Q-AIM: A Unified Portable Workflow for Seamless Integration of Quantum Resources

Zhaobin Zhu[1,*], Cedric Gaberle[2,*],
Sarah M. Neuwirth[1,3] Thomas Lippert[2,3], and Manpreet S. Jattana[2]

[1]Institute of Computer Science and Zentrum für Datenverarbeitung, Johannes Gutenberg University Mainz, D-55099 Mainz, Germany
[2]Modular Supercomputing and Quantum Computing, Institute of Computer Science, Goethe University Frankfurt, D-60325 Frankfurt, Germany
[3]Jülich Supercomputing Centre, Forschungszentrum Jülich GmbH, D-52428, Jülich, Germany

Abstract

Quantum computing (QC) holds the potential to solve classically intractable problems. Although there has been significant progress towards the availability of quantum hardware, a software infrastructure to integrate them is still missing. We present Q-AIM (Quantum Access Infrastructure Management) to fill this gap. Q-AIM is a software framework unifying the access and management for quantum hardware in a vendor-independent and open-source fashion.

Utilizing a dockerized micro-service architecture, we show Q-AIM's lightweight, portable, and customizable nature, capable of running on different hosting paradigms ranging from small personal computing devices to cloud servers and dedicated server infrastructure. Q-AIM exposes a single entry point into the host's infrastructure, providing secure and easy interaction with quantum computers on different levels of abstraction. With a minimal memory footprint, the container is optimized for deployment on even the smallest server instances, reducing costs and instantiation overhead while ensuring seamless scalability to accommodate increasing demands. Q-AIM intends to equip research groups and facilities purchasing and hosting their own quantum hardware with a tool simplifying the process from procurement to operation and removing non-research related technical redundancies.

1 Introduction

Quantum computing (QC), currently in its developmental phase, promises substantial acceleration of classical computations across various fields ranging from cryptography to materials science [1–4]. Quantum computing scientists are constantly striving to overcome the limitations imposed by the current noisy intermediate-scale quantum (NISQ) era to fully realize quantum computing's potential.

However, while large private-sector enterprises are advancing the field through their own hardware, software, and algorithmic developments, smaller academic research groups lack direct on-site access to such resources. Although corporations such as IBM [5], Google [6], and Amazon [7] offer access to their own or hosted third-party infrastructure on a pay-to-use basis over the cloud, fundamental research is limited by restricted privilege policy and physical inaccessibility. Consequently, the acquisition of small-scale devices emerges as a viable solution to delve deeper into hardware and software enhancement studies, especially since the devices are getting cheaper. Yet, a critical challenge remains: the lack of a portable, open source, and easily integrable software solution for small-scale hardware integration and provision.

[*]These authors contributed equally to this work

Ultimately, procuring quantum hardware serves not only to enable deeper interaction with the device but also to facilitate its utilization on an abstract software level. This requires granting access to the resource over the host's network infrastructure, and possibly even beyond that, by a service either hosted in the cloud or also on-premise, dependent on the requirements and capabilities. For instance, a device could be made accessible to external users, such as students, for educational purposes or to demonstrate advancements to a broader audience. Yet, the absence of a common, open-source integration platform forces researchers to spend valuable time and expertise developing such a solution on their own. Such efforts can detract from their primary focus of advancing scientific knowledge. A flexible, streamlined, and universally adaptable integration software is therefore crucial, not only to eliminate redundancies but also to ensure compatibility with existing workflows.

This work presents Q-AIM, a flexible, streamlined, and universally adaptable quantum integration workflow designed to address key challenges in quantum resource utilization, particularly for small enterprise and academic research groups. Typically, quantum systems are equipped with peripheral classical hardware providing a hardware- and vendor-dependent interface to the quantum computer, facilitating their use on an abstract software level. But, without a standardized, open-source platform, researchers face significant hurdles in integrating quantum systems into existing workflows. To eliminate redundancies and enhance compatibility of the necessary integration software solution, we make the following contributions:

- ***Unified and Portable Platform***: A Docker-based, microservice architecture ensures seamless deployment and scalability across various infrastructures e.g, on a local machine, server, and cloud [8].
- ***Flexible Access and Control***: Offers resource access via multiple abstraction levels (from algorithmic to pulse-level) with a role-based permission scheme for secure and tailored utilization among diverse user groups.
- ***Classical Workflow Integration***: Standardized and flexible APIs enable easy hybrid computing, reproducibility, and cross-institution collaboration without major infrastructure changes.
- ***Prototype Validation***: A lightweight prototype demonstrated adaptability and efficient resource usage across on-premise and cloud infrastructures, supporting broad research and educational application possibilities.

2 Background and Related Work

Quantum computing offers great potential, but the integration of quantum hardware into classical workflows faces major challenges due to proprietary systems and lack of standardization. This chapter provides a brief overview of quantum instruction workflows and existing integration solutions.

2.1 Quantum Instruction Workflow

The instruction of any quantum hardware involves multiple steps, based on the level of abstraction, to match the desired logic to the operations exposed by the architecture. A stepwise workflow example from high-level algorithm definition to device-specific machine instructions is shown in Fig. 1, which includes the abstraction levels commonly used in quantum computing.

The first step in every quantum execution is the definition of the algorithm as a quantum circuit (*Circuit Definition*) making use of any arbitrary quantum operations in an abstract high-level software framework. Commonly used ones include Qiskit [9], Cirq [10], and Braket [11], provided by IBM, Google, and Amazon respectively. This is the same principle as writing algorithms and using resources in classical computing, i.e. abstract away all hardware constraints and focus solely on the implementation of the desired logic.

The closest quantum computing equivalent to the compilation is *transpilation*, with an intermediate representation (IR) comparable to the executable [12]. The transpilation introduces some form of optimization (*HW-Independent IR*) and most importantly takes hardware constraints into account to fit the abstract level implementation to a backend (*HW-Dependent IR*) [13]. The latter is due to the native gate-set exposed by the quantum processor architecture, i.e. the logic of the high-level implementation must be represented using only a limited set of universal gates natively supported by the hardware in question, usually a few

single-qubit rotation gates and a single two-qubit gate. BQSKit (Berkeley Quantum Synthesis Toolkit) [14] is an exemplary compiler framework to perform optimization and gate-set transpilation combining state-of-the-art algorithms in a stand-alone, end-to-end solution to reduce the depth of the quantum circuit. A more extensive comparison of quantum software development kits and compilers can be found in Ref. [15]. Like the binary in classical computing, the IR can be of different formats. One of the most commonly used formats is OpenQASM [16, 17], a quasi-assembly language for quantum computing.

As in classical computer science, assembler is not yet an instruction at machine level, but is used to communicate with remote resources if used. Therefore, the last step of instruction modification is the translation to machine code (*Machine Instructions*). In quantum computing, this oftentimes means microwave pulse modification where specific pulses modify the state of the quantum system likewise to an instruction in the high-level abstraction implementation.

2.2 Accessing Quantum Resources

For users to be able to execute algorithms on real quantum hardware, two parts must be taken into consideration. As depicted in Fig. 1 and discussed in the previous section, the first involves the transpilation of any algorithm or circuit to the underlying hardware in use. Usually, algorithms are developed and translated into quantum formalisms in a hardware-agnostic fashion. The high-level code abstraction must be transformed into a device-specific instruction set.

The second part addresses with the access of the resources. As mentioned before, quantum computing providers often grant access to their devices over the cloud using an API interface. Therefore, accessing quantum computers as a form of highly specialized compute resource does not differ from accessing any other remote compute resource. Providers will most likely impose restrictions in terms of supported IR formats and the ways to interact. Invariably, they implement stringent security measures to safeguard their resources, mandating users to authenticate themselves via provided credentials and implement different levels of permissions. Typically, access to the protected resource is facilitated through an API, overseeing the bidirectional flow of data to and from the resource. In the workflow diagram depicted in Fig. 1, the API call can happen at any level between the circuit definition and the machine instructions, depending on the services provided by bespoken API. The data flow from the resource to the user happens after execution, containing the result of the computation in a pre-defined format dependent on the provider.

