibm_perth	Processor: Falcon r5.11H Qubits: 7 Quantum Volume: 32
ibm_lagos	Processor: Falcon r5.11H Qubits: 7 Quantum Volume: 32
ibm_nairobi	Processor: Falcon r5.11H Qubits: 7 Quantum Volume: 32

Here is a little secret: there is a very interesting way to get an updated list of available machines in real time using the excellent REST API. This API is described in more detail in the "Remote Access" section in this chapter, but for now let's demonstrate how to obtain an always up-to-date list of backends using the *Available Backend List* REST endpoint: `https://api-qcon.quantum-computing.ibm.com/api/Backends?access_token=ACCESS-TOKEN`.

Tip To obtain an access token, see the section "Authentication via API Token" under "Remote Access" of this chapter. Note that an API token is not the same as an access token. API tokens are used for authentication. Access tokens are used to invoke operations using the REST API.

The URL in the previous paragraph returns a list of quantum processors in JSON format. This is what it looks like by the time of this writing. Note that your results may be different:

Listing 4-1. HTTP response from the backend information REST API call


```
[{
  "name": " ibm_lagos",
  "version": "1",
  "status": "on",
  "serialNumber": "Real5Qv2",
  "description": "5 transmon bowtie",
  "basisGates": "u1,u2,u3,cx,id",
  "onlineDate": "2017-01-10T12:00:00.000Z",
  "chipName": "Sparrow",
  "id": "28147a578bdc88ec8087af46ede526e1",
  "topologyId": "250e969c6b9e68aa2a045ffbceb3ac33",
  "url": "https://ibm.biz/qiskit-ibmqx2",
  "simulator": false,
  "nQubits": 7,
  "couplingMap": [
    [0, 1],
    [0, 2],
    [1, 2],
    [3, 2],
    [3, 4],
    [4, 2]
  ]
}, {
  "name": "ibm_nairobi ",
  "version": "1",
  "status": "on",
  "serialNumber": "ibmqx5",
  "description": "16 transmon 2x8 ladder",
  "basisGates": "u1,u2,u3,cx,id",
  "onlineDate": "2017-09-21T11:00:00.000Z",
  "chipName": "Albatross",
  "id": "f451527ae7b9c9998e7addf1067c0df4",
  "topologyId": "ad8b182a0653f51dfbd5d66c33fd08c7",
  "url": "https://ibm.biz/qiskit-ibmqx5",
```

```
  "simulator": false,
  "nQubits": 7,
  "couplingMap": [
    [1, 0],
    ...
    [15, 14]
  ]
},
{
  "name": "ibmqx_hpc_qasm_simulator",
  "status": "on",
  "serialNumber": "hpc-simulator",
  "basisGates": "u1,u2,u3,cx,id",
  "onlineDate": "2017-12-09T12:00:00.000Z",
  "id": "084e8de73c4d16330550c34cf97de3f2",
  "topologyId": "7ca1eda6c4bff274c38d1fe66c449dff",
  "simulator": true,
  "nQubits": 32,
  "couplingMap": "all-to-all"
},
{
  "name": "ibmqx_qasm_simulator",
  "status": "on",
  "description": "online qasm simulator",
  "basisGates": "u1,u2,u3,cx,id",
  "id": "18da019106bf6b5a55e0ef932763a670",
  "topologyId": "250e969c6b9e68aa2a045ffbceb3ac33",
  "simulator": true,
  "nQubits": 24,
  "couplingMap": "all-to-all"
}]
```

Listing 4-1 shows that there is a lot of extra interesting information about the structural layout of these machines:

- Extra processors and simulators:
  - There are a few remote simulators available for use (ibmq_qasm_simualtor, simulator_mps, ibmq_qasm_simualtor). This information can come in handy when testing complex circuits: more simulators are always a good thing.
  - Rumors of a 475 qubit processor have been swirling around for some time. There is even talk of an upcoming 1000 qubit processor by the end of 2023, but don't get excited just yet; this machine is only available for corporate customers. IBM has put an ambitious road map to reach a million qubits in the future. Check it out at `www.ibm.com/quantum/roadmap`.
- Besides the usual information such as machine name, version, status, number of qubits, and others. There are some terms we should be familiarized with:
  - basisGates: These are the physical qubit gates of the processor. They are the foundation under which more complex logical gates can be constructed. Most of the processors in the list use u1,u2,u3,cx,id.
    - Gates u1, u2, u3 are called *partial NOT gates* and perform rotations on axes X, Y, Z by theta, phi, or lambda radians of a qubit.
    - Cx is called the *controlled NOT* gate (CNOT or CX). It acts on 2 qubits and performs the NOT operation on the second qubit only when the first qubit is |1>, and otherwise leaves it unchanged.
    - Id is the identity gate that performs an idle operation on a qubit for one unit of time.

- couplingMap: The coupling map defines interactions between individual qubits while retaining quantum coherence. Qubit coupling is important, to simplify quantum circuitry and allow the system to be broken up into smaller units.

Now back to the composer for our first quantum circuit.

Entanglement: Bell and GHZ States

Entanglement experiments are used to demonstrate the weirdness of quantum mechanics and come in two flavors:

- Bell states: They demonstrate that physics is not described by local reality. This is what Einstein called *spooky action at a distance* (2-qubit entanglement).
- GHZ states: Even stranger than Bell states (named after their creators: Greenberger-Horne-Zeilinger), they describe 3 qubit entanglement.

Let's look at them in more detail.

Two Qubit Entanglement with Bell States

Bell states are the experimental test of the famous Bell inequalities. In 1964 Irish Physicist John Bell proposed a way to put quantum entanglement (spooky action at a distance) to the test. He came up with a set of inequalities that have become incredibly important in the physics community. This set of inequalities is known as Bell's theorem and it goes something like this.

Consider photon polarization (when light oscillates in a specific plane) at three different angles A = 0, B = 120, C = 240. Realism says that a photon has definite simultaneous values for these three polarization settings, and they must correspond to the eight cases shown in Table 4-3.

Table 4-3. *Permutations for photon polarizations at three angles*

Count	A(0)	B(120)	C(240)	[AB]	[BC]	[AC]	Sum	Average
1	A+	B+	C+	1(++)	1(++)	1(++)	3	1
2	A+	B+	C-	1(++)	0	0	1	1/3
3	A+	B-	C+	0	0	1(++)	1	1/3
4	A+	B-	C-	0	1(--)	0	1	1/3
5	A-	B+	C+	0	1(++)	0	1	1/3
6	A-	B+	C-	0	0	1(--)	1	1/3
7	A-	B-	C+	1(--)	0	0	1	1/3
8	A-	B-	C-	1(--)	1(--)	1(--)	3	1

Now Bell's theorem asks: *What is the probability that the polarization at any neighbor will be the same as the first?* We also calculate the sum and average of the polarization. Assuming realism is true, then by looking at Table 4-1, the answer to the question must be the probability must be >= 1/3. This is what the Bell inequality gives: A means to put this assertion to the test. Here is the incredible part: Believe it or not quantum mechanics violates Bell's inequality giving probabilities less than 1/3. This was proven experimentally for the first time in 1982 by French physicist Alain Aspect.

