

Installation Guide for Qiskit Based Quantum Programming

Torben Larsen¹, Loke Walsted²
{tola¹,loke²}@cs.aau.dk
November, 2024; Version: 1.0



AALBORG UNIVERSITY

Aalborg University
Department of Computer Science
Selma Lagerlöfs Vej 300, Aalborg, Denmark

Preface

This report describes how to install, configure, and maintain a local core software platform that is build around the popular and flexible source code editor Microsoft Visual Studio Code combined with a selection of added tools, which in version 1.0 consists of Python, Jupyter, Git, ... and the quantum back-ends are:

- Microsoft Azure Quantum.
- IBM Quantum.
- Amazon Web Services Braket.

The overall objective of this note is to provide the insight and direct guides to make the following. We wish to establish a simulation, emulation and Quantum Computing hardware management entity on a local laptop or stationary computer. From this typical computing device we wish to have access to all our developed code while having easy access to a number of back-ends of different vendors. We do this by using:

- Microsoft Visual Studio Code as the center of management control. We install various support code such as Qiskit, Python, matplotlib, numpy etc.
- Github is used for software sharing and version control. This can also be coupled to testing of various kind.
- Next, we create a user account on each back-end, we wish to be able to access – e.g. IBM, Quantinuum (via Azure for example), ... We set up an automatic control from Visual Studio Code and the back-end via account and ID information. This can be done for various back-ends, and all back-ends can now be controlled from Visual Studio Code.

The note should seen as a living document that continuously is updated according to its components such as e.g. Microsoft Visual Studio Code and IBM Quantum. There is a Git repository <https://github.com/LowkeyCoding/QuantumSetup> for the report where you can always get the most recent version.

Torben Larsen
13.11.2024

Loke Walsted

Abstract

This report describes how to install, configure, and maintain a local core software platform (e.g. your own laptop) that is build around the popular and flexible source code editor Microsoft Visual Studio Code combined with a selection of added tools, which in version 1.0 consists of Python, Jupyter, Git etc. Further, it includes installation guides for backends from Microsoft Azure Quantum, IBM Quantum, and Amazon Web Services Braket. The report and the included installation recommendations are useful for both beginners and experienced developers of Qiskit code. Also, The guide presents full instructions for Microsoft Windows, Linux, and MacOS operating systems.

Keywords

- Quantum computing, quantum simulation and execution.
- Qiskit installation guide.
- Apple MacOS, Linux, Microsoft Windows.
- Microsoft Visual Studio Code, Python, Git, Jupyter.
- Microsoft Azure Quantum, Amazon Web Services Braket, IBM Quantum Computing.
- Quantum backends.

Contents

1	Introduction	6
1.1	Simulation	6
1.2	Microsoft Azure Quantum	7
1.3	Amazon Web Services Braket	7
1.4	IBM Quantum Computing	8
1.5	Why Use This Guide?	8
2	The Basic Tools	10
2.1	Homebrew (MacOS)	11
2.1.1	Installation	11
2.1.2	Usage	12
2.2	Snap (Linux)	12
2.2.1	Installation	13
2.2.2	Usage	13
2.3	Microsoft Visual Studio Code	13
2.3.1	Installation	13
2.3.2	Setup	14
2.4	Python	15
2.4.1	Installation	15
2.4.2	Virtual Environment	15
2.4.3	Installing Packages in Your Virtual Environment	17
2.5	Git	17
2.5.1	Installation	17
2.5.2	Setting up SSH (Github)	18
2.5.3	Usage	20
2.6	Jupyter (Optional)	21
3	Setting up backends	22
3.1	Virtual Environment	22
3.1.1	Portable environments	23
3.1.2	Updating Portable environments	23
3.2	Basic Packages	23
3.3	Environment File	23
3.4	Microsoft Azure Quantum	24
3.4.1	Get the Connection string	24
3.4.2	Login in from python	24
3.4.3	Running a quantum circuit	25

3.5	IBM Quantum	25
3.5.1	Get API Token	25
3.5.2	Login With Python	26
3.5.3	Running a quantum circuit	26
3.5.4	Revoking Access	26
3.6	Amazon Web Services Braket	27
3.6.1	Getting Access key	27
3.6.2	Login with Python	27
3.6.3	Running a quantum circuit	28
3.7	Aer	28
3.7.1	Running a quantum circuit	28
3.7.2	Running a quantum circuit with noise	29
3.8	Example Quantum Programs	29
4	UCloud	30
4.1	Setup	31

Chapter 1

Introduction

Leading Quantum Computing as a Service (QCaaS) [1, 2] providers like Microsoft Azure Quantum¹, Amazon Braket², and IBM Quantum³ are paving the way for advancements in the field of Cloud based quantum simulation and hardware execution. To empower researchers and foster innovation, it's crucial to provide accessible resources for setting up development environments. These resources should cater to both simulation-based testing and experimentation on real quantum hardware, while assuming no prior knowledge of development tools to ensure inclusivity for researchers across disciplines. This document is a living resource, designed to remain up-to-date with the latest Qiskit versions and backend setup processes. Further, it is the plan to expand the document to include several examples and make these open source available via `git`⁴ and/or EOSC⁵ (when available). In this guide, backends refer to the various simulators and hardware platforms available.

1.1 Simulation

In the realm of quantum simulations, two primary types exist: ideal and noisy. Ideal simulators operate under the assumption of a flawless, error-free quantum system where all states are pure. These are invaluable for comprehending the theoretical behavior of quantum algorithms within a perfect environment. However, they fall short of accurately representing real-world quantum hardware, which is inherently susceptible to errors.

Noisy simulations aim to replicate the real-world noise encountered in quantum hardware by considering various sources like decoherence, gate errors, and measurement errors. These simulations are particularly valuable for near-term quantum devices, which inherently exhibit noise. However, the drawback of noisy simulations lies in their increased computational demands compared to

¹See: <https://azure.microsoft.com/en-us/products/quantum/>.

²See: https://aws.amazon.com/products/quantum/?nc2=h_q1_prod_qt.

³See: <https://www.ibm.com/quantum/>.

⁴See: <https://github.com>.

⁵See: <https://eosc.eu>.

ideal simulations, primarily due to the complexity involved in modeling noise accurately.

The following Table 1.1 outlines the different quantum computing providers, their qubit technologies, and their simulation/emulation capabilities:

Company	Hardware technology	Simulator	Emulator
IonQ ⁶	Trapped Ion	✓	
Pasqal ⁷	Neutral Atom	✓	
Quantinuum ⁸	Trapped Ion	✓	✓
Rigetti ⁹	Transmon	✓	
OQC ¹⁰ (Oxford Quantum Circuit)	Coaxmon	✓	
QuEra ¹¹	Neutral Atom	✓	
IBM Quantum Computing ¹²	Transmon	✓	

Table 1.1: Hardware methods and Simulation/Emulation capabilities.

1.2 Microsoft Azure Quantum

Microsoft Azure Quantum is a cloud based service that offers access to lectures, documentation, simulators, and quantum computers. At the time of writing (Wednesday 27th November, 2024) Microsoft Azure Quantum provides access to simulators and quantum computers from:

- IonQ – <https://ionq.com>.
- Quantinuum – <https://www.quantinuum.com>.
- Rigetti – <https://www.rigetti.com>.

1.3 Amazon Web Services Braket

Amazon Web Services Braket is a fully managed quantum computing service that provides access to a variety of quantum hardware technologies, simulators, and tools for building, testing, and running quantum algorithms. At the time of writing (Wednesday 27th November, 2024), Amazon Web Services Braket offers access to simulators and quantum computers from:

- IonQ – <https://ionq.com>.
- IQM – <https://www.meetiqm.com>.
- |QuEra – <https://www.quera.com>.
- Rigetti – <https://www.rigetti.com>.

