


gaqqie: an open-source quantum computer cloud platform

Satoyuki Tsukano

 @snuffkin

目次

1. 本プロジェクトが実現する、量子コンピュータのインフラ
2. 量子コンピュータ・クラウドサービスの現状
3. 開発内容
4. 設計・ノウハウ
5. API
6. 利用方法(サンプルコード)

本プロジェクトが実現する、
量子コンピュータのインフラ

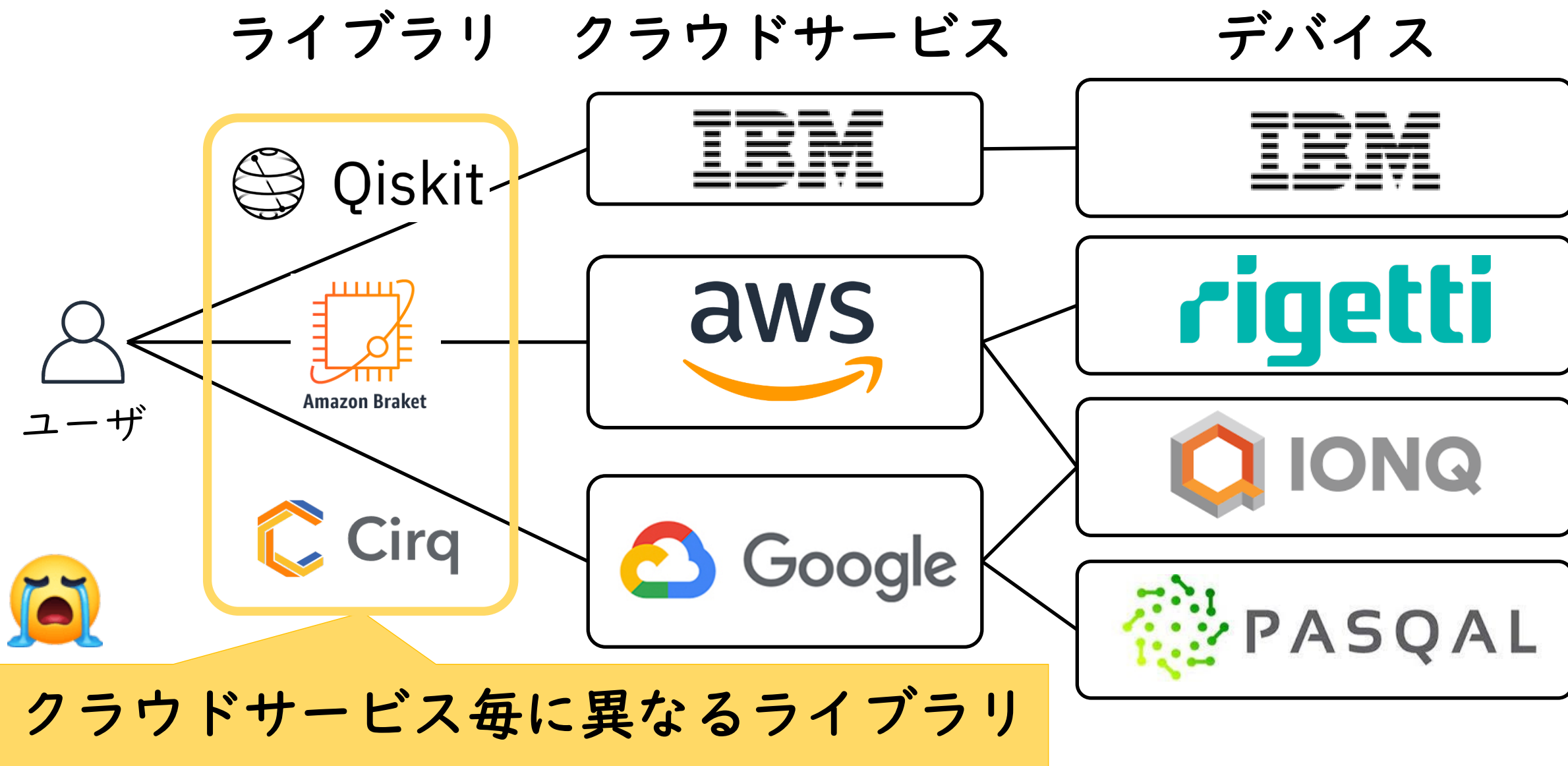
本プロジェクトが実現する、量子コンピュータのインフラ

1. 一般のユーザ
好きなライブラリで量子回路を実装し、
好きなデバイスで実行(将来的に)
2. 量子計算の研究室
さまざまなデバイスでの実行結果を一元管理
独自のデバイスも追加可能
3. 量子コンピュータの研究開発
クラウドサービスを少ない手間で公開し、低コストで運用可能

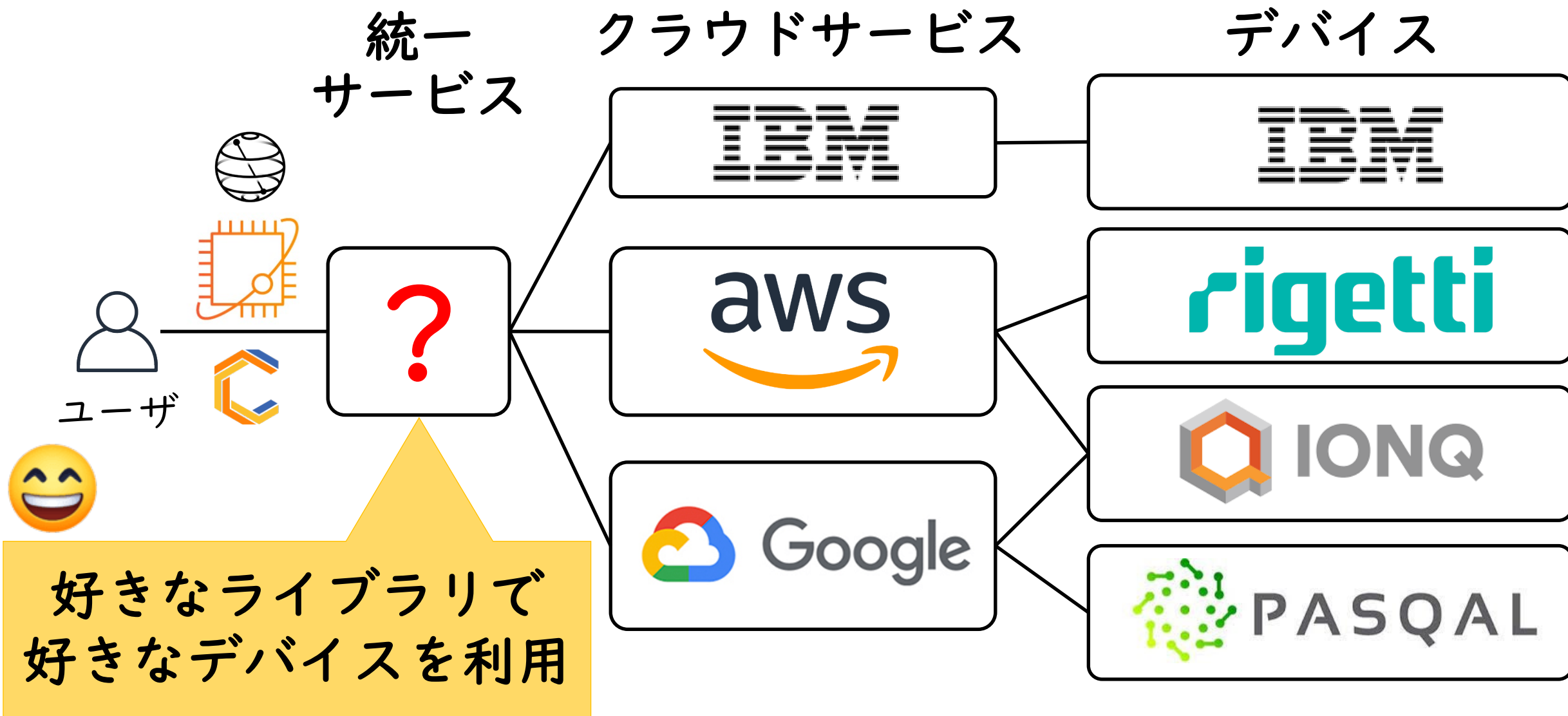
多様なユーザと、多様な量子コンピュータがつながり
量子コンピュータの研究や利用をスムーズにする