So now let's translate the photon polarization above into an experiment that can be run in a quantum computer. In 1969 John Clauser, Michael Horne, Abner Shimony, and Richard Holt came up with a proof for Bell's theorem: The CHSH inequality which formally states:

$$S = \langle A, B \rangle - \langle A, B' \rangle + \langle A', B \rangle + \langle A', B' \rangle$$

$$S \leq 2$$

To illustrate this we have two detectors: Alice and Bob. Given A and A′ are detector settings on side Alice, B and B′ on side Bob, with the four combinations being tested in separate experiments. Realism says that for a pair of entangled particles the parity table showing all possible permutations looks as shown here:

A	B	1
A	B'	0
A'	B	0
A'	B'	1

In classical realism, the CHSH inequality becomes |S| = 2. However, the mathematical formalism of quantum mechanics predicts a maximum value for S of $|S|=2\sqrt{2}$, thus violating this inequality. This can be put to the test using four separate quantum circuits (1 per measurement) with 2 qubits each. To simplify things, let measurements on Alice's detector be A = Z, A' = X, and Bob's detector B = W, B' = V (see Table 4-2). To begin the experiment, a basis Bell state must be constructed which matches the identity (see Figure 4-5):

$$1/\sqrt{2}\left(|00\rangle+|11\rangle\right)$$

The preceding expression essentially means: The qubit held by Alice can be 0 or 1. If Alice measured her qubit in the standard basis, the outcome would be perfectly random, either possibility having probability 1/2. But if Bob then measured his qubit, the outcome would be the same as the one Alice got. So, if Bob measured, he would also get a random outcome on first sight, but if Alice and Bob communicated they would find out that, although the outcomes seemed random, they are correlated.

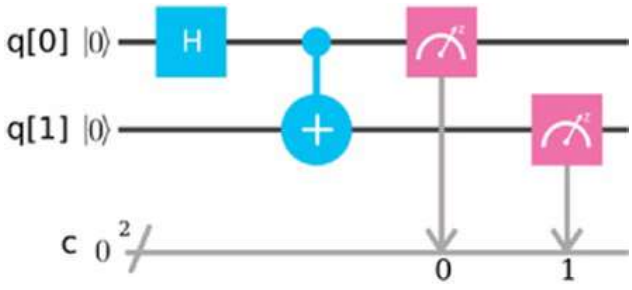


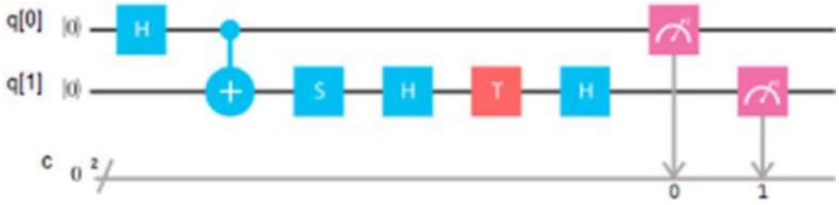
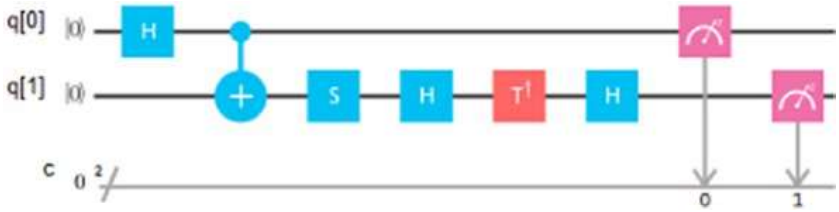
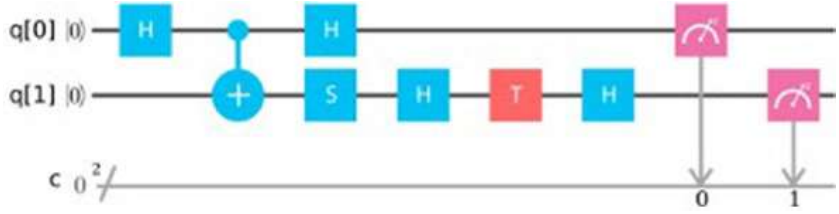
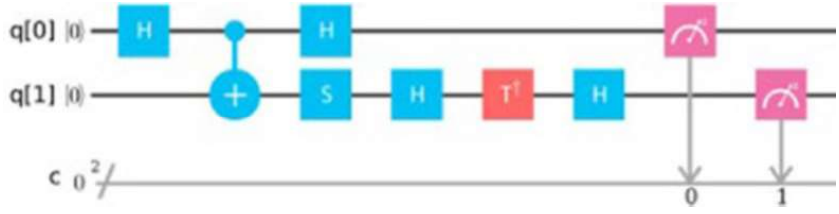
Figure 4-3. *Basis Bell state*

In Figure 4-3 two qubits are prepared in the ground state |0>. The H gate creates a superposition of the first qubit to the state $1/\sqrt{2}\left(|00\rangle+|10\rangle\right)$. Next, the CNOT gate flips the second qubit if the first is excited, resulting in the state $1/\sqrt{2}\left(|00\rangle+|11\rangle\right)$. This is the initial entangled state required for the four measurements in Table 4-2 (All reprints courtesy of International Business Machines Corporation, © International Business Machines Corporation).

- To rotate the measurement basis to the ZW axis, use the sequence of gates S-H-T-H.
- To rotate the measurement basis to the ZV axis, use the sequence of gates S-H-T'-H.
- The XW and XV measurement is performed the same way as above and the X via a Hadamard gate before a standard measurement.

Tip Before performing the experiment in the composer, make sure its topology (the number of qubits and target device) in the score is set to two over a simulator. Some topologies (like the 5 qubit in a real quantum device) do not support entanglement for qubits 0 and 1 giving errors at design. Note that the target device can be a real quantum processor or a simulator.