1.4 IBM Quantum Computing

IBM¹³ is one of the top players in quantum computing in both development of quantum computers and in software, stack, applications and in general terms the entire ecosystem around quantum computing. IBM is focusing on a superconducting platform, which is characterized by a relatively high and scalable state frequency. Researchers have generally assumed that state-of-the-art in error-correction techniques demands more than 1,000 physical qubits for each logical qubit [3].

However, in the fast moving quantum technology area improvements in materials, error-correction techniques/algorithms may change the expectations in a short time [3]. IBM and others are working hard on solving or mitigating the noise problem, and they have publicly announced a very ambitious road map for handling the noise issue while at the same time upscaling the number of qubits.

1.5 Why Use This Guide?

This guide offers a unique combination of comprehensive setup instructions and streamlined backend integration, catering to both beginners and experienced quantum developers:

For Beginners:

- **Comprehensive Tool Setup:** Walk through a detailed setup process, ensuring you have the necessary tools for basic quantum development. This is especially helpful for non-developers who might not be familiar with managing Python environments or other technical aspects.
- **Learning Curve:** Beginners may need to invest time in learning a suite of new tools and concepts, which can be a steep learning curve. However, this foundational knowledge will prove invaluable for future collaboration on larger quantum projects.

For Experienced Developers:

- **Minimalistic Package Approach:** Focus on the essential packages needed for implementing quantum algorithms, avoiding unnecessary dependencies.
- **Simplified Backend Setup:** Get up and running with various backends quickly, thanks to clear instructions and a unified approach.
- **Enhanced Collaboration:** Easily share your code and projects with others, as the guide emphasizes using common base packages and Python versions, minimizing compatibility issues.
- **Cross-Platform Compatibility:** Benefit from a suite of example programs tested on macOS, Linux, and Windows, ensuring smooth operation on your preferred platform.
- **Inspiration and Guidance:** Explore a collection of tested example [4] programs to spark your own quantum algorithm development.

¹³See: <https://www.ibm.com/quantum>.

By choosing this guide, you gain a valuable resource that simplifies the quantum development process, regardless of your experience level. It empowers you to focus on what truly matters: exploring the exciting world of quantum algorithms and their potential applications.

Chapter 2

The Basic Tools

This guide is intended to aid a quantum simulation user to easy local installation of common development tools and package managers. The guidelines are intended for:

- Microsoft Windows (referred to as **Win**).
- Apple MacOS (referred to as **MacOS**).
- Linux (referred to as **Linux**).

Installation begins by first providing an overview of the installation process for **Win**, **MacOS**, and **Linux**. Next, the installation process begins by using package managers to install essential programs: **Python** for creating quantum programs with **Qiskit**, **Microsoft Visual Studio Code** for editing, and **Git** for version control and sharing. Next, we'll explore the various backends available for running quantum programs on both simulators and real quantum hardware. The following dependency graph in Figure 2.1 illustrates the relationships between the relevant components.

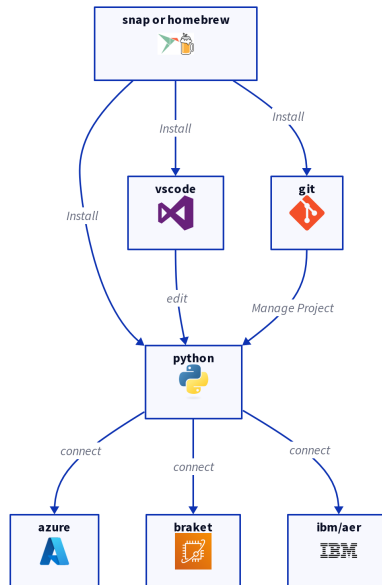


Figure 2.1: Dependency graph for basic tools.

2.1 Homebrew (MacOS)

2.1.1 Installation

To install, go to [homebrew](https://brew.sh) and click on , which provides Figure 2.2.

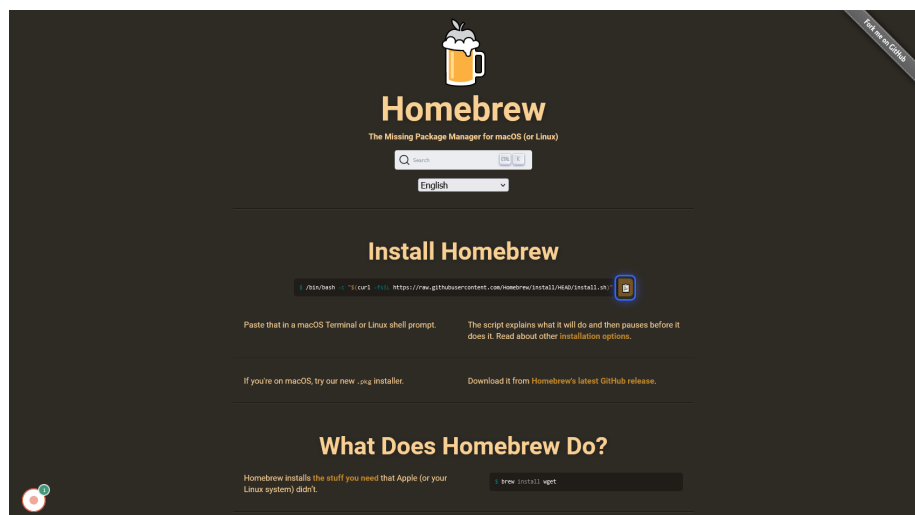


Figure 2.2: Install interface for Homebrew.

Then, (1) click the Launchpad icon in the Dock, (2) type Terminal in the search field, (3) then click Terminal, and finally (4) then in the Terminal, choose Edit > Paste and press enter. From here on, just follow the instructions in the Terminal.

2.1.2 Usage

This subsection introduces you to the fundamentals of using Homebrew, which is a powerful package manager for MacOS and Linux. The following covers its basic usage and explain some of the terminology encountered by a user.

- **Search:** `brew search <package_name>`. Finds packages matching the given name.
- **Install:**
 - General: `brew install <package_name>`.
 - MacOS (preferred): `brew install --cask <package_name>`. Installs native macOS binaries for a smoother experience.
- **Uninstall:** `brew uninstall <package_name>`.
- **Update:** `brew update`. Gets the newest version of Homebrew and install packages.
- **List Installed Packages:** `brew list`. Displays all packages currently installed via Homebrew.

In line with its name, Homebrew uses brewing-related terms for different package types:

- **Keg:** A keg is a binary program compiled from its source code on your local machine.
- **Bottle:** A bottle is a pre-compiled binary downloaded file sfrom Homebrew’s servers.
- **Cellar:** The cellar is the directory where Homebrew stores both kegs and bottles (the binaries of the installed packages).
- **Tap:** A tap is a Git repository containing additional software formulas (package definitions) that are not part of Homebrew’s main repository.
- **Cask:** Specifically for macOS, a cask is a formula that installs native macOS applications (apps with graphical interfaces) instead of command-line tools.

2.2 Snap (Linux)

Snap is a user-friendly package manager designed to simplify the process of installing software applications on your Linux system.

2.2.1 Installation

To test your system for the installment and availability of the Snap package manager, try running:

```
sudo snap install hello-world
```

If Snap is not installed, you'll receive an error message. To install Snap on e.g. Linux Ubuntu, execute the following commands in your terminal:

```
sudo apt update  
sudo apt install snapd
```

For Linux distributions other than Ubuntu, detailed instructions on installing Snap can be found [here](#). Simply select your distribution from the list to access the specific installation steps.

2.2.2 Usage

This subsection introduces you to the fundamentals of using Snap, a versatile package manager for Linux distributions.

- **List installed packages:** `snap list`. Displays all the currently installed snap packages on your system.
- **Find packages:** `snap find "<packag_name>"`. Searches for packages with names similar to the one you provided.
- **Install a package:** `sudo snap install <package_name>`. Downloads and installs the specified package.
- **Refresh a package:** `snap refresh <package_name>`. Updates the package to the latest version if it's already installed.
- **Run a program:** `snap run <package_name>`. Can be used to launch an installed program if executing it directly does not work.
- **Remove a package:** `sudo snap remove <package_name>`. Uninstalls the specified package from your system.