2.3 Analysis of Related Work

Recent efforts aim to integrate quantum devices into classical computational resources, either by allowing them to be accessed during computation in a manner similar to GPUs (indirect access) [18–20] or through an API service (direct access) [10, 11, 20–25]. Ruefenacht et al. [19] provide a fundamental overview of different integration architectures, from loosely- (on-premise) to tightly-coupled (on-chip), for quantum processors as accelerators for specific high-performance computing (HPC) workloads. Schulz et al. [18] particularly emphasize the necessity of developing an unified software stack to harness the strength of both radically different systems for the latter time and space shared integration scheme. However, Humble et al. [20] state in their analysis of the different possibilities of integrating QC and HPC that "existing QC prototypes are based on loosely integrated client-server interactions that lack the sophistication or technological maturity to be used as accelerators". But exactly these early prototypes are of particular interest for fundamental research done by academic groups and the center of possible application of the software proposed in this paper alike. Therefore, the focus for now is on the loose integration of (less sophisticated) quantum hardware. Such a solution in a corporation independent and open source fashion is the missing key to enable small scale device utilization and integration for academic purposes. This paper addresses the development of such a solution: a software platform that enables secure, low-overhead, and flexible access to quantum devices on different environments, such as on-premise and/or cloud infrastructures. Delivered as a Docker container, it ensures ease of deployment, hardware independence, and extensibility, creating a foundation for efficient and portable quantum computing research.

Different research studies [26–28] focusing on providing quantum computing as a service use dockerization for the same purposes. The services provided are usually some quantum application/algorithm, which abstract away the complexities of the hardware, simplify development and deployment, and allow for portability

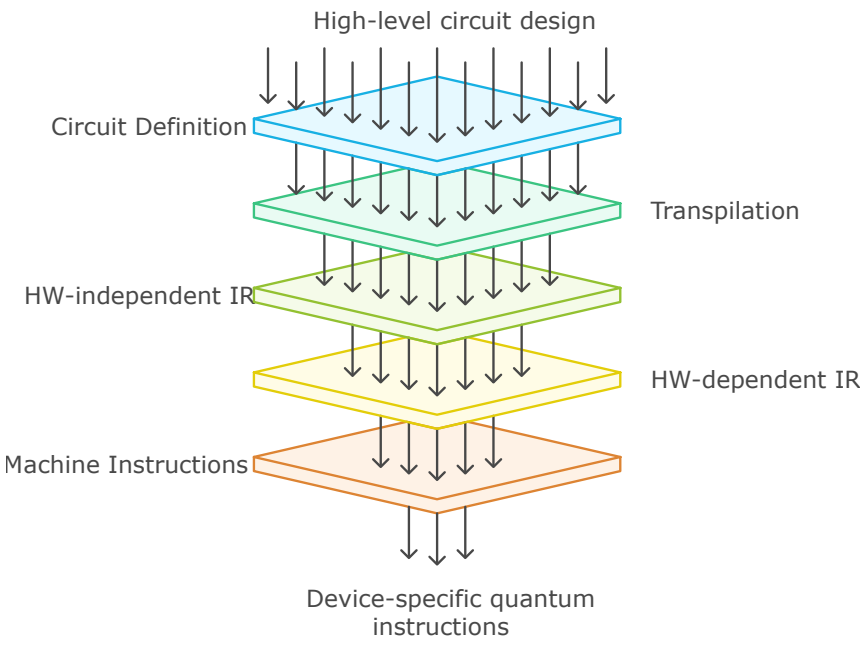

Figure 1: Instruction abstraction levels in quantum computing. From high-level circuit design (highest abstraction) to hardware-independent and hardware-specific intermediate representations (IR), ultimately becoming device-specific machine instructions (no abstraction) to be used with the quantum device.

across different platforms. Grossi et al. [28] proposes a decoupled function-as-a-service (FaaS) quantum computing integration, exposing a single HTTP API endpoint to provide an arbitrary quantum service, which is offloaded to the IBM backend, thereby separating provision and usage. Nguyen et al. [27] expand on the idea, pointed out the vendor lock-in effect and extended the capabilities of the serverless model to different backend platforms. Overall, the goal is a higher abstraction for ease of use in enterprise scenarios. This work addresses the exact opposite. We provide services also to be hardware-agnostic, portable, and flexible, but with the main purpose of allowing fine-grained access to the device on levels of less abstraction, e.g. interaction on pulse-level. Researchers are dependent on a low-level access since the focus is not on the usage through quantum algorithms as a service but on being able to accurately manipulate certain workflows and interactions in the pursuit of knowledge extraction.

During the time of writing, a paper [29] emphasizing the necessity and challenges of a hardware-agnostic framework integrating hardware, software, workflows, and user interfaces, also with particular focus on academia to foster a synergistic environment for quantum and classical computing research was published. The main difference to the work in this paper is that they present developments in software for a hybrid quantum computing approach integrating quantum computers as accelerators to complement high performance computing systems. These advancements also primarily target large academic institutions already running sophisticated HPC systems as a form to further improve computations of the specific field of interest. Our work focuses on enabling small (research) groups to integrate, directly access, and make own quantum devices publicly accessible in an unified fashion, streamlining the process after procurement until usage, e.g. investigative research.

3 Design Consideration

As described in Section 2, the integration of quantum computing resources into existing research and industrial workflows requires careful orchestration of software across multiple domains. However, today's quantum computing solutions are often associated with specialized hardware and proprietary environments, limiting their applicability to on-premise installations and creating major barriers to entry. This is a significant

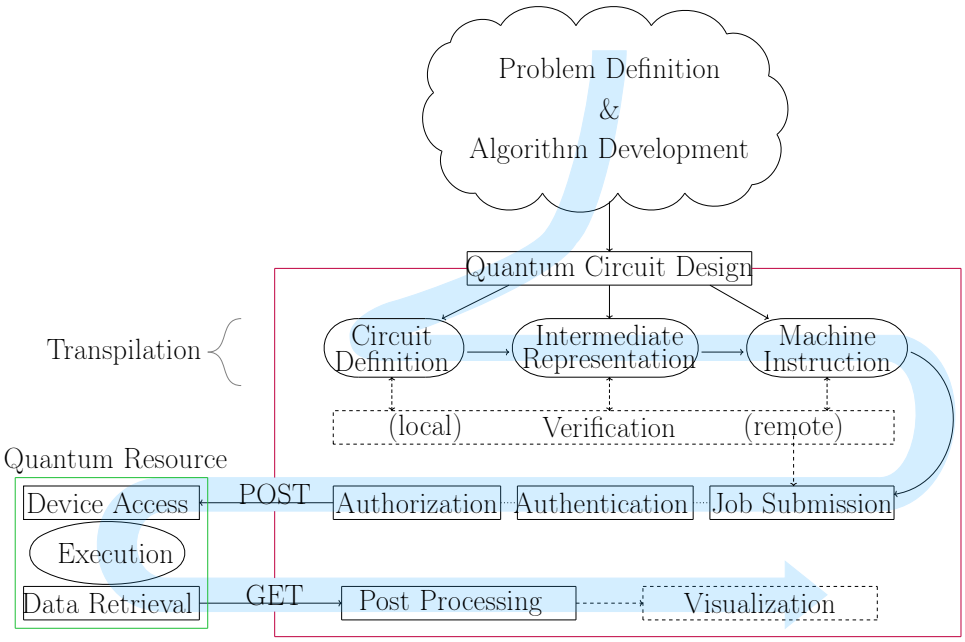

Figure 2: Overview of the quantum computation workflow. The larger, red box (right) indicates classical hardware, the smaller, green one (left) the quantum system. The problem is defined as quantum circuit at any abstraction level and undergoes transpilation until machine instruction level is reached. Communication between the classical and quantum system is facilitated through API calls.

obstacle, especially for smaller institutions and educational facilities.

