Amazon Braket

Building quantum software in the cloud

Eric Kessler

Sr. Manager Applied Science
Amazon Web Services

« Quantum Technologies

Amazon Braket

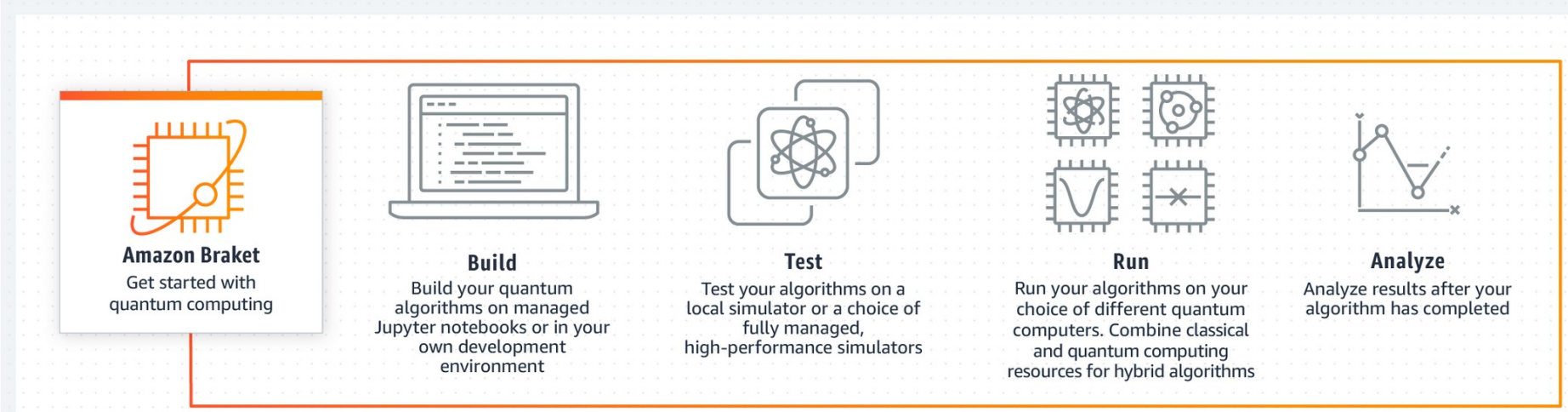
Accelerate quantum computing research

Get Started with Amazon Braket

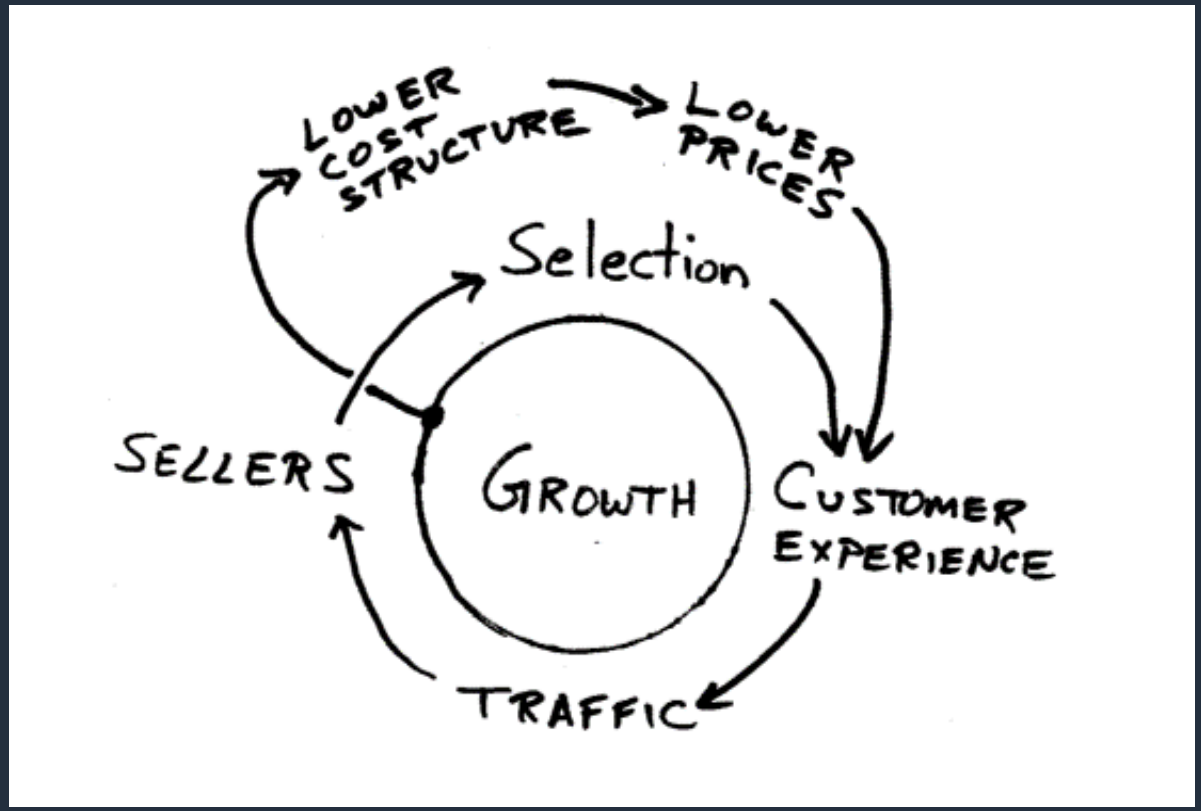
1 free hour of simulation time per month
with AWS Free Tier