2.3 Microsoft Visual Studio Code

2.3.1 Installation

Windows

Go to [Visual Studio Code](#) and press the windows download button to get the interface in Figure 2.3.

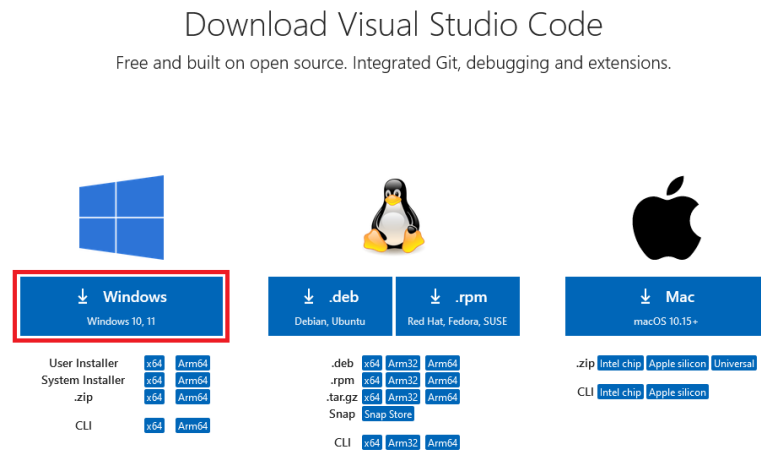


Figure 2.3: Install interface for Visual Studio Code.

Then run the executable and follow the installation instructions.

MacOS

To install Visual Studio Code on MacOS, use brew to install it or use your preferred package manager.

```
brew install --cask visual-studio-code
```

Linux

To install Visual Studio Code on Linux, use snap to install it or use your preferred package manager.

```
sudo snap install code --classic
```

2.3.2 Setup

This section guides you through setting up a basic Visual Studio Code Environment tailored for Python development, complete with essential extensions to streamline your workflow. To enhance your Python development experience in Visual Studio Code, follow these steps:

- **Access the Extensions Tab:** Click on the Extensions icon (four squares) located in the sidebar.
- **Search and Install Python Extension:** In the search bar, type "@id:ms-python.python", and click the "Install" button to gain IntelliSense (code completion) and debugging support.
- **Install Jupyter Extension (Optional):** If you work with Jupyter notebooks, search for "@id:ms-toolsai.jupyter", and click the "Install" button to enable opening, and running Jupyter notebooks directly within Visual Studio Code.

2.4 Python

2.4.1 Installation

Windows

Go to [python 3.12](#), scroll to the files section as shown in Figure 2.4.

Files						
Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore
Gzipped source tarball	Source release		d6eda3e1399cef5dfde7c4f319b0596c	25.9 MB	SIG	.sigstore
XZ compressed source tarball	Source release		f6f4616584b23254d165f4db90c247d6	19.6 MB	SIG	.sigstore
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	eddf6f35a3cab94f2f83b2875c5fc27	43.3 MB	SIG	.sigstore
Windows installer (64-bit)	Windows	Recommended	32ab6a1058dfbde76951b7aa7c2335a6	25.3 MB	SIG	.sigstore
Windows installer (ARM64)	Windows	Experimental	230c703e3b8b3d92765d118afa7b2f78	24.5 MB	SIG	.sigstore
Windows embeddable package (64-bit)	Windows		8e24d2b26a8dbf1da0694b9da1a08b2c	10.5 MB	SIG	.sigstore
Windows embeddable package (32-bit)	Windows		c2047dc270c4936f9c4619bb193b721	9.4 MB	SIG	.sigstore
Windows embeddable package (ARM64)	Windows		3da91ef1a86a8a210a32ea99c709dd93	9.8 MB	SIG	.sigstore
Windows installer (32-bit)	Windows		de59862985bf7afa639f2e4f9e2a722c	24.0 MB	SIG	.sigstore

Figure 2.4: Display files to find python 3.12.

and download "Windows Installer (64bit)". Then run the executable and follow the installation instructions.

MacOS

To install python on MacOS, use brew to install it or use your preferred package manager:

```
brew install python@3.12
```

Linux

To install python on Linux, use snap to install it or use your preferred package manager.

```
sudo snap install python3-alt
sudo snap alias python3-alt.3-12 python3
```

2.4.2 Virtual Environment

Virtual environments play a crucial role in software development by creating self-contained spaces for each project. This isolation prevents conflicts that can arise when different projects rely on varying versions of the same package. To set up a virtual environment, use the appropriate command for your operating system:

Windows:

```
py -m venv <environment_name>
```

Linux/MacOS:

```
python3 -m venv <environment_name>
```

Replace `<environment_name>` with your desired name for the virtual environment.

After setting up a virtual environment, you need to activate it before using it. This ensures that your project uses the correct package versions and dependencies within the isolated environment.

To activate your virtual environment, navigate to your project directory in your preferred terminal. Then, execute the following command based on your operating system:

Windows:

```
.\<environment_name>\Scripts\activate
```

If you encounter an error like `scripts cannot be loaded because running scripts is disabled on this system` you need to adjust your PowerShell execution policy. This can be done by running the following command in an elevated PowerShell window (run as administrator):

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

This command allows the execution of locally stored scripts and remote scripts signed by trusted publishers.

Linux/MacOS:

```
source <environment_name>/bin/activate
```

Replace `<environment_name>` with the name you chose for your virtual environment during creation. Once activated, you'll notice the environment name in parentheses at the beginning of your terminal prompt, indicating that you're working within the isolated environment.

Once you've finished working within your virtual environment, there are two simple ways to exit:

- **Close the Terminal:** The most straightforward method is to simply close the terminal window or tab where you activated the environment. This will automatically deactivate the environment.
- **Run the Deactivation Script:** If you prefer to keep the terminal open, you can deactivate the environment by running the following command for your operating system:

Windows:

```
.\<environment_name>\Scripts\deactivate
```

Linux/MacOS:

```
source <environment_name>/bin/deactivate
```

2.4.3 Installing Packages in Your Virtual Environment

Once your virtual environment is activated, you can start installing the packages your project needs. There are two primary approaches. Manual Installation as:

```
pip install <package_name>
```

Replace `<package_name>` with the name of the package you want to install. You can specify a version by adding `==<version_number>`. For example:

```
pip install numpy==1.23.5
```

For existing projects that come with a `requirements.txt` file, you can install all the necessary packages in one go:

```
pip install -r requirements.txt
```

This file lists all the packages and their versions required by the project. Pip will automatically download and install them for you. This is recommended for reasons of easy, reliable, and secure use. Remember, always activate your virtual environment before installing packages to ensure they are installed within the isolated environment.

2.5 Git

Git is essential for version control, enabling collaboration, tracking changes, and managing projects effectively. It provides features like branching, tagging, and reverting to previous versions, making it indispensable for software development and collaboration.

2.5.1 Installation

Windows

Go to [git](#) and download "64-bit Git for Windows Setup".

Download for Windows

[Click here to download](#) the latest (2.44.0) **64-bit** version of **Git for Windows**. This is the most recent [maintained build](#). It was released **about 2 months ago**, on 2024-02-23.

Other Git for Windows downloads

Standalone Installer

[32-bit Git for Windows Setup.](#)

[64-bit Git for Windows Setup.](#)

Portable ("thumbdrive edition")

[32-bit Git for Windows Portable.](#)

[64-bit Git for Windows Portable.](#)

Figure 2.5: Downloading git for Microsoft Windows.