To address these challenges, we propose Q-AIM, a standardized, portable workflow and a corresponding software implementation that enables seamless integration of quantum resources into existing infrastructures. As shown in Fig. 2, the required process steps are highlighted in blue, the involved services on the classical hardware side by the red box and the vendor-dependent quantum resources by the green box.

With Q-AIM, quantum circuit design can be performed at various levels of abstraction. We also introduce an optional simulation step that serves as a verification phase before running computations on potentially busy remote quantum computers. All services indicated in the red box are abstractions that can be made available within the proposed software. Rather than enforcing a single standardized software framework, we allow each component or service to be replaced or customized, thereby giving both users and hardware providers granular control over implementation details. This flexibility accommodates diverse requirements and preferences. Because there are multiple execution environments and a wide range of components or services, each with distinct access control needs, we rely on well-established Representational State Transfer (REST) APIs, using common calls such as *POST* and *GET* to ensure broad compatibility across different software languages, platforms, and hardware architectures.

Another key design consideration is the decoupling of quantum hardware, ensuring that the software layer operates independently of specific hardware, thereby minimizing manufacturer dependency. Consequently, tasks such as qubit manipulation, machine instruction execution, and qubit state measurement are handled directly by the quantum hardware and its peripherals. Together with the microservices, these components form the backend for Q-AIM.

4 Methodology

The key principle of the proposed approach is to ensure that classical workflows remain largely unaffected by the introduction of quantum computers. Instead of having to rebuild or heavily modify pre-existing computational frameworks, end-users can embed quantum tasks and pipelines into their established processes. With this design approach, a quantum resource can be quickly adapted with minimal changes, while maxi-

mizing the advantages of quantum computing. The effectiveness and versatility of the proposed system are underpinned by four core methodologies:

- Fully Integrated Classical Workflow
- Encapsulated System Architecture
- Micro-Service-Based Software Architecture
- Flexible and Fine-Grained User Management

4.1 Fully Integrated Classical Workflow

Based on the design consideration as mentioned in Section 3, the classical quantum computing workflow, i.e. the steps from algorithm definition to machine instruction, is a multi-stage process that spans tasks from algorithm development to analysis and optimization. Depending on the manufacturer and application scenarios, the code often needs to be compiled into an appropriate representation, such as gate-level or pulse-level instructions, to execute on a quantum computer. To allow users to operate at different levels of abstraction, it is crucial to account for these variations during the integration workflow.

To support this flexibility and maintain vendor independence, the entire classical quantum computing workflow is treated as a black box and integrated as a unified entity within our infrastructure. This abstraction ensures seamless interaction between the classical and quantum workflows without requiring users to manage low-level specifics or adapt to API changes, thereby enhancing usability and interoperability. Therefore, the classical workflow is incorporated into our integration pipeline as a self-contained component and augmented with additional functionality. These functionalities range from custom user management, authentication services, and access control to result visualization and system monitoring. This approach allows users to work with different programming languages at different levels of abstraction while taking advantage of the unique features of different quantum hardware backends. It also supports adaptability to emerging quantum computing platforms, ensuring that the architecture is future-proof.

4.2 Encapsulated System Architecture

To enable a standardized and transparent quantum computing workflow, we rely on an encapsulated system architecture that decouples the software layer from the underlying quantum computing hardware. This architecture acts as an abstraction layer that simplifies and hides the complexity of the individual components. As shown in Fig. 3, the system is divided into two key segments: *the Q-AIM software* and *the quantum computing hardware*. Given that the Q-AIM framework provides users with exclusive access to quantum resources via various microservices (see 4.3), the central component of this system architecture is the API gateway, which serves as the primary entry point for users. It abstracts the underlying microservices and prevents direct access or communication between clients and service components. This isolation significantly simplifies implementation for both clients and microservice applications, as the complexity of the application is decoupled from its clients. Another important element is the Reverse Proxy. The API gateway can be understood as a superset of the reverse proxy and offers additional functions that go beyond the simple forwarding of requests. The reverse proxy assigns the physical ports to those of the encapsulated environment and acts as an intermediary that communicates with the server on behalf of the client(s), forwards requests and returns responses. The proxy is located at the edge of the API gateway, which centralizes the processing of API requests and enforces additional security policies such as authentication, authorization and access control, as well as other functions not covered by the microservices.

As a result, the entire architecture is generic, portable, and easily extendable. It provides a standardized way for those to communicate, interact, and be managed within the environment, thus allows for modularity, scalability, and adaptability, making it possible to integrate the services seamlessly while maintaining a consistent and manageable architecture.

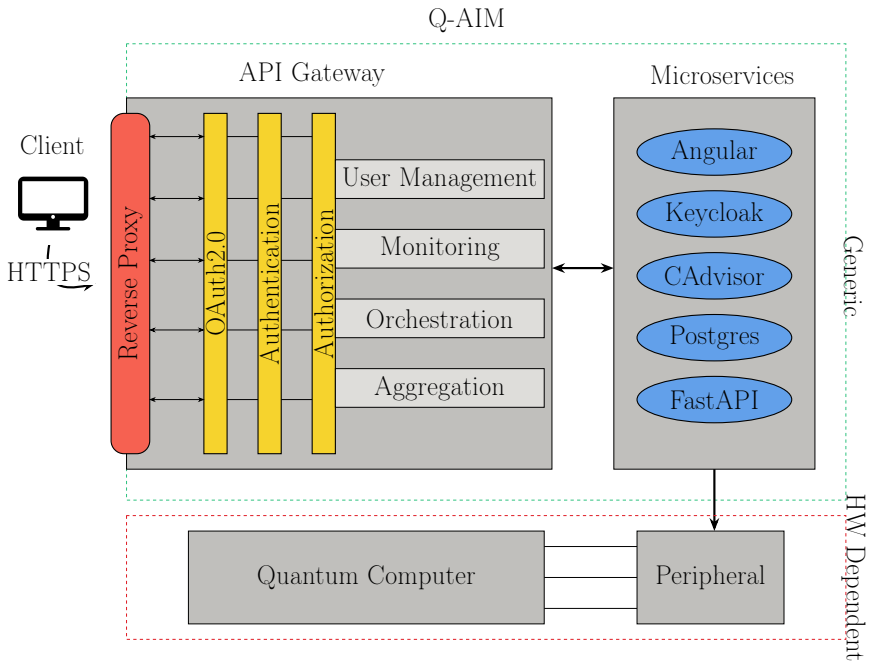

Figure 3: Microservice-based architecture of Q-AIM. It facilitates secure client interactions via HTTPS and a reverse proxy, providing access to quantum systems through a structured microservice architecture. The API Gateway manages authentication, authorization, and orchestration, while the microservices provide the software's functionalities.

4.3 Micro-Service-Based Software Architecture

To meet the challenge of a standardized, portable integration workflow, in this work we develop a microservice-based software architecture that enables quantum computing hardware to be integrated into existing and future infrastructures in a consistent manner. A key aspect for Q-AIM is therefore portability and transparency.

To achieve this, the software requires the ability to run in a virtualized and isolated environment. Lightweight virtualization technologies, i.e. containers such as Docker [30] or Apptainer [31], are highly portable, which means that they can be easily run on different operating systems and infrastructures. The isolated nature of container virtualization also ensures that all required dependencies are bundled in the container and services can be quickly deployed and replicated on different hosts. As container-based software deployment is typically based on a microservice architecture, the functionality of the software can be customized and extended according to user-specific requirements. This gives Q-AIM greater versatility and adaptability, which is beneficial for research institutions and companies alike.

Overall, Q-AIM's microservices-based architecture not only reduces the dependency on specific vendors, but also allows researchers and developers to transfer and scale their work to different environments [32]. This is particularly important for reproducibility and enables the building of a community that promotes the exchange of ideas, best practices and resources to further advance the development of quantum computing technology.

