Interface between Quantum and Classical Computers

A study in Cloud-Based Quantum Computing

Ioannis Savvaidis Reg. 60663

Master in Quantum Computing and Quantum Technologies

Dept. of Electrical & Computer Engineering, School of Engineering

Democritus University of Thrace

Supervisor: Prof. Ioannis Karafyllidis

Co-supervisors: Prof. Georgios Sirakoulis Dr. Panagiotis Dimitrakis

Athens, June 2026

Outline

Motivation

A systems engineering perspective

- The quantum–classical interface is treated as a problem of systems engineering: abstractions, layers, and orchestration, not device physics.
- The focus is the system's internals and behaviour, not the hardware's physical realisation. The questions are how circuits are compiled, scheduled, executed, and observed.
- The perspective draws on Site Reliability Engineering, the discipline of operating large-scale production systems.
- Those systems are run through scheduling, orchestration, observability, and reliability engineering.
- The interest dates to a 2012 BSc dissertation on cloud computing.

An operational gap at the quantum execution boundary

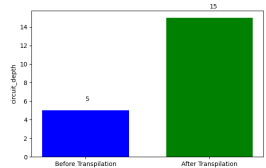
First executions on real IBM hardware, during the MSc coursework (August 2024, IBM Quantum Open Plan):

- A two-qubit Deutsch circuit on `ibm_sherbrooke` reported $\approx 3\,\mathrm{s}$ of QPU time, orders of magnitude more than a six-gate circuit requires.
- Qiskit exposed no breakdown of that latency.
- Two diagnostics profiled it: `metrics()` (depth, gates, schedule, per-gate duration) and `aer_mimic_simulation()` (`SamplerV2`, 4096 shots).
- Both ran on `AerSimulator` (ideal), `FakeSherbrooke` (noise model), and real `ibm_sherbrooke`.

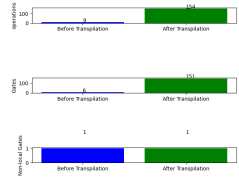
Deutsch: transpilation reshapes the circuit

Transpilation lays the 2-qubit circuit across all of the device's qubits (127 on `ibm_sherbrooke`) and schedules it:

- Circuit depth ($5 \to 15$): the longest chain of dependent instructions, lengthened by native-gate decomposition. Deeper is noisier.
- Operations ($9 \to 154$): every instruction in the circuit. Almost all are `delay` instructions placed on the idle qubits.
- Gates ($6 \to 151$): Qiskit's reported circuit size. About 19 are real gates (`rz`, `sx`, `x`, `ecr`), the remainder delays.
- Non-local gates ($1 \to 1$): operations on two qubits. One `ecr`, the dominant error source.



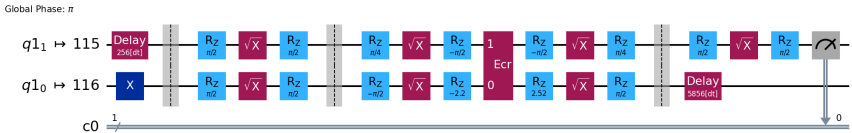
Circuit depth, before / after.



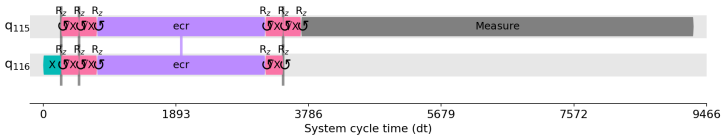
Counts before / after (151 is mostly delays).

Deutsch: the transpiled circuit and its schedule

The Deutsch circuit, transpiled onto physical qubits 115–116 of `ibm_sherbrooke` and scheduled. Idle qubits are hidden.

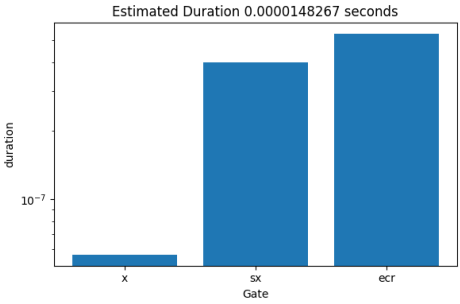


- **Top, the circuit.** The native-gate sequence the device runs (`rz`, `sx`, `x`, `ecr`), with `delay` gaps for synchronisation.



- **Bottom, the schedule** (dt units). Measurement and delay take most of the timeline. Gate execution is a small fraction.

Deutsch: execution-time breakdown



Per-gate durations for the timed native gates `x`, `sx`, `ecr`. `rz` is virtual (zero duration).

- Gate time is microseconds. The scheduled timeline is 1.89 µs $(9466\,dt)$, and summing every gate serially gives 14.8 µs. Both sit six orders of magnitude below 3 s.
- Almost all of 3 s wall-time is spent in off-QPU time, in `SamplerV2`, IBM Runtime, the job queue, and classical–quantum communication. None of it is observable from Qiskit.
- Simulation under-predicts error. `FakeSherbrooke` gives 38 erroneous shots against 140 on hardware ($\sim 4\times$), so fidelity must be measured, not modelled.

Shor: from measured peaks to factors

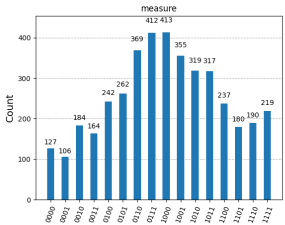
Shor reduces factoring $N = 15$ ($a = 4$) to period-finding, recovering the period r of $a^x \bmod N$. On the ideal simulator, phase estimation yields two peaks, decoded by continued fractions:

state	l	$\varphi = l/2^4$	r	result
$\|1000\rangle$	8	0.5000	2	$15 = 3 \times 5$
$\|0000\rangle$	0	0.0000	1	(trivial)

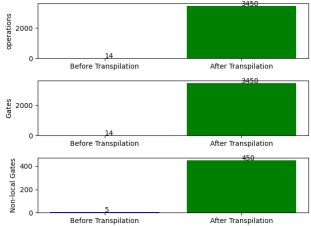
- Only $|1000\rangle$ encodes the period. With $r = 2$, $\gcd(4 \pm 1,\ 15) = \{3, 5\}$.
- These peaks define the reference distribution for scoring hardware runs.

Backend selection governs the result

On hardware, the backend determines whether the reference peaks are recoverable.



`ibm_brisbane`: counts spread across all 16 states, no period signal.



`ibm_brisbane`: 3,450 operations.

- `least_busy()` chose `ibm_brisbane` on queue depth alone, and blocked until the job returned. Its output was unusable, where `ibm_sherbrooke` was noisy but readable.
- It ran fewer operations than `ibm_sherbrooke` (3,450 vs 3,456), yet performed far worse. Qubit mapping and physical topology mattered more than the operation count.

Operational gaps identified

The experiments identified recurring gaps:

- Fidelity is opaque: run quality is observable only post-execution.
- No gate-level observability: which gates cost time or introduce error is not exposed, so the problem cannot be targeted.
- Noise scales with complexity: tolerable for Deutsch, fatal for Shor, and worsening with depth.
- Backend selection is manual: calibration inspection, trial jobs, `least_busy()` (queue depth only).
- No backend ranking: no model maps a circuit to a best-fit QPU.

> *Realising the potential of quantum computing requires not hardware refinement alone but an "integrated Quantum–Classical systems architecture".* — *Seelam et al.* (2026)

Thesis objectives and questions

The thesis brings cloud-native practices to the classical layer around quantum execution: orchestration, declarative deployment, standards-based observability.

Objectives

- **O1** Survey existing approaches for quantum–classical interfacing across architectural layers.
- **O2** Analyse interface models at the programming-framework level and the infrastructure level.
- **O3** Identify gaps related to production deployment, workload orchestration, and observability.
- **O4** Design and demonstrate a cloud-native hybrid execution workflow using Kubernetes and OpenTelemetry.

Guiding questions

- **Q1** At which layers does interfacing occur?
- **Q2** How do framework and infrastructure approaches differ?
- **Q3** What limits production deployment and observability?
- **Q4** How can cloud-native practices be adapted to quantum–classical workloads?