MacOS

To install git on MacOS, use brew to install it or use your preferred package manager:

```
brew install git
```

Linux

To install git on Linux, use snap to install it or use your preferred package manager:

```
sudo snap install git-confined
snap alias git-confined git
```

2.5.2 Setting up SSH (Github)

After creating a [Github](#) account and installing Git, the easiest way to authenticate with Github is through SSH. To begin, configure your Git settings with your name and email:

```
git config -global user.name "Your name here"
git config -global user.email "your_email@example.com"
```

Next, locate your `.ssh` folder, typically found in your user's home directory:

- For Windows, this path is usually `C:\Users\<User>\.ssh\`
- For Linux/macOS, it is `~/.ssh/`

Replace `<User>` with your actual username.

The next step is going to the `.ssh` directory in your terminal using the following command:

```
cd <ssh_path>
```

The next step is to generate an SSH key pair using the following command replacing `your_email@example.com` with the email you used to create your Github account:

```
ssh-keygen -t rsa -C "your_email@example.com"
```

Follow the prompts to enter a filename for your key pair (e.g., `id_git`) and optionally, a passphrase for added security.

Within the `.ssh` directory, create or edit a file named `config` (without file extension). Add the following lines to this file:

```
Host Github.com
  User git
  Hostname Github.com
  PreferredAuthentications publickey
  IdentityFile <ssh_path><your_private_key_filename>
```

Then replace `<your_private_key_filename>` with the actual filename of your private key (e.g., `id_git`). This configuration tells Git to use your SSH key whenever you interact with Github.

When you generated your SSH key pair, a public key was also created alongside the private key. This public key needs to be added to your Github account to enable secure SSH authentication.

You should find the public key in the same `.ssh` directory as your private key, with a similar name but with the `.pub` extension (e.g., `id_git.pub`):

- Login to your Github account.
- Navigate to [SSH Settings](#).
- Click on the **New SSH key** button.
- Provide a title for the key.
- Open the public key file (e.g., `id_git.pub`) in a text editor and copy the entire contents.
- Paste the copied public key into the **Key** field on Github.
- Click **Add SSH key** to save it.

Now, Github will be able to authenticate you using the SSH key pair you generated.

To test your SSH setup and ensure it's working correctly with Github, run the following command in your terminal:

```
ssh -T git@github.com
```

If successful, you should see a message similar to this:

```
Hi username! You've successfully authenticated, but Github
does not provide shell access.
```

Where `username` is your actual Github username. This message confirms that your SSH key is recognized by Github, and you're now able to securely interact with your repositories using SSH.

If you have any issues Github has more extensive documentation on potential errors during setup: [documentation](#).

2.5.3 Usage

This section provides a brief overview of essential Git commands for beginners:

- `git clone <repository_url>`: Creates a local copy of a remote repository. Replace `<repository_url>` with the actual URL of the repository.
- `git checkout <branch_name>`: Switches to a different branch, allowing multiple developers to work on the same code base without interfering with each other.
- `git branch` or `git branch <branch_name>`: Lists all branches in your repository or creates a new branch with the specified name.
- `git add <file>` or `git add .`: Adds changes in a specific file or all changes in the current directory to staging, preparing them for the next commit.
- `git commit -m "<message>"`: Records the changes to the repository in staging along with a message summarizing the changes.
- `git push`: Uploads your local commits to the remote repository, making your changes accessible to others. Make sure not to push anything confidential or information of sensitive kind.
- `git pull`: Downloads any changes from the remote repository and merges them into your local branch, keeping the code up to date.
- `git merge`: Combines changes from a specified branch into your current branch.

These are just a few of the Git commands available. For more information look at the [reference manual](#).

2.6 Jupyter (Optional)

Jupyter is a powerful tool for developing quantum algorithms due to its interactive nature. With Jupyter notebooks, developers can write code interspersed with explanatory text, equations, and visualizations. This makes it easy to experiment with different quantum algorithms, visualize quantum states, and analyze results in real-time. To install the Jupyter Python package, make sure your virtual environment is activated and then run the following command in your terminal:

```
pip install jupyter ipykernel
```

After installing the Python package, Jupyter extension for VSCode must also be installed.¹

¹See: [2.3.2](#) to install the extension.

Chapter 3

Setting up backends

This chapter is about getting a basic setup, creating a virtual Python environment for your project, and installing basic packages that are useful for all backends. Finally, setting up your desired backend and running a simple test quantum program.

3.1 Virtual Environment

Go to your desired folder for your quantum project and open a terminal or command prompt and use the following commands to create and activate an environment.

Windows

```
py -m venv .venv
.venv\Scripts\activate
```

If you encounter an error like `scripts cannot be loaded because running scripts is disabled on this system` you need to adjust your PowerShell execution policy. This can be done by running the following command in an elevated PowerShell window (run as administrator):

```
Set-ExecutionPolicy RemoteSigned -Scope CurrentUser
```

This command allows the execution of locally stored scripts and remote scripts signed by trusted publishers.

Linux/MacOS

```
python3 -m venv .venv
source .venv/bin/activate
```

To enter the project's virtual environment, you'll need to run the second command. It's crucial to ensure you're in the virtual environment, especially when installing packages. This helps prevent conflicts with system-wide packages.

Make sure to check if you're in the virtual environment by verifying if `(.venv)` is appended to the current path in the terminal before installing any packages. If you are still unsure, then you can run the second command to enter it.

3.1.1 Portable environments

To enhance project portability and ensure consistent environments for collaborators, consider freezing package versions. This can be achieved by generating a `requirements.txt` file listing exact package dependencies and their versions.

```
pip freeze > requirements.txt
```

3.1.2 Updating Portable environments

To update packages in your environment, install the `pur` package using `pip` and run it on your `requirements.txt` file. If you do not have a `requirements.txt` file, follow the steps in Section 3.1.1 to generate one. This will update the file to list the latest versions of the packages currently installed in your `pip` environment.

```
pip install pur
pur -r requirements.txt
pip install --force-reinstall -r requirements.txt
```

After updating packages, there may be differences in package usage. It is recommended to check the [Quantum Setup](#) repository for updated examples or consult the documentation of individual packages.

3.2 Basic Packages

To create a basic setup with a couple of helpful packages for quantum computing, install the following packages for your operating system.

```
pip install qiskit matplotlib pylatexenc numpy
pip install python-dotenv
```

3.3 Environment File

First step is creating an environment file with the name `.env` in your project folder. This file should never be shared, as anyone who has access will be able to spend credits. If you are using `git`, this can be avoided by adding it to the `.gitignore` file, by adding the line `.env`. The environment file operates as a key-value store, utilizing the format `key = value`, with each key on a separate line. In subsequent guides for various backends, we will employ the `.env` file to store login information.

3.4 Microsoft Azure Quantum

To create a basic Microsoft Azure Quantum setup, then install the following packages for your operating system.

Windows

```
py -m pip install azure-quantum azure-quantum[qiskit]
```

Linux/MacOS

```
pip install azure-quantum "azure-quantum[qiskit]"
```

Linux/MacOS users may need to install PyQt5 if they don't use Jupyter to show plots.

```
pip install PyQt5
```

3.4.1 Get the Connection string

To find the Connection String go to [azure portal](#) and locate Quantum workspace under services. From there you should be able to click Operations and then Access Keys. Finally copy the Connection String.

