

---

**GraphQL APIを迅速に構築できる  
OSS「Hasura」を触ってみた**

**2022/03/11**  
**株式会社 日立製作所**  
**加藤優佑**

## Contents

---

1. GraphQLとHasuraの概要
2. Hasuraの主要機能
3. Hasuraの認証・認可機能
4. Keycloakを使用したHasuraの認可・認証環境の構築
5. まとめ

# Contents

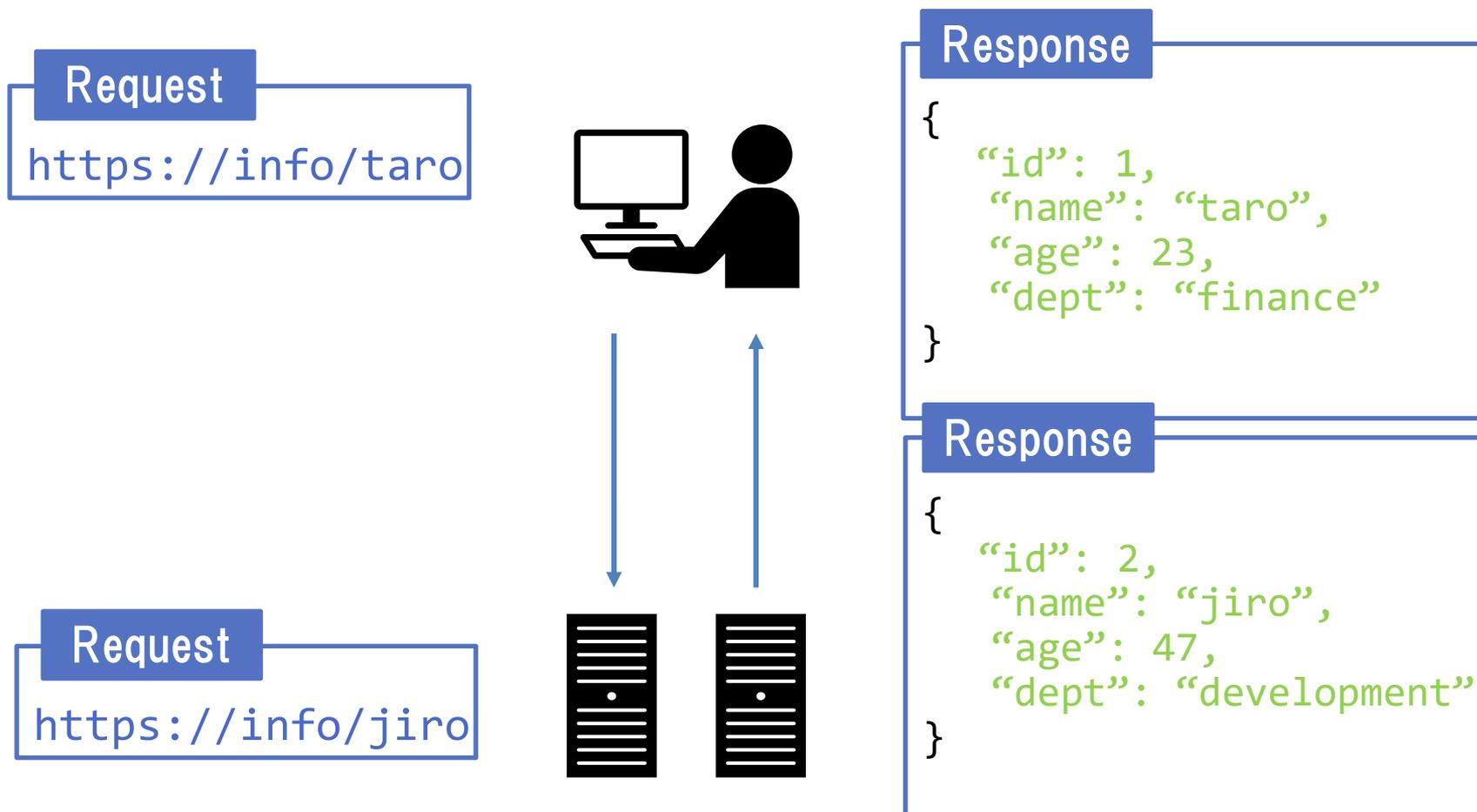
---

- 1. GraphQLとHasuraの概要**
2. Hasuraの主要機能
3. Hasuraの認証・認可機能
4. Keycloakを使用したHasuraの認可・認証環境の構築
5. まとめ

# 1-1. RESTful APIの問題点

従来のRESTful APIでは特定のURIにアクセスすれば、そこに紐づいた情報を取得できるが以下の問題点があった

- レスポンスされるデータは固定のため、不要なデータまで返ってくる
- 必要なデータが別のURIにある場合、何度もAPIを叩く必要がある

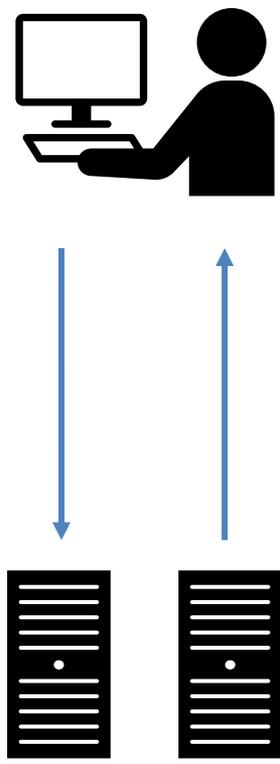


# 1-2. GraphQLとは

GraphQLはAPI向けのクエリ言語

- 取得したいデータをクエリに書くと、そのデータがレスポンスされる
- 1回のリクエストで必要なデータを取得可能
- データモデルの設計も行うため、RESTful APIに比べて開発コストが高い

```
Request
query {
  User {
    id
    name
    age
  }
}
```

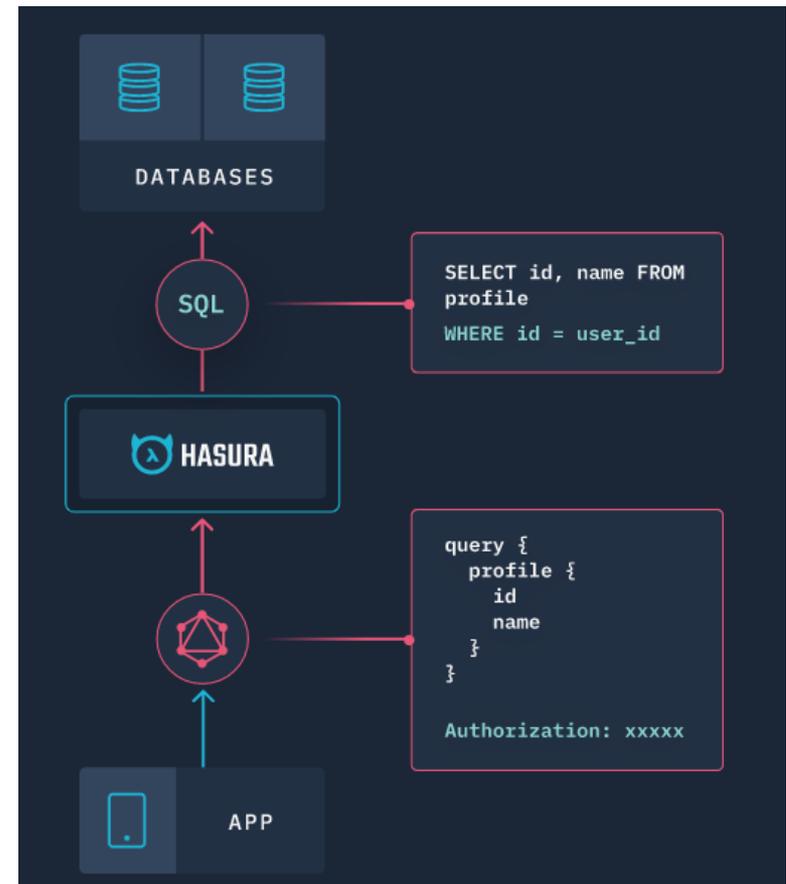


```
Response
{
  "data": {
    "User": [
      {
        "id": 1,
        "name": "taro",
        "age": 23,
      },
      {
        "id": 2,
        "name": "jiro",
        "age": 47,
      }
    ]
  }
}
```