Table 4-4. *Quantum circuits for Bell states*

Bell state measurement	Result for 100 shots	
AB (ZW)		
	c[2]	Probability
	11	0.39
	10	0.06
	00	0.46
	01	0.09
AB' (ZV)		
	c[2]	Probability
	11	0.49
	10	0.07
	00	0.36
	01	0.08
A'B (XW)		
	c[2]	Probability
	11	0.42
	10	0.05
	00	0.49
	01	0.04
A'B' (XV)		
	c[2]	Probability
	11	0.05
	10	0.52
	00	0.03
	01	0.40

Now we need to construct a table with the results of each measurement from Table 4-4 plus the correlation probability between A and B <AB>. The sum of the probabilities for the parity of the entangled particles is given by:

$$\langle AB \rangle = P(1,1) + P(0,0) - P(1,0) - P(0,1)$$

Remember that the ultimate goal is to determine if $S \leq 2$ or $|S| = 2$; thus, by compiling the results of all measurements, we obtain Table 4-5.

Table 4-5. *Compiled results from the Bell experiment*

	P(00)	P(11)	P(01)	P(10)	<AB>
AB (ZW)	0.46	0.39	0.09	0.06	0.68
AB' (ZV)	0.36	0.49	0.08	0.07	0.73
A'B (XW)	0.49	0.42	0.04	0.05	0.47
A'B'(XV)	0.03	0.05	0.4	0.52	-0.32

Add the absolute values of column <AB> and we obtain $|S| = 2.2$. These results violate the Bell inequality (as predicted by quantum mechanics).

Three Qubit Entanglement with GHZ States Tests

These are named after physicists Greenberger-Horne-Zeilinger who came up with a generalization test for N entangled qubits with the simplest being a 3 qubit GHZ state:

$$|GHZ\rangle = 1/\sqrt{2}\left(|000\rangle - |111\rangle\right)$$

Note The importance of the GHZ states is that they show that the entanglement of more than two particles conflicts with local realism not only for statistical (probabilistic) but also nonstatistical (deterministic) predictions.

In simple terms, GHZ states show a stronger violation of Bell's inequality. Let's see how with a simple puzzle: imagine three independent boxes each containing two variables X, Y. Each variable has two possible outcomes: 1, -1. The question is to find a set of values for X, Y that solves the following set of identities:

```
(1) XYY = 1
(2) YXY = 1
(3) YYX = 1
(4) XXX = -1
```

For the impatient out there, there is no solution for this. For example, replace Y = 1 in (1) (2) and (3) then multiply them, that is, (5) = (1) * (2) * (3). The set then becomes:

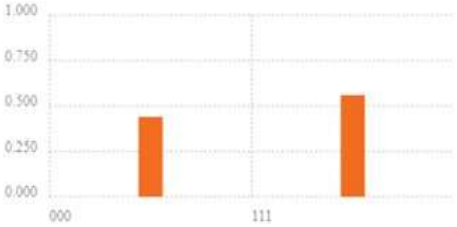
```
(1) X11 = 1
(2) 1X1 = 1
(3) 11X = 1
(4) XXX = -1
(5) Multiply (1) (2) (3) and we get XXX = 1
```

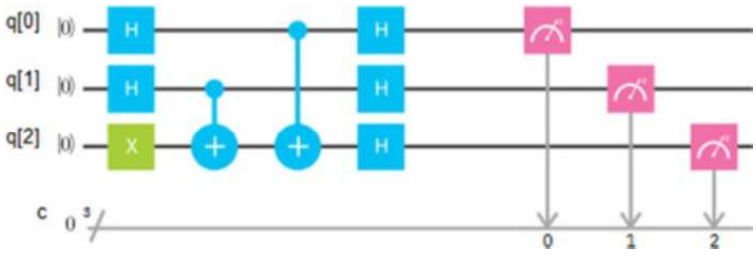
There is no solution because identity (4) XXX = -1 contradicts identity (5) XXX = 1. The scary part is that a GHZ state can indeed provide a solution to this problem. This seems impossible in the deterministic view of classical reality, but nothing is impossible in the world of quantum mechanics, just improbable.

Incredibly, GHZ tests can rule out the local reality description with certainty after a single run of the experiment, but first we must construct a GHZ basis state. Try building the circuit in Table 4-6 in the composer to see the probabilities match the left side chart.

Table 4-6. *Basis GHZ state*

$$|GHZ\rangle = 1/\sqrt{2}\left(|000\rangle - |111\rangle\right)$$



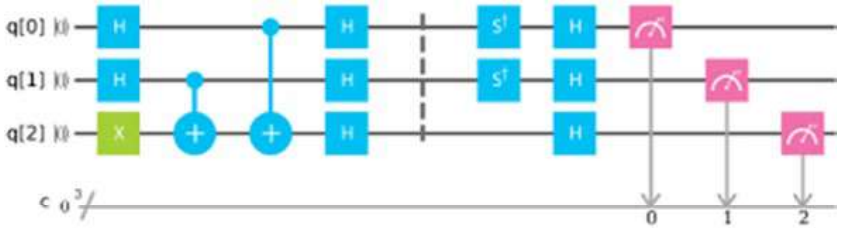
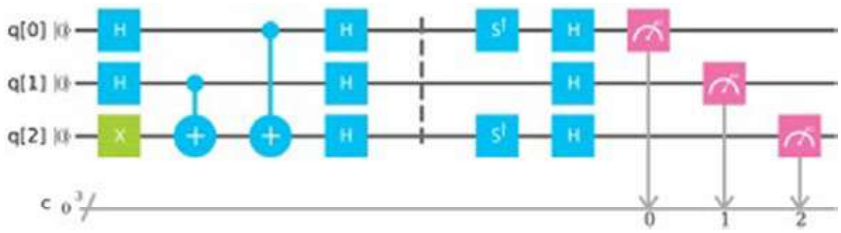


To kick start the experiment, the basis GHZ state (as well as probability results which should be around half) is shown in Table 4-6:

1. In the basis circuit, Hadamard gates in qubits 1 and 2 put them in superposition |00,01,01,11>. At the same time, the X gate negates qubit 3; thus, we end up with the states $1/\sqrt{2}\left(|001\rangle+|101\rangle+|011\rangle+|111\rangle\right)$.
2. The two CNOT gates entangle all qubits into the state $1/\sqrt{2}\left(|001\rangle+|010\rangle+|100\rangle+|111\rangle\right)$.
3. Finally, the three Hadamard gates map step 2 to the state ½(|000⟩ − |111⟩).

