ISSN 2464-4617 (print)
ISSN 2464-4625 (online)
Quantum Informatics, Computing & Technology 2025
QC-Horizon 2025
https://www.qc-horizon.eu/

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

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

[*] *Joint first authors*
[1]Institute of Computer Science, 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 of 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 at 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 with purchasing and hosting their own quantum hardware with a tool simplifying the process from procurement to operation and removing non-research-related technical redundancies.

Keywords

Quantum Computing, Quantum Computing Cloud Solution, Quantum Computing Infrastructure, Quantum Computing Integration, Quantum Computing Software Stack, User Management, Micro-Service Architecture, Docker

1 INTRODUCTION

Quantum computing, currently in its developmental phase, promises substantial acceleration of classical computations across various fields ranging from cryptography to materials science [GRTZ02, PAB+20, BBMC20, QBB+21]. 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, Google, and Amazon 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 policies 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.

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.
- ***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 demonstrating adaptability and efficient resource usage across on-premise and cloud infrastructures, supporting broad research and educational application possibilities.

2 BACKGROUND

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 Resource Workflow

Executing quantum algorithms on hardware requires translating high-level logic into device-specific operations through multiple abstraction layers, as shown in Fig. 1. The process begins with circuit definition in hardware-agnostic frameworks like Qiskit [WVMN19], Cirq [OTC+20], or Braket [GARV+22], analogous to classical algorithm development. The quantum equivalent of compilation is transpilation, which generates an intermediate representation (IR) [CVPP+25]. This involves both hardware-independent optimizations and hardware-dependent adaptations to match the processor's native gate set [CAF+24]. Tools like BQSKit [YIL+21] perform these transformations, optimizing circuit depth while respecting hardware constraints. Common IR formats include OpenQASM [CBSG17, CJAA+22], serving as a quantum assembly language.

The final stage converts the IR into machine instructions, typically implemented as precisely controlled microwave pulses that manipulate qubit states. This completes the translation from abstract algorithm to physical implementation, with compiler comparisons available in [SBL+21]. 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.

To execute algorithms on real quantum hardware, two key aspects must be considered. First, as shown in Fig. 1, any algorithm or circuit must be transpiled into the underlying hardware's instruction set. Typically, algorithms are developed in a hardware-agnostic manner, requiring translation into device-specific operations.

Second, access to quantum resources must be established. Providers usually offer cloud-based access via APIs, treating quantum computers as specialized remote resources. They enforce restrictions on supported IR formats, interaction methods, and security protocols, requiring authentication and permissions. Access is managed through an API, which handles data flow to and from the resource. In Fig. 1, the API call can occur at any stage between circuit definition and machine instructions, depending on the service. After execution, results are returned in a provider-defined format.

2.2 Analysis of Related Work

Recent work explores integrating quantum devices with classical resources through two paradigms: GPU-like indirect access [SRKS22, RTL+22, HML+21] or API-driven direct access [MFML23, SP21, GHG+24, HML+21,JATK+24,GARV+22,OTC+20]. Ruefenacht et al. [RTL+22] categorize integration architectures from loosely-coupled (on-premise) to tightly-coupled (on-chip) for HPC workloads. While Schulz et

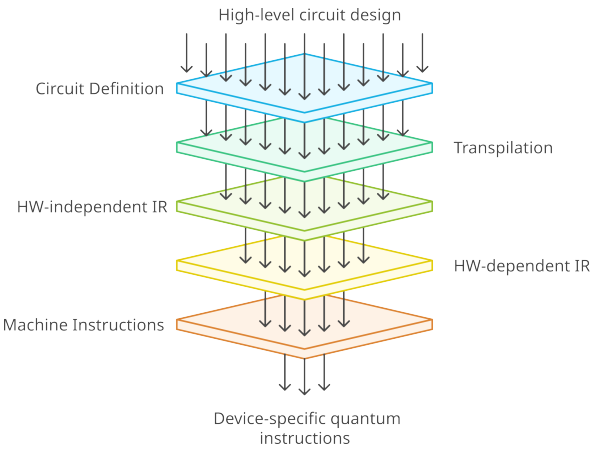

Figure 1: Instruction abstraction levels in quantum computing. From high-level circuit design (highest abstraction) to hardware-independent and hardware-specific intermediate representations (IR), ultimately becoming device-specific machine instructions (no abstraction) to be used with the quantum device.