# 1-3. Hasuraとは

HasuraはGraphQLのバックエンドサービス

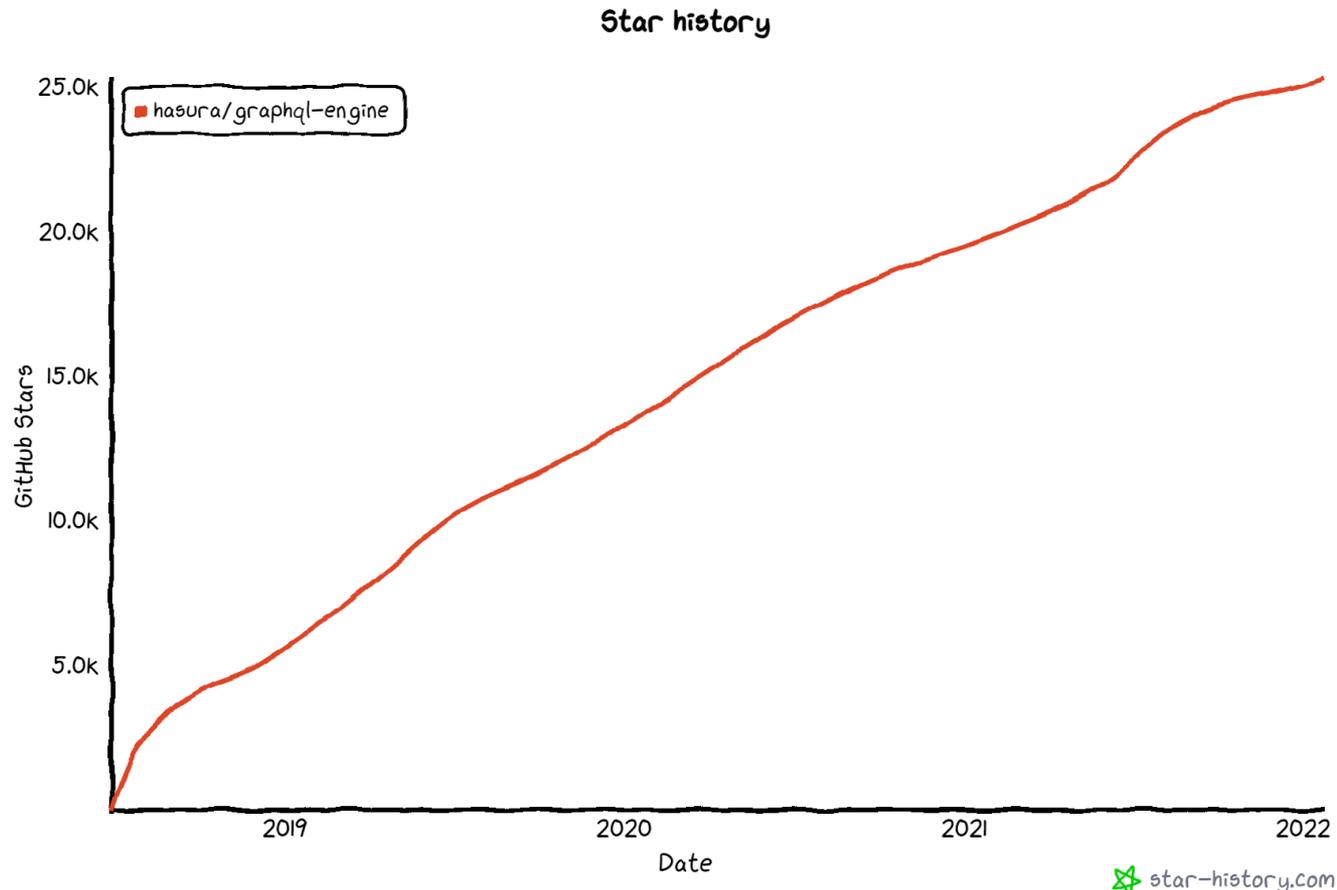
- クライアントとDBの間に入り、GraphQLとSQLを相互変換する
- DBへアクセスするGraphQL APIの開発時間を短縮可能



(<https://hasura.io>)

# 1-3. Hasuraとは

- ライセンスはApache License 2.0であり、バージョン2.2.2までリリース
- Githubのスター数推移からみても注目度の高いOSSである



(<https://star-history.com/#hasura/graphql-engine>)