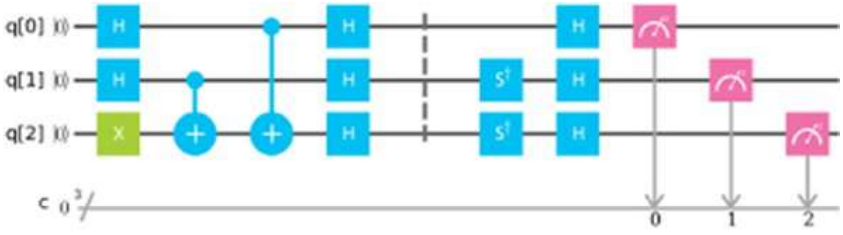
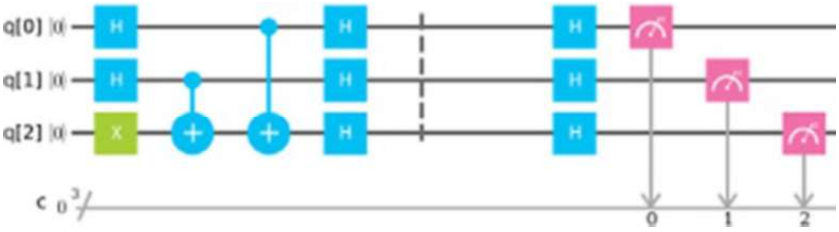
Now, create the quantum circuits for identities XYY, YXY, XYY, and XXX from the previous section as shown in Table 4-7 (All reprints courtesy of International Business Machines Corporation, © International Business Machines Corporation).

Table 4-7. *Quantum circuits for GHZ states*

Measurement	Results for 100 shots	
YYX		
	c[3]	Probability
	011	0.34
	101	0.23
	110	0.23
	000	0.20
YXY		
	c[3]	Probability
	011	0.23
	101	0.28
	110	0.25
	000	0.24

(*continued*)

Table 4-7. (*continued*)

Measurement	Results for 100 shots	
XYY		
	c[3]	Probability
	011	0.23
	101	0.26
	110	0.35
	000	0.16
XXX		
	c[3]	Probability
	010	0.25
	100	0.32
	111	0.22
	001	0.21

- For the measurement of X apply the H gate to the corresponding qubit.
- For each instance of Y apply the S† (S-dagger), and H gates to the corresponding qubit.

All in all, the principles of entanglement shown in this section were not popular in the early years of the 20th century. As a matter of fact, they were challenged by a theory called super determinism which sought to give a way out.

Super Determinism: A Way Out of the Spookiness. Was Einstein Right All Along?

In an interview for BBC in 1969, Physicist John Bell talked about his work on quantum mechanics. He said that we must accept the predictions that actions are transferred faster than the speed of light between entangled particles, but at the same time we cannot do anything with it. Information cannot travel faster than the speed of light, a

fact that is also predicted by quantum mechanics. As if nature is playing a trick on us. He also mentioned that there is a way out of this riddle through a principle called super determinism.

Particle entanglement implies that measurements performed in one particle affect the other instantaneously, even across large distances (think opposite sides of the galaxy or the universe), and even across time. Einstein was an ardent opponent of this theory famously writing to Neils Bohr *God does not throw dice*. He could not accept the probabilistic nature of quantum mechanics so in 1935, along with colleagues Podolsky and Rosen, they came up with the infamous EPR paradox to challenge its foundation. In the EPR paradox if two entangled particles are separated by a tremendous distance, a measurement in one could not affect the other instantaneously as the event will have to travel faster than the speed of light (the ultimate speed limit in the universe). This will violate general relativity, thus creating a paradox: nothing travels faster than the speed of light that is the absolute rule of relativity.

Nevertheless, in 1982 the predictions of quantum mechanics were confirmed by French Physicist Alain Aspect. He devised an experiment that showed Bell's inequality is violated by entangled photons. He also proved that a measurement in one of the entangled photons travels faster than the speed of light to signal its state to the other. Since then, Aspect's results have been proven correct time and again (details on his experiment are shown in Chapter 1). The irony is that there is a chance that Einstein was right all along and entanglement is just an illusion. It is the principle of super determinism.

Tip In simple terms super determinism says that freedom of choice has been removed since the beginning of the universe. All particle correlations and entanglements were established at the moment of the big bang. Thus, there is no need for a faster than light signal to tell particle B what the outcome of particle A is.

If true, this loophole will prove that Einstein was right when postulating the EPR paradox and all our hard work thus far is just an illusion. But this principle sounds more like religious dogma (all outcomes are determined by fate) than science as Bell argued that super determinism was implausible. His reason was that freedom of choice is effectively free for the purpose at hand due to alterations introduced by a large number

of very small effects. Super determinism has been called untestable as experimenters would never be able to eliminate correlations that were created at the beginning of the universe. Nevertheless, this hasn't stopped scientists from trying to prove Einstein right and particle entanglement an illusion. As a matter of fact, there is an experiment hard at work to settle things up and is really inventive. Let's see how.