How it works

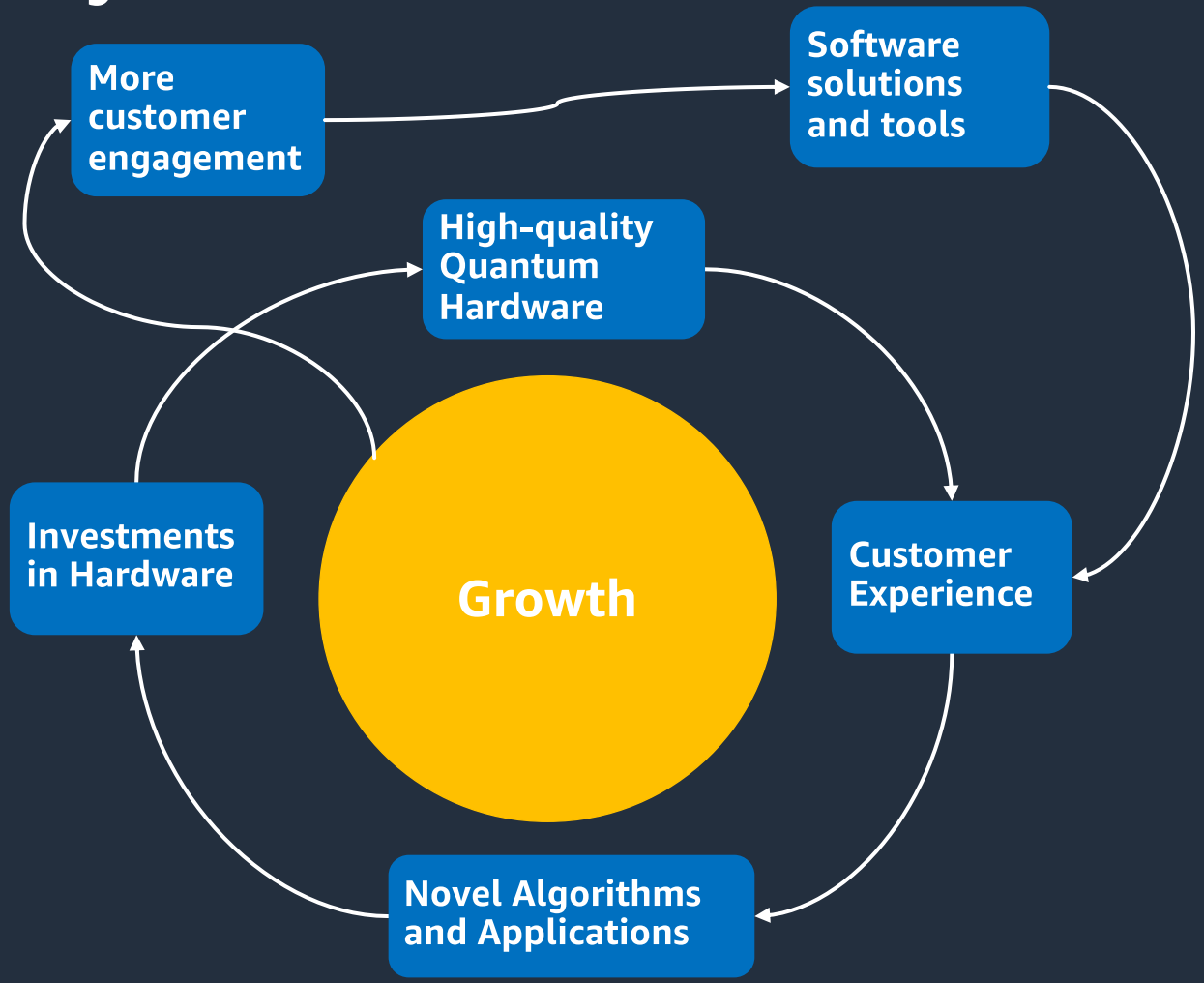
Amazon Braket is a fully managed quantum computing service designed to help speed up scientific research and software development for quantum computing.



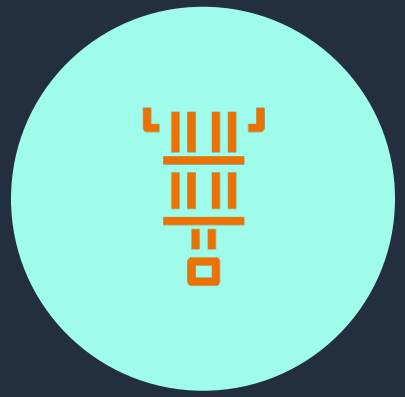
The Amazon Flywheel



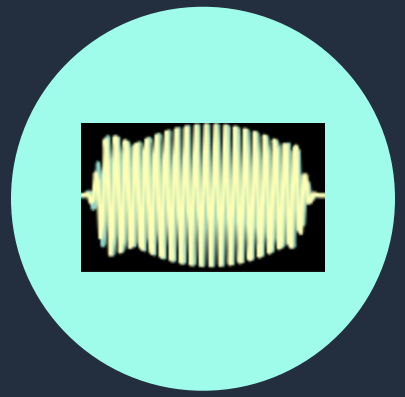
The Quantum Flywheel



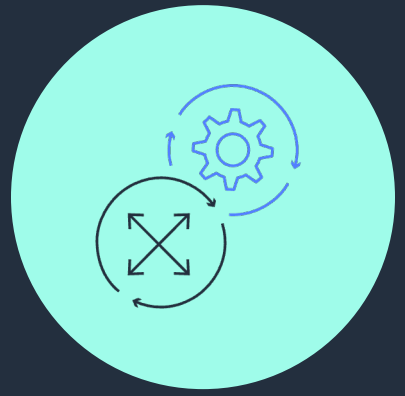
How can we support quantum software development?