Scope and boundaries

In scope

- Software interfaces for gate-based QC.
- Cloud-accessible execution models.
- Hybrid workflows, with Shor $N = 15$ as the reference workload.
- Orchestration, lifecycle, observability.

Out of scope

- Pulse engineering / control electronics.
- New algorithms or quantum-advantage claims.
- Quantum error correction.
- Quantum networking.

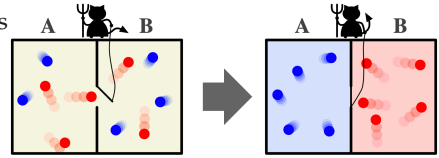
Why Qiskit and Shor $N = 15$

Qiskit for its maturity and IBM Quantum access. Shor $N = 15$ for a known-correct answer to score against, and a post-transpile depth that exercises the full compile–route–execute–measure pipeline.

Information and Computation

Information is physical

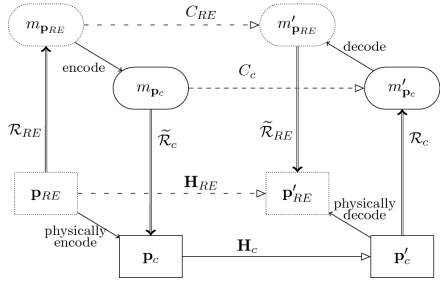
- Maxwell's demon (1871): a thought experiment. An agent sorts fast from slow molecules to build a temperature gap, lowering entropy with no work and appearing to violate the second law of thermodynamics.
- Entropy is the shared quantity: the same measure describes disorder in a physical system (Boltzmann, 1877) and, as Shannon entropy, the average uncertainty of an information source (1948).
- Landauer (1961): erasing one bit is logically irreversible and has an unavoidable minimum energy cost, released as heat.
- Bennett (1982): the demon's finite memory must eventually be erased, and that erasure restores the entropy the sorting removed. The second law holds.
- Computation is therefore a physical process, with real and measurable costs.



Maxwell's demon (CC BY-SA 3.0, Htkym / Wikimedia).

When does a physical system compute?

- Turing machine (1936): a tape of symbols and a head that reads, writes, and moves by fixed rules. It defines what is computable, and by the Church–Turing thesis (1936) it is universal.
- Abstraction/Representation Theory (Horsman et al., 2014): a system computes only when a *representation relation* links physical states to abstract objects, and computational entities encode the problem in and decode the result out. Without that, it is only evolving.
- Stepney & Kendon (2021): the *Representational Entity* is the agent that owns the problem and supports that relation. It need not be human.
- A physical system therefore computes only when it is used to predict the outcome of an abstract calculation.

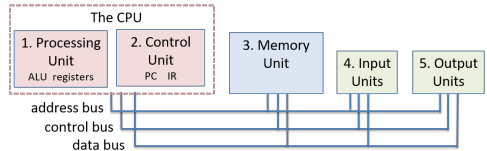


The compute cycle with the RE (Stepney & Kendon, 2021, Fig. 5).

Classical Computing

The machine

- **Binary encoding**: every value is bits, deterministic and read without disturbing it.
- **Von Neumann architecture**: a CPU and memory connected by a shared bus.
- **Stored-program model**: data and instructions share one memory, so loading a program, not rewiring, changes what the machine computes.
- **Fetch–decode–execute**: the control unit steps through instructions with the program counter (PC) and instruction register (IR).
- **Irreversible gates**: erasing a bit has a minimum heat cost (Landauer).



The von Neumann architecture.

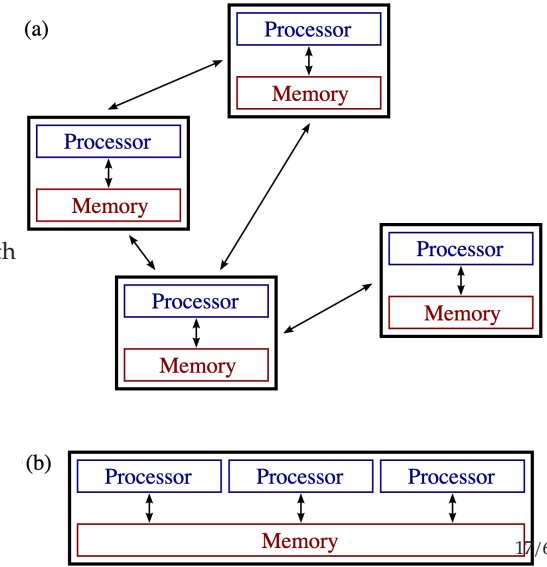
The kernel

The privileged core of the operating system, sharing the machine's resources among programs.

- **Process**: a program in execution, the unit the kernel schedules and isolates, with its own address space, registers, and open files.
- **System call**: a process's only entry into the kernel. It traps from user mode to kernel mode, runs a privileged operation, then returns.
- **CPU**: the scheduler decides which process runs, and for how long.
- **Memory**: a private virtual address space per process.
- **Storage**: the virtual file system (VFS) presents many filesystems as one.
- **Network**: the TCP/IP stack and sockets carry data between machines.
- **Devices**: drivers expose hardware through uniform interfaces.

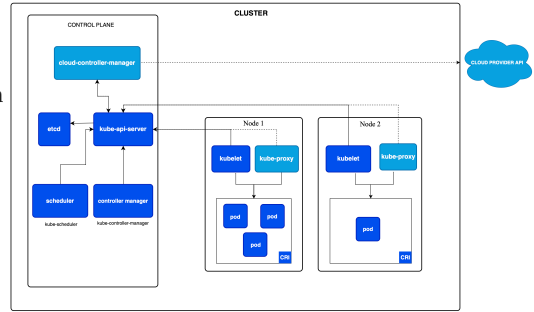
Distributed systems

- **No shared memory**: every exchange is explicit.
- **No shared clock**: no global ordering of events.
- **Independent failure**: any node can crash or slow down.
- **Communication**: message passing, or remote calls over `gRPC`.
- **Reliability**: timeouts, retries with idempotency, health checks.



Cloud-native

- **Cloud**: infrastructure as a utility, provisioned on demand through APIs and paid by use.
- **Virtual machines**: each runs its own guest kernel on a hypervisor. Strong isolation, but heavy, gigabytes and seconds to boot.
- **Containers**: share the host kernel, isolated by namespaces and cgroups. A reproducible image that starts in milliseconds.
- **Kubernetes**: declare the desired state, and controllers reconcile the cluster toward it, continuously.
- **Extensible**: `CRDs` add resource types, `Operators` give them behaviour, `Device Plugins` schedule hardware.



Kubernetes cluster architecture.

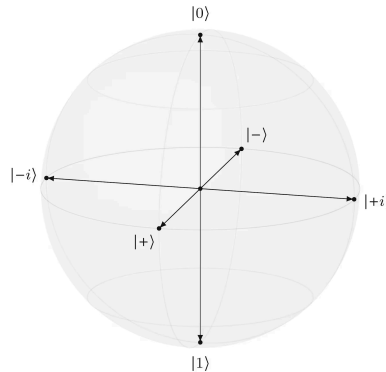
Observability

- **Kálmán** (1960): reconstruct a system's internal state from its outputs.
- In software, that is telemetry: metrics, traces, logs, profiles.
- The power comes from correlating them.
- **Cardinality**: a high-cardinality field, like a unique request id, pinpoints one event among millions, at a storage cost.
- **OpenTelemetry**: the vendor-neutral standard, with semantic conventions for attribute names.
- **Pipeline**: instrument once. The Collector and exporters route each signal to its store, so the backend is a config change, not code.

Quantum Computing

Quantum information and gates

- **Qubit**: $|\psi\rangle = \alpha\,|0\rangle + \beta\,|1\rangle$, a blend of 0 and 1.
- **Measurement** collapses it at random: a run returns statistics, not one value.
- **Phase and interference**: wrong answers cancel, correct ones reinforce.
- **Coherence is finite**: T_1 and T_2 bound how long a run has.
- **Exponential state**: n qubits hold 2^n amplitudes, beyond any classical simulation.
- **Entanglement**: a joint state that does not factor into independent qubits (Bell states).
- **Reversible gates**: unitary, e.g. Pauli X, Y, Z and Hadamard.
- **No-cloning**: a qubit cannot be copied.