4.4 Flexible and Fine-Grained User Management

Another key challenge is managing access from different environments with corresponding user affiliations. Users can generally be categorized into internal and external groups, each requiring specific levels of access to quantum resources. For example, a physicist conducting physical experiments on a quantum computer needs easy access to enter signals or waveforms. In contrast, users from business or other fields usually require high-level access to test their algorithms or circuits on the quantum computer.

```
services:
    database:
        image: postgres
        ...
    authentification:
        depends_on:
            - database
        image: jboss/keycloak:11.0.3
        ...
    Q-AIM-API:
        image: fastapi:dev
        ...
    Q-AIM-Frontend:
        image: Q-AIM:dev
        ...
    reverse-proxy:
        image: nginx:alpine
        ...
    monitoring:
        image: gcr.io/cadvisor/cadvisor:latest
        ...
volumes:
    ...
```

Listing 1: Overview of the microservices and their images in the docker compose file.

To enable fine-grained access control to quantum resources and flexible user management, it is essential to integrate different user groups into a single infrastructure, manage them effectively and meet their different access requirements. This requires the integration of the industry standard LDAP [33] protocol into our solution for authenticating internal users. In addition, the system should support the creation and management of a special user database for external users to ensure seamless integration and secure access for all user types. As interaction with quantum computing resources takes place exclusively via the API gateway, Q-AIM enables authentication for different user groups and supports fine-grained authorization, ensuring that users can only interact with the resources that correspond to their assigned roles.

5 Prototype Implementation

In the following, we present an early prototype implementation of our portable, unified, and generic quantum computing integration workflow. The integration of self-written or third-party libraries as a service in the example implementation of our microservice architecture underlines the aforementioned adaptability. Similarly, other entities can implement different services specific for their use cases.

5.1 Container-based Deployment

From the high-level system architecture shown in Fig. 3, it is clear that deploying the Q-AIM application requires a complex environment with a number of microservices working together. To improve transparency and portability in the deployment process, Docker containers are used to ensure consistency. Also, a Docker Compose file is used to simplify the management of multiple microservices and their dependencies within the application. Consequently, this approach facilitates the deployment of the entire application environment with a single command, i.e. *docker compose up*.

To provide an overview of the main services of Q-AIM, as shown in Listing 1, the services are described below:

- **Database Service**: This initiates a PostgreSQL database utilizing the official Postgres Docker image [34]. To ensure persistent storage of the database data, a Docker volume is created alongside.
- **Authentication Service**: Utilizing the official Keycloak Docker image [35], this service delivers identity and access management functionalities. It relies on the database service and necessitates a Keycloak

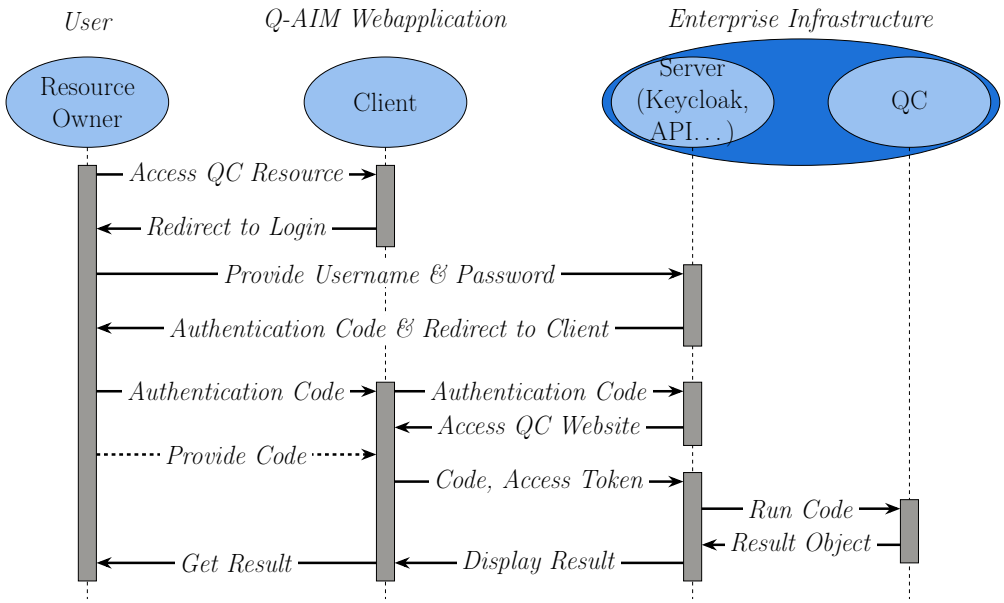

Figure 4: Representation of the first authentication process for an approved user attempting to run code on a protected quantum computing resource.

configuration file. For illustrative purposes, environment variables for the Keycloak administrator user, password and other settings are also configured via the docker compose file.

- **Q-AIM-API Service**: This employs the Docker image fastapi:dev and is built using a custom Dockerfile, which sets up an environment tailored for a FastAPI [36] application and installs specific dependencies.
- **Q-AIM-Frontend Service**: Built upon the Q-AIM:dev Docker image using a custom Dockerfile, this Dockerfile ensures that the actual Angular Web-Application [37] is built in a Node.js [38] environment and then the resulting build is deployed within an NGINX [39] container. The NGINX container is used to serve the static files of the Angular application and provide the configuration for the web server.
- **Reverse Proxy Service**: Based on the Docker image nginx:alpine, this service initializes an NGINX proxy server. Configured with a corresponding configuration file and SSL certificates, the proxy server forwards incoming requests to various services provided within Docker containers.
- **Monitoring Service (Optional)**: Leverages the official CAdvisor Docker image [40] to efficiently gather and present container statistics. To facilitate access to files or directories within the host system, it is imperative to include relevant directories or files from the host within the container. For instance, directories such as /sys/ need to be mounted for this service.

Overall, the division of microservices illustrates the basic principles of modern software development and architecture. This approach promotes customizability, scalability, security and reproducibility in application deployment. By using Docker and Docker Compose, both developers and professionals can seamlessly adapt Q-AIM to their specific requirements and deploy it efficiently in their infrastructure.

5.2 Authentication Workflow

An exemplary workflow accessing a quantum device as protected resource is depicted in Fig. 4. During the user's initial access, they are required to provide their credentials. Only after the identity and access management tool Keycloak validates the provided credentials and returns an authentication code, including an access token holding information about the authenticated user's roles and permissions, an ID token with

general information about the authenticated user, and a refresh token, does the user gain access to the quantum computer frontend component on the Angular webapplication. Provided quantum code of the user on the frontend component serves as input data to the API endpoint managing access to the protected quantum resource. The API therefore validates the provided authentication code at the identity and access management tool and checks the user's permissions in the access token. If the user is permitted, it controls the bidirectional flow to and from the quantum resource. Lastly, the result is displayed on the frontend web application.

5.3 Q-AIM User Interface

The Q-AIM frontend serves as a user-friendly gateway to access quantum computing resources. To safeguard the underlying endpoints and enable fine-grained permissions management, integration of the Keycloak service and authentication functionality has been embedded within the Angular application. As can be seen from the Fig. 5 ①, users must be authenticated to access certain resources and have certain permissions. Furthermore, the authorization framework's distinction between groups and roles facilitates the assignment of users to various domains, institutions, and systems, allowing for the allocation of grouping-specific roles. To exemplify the granularity of rights management, the prototype establishes two groups, i.e., internal and external and each featuring user or admin roles. Depending on whether the user is already authenticated via the authentication server, the user is either redirected to the login page to process the authentication workflow as shown in Fig. 4 or to the interface for the corresponding compute resources, as shown in Fig. 5.

A standardized user interface ensures a seamless workflow for accessing different backend functionalities. As can be seen in ②, the resource utilization of the respective quantum resource is displayed. ③ shows, Q-AIM currently supports OpenQASM source code or Pauli representation, a format introduced in [41] and parsed into OpenQASM by a library made available as binding at [42], as input. The Pauli representation takes advantage of the fact that the Pauli rotations together with the controlled-NOT (CNOT) operation form a complete basis set, which means, every computation can be represented using appropriate Pauli and CNOT operations. This not only shortens but also simplifies the code input, enhancing its portability. An exemplary operation in Pauli representation can be seen in Listing 2, with the Pauli operators of the unitary given as string, characters and index corresponding to the respective Pauli or Identity operation on a specific qubit, a coefficient, and a variable indicating optional parameters. The same circuit in OpenQASM format is also partially displayed and used in the example run depicted in Fig. 5 ③. Since Qiskit simulators are used in this work for demonstration purposes and many devices accept OpenQASM as IR, the library converting the Pauli representation into OpenQASM is part of the dependencies for the API microservice and ships with the image by default. Users have the option of either entering their code via the live editor or selecting the corresponding file and uploading it.