On-demand access to hardware



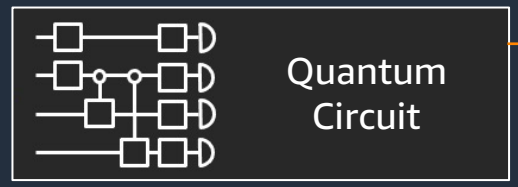
Low-level device control

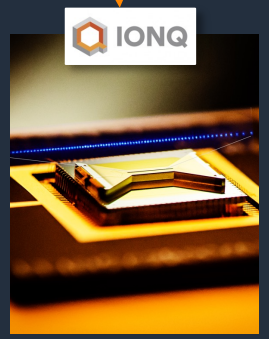


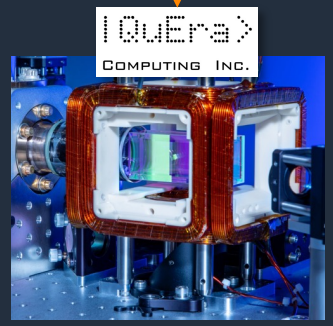
High performance algorithm execution

Cloud integration

It all starts with the hardware





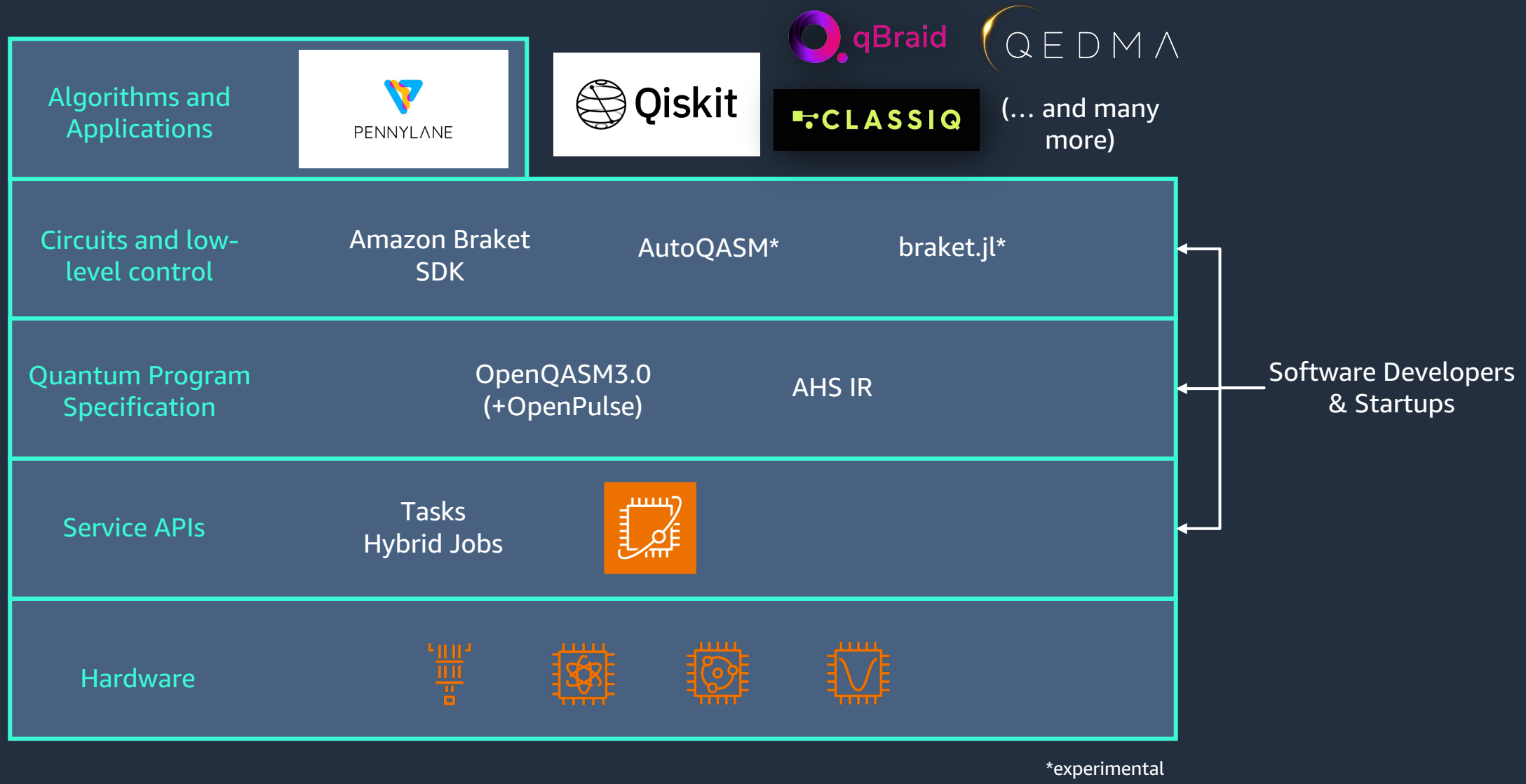


On-demand access

No upfront commitment

Consistent user experience

Built for builders – The Amazon Braket stack



Low-level control to get the most out of hardware

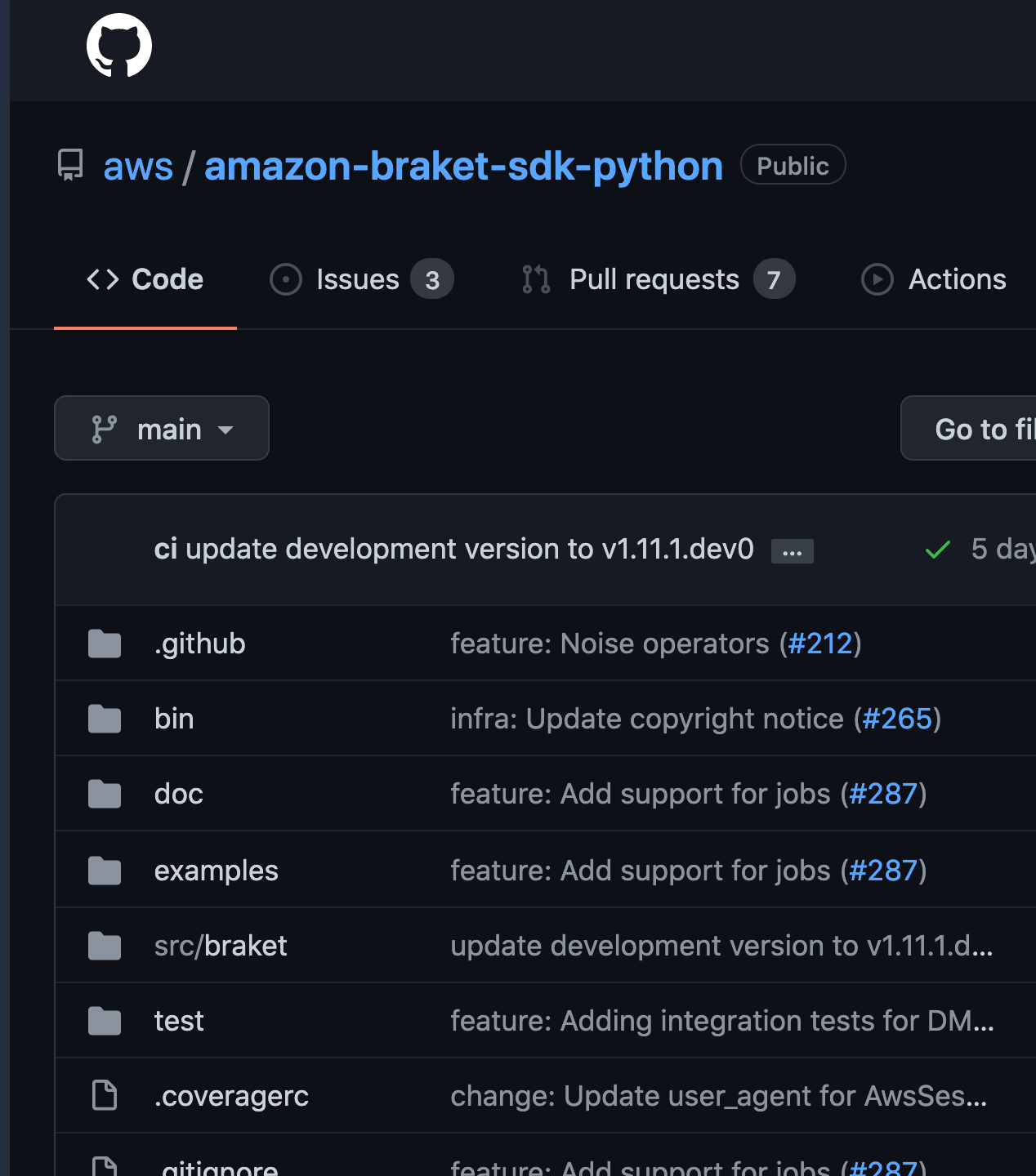
The Amazon Braket Python SDK

Design and build quantum programs

- Create quantum programs (Circuits/AHS)
- Low-level control with Braket Pulse
- Test locally with built-in simulators