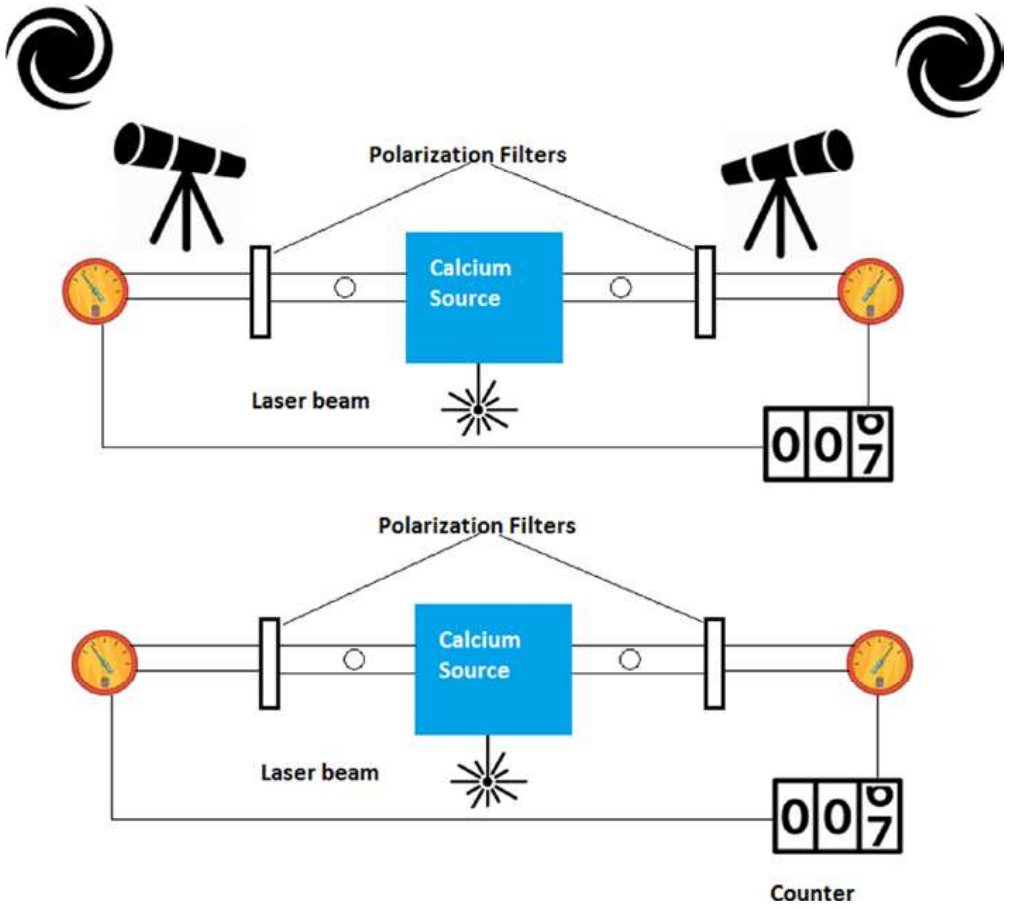


Figure 4-4. *Bell's inequality experiment using cosmic photons vs. the standard test*

Figure 4-4 shows the standard Bell inequality test experiment (at the bottom) and a variation of the experiment using cosmic photons (at the top) by Andrew Friedman and colleagues at MIT.[2]

[2] Jason Gallicchio, Andrew S. Friedman, and David I. Kaiser. Testing Bell's Inequality with Cosmic Photons: Closing the Setting-Independence Loophole. available online at https://arxiv.org/abs/1310.3288

Tip For a full description of the standard Bell inequality test, see Chapter 1.

Friedman and colleagues came up with a novel variation of the standard Bell experiment using cosmic rays. The idea is to use real-time astronomical observations of distant stars in our galaxy, distant quasars, or patches of the cosmic microwave background, to essentially let the universe decide how to set up the experiment instead of using a standard quantum random number generator. That is, photons from distant galaxies are used to control the orientation of the polarization filters just before the arrival of entangled photons.

If successful, the implications would be groundbreaking. If the results from such an experiment do not violate Bell's inequality, it would mean that super determinism could be true after all. Particle entanglement will be an illusion, and signal transfer between entangled particles could not travel faster than light as predicted by relativity. Einstein will be right and there is no spooky action at a distance.

Luckily for quantum mechanics, no such thing has happened so far. Keep in mind that Friedman and colleagues are not the only team getting into the action. There are multiple teams trying to crack this riddle. As a matter of fact most of their results agree with quantum mechanics. That is, their results violate Bell's inequality. So it seems that the rift created by Einstein and Bohr in their struggle between relativity and quantum mechanics has not been settled yet. The next section shows how IBM Quantum can be accessed remotely via its slick REST API.

Remote Access via the REST API

Quantum features a relatively unknown REST API that handles all remote communications behind the scenes. It is used by the current Python SDKs:

- Qiskit: The Quantum Information Software Kit is the de facto access tool for quantum programming in Python.
- QiskitIBMQProvider: A lesser known library bundled with Qiskit that wraps the REST API in a Python client.

In this section, we peek inside IBMQprovider and look at the different REST endpoints for remote access. But first, authentication is required.

Authentication

To invoke any REST API call, we must first obtain an access token. This will be the access key to invoke any of the calls in this section. There are two ways of obtaining this token:

- Using your API token: To obtain the API token, login to the IBM Q console and copy it from the dashboard (see Figure 4-5).
- Using your account user name and password: Let's see how this is done using REST.

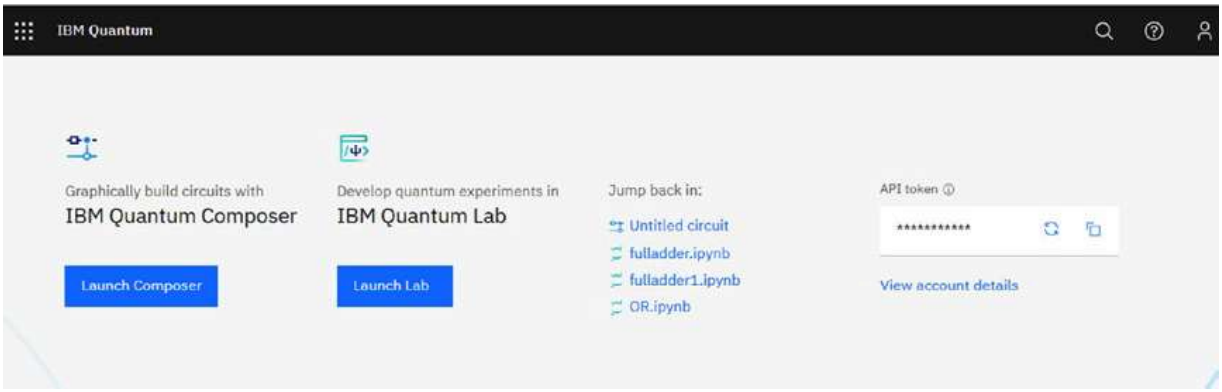


Figure 4-5. *Obtain your API token from the console.*

Authentication via API Token

- **HTTP method:** POST
- **URL:** `https://auth.quantum-computing.ibm.com/api/users/loginWithToken`
- **Payload:** {"apiToken": "YOUR_API_TOKEN"}

Authentication via User-Password

- **HTTP method:** POST
- **URL:** `https://auth.quantum-computing.ibm.com/api/users/login`
- **Payload:** {"email": "USER-NAME", "password": "YOUR-PASSWORD"}

The response for both methods is:

```
{
  "id": "ACCESS_TOKEN",
  "ttl": 1209600,
  "created": "2018-04-15T20:21:03.204Z",
  "userId": "USER-ID"
}
```

where *id* is your access token, *ttl* is the time to live (or expiration time) in milliseconds, and *userId* is your user id. Save the access token and the user id for use in this section. Note that when your session expires, a new access token needs to be generated.