A pure single-qubit state on the Bloch sphere.

The quantum computer: four planes

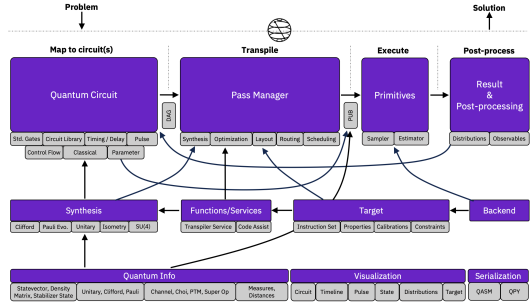
A QPU is an accelerator behind a classical host, like a GPU, so every computation is hybrid. It runs no operating system of its own. Across qubit technologies the National Academies report gives one shared architecture:

Plane	Role and key property
Quantum data	Holds the qubits and runs the gates. Keeps no persistent state, bounded by coherence time.
Control & measurement	Drives the qubits and reads them out. Drifts, so needs recalibration, often daily.
Control processor	Sequences operations in real time.
Host processor	Runs the SDK, transpiler, and scheduler. Holds all persistent state.

The first two planes are the QPU, the quantum hardware. The control processor and host are classical, and their boundary with the QPU is the classical–quantum interface this thesis addresses. Because the data plane keeps no state, scheduling and calibration live on the classical side, and today's NISQ devices remain far from perfect.

Compilation and runtime

- **A circuit cannot run as written**: each device has its own native gates and connectivity.
- **The stack mirrors the classical one**: SDK $\rightarrow$ IR $\rightarrow$ transpiler $\rightarrow$ ISA $\rightarrow$ pulses $\rightarrow$ QPU.
- **The transpiler** maps a circuit to a device: layout, routing, translation, optimisation, scheduling.
- **Routing** is NP-hard, the dominant fidelity cost.
- **Temporally-dynamic ISA**: calibration drifts daily, so transpile against the live calibration.



The Qiskit workflow (Javadi-Abhari et al. 2024, Fig. 2).

Literature review and the gap

Review methodology

- The layer model is a thesis-specific synthesis, not a taxonomy, aligning the QPU plane, QCSC, and the HPC–QC interfacing survey.
- Each layer: its problem, approaches, and residual gap.
- Scope runs 2016 (IBM Quantum Experience) to 2026 (QCSC).
- Drawn from IEEE, ACM, arXiv, *Nature*, and *Physical Review*.
- Weighted toward the IBM ecosystem: its systems references, mature open Qiskit and OpenQASM, the coursework platform.

Applications

- A circuit has no process environment. A classical system submits it as a job, so every workload is hybrid.
- VQE (Peruzzo, 2014) and QAOA make the loop visible. The same coupling surrounds Shor's deep circuit.
- Hoefler (2023): only super-quadratic speedups are worthwhile (chemistry, factoring), and these are the deepest circuits.
- A correct result depends on the whole stack on the day (Kim, 2023).
- Hardware drifts between runs, so reproducibility means recording backend state, done by hand today.

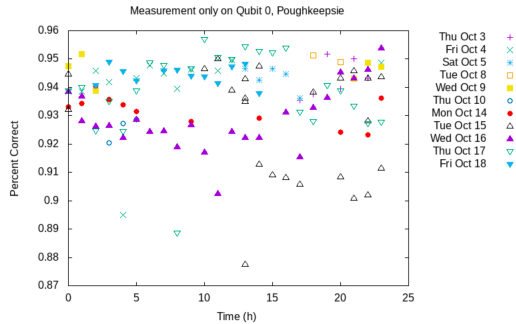
Frameworks and SDKs

SDK	Developer	Abstraction	Multi-vendor posture
Qiskit	IBM	Circuit + primitives	IBM-native, provider-extensible
Cirq + TFQ	Google	Circuit / differentiable	Vendor-native (Google)
CUDA-Q	NVIDIA	Kernel-based	QPU-agnostic, GPU-coupled
PennyLane	Xanadu	Differentiable	Plugin-based (vendor-neutral)
t|ket⟩	Quantinuum	Compiler IR	Retargetable compiler frontend
OpenQASM 3	(interchange)	Text IR	De-facto interchange, not hardware

- The SDK layer is where vendor coupling concentrates. Portability is real at the SDK (PennyLane, CUDA-Q), compiler (t|ket⟩), and interchange (OpenQASM) levels.
- But execution still delegates to a vendor-specific runtime, so that neutrality does not propagate up. Orchestration and observability must build their own.

Compilation

- Compilation maps a circuit to one device (layout, routing, native-gate rewrite). Routing is NP-complete (Siraichi, 2018), handled by the stochastic SABRE heuristic (Li, 2019).
- The transpiler reads the last published calibration snapshot, not a live reading.
- Wilson (2020): fidelity drifts chaotically between daily calibrations, and the snapshot often goes stale before the job runs (Ravi, 2021).
- Just-in-time transpilation against fresh data recovers accuracy by +3–304%, so compilation is a runtime concern. Yet no toolchain reads the device live.
- Hybrid loops re-transpile every iteration, so re-transpilation must be cheap. QASMTrans (C++) runs 10–369× faster than Qiskit.



Qubit fidelity between daily calibrations (Wilson, 2020).

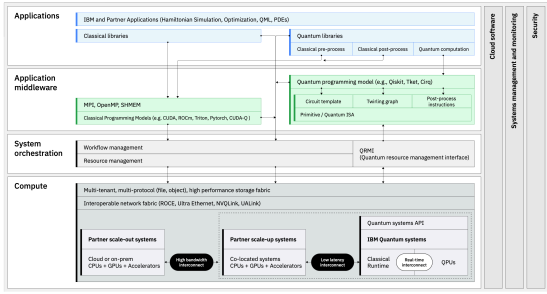
Infrastructure and orchestration

- Access is first-party clouds (deep instrumentation, vendor lock-in) or aggregators (portable, but unable to see into the backend).
- Queues vary widely, with a median wait near an hour (Ravi, 2021) and up to 38% fidelity spread (Giortamis). The best backend is the most contended, the fidelity-versus-wait tradeoff.

System	Substrate / scheduler	Reach & maturity
Qubernetes	K8s Job, default scheduler	single QPU (HELMI), no fidelity awareness
QRMI	vendor-neutral middleware (Slurm, K8s)	IBM and PASQAL adapters, acquire/execute/release
Qonductor	K8s scheduler, NSGA-II	fidelity/runtime prediction, single-vendor, simulation

QCSC reference architecture

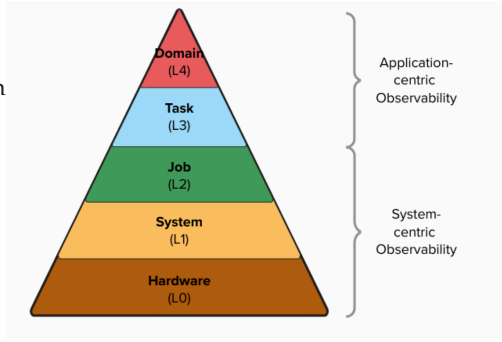
- QCSC (Seelam, 2026) synthesises the orchestration systems into one reference architecture.
- Four layers: Hardware Infrastructure, System Orchestration, Application Middleware, Applications.
- Three cross-cutting concerns: Cloud Software, System Management and Monitoring, Security.
- QRMI sits inside System Orchestration as a lightweight library, used by the resource manager (Slurm in the reference integration).
- It is a specification, not a deployed cross-vendor system.



QCSC reference architecture (Seelam, 2026).

Observability

- Classical observability is standardised (Prometheus, OpenTelemetry). Quantum has no convention for calibration, queue, or backend selection, and no quantum namespace in the OTel registry.
- Every system reaches for the classical stack but stops short of quantum semantics: QCSC calls for QPU exporters but leaves the implementation open, Qubernetes places quantum metrics out of scope, Qonductor keeps telemetry internal to its scheduler, and QRMI exposes a metrics group but leaves its schema open.
- Providers expose per-job statistics through REST, not scrapeable telemetry.
- The two gaps are a metrics specification and a telemetry pipeline.