量子コンピュータ・クラウドサービスの 現状

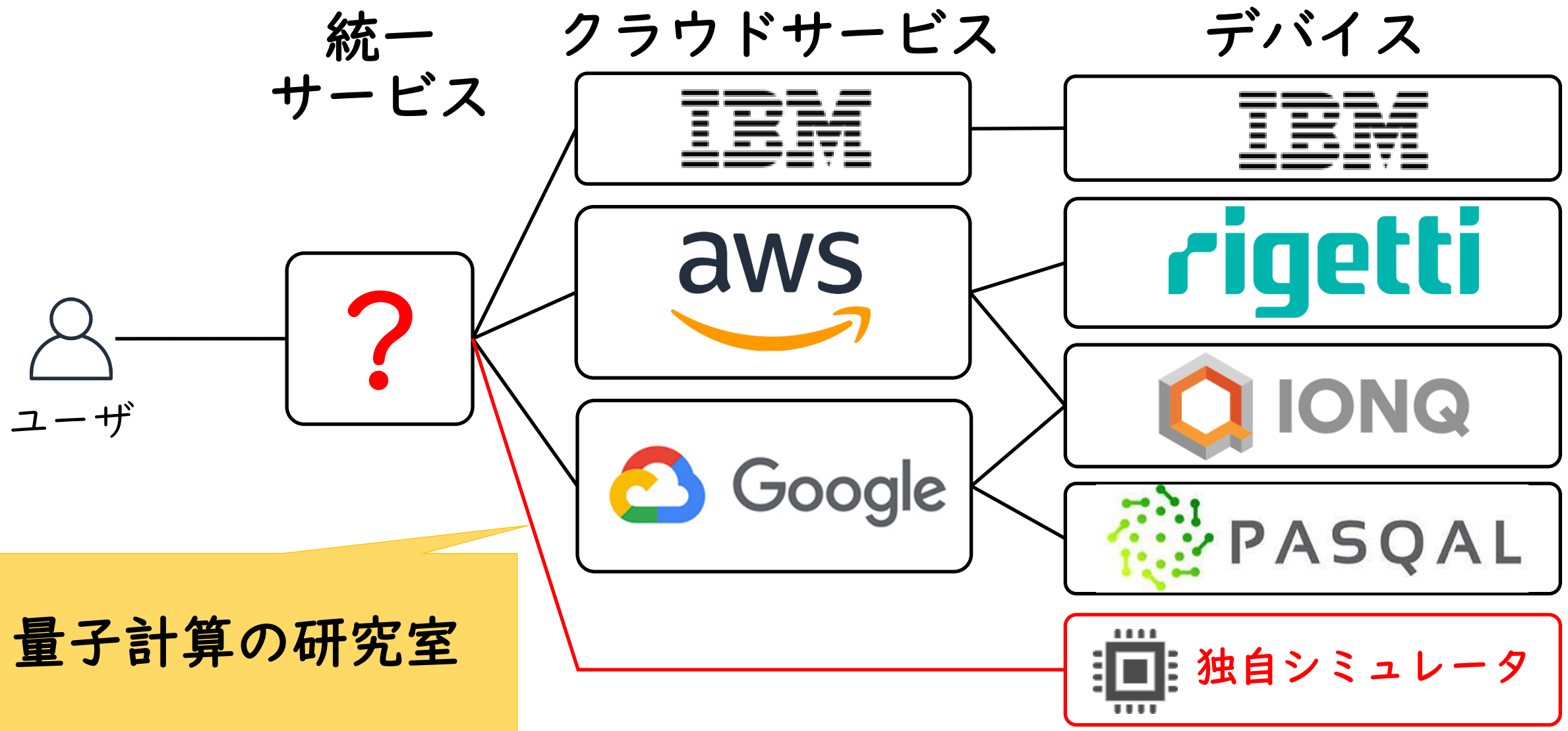
量子コンピュータ・クラウドサービスの現状



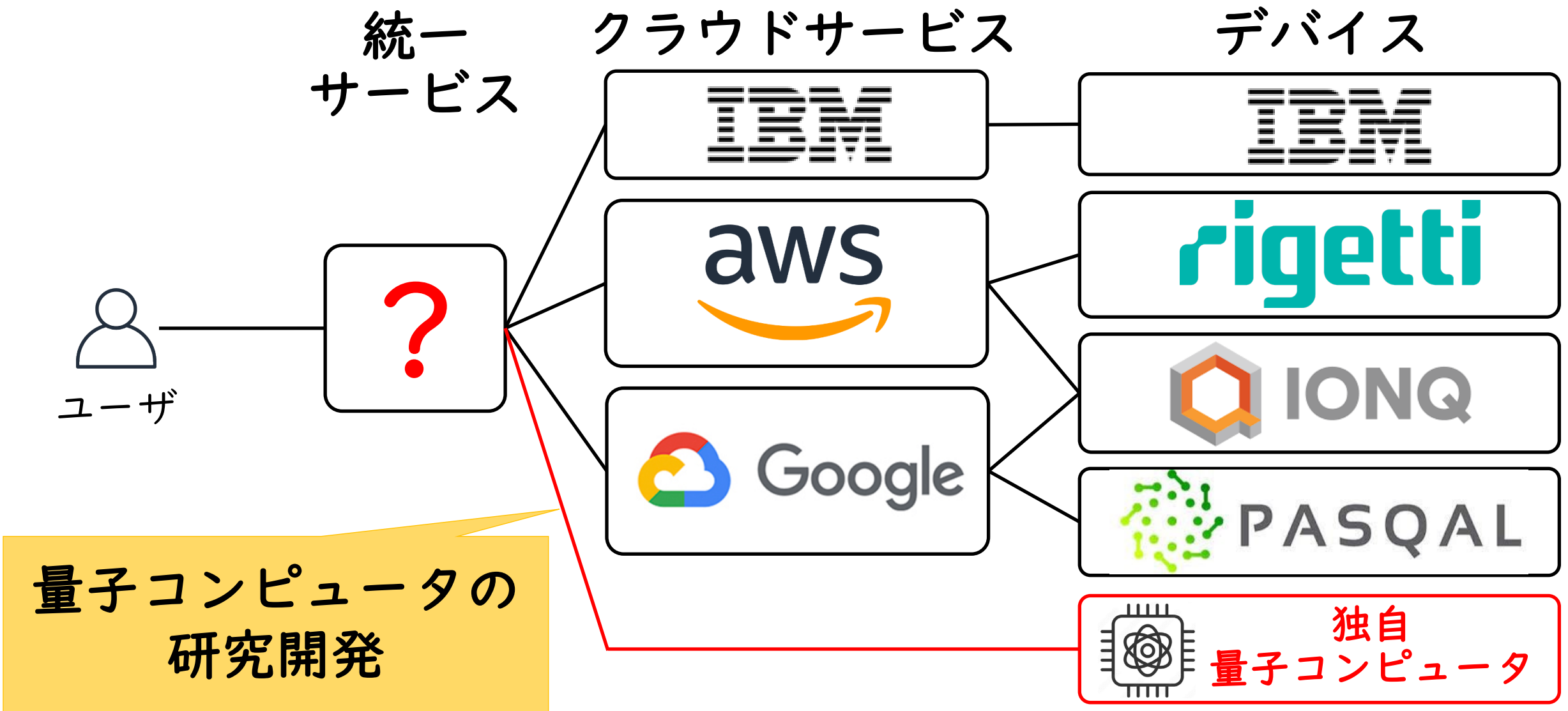
量子コンピュータ・クラウドサービスの現状



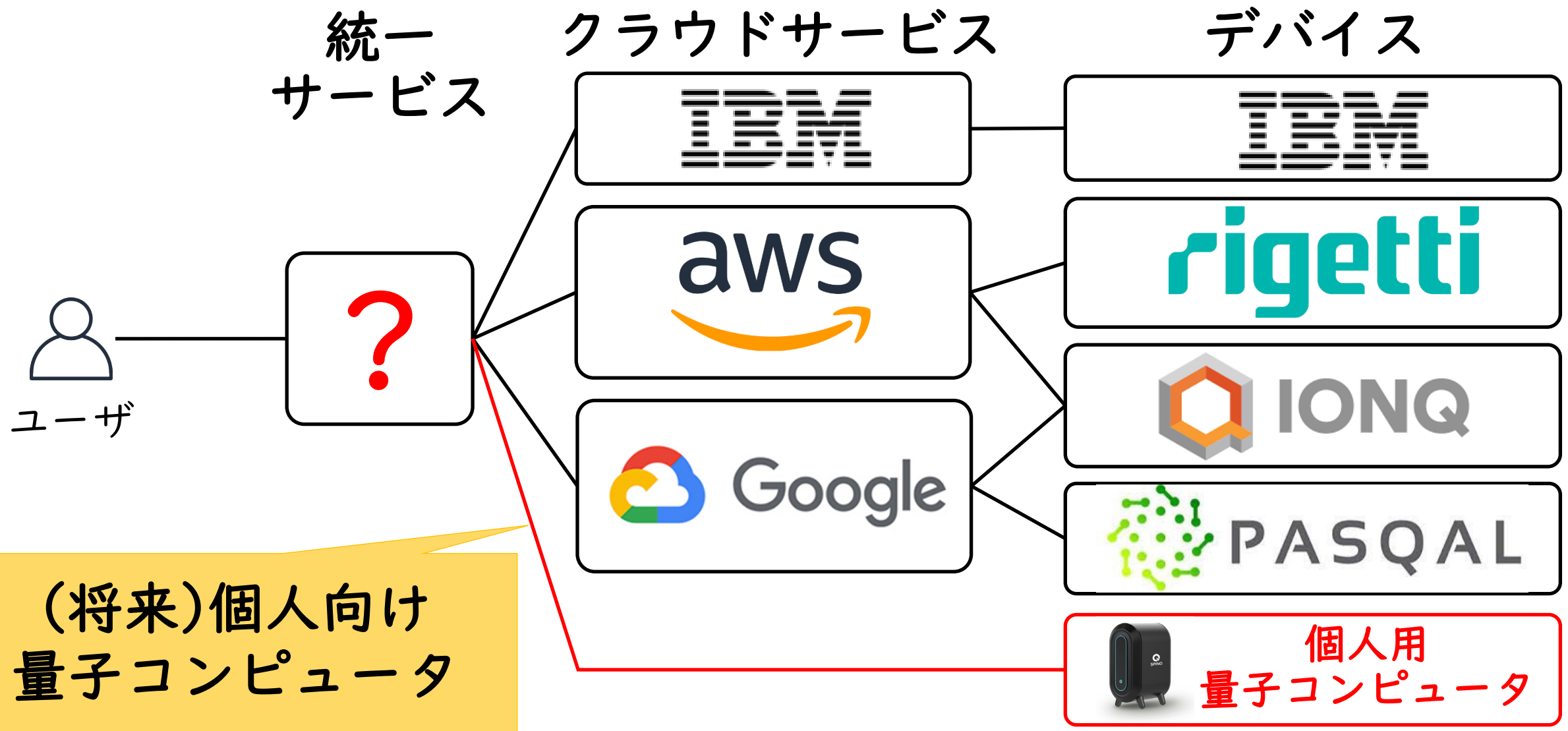
量子コンピュータ・クラウドサービスの現状



量子コンピュータ・クラウドサービスの現状

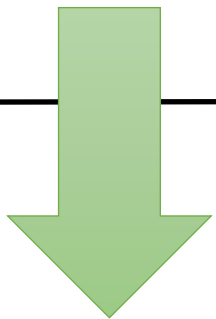
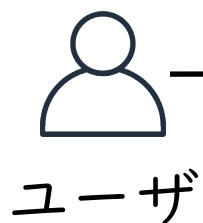


