Librerías

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import scipy as sp
import csv
import qiskit
from qiskit import *
from qiskit.tools.monitor import job_monitor
from qiskit.quantum_info import random_statevector
from IPython.display import clear_output

from qiskit import QuantumCircuit, QuantumRegister, ClassicalRegister
from qiskit import IBMQ, Aer, transpile, assemble
from qiskit.visualization import plot_histogram, plot_bloch_multivector, array_to_latex
from qiskit.extensions import Initialize
from qiskit.ignis.verification import marginal_counts
from qiskit.quantum_info import random_statevector
```

IBM Account

In [2]:
```
IBMQ.load_account()
```

Out[2]:
```
<AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>
```

Computadores

In [3]:
```
provider = IBMQ.get_provider('ibm-q')

# Simuladores
aer_sim = Aer.get_backend('aer_simulator')

# Computadores
armonk = provider.get_backend('ibmq_armonk')
quito = provider.get_backend('ibmq_quito')
lima = provider.get_backend('ibmq_lima')
```

Simulación

Ejecución del circuito en un simulador de IBM, posteriormente se realiza el experimento en un computador real.

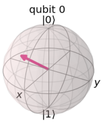
Estado Aleatorio

Se genera un estado aleatorio $|\psi\rangle$ para transportar desde el qubit q_0 a q_2.

In [8]:
```
psi = random_statevector(2)

display(array_to_latex(psi, prefix="|\\psi\\rangle ="))

plot_bloch_multivector(psi)
```

$$|\psi\rangle = \left[-0.75868 + 0.48774i \quad -0.23365 + 0.3632i \right]$$

Out[8]:

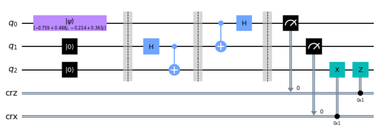


Circuito

Se configura el circuito de teleportación.

In [9]:
```
qr = QuantumRegister(3, name = 'q')
crz = ClassicalRegister(1, name = 'crz')
crx = ClassicalRegister(1, name = 'crx')

qc = QuantumCircuit(qr, crz, crx)

qc.initialize(psi, qr[0])
qc.reset(qr[1])
qc.reset(qr[2])
qc.barrier()

qc.h(qr[1])
qc.cx(qr[1], qr[2])
qc.barrier()

qc.cx(qr[0], qr[1])
qc.h(qr[0])
qc.barrier()

qc.measure(qr[0], crz)
qc.measure(qr[1], crx)

qc.x(qr[2]).c_if(crx, 1)
qc.z(qr[2]).c_if(crz, 1)

qc.draw(output = 'mpl')
```

Out[9]:

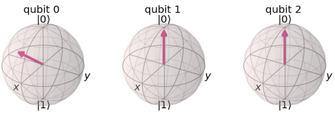


Estado Inicial

Los estados iniciales de los qubits del sistema.

In [10]:
```
qrInit = QuantumRegister(3, name = 'q')
crInit = ClassicalRegister(3, name = 'c')
qcInit = QuantumCircuit(qrInit, crInit)

qcInit.initialize(psi, qrInit[0])
qcInit.reset(qrInit[1])
qcInit.reset(qrInit[2])


qcInit.save_statevector()
out_vector = aer_sim.run(qcInit).result().get_statevector()
plot_bloch_multivector(out_vector)
```

Out[10]:

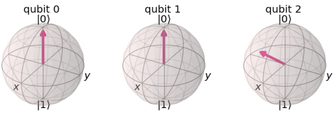


Estado Final

Los estados finales de los qubits del sistema, donde q_2 ahora contiene el estado inicial de q_0 como se esperaba.

In [11]:
```
qc.save_statevector()
out_vector = aer_sim.run(qc).result().get_statevector()
plot_bloch_multivector(out_vector)
```

Out[11]:



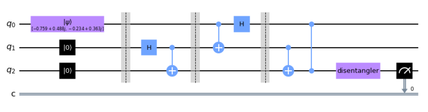
Experimento

Circuito

Se configura el circuito de teleportación para correr en un computador real de IBM. En este caso, no es posible continuar operando luego de hacer mediciones en un qubit, por lo cual las compuertas Z y X se sustituyen por compuertas controladas por los qubits q_0 y q_1 respectivamente. Finalmente, se aplica en q_2 una compuerta que invierte la inicialización del estado aleatorio en q_0, es decir que se espera que el resultado de la medición sea exactamente $|0\rangle$.

In [13]:
```
qr = QuantumRegister(3, name = 'q')
cr = ClassicalRegister(1, name = 'c')

qc = QuantumCircuit(qr, cr)

qc.initialize(psi, qr[0])
qc.reset(qr[1])
qc.reset([2])
qc.barrier()

qc.h(qr[1])
qc.cx(qr[1], qr[2])
qc.barrier()

qc.cx(qr[0], qr[1])
qc.h(qr[0])
qc.barrier()

qc.cx(qr[1], qr[2])
qc.cz(qr[0], qr[2])

init_gate = Initialize(psi)
inverse_init_gate = init_gate.gates_to_uncompute()

qc.append(inverse_init_gate, [2])

qc.measure(qr[2], cr)

qc.draw(output = 'mpl')
```

Out[13]:



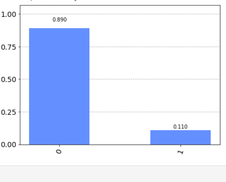
Resultados

Idealmente se obtendría una probabilidad de 1 de encontrar a q_2 en el estado $|0\rangle$, los resultados muestran una probabilidad alta pero que contiene una baja probabilidad debido a errores de hallar a este qubit en el estado $|1\rangle$

In [15]:
```
job = execute(qc, backend = quito, shots = 10000)
job_monitor(job)

exp_result = job.result()
exp_counts = exp_result.get_counts(qc)
print(exp_counts)
plot_histogram(exp_counts)
```

```
Job Status: job has successfully run
{'0': 8902, '1': 1098}
```

Out[15]:



In [ ]:
```
```