Kanazawa pyramid (2025): the top two levels (task, domain) need the workflow orchestrator.

Gap summary

Layer	Core gap
Application	Runs are not reproducible. The hardware changes between runs and is recorded only by hand, so a result cannot be traced to the code or the machine.
Framework	Neutrality stops at the code. Circuits are portable, but execution stays vendor-specific, so the layers above cannot inherit it.
Compilation	No just-in-time transpilation. The compiler uses a stale published snapshot, not the live device, so a job can run against a device that no longer matches it.
Orchestration	No shared cross-vendor approach. The neutral system is a library, and the most-tested ran on a single vendor.
Observability	No quantum telemetry standard. Nothing defines a quantum signal, so a metrics specification and a telemetry pipeline are both missing.

- At every layer the techniques exist. The missing piece is the standardisation layer that makes them repeatable across providers.

Architecture: the Quantum Circuit Controller

Requirements for the Architecture

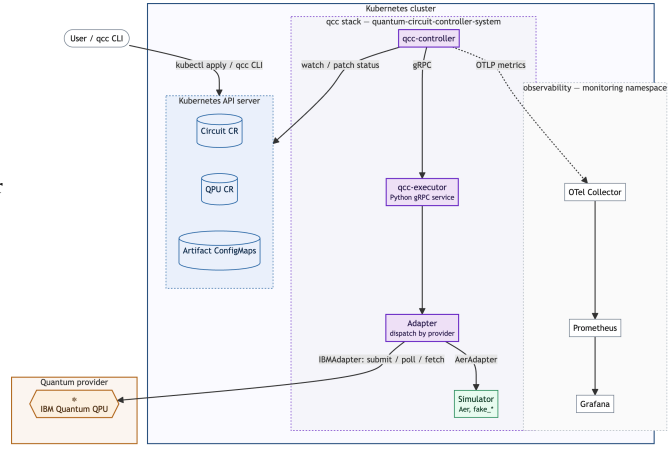
Five criteria drawn from the Chapter 5 gaps. R1 to R4 are standard SRE expectations. R5 is quantum-specific.

Req.	Name	Requirement
R1	Declarative infrastructure	Stand up on commodity hardware so one developer runs circuits without specialised infrastructure. HPC stays a target, not a prerequisite.
R2	Observable through standards	Surface every circuit and backend's state over industry-standard telemetry, into any existing stack.
R3	Portable and modular	Each provider behind an out-of-tree adapter against a stable contract, so a new provider is additive.
R4	Correlated across the boundary in real time	Follow a circuit across the boundary via one identifier that resolves both directions.
R5	Monitored backend quality	Backend quality is a measured signal. Its characteristics and each run's outcome are first-class telemetry.

System overview

QCC is a Kubernetes operator: two custom resources driven by a controller, with all quantum work behind an executor.

- A request becomes a `Circuit` on the API server, the system's only entry point.
- The controller selects a registered `QPU` and drives the lifecycle, loading no quantum SDK.
- Quantum work is delegated to the executor over gRPC. An adapter dispatches by provider (local Aer or IBM Quantum).
- All telemetry originates in the controller, emitted over open standards.



Control plane components

Component	Kind	Role
`qcc-controller`	Go, Deployment	Reconciles the resources, drives the lifecycle, emits telemetry. Imports no quantum SDK.
`qcc-executor`	Python, gRPC	The only component that imports Qiskit: conversion, transpilation, submission, retrieval.
`qcc` CLI	Go, user-side	Builds `Circuit` resources from local files, submits via the Kubernetes API.
`Circuit`	CR, namespaced	Declarative submission unit: one execution intent.
`QPU`	CR, cluster-scoped	Registered backend, with probe-discovered status.
`Adapter`	Python module	One class per provider (`local`, `ibm`), registered by the `provider` string. A new provider is one more adapter.

Interacting with QCC: the CLI

`qcc` builds `Circuit` resources from local files and submits them through the Kubernetes API. Everything it does is equally reachable through `kubectl`.

```
qcc run bell.qasm                                  # submit and wait
qcc run shor.py --shots 4096
qcc run bell.qasm --backend ibm_kingston --detach  # submit and exit, check later
qcc run shor.py --performance-test --algorithm shor
qcc get qpus                                       # list registered backends
qcc get circuit shor-7p4jp                         # one run's status and artefacts
```

- **Modes**: `run` executes. `draw` and `schedule` render an ASCII diagram or a per-instruction timeline. `get` inspects, `kubectl`-style.
- **`--detach`**: submit a hardware job and exit once it is queued. The controller keeps polling in the background.
- **`--performance-test`**: run the same circuit across every simulator (and real hardware with `--include-hardware`) and compare them.

The Circuit resource

The declarative submission unit, with a two-tier spec. **Tier 1** is the typed vocabulary QCC owns. **Tier 2** blocks pass straight to Qiskit, so new SDK options need no schema change.

```
apiVersion: qcc.io/v1alpha1
kind: Circuit
metadata:
  name: bell-state
  labels:
    qcc.io/algorithm: bell-state
    qcc.io/algorithm-version: v1
spec:
  mode: run
  shots: 1024
  source:
    format: openqasm3
    body: |
      OPENQASM 3.0;
      include "stdgates.inc";
      qubit[2] q;
      bit[2] c;
      h q[0];
      cx q[0], q[1];
      c[0] = measure q[0];
      c[1] = measure q[1];
  backendSelector:
    kind: simulator           # Tier 1: typed, QCC-owned
  transpile:
    scheduling_method: alap   # Tier 2: passed straight to Qiskit
```

The QPU resource

A registered backend. The operator declares how to reach it. The controller discovers what it is by probing the backend through the executor.

Declared spec

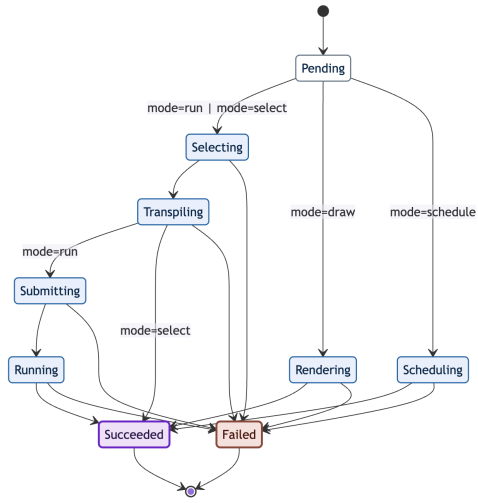
```
apiVersion: qcc.io/v1alpha1
kind: QPU
metadata:
  name: ibm-kingston
  labels:
    qcc.io/provider: ibm
    qcc.io/family: heron-r2
spec:
  provider: ibm
  backendName: ibm_kingston
  kind: hardware
  access:
    credentialSecretRef:
      name: ibm-quantum-token
      namespace:
        quantum-circuit-controller-system
  capabilities:
    maxShots: 100000
```

Discovered status for `ibm-kingston` after the probe

```
# kubectl get qpu ibm-kingston -o yaml  (status only)
status:
  availability: Available
  qubits: 156
  processor: {family: Heron, revision: "2"}
  couplingEdges: 352
  basisGates: [cz, delay, id, if_else, measure,
               measure_2, reset, rz, sx, x]
  lastCalibrationTime: "2026-05-16T10:35:26Z"
  coherenceMedians: {t1Micros: 174.6, t2Micros: 117.0}
  errorMedians: {singleQubit: 0.000274, twoQubit: 0.00203,
                 readout: 0.0165}
  instructionDurationMedians: {singleQubitSeconds: 3.2e-08,
                               twoQubitSeconds: 6.8e-08}
  dtSeconds: 4e-09
  conditions:
  - type: Ready
    status: "True"
    reason: ProviderProbeOK
    message: ibm provider is Available; live calibration via probe
```

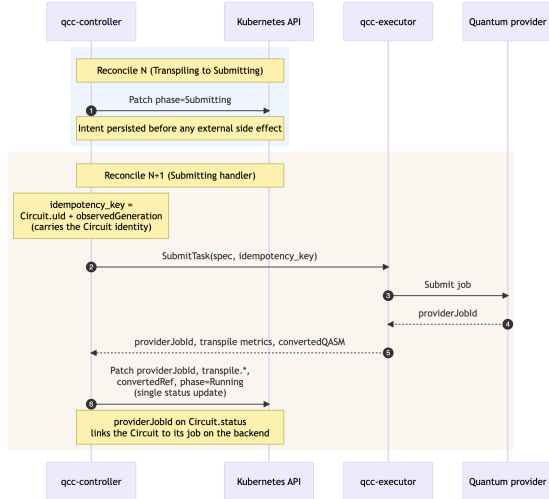
Execution lifecycle: state machine

- Four mode paths chosen by `spec.mode`: `run`, `select`, `draw`, `schedule`.
- All begin at `Pending` and end at `Succeeded` or `Failed`. The modes share early phases and diverge by how far they go.
- `Submitting` commits the job to the backend, the lifecycle's only external call.