量子コンピュータ・クラウドサービスの現状

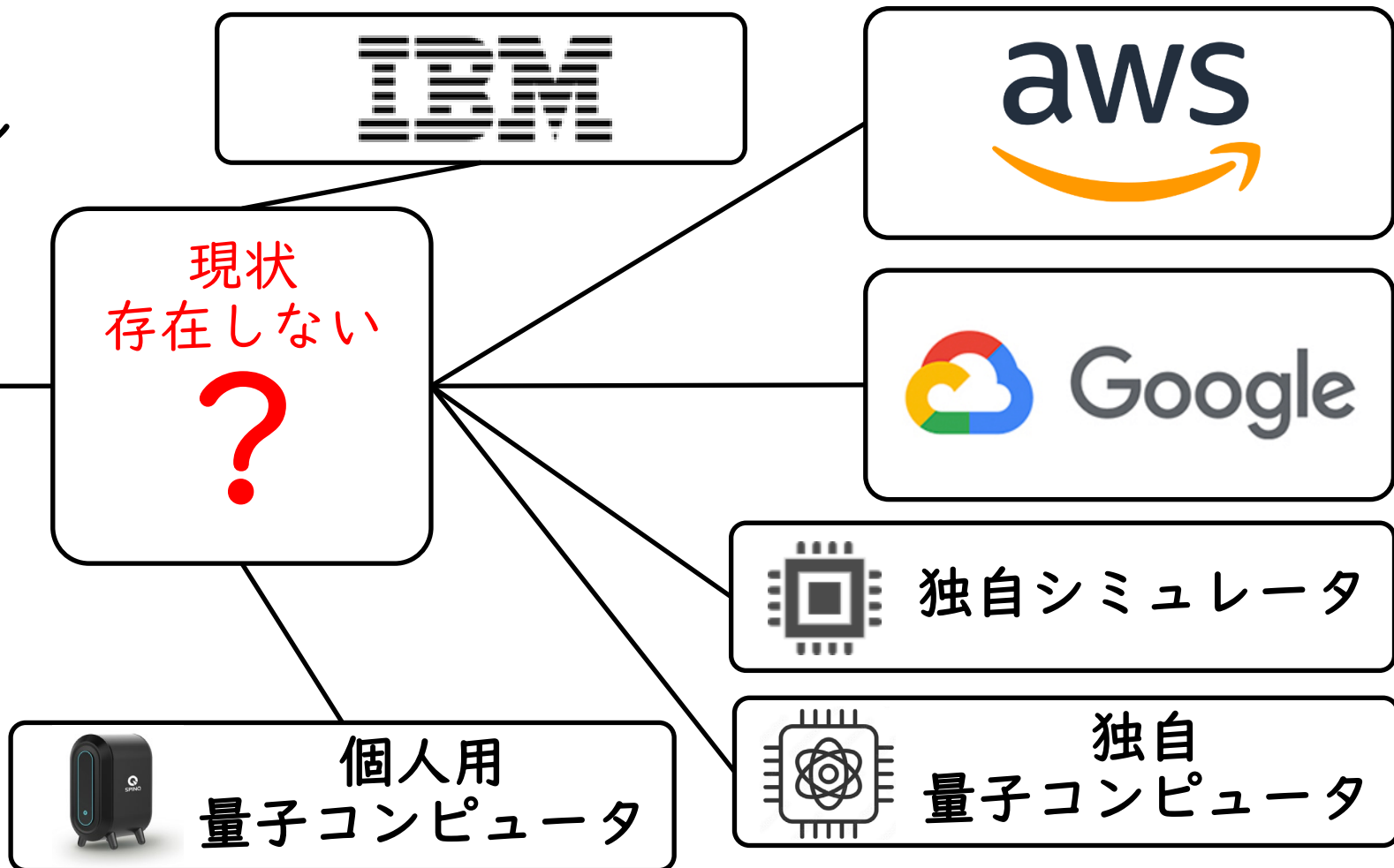


量子コンピュータ・クラウドサービスの現状

1. 耐障害性
2. 運用コスト
3. スケーラビリティ
4. 拡張性



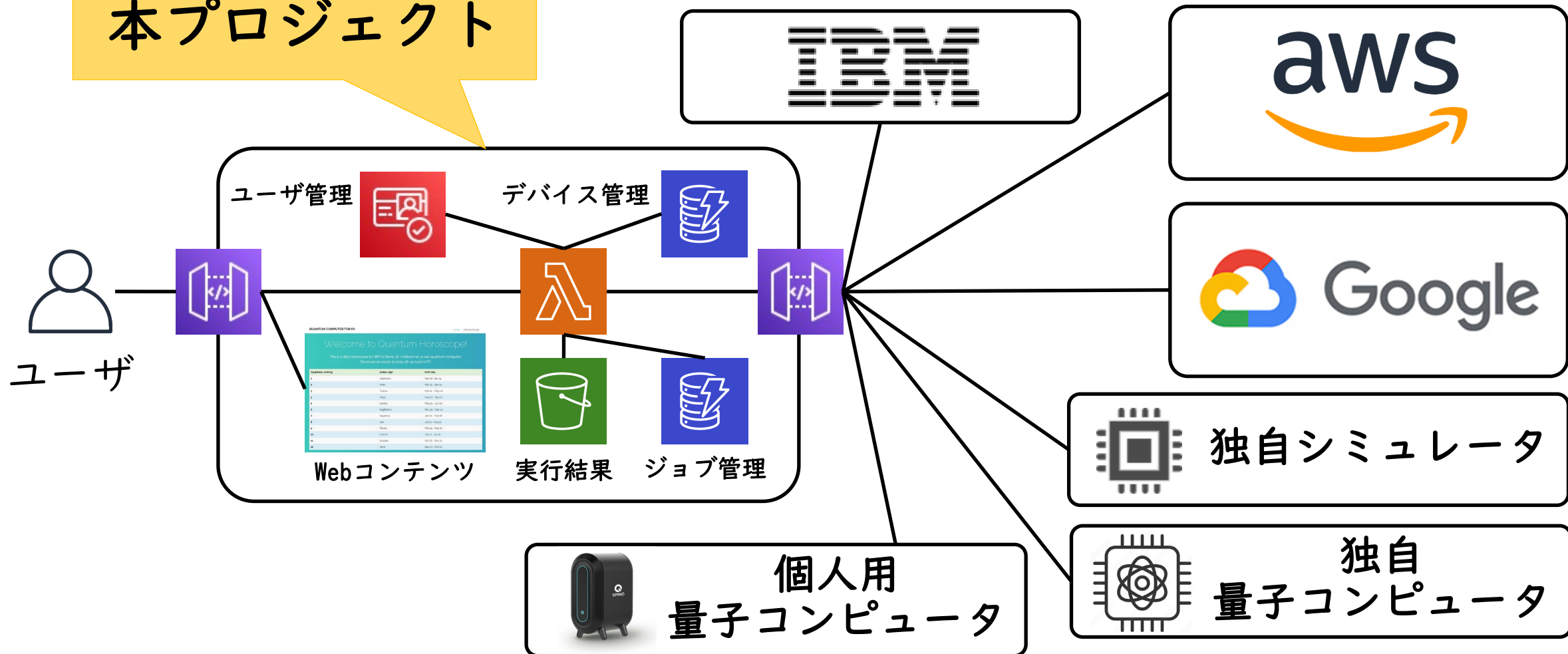
- ・開発コストかかる
- ・研究者は研究に集中したい



量子コンピュータ・クラウドサービスの現状

Gate-based Quick Quantum Infrastructure (gaqqie)