List Available Backends

This call returns a JSON list of all available backends and simulators in IBM Q:

- **HTTP method:** GET
- **URL:** https://api-qcon.quantum-computing.ibm.com/api/Backends?access_token=ACESS-TOKEN

Request Parameters

Name	Value
access_token	Your account access token

HTTP Headers

Name	Value
x-qx-client-application	Defaults to qiskit-api-py

Response Sample

The response content type for all API calls is application/json. The next paragraph shows the partial result of a call to this endpoint. Note that this endpoint will return both real processors and simulators.


```
[{
      "name": "ibmqx2",
      "version": "1",
      "status": "on",
      "serialNumber": "Real5Qv2",
      "description": "5 transmon bowtie",
      "basisGates": "u1,u2,u3,cx,id",
      "onlineDate": "2017-01-10T12:00:00.000Z",
      "chipName": "Sparrow",
      "id": "28147a578bdc88ec8087af46ede526e1",
      "topologyId": "250e969c6b9e68aa2a045ffbceb3ac33",
      "url": "https://ibm.biz/qiskit-ibmqx2",
      "simulator": false,
      "nQubits": 5,
      "couplingMap": [
            [0, 1],
            [0, 2],
            [1, 2],
            [3, 2],
            [3, 4],
            [4, 2]
      ]
},..]
```


The most important keys from the response above are described in Table 4-8.

Table 4-8. *Available backends response keys*

Key	Description
Name	The name id of the processor to be used when executing code against it
Version	A string or positive integer used to track changes to the processor
Description	This is probably a description of the hardware used to build the chip. You may see things like • 5 transmon bowtie • 16 transmon 2x8 ladder Note: A trasmon is defined as a type of noise-resistant superconducting charge qubit. It was developed by Robert J. Schoelkopf, Michel Devoret, Steven M. Girvin, and their colleagues at Yale University in 2007[3]
basisGates	These are the physical qubit gates of the processor. They are the foundation under which more complex logical gates can be constructed
nQubits	The number of qubits used by the processor
copulingMap	The coupling map defines interactions between individual qubits while retaining quantum coherence. It is used to simplify quantum circuitry and allow the system to be broken up into smaller units

Get Backend Parameters

This call returns a JSON list of the backend parameters for a given processor in Q Experience. Some of these parameters include

- Qubit cool down temperature in Kelvin degrees: For example, I got 0.021 K for ibmqx4 – that is a super frosty -459.6 F or -273.1 C.
- Buffer times in ns.

[3] J. Koch et al., "Charge-insensitive qubit design derived from the Cooper pair box," Phys. Rev. A 76, 04319 (2007), doi:10.1103/PhysRevA.76.042319, arXiv:0703002

- Gate times in ns.
- Other quantum specs are documented in more detail at the backend information site.[4]

The request type and endpoint URL are

- **HTTP method:** GET
- **URL:** https://api-qcon.quantum-computing.ibm.com/api/Backends/NAME/properties?access_token=ACCESS-TOKEN

Request Parameters

Name	Value
access_token	Your account access token

HTTP Headers

Name	Value
x-qx-client-application	Defaults to qiskit-api-py

Response Sample

Listing 4-2 shows a simplified response for ibmqx4 parameters in JSON.

Listing 4-2. Simplified response for ibmqx4 parameters

```
{
      "lastUpdateDate": "2018-04-15T10:47:03.000Z",
      "fridgeParameters": {
            "cooldownDate": "2017-09-07",
            "Temperature": {
                  "date": "2018-04-15T10:47:03Z",
```

[4] IBM Quantum backend information available online at https://github.com/QISKit/ibmqx-backend-information

```
                "value": 0.021,
                "unit": "K"
        }
    },
    "qubits": [{
        "name": "Q0",
        "buffer": {
                "date": "2018-04-15T10:47:03Z",
                "value": 10,
                "unit": "ns"
        },
        "gateTime": {
                "date": "2018-04-15T10:47:03Z",
                "value": 50,
                "unit": "ns"
        },
        "T2": {
                "date": "2018-04-15T10:47:03Z",
                "value": 16.5,
                "unit": "µs"
        },
        "T1": {
                "date": "2018-04-15T10:47:03Z",
                "value": 45.2,
                "unit": "µs"
        },
        "frequency": {
                "date": "2018-04-15T10:47:03Z",
                "value": 5.24208,
                "unit": "GHz"
        }
    },..]
```

Get the Status of a Processor's Queue

This call returns the status of a specific quantum processor event queue.

- **HTTP method:** GET
- **URL:** `https://api-qcon.quantum-computing.ibm.com/api/Backends/NAME/queue/status?access_token=ACCESS-TOKEN`

Request Parameters

It seems strange but this API call appears not to ask for an access token.

HTTP Headers

Name	Value
x-qx-client-application	Defaults to qiskit-api-py

Response Sample

For example, to get the event queue for ibmqx4, paste the following URL in your browser:

`https://quantumexperience.ng.bluemix.net/api/Backends/ibmqx4/queue/status`

The response looks like: `{"state":true,"status":"active","lengthQueue":0}` where

- **state:** It is the status of the processor. If alive true else false.
- **status:** It is the status of the execution queue: active or busy.
- **lengthQueue:** It is the size of the execution queue or the number of simulations waiting to be executed.

Tip When you submit an experiment to IBM Q Experience, it will enter an execution queue. This API call is useful to monitor how busy the processor is at a given time.

List Jobs in the Execution Queue

This call returns a list of jobs in the processor execution queue.

- **HTTP method:** GET
- **URL:** https://api-qcon.quantum-computing.ibm.com/api/jobs?access_token=ACCESS-TOKEN&filter=FILTER

Request Parameters

Name	Value
access_token	Your account access token
filter	A result size hint in JSON. For example: {"limit":2} returns a maximum of 2 entries

HTTP Headers

Name	Value
x-qx-client-application	Defaults to qiskit-api-py

Response Sample

Listing 4-3 shows the response format for this call. The information appears to be a historical record of experiment executions containing information such as status, dates, results, code, calibration, and more.

Listing 4-3. Simplified response for the get jobs API call

```
[{
      "qasms": [{
            "qasm": "...",
            "status": "DONE",
            "executionId": "331f15a5eed1a4f72aa2fb4d96c75380",
            "result": {
                  "date": "2018-04-05T14:25:37.948Z",
                  "data": {
```

```
                        "creg_labels": "c[5]",
                        "additionalData": {
                              "seed": 348582688
                        },
                        "time": 0.0166247,
                        "counts": {
                              "11100": 754,
                              "01100": 270
                        }
                  }
            }
      }],
      "shots": 1024,
      "backend": {
            "name": "ibmqx_qasm_simulator"
      },
      "status": "COMPLETED",
      "maxCredits": 3,
      "usedCredits": 0,
      "creationDate": "2018-04-05T14:25:37.597Z",
      "deleted": false,
      "id": "d405c5829274d0ee49b190205796df87",
      "userId": "ef072577bd26831c59ddb212467821db",
      "calibration": {}
}, ...]
```

Note Depending on the size of the execution queue, you may get an empty result ([]) if there are no jobs in the queue or a formal result as shown in Listing 4-3. Whatever the case, make sure the HTTP response code is 200 (OK).