The cross-boundary identifier

- QCC sends the circuit to the backend with a unique key, so a retry never runs it twice.
- The backend takes the job and returns its own job id.
- QCC saves that job id on the Circuit, so it is kept even if the controller restarts.
- The id links both ways: from a Circuit to its backend job, and from a job id back to its Circuit.



Observability

Observability is QCC's principal contribution. It instruments the quantum–classical boundary with the same telemetry stack used for any cloud-native service.

- A vendor-neutral pipeline: the OpenTelemetry SDK to an OTLP Collector to a stock `kube-prometheus-stack`, read in Grafana.
- Fourteen metrics in two families: `qcc_qpu_*` for backend health, `qcc_circuit_*` for each run.
- Two standard operator patterns: resource-state gauges (`kube-state-metrics`) and operational-event counters and histograms (`controller-runtime`).

Observability: QPU metrics

Six observable gauges populated from `QPU.status` on each scrape. All series carry a `qpu` identity label.

Metric	Labels
`qcc_qpu_info`	`uid, provider, kind, processor_family, processor_revision`
`qcc_qpu_operation_error_median`	`operation` $\in$ {`gate_1q`, `gate_2q`, `readout`}
`qcc_qpu_operation_duration_median_seconds`	`operation` $\in$ {`gate_1q`, `gate_2q`}
`qcc_qpu_coherence_seconds`	`type` $\in$ {`t1`, `t2`}
`qcc_qpu_last_calibration_timestamp_seconds`	(none)
`qcc_qpu_condition`	`condition, status` $\in$ {`true, false, unknown`}

Observability: Circuit metrics

All series carry `circuit`, `namespace`, and `uid` as identity labels.

Metric	Type	Labels
`qcc_circuit_info`	gauge	`mode, shots, qpu, provider_job_id, algorithm, run_index`
`qcc_circuits_total`	counter	`mode, qpu, phase, reason, provider_job_id`
`qcc_circuit_phase_duration_seconds`	histogram	`qpu, phase, provider_job_id`
`qcc_circuit_phase_duration_seconds_observed`	gauge	`qpu, phase, provider_job_id`
`qcc_circuit_usage_seconds`	gauge	`qpu, provider_job_id`
`qcc_circuit_transpile_depth`	gauge	`qpu`
`qcc_circuit_transpile_gates`	gauge	`qpu, kind` $\in$ {`single_qubit, two_qubit, total`}
`qcc_circuit_result_count`	gauge	`qpu, bitstring`

Implementation and demonstration

Bringing up the platform

- QCC v1.0.0 ships as a declarative package: controller, executor, `qcc` CLI, a local cluster, the observability stack, and two Grafana dashboards.
- Evaluated on a local `kind` cluster (v0.30.0), Docker via Colima 0.9.1 on Apple Silicon.
- Stock `kube-prometheus-stack` and OpenTelemetry Collector, installed with Helm.

Registering a backend

- A backend enters the platform as a manifest declaring only provider and kind.
- After the periodic probe, the discovered status is written to the CRD `status`: qubit count, basis gates, coupling-map size, calibration vintage, and per-operation error medians.

```
$ kubectl apply -f fake-fez.yaml
$ kubectl get qpu fake-fez -o yaml
```

```yaml
apiVersion: qcc.io/v1alpha1
kind: QPU
metadata:
  name: fake-fez
  labels:
    app.kubernetes.io/name: quantum-circuit-controller
    app.kubernetes.io/managed-by: kustomize
    qcc.io/provider: local
    qcc.io/family: heron-r2
spec:
  provider: local
  kind: simulator
```

The minimal `fake-fez` manifest: provider and kind only.

The probe-populated `QPU` status.

The QPU registry

```
$ kubectl get qpu
```

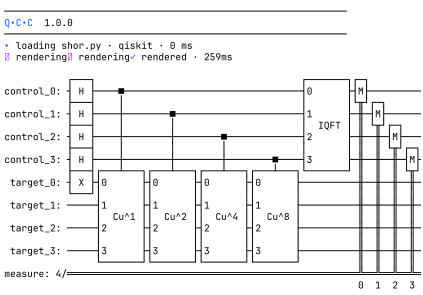
- Three live IBM Heron r2 backends (`ibm-fez`, `ibm-kingston`, `ibm-marrakesh`) registered alongside the simulators.

```
NAME                 PROVIDER   KIND        QUBITS   AVAILABLE   AGE
aer-statevector      local      simulator   28       Available   7d22h
fake-belem           local      simulator   5        Available   9d
fake-brisbane        local      simulator   127      Available   9d
fake-brisbane-test   local      simulator   127      Available   6d20h
fake-fez             local      simulator   156      Available   2m1s
fake-kyoto           local      simulator   127      Available   9d
fake-marrakesh       local      simulator   156      Available   9d
fake-osaka           local      simulator   127      Available   9d
fake-sherbrooke      local      simulator   127      Available   9d
fake-torino          local      simulator   133      Available   9d
ibm-fez              ibm        hardware    156      Available   7d22h
ibm-kingston         ibm        hardware    156      Available   7d22h
ibm-marrakesh        ibm        hardware    156      Available   7d22h
```

The workload: Shor's circuit

- Shor $N = 15$, $a = 4$, the same circuit first run on `ibm_sherbrooke` in the Motivation.
- `mode=draw` renders the circuit through the CLI without consuming QPU time, the first feedback step in development.
- Success is scored as *correct-period mass* $=$ (count[0000] + count[1000])/shots. The ideal run gives 100%.

```
$ qcc draw examples/thesis/algorithms/shor.py
```



Running on the ideal simulator

```
$ qcc run examples/thesis/algorithms/shor.py \
    --algorithm shor --experiment thesis \
    --version v1 --backend aer-statevector
```

- Produces Circuit `shor-lpdb7`, completed in 510 ms (transpiled depth 506).
- All 1024 shots on the two period states ($517 + 507$).
- Correct-period mass $= 100\%$, the noise-free baseline.

```
Q·C·C  1.0.0

▸ loading shor.py · qiskit · 0 ms
▸ submitted default/shor-lpdb7
⠏ waiting for selection⠏ waiting for selection⠏ waiting for selection⠏
✓ queued · job aer-eccea297-b1f6-49e9-a044-f50a5225474e
✓ completed · 510ms

results

  ✓ shor-lpdb7 · run · on aer-statevector succeeded

  setup
    phase   Succeeded
    source  qiskit
    shots   1024
    job     aer-eccea297-b1f6-49e9-a044-f50a5225474e

  backend
    name  aer-statevector  (28q · 87 basis gates)

  circuit
    transpiled  depth 506 · 665 gates · 288 2Q

  results

    0000  ████████████████████  517
    1000  ███████████████████▒  507

▸ qasm qcc get circuit shor-lpdb7 --qasm
```

The run as a Kubernetes resource

```
$ kubectl get circuits shor-lpdb7 -o yaml
```