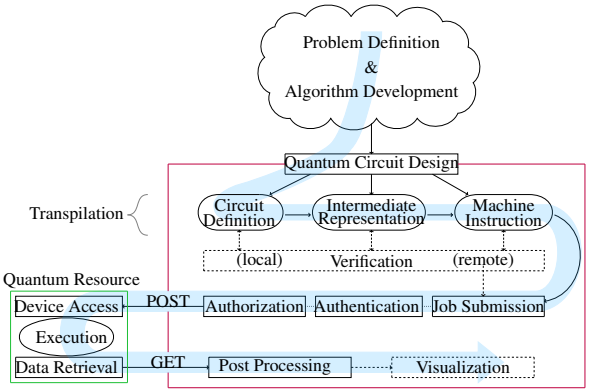

Figure 2: Overview of the quantum computation workflow. The larger, red box (right) indicates classical hardware, the smaller, green one (left) the quantum system. The problem is defined as quantum circuit at any abstraction level and undergoes transpilation until machine instruction level is reached.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

al. [SRKS22] advocate for unified software stacks, Humble et al. [HML+21] note that current prototypes rely on primitive client-server interactions unsuitable for true acceleration.

Precisely, these early-stage systems are crucial for academic research and motivate our work: an open-source platform enabling secure, low-overhead access to quantum devices across environments (on-premise/cloud) via Docker containers. This addresses the gap in accessible solutions for small-scale research, contrasting with service-oriented approaches [MRV22, NUB24, GCA+21] that abstract hardware for enterprise use. For instance, Grossi et al. [GCA+21] propose quantum FaaS via HTTP APIs, while Nguyen et al. [NUB24] mitigate vendor lock-in.

Our focus reverses this paradigm, instead of high-level abstraction, we enable fine-grained control (e.g., pulse-level access) critical for research. Concurrent work by Beck et al. [BBB+24] targets HPC-integrated quantum acceleration for large institutions, whereas our solution democratizes access for smaller groups, streamlining device integration from procurement to experimental use.

3 DESIGN CONSIDERATION

As described in Section 2, integrating quantum computing resources into existing research and industrial workflows requires careful orchestration of software across multiple domains. However, current quantum solutions are tied to specialized hardware and proprietary environments, limiting applicability and creating barriers.

To address this, we propose Q-AIM: a standardized, portable workflow and corresponding software implementation enabling seamless integration of quantum resources. In Fig. 2, required process steps are shown in blue, classical services in the red box, and vendor-dependent quantum resources in the green box. Therefore, with Q-AIM, quantum circuit design supports various abstraction levels. An optional simulation step allows verification before using remote quantum computers. All services in the red box are abstractions provided by the software, with each component replaceable or customizable, allowing granular control for users and providers. This flexibility supports diverse requirements. Due to varying execution environments and access control needs, we use REST APIs for broad compatibility across languages, platforms, and architectures.

Another key design aspect is the decoupling of quantum hardware, ensuring software operates independently of specific hardware. Tasks like qubit manipulation, instruction execution, and measurement are handled by the hardware and its peripherals. Together with microservices, these form the Q-AIM backend.

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

1. Fully Integrated Classical Workflow: Based on the design considerations 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 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.

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*. The central component of this system architecture is the API gateway, which 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 that offers additional functions that go beyond the simple forwarding of requests. It 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.

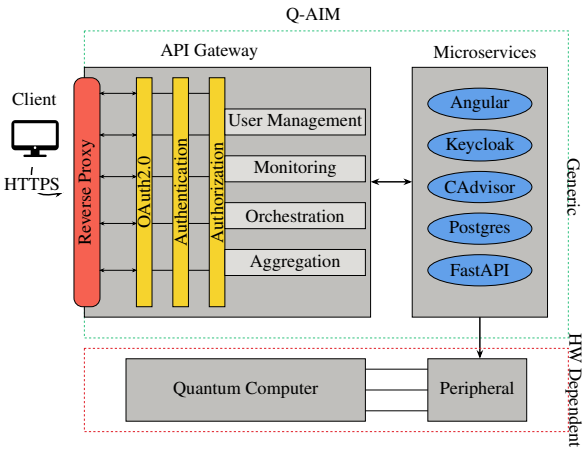

Figure 3: Microservice-based architecture of Q-AIM. It facilitates secure client interactions via HTTPS and a reverse proxy, providing access to quantum systems through a structured microservice architecture. The API Gateway manages authentication, authorization, and orchestration, while the microservices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

As the result, our architecture provides a standardized way for those to communicate, interact thus allows for modularity, scalability, and adaptability, making it possible to integrate the services seamlessly while maintaining a consistent and manageable architecture.