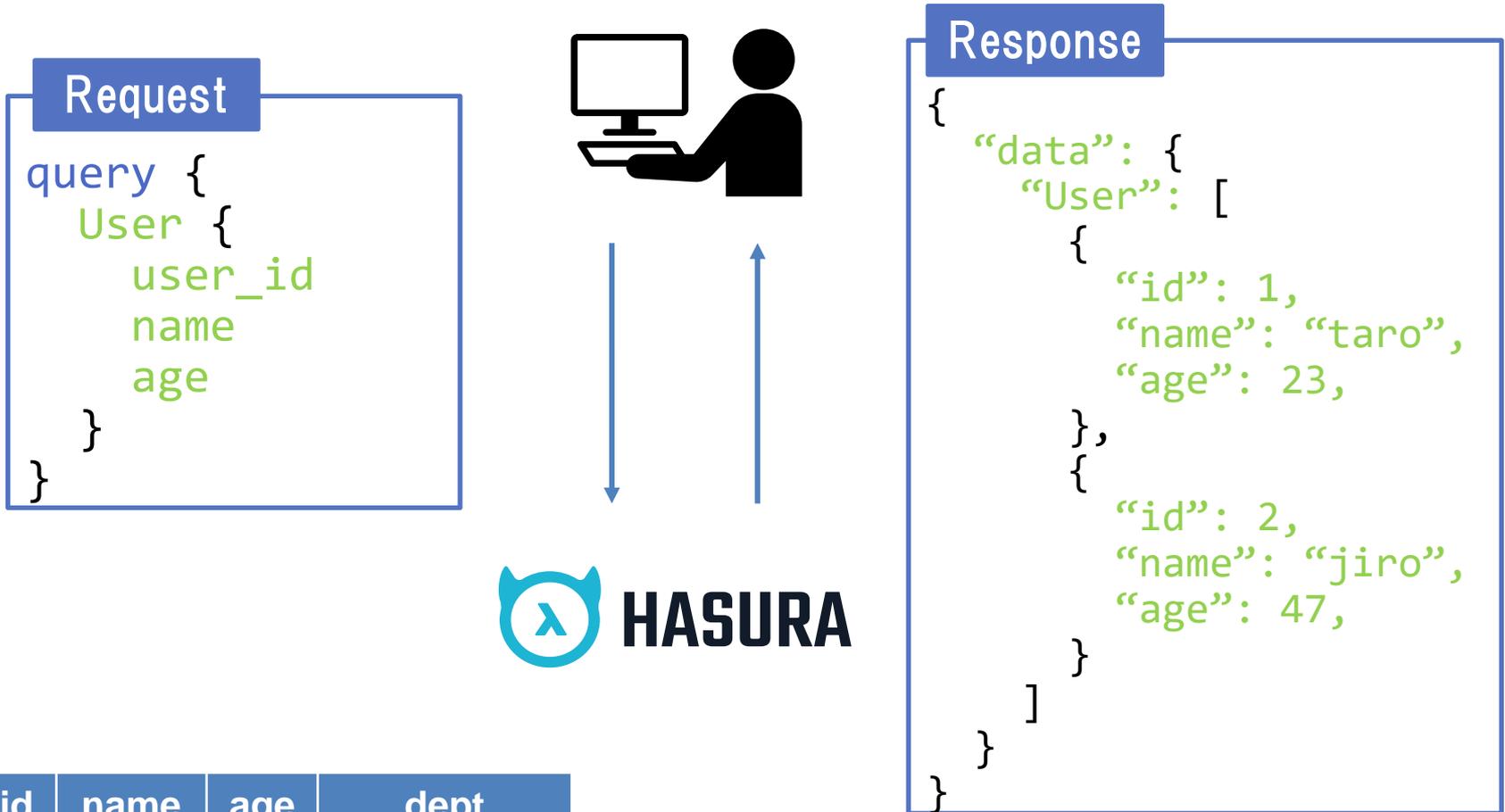
## Contents

---

1. GraphQLとHasuraの概要
- 2. Hasuraの主要機能**
3. Hasuraの認証・認可機能
4. Keycloakを使用したHasuraの認可・認証環境の構築
5. まとめ

# 2-1. Database

Hasura上でDBに対してテーブルを作成するとGraphQLスキーマに自動変換され、GraphQL APIで使用できるようになる



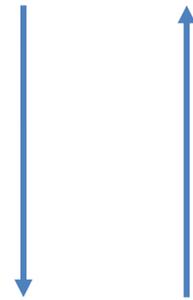
user_id	name	age	dept
1	taro	23	finance
2	jiro	47	development

# 2-1. Database

## Request

```

Mutation insert_article {
  insert_User (
    objects: [
      {
        user_id: 3
        name: "saburo"
        age: 35
        dept: "legal"
      }
    ]
  ) {
    returning {
      user_id
      name
      age
    }
  }
}
  
```



## Response

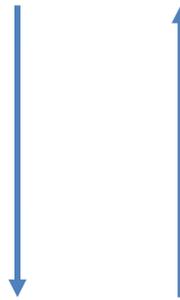
```

{
  "data": {
    "insert_User": {
      "returning": [
        {
          "id": 3,
          "name": "saburo",
          "age": 35,
        }
      ]
    }
  }
}
  
```

user_id	name	age	dept
1	taro	23	finance
2	jiro	47	development
3	saburo	35	legal

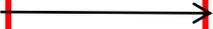
# 2-1. Database

Hasura上でテーブル間のRelationを設定するとGraphQLスキーマで  
テーブル同士を結合することが可能



device_id	name	rental_date	id
1	ipad	2022/11/23	3
2	iphone	2022/02/03	1

user_id	name	age	dept
1	taro	23	finance
2	jiro	47	development
3	saburo	35	legal



# 2-1. Database

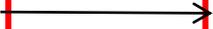
Hasura上でテーブル間のRelationを設定すると、  
テーブル同士を結合することが可能

```
Request
query {
  Device {
    device_id
    name
    User {
      name
      department
    }
  }
}
```



device_id	name	rental_date	id
1	ipad	2022/11/23	3
2	iphone	2022/02/03	1

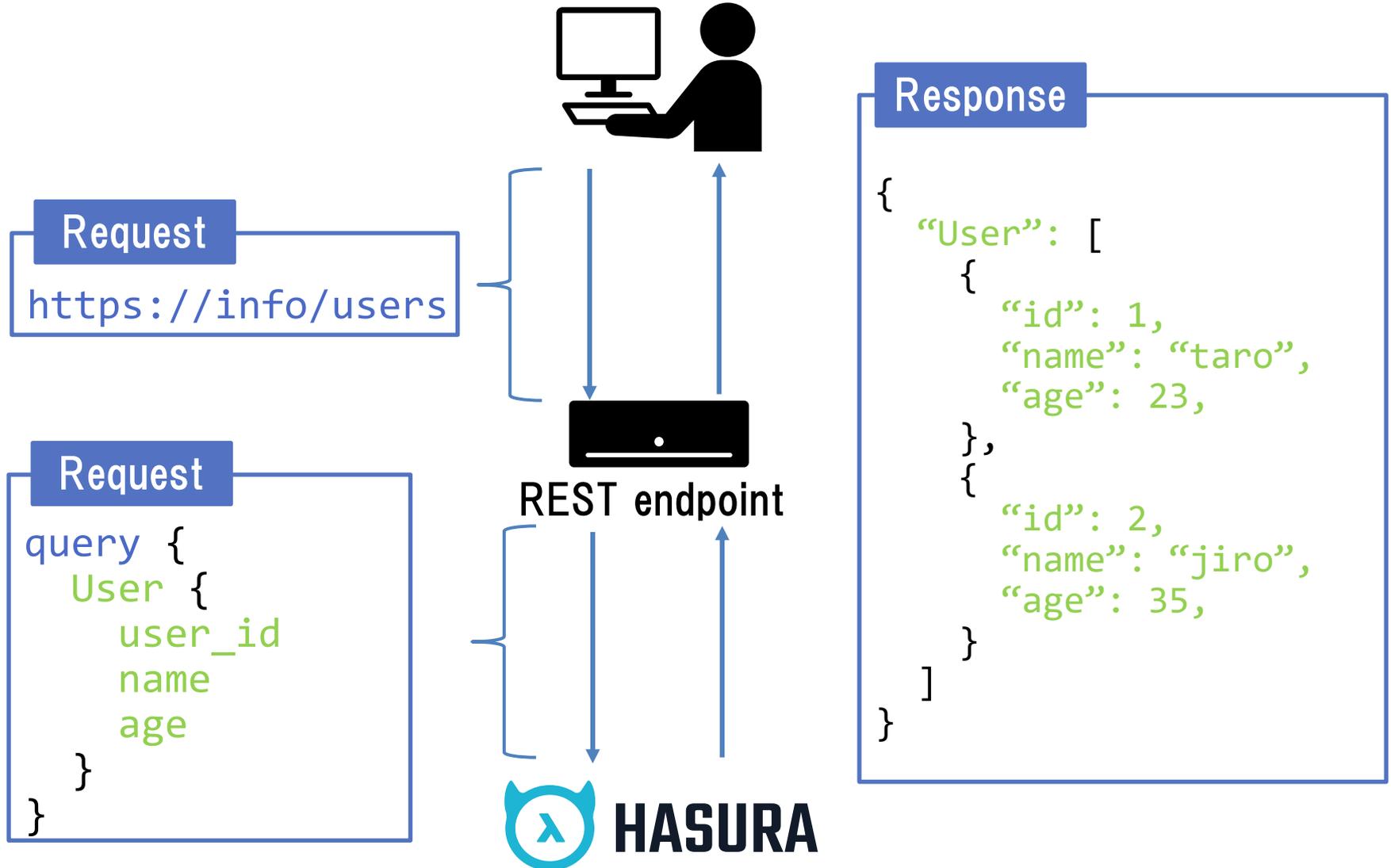
us



```
Response
{
  "data": {
    "Device": [
      {
        "device_id": 3,
        "name": "ipad",
        "User": {
          "name": "saburo",
          "dept": "legal",
        }
      },
      {
        "device_id": 1,
        "name": "iphone",
        "User": {
          "name": "taro",
          "dept": "finance",
        }
      }
    ]
  }
}
```