- Every run is a `Circuit` custom resource, inspected like any other Kubernetes object.


```
apiVersion: qcc.io/v1alpha1
kind: Circuit
metadata:
  creationTimestamp: "2026-05-24T09:44:25Z"
  generation: 1
  labels:
    qcc.io/algorithm: shor
    qcc.io/algorithm-version: v1
    qcc.io/experiment: thesis
    qcc.io/run-index: "1"
    qcc.io/source-sha256: a721bd0a0a24dad6
  name: shor-lpdb7
  namespace: default
  resourceVersion: "793723"
  uid: 0a5d99cc-9486-4678-a333-54e63a845475
spec:
  backendSelector:
    backendName: aer-statevector
  mode: run
  shots: 1024
  source:
    body: |
      """
      Shor's algorithm — quantum period-finding for N=15, a=4.
                        section suppressed
      the same one whose ibm brisbane / ibm sherbrooke runs open Chapter 1, kept
```


`spec`: declared intent, reserved `qcc.io/*` labels, selected backend.


```
status:
  conditions:
  - lastTransitionTime: "2026-05-24T09:44:25Z"
    message: Circuit accepted by controller
    observedGeneration: 1
    reason: CircuitAccepted
    status: "True"
    type: Accepted
  - lastTransitionTime: "2026-05-24T09:44:25Z"
    message: Static validation succeeded
    observedGeneration: 1
    reason: CircuitValidated
    status: "True"
    type: Validated
  - lastTransitionTime: "2026-05-24T09:44:25Z"
    message: Selected QPU "aer-statevector" (provider=local, backend=aer_statevector)
    observedGeneration: 1
    reason: BackendSelected
    status: "True"
    type: Selected
  - lastTransitionTime: "2026-05-24T09:44:25Z"
    message: Submitted to aer_statevector (taskId=aer-eccea297-b1f6-49e9-a044-f50a5225474e)
    observedGeneration: 1
    reason: ProviderSubmitted
    status: "True"
    type: Submitted
  - lastTransitionTime: "2026-05-24T09:44:25Z"
    message: Execution completed with 2 outcome(s)
    observedGeneration: 1
    reason: ExecutionCompleted
    status: "True"
    type: Completed
  convertedCircuit:
    name: shor-lpdb7-converted
  observedGeneration: 1
  phase: Succeeded
  providerJobId: aer-eccea297-b1f6-49e9-a044-f50a5225474e
  results:
    "0000": 517
    "1000": 507
  selectedQPU: aer-statevector
  selectionSummary:
    candidates: 1
    score: 'n/a (M1: first-match policy; M2 adds calibration-aware scoring)'
    selected: aer-statevector
  transpile:
    depth: 586
    totalGates: 665
    twoQubitGates: 288
```


`status`: conditions, selected backend, measured counts, transpile shape.

Comparing the simulators

```
$ qcc run examples/thesis/algorithms/shor.py \
    --performance-test \
    --algorithm shor \
    --version v1 \
    --experiment thesis-perf-test
```

- One run per simulator, collected into a single comparison table.
- `fake-belem` (5 qubits): `TranspilationFailed`, the circuit does not fit.
- Heron r2 keeps `0000/1000` on top, while Eagle r3 loses the period signal.

```
Q·C·C  1.0.0

• loading shor.py · qiskit · 8 ms
• performance test · 10 candidates algorithm=shor · experiment=thesis-perf-test
• submitted default/shor-zcztt → aer-statevector
• submitted default/shor-n7mbm → fake-belem
• submitted default/shor-kg7nr → fake-brisbane
• submitted default/shor-xnrck → fake-brisbane-test
• submitted default/shor-n4n5v → fake-fez
• submitted default/shor-s8626 → fake-kyoto
• submitted default/shor-qpp8j → fake-marrakesh
• submitted default/shor-ggvf5 → fake-osaka
• submitted default/shor-4x992 → fake-sherbrooke
• submitted default/shor-rcjm9 → fake-torino
• completed shor-zcztt on aer-statevector · 1.395s
✗ failed · shor-n7mbm on fake-belem TranspilationFailed
• completed shor-kg7nr on fake-brisbane · 3.186s
• completed shor-xnrck on fake-brisbane-test · 3.783s
• completed shor-n4n5v on fake-fez · 9.181s
• completed shor-s8626 on fake-kyoto · 11.779s
• completed shor-qpp8j on fake-marrakesh · 16.776s
• completed shor-ggvf5 on fake-osaka · 18.375s
• completed shor-4x992 on fake-sherbrooke · 19.579s
• completed shor-rcjm9 on fake-torino · 23.305s

results · thesis-perf-test

 BACKEND            PHASE                      DEPTH  1Q    2Q   TOTAL  TOP OUTCOMES                  TIME     JOB
 aer-statevector    Succeeded                  506    377   288  665    0000:520 1000:504             1.395s   aer-1a7f54b2-…
 fake-belem         FAILED · TranspilationFailed  –    –     –    –                                    1.79s    –
 fake-brisbane      Succeeded                  1785   2358  477  2835   1000:124 0000:119 0001:69     3.186s   aer-ed9b7cef-…
 fake-brisbane-test Succeeded                  1811   2348  474  2822   1000:155 0000:145 1111:86     3.783s   aer-87f1dbfe-…
 fake-fez           Succeeded                  2862   1983  483  2466   0000:236 1000:217 1001:77     9.181s   aer-ad1e59db-…
 fake-kyoto         Succeeded                  1774   2315  474  2789   0000:81 1100:73 1111:72       11.779s  aer-f35a7344-…
 fake-marrakesh     Succeeded                  2868   1995  483  2478   0000:289 1000:239 1001:66     16.776s  aer-d5b47da9-…
 fake-osaka         Succeeded                  1811   2348  474  2822   0000:125 1000:115 0001:92     18.375s  aer-7251fceb-…
 fake-sherbrooke    Succeeded                  1789   2348  474  2822   0000:103 1001:83 1000:75      19.579s  aer-e2faf5c6-…
 fake-torino        Succeeded                  2848   1967  474  2441   0000:203 1000:185 1001:69     23.305s  aer-194314a9-…

• dashboard /dj-qcc-circuit?var-algorithm=shor&var-experiment=thesis-perf-test
```

Submitting to real hardware: `--detach`

- The vendor queue runs minutes to hours, so waiting synchronously is unworkable.
- `--detach` exits as soon as the provider job is queued.
- The controller keeps polling in the background.

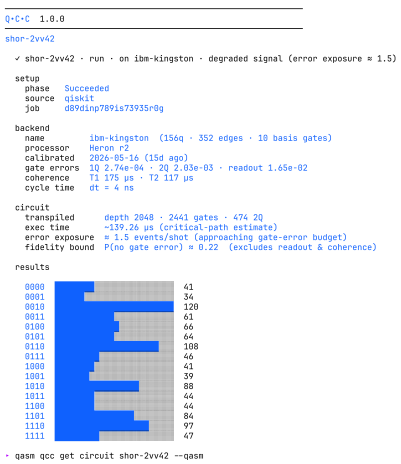
```
$ qcc run examples/thesis/algorithms/shor.py \
    --algorithm shor --experiment thesis --version v1 \
    --backend ibm-kingston --detach
```

```
Q•C•C  1.0.0

▸ loading shor.py · qiskit · 0 ms
▸ submitted default/shor-2vv42
✓ targeting ibm-kingston
✓ queued · job d89dinp789is73935r0g

▸ detached check progress with: qcc get circuit shor-2vv42
```

Prints the Circuit name, UID, and provider job id: `shor-2vv42`, `d89dinp789is73935r0g`.

Real hardware: reading the result card

```
$ qcc get circuit shor-2vv42
```



- `backend`: `ibm-kingston`, Heron r2, 156 qubits, calibrated 15 days prior, with its 1Q, 2Q, and readout error and T_1/T_2 coherence.
- `job`: the IBM provider job id, the cross-boundary link to the vendor run.
- `transpiled`: depth 2048, 2441 gates, 474 two-qubit gates.
- `results`: the 16-outcome histogram, with period states `0000` and `1000` at 41 each, giving $(41{+}41)/1024 \approx 8.0\%$.

Reading the card: the quality indicators

Three indicators score the run from the transpiled circuit and the backend's calibration *medians*, the typical per-operation value across the device's qubits and gates.