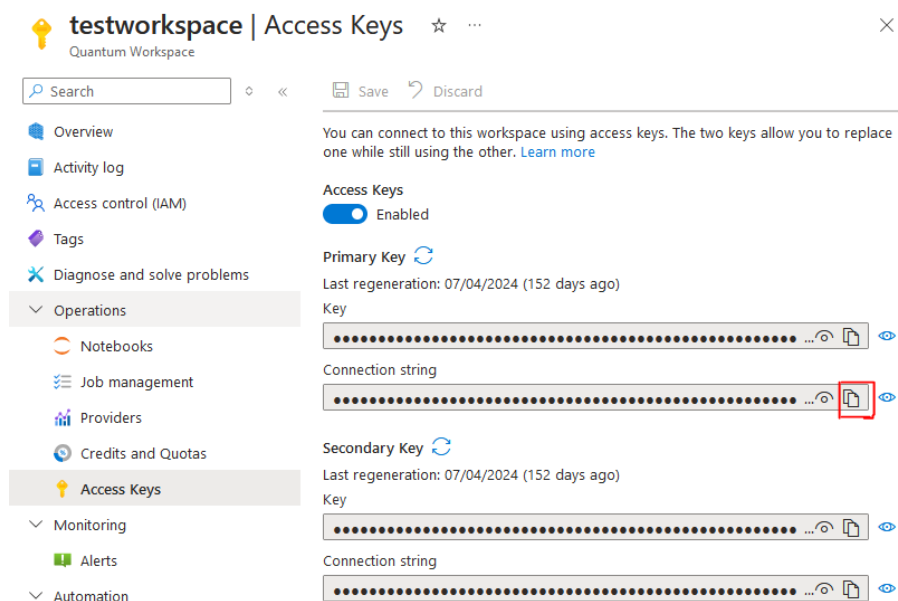


Figure 3.1: Azure Connection String from quantum workspace.

3.4.2 Login in from python

To login using python add the line "ID=Your Resource ID Here" and "Location = Your Location Here" to the environment file .env:

```

1 import os
2 from azure.quantum import Workspace
3 from azure.quantum.qiskit import AzureQuantumProvider
4 from dotenv import load_dotenv
5
6 # Load environment variables
7 load_dotenv(dotenv_path="path to environmnet file")
8 # If environment file is in the same folder as the script then use it without parameters "load_dotenv()"
9
10 workspace = Workspace.from_connection_string(os.environ['azure_connection'])
11 provider = AzureQuantumProvider(workspace)

```

To list the currently available backends add the snippet below to the login example:

```

1 print("This workspace's targets:")
2 for backend in provider.backends():
3     print("- " + backend.name())

```

3.4.3 Running a quantum circuit

To run a quantum circuit, a specific backend has to be chosen from the list of currently available ones. This is done by changing the backend on the line shown below in the [sample script](#).

```

1 backend = provider.get_backend("ionq.simulator")

```

3.5 IBM Quantum

To install the required packages for IBM Quantum run the following command:

```

pip install qiskit-ibm-runtime

```

3.5.1 Get API Token

Go to the [IBM Quantum Dashboard](#). If you don't have an account, sign up for one. Otherwise, log in. Once logged in, navigate to your dashboard. You'll find your API key there. Copy your API key. This is used to authenticate your access to IBM Quantum services.

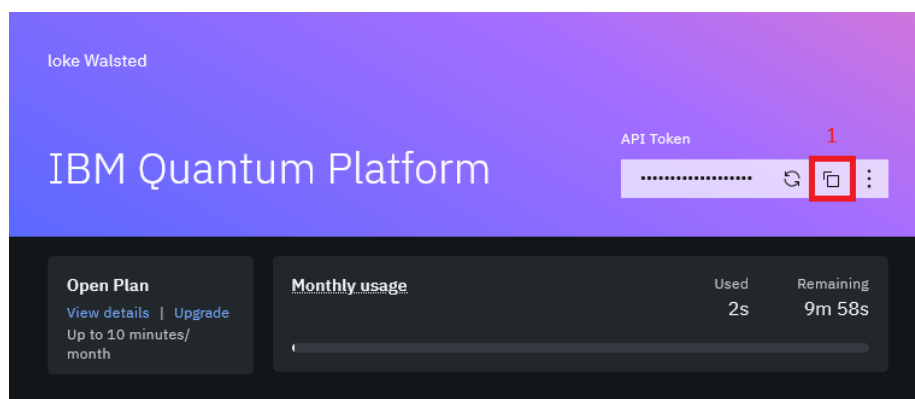


Figure 3.2: API token from the IBM quantum platform.

3.5.2 Login With Python

To login using python add the line `ibm_token=Your API Token Here` to the `.env` file:

```
1 from qiskit import QuantumCircuit, transpile
2 from qiskit.visualization import plot_histogram
3 from qiskit_ibm_runtime import QiskitRuntimeService
4 from dotenv import load_dotenv
5
6
7 # Load environment variables
8 load_dotenv(dotenv_path="path to environmmnet file")
9 # If environment file is in the same folder as the script then use it without parameters "load_dotenv()"
10
11 provider = QiskitRuntimeService(token=os.environ['ibm_token'])
```

To list the currently available backends add the snippet below to the login example.

```
1 print("This workspace's targets:")
2 for backend in provider.backends():
3     print("- " + backend.name())
```

3.5.3 Running a quantum circuit

To run a quantum circuit, a specific backend has to be chosen from the list of currently available ones. This is done by changing the backend on the line shown below in the [sample script](#).

```
1 backend = provider.get_backend("ibm_brisbane")
```

3.5.4 Revoking Access

Go to the [IBM Quantum Dashboard](#). Then click the refresh icon by the API Key marked by (1) then click **Regenerate token** marked by (2):

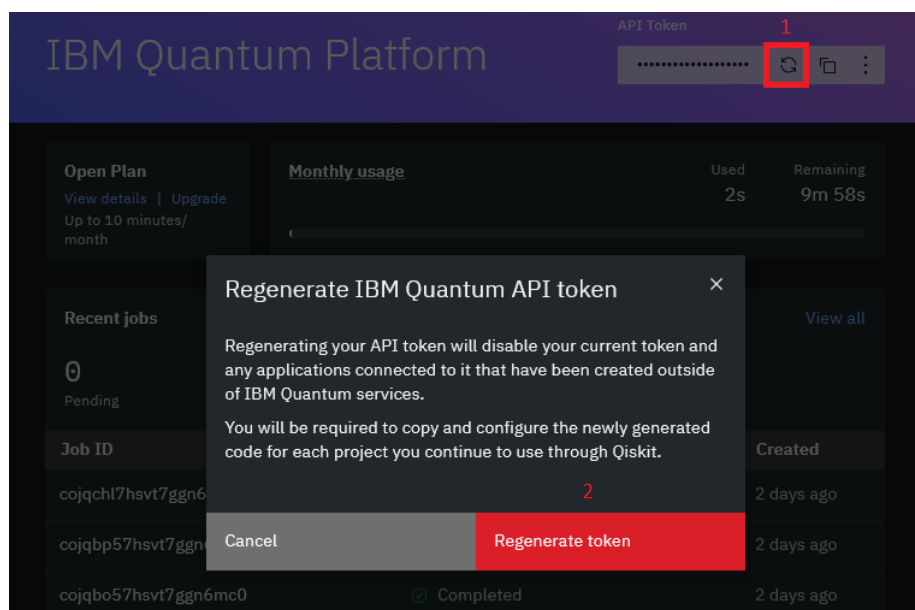


Figure 3.3: Regenerate IBM quantum API token.

After revocation, any unauthorized user is no longer able to utilize the old API key.

3.6 Amazon Web Services Braket

To install the required packages for Amazon Braket run the following command:

```
pip install qiskit-braket-provider boto3 amazon-braket-sdk
```

3.6.1 Getting Access key

First, click on your username (1), then click on Security credentials (2) as shown in Figure 3.4.

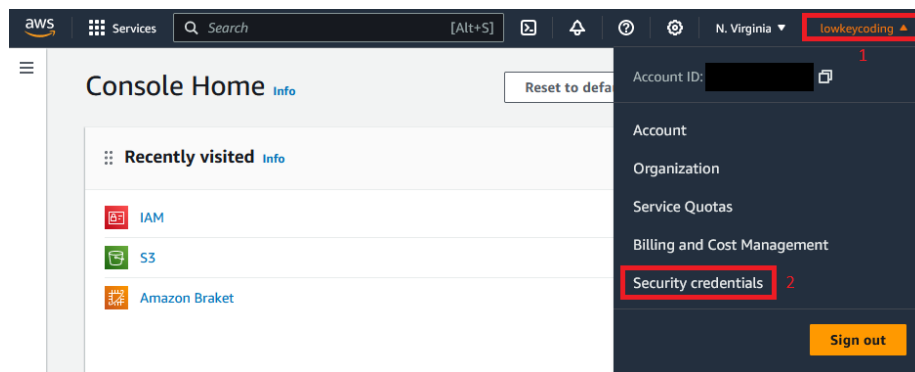


Figure 3.4: Setup AWS braket access key.