Interact with Braket APIs

- Submit tasks and jobs to devices
- Track and monitor their execution

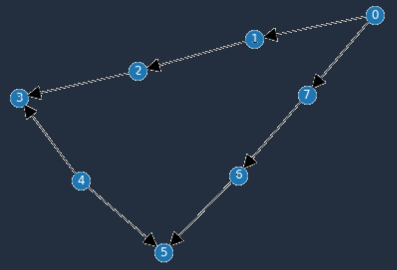


Device-native programming with Braket

Inspect device topologies and properties

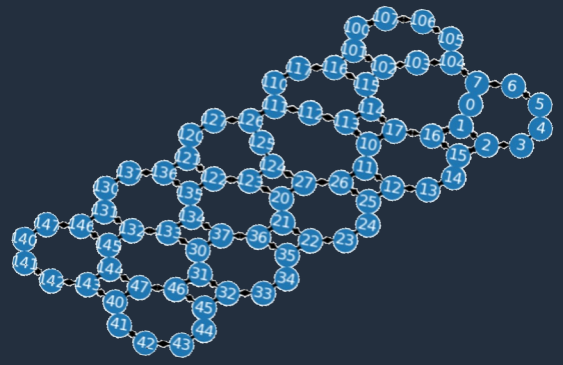
```
# set up the OQC Lucy device
oqc_device = AwsDevice("arn:aws:braket:eu-west-2::device/qpu/oqc/Lucy")

# access and visualize the device topology
print(oqc_device.properties.paradigm.connectivity.connectivityGraph)
nx.draw_kamada_kawai(oqc_device.topology_graph, with_labels=True,
                     font_color="white", arrows=True, arrowsize=30)
```



```
# set up the Rigetti Aspen-M-3 device
rigetti = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3")

# access and visualize the device topology
# note that device topology can change day-to-day based on edge fidelity data
print(rigetti.properties.paradigm.connectivity.connectivityGraph)
nx.draw_kamada_kawai(rigetti.topology_graph, with_labels=True, font_color="white")
```



Program verbatim circuits with native gates

```
# set up the Rigetti Aspen-M-3 device
device = AwsDevice("arn:aws:braket:us-west-1::device/qpu/rigetti/Aspen-M-3")

# list the native gate set
print("The native gates for the", device.name, "device are:")
for gate in device.properties.paradigm.nativeGateSet:
    print(gate)
```

```
The native gates for the Aspen-M device are:
rx
rz
cz
cphaseshift
xy
```

```
circ = Circuit().xy(10,11,pi/4).xy(10,17,pi/2).rx(10,pi).xy(10,11,pi/4)
verbatim_circ = Circuit().add_verbatim_box(circ)
print(verbatim_circ)
```

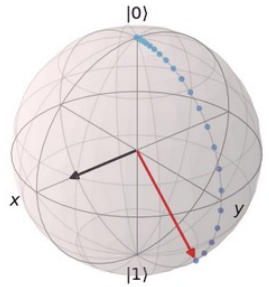
```
T  : |     0     |  1  |  2  |  3  |  4  |     5     |

q10 : -StartVerbatim-XY(0.79)-XY(1.57)-Rx(3.14)-XY(0.79)-EndVerbatim-
      |            |     |               |       |
q11 : -|------------XY(0.79)-|----------------XY(0.79)-|-----------
      |                     |                        |
q17 : -*************----------XY(1.57)-------------------***********-
```

Amazon Braket Pulse

AWS open-sources OQpy to make it easier to write quantum programs in OpenQASM 3

by Philip Reinhold, Li Chen, Jean-Christophe Jaskula, Peter Karalekas, and Stephanie Teo | on 20 OCT 2022 | in Amazon Braket, Announcements, AWS Center For Quantum Computing, Open Source, Quantum Technologies, Thought Leadership | Permalink | Share



Amazon Braket launches Braket Pulse to develop quantum programs at the pulse level

by Jean-Christophe Jaskula, Kshitij Chhabra, Peter Karalekas, and Stefan Natu | on 20 OCT 2022 | in Amazon Braket, AWS Center For Quantum Computing, Quantum Technologies | Permalink | Share

When experimenting on a quantum computer, customers often need to program at the lower-level language of the device. Today, we are launching Braket Pulse, a feature that provides pulse-level access to quantum processing units (QPUs) from two hardware providers on Amazon Braket, Rigetti Computing and Oxford Quantum Circuits (OQC). In this blog, we present an […]

https://aws.amazon.com/blogs/quantum-computing

Pulse-level programming with Braket Pulse

Standalone pulse sequences

```
gaussian = GaussianWaveform(
    length=100e-9, sigma=25e-9, amplitude=0.1)
pulse_sequence = (
    PulseSequence()
    .set_phase(drive_frame, np.pi / 2)
    .play(drive_frame, gaussian)
    .barrier([drive_frame, readout_frame])
    .capture_v0(readout_frame)
)
```

Use pulses within circuits

```
circ = (
    Circuit()
    .rx(0, np.pi / 2)
    .pulse_gate(0, pulse_sequence)
)
```

Optimize your circuits

```
task = device.run(
    circ,
    gate_definitions={
        rx_q0_id: rx_pulse_sequence
    }
)
```

Design **analog quantum algorithms** at the pulse level

Create **modular circuits** including both gates and pulses

Experiment with novel **custom gate implementations**

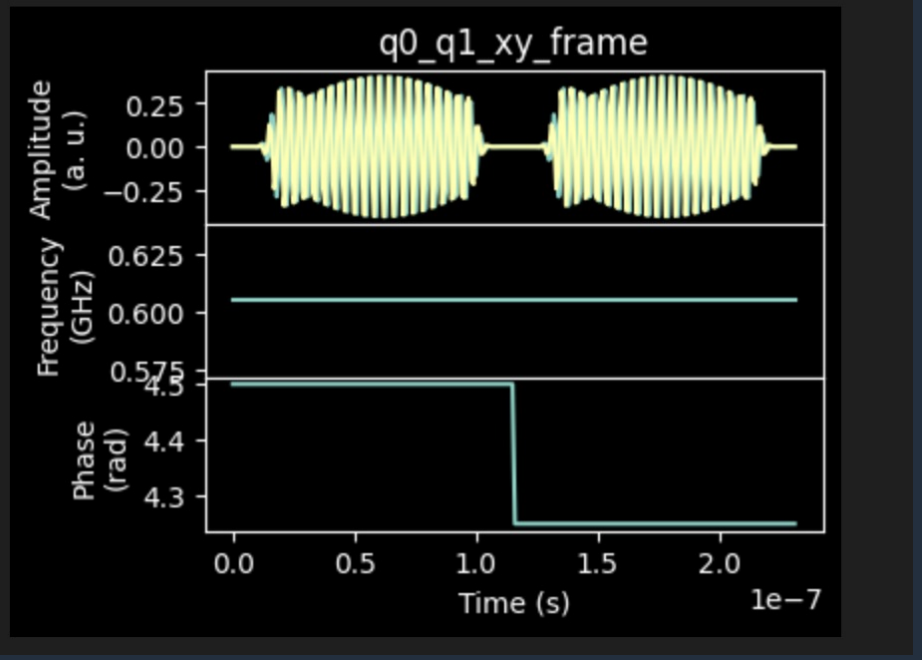
Now available on Rigetti

Access and customize native gate calibrations

Inspect hardware provider implementations

```
calibrations = device.gate_calibrations
draw(calibrations.pulse_sequences[xy_q0_q1](math.pi/2))
```