As many circuits performing the algorithm's desired computation need to be parameterized, users must be able to provide the parameters. They can do so either using a dictionary, naming the specific variable to be set and its value, or as a list (array), only providing the variables' values which are then assigned in order of appearance in the circuit. This provision is done on the webpage shown at ④. A prominent example of

```
IIXY 1. 1
IXIY 1. 2
IXYI 1. 3
XIIY 1. 4
XIYI 1. 5
XYII 1. 6
IIYX 1. 7
IYIX 1. 8
IYXI 1. 9
YIIX 1. 10
YIXI 1. 11
YXII 1. 12
```

Listing 2: Exemplary ansatz Pauli representation. Operators encoded as strings (left), coefficient (middle), and parameter named as number (right).

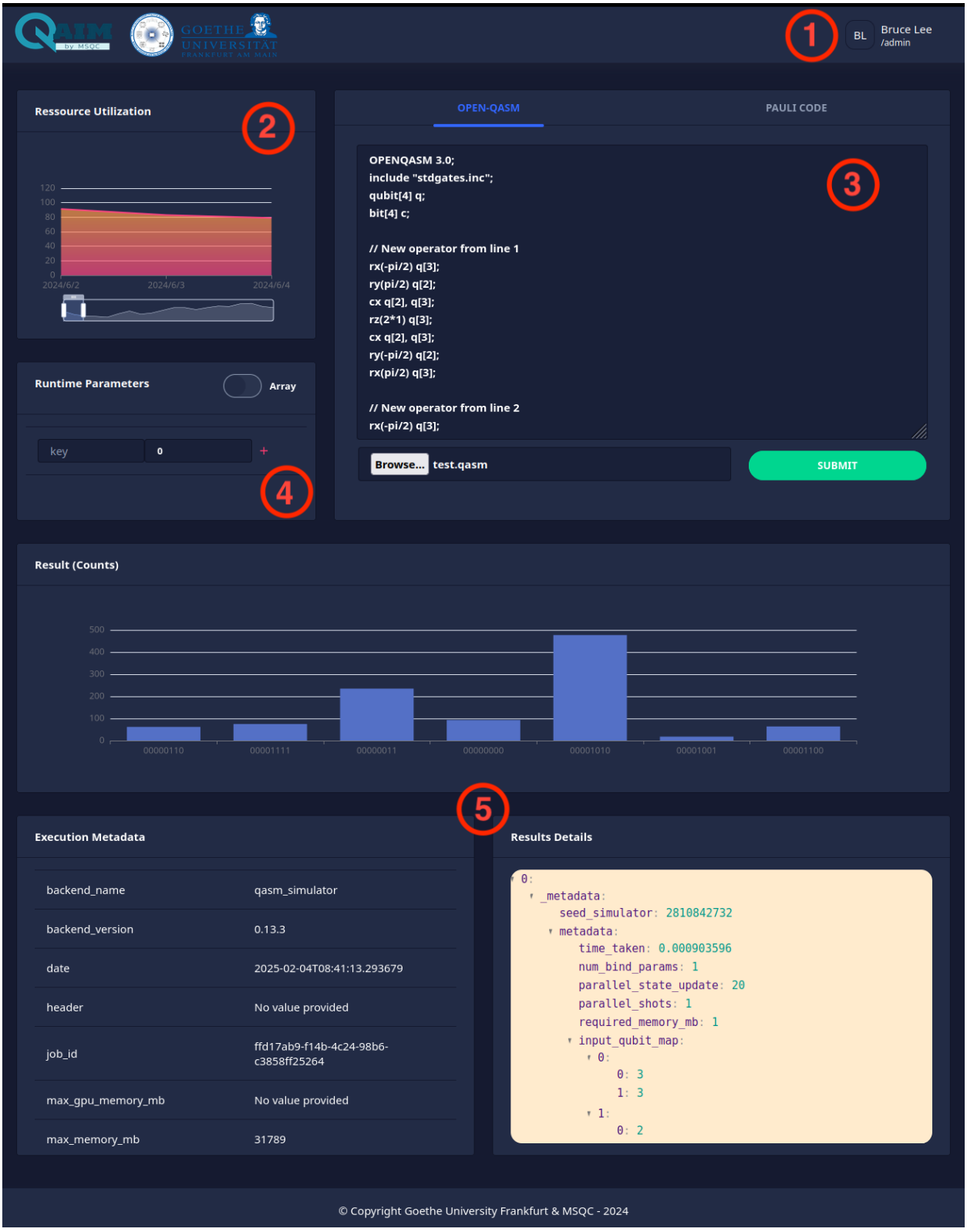

Figure 5: Q-AIM Web User Interface. Users can provide code and runtime parameters in different formats, monitor resource utilization, and visualize results and metadata.

an algorithm necessitating parameterization is the Variational Quantum Eigensolver (VQE) [43–45]. Since parameter optimization is hardware-dependent, a set of optimized parameters obtained on one quantum device cannot be directly fixed into the circuit while ensuring reproducibility across different hardware. However, these parameters can still serve as a good initialization point, reducing the optimization effort on other devices. Therefore, the optimized parameters are included in the result object.

After submission, the provided code is executed via the API on the hardware-specific backend. Following successful execution, the resulting data and metadata are visualized as interactive diagrams or JSON objects as shown in ⑤ of the user interface, with the option of downloading them as CSV files or image files.

5.4 Q-AIM API

The Q-AIM API is designed to handle a variety of requests related to both quantum computing tasks and user-specific operations. It is developed using Python and the FastAPI framework and serves as the backbone for processing tasks. Since real quantum hardware is not available for testing, the API utilizes simulators to query as endpoints instead, with the Qiskit library employed for quantum computing task execution using its Qasm Simulator [46], a noisy quantum circuit simulator backend.

Primarily, an API comprises public and private endpoints. Public endpoints are accessible without requiring authentication, enabling direct access to the endpoints. Conversely, protected endpoints necessitate authentication via a Json Web Token (JWT), issued by Keycloak, for example. Authentication is facilitated through an authentication function *auth()*, assigned to endpoints requiring authentication as a dependency function using FastAPI's own dependency resolution mechanism. The function issues the query to the identity management using an OAuth2.0 scheme, as described in Section 5.2. For this work, only private endpoints are used to showcase the finely granulated permissions management. These include the endpoint /api/user/me, which retrieves information about the authenticated user. Furthermore, access to endpoints responsible for quantum computing is restricted to authenticated users with appropriate permissions. For illustrative purposes, the prototype offers four more endpoints: for uploading and processing OpenQASM code (`/api/qc/qasm/{upload, code}`), one for each uploading a file and coding on the web page, and the same for code in Pauli representation (`/api/qc/pauli/{upload, code}`). The calculated results are subsequently returned to the Q-AIM frontend as part of the response.

6 Evaluation

In the following, we present an evaluation of the integration workflow's key attributes, focusing on its portability and lightweight nature, designed to seamlessly integrate with diverse computing environments. We examine these aspects using different combinations of hardware, software, and hosting paradigms in the following.

6.1 Test System Setup

Docker provides a level of abstraction that allows containers to be portable across different environments, including various hardware configurations and operating systems. To assess this portability of the integration workflow's software implementation, our prototype was deployed and tested on three distinct environments: a local machine, an on-premise hosted server, and a cloud instance. These environments span different hardware architectures and operating systems. This multifaceted evaluation aimed to validate Q-AIM's claim of adaptability to diverse computing environments, emphasizing its suitability for individual users with varied system configurations and requirements. The specifications for the different evaluation configurations can be seen in Table 1.