Next, click on **Create access key** (3) in Figure 3.5, and follow the instructions. You'll need both the access key and the Secret access key for the next step.

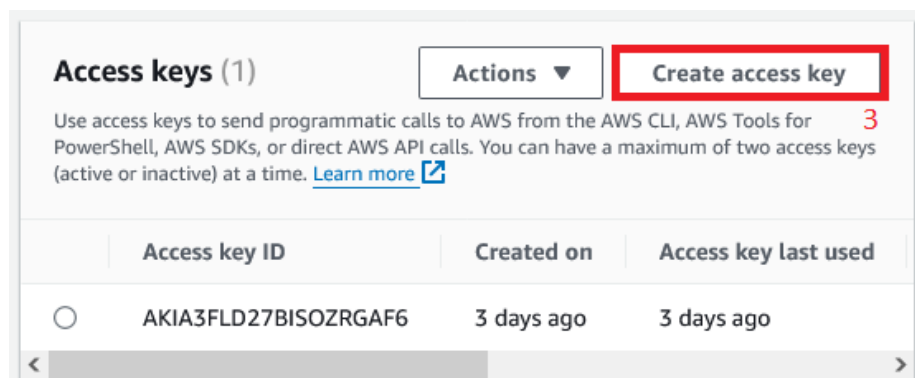


Figure 3.5: Create AWS braket access key.

3.6.2 Login with Python

Setup the environment file `.env` by adding the lines:

```

1 aws_access = Your Access key
2 aws_secret = Your Secret access key
3 aws_region = Your Region

```

Then the minial login script below should provide an AWS session used to access Amazon Web Services Braket backends.

```

1 from braket.aws.aws_session import AwsSession
2 import boto3
3 import os
4 from dotenv import load_dotenv
5
6 # Load environment variables
7 load_dotenv()
8
9 boto_session = boto3.Session(
10     aws_access_key_id=os.environ['aws_access'],
11     aws_secret_access_key=os.environ['aws_secret'],
12     region_name=os.environ['aws_region'],
13 )
14 session = AwsSession(boto_session)

```

To list the currently available backends, include the following snippet in the login example:

```

1 print("This workspace's targets:")
2 for backend in provider.backends(aws_session = session):
3     print("- " + backend.name)

```

3.6.3 Running a quantum circuit

To run a quantum circuit, a specific backend has to be chosen from the list of currently available ones. This is done by changing the backend on the line shown below in the [sample script](#).

```

1 backend = provider.get_backend("SV1", aws_session = session)

```

3.7 Aer

To install the required packages for the Aer simulator run the following command:

```
pip install qiskit-aer
```

3.7.1 Running a quantum circuit

Here, you'll find a simple Bell-state quantum circuit, ideal for testing your installation:

```

1 from qiskit import *
2 import numpy as np
3 from qiskit_aer import AerSimulator
4 from qiskit.visualization import plot_histogram
5 from matplotlib import pyplot
6
7 circ = QuantumCircuit(3)
8
9 circ.h(0)
10 circ.cx(0, 1)
11 circ.cx(0, 2)
12
13 circ.measure_all()
14 circ.draw('mpl')
15
16 # Setting a backend
17 backend = AerSimulator()
18

```

```

19 # Transpile circuit to work with the current backend.
20 qc_compiled = transpile(circ, backend)
21 # Run the job
22 # This will cause a pop where you have to authenticate with azure.
23 job_sim = backend.run(qc_compiled, shots=1024)
24
25 # Get the result
26 result_sim = job_sim.result()
27 counts = result_sim.get_counts(qc_compiled)
28
29 # Plot the result
30 plot_histogram(counts)
31 pyplot.show()

```

3.7.2 Running a quantum circuit with noise

Follow the steps outlined to set up the IBM backend (refer to Section 3.5), and then configure Aer using the noise model specific to the chosen IBM backend:

```

1 from qiskit_aer import AerSimulator
2 from qiskit import QuantumCircuit, transpile
3 from qiskit.visualization import plot_histogram
4 from qiskit_ibm_runtime import QiskitRuntimeService
5 from matplotlib import pyplot
6
7 provider = QiskitRuntimeService()
8
9 # Selecting a backend hardware from ibm.
10 real_backend = provider.backend("ibm_brisbane")
11 # Instantiate Aer simulator with hardware backend.
12 backend = AerSimulator.from_backend(real_backend)

```

3.8 Example Quantum Programs

The QuantumSetup [4] repository features branches for each backend, each containing example programs to test your setup and familiarize yourself with Qiskit. As of (Wednesday 27th November, 2024), the available example programs include:

- Grover’s Algorithm.
- BB84 Quantum Key Distribution.
- Bernstein-Vazirani Algorithm.
- Quantum Approximate Optimization Algorithm for Max Cut.
- Quantum Random Number Generator.
- Shor’s Algorithm.

To access example programs, navigate to the appropriate backend folder within the [QuantumSetup repository](#) and then locate the examples subdirectory. Alternatively, consult the `README.md` file for specific instructions related to your chosen backend.

Chapter 4

UCloud

UCloud (in full name ‘DeiC Interactive HPC Type 1’ but in this text referred to as UCloud) is a shared cloud supercomputer accessible to researchers at the University of Southern Denmark, Aalborg University, and Aarhus University. For this course, it offers a convenient alternative to a local setup. With UCloud, you can access a pre-configured virtual environment with all necessary dependencies directly from the cloud, eliminating the need for local setup. This is particularly beneficial if you encounter issues with your local setup or are using a computer where software installation is restricted.

4.1 Setup

To join the class, click on the invite link, which should look something like "cloud.sdu.dk/app/projects/invite/invite_code". This link will take you to a login page where you can sign in using your university Microsoft account.

Next, make sure you're working in the correct Ucloud project. To do this, click on the workspace selector (shown in 4.1) and choose the workspace for the class, which should look something like:

"ES_Quantum_Information_and_Computing_2024_fall".

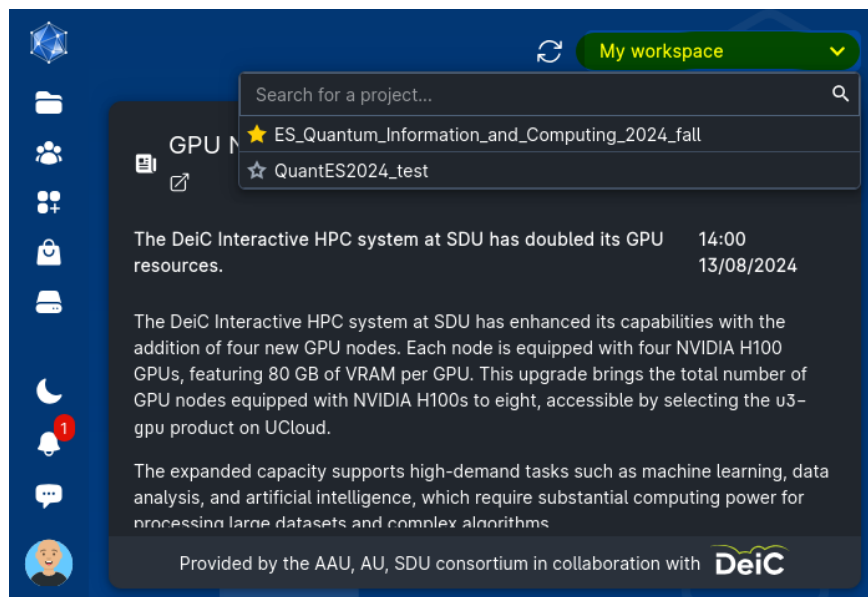


Figure 4.1: Ucloud select workspace

Next, set up your personal drive. Each user will have an automatically created drive that only you and the Ucloud project administrators can access. To create a folder for your project files, hover over the folder icon highlighted in 4.2 and click on "Member Files: <your_name_here>"