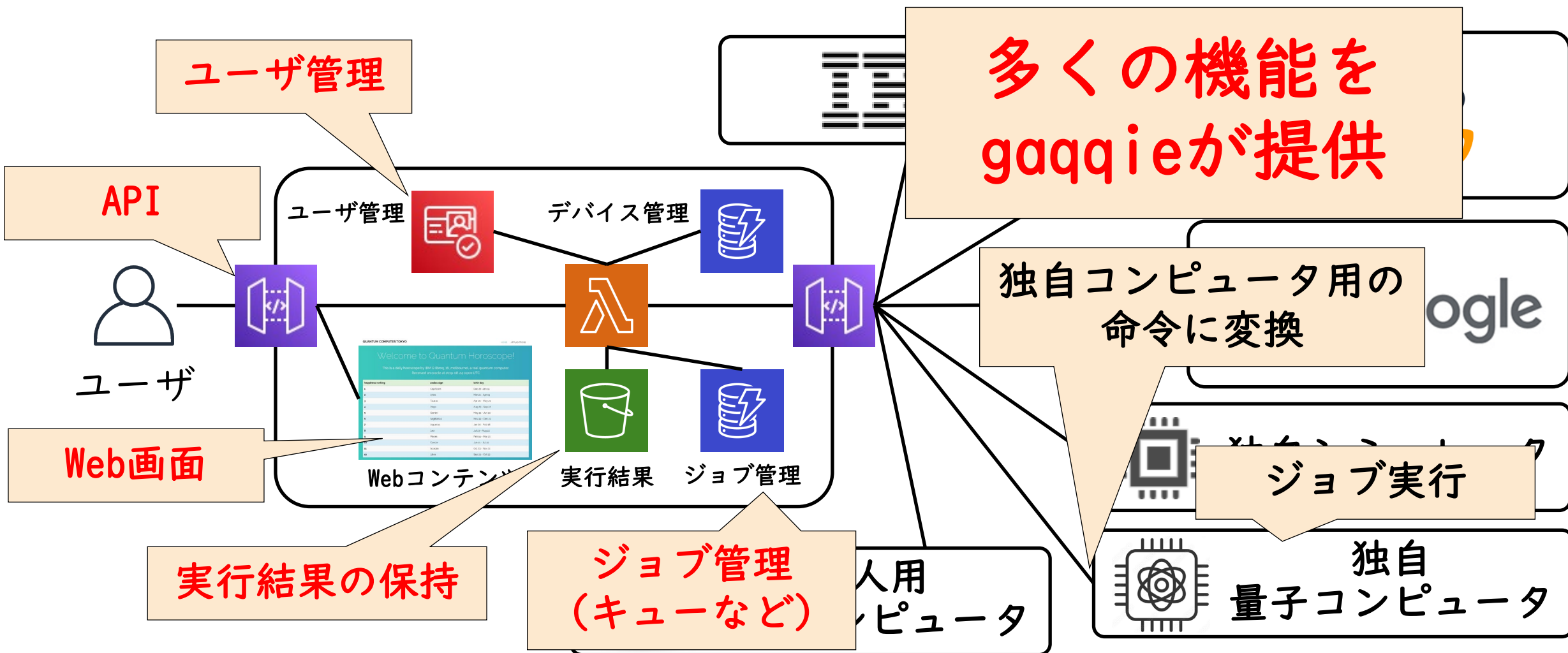
本プロジェクト



開発内容

開発内容

独自量子コンピュータをクラウド公開するときに必要な機能

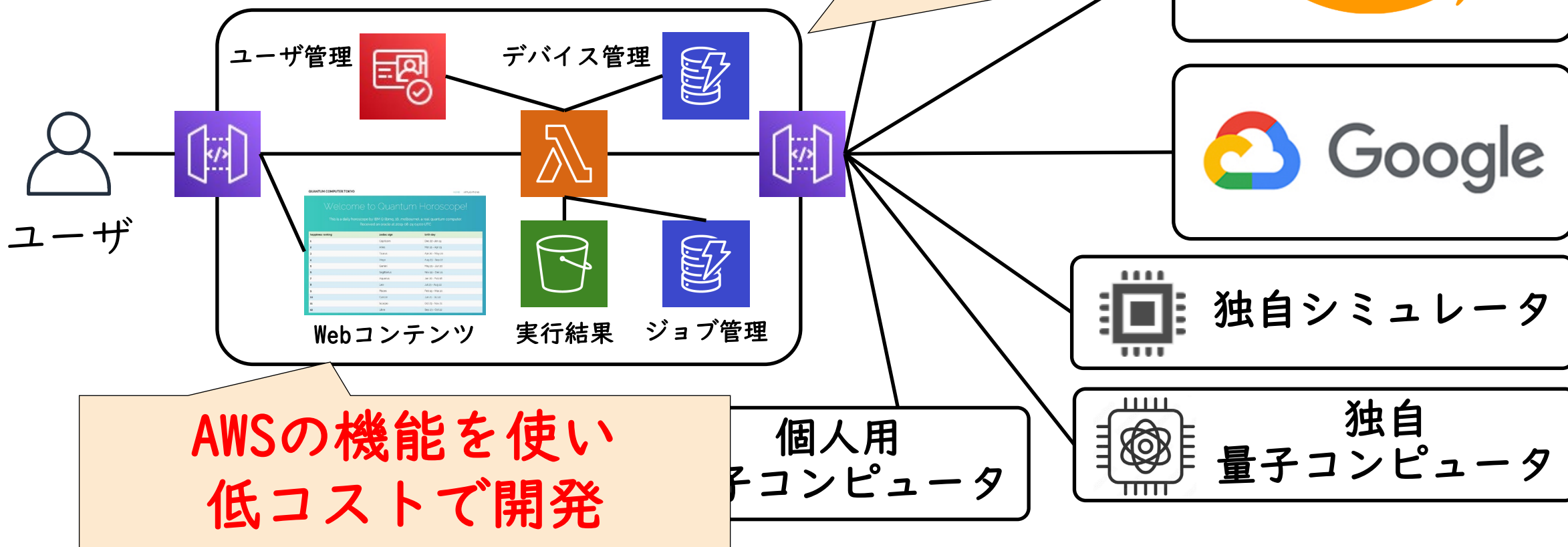


開発内容

よくある常駐サーバの課題

1. システム障害に備えた冗長化
2. ジョブが無いときもコスト発生
3. システム負荷に応じたサーバ増強

サーバレスな構成



開発内容

APIのフォーマット仕様

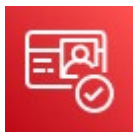
- ・ RESTful + JSON

→ Open APIで定義し、
Swagger CodegenでAPI周りのコードを生成



Webコンテンツ

ユーザ管理



デバイス管理



実行結果



ジョブ管理



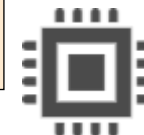
IBM

aws

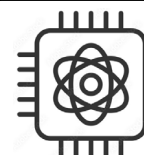


Google

独自のコンピュータや
シミュレータと
プル型で通信



独自シミュレータ



独自
量子コンピュータ



個人用
量子コンピュータ

開発言語

- ・ (Web)JavaScript、Vue2+Vuetify
- ・ (Web以外)Python

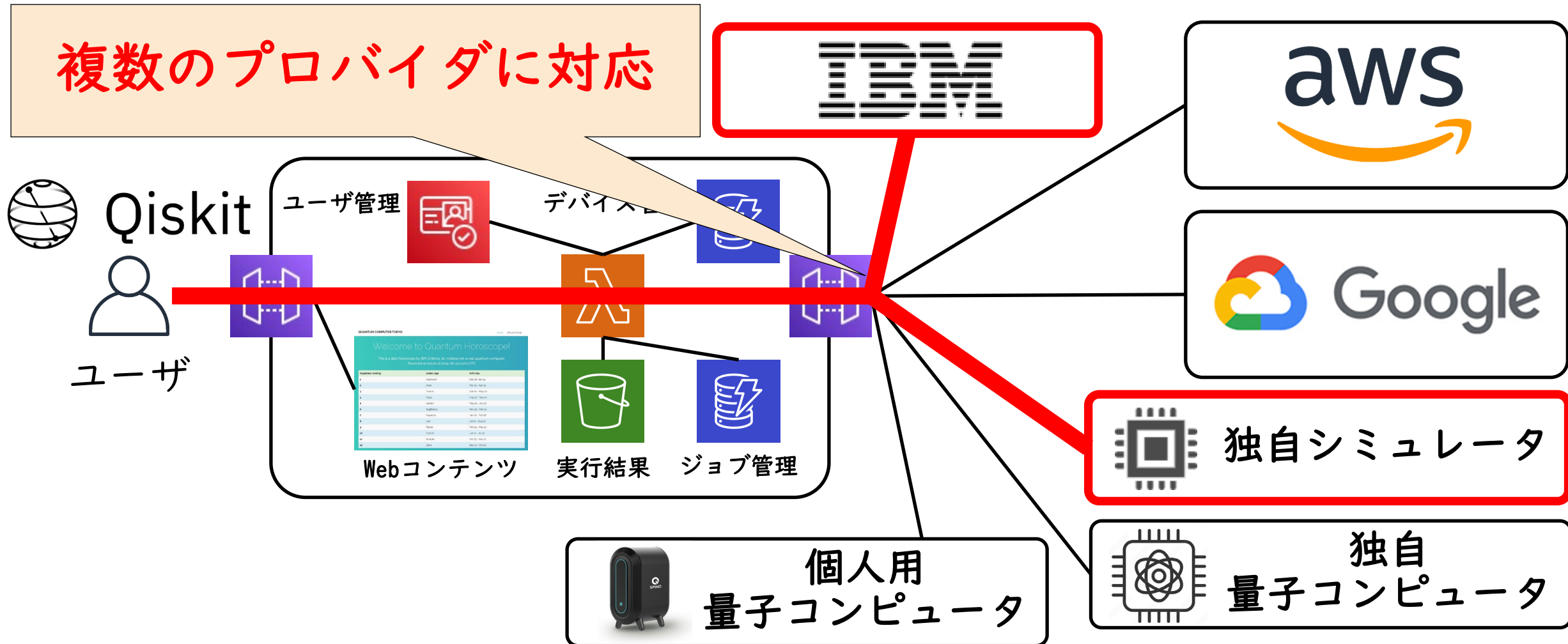
開発環境

- ・ Serverless Framework

開発内容

Gate-based Quick Quantum Infrastructure (gaqqie)

複数のプロバイダに対応



開発内容

ジョブ一覧画面

ジョブのステータスや
実行しているデバイスを表示

gaqqie - Quantum Computer Cloud Service

[Jobs](#)
[Devices](#)
[Providers](#)

実行待ちのジョブは
キャンセル可能

Jobs

詳しい実行結果を
ダウンロード

	Job ID	Status	Provider	Device	Create Time ↓ 1	End Time	Download
🚫	33ff1afc-a3b3-4959-85f9-22a312b1642d	QUEUED	gaqqie	qiskit_simulator	2022-02-08 20:47:20		
	b42b0abb-88d9-4842-899e-7ec268674d54	SUCCEEDED	gaqqie	qiskit_simulator	2022-02-08 14:06:03	2022-02-08 14:06:16	↓
	5d7d8016-aa54-4cd0-9d16-94ded1563fc9	SUCCEEDED	gaqqie	qiskit_simulator	2022-02-08 13:41:25	2022-02-08 13:41:37	↓
	ffe3f9f7-1d1d-4cb3-8e9a-00b5717955c7	SUCCEEDED	IBM	ibmq_quito	2022-02-08 07:22:12	2022-02-08 07:30:48	↓
	a042e97e-dbe4-48a4-817e-23b7f2963c64	SUCCEEDED	IBM	ibmq_quito	2022-02-08 04:09:29	2022-02-08 07:07:06	↓

Rows per page: All 1-5 of 5

(共通)
サイドメニューから
画面遷移

開発内容

デバイス一覧画面