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 of Q-AIM is therefore portability and transparency.

Lightweight virtualization technologies, i.e., containers such as Docker or Apptainer are highly portable. 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 [MWB23]. 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. 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.

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 [Ser06] 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 to 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:

```
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
        ...
    ...
```

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

- **Database Service**: This initiates a PostgreSQL database utilizing the official Postgres Docker image. To ensure persistent storage of the database data, a Docker volume is created alongside.
- **Authentication Service**: Utilizing the official Keycloak Docker image, this service delivers identity and access management functionalities. It relies on the database service and necessitates a Keycloak 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 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 is built in a Node.js environment and then the resulting build is deployed within an NGINX 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 to efficiently gather

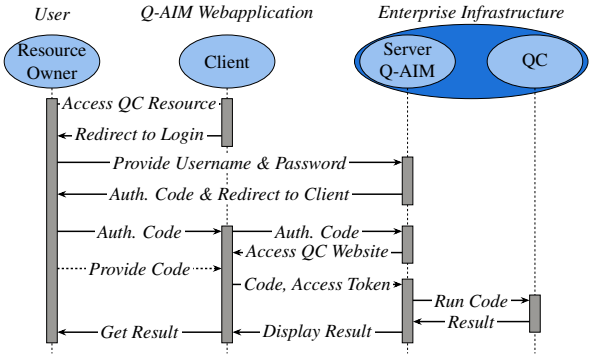

Figure 4: Representation of the first authentication process for an approved user attempting to run code on a protected quantum computing resource.

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.

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 a 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 permission 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.

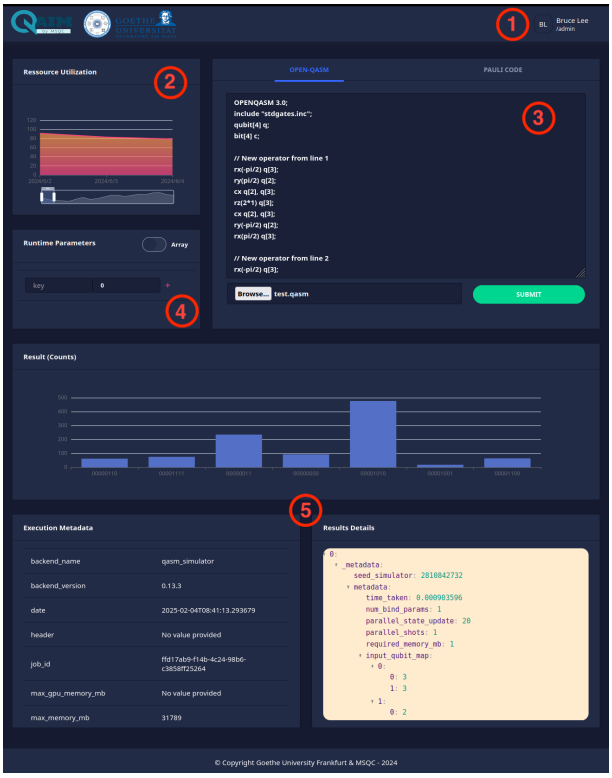

Figure 5: Q-AIM Web User Interface. Users can provide code and runtime parameters in different formats, monitor resource utilization, and visualize results and metadata.

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 as an input. The Pauli representation takes advantage of the fact that the Pauli rotations together with the controlled-NOT (CNOT) oper-

ation 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 circuit in OpenQASM format is 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 editor or uploading the corresponding file.

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 by 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 an algorithm necessitating parameterization is the Variational Quantum Eigensolver (VQE) [JJDRM23, PMS+14, JJDRM22]. 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 [Qis25], 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

To assess the portability of the integration workflow's software implementation, our prototype was deployed and tested on three distinct environments: (1) a local machine, (2) an on-premise hosted server, and (3) a cloud instance. These environments span different hardware architectures and operating systems. This multifaceted evaluation aims 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 is described in Table 1.

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

Table 1: Hardware settings for the evaluation setups.

First, we demonstrate a proof of concept by deploying on a local machine, i.e., a personal computer aimed at stimulating 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 of more sophisticated hosting paradigm scenarios in the following.

