PLANQK Docs

Menu On this page

Accessing Quantum Backends using the SDK

The PLANQK Quantum SDK provides an easy way to develop quantum circuits that can be executed on quantum devices available through the PLANQK Platform.

The SDK supports both the Qiskit 1.3 SDK and the Amazon Braket SDK, allowing you to choose your preferred framework for quantum programming:

- Qiskit: Access all gate-based quantum backends and simulators provided by PLANQK.
- Amazon Braket SDK: Access all devices provided by PLANQK through AWS, such as the QuEra Aquila quantum device.

Installation

You need to have Python 3.11 or higher installed. The package is released on PyPI and can be installed via `pip`:

bash

```
pip install --upgrade planqk-quantum
```

NOTE

Ensure that you have versions older than Qiskit SDK 1.0 uninstalled before installing the PLANQK Quantum SDK. The best practice is to create a new virtual environment and install the PLANQK Quantum SDK there.

Getting Started

After installation, you can start using the SDK with your preferred framework:

Using Qiskit

Follow the Qiskit Getting Started Guide to begin creating quantum circuits and running them on supported devices through PLANQK.

Using Amazon Braket SDK

Refer to the Braket Getting Started Guide for building circuits or Analog Hamiltonian Simulations with Braket and running them on devices provided via AWS through PLANQK.

Execute your first circuit with Qiskit

In your Python code you can access the PLANQK quantum backends through the `PlanqkQuantumProvider` class. Import the class and instantiate it as shown below:

python
```
from planqk import PlanqkQuantumProvider
```

If you are already logged in with the PLANQK CLI you can create the provider object without any parameters:

python
```
provider = PlanqkQuantumProvider()
```

Alternatively, you can also create the provider object by passing your PLANQK personal access token:

python
```
provider = PlanqkQuantumProvider(access_token="your-access-token")
```

If you want to log in with your organization, you can additionally pass the organization id as a parameter. The organization id can be found in the organization settings on the PLANQK Platform:

python
```
provider = PlanqkQuantumProvider(organization_id="your-organization-id")
```

After you have created the provider object, you can list all backends supported by the PLANQK Platform and select the one you want to use. The available backends and their ids can be also found here :

python

```
# List all available PLANQK quantum backends
backends = provider.backends()

# Select a certain backend
backend = provider.get_backend("azure.ionq.simulator")
```

NOTE

To access other QPUs, either you or your organization must have a Pro account. To upgrade to a Pro account, log in to your Account settings , click *Upgrade*, and select the *Subscribe* button under the Pro section. Follow the prompts to enter your payment details.

Working with Qiskit Backends

Now you can execute your Qiskit circuit on the selected backend, retrieve its `job` object, retrieve its results, cancel it etc. The full example would look like this:

python

```
from planqk import PlanqkQuantumProvider
from qiskit import QuantumCircuit, transpile

# Initialize the provider
provider = PlanqkQuantumProvider()

# Select a backend
backend = provider.get_backend("azure.ionq.simulator")

# Create a qiskit circuit
circuit = QuantumCircuit(3, 3)
circuit.h(0)
circuit.cx(0, 1)
circuit.cx(1, 2)
circuit.measure(range(3), range(3))

# Transpile the circuit for the selected backend
circuit = transpile(circuit, backend)

# Execute the circuit on the selected backend
job = backend.run(circuit, shots=100)

# Monitor job status and get results
```

```
print(f"Status: {job.status()}")
print(f"Result: {job.result()}")
```

NOTE

Executing your quantum circuits or programs on the PLANQK platform may lead to execution costs depending on selected backend and number of shots. Please find an overview about the costs for each backend on our pricing page .

Retrieving Quantum Jobs

Due to queuing at the quantum provider, job execution may take hours or even days. To retrieve your job later, you can use the `retrieve_job` function provided by the backend:

python

```
# Submit the quantum circuit to the backend
job = backend.run(circuit, shots=10)

# Get the job ID for future reference
print("Job ID:", job.id)
# Example Output: Job ID: 6ac422ad-c854-4af4-b37a-efabb159d92e
...
# Get the backend
backend = provider.get_backend("azure.ionq.simulator")
# Retrieve the job through its ID
job = backend.retrieve_job("6ac422ad-c854-4af4-b37a-efabb159d92e")
```

You can also get an overview of all your jobs by executing `provider.jobs()` or by visiting the Quantum Jobs page.