✓ 1.4s



Customize them to investigate noise effects

```
for factor in factors:
    # stretch lengths, reduce amplitudes
    new_calibrations = stretch_rx_gates(calibrations, factor=factor)
    # run the circuit with longer RX gates
    tasks.append(
        device.run(
            circ,
            gate_definitions=new_calibrations.pulse_sequences
        )
    )
extrapolate_at_zero_noise(tasks)
```

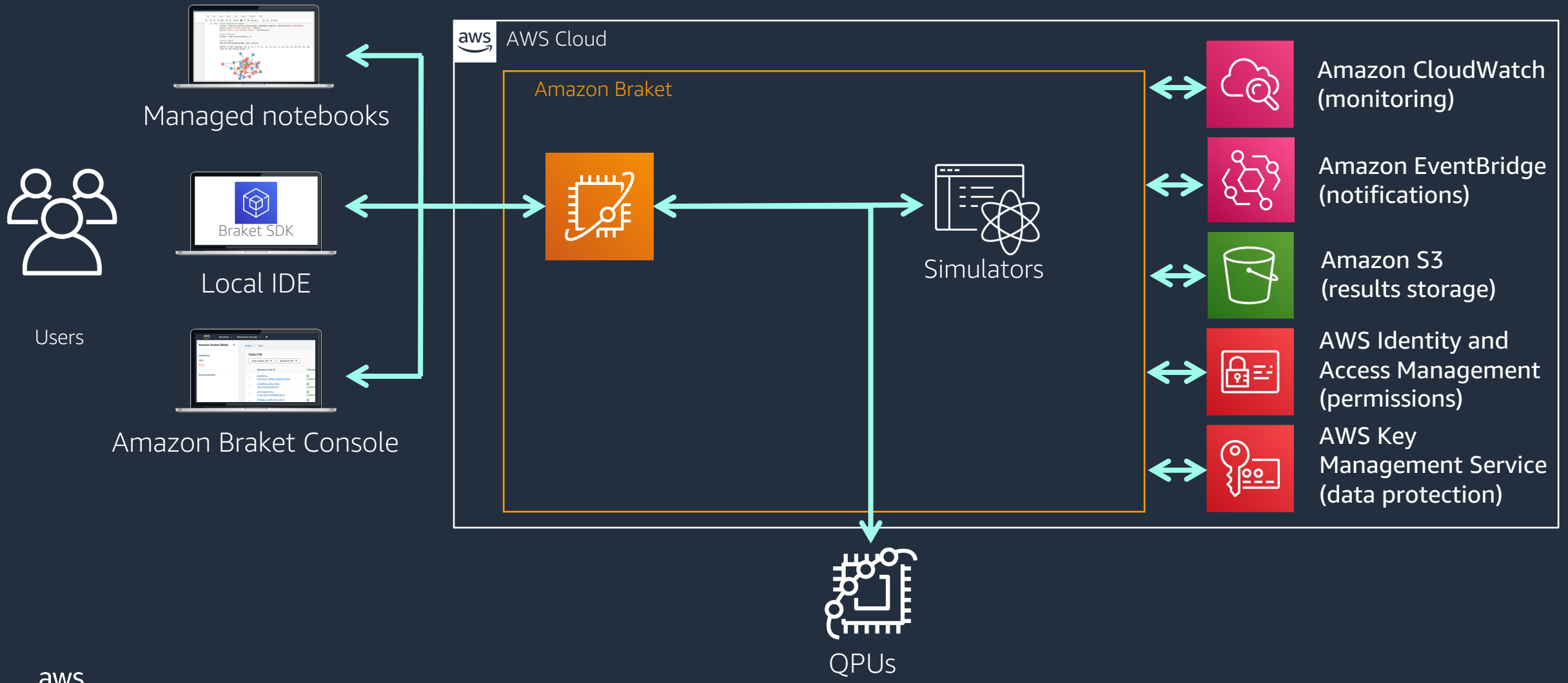
Example: deploy zero-noise extrapolation without modifying your circuits

Optimizing gate implementations at the pulse level

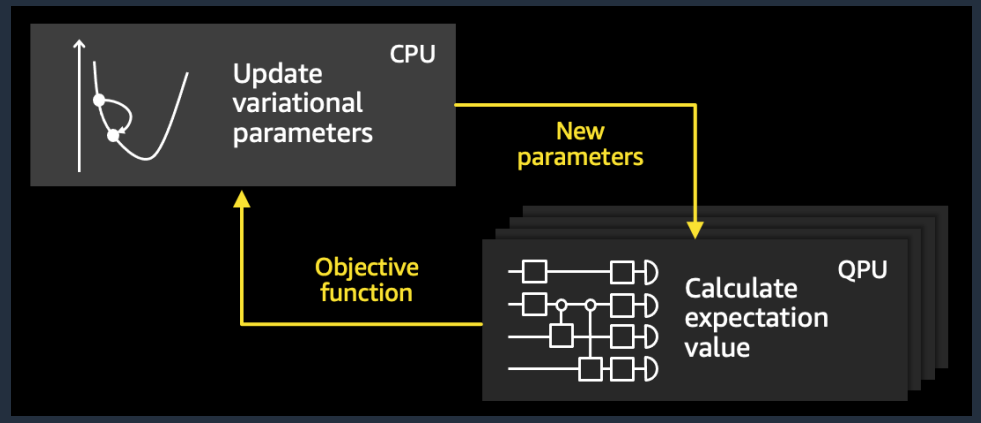
"The team at Q-CTRL is excited to have been an early user of pulse-level control through Amazon Braket. Q-CTRL's published results have demonstrated increased algorithm performance by up to 9000x over conventional approaches. We're thrilled that now, with the reach of the pulse control feature release from Amazon Braket, we'll be able to utilize this capability on several backends to provide multiple orders of magnitude improvement in QPU performance."