The screenshot displays the 'gaqqie - Quantum Computer Cloud Service' interface. On the left, there are navigation links for 'Jobs', 'Devices', and 'Providers'. The main section is titled 'Devices' and contains a table with the following data:

Device name	Provider name	Status	Type	Qubits	Max shots	Description
qiskit_simulator	gaqqie	ACTIVE	simulator	10	8192	This is a simulator.
ibmq_montreal	IBM	ACTIVE	QPU	27	8192	This is a real machine.
ionQdevice	IonQ_AWS	ACTIVE	QPU	11	8192	This is a real machine.
Aspen-9	Rigetti_AWS	ACTIVE	QPU	32	8192	This is a real machine.

A red box highlights the 'Aspen-9' row. A yellow callout bubble points to this row with the text: 'デバイスの稼働状況やパラメータを表示' (Display device operating status and parameters).

Below the table, the 'Aspen-9' device details are shown. A yellow callout bubble points to the 'Aspen-9' header with the text: '一覧から選択' (Select from the list). The details include a 'Common properties' table:

Property	Value
Provider name	Rigetti_AWS
Status	ACTIVE
Device type	QPU
Qubits	27
Max shots	8192

Below this table is a descriptive paragraph: 'Rigetti quantum processors are universal, gate-model machines based on all-tunable superconducting qubits. Just like the Rigetti Aspen-8 chip, the Aspen-9 chip features tileable lattices of alternating fixed-frequency and tunable superconducting qubits within a scalable 32-qubit node technology. Distinguishing characteristics include direct coupling between one qubit and its three nearest neighbors; fast gate times for multiple entangling gate families; rapid sampling via active register reset; and parametric control.'

A yellow callout bubble points to the detailed information section with the text: 'デバイスの詳細情報' (Device detailed information).