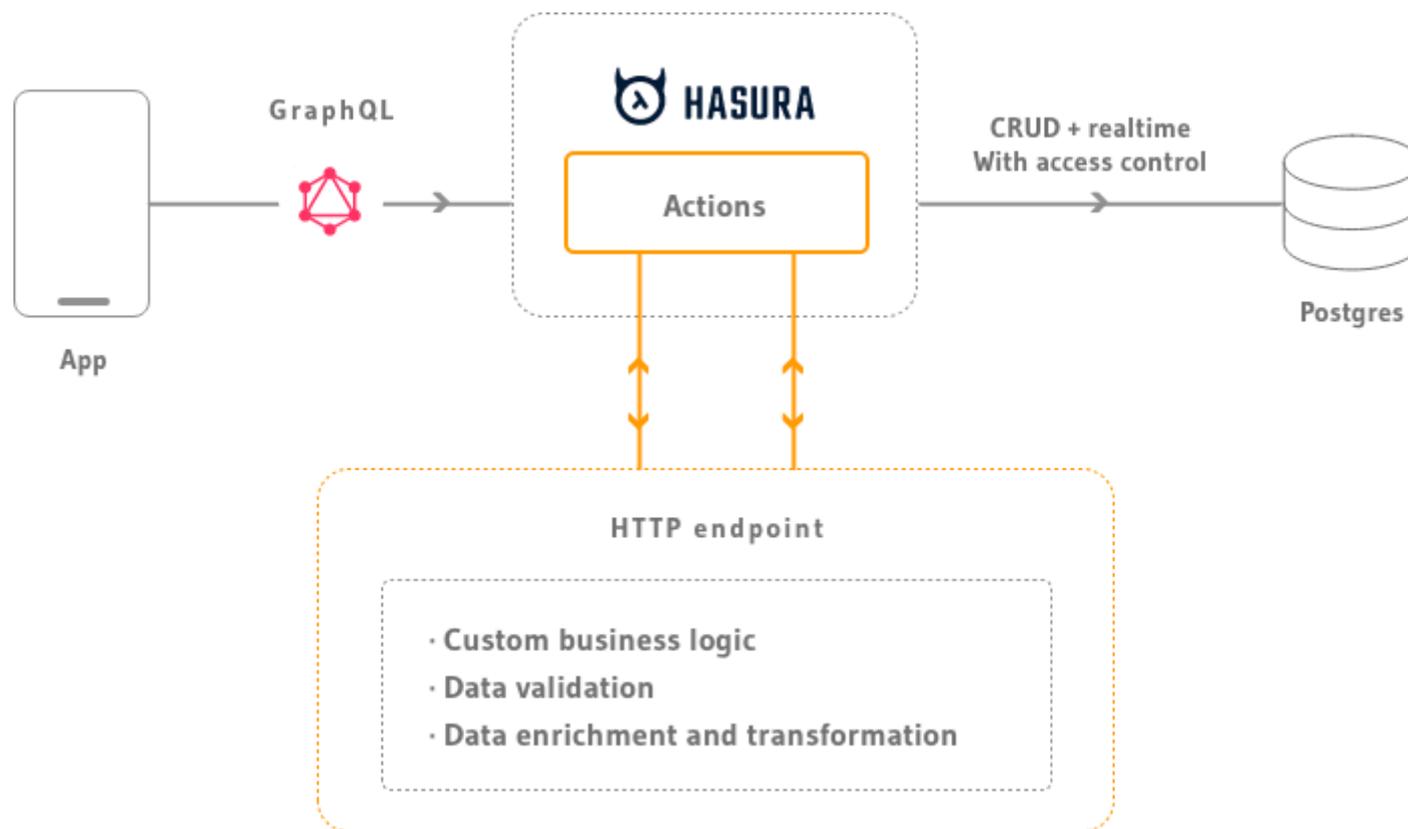
# 2-2. RESTified GraphQL Endpoints

HasuraではGraphQLリクエストをRESTエンドポイントとして公開可能



## 2-3. Actions

- Hasuraでは外部APIと連携することでDB操作以外のユーザー独自のビジネスロジックを組み込むことが可能
- Actionは特にRESTエンドポイントを呼び出す際に使用



(<https://hasura.io/docs/latest/graphql/core/actions/index.html>)

## 2-3. Actions

- Hasuraでは外部APIと連携することでDB操作以外のユーザー独自のビジネスロジックを組み込むことが可能
- Actionは特にRESTエンドポイントを呼び出す際に使用

### Action

```

type Query {
  Login (
    arg1: AccessInput!
  ): AccessOutput
}

type AccessInput {
  user: String!
  pass: String!
}

type AccessOutput {
  accesstoken: String!
}

```



### Handler

```

const handler = async (req, res) => {
  const {user, pass} = req.body.input;
  return res.json ({
    "accesstoken": "4f3GFm"
  })
};