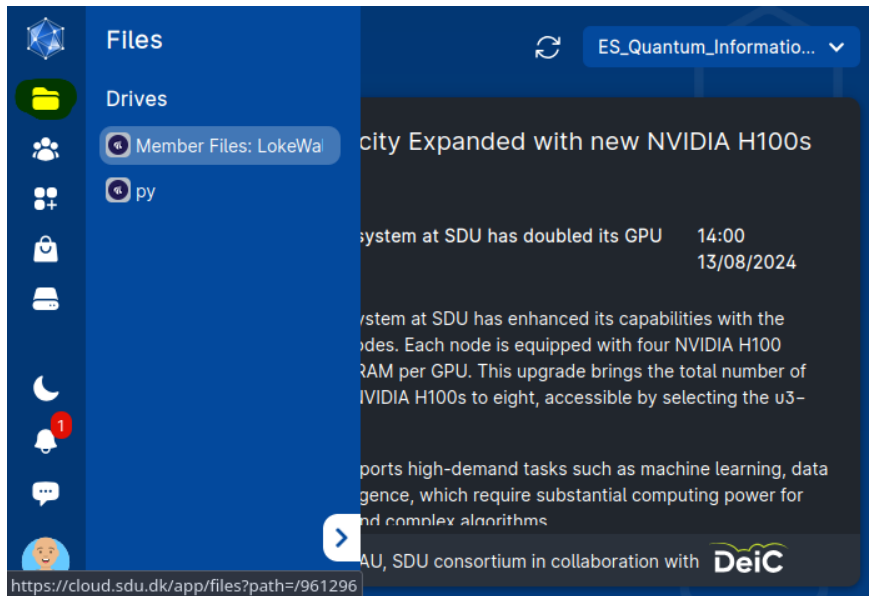


Figure 4.2: Ucloud select drive

Then, click the "Create folder" button highlighted in 4.3 and give the folder a name.

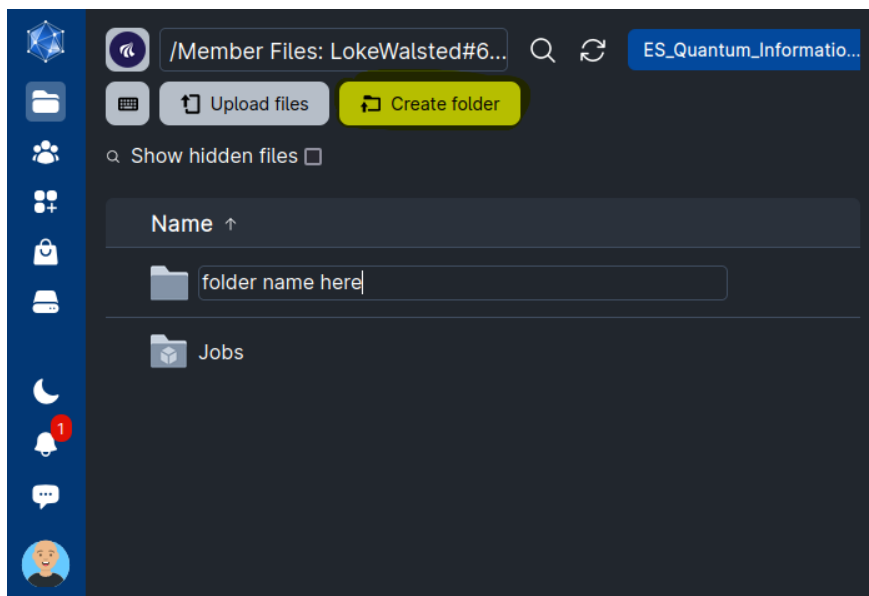


Figure 4.3: Ucloud create folder

Next, head over to the Ucloud coder app by clicking [here](#). Once there, download the "QuantumSetup.json" file [here](#). This file contains the parameters needed to set up your development environment. Next, click the star highlighted in 4.4 with red to add the coder app to your quick access menu by hovering over the

shopping bag icon. Then, click the "Import Parameters" button highlighted in 4.4 with yellow.

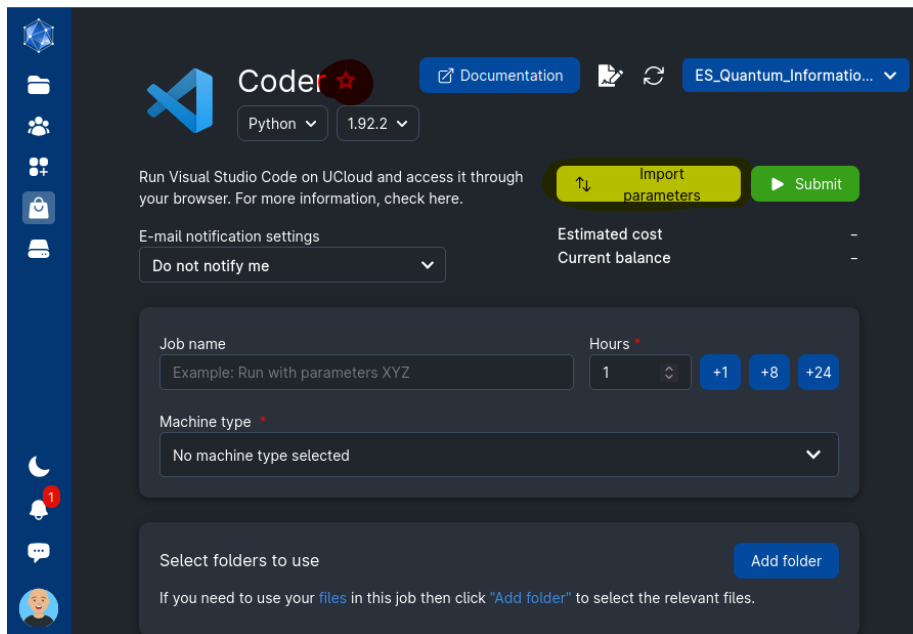


Figure 4.4: Ucloud coder app

Next, click "Upload JobParameters.json" highlighted in 4.5 and select the "QuantumSetup.json" file you downloaded earlier.

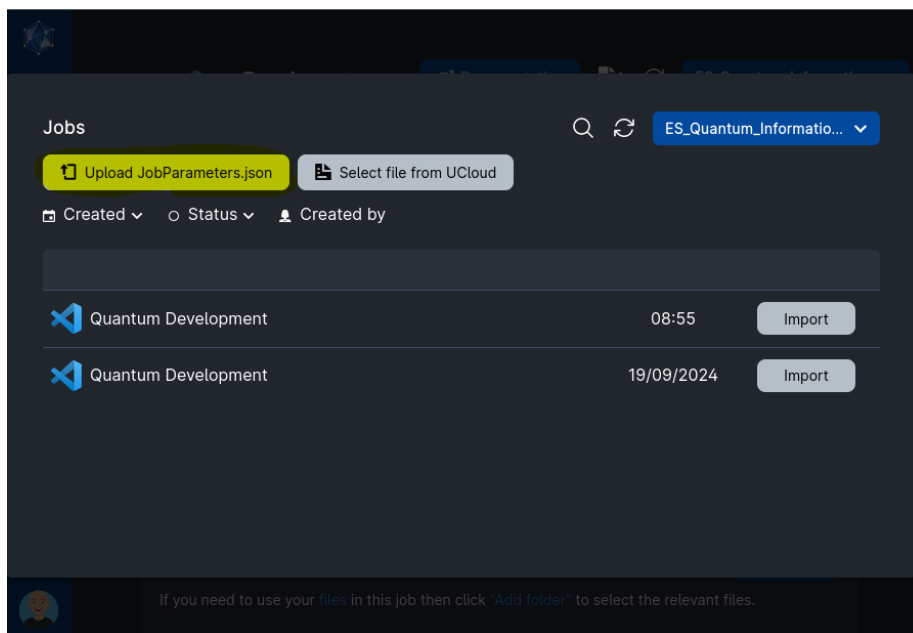


Figure 4.5: Ucloud upload parameters

Next, choose the estimated number of hours you'll be working on the project. Don't worry, you can always increase this later if needed.