Execute your first circuit with Braket

In your Python code you can access the PLANQK quantum backends through the `PlanqkBraketProvider` class. We refer to these backends as *devices* in the following to adhere to the Braket SDK naming conventions. Import the class and instantiate it as shown below:

python

```
from planqk import PlanqkBraketProvider
```

If you are already logged in with the PLANQK CLI you can create the provider object without any parameters:

```python
provider = PlanqkBraketProvider()
```

Alternatively, you can also create the provider object by passing your PLANQK personal access token:

```python
provider = PlanqkBraketProvider(access_token="your-access-token")
```

If you want to log in with your organization, you can additionally pass the organization id as a parameter. The organization id can be found in the organization settings on the PLANQK Platform:

```python
provider = PlanqkBraketProvider(organization_id="your-organization-id")
```

After you have created the provider object, you can list all devices (backends) provided by the PLANQK Platform that can be accessed through Braket.

```python
# List all available PLANQK quantum devices
devices = provider.devices()

# Select a certain device
device = provider.get_device("aws.ionq.forte")
```

NOTE

To access devices through Braket you or your organization must have a Pro account. To upgrade to a Pro account, log in to your Account settings , click *Upgrade*, and select the *Subscribe* button under the Pro section. Follow the prompts to enter your payment details.

Working with Braket Devices

Now you can execute your Braket circuit on the selected device, retrieve its `task` object, retrieve its results, cancel it etc. The full example would look like this:

python

```python
from braket.circuits import Circuit
from planqk import PlanqkBraketProvider
from planqk.braket.planqk_quantum_task import PlanqkAwsQuantumTask

# Select the IonQ Forte device
device = PlanqkBraketProvider().get_device("aws.ionq.forte")

# Create a Braket circuit
circuit = Circuit().h(0).cnot(0, 1).cnot(1,2)

# Execute the circuit with 100 shots
task = device.run(circuit, 100)

# Monitor task status and get results
print(f"Status: {task.state()})
print(f"Result: {task.result()})
```

To execute a task on the QuEra Aquila device, you'll need to create an Analog Hamiltonian Simulation (AHS) program and discretize it according to the device specifications. This is described in detail using the Maximum Independent Set Problem in our Quera Aquila tutorial.

Retrieving Braket Tasks

To retrieve a task you ran earlier, note down its ID and create a PlanqkAwsQuantumTask object by providing the ID. Optionally, you can also provide an access token and an organization id.

python

```python
# Submit the program to the device
task = device.run(circuit, 100)

# Get the task ID for future reference
print("Task ID:", task.id)
# Example Output: Task ID: 6ac422ad-c854-4af4-b37a-efabb159d92e

# Retrieve the task using its ID
task = PlanqkAwsQuantumTask("6ac422ad-c854-4af4-b37a-efabb159d92e")
```

You can also get an overview of your tasks by visiting the Quantum Jobs page. Note that your tasks are referred to as “jobs” on this page.

Deploy your Quantum Workload as a PLANQK Service

To deploy your circuit to the PLANQK Platform you may adapt the `program.py` file of the `python-starter` template.

> NOTE
>
> To create a new Qiskit development project, you may run `planqk init` and select `Starter Qiskit` as coding template. Further instructions are available here.

The `program.py` file contains a `run()` method which is called when the service is executed. Copy and paste the code from above into the `run()` method and add the following line at the end of the function:

```python
return ResultResponse(result={"status": "COMPLETED"})
```

You may want to add some additional information to the `ResultResponse` object, e.g., the actual results of your circuit. You are now able to deploy your circuit as a PLANQK Service.

Use `planqk up` to deploy your service to the PLANQK Platform. Next, you may use `planqk run` to execute your service.

What's next?

- See our supported quantum backends and simulators .
- Check out this Jupyter notebook showing how to utilize the PLANQK Quantum SDK.

Help us improve this page!

Last updated: 22.01.25, 23:16

Previous page
Quickstart

Next page
Available Backends