```



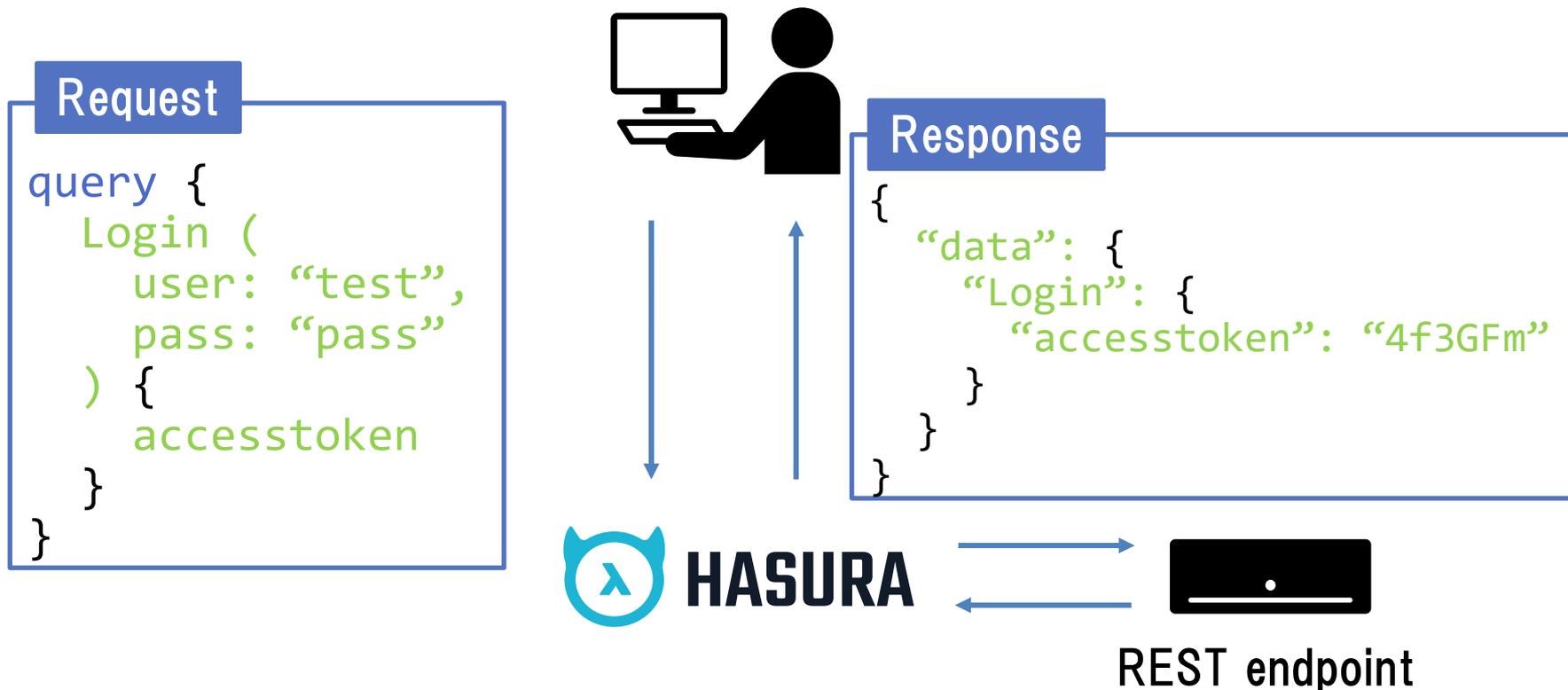
HASURA



REST endpoint

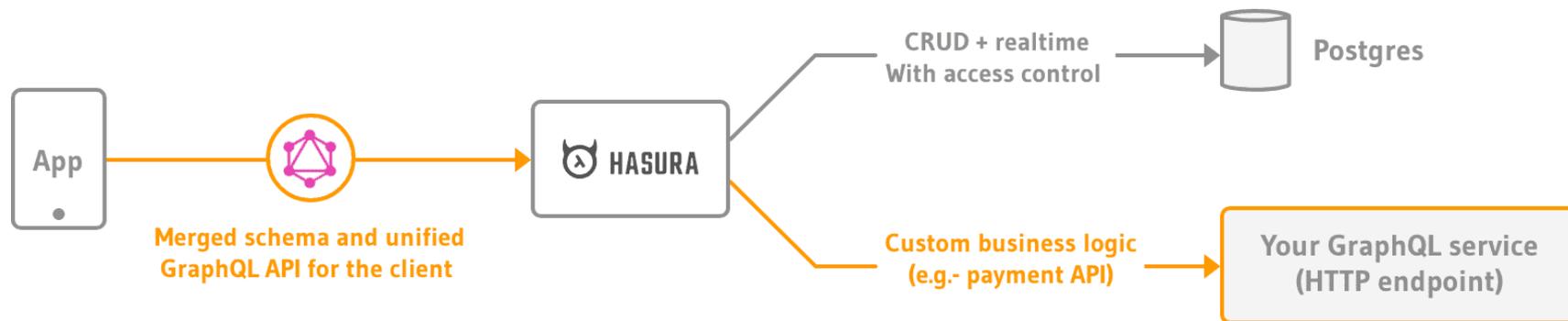
## 2-3. Actions

- Hasuraでは外部APIと連携することでDB操作以外のユーザー独自のビジネスロジックを組み込むことが可能
- Actionは特にRESTエンドポイントを呼び出す際に使用



## 2-4. Remote Schemas

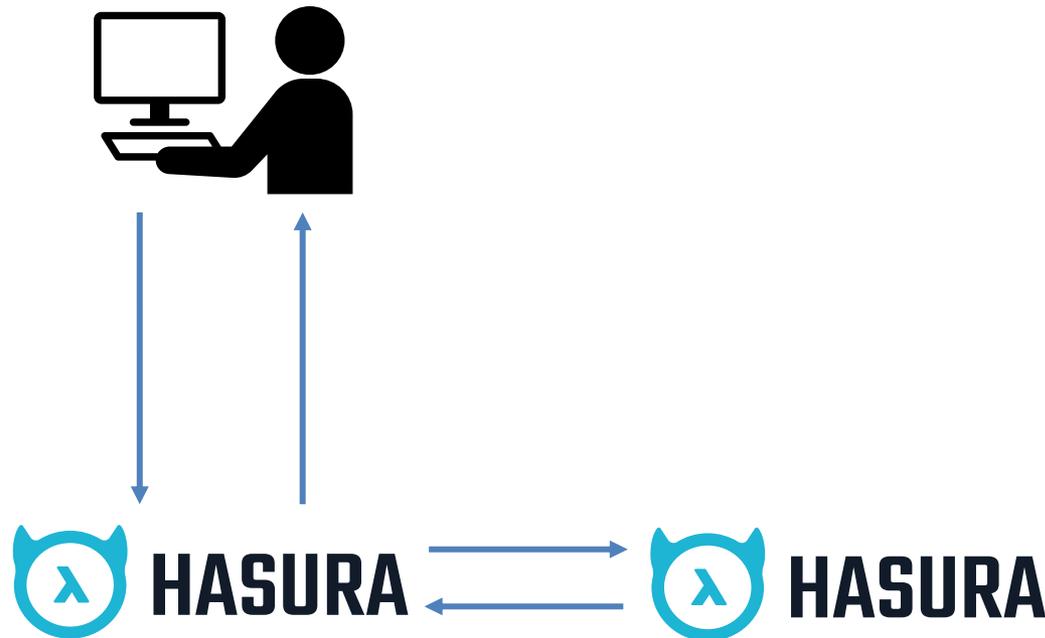
- Hasuraは他のGraphQLスキーマに接続し、GraphQL APIを統合することが可能
- Remote Schemasを利用することで統合元のスキーマと区別せずに外部のGraphQLスキーマにアクセスすることが可能



(<https://hasura.io/docs/latest/graphql/core/remote-schemas/index.html>)

## 2-4. Remote Schemas

- Hasuraは他のGraphQLスキーマに接続し、GraphQL APIを統合することが可能
- Remote Schemasを利用することで統合元のスキーマと区別せずに外部のGraphQLスキーマにアクセスすることが可能