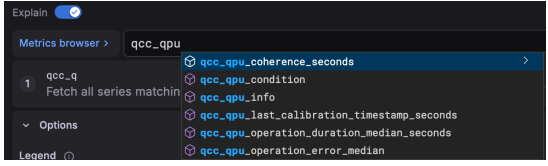
- `exec time`: how long the circuit occupies the QPU, a critical-path estimate. The median gate durations are $t_{1Q} = 32\,\text{ns}$ and $t_{2Q} = 68\,\text{ns}$, so $\max(t_{1Q}, t_{2Q}) = t_{2Q} = 68\,\text{ns}$ (the two-qubit gate is slower). $\text{depth} \times \max = 2048 \times 68\,\text{ns} \approx 139\,\mu\text{s}$, close to $T_2 = 117\,\mu\text{s}$, so coherence decay matters.
- `error exposure`: the expected gate-error events per shot.

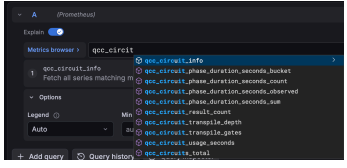
$$n_{1Q}\varepsilon_{1Q} + n_{2Q}\varepsilon_{2Q} = 1967 \times 2.74 \times 10^{-4} + 474 \times 2.03 \times 10^{-3} \approx 0.54 + 0.96 \approx 1.5.$$

  Bands: preserved < 0.1, noisy ≥ 0.1, degraded ≥ 1, lost ≥ 5.
- `fidelity bound`: the share of shots expected to run error-free. $P(\text{no gate error}) = e^{-1.5} \approx 0.22$. Gate errors only, so an optimistic bound.

The qcc.* metrics in Prometheus

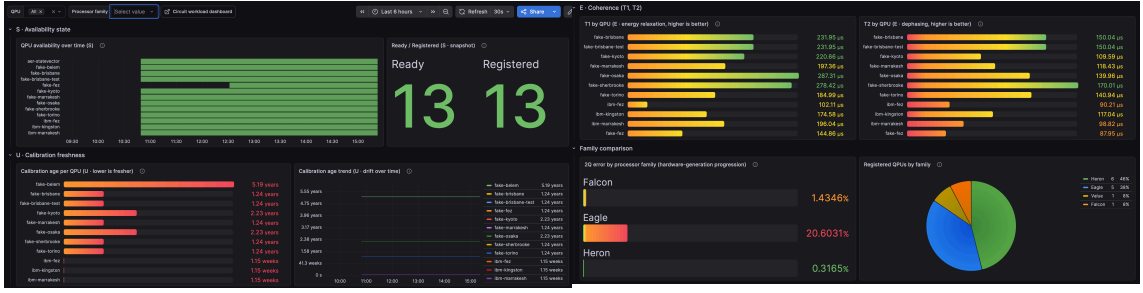
- The `qcc.*` specification emits as standard Prometheus series: the `qcc_qpu_*` family (backends) and the `qcc_circuit_*` family (runs).
- Collected by the stock stack and queryable with PromQL.
- Prometheus is a time-series database, so the signals are retained and queryable over time, not only at the moment of execution.





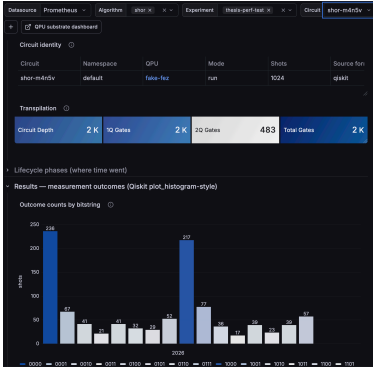
QPU dashboard

- USE (Utilisation, Saturation, Errors): a resource-oriented method. The QPU dashboard is organised around it, reading the backend as the resource.
- Here: Saturation as availability, Utilisation as calibration freshness, Errors as the gate and coherence envelope.
- A design choice for this prototype, not a settled methodology.

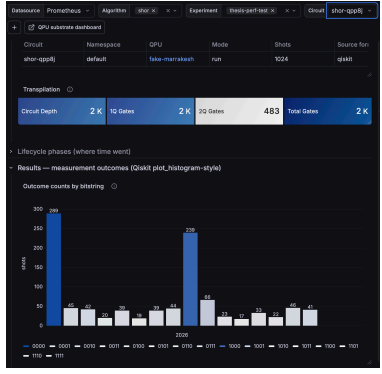


Circuit dashboard

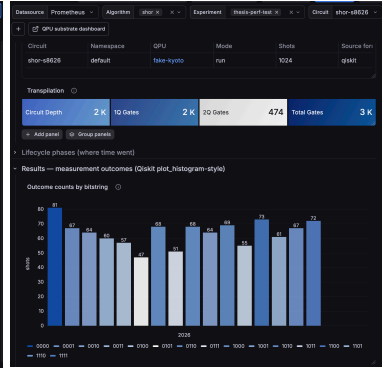
- RED (Rate, Errors, Duration) per-run view: circuit metrics per run (phase, transpiled size, results), correlated by the labels added at execution: algorithm, experiment, version.



fake-fez



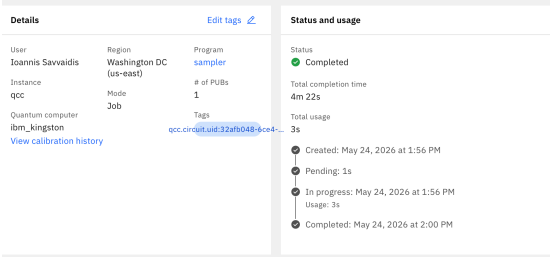
fake-marrakesh



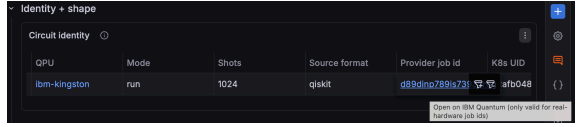
fake-kyoto (Eagle r3)

Cross-boundary correlation, both directions

- Forward: the executor stamps the Circuit's Kubernetes UID onto the IBM job as a `qcc.circuit.uid` tag.
- Reverse: `provider_job_id` on `status.providerJobId` renders as an *Open on IBM Quantum* deep-link.
- Both resolve while the job runs and after it terminates, using standard QCC and vendor tooling.



Forward: UID tag on the IBM Console.



Reverse: deep-link from Grafana.

Tuning the transpiler: the two-tier edit

- **Tier 1**: stable orchestration fields, typed in the CRD and supported by the CLI (`shots`, `optimizationLevel`).
- **Tier 2**: a passthrough block forwarded to the executor unchanged (`transpile.scheduling_method`). Submitted with `kubectl create -f`, as it is not in the CLI or typed in the CRD.

`shor-v2.yaml`

```yaml
apiVersion: qcc.io/v1alpha1
kind: Circuit
metadata:
  generateName: shor-tuned-kingston-
  labels:
    qcc.io/algorithm: shor
    qcc.io/algorithm-version: v2
    qcc.io/experiment: thesis
spec:
  mode: run
  shots: 4096
  optimizationLevel: 3
```

`shor-v3.yaml`

```yaml
apiVersion: qcc.io/v1alpha1
kind: Circuit
metadata:
  generateName: shor-tuned-kingston-
  labels:
    qcc.io/algorithm: shor
    qcc.io/algorithm-version: v3
    qcc.io/experiment: thesis
spec:
  mode: run
  shots: 4096
  optimizationLevel: 3
  transpile:
    scheduling_method: alap
```