開発内容

プロバイダー一覧(IBM、AWS、独自シミュレータ等)

The screenshot displays the 'gaqqie - Quantum Computer Cloud Service' interface. On the left, a sidebar contains links for 'Jobs', 'Devices', and 'Providers'. The main area is titled 'Providers' and contains a table with the following data:

Provider name	Status	Description
gaqqie	ACTIVE	a simulator on Qiskit.
IBM	ACTIVE	IBM provides superconducting qubits machines.
IonQ_AWS	ACTIVE	IonQ provides trapped ion qubits machines.
Rigetti_AWS	ACTIVE	Rigetti provides superconducting qubits machines.

An orange arrow points from the 'IBM' row in the table to the 'IBM' provider details section below. The details section includes:

- Common properties:** Status: ACTIVE. Description: IBM quantum processors are universal, gate-model machines based on superconducting qubits.
- Provider specific properties:** Requirements: need IBM Q account.
- Devices:** A table listing specific devices:

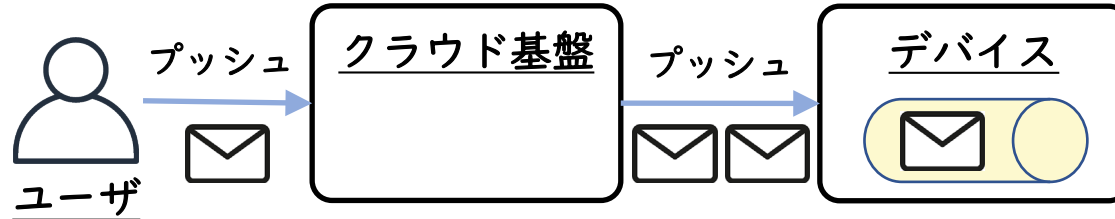
Device name	Status	Type	Qubits	Max shots	Description
ibmq_montreal	ACTIVE	QPU	27	8192	This is a real machine.
ibmq_manhattan	ACTIVE	QPU	65	8192	This is a real machine.
ibmq_santiago	ACTIVE	QPU	5	8192	This is a real machine.

Three callout boxes provide additional context:

- 一覧から選択** (Select from the list): Points to the 'IBM' row in the Providers table.
- プロバイダの詳細情報** (Provider details): Points to the 'IBM' provider details section.
- プロバイダの稼働状況やパラメータを表示** (Display provider status and parameters): Points to the Providers table.

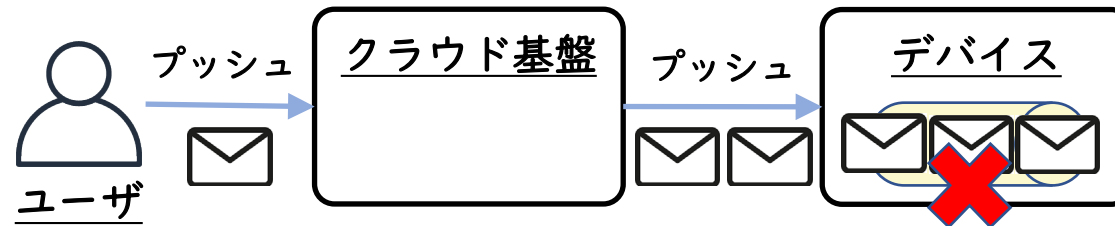
設計・ノウハウ

デバイス側とプル型で通信



量子コンピュータのシステムにおいて、
処理時間がかかるのはデバイス側

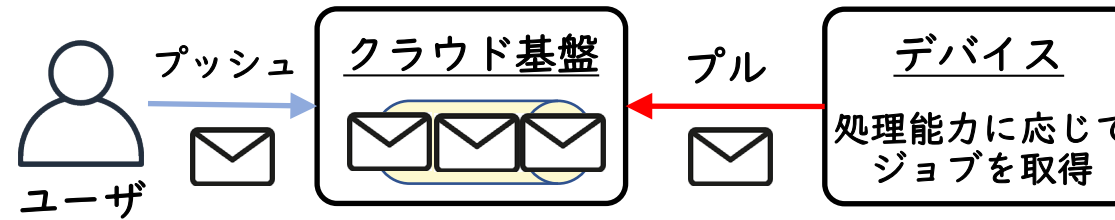
ジョブが増え続けると。。。↓



デバイス側に負荷がかかると
システムが不安定になりやすい

対策

プル型でデバイス側と通信可能
(そのためのライブラリも提供)