High performance quantum algorithm execution

Running *quantum tasks* on Amazon Braket



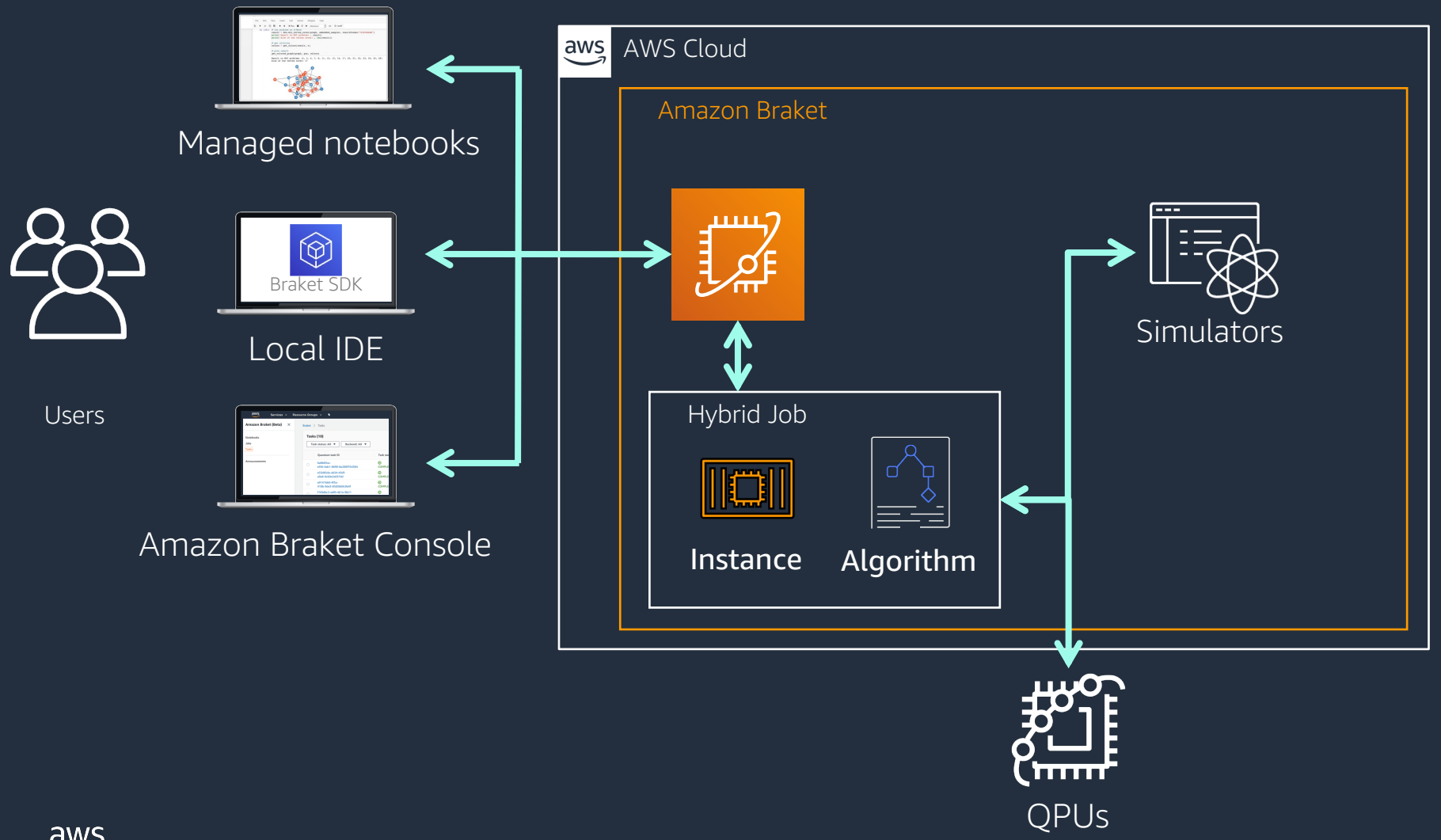
Advanced workloads require fast execution of thousands of tasks



- Setup and maintain classical environment
- Monitor convergence / progress
- Make sure algorithm executes fast to avoid device drift

Running Algorithms with Braket Hybrid Jobs

Now up to 10x faster



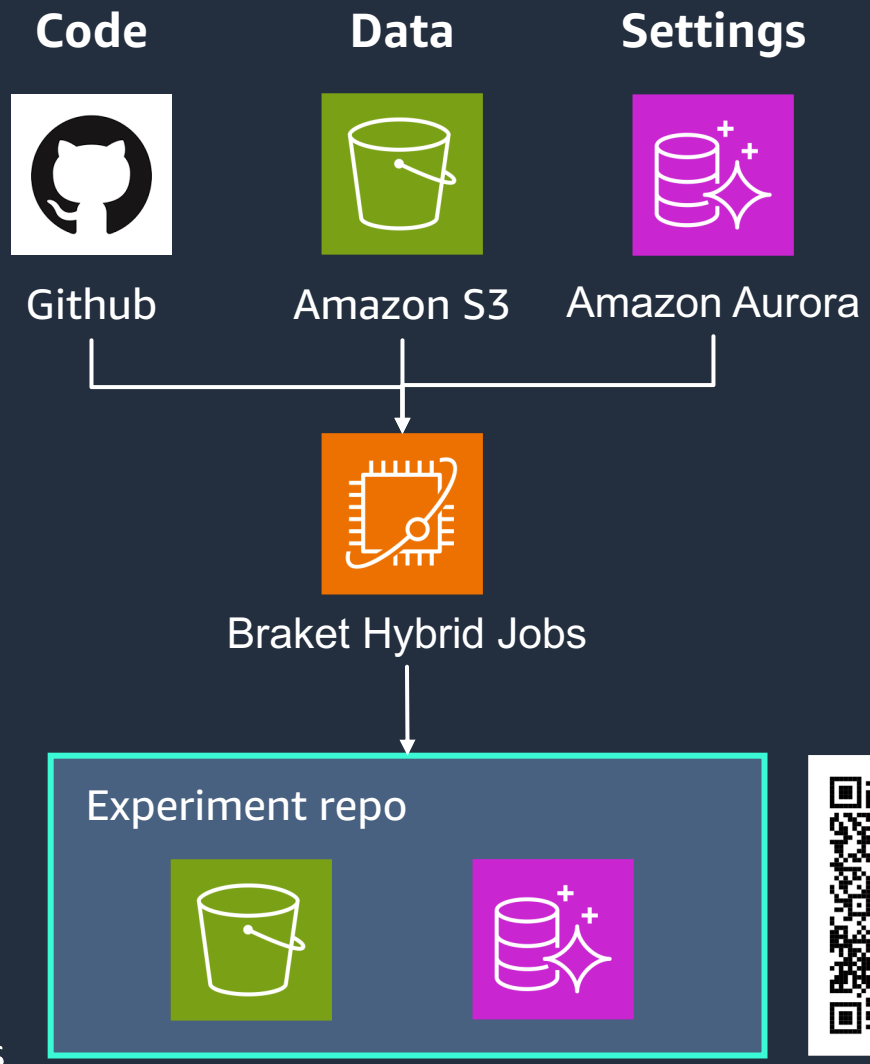
Convenience
Fire-and-forget algorithm execution