Second, to validate the container's applicability in enterprise settings, we deployed it on real server infrastructure belonging to the Modular Supercomputing and Quantum Computing (MSQC) research group 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 its applicability in larger research groups 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 an E2-standard-2 instance, intended for moderate use, providing a good trade-off between cost and performance. 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.

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

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 and 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, as a 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 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 the aforementioned evaluations underscore the integration workflow's software implementation's pivotal attributes: portability across diverse sys-

tems and resource-efficient operation. The demonstrated success in real-world scenarios, shown by the seamless deployment on different server infrastructure, 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

We propose Q-AIM, a vendor-agnostic single-access solution for integrating quantum resources. Designed for research groups and small entities, it streamlines quantum device management from procurement through usage. The API-based solution provides administration tools with enhanced security for remote access while maintaining flexibility across diverse infrastructures. Implemented as an open-source containerized microservice, Q-AIM offers easy modification and maintainability. Our prototype currently interfaces with quantum simulators, demonstrating real-world integration scenarios.

Q-AIM's Docker-based deployment supports various infrastructures - from personal machines to cloud and on-premise servers - requiring minimal expertise for setup. Its open-source nature enables customization for specific hardware needs, ensuring complete vendor independence. We are implementing Q-AIM with Goethe University's Modular Supercomputing group for their first quantum device, enabling controlled access both within and beyond the research group. Future developments include integrating error mitigation protocols [JJDRM20, DPJ+24] and multi-hybrid quantum algorithms [Jat24].

The platform provides precise low-level hardware control through a unified interface, with planned extensions for quantum hardware monitoring and hybrid computing support. By exposing quantum resources via API, Q-AIM enables classical systems to leverage them as accelerators, potentially using RPC or pragmas for runtime access. The production version will serve as a comprehensive quantum hardware management solution, streamlining integration for researchers.

As quantum computing advances beyond the NISQ era, tools like Q-AIM are essential for bridging current limitations and future capabilities.

8 REFERENCES

[BBB+24] Thomas Beck, Alessandro Baroni, Ryan Bennink, Gilles Buchs, et al. Integrating quantum computing resources into scientific hpc ecosystems. *Future Generation Computer Systems*, 161:11–25, 2024.

[BBMC20] Bela Bauer, Sergey Bravyi, Mario Motta, and Garnet Kin-Lic Chan. Quantum algorithms for quantum chemistry and quantum materials science. *Chemical Reviews*, 120(22):12685–12717, Oct 2020.

[CAF+24] Marcello Caleffi, Michele Amoretti, Davide Ferrari, Jessica Illiano, et al. Distributed quantum computing: A survey. *Computer Networks*, 254:110672, 2024.

[CBSG17] Andrew W. Cross, Lev S. Bishop, John A. Smolin, and Jay M. Gambetta. Open quantum assembly language, 2017.

[CJAA+22] Andrew Cross, Ali Javadi-Abhari, Thomas Alexander, Niel De Beaudrap, et al. Openqasm 3: A broader and deeper quantum assembly language. *ACM Transactions on Quantum Computing*, 3(3), sep 2022.

[CVPP+25] F. Javier Cardama, Jorge Vázquez-Pérez, César Piñeiro, Juan C. Pichel, et al. Review of intermediate representations for quantum computing. *The Journal of Supercomputing*, 81(2), Jan 2025.

[DPJ+24] Philip Döbler, Jannik Pflieger, Fengping Jin, Hans De Raedt, et al. Scalable general error mitigation for quantum circuits, 2024.

[GARV+22] Jose Garcia-Alonso, Javier Rojo, David Valencia, Enrique Moguel, et al. Quantum software as a service through a quantum api gateway. *IEEE Internet Computing*, 26(1):34–41, 2022.

[GCA+21] M. Grossi, L. Crippa, A. Aita, G. Bartoli, et al. A serverless cloud integration for quantum computing, 2021.

[GHG+24] Muhammed Golec, Emir Sahin Hatay, Mustafa Golec, Murat Uyar, et al. Quantum cloud computing: Trends and challenges. *Journal of Economy and Technology*, 2:190–199, 2024.

[GRTZ02] Nicolas Gisin, Grégoire Ribordy, Wolfgang Tittel, and Hugo Zbinden. Quantum cryptography. *Rev. Mod. Phys.*, 74:145–195, Mar 2002.

[HML+21] Travis S. Humble, Alexander McCaskey, Dmitry I. Lyakh, Meenambika Gowrishankar, et al. Quantum computers for high-performance computing. *IEEE Micro*, 41(5):15–23, 2021.