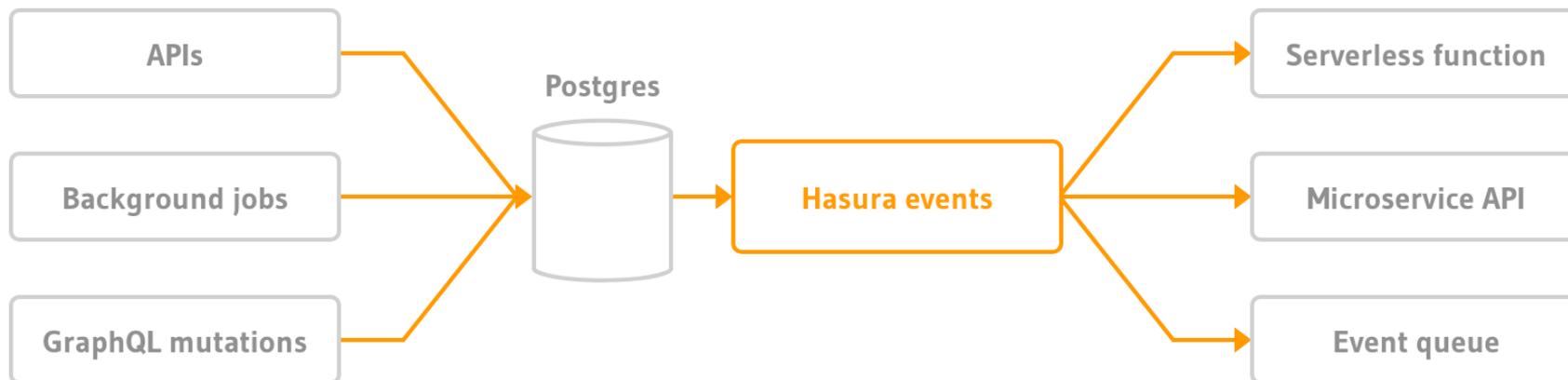


device_id	name	rental_date	id	user_id	name	age	dept
1	ipad	2022/11/23	3	1	taro	23	finance
2	iphone	2022/02/03	1	2	jiro	47	development
				3	saburo	35	legal

An arrow points from the 'id' column of the first table to the 'user\_id' column of the second table, indicating a relationship between the two datasets.

## 2-5. Event Triggers

- DBで特定のイベント(INSERT、UPDATE、DELETE操作等)が発生した際にあらかじめ定義された外部APIを呼び出す機能
- 外部APIへの通知はat least once方式で実施



(<https://hasura.io/docs/latest/graphql/core/event-triggers/index.html>)

## 2-5. Event Triggers

- 用途としては会員データ登録後のユーザー通知といったポーリングするほど高頻度でないDB更新で処理が開始されるケースがある
- またEvent TriggersのJSON形式のPayloadには発生原因となったイベントや変更前後の値が含まれているので、ボイラープレートとしても利用可能

Create a new event trigger

Trigger Name ⓘ

Database ⓘ

Schema/Table ⓘ

Trigger Operations ⓘ

Insert  Update  Delete  Via console ⓘ [\(Know more\)](#)

Webhook URL ⓘ

URL

Note: Specifying the webhook URL via an environmental variable is recommended if you have different URLs for multiple environments.

> Advanced Settings

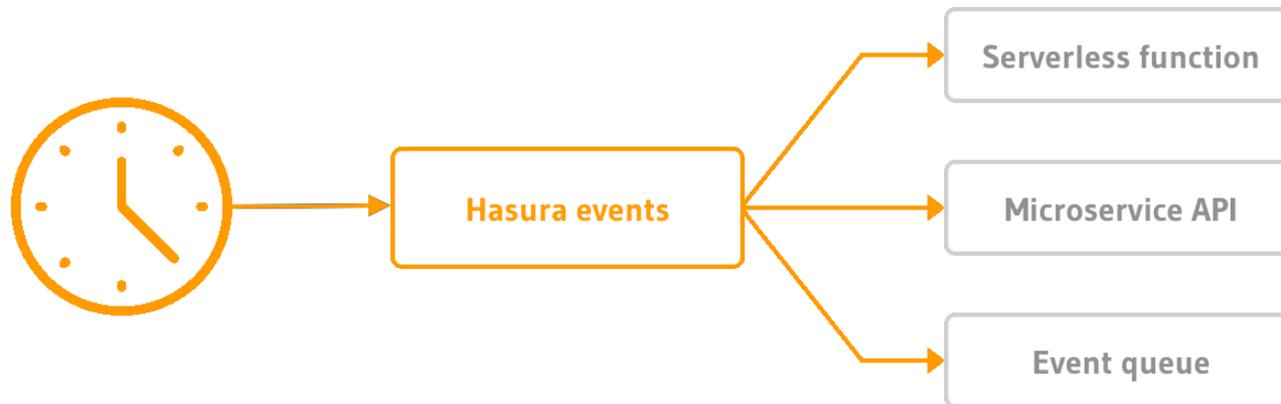
Create Event Trigger

```
{
  "event": {
    "session_variables": <session-variables>,
    "op": "<op-name>",
    "data": {
      "old": <column-values>,
      "new": <column-values>
    }
  },
  "created_at": "<timestamp>",
  "id": "<uuid>",
  "trigger": {
    "name": "<name-of-trigger>"
  },
  "table": {
    "schema": "<schema-name>",
    "name": "<table-name>"
  }
}
```

(<https://hasura.io/docs/latest/graphql/core/event-triggers/payload.html>)

## 2-6. Scheduled Triggers

- ユーザーが定義したロジックを指定した時間に実行する機能
- トリガの設定方式には定期実行するcron triggersと設定した時刻に1度だけ実行するone-off scheduled eventsがある



## 2-6. Scheduled Triggers

- どちらの形式でもHasura上で各イベントのステータス、ログが見れる
- リトライ回数やインターバル、リトライ期限も設定できるため、ジョブ管理サーバーとして用いることができる

You are here: [Events](#) > [Scheduled](#) > [eod\\_reports](#) > pending

**eod\_reports**

Modify Pending events Processed events Invocation logs

Filter

-- column -- -- op -- -- value --

Sort

-- column -- asc

Run query

	id	status	scheduled_time	created_at	tries
▶	fc4e272e-cc05-4de0-8abd-68...	⊙	Fri, Jun 19th 15:00:00 -07:00	Thu, Jun 18th 19:11:59 -07:00	0
▶	59260c54-dcb8-44e6-be7f-55...	⊙	Mon, Jun 22nd 15:00:00 -07:00	Thu, Jun 18th 19:11:59 -07:00	0
▶	b17a4b30-1ba9-43b4-b4ea-b...	⊙	Tue, Jun 23rd 15:00:00 -07:00	Thu, Jun 18th 19:11:59 -07:00	0
▶	d92e18da-0f57-4825-b3f9-77...	⊙	Wed, Jun 24th 15:00:00 -07:00	Thu, Jun 18th 19:11:59 -07:00	0
▶	eb65de3b-80fd-41a9-80cb-13...	⊙	Thu, Jun 25th 15:00:00 -07:00	Thu, Jun 18th 19:11:59 -07:00	0
▶	41896fa5-3134-4d4f-8486-3fe...	⊙	Fri, Jun 26th 15:00:00 -07:00	Thu, Jun 18th 19:11:59 -07:00	0
▶	064cb204-4054-4352-b889-b...	⊙	Mon, Jun 29th 15:00:00 -07:00	Thu, Jun 18th 19:11:59 -07:00	0
▶	10134d82-47ce-4f3b-b303-31...	⊙	Tue, Jun 30th 15:00:00 -07:00	Thu, Jun 18th 19:11:59 -07:00	0
▶	65c3243b-b6e5-4af6-a6a9-f8...	⊙	Wed, Jul 1st 15:00:00 -07:00	Thu, Jun 18th 19:11:59 -07:00	0
▶	428c9c3b-9354-4cd8-b795-00...	⊙	Thu, Jul 2nd 15:00:00 -07:00	Thu, Jun 18th 19:11:59 -07:00	0