Insights
Live monitoring of custom metrics

Performance
Priority access and optimized throughput

Demo – Hybrid Jobs

Learning from Machine Learning – Experiment management best practices



Entropica Labs

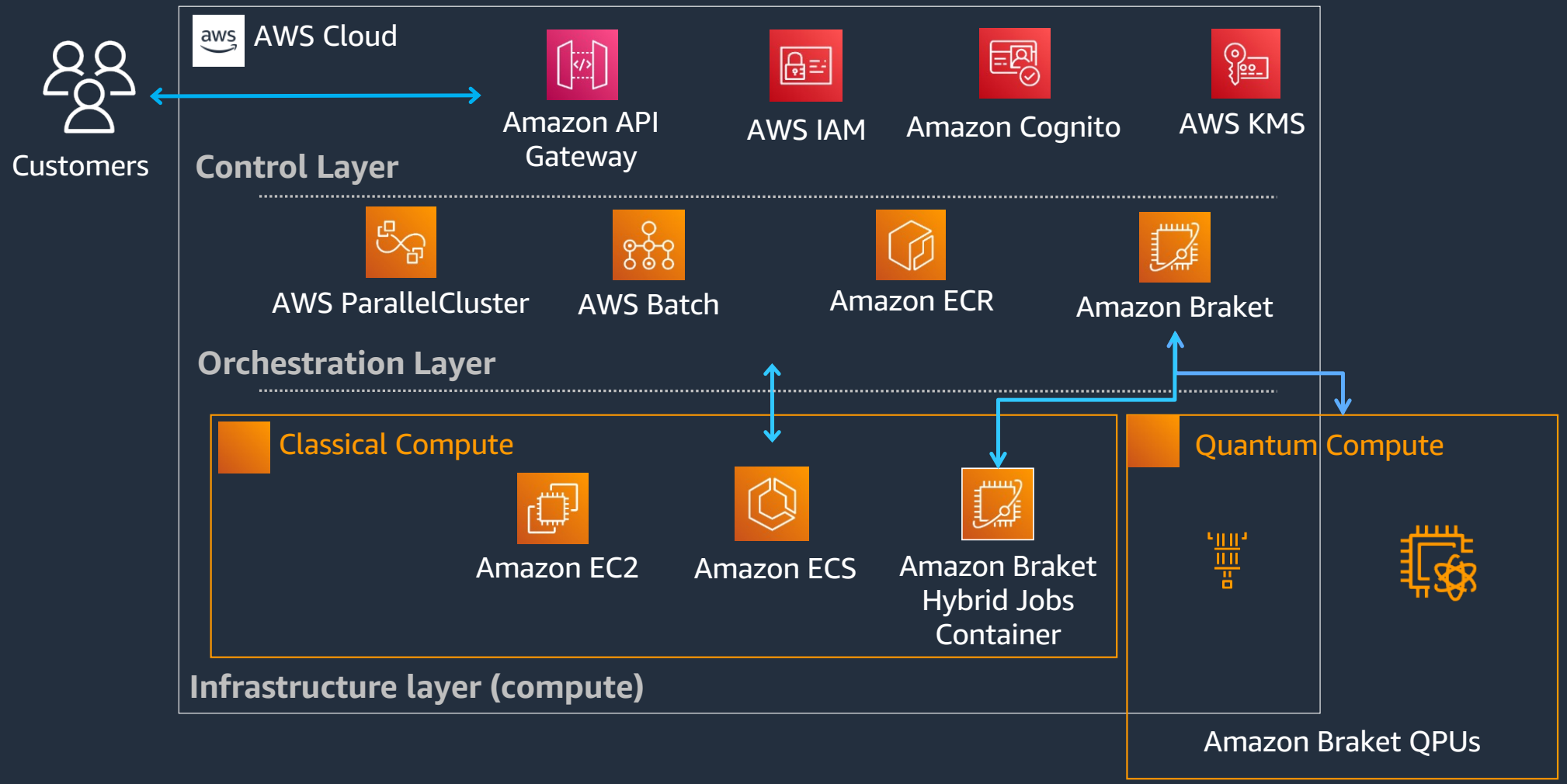
Integrated OpenQAOA with Braket Hybrid Jobs

Used it for running and organizing quantum experiments in engagement with Continental AG

Read about it on our blog

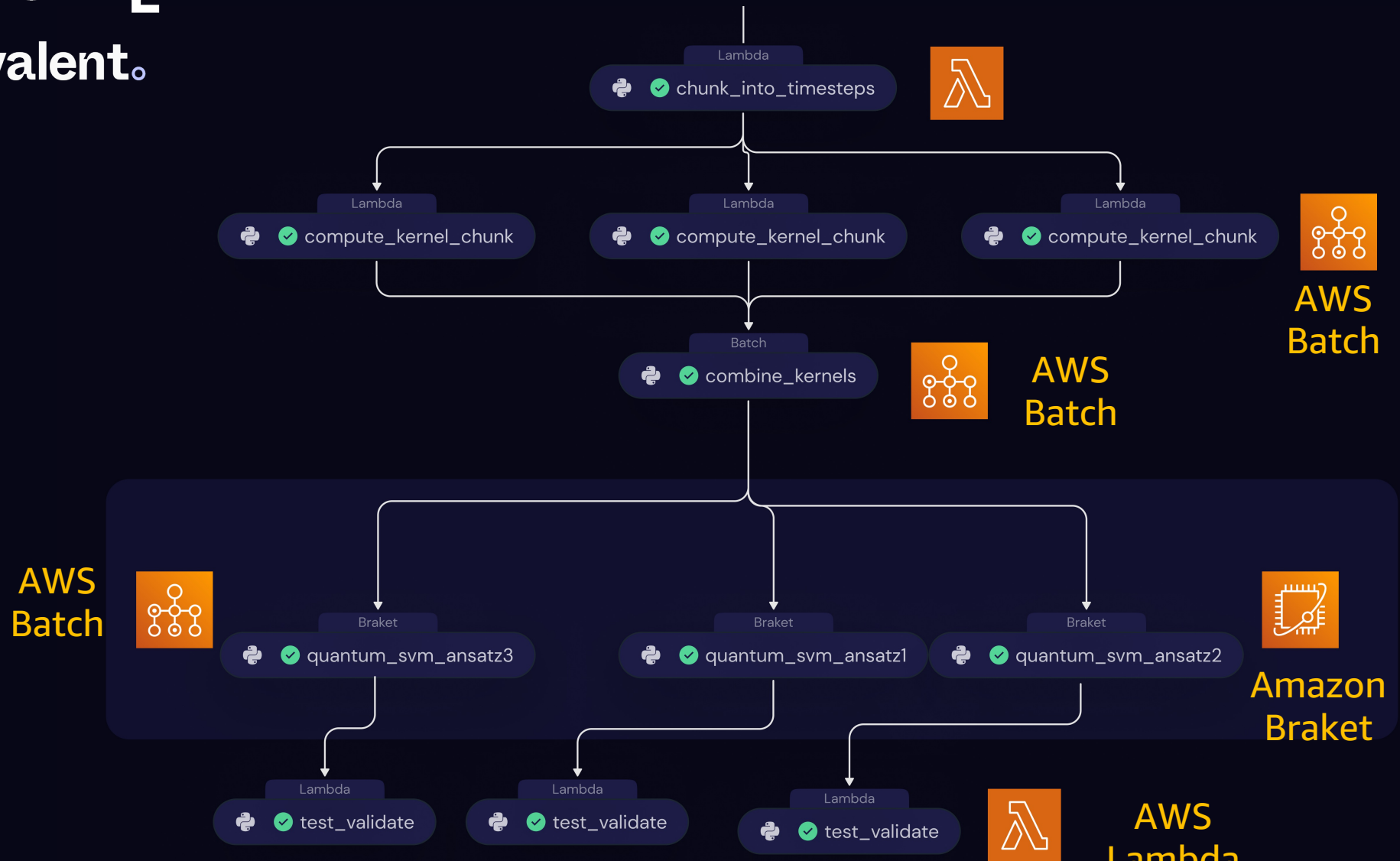
Cloud integration

Amazon Braket – “Just another AWS service”