First, we demonstrate a proof of concept by deploying on a local machine, i.e., personal computer aimed to simulate real-world scenarios where end-users with diverse machines might seek to utilize the software implementation. The successful deployment on the author's machine confirms that the API functions as intended, providing a sound foundation for further evaluation on more sophisticated hosting paradigm scenarios in the following.

Second, to validate the container's applicability in enterprise settings, we deploy it on real server infrastructure belonging to the Modular Supercomputing and Quantum Computing (MSQC) research group

Table 1: Hardware settings for the three evaluation setups.

Parameter	**Cluster Node**	**Local Machine**	**Cloud**
CPU	Intel Xeon E5-2660 v2	Intel i7-12700H	Intel Xeon E5-2696V4 (vCPU)
Cores	20	20	2
RAM	128 GB	32 GB	8 GB
OS	Rocky Linux 9	Ubuntu 24.04	Debian GNU/Linux 12
Network	Ethernet and InfiniBand (FDR)	Ethernet	Public Internet

at Goethe University, Frankfurt am Main, Germany. As part of this process, we reconfigured a compute node from the cluster to function as an independent server, ensuring it could operate separately from the main cluster. This emphasizes the applicability in larger research group's and enterprise settings, capable of hosting on-premise solutions, providing full control over the whole workflow.

Third, given the increasing reliance on cloud services in enterprise environments, we also test our solution on Google Cloud using a E2-standard-2 instance, intended for moderate use, providing good trade-off between cost and performance [47]. This deployment is designed to evaluate the feasibility of using the solution in environments with limited computing resources, such as startups, small businesses, or individual developers who often prioritize cost-effective cloud solutions. The successful deployment, despite the limited resources of the cloud instance, underscored the solution's lightweight design and its ability to perform efficiently in resource-constrained cloud environments. Additionally, deploying the solution in the cloud highlights its potential for scalability. Without requiring any modifications to the docker image itself, the container setup can be scaled to more powerful instances, enabling it to handle more demanding workloads as needed.

The consistent behavior observed across different systems and settings underscores the portability and universality of the composed Q-AIM Docker image, substantiating its viability for widespread adoption.

6.2 Result Discussion

The ability to deploy and use the sample software implementation on all three distinct infrastructure configurations showcases the portability of the proposed solution. Users are not limited to a single hosting paradigm. From the most straight-forward solution, hosting on personal hardware, to more sophisticated solutions, like cloud-hosting, to ultimately fully on-premise server hosting, every use case can be covered by Q-AIM.

Changing the hosting paradigm, e.g. due to higher demand, is just a matter of copying the image and letting it run on the new host, providing the exact same functionality and equal behavior. This reduces the dependency on a particular infrastructure and allows the application of the software to diverse users and use cases.

The evaluation of the portability made it necessary to deploy the same image on different backends, underlining another key aspect of the docker-based microservice implementation: its reproducibility. The same image of the software, with all its configurations specifically designed for our use case, was easily distributed across multiple infrastructures, which can be understood as providing it to different enterprises. Ultimately, this means enabling other users to use a fully fledged and specifically tailored implementation reduces the overhead of creating a common basis for further research/collaboration.

Another critical aspect of the evaluation pertains to the integration workflow's resource efficiency. To investigate resource consumption, the composed Docker container incorporates a resource monitoring software image, cAdvisor [40], as microservice. Running the Q-AIM container automatically starts the monitoring provided by cAdvisor. Utilizing this library, we examined the container's consumption of CPU and memory usage for logging in and running the example from Listing 2 as shown in Fig. 6. Notably, the container exhibited remarkable efficiency, utilizing less than 3 GB of memory in our configuration, whereby Docker uses free memory for caching and frees it as soon as it is needed.

The findings of aforementioned evaluations underscore the integration workflow's software implementation's pivotal attributes, portability across diverse systems and resource-efficient operation. The demonstrated success in real-world scenarios, showed by the seamless deployment on different server infrastructure,

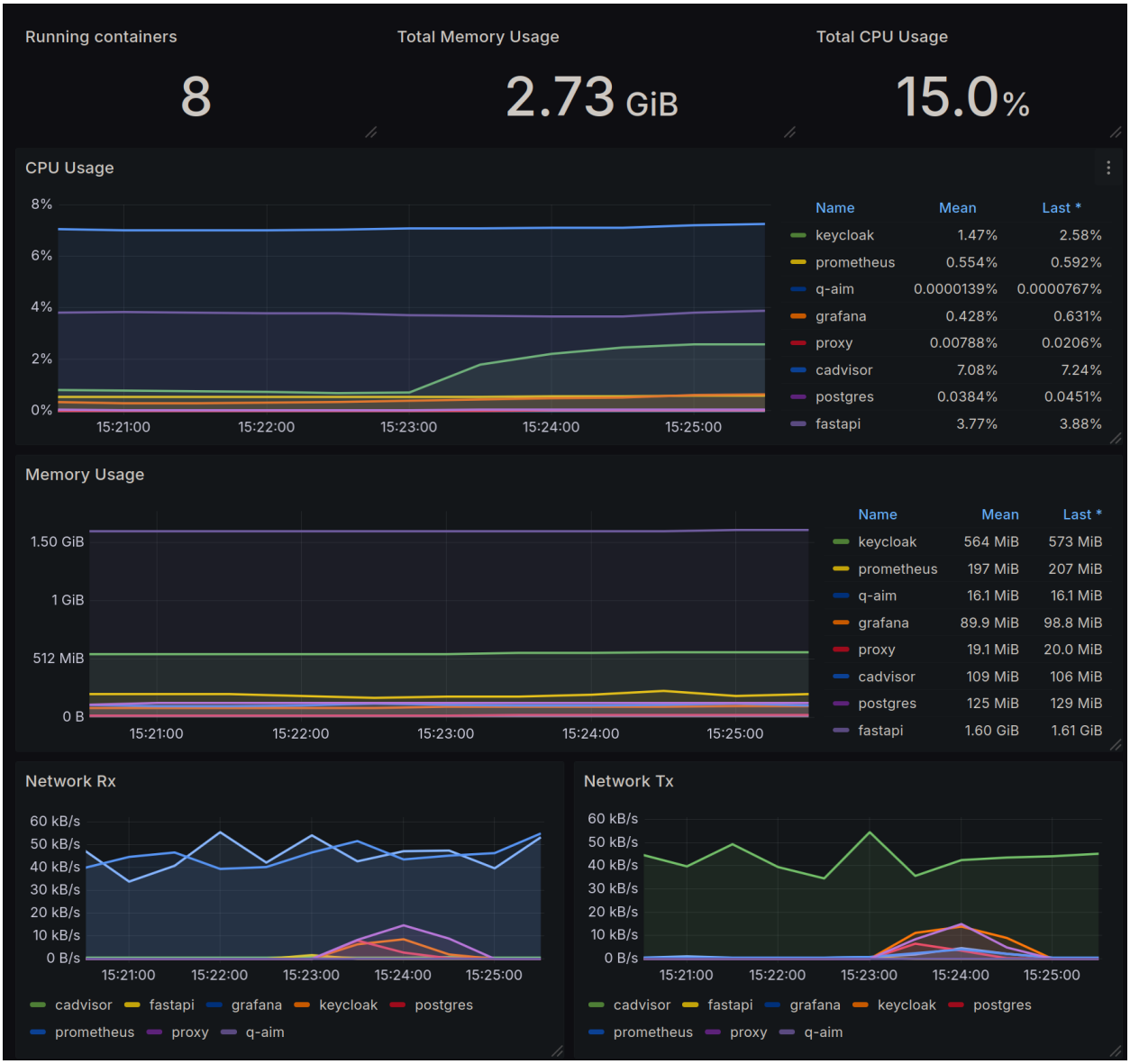

Figure 6: Grafana-based Monitoring Dashboard visualizing memory and CPU usage, as well as network traffic (receiving and transmitting) for the different containers in Q-AIM run on the local machine evaluation setup.

positions Q-AIM as a promising solution for users seeking a lightweight, unified, and universally deployable software solution to incorporate quantum computing hardware and offer access to an on-premise device.