デバイス側で負荷をコントロール

副次効果

- ・デバイス側インターネット非公開にでき、
第三者からの攻撃を防げる
- ・プロバイダはキューを構築する必要がない

※IBMやAmazon Braket等の既存サービスはプルしてくれないので、そちらはプッシュ型で通信

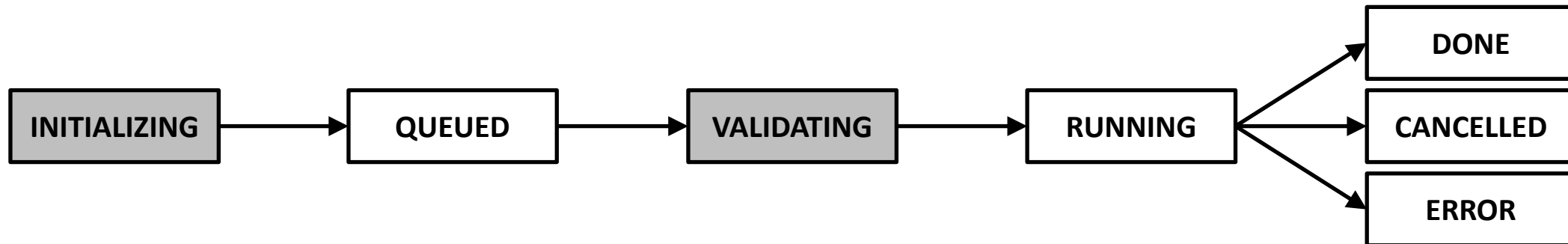
ジョブ状態遷移



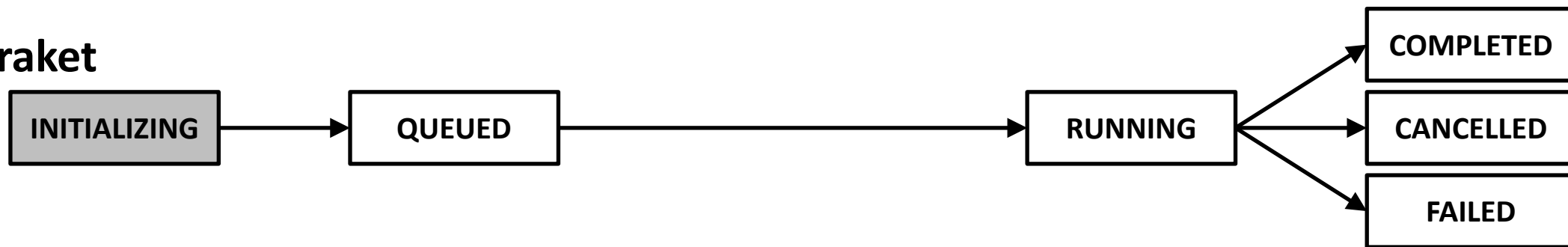
基本的には外から見えない状態

実際には途中の状態から「CANCELLED」
「FAILED」に遷移する可能性あり

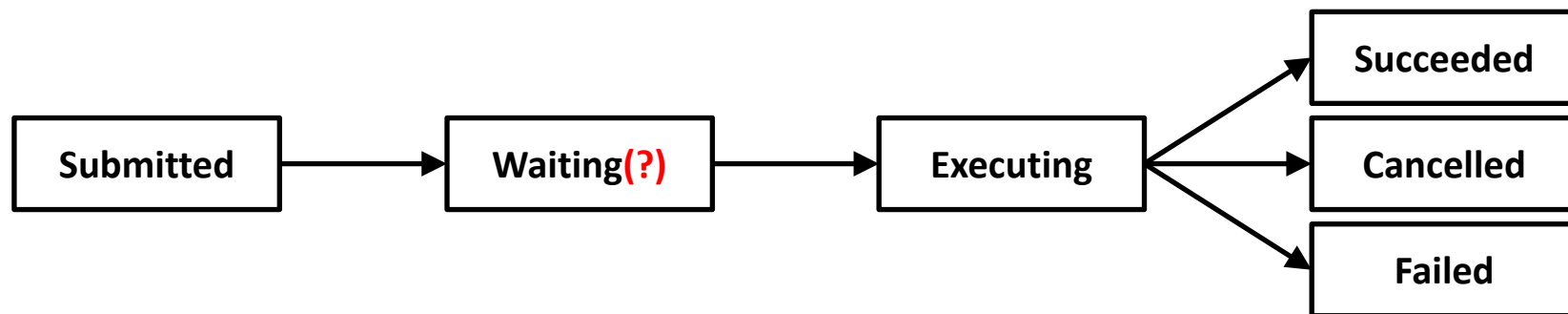
IBM Q



Amazon Braket



Azure Quantum



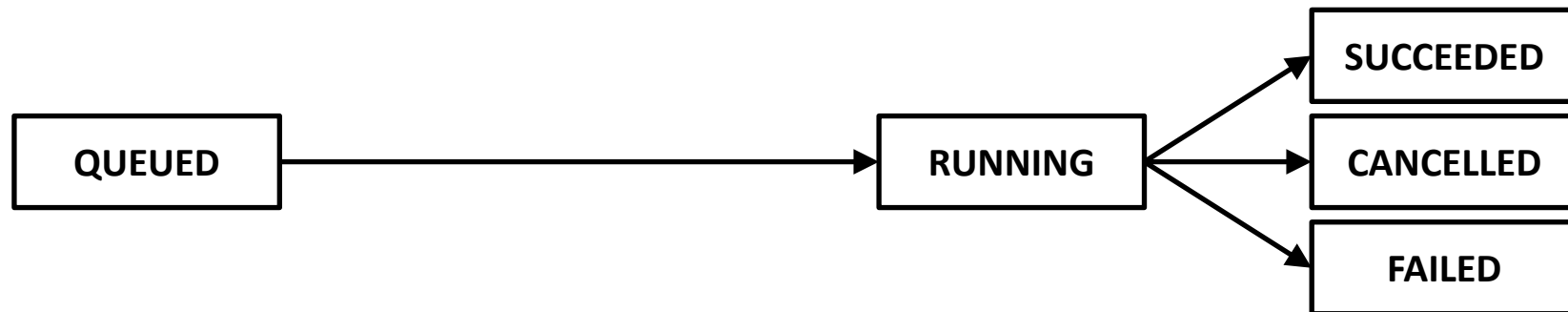
ジョブ状態遷移



基本的には外から見えない状態

実際には途中の状態から「CANCELLED」
「FAILED」に遷移する可能性あり

gaqqie



- ・ ユーザの立場で考えると、INITIALIZING、VALIDATING、Waitingは気にならいため、gaqqieでは採用しない。
- ・ 最終状態が成功か失敗か分かりやすいように、「SUCCEEDED」「FAILED」の名称にした。

データスキーマ(ジョブの概要情報)

データベースに格納しておき、一覧検索できる
具体的な回路の内容は、概要情報には含めない

項目名(和名)	項目名(変数名)	型	説明
ジョブ識別子	id	string	ジョブを一意に特定する識別子。
ジョブ名	name	string	ジョブの名称。(イテレーション2で実装)
状態	status	string	ジョブの状態を表す文字列。詳細は「ジョブ状態遷移」を参照。
プロバイダ名	provider_name	string	プロバイダの名称。
デバイス名	device_name	string	デバイスの名称。
作成時刻	create_time	string	クライアントがクラウドにジョブ実行を命令した時刻。正確には、gaqqie-skyが受信した際の時刻。
完了時刻	end_time	string	ジョブ実行が完了した時刻。正確には、gaqqie-skyがプロバイダから実行結果を受信した時刻。

- ・時刻の型について

JSONには時間の型が存在しないため、JSONではstringとして扱う。

プログラム内部では時刻の型として扱う。

時刻をstringで扱う際のフォーマットは次のフォーマットで扱う。(ISO形式、ミリ秒単位、タイムゾーンはUTC)。

2021-06-01T01:02:03.456Z

- ・この表では、システムの内部的な項目は記載を省略している

データスキーマ(デバイスの概要情報)

データベースに格納しておき、一覧検索できる
精度・トポロジ等は、概要情報には含めない

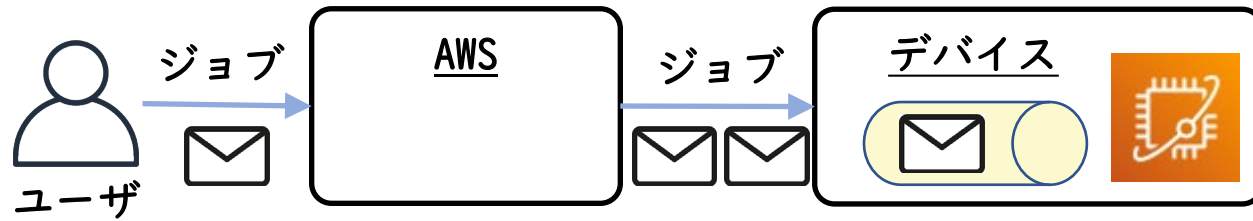
項目名(和名)	項目名(変数名)	型	説明
デバイス名	name	string	デバイスの名称。一意に特定する識別子としても利用する。
プロバイダ名	provider_name	string	プロバイダの名称。
状態	status	string	デバイスの状態を表す文字列。 <ul style="list-style-type: none">• ACTIVE: 新規ジョブを登録でき、登録されたジョブが順次実行される状態。• SUBMITTABLE: 新規ジョブを登録できるが、ジョブの実行が停止している状態。デバイスの一時的なメンテナンス時等に用いる。• UNSUBMITTABLE: 新規ジョブを登録できない状態。長時間に渡るデバイスの停止時などに用いる。
デバイスの状態は 3種類 →理由は次頁参照			
説明	description	string	デバイスの説明文。画面から参照できる。
デバイスタイプ	device_type	string	デバイスの状態を表す文字列。 <ul style="list-style-type: none">• QPU: 量子コンピュータの実機• simulator: シミュレータ
量子ビット数	num_qubits	int	デバイスの量子ビット数。
最大ショット数	max_shots	int	実行可能な最大のショット数。

- ・ジョブがデバイスに届く前に、量子ビット数、最大ショット数をクラウド側でチェック可能
- ・デバイスの状態がUNSUBMITTABLEの場合は、ジョブをデバイス向けのキューに登録しない

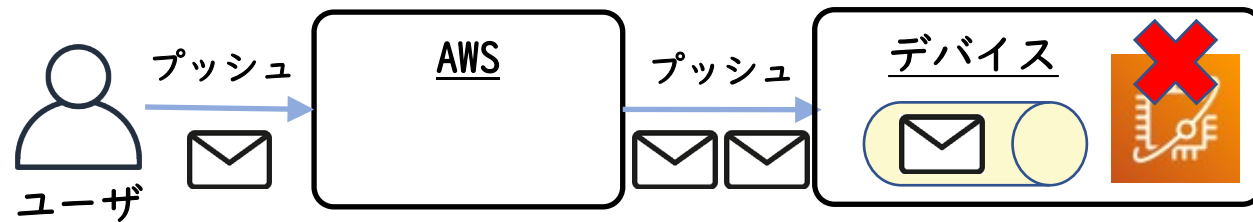
デバイス状態の種類(Amazon Braketの場合)