Tuning the transpiler: the result cards

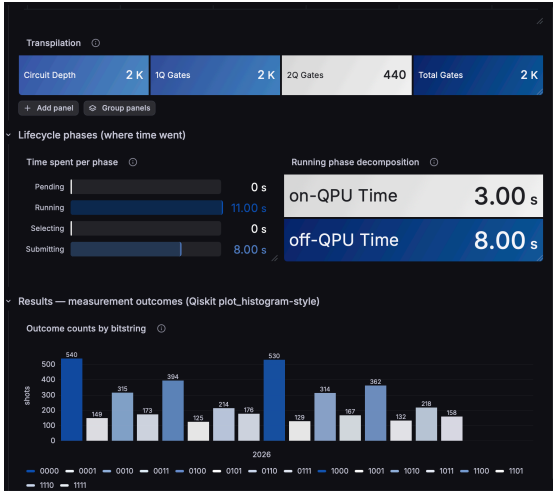
```
Q·C·C  1.0.0

shor-tuned-kingston-nb7q9

  ✓ shor-tuned-kingston-nb7q9 · run · on ibm-kingston · degraded signal (error exposure ≈ 1.3)

  setup
    phase   Succeeded
    source  qiskit
    shots   4096
    job     d89e1fqs46sc73fah88g

  backend
    name          ibm-kingston  (156q · 352 edges · 10 basis gates)
    processor     Heron r2
    calibrated    2026-05-16 (8d ago)
    gate errors   1Q 2.74e-04 · 2Q 2.03e-03 · readout 1.65e-02
    coherence     T1 175 µs · T2 117 µs
    cycle time    dt = 4 ns

  circuit
    transpiled       depth 1523 · 2054 gates · 440 2Q
    exec time        ~103.56 µs (critical-path estimate)
    error exposure   ≈ 1.3 events/shot (approaching gate-error budget)
    fidelity bound   P(no gate error) ≈ 0.26  (excludes readout & coherence)

  results

    0000 ████████████████████ 540
    0001 ██████               149
    0010 ████████             315
    0011 ██████               173
    0100 ████████████         394
    0101 ██████               125
    0110 ███████              214
    0111 ██████               176
    1000 ████████████████     530
    1001 ██████               129
    1010 ████████             314
    1011 ██████               167
    1100 ████████████         362
    1101 ██████               132
    1110 ███████              218
    1111 ██████               158

▸ qasm qcc get circuit shor-tuned-kingston-nb7q9 --qasm
```

```
Q·C·C  1.0.0

shor-tuned-kingston-62qzb

  ✓ shor-tuned-kingston-62qzb · run · on ibm-kingston · degraded signal (error exposure ≈ 1.5)

  setup
    phase   Succeeded
    source  qiskit
    shots   4096
    job     d89el10p0eas73doh6u0

  backend
    name          ibm-kingston  (156q · 352 edges · 10 basis gates)
    processor     Heron r2
    calibrated    2026-05-16 (8d ago)
    gate errors   1Q 2.74e-04 · 2Q 2.03e-03 · readout 1.65e-02
    coherence     T1 175 µs · T2 117 µs
    cycle time    dt = 4 ns

  circuit
    transpiled       depth 1524 · 2548 gates · 442 2Q
    exec time        ~103.63 µs (critical-path estimate)
    error exposure   ≈ 1.5 events/shot (approaching gate-error budget)
    fidelity bound   P(no gate error) ≈ 0.23  (excludes readout & coherence)

  results

    0000 ████████████████████ 554
    0001 ████████             139
    0010 ███████              298
    0011 ████████             144
    0100 ████████████         350
    0101 ███████              110
    0110 ████████             253
    0111 ████████             238
    1000 ████████████████     546
    1001 ███████              150
    1010 ███████              288
    1011 ████████             143
    1100 ████████████         349
    1101 ███████              110
    1110 ████████             237
    1111 ████████             187

▸ qasm qcc get circuit shor-tuned-kingston-62qzb --qasm
```

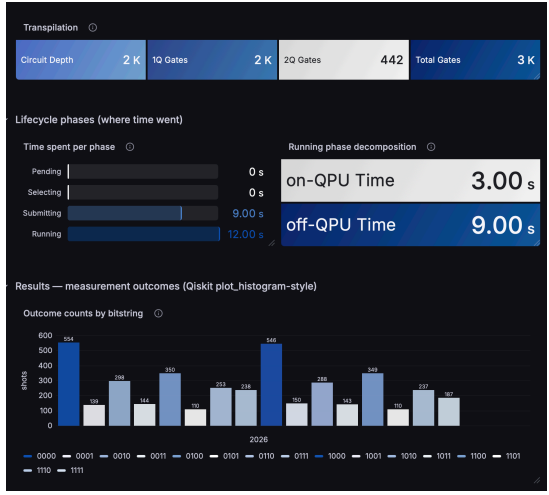
The full run summary, end to end

Run	Transpilation	Shots	`0000+1000`	Success
Aer (ideal)	default	1024	517+507	$\approx 100\%$
`ibm-fez` (v1)	default, L1	1024	56+55	10.8%
`ibm-marrakesh` (v1)	default, L1	1024	42+44	8.4%
`ibm-kingston` (v1)	default, L1	1024	41+41	8.0%
`ibm-kingston` (v2)	level 3 (Tier-1)	4096	540+530	26.1%
`ibm-kingston` (v3)	level 3 + `alap` (Tier-2)	4096	554+546	26.9%

The tuned runs on the Circuit dashboard



v2



v3

Discussion

Acceptance: all five requirements satisfied

Req.	Requirement	Evidence
R1	Declarative deployment on commodity hardware	One package brings up the controller, executor, CRDs, and observability on a local `kind` cluster, with 3 IBM Heron QPUs and 10 simulators registered.
R2	Observable through industry standards	Circuits and backends emit `qcc.*` over Open-Telemetry and Prometheus, shown on two Grafana dashboards with nothing custom in between.
R3	Portable and modular across providers	Two adapters (Aer, IBM) over one gRPC contract, controller free of vendor code. The unseen `alap` option rode through as a Tier-2 passthrough.
R4	Correlated across the boundary, in real time	`providerJobId` links a Circuit to its vendor job both ways, on `qcc_circuit_info` and a console deep-link, visible while the job runs.
R5	Backend quality as a measured signal	Coherence, gate and readout error, timing, and calibration exposed per backend. The real-hardware sweep showed a single number mis-ranks.

Limitations

- **Vendor coverage**: neutrality rests on the gRPC contract. Two adapters (Aer, IBM) exercise it; further providers such as Rigetti or AWS Braket are added as more adapters, not schema changes.
- **Input format**: Qiskit source runs in the executor's Python image, so it depends on that image's libraries. OpenQASM is self-contained, the portable submission format.
- **Workload**: one algorithm (Shor $N = 15$), submitted once. Iterative VQE is tightly-looped, unlike the asynchronous submission QCC targets.
- **Calibration freshness**: a backend's calibration is read once, at registration, and ages between probes. A periodic refresh would keep it current.
- **Submission boundary**: a short window sits between submission and recording the `providerJobId`. A controller restart there could leave a job unreconciled (none did across the evaluation).

Conclusions

Contributions

At the interface layer, not in algorithms or execution speed:

1. **Quantum Circuit Controller**: a cloud-native form of the orchestration layer in QCSC Reference Architecture.
2. **Observability primitives**: Circuits and QPUs `status` audit trail, the `qcc.*` metrics specification, and a cross-boundary identifier.
3. **A cross-substrate evaluation primitive**: `--performance-test`. Measurement, not prediction.
4. **Operator-facing dashboards**: the USE method (backend view) and RED method (per-circuit view), recast for quantum substrates.

Future work and standardisation

Extensions, each fitting an extension point the architecture already exposes:

- **Predictive selection**: calibration-aware scoring over the remaining stages of the selection chain, as a drop-in strategy implementation.
- **Calibration refresh**: a TTL-based re-probe in the QPU reconciler.
- **Distributed tracing**: emit spans at phase boundaries (the tracer is already instrumented).
- **Provider coverage**: generic Qiskit-provider, QRMI, and CUDA-Q adapters.
- **Iterative workloads**: a `Workflow` CRD composing Circuits (VQE, QAOA).
- **Production hardening**: mTLS, multi-tenant isolation, durable result storage.

Towards standards, candidates to outlive the thesis:

- **A Quantum Execution Interface (QEI)**: a standard execution interface for backends in the Kubernetes ecosystem, modelled on CRI. Vendors implement one plugin, any orchestrator consumes it.
- **A `quantum.*` OTel namespace**: a vendor-agnostic telemetry schema, shared across vendors and tools, proposed to the OpenTelemetry semantic-conventions registry.

Thank you!

Questions?

Ioannis Savvaidis

ioaiaaii.github.io