7 Conclusion and Future Work

We proposed the idea of a single-access software solution integrating quantum resources completely vendor-agnostic. The goal is to enable research groups and small entities willing to procure small quantum devices to streamline the process after procurement until usage and provision. Moreover, the proposed API solution should function as an administration and management tool, providing additional security measures for usage outside of the host's direct scope. Lastly, it must be flexible and portable enough to fit to diverse requirements and infrastructures. Therefore, an open-source containerized microservice solution allows for easy modifications, improved maintainability, and ease of use. Combining all of the above, we introduced Q-AIM, a prototype implementation fitted to our needs of evaluation, providing an interface to a quantum simulator mimicking real integration scenarios to serve as proof-of-concept.

Hosting Q-AIM on diverse infrastructures, ranging from personal machines to cloud instances and up to full on-premise servers, emphasizes its applicability in a wide range of use-cases for different demands. Regardless of the hosting paradigm, Q-AIM ships as self-sustained docker image, incorporating all requirements, demanding no profound knowledge to utilize it as a single access point integration solution for quantum computing devices. Not only the hosting paradigm can be decided upon depending on the host's preferences and demands, but owing to the open-source nature the whole image may be fitted accordingly. This also allows Q-AIM to be fully vendor-independent by enabling specific modifications to be made, tailored for downstream tasks to the connected hardware.

To further showcase our streamlined and portable integration workflow, we will use Q-AIM to integrate the first real quantum computing device for the Modular Supercomputing and Quantum Computing research group at Goethe University in Frankfurt am Main. This milestone will enable efficient management of control and access to the computing service both within and beyond the research group. Future works will include the implementation of error mitigation protocols [48,49] within Q-AIM as well as deployment of multi-hybrid quantum algorithms [50].

By facilitating dedicated hardware that bridges the interface to the quantum device's low-level specifications, Q-AIM enables precise control, allowing users to work at the physical level of quantum hardware—all through a single access point. Moreover, we plan to extend its capabilities to monitor hardware utilization in quantum systems, complementing existing monitoring functions in classical computing. This feature is particularly valuable for scientists engaged in hybrid quantum computing, as it will allow them to seamlessly track the hardware utilization of their quantum-hybrid code. Additionally, by exposing quantum resources via an API, classical computing can leverage these resources as accelerators for specific tasks. To further facilitate hybrid computing, methods such as RPC or customized code constructs like pragmas can be employed to enable asynchronous access to quantum computing resources at runtime.

The production-ready version of Q-AIM aims to serve as a unified access point to quantum hardware, efficiently managing all necessary interactions. This comprehensive approach is designed to streamline integration and control processes, providing scientists and early adopters with a robust and efficient solution for advancing quantum research.

Extensive research is essential in the coming years to enhance devices in the current NISQ era and pave the way for more broadly applicable systems and software that will ultimately surpass it. The development of software like Q-AIM is therefore crucial.

References

[1] N. Gisin, G. Ribordy, W. Tittel, and H. Zbinden, "Quantum cryptography," *Rev. Mod. Phys.*, vol. 74, pp. 145–195, Mar 2002. [Online]. Available: https://link.aps.org/doi/10.1103/RevModPhys.74.145

[2] S. Pirandola, U. L. Andersen, L. Banchi, M. Berta, D. Bunandar, R. Colbeck, D. Englund, T. Gehring, C. Lupo, C. Ottaviani, J. L. Pereira, M. Razavi, J. S. Shaari, M. Tomamichel, V. C. Usenko, G. Vallone, P. Villoresi, and P. Wallden, "Advances in quantum cryptography,"

Adv. Opt. Photon., vol. 12, no. 4, pp. 1012–1236, Dec 2020. [Online]. Available: https://opg.optica.org/aop/abstract.cfm?URI=aop-12-4-1012

[3] B. Bauer, S. Bravyi, M. Motta, and G. K.-L. Chan, "Quantum algorithms for quantum chemistry and quantum materials science," *Chemical Reviews*, vol. 120, no. 22, p. 12685–12717, Oct 2020.

[4] Quantum Technology and Application Consortium – QUTAC, Bayerstadler, Andreas, Becquin, Guillaume, Binder, Julia, Botter, Thierry, Ehm, Hans, Ehmer, Thomas, Erdmann, Marvin, Gaus, Norbert, Harbach, Philipp, Hess, Maximilian, Klepsch, Johannes, Leib, Martin, Luber, Sebastian, Luckow, Andre, Mansky, Maximilian, Mauerer, Wolfgang, Neukart, Florian, Niedermeier, Christoph, Palackal, Lilly, Pfeiffer, Ruben, Polenz, Carsten, Sepulveda, Johanna, Sievers, Tammo, Standen, Brian, Streif, Michael, Strohm, Thomas, Utschig-Utschig, Clemens, Volz, Daniel, Weiss, Horst, and Winter, Fabian, "Industry quantum computing applications," *EPJ Quantum Technol.*, vol. 8, no. 1, p. 25, 2021. [Online]. Available: https://doi.org/10.1140/epjqt/s40507-021-00114-x

[5] M. Facchini and S. Thoß, "Build utility-scale quantum applications with the updated open plan and ibm quantum credits," 2023, accessed: 2024-11-27. [Online]. Available: https://www.ibm.com/quantum/blog/utility-scale-quantum-credits

[6] K. Kissell and N. DeSantis, "Expanding access to quantum today for a better tomorrow," 2021, accessed: 2024-11-27. [Online]. Available: https://cloud.google.com/blog/products/compute/ionq-quantum-computer-available-through-google-cloud?hl=en

[7] A. W. S. I. or its affiliates, "Amazon braket pricing," 2024, accessed: 2024-11-27. [Online]. Available: https://aws.amazon.com/braket/pricing/

[8] D. Inc., "Docker: Open platform for developing, shipping, and running applications," https://www.docker.com, 2025, accessed: 2025-02-07.

[9] R. Wille, R. Van Meter, and Y. Naveh, "Ibm's qiskit tool chain: Working with and developing for real quantum computers," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2019, pp. 1234–1240.

[10] V. Omole, A. Tyagi, C. Carey, A. Hanus, A. Hancock, A. Garcia, and J. Shedenhelm, "Cirq: A python framework for creating, editing, and invoking quantum circuits," *0 0*, 2020.

[11] J. Garcia-Alonso, J. Rojo, D. Valencia, E. Moguel, J. Berrocal, and J. M. Murillo, "Quantum software as a service through a quantum api gateway," *IEEE Internet Computing*, vol. 26, no. 1, pp. 34–41, 2022.

[12] F. J. Cardama, J. Vázquez-Pérez, C. Piñeiro, J. C. Pichel, T. F. Pena, and A. Gómez, "Review of intermediate representations for quantum computing," *The Journal of Supercomputing*, vol. 81, no. 2, Jan 2025.

[13] M. Caleffi, M. Amoretti, D. Ferrari, J. Illiano, A. Manzalini, and A. S. Cacciapuoti, "Distributed quantum computing: A survey," *Computer Networks*, vol. 254, p. 110672, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1389128624005048

[14] E. Younis, C. C. Iancu, W. Lavrijsen, M. Davis, E. Smith, and USDOE, "Berkeley quantum synthesis toolkit (bqskit) v1," 04 2021. [Online]. Available: https://www.osti.gov/biblio/1785933

[15] M. Salm, J. Barzen, F. Leymann, B. Weder, and K. Wild, "Automating the comparison of quantum compilers for quantum circuits," in *Service-Oriented Computing*, J. Barzen, Ed. Cham: Springer International Publishing, 2021, pp. 64–80.

[16] A. W. Cross, L. S. Bishop, J. A. Smolin, and J. M. Gambetta, "Open quantum assembly language," 2017.