Previous Page 1 of 10 10 rows Next

(<https://hasura.io/docs/latest/graphql/core/scheduled-triggers/create-cron-trigger.html>)

## 2-7. Migrations & Metadata

- HasuraではDBのスキーマやHasuraのメタデータをファイルとして出力でき、これを使用して類似環境の構築が可能
- 出力したファイルは編集することで環境のカスタマイズが可能
- 各時点でのスキーマやメタデータをバージョンで管理するので、環境をロールバックすることも可能

### Hasura CLI

```
hasura migrate apply
--version 1234567890
--type down
--database-name db1

git checkout meta1
Hasura metadata apply
```



## 2-8. Deploying

- Hasuraではデプロイする際のガイドや本番環境でセキュアなGraphQLエンジンを実現するためのガイドが提供されている
- 具体例としては以下が挙げられる
  - Set an admin secret
    - コンソールやエンドポイントへの不正アクセスを防ぐために設定
    - Hasuraでユーザー認証を行う際には必ず設定
  - Enable HTTPS
    - Hasura自身はAPIをSSL/TLSで保護できないため、APIをセキュアにするにはリバースプロキシを設定する必要がある
  - Configure logging
    - Hasuraではquery-log(Hasuraのquery全般に関するlog)やhttp-log(HasuraにHTTPリクエストした際のlog)等のlogが12種類提供されており、その中で出力するlogを設定できる

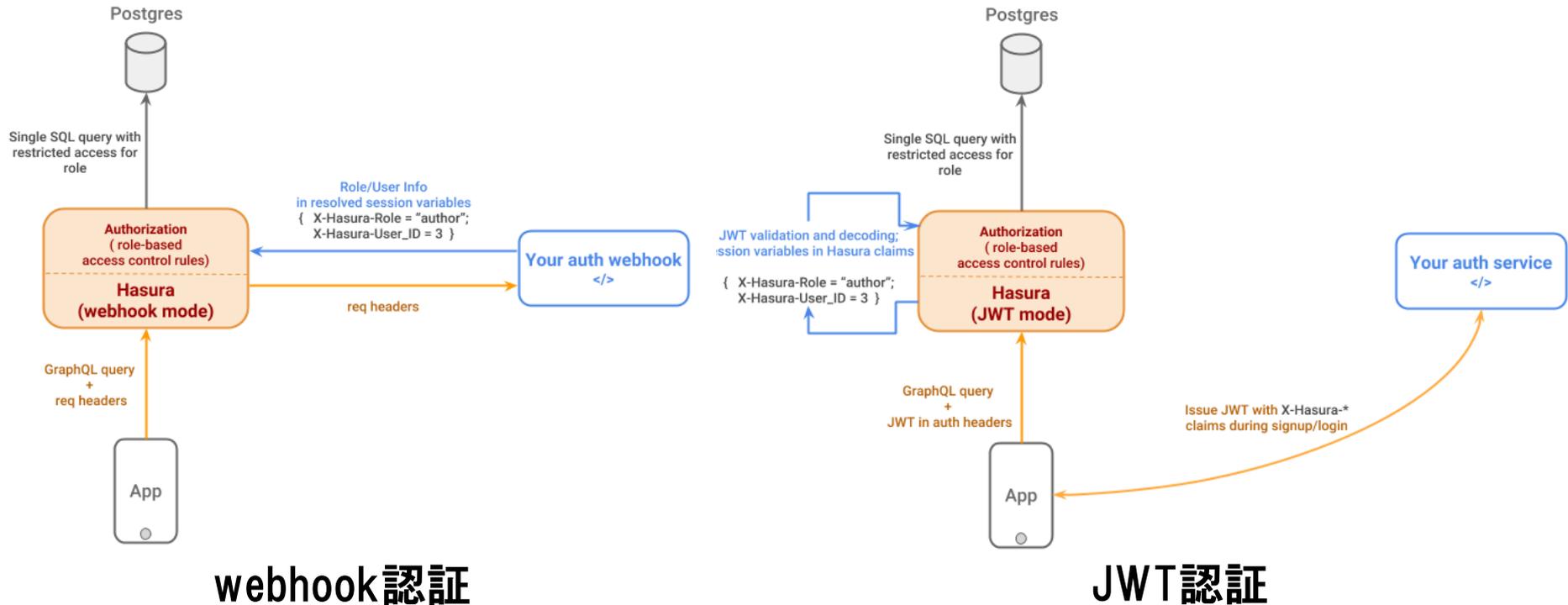
## Contents

---

1. GraphQLとHasuraの概要
2. Hasuraの主要機能
- 3. Hasuraの認証・認可機能**
4. Keycloakを使用したHasuraの認可・認証環境の構築
5. まとめ

# 3-1. Authentication

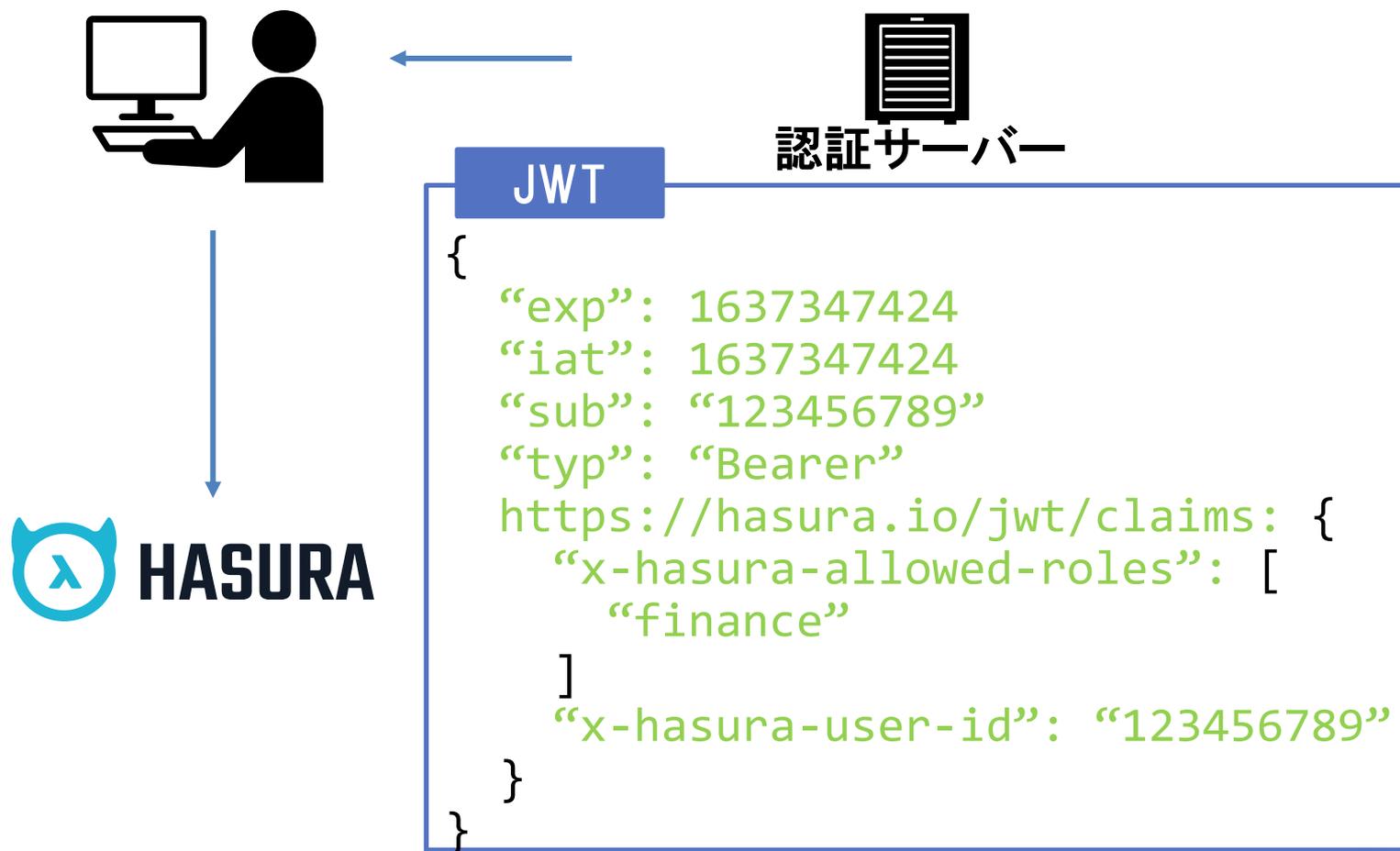
- Hasuraではユーザー認証はHasura外で行う必要があり、認証方式にはwebhook認証とJWT(JSON Web Token)認証の2種類が存在
- いずれの方式もSession Variablesという変数(X-Hasura-Role等)をHasuraに渡すことでユーザーの識別を実施