Get Account Information

When an account is created, each user is assigned several execution credits which are spent when running experiments. This call lists your credit information.

- **HTTP method:** GET
- **URL:** `https://auth.quantum-computing.ibm.com/api/users/USER-ID?access_token=ACCESS-TOKEN`

Tip The user id can be obtained from the authentication response via API token or user-password. See the "Authentication" section for details.

Request Parameters

Name	Value
access_token	Your account access token

HTTP Headers

Name	Value
x-qx-client-application	Defaults to qiskit-api-py

Response Sample

Listing 4-4 shows a sample response for this call.

Listing 4-4. Credit information sample response

```
{
      "institution": "Private Research",
      "status": "Registered",
      "blocked": "None",
      "dpl": {
```

```
            "blocked": false,
            "checked": false,
            "wordsFound": {},
            "results": {}
      },
      "credit": {
            "promotional": 0,
            "remaining": 150,
            "promotionalCodesUsed": [],
            "lastRefill": "2018-04-12T14:05:09.136Z",
            "maxUserType": 150
      },
      "additionalData": {
      },
      "creationDate": "2018-04-01T15:36:16.344Z",
      "username": "",
      "email": "",
      "emailVerified": true,
      "id": "",
      "userTypeId": "…",
      "firstName": "…",
      "lastName": "…"
}
```


List User's Experiments

This call lists all experiments for a given user id.

- **HTTP method:** GET
- **URL:** https://api-qcon.quantum-computing.ibm.com/api/users/USER-ID/codes/lastest?access_token=ACCESS-TOKEN&includeExecutions=true

Request Parameters

Name	Value
USER-ID	Your user id obtained from the authentication step
access_token	Your account access token
includeExecutions	If true, include executions in the result

HTTP Headers

Name	Value
x-qx-client-application	Defaults to qiskit-api-py

Response Sample

Listing 4-5 shows a sample response from this call.

Listing 4-5. Experiment list response

```
{
  "total": 17,
  "count": 17,
  "codes": [{
    "type": "Algorithm",
    "active": true,
    "versionId": 1,
    "idCode": "…",
    "name": "3Q GHZ State YXY-Measurement 1",
    "jsonQASM": {
      ...
    },
    "qasm": "",
    "codeType": "QASM2",
    "creationDate": "2018-04-14T19:09:51.382Z",
    "deleted": false,
```

```
    "orderDate": 1523733740504,
    "userDeleted": false,
    "displayUrls": {
      "png": "URL"
    },
    "isPublic": false,
    "id": "…",
    "userId": "…"
  }]}
```

Run a Job on Hardware

Use this call to submit a Quantum Job to a hardware processor available in your customer plan.

- **HTTP method:** POST
- **URL:** https://runtime-us-east.quantum-computing.ibm.com/jobs

HTTP Headers

Name	Value
X-Access-Token	Access token
Content-Type	application/json

Payload Format

The format of the payload embeds all execution parameters: backend name, shots, and code in a single JSON document as shown in the following.

```
{
  "program_id": "circuit-runner",
  "hub": "ibm-q",
  "group": "open",
  "project": "main",
  "backend": "ibmq_quito",
```

```
  "params": {
    "shots": 1024,
    "circuits": [
        " OPENQASM 2.0;\ninclude \"qelib1.inc\";\n\nqreg q[4];\ncreg c[4];\
        nh q[0];\nh q[1];\nx q[0];\nmeasure q[0] -> c[0];"
    ]
  },
  "tags": [
    "composer-info:composer:true",
    "composer-info:code-id:d2262dd4e07a9f894caabd977b10c7e98d90537228723dd1
    4dceaab023c36098",
    "composer-info:code-version-id:646d2a856260a179aa094d3e"
  ]
}
```

The preceding payload submits a random experiment to the real device ibm_quito. Make sure it is online before submission (or use the simulator instead). Also make sure the QASM code is in a single line including line feeds (\n). Note that double quotes must be escaped. If the submission fails, it probably means that the device is offline or your QASM payload is invalid.

Response Format

If everything goes OK, the response will look like:

```
{
    "id": "chn19jpike34bjj7d400",
    "backend": "ibmq_quito"
}
```

The id value represents the Job id. You can monitor its status in the Jobs menu of the dashboard (see Figure 4-6).

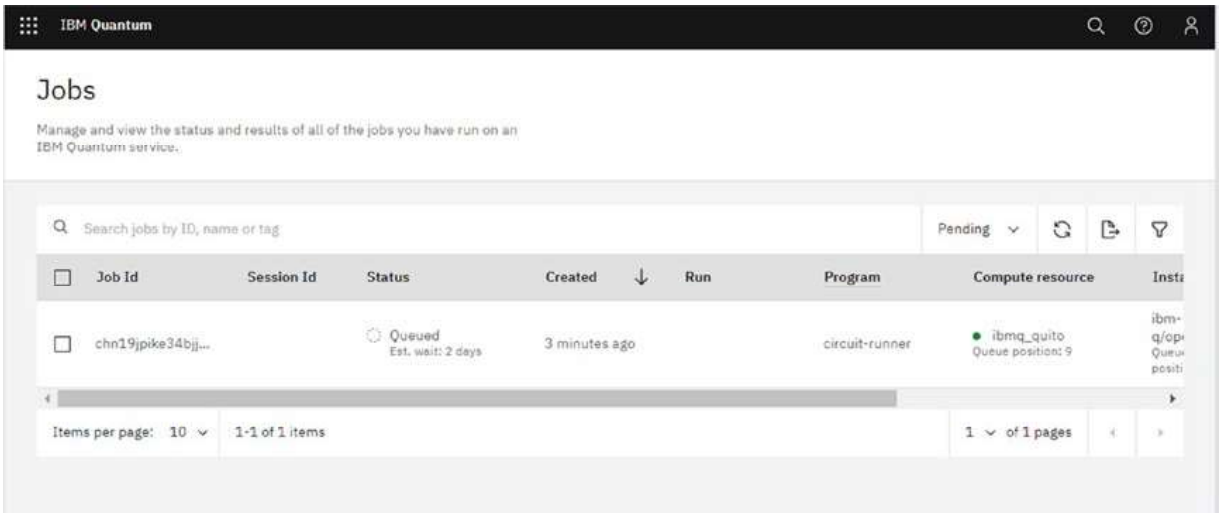


Figure 4-6. *Job status in the web console*

Note Depending on the size of the execution queue, your job may sit for a while. Please keep the queue clean by canceling your test jobs.

Get the API Version

It returns the version of the Q Experience REST API.