[Jat24] Manpreet Singh Jattana. Quantum annealer accelerates the variational quantum eigensolver in a triple-hybrid algorithm. *Physica Scripta*, 99(9):095117, aug 2024.

[JATK+24] Ali Javadi-Abhari, Matthew Treinish, Kevin Krsulich, Christopher J. Wood, et al. Quantum computing with qiskit, 2024.

[JJDRM20] Manpreet Singh Jattana, Fengping Jin, Hans De Raedt, and Kristel Michielsen. General error mitigation for quantum circuits. *Quantum Information Processing*, 19(11):414, 2020.

[JJDRM22] Manpreet Singh Jattana, Fengping Jin, Hans De Raedt, and Kristel Michielsen. Assessment of the variational quantum eigensolver: Application to the heisenberg model. *Frontiers in Physics*, 10, 2022.

[JJDRM23] Manpreet Singh Jattana, Fengping Jin, Hans De Raedt, and Kristel Michielsen. Improved variational quantum eigensolver via quasidynamical evolution. *Phys. Rev. Appl.*, 19:024047, Feb 2023.

[MFML23] Attila Csaba Marosi, Attila Farkas, Tamás Máray, and Róbert Lovas. Toward a quantum-science gateway: A hybrid reference architecture facilitating quantum computing capabilities for cloud utilization. *IEEE Access*, 11:143913–143924, 2023.

[MRV22] E. Moguel, J. Rojo, and D. et al. Valencia. Quantum service-oriented computing: current landscape and challenges. *Software Quality Journal*, 30:983–1002, 2022.

[MWB23] David Moreau, Kristina Wiebels, and Carl Boettiger. Containers for computational reproducibility. *Nature Reviews Methods Primers*, 3(1):50, 2023.

[NUB24] Hoa T. Nguyen, Muhammad Usman, and Rajkumar Buyya. Qfaas: A serverless function-as-a-service framework for quantum computing. *Future Generation Computer Systems*, 154:281–300, 2024.

[OTC+20] Victory Omole, Akhilesh Tyagi, Calista Carey, AJ Hanus, et al. Cirq: A python framework for creating, editing, and invoking quantum circuits. *0 0*, 2020.

[PAB+20] S. Pirandola, U. L. Andersen, L. Banchi, M. Berta, et al. Advances in quantum cryptography. *Adv. Opt. Photon.*, 12(4):1012–1236, Dec 2020.

[PMS+14] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, et al. A variational eigenvalue solver on a quantum processor. *Nature Communications*, 5:4213, 2014.

[QBB+21] Quantum Technology and Application Consortium – QUTAC, Andreas Bayerstadler, Guillaume Becquin, Julia Binder, et al. Industry quantum computing applications. *EPJ Quantum Technol.*, 8(1):25, 2021.

[Qis25] Qiskit Development Team. Qiskit aer QasmSimulator backend. `https://qiskit.github.io/qiskit-aer/stubs/qiskit_aer.QasmSimulator.html`, 2025. Accessed: 2025-02-10.

[RTL+22] Martin Ruefenacht, Bruno G Taketani, PASI Lähteenmäki, VILLE Bergholm, et al. 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.

[SBL+21] Marie Salm, Johanna Barzen, Frank Leymann, Benjamin Weder, et al. Automating the comparison of quantum compilers for quantum circuits. In Johanna Barzen, editor, *Service-Oriented Computing*, pages 64–80, Cham, 2021. Springer International Publishing.

[Ser06] J Sermersheim. Rfc 4511: Lightweight directory access protocol (ldap): The protocol, 2006.

[SP21] Haryono Soeparno and Anzaludin Samsinga Perbangsa. Cloud quantum computing concept and development: A systematic literature review. *Procedia Computer Science*, 179:944–954, 2021. 5th International Conference on Computer Science and Computational Intelligence 2020.

[SRKS22] Martin Schulz, Martin Ruefenacht, Dieter Kranzlmüller, and Laura Brandon Schulz. Accelerating hpc with quantum computing: It is a software challenge too. *Computing in Science and Engineering*, 24(4):60–64, 2022.

[WVMN19] Robert Wille, Rod Van Meter, and Yehuda Naveh. Ibm's qiskit tool chain: Working with and developing for real quantum computers. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1234–1240, 2019.

[YIL+21] Ed Younis, Costin C Iancu, Wim Lavrijsen, Marc Davis, et al. Berkeley quantum synthesis toolkit (bqskit) v1, 04 2021.