Next, click the "Add folder" button highlighted in 4.6 with yellow, and then select "No directory selected" highlighted in 4.6 with red. Choose the folder you created earlier. This folder is used to store all files that need to be saved between jobs.

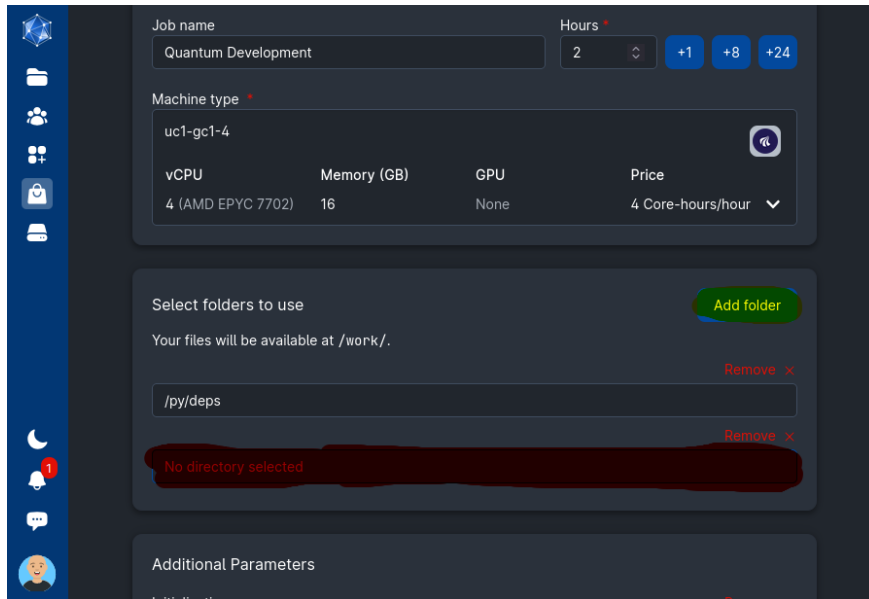


Figure 4.6: Ucloud add folder to job

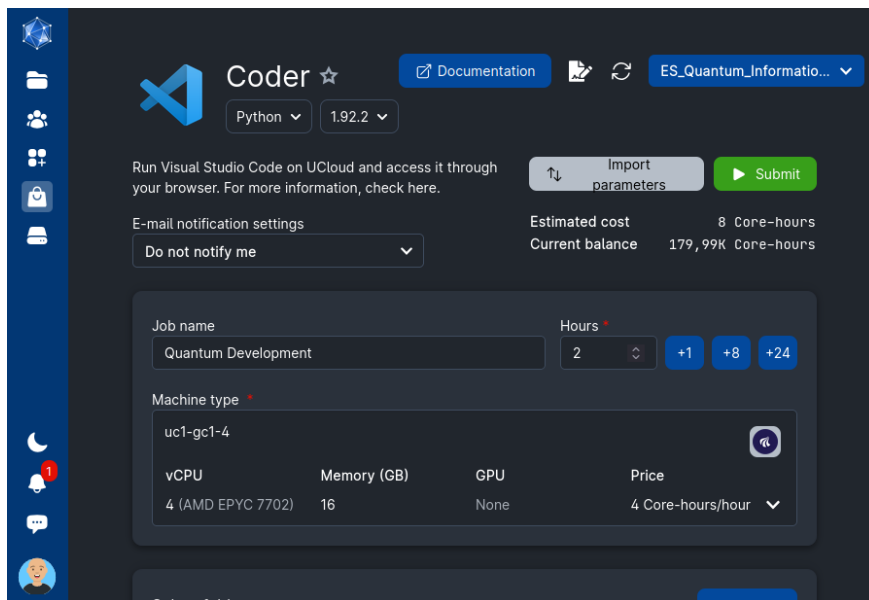


Figure 4.7: Ucloud submit job

Finally, click the "Submit" button highlighted in 4.7 with green to start the job.

The setup process for your development environment should take about 1 minute and 30 seconds. Once it's complete, you can click the "Open interface" button highlighted in 4.8 with yellow to launch Visual Studio Code in your browser. This will provide you with all the dependencies needed to create Qiskit quantum programs, including access to the backends mentioned in the local installation guide. If you'd like to explore some sample programs to test the environment, check out the QuantumSetup repository [here](#). To extend the allocated time for your workspace, use the buttons highlighted in 4.8 with red. These buttons allow you to increase the time by 1, 8, or 24 hours, respectively.

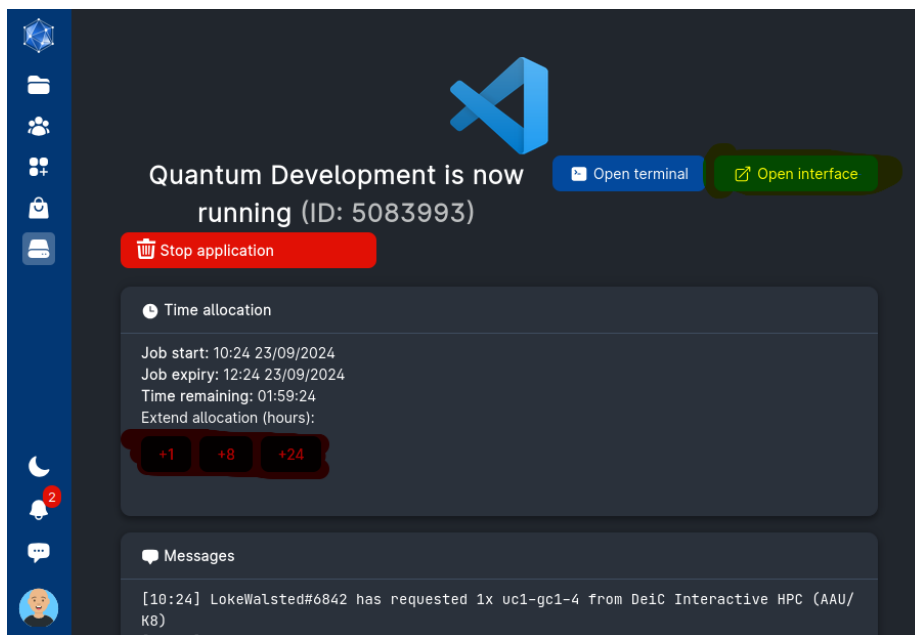


Figure 4.8: Ucloud open/extend job

Bibliography

- [1] Aakash Ahmad, Ahmed B. Altamimi, and Jamal Aqib. *A Reference Architecture for Quantum Computing as a Service*. 2023. arXiv: [2306.04578](https://arxiv.org/abs/2306.04578) [quant-ph]. URL: <https://arxiv.org/abs/2306.04578>.
- [2] Aakash Ahmad et al. *Engineering Software Systems for Quantum Computing as a Service: A Mapping Study*. 2023. arXiv: [2303.14713](https://arxiv.org/abs/2303.14713) [cs.SE]. URL: <https://arxiv.org/abs/2303.14713>.
- [3] Davide Castelvecchi. “IBM releases first-ever 1,000-qubit quantum chip”. eng. In: *Nature (London)* 624.7991 (2023), pp. 238–238. ISSN: 1476-4687.
- [4] Loke Walsted and Torben Larsen. *QuantumSetup*. <https://github.com/LowkeyCoding/QuantumSetup>. 2024.