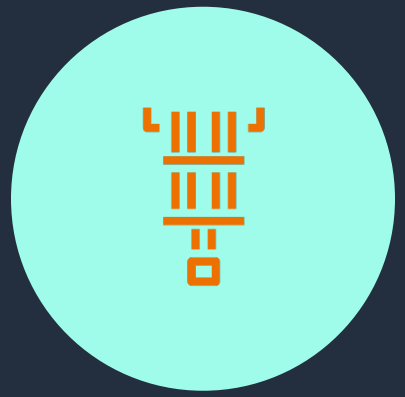


agnostiq

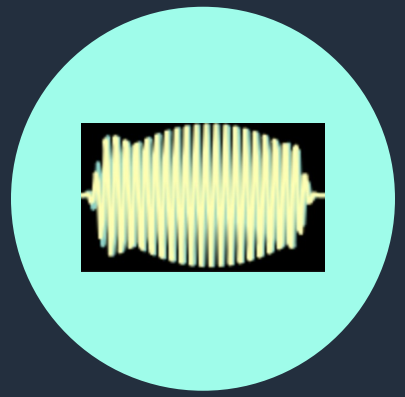
Covalent



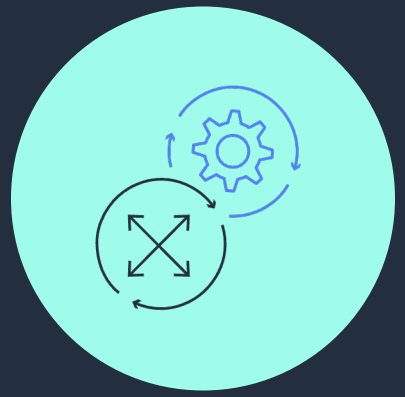
Build on Braket!



On-demand access to hardware



Low-level device control



Programmatic algorithm execution

Cloud integration

Build on Braket!

Bonus content - AutoQASM

Today’s quantum programming tools: 2 main strategies

Python library
(e.g., BDK, Cirq, Qiskit)

- Mature programming tools (e.g., numpy, matplotlib)
- Familiar language

- Need to shoehorn quantum concepts into Python
- Overloading of classical instructions

Domain-specific language
(e.g., Q#)

- Tailored experience for quantum programming
- Real-time classical/quantum programs

- Need to learn a new language
- Need to build all other tooling from scratch (or interoperate with another language)

Experimental

An intuitive, interoperable approach

With the experimental AutoQASM library for Braket, we are working toward a low-level quantum programming interface that is:

Python native

Builds on the AutoGraph library from TensorFlow to provide an intuitive quantum programming interface in pure Python.

Built for quantum

Builds on open-source OpenQASM libraries to generate interoperable OpenQASM 3.0 and OpenPulse programs.

Experimental

Example: Bit-flip error correcting code

Existing Python SDK

```
data = QuantumRegister(3)
ancilla = QuantumRegister(2)
syndrome = ClassicalRegister(2)
qc = QuantumCircuit(data, ancilla, syndrome)

qc.cx(data[0], ancilla[0])
qc.cx(data[1], ancilla[0])
qc.cx(data[0], ancilla[1])
qc.cx(data[2], ancilla[1])
qc.measure(ancilla, syndrome)

with qc.if_test((syndrome, 1)):
    qc.x(data[1])
with qc.if_test((syndrome, 2)):
    qc.x(data[2])
with qc.if_test((syndrome, 3)):
    qc.x(data[0])
```

Braket SDK with AutoQASM

```
data = [0, 1, 2]
ancilla = [3, 4]

@aq.subroutine
def measure_syndrome():
    cnot(data[0], ancilla[0])
    cnot(data[1], ancilla[0])
    cnot(data[0], ancilla[1])
    cnot(data[2], ancilla[1])
    return measure(ancilla)

@aq.subroutine
def correct_error():
    syndrome = measure_syndrome()
    if syndrome:
        x(syndrome % 3)
```

Experimental

Python-native quantum programming interface

AutoQASM is a new experimental module of the Braket SDK

Provides an intuitive interface for low-level quantum programming

Available on GitHub:

AutoQASM

AutoQASM is not an officially supported AWS product.

This experimental module offers a new quantum-imperative programming experience embedded in Python for developing quantum programs.

Installation

AutoQASM is an experimental module and is not yet part of the released Amazon Braket SDK. To use AutoQASM, you'll need to install directly from the `feature/autoqasm` branch:

```
git clone https://github.com/aws/amazon-braket-sdk-python.git
cd amazon-braket-sdk-python
git checkout feature/autoqasm
pip install -e .
```

Quick start

In this section, we will show how to get started with AutoQASM. AutoQASM allows you to build quantum programs with a simplified syntax and run the programs on the service. It uses the circuit model programming paradigm that is also used in the Amazon Braket SDK.

First, import the following modules and functions:

```
import braket.experimental.autoqasm as aq
from braket.experimental.autoqasm.instructions import h, cnot, measure
```

To create a quantum program using the AutoQASM experience, you decorate a function with `@aq.main`. This allows AutoQASM to hook into the program definition and generate an output format that is accepted by quantum devices.

For instance, we can create a Bell state like so:

```
# A program that generates a maximally entangled state
@aq.main
def bell_state() -> None:
    h(0)
    cnot(0, 1)
```

Evolving low-level quantum programming on Braket

Potential future extensions include:

- Circuit ensembles/batches (benchmarking, randomized compiling, …)
- Programmatic transformations (dynamical decoupling for error suppression, qubit remapping, …)
- Complex real-time classical logic (decoding for quantum error correction, …)
- Integration with Braket hybrid jobs (quantum-classical hybrid algorithms, …)
- …and more

Get involved with the open-source Braket SDK and AutoQASM library

We want your feedback!

- Experimental project – we want users to guide our roadmap

Contact me (**erikessl@amazon.com**) at any time for discussions, demos, suggestions, etc., or if you're interested in contributing.

Thank you!

Eric Kessler

erikessl@amazon.com

Apply for AWS Cloud Credits for Research to use with Braket

Read more on the AWS Quantum Technologies blog

Experimental

AutoQASM quantum programming module for Amazon Braket SDK

Thank you!