Amazon Braket

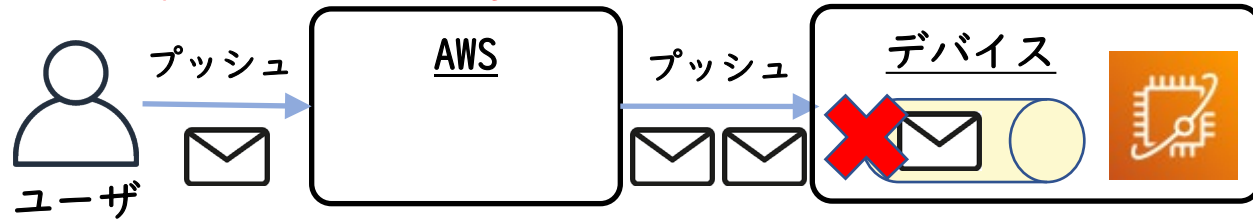
デバイス稼働中。ジョブ受付可



デバイス停止。ジョブ受付可



デバイス停止。ジョブ受付不可



デバイスがメンテナンス中などで停止している場合でも、ジョブを受け付けている状態であれば、**ONLINE**となる。

課題

「今、デバイスが稼働しているか」がユーザには分からない。

→デバイス状態が、**ONLINE** / **OFFLINE**の2値しかないため。

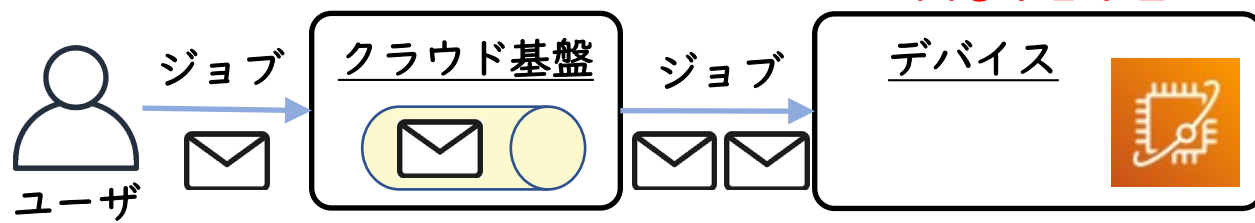
ジョブを受け付けていない状態は、**OFFLINE**となる。

デバイス状態の種類(gaqqieでの対策)

gaqqie 対策 デバイス状態を3種類にする

デバイス稼働中。ジョブ受付可

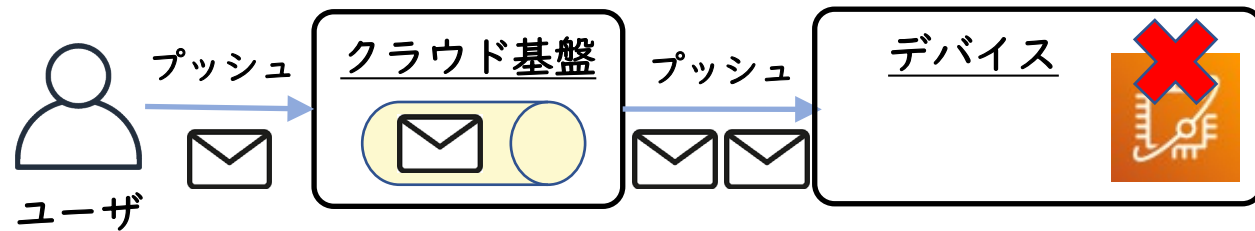
ACTIVE



「今、デバイスが稼働しているか」がユーザに分かるようにする。
デバイスが稼働していれば、**ACTIVE**となる。

デバイス停止。ジョブ受付可

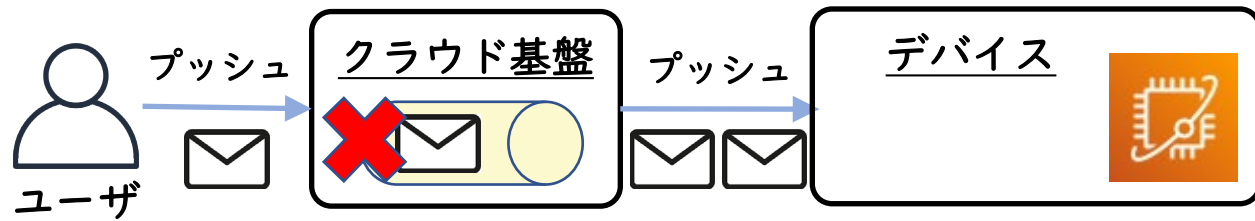
SUBMITTABLE



デバイスが稼働していなくても、ジョブを受け付けていけば、**SUBMITTABLE**となる。

デバイス停止。ジョブ受付不可

UNSUBMITTABLE



ジョブを受け付けていない状態は、**UNSUBMITTABLE**となる。

データスキーマ(プロバイダの概要情報)

データベースに格納しておき、
一覧検索できる

項目名(和名)	項目名(変数名)	型	説明
プロバイダ名	name	string	プロバイダの名称。一意に特定する識別子としても利用する。
状態	status	string	プロバイダの状態を表す文字列。 <ul style="list-style-type: none">• ACTIVE: プロバイダを利用できる状態。• INACTIVE: プロバイダを利用できない状態。これは表示データとして用いる。実際にデバイスを利用できないようにするには、デバイスの状態を個別に更新すること。
説明	description	string	プロバイダの説明文。画面から参照できる。



API

共通事項

各APIは次のフォーマットのURLとする

`https://<API GatewayのID>.execute-api.<リージョン>.amazonaws.com/<ステージ>/<バージョン>/<パス>`

項目名	説明
<API GatewayのID>	gaqqie-skyをデプロイしたAPI GatewayのID。ユーザAPIとプロバイダAPIで異なる。
<リージョン>	gaqqie-skyをデプロイしたAWSのリージョン。例: ap-northeast-1
<ステージ>	gaqqie-skyをデプロイしたステージ。例:dev, prd
<バージョン>	APIのバージョン。例:v1
<パス>	個別のAPIで規定するURLのパス。

個別APIはRESTful形式する

詳細な情報はJSON形式でHTTPのボディに格納する

ユーザAPI一覧

API名	パス	メソッド	説明
ジョブ実行	jobs	post	量子回路のジョブを実行する
ジョブ一覧	jobs	get	ジョブ一覧を取得する
ジョブ取得	jobs/{id}	get	ジョブIDを指定してジョブ情報を取得する
ジョブキャンセル	jobs/{id}/cancel	get	ジョブIDを指定してジョブをキャンセルする。ジョブのステータスが「QUEUED」の場合のみキャンセル可能
実行結果取得	results/{job_id}	get	ジョブの実行結果を取得する
デバイス一覧	devices	get	デバイス一覧を取得する
デバイス取得	devices/{name}	get	デバイス名を指定してデバイス情報を取得する
デバイス取得 (プロバイダ指定)	devices/provider/{provider_name}	get	プロバイダ名を指定してデバイス情報を指定する
デバイス画像取得	devices/{name}/image	get	デバイス名を指定してデバイス画像を取得する
プロバイダ一覧	providers	get	プロバイダ一覧を取得する
プロバイダ検索	providers/{name}	get	プロバイダ名を指定してプロバイダ情報を取得する

プロバイダAPI一覧

API名	パス	メソッド	説明
ジョブ取得	jobs/{device_name}	get	デバイス名を指定して、該当デバイスで実行するジョブを取得する
実行結果登録	results/{job_id}	post	ジョブの実行結果をgaqqie-skyに登録する
デバイス更新	devices/{name}	post	デバイス名を指定して、デバイス情報を更新する
プロバイダ更新	providers/{name}	post	プロバイダ名を指定して、プロバイダ情報を更新する

利用方法(サンプルコード)

利用方法 ユーザ側の実装サンプル(Qiskit)

```
from qiskit import QuantumCircuit, execute
from gaqqie_door import QiskitGaqqie
```

ライブラリをインポート

```
circuit = QuantumCircuit(2, 2)
circuit.h(0)
circuit.cx(0, 1)
circuit.measure([0, 1], [0, 1])
```

Qiskitの経験者にとって
自然な書き方

```
url = "https://<api-id>.execute-api.<region>.amazonaws.com/<stage>"
QiskitGaqqie.enable_account(url)
backend = QiskitGaqqie.get_backend("qiskit_simulator")
```

デバイスを指定

```
job = execute(circuit, backend)
result = job.result()
print(f"result job_id={job.job_id()}, counts={result.get_counts()}")
```

利用方法 デバイス側の実装サンプル(Qiskitを使ったシミュレータ)

```
url = "https://<api-id>.execute-api.<region>.amazonaws.com/<stage>"
app = Gaqqie(url)

@app.receive_job(device_name="qiskit_simulator", interval=10)
def receive_job(job):
    # parse circuit
    ...

    # execute circuit
    ...
    result = aer_job.result()
    print(f"result      job_id={job_id}, counts={result.get_counts()}")
    result_dict = result.to_dict()
    result_dict["backend_name"] = job.device_name
    result_json = json.dumps(result_dict, indent=2)

    # register result
    job_result = Result(job_id=job_id, status="SUCCEEDED", results=result_json)
    response = app.register_result(job_result)

app.join()
```

ジョブを受信時に呼び出される関数に
デコレータ(@app. ~)を付与

実行結果をクラウド基盤に登録