- **HTTP method:** GET
- **URL:** `https://api-qcon.quantum-computing.ibm.com/api/version?access_token=ACCESS-TOKEN`

Request Parameters

Name	Value
access_token	Your account access token

HTTP Headers

Name	Value
x-qx-client-application	Defaults to qiskit-api-py

Response Format

It returns a string with the version of the API, by the time of this writing 0.153.0.

We have peaked inside the IBM Quantum REST API to see what goes on behind the scenes. Now let's put that knowledge to the test with a simple set of exercises.

Exercises

The IBM Quantum REST API can be a powerful tool to add functionality to your client apps. In this exercise set, we will use a platform desktop client to set up a REST workspace for the API calls we have seen in this chapter.

1. Install the Postman desktop client from www.postman.com/. Hint: When you start Postman, do not sign in to the cloud. This will allow you to work locally.
2. Create a new collection for the IBM Quantum REST API. Name the workspace *IBM Quantum*. **Tip: Read about the very useful Postman Global Variables to quickly store information such as API or access tokens.** See https://learning.postman.com/docs/sending-requests/variables/.
3. Create a new request: Authenticate via token. Use the following parameters:
   a. Type: POST
   b. URL: from the authentication section of this chapter: https://auth.quantum-computing.ibm.com/api/users/loginWithToken

   c. Payload: {"apiToken": "YOUR_API_TOKEN"}. Tip: Copy the API token from the dashboard of the IBM Q console. Verify the Response returns an id (access token). **Please note that the API token (used for authentication) is not the same as the access token (used to invoke operations).** *Tip: Store the access token in a Postman global variable for use in further operations.*

4. Create a new request: Authenticate via password. Use the following parameters:

   a. Type: POST

   b. URL: from the authentication section of this chapter: `https://auth.quantum-computing.ibm.com/api/users/login`

   c. Payload: {"email": "USER-NAME", "password": "YOUR-PASSWORD"}. Verify the Response returns an id (access token).

5. Create a GET request to fetch the list of available backends:

   a. Type: GET

   b. URL: `https://api-qcon.quantum-computing.ibm.com/api/Backends?access_token=ACCESS_TOKEN`

   c. Verify the response information. Tip: Use the access token from Exercise 3. **Do not use the API token from the dashboard.**

6. Create a GET request to fetch processor parameters (use processor *ibm_perth* or your favorite name from the console):

   a. Type: GET

   b. URL: `https://api-qcon.quantum-computing.ibm.com/api/Backends/ibm_perth/properties?access_token=ACCESS_TOKEN`

   c. Verify the response JSON. Name your request Get Backend properties. Tip: Use a Global variable to store the processor name.

7. Create a GET request to fetch the queue status of the previous processor:

   a. URL: `https://api-qcon.quantum-computing.ibm.com/api/Backends/ibm_perth/queue/status?access_token=ACCESS_TOKEN`

   b. Type: GET

   c. Verify the JSON response.

8. Create a GET request to List Jobs in the execution queue:

   a. Name: List Jobs

   b. Type: GET

   c. URL: `https://api-qcon.quantum-computing.ibm.com/api/jobs?access_token=ACCESS_TOKEN`

   d. Verify the JSON response.

9. Create a GET request to List user experiments:

   a. Name: List Experiments

   b. Type: GET

   c. URL: `https://api-qcon.quantum-computing.ibm.com//api/users/USER_ID/codes/lastest?access_token=ACCESS_TOKEN` (Tip: The user id is returned in the authentication request).

   d. Verify the JSON response.

10. Create a GET request to get the API version:

    a. Type: GET

    b. Name: Get API Version

    c. URL: `https://api-qcon.quantum-computing.ibm.com/api/version?access_token=ACCESS_TOKEN`

    d. Verify the JSON response.

Finally, export your collection. This will be a helpful tool if you plan to integrate the IBM Quantum REST API into your existing desktop or web app. Your Postman workspace should look similar to Figure 4-7.

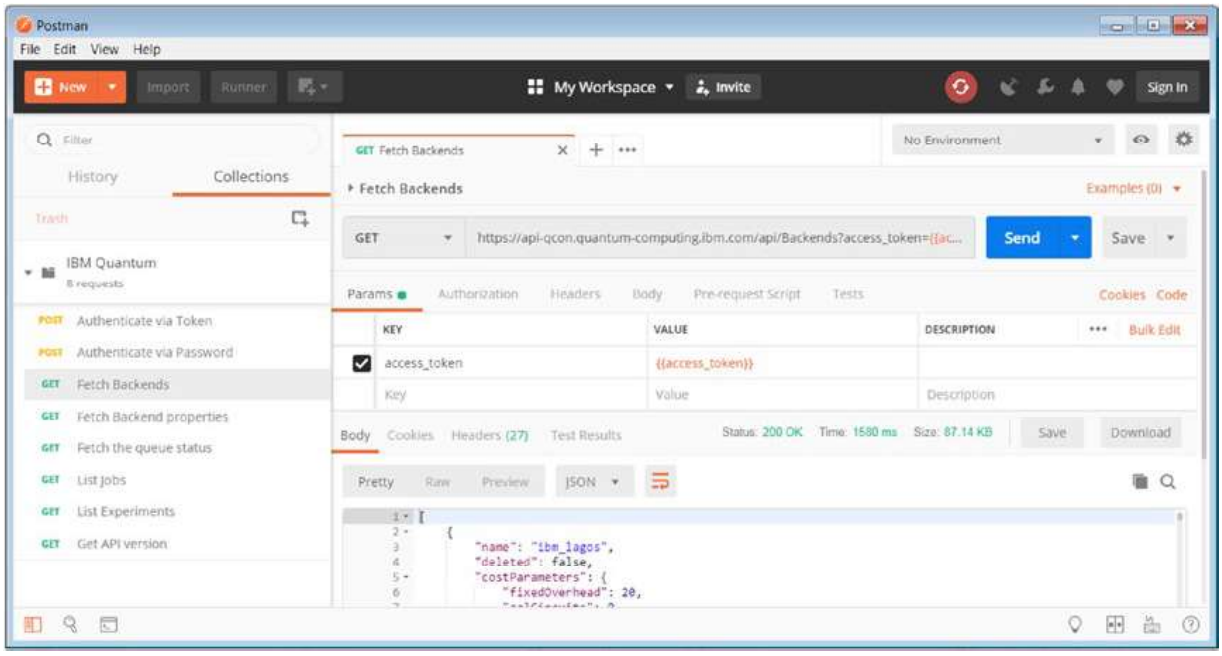


Figure 4-7. *Postman desktop app with the Quantum REST API collection*