[17] A. Cross, A. Javadi-Abhari, T. Alexander, N. De Beaudrap, L. S. Bishop, S. Heidel, C. A. Ryan, P. Sivarajah, J. Smolin, J. M. Gambetta, and B. R. Johnson, "Openqasm 3: A broader and deeper quantum assembly language," *ACM Transactions on Quantum Computing*, vol. 3, no. 3, sep 2022. [Online]. Available: https://doi.org/10.1145/3505636

[18] M. Schulz, M. Ruefenacht, D. Kranzlmüller, and L. B. Schulz, "Accelerating hpc with quantum computing: It is a software challenge too," *Computing in Science & Engineering*, vol. 24, no. 4, pp. 60–64, 2022.

[19] M. Ruefenacht, B. G. Taketani, P. Lähteenmäki, V. Bergholm, D. Kranzlmüller, L. Schulz, and M. Schulz, "Bringing quantum acceleration to supercomputers," *IQM/LRZ Technical Report, https://www. quantu m. lrz. de/fileadmin/QIC/Downloads/IQM HPC-QC-Integration-White paper. pdf*, 2022.

[20] T. S. Humble, A. McCaskey, D. I. Lyakh, M. Gowrishankar, A. Frisch, and T. Monz, "Quantum computers for high-performance computing," *IEEE Micro*, vol. 41, no. 5, pp. 15–23, 2021.

[21] A. C. Marosi, A. Farkas, T. Máray, and R. Lovas, "Toward a quantum-science gateway: A hybrid reference architecture facilitating quantum computing capabilities for cloud utilization," *IEEE Access*, vol. 11, pp. 143 913–143 924, 2023.

[22] H. Soeparno and A. S. Perbangsa, "Cloud quantum computing concept and development: A systematic literature review," *Procedia Computer Science*, vol. 179, pp. 944–954, 2021, 5th International Conference on Computer Science and Computational Intelligence 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050921001150

[23] M. Golec, E. S. Hatay, M. Golec, M. Uyar, M. Golec, and S. S. Gill, "Quantum cloud computing: Trends and challenges," *Journal of Economy and Technology*, vol. 2, pp. 190–199, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2949948824000271

[24] J. Alvarado-Valiente, J. Romero-Álvarez, E. Moguel, J. García-Alonso, and J. M. Murillo, "Technological diversity of quantum computing providers: a comparative study and a proposal for api gateway integration," *Software Quality Journal*, vol. 32, pp. 53–73, 2024.

[25] A. Javadi-Abhari, M. Treinish, K. Krsulich, C. J. Wood, J. Lishman, J. Gacon, S. Martiel, P. D. Nation, L. S. Bishop, A. W. Cross, B. R. Johnson, and J. M. Gambetta, "Quantum computing with qiskit," 2024. [Online]. Available: https://arxiv.org/abs/2405.08810

[26] E. Moguel, J. Rojo, and D. e. a. Valencia, "Quantum service-oriented computing: current landscape and challenges," *Software Quality Journal*, vol. 30, pp. 983–1002, 2022.

[27] H. T. Nguyen, M. Usman, and R. Buyya, "Qfaas: A serverless function-as-a-service framework for quantum computing," *Future Generation Computer Systems*, vol. 154, pp. 281–300, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X24000189

[28] M. Grossi, L. Crippa, A. Aita, G. Bartoli, V. Sammarco, E. Picca, N. Said, F. Tramonto, and F. Mattei, "A serverless cloud integration for quantum computing," 2021. [Online]. Available: https://arxiv.org/abs/2107.02007

[29] T. Beck, A. Baroni, R. Bennink, G. Buchs, E. A. C. Pérez, M. Eisenbach, R. F. da Silva, M. G. Meena, K. Gottiparthi, P. Groszkowski, T. S. Humble, R. Landfield, K. Maheshwari, S. Oral, M. A. Sandoval, A. Shehata, I.-S. Suh, and C. Zimmer, "Integrating quantum computing resources into scientific hpc ecosystems," *Future Generation Computer Systems*, vol. 161, pp. 11–25, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X24003583

[30] D. Merkel, "Docker: lightweight linux containers for consistent development and deployment," *Linux journal*, vol. 2014, no. 239, p. 2, 2014.

[31] G. M. Kurtzer, M. Bauer, I. Kaneshiro, D. Trudgian, D. Godlove, Y. Cote, C. E. Arango Gutierrez, G. Vallee, A. Hughes, J. Cook *et al.*, "hpcng/singularity: Singularity 3.7. 1," *Zenodo*, 2021.

[32] D. Moreau, K. Wiebels, and C. Boettiger, "Containers for computational reproducibility," *Nature Reviews Methods Primers*, vol. 3, no. 1, p. 50, 2023.

[33] J. Sermersheim, "Rfc 4511: Lightweight directory access protocol (ldap): The protocol," 2006.

[34] P. Community, "Official postgresql docker image," 2025, accessed: 2025-01-29. [Online]. Available: https://hub.docker.com/_/postgres

[35] K. Community, "Keycloak docker image," 2025, accessed: 2025-01-29. [Online]. Available: https://hub.docker.com/r/keycloak/keycloak

[36] S. Ramírez, "Fastapi," 2018, accessed: 2025-01-28. [Online]. Available: https://github.com/fastapi/fastapi

[37] A. Team, "Angular - the modern web framework," 2025, accessed: 2025-01-29. [Online]. Available: https://angular.io/

[38] M. Cantelon, M. Harter, T. Holowaychuk, and N. Rajlich, *Node.js in Action*. Manning Greenwich, 2014.

[39] W. Reese, "Nginx: The high-performance web server and reverse proxy," *Linux Journal*, vol. 2008, no. 173, p. 2, 2008.

[40] Google, "cadvisor docker image," 2025, accessed: 2025-01-29. [Online]. Available: https://hub.docker.com/r/google/cadvisor

[41] C. Gaberle, "Design and implementation of a quantum circuit preparation algorithm," 2023, unpublished thesis, Goethe University Frankfurt.

[42] ——, "Qasmparserlibrary," 2023, accessed: 2025-01-28. [Online]. Available: https://github.com/msqc-goethe/QasmParserLibrary

[43] M. S. Jattana, F. Jin, H. De Raedt, and K. Michielsen, "Improved variational quantum eigensolver via quasidynamical evolution," *Phys. Rev. Appl.*, vol. 19, p. 024047, Feb 2023. [Online]. Available: https://link.aps.org/doi/10.1103/PhysRevApplied.19.024047

[44] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien, "A variational eigenvalue solver on a quantum processor," *Nature Communications*, vol. 5, p. 4213, 2014.

[45] M. S. Jattana, F. Jin, H. De Raedt, and K. Michielsen, "Assessment of the variational quantum eigensolver: Application to the heisenberg model," *Frontiers in Physics*, vol. 10, 2022. [Online]. Available: https://www.frontiersin.org/journals/physics/articles/10.3389/fphy.2022.907160

[46] Qiskit Development Team, "Qiskit aer QasmSimulator backend," https://qiskit.github.io/qiskit-aer/stubs/qiskit_aer.QasmSimulator.html, 2025, accessed: 2025-02-10.

[47] G. Cloud, "Google cloud compute vm instance pricing," 2025, accessed: 2025-01-29. [Online]. Available: https://cloud.google.com/compute/vm-instance-pricing

[48] M. S. Jattana, F. Jin, H. De Raedt, and K. Michielsen, "General error mitigation for quantum circuits," *Quantum Information Processing*, vol. 19, no. 11, p. 414, 2020. [Online]. Available: https://doi.org/10.1007/s11128-020-02913-0

[49] P. Döbler, J. Pflieger, F. Jin, H. D. Raedt, K. Michielsen, T. Lippert, and M. S. Jattana, "Scalable general error mitigation for quantum circuits," 2024. [Online]. Available: https://arxiv.org/abs/2411.07916

[50] M. S. Jattana, "Quantum annealer accelerates the variational quantum eigensolver in a triple-hybrid algorithm," *Physica Scripta*, vol. 99, no. 9, p. 095117, aug 2024. [Online]. Available: https://dx.doi.org/10.1088/1402-4896/ad6aea