(<https://hasura.io/docs/latest/graphql/core/auth/authentication/index.html>)

## 3-1. JWT認証

- JWT認証ではまずクライアントが認証サーバーからJWTトークンを取得し、そのJWTをヘッダに含めたHTTPリクエストをHasuraに送信
- JWT内のclaimに記述されたX-Hasura-Allowed-Rolesの中からユーザーに割り当てるRoleを選択し、後述のAuthorizationを実施



# 3-2. Authorization

- Hasuraにおいてアクセス制御はRoleを用いて実施
- Role、Table、CRUD毎にPermissionと呼ばれるアクセスルールを設定し、DBの行・列単位で制御

Role	insert	select	update	delete	
admin	✓	✓	✓	✓	
finance	✗	⌵	✗	✗	<input type="checkbox"/>
manager	✗	✓	✗	✗	<input type="checkbox"/>
user	✗	⌵	✗	✗	<input type="checkbox"/>
Enter new role	✗	✗	✗	✗	

Close Role: finance Action: select

▼ Row select permissions ⓘ - with custom check

Allow role **finance** to select rows:

Without any checks  
 With custom check: ⓘ

```

1 [{"department":{"_eq":"財務部"}}]

{
  " department ▼ ": {
    " _eq ▼ ": " 財務部 " [X-Hasura-User-Id]
  }
}
    
```

Limit number of rows:  ⓘ

▼ Column select permissions ⓘ - partial columns

Allow role **finance** to access columns:

user\_num  
  name  
  department  
  password

▶ Aggregation queries permissions ⓘ - disabled

## Contents

---

1. GraphQLとHasuraの概要
2. Hasuraの主要機能
3. Hasuraの認証・認可機能
- 4. Keycloakを使用したHasuraの認可・認証環境の構築**
5. まとめ

## 4-1. 検証環境の構築

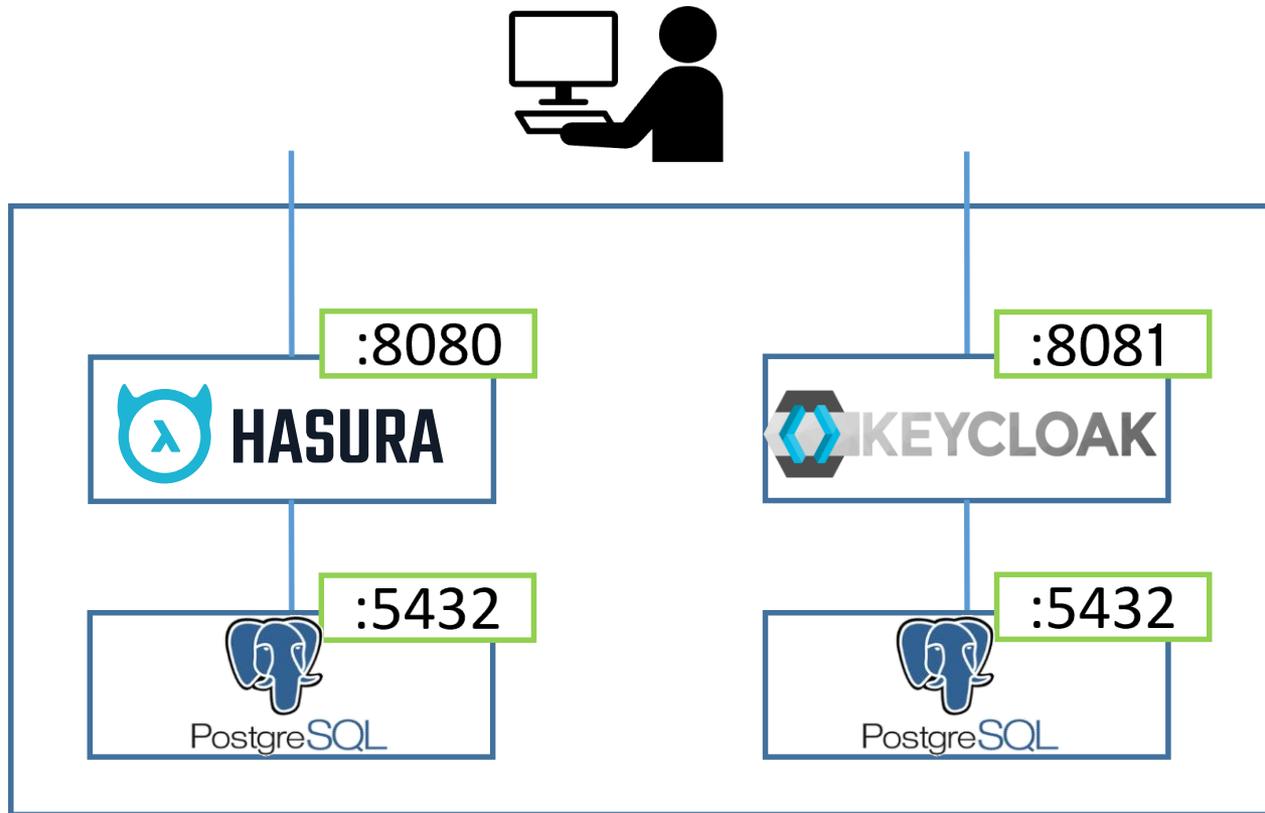
Hasuraに接続された以下のテーブルに  
3つのRole”manager”、”user”、”finance”でアクセスすることを想定

user_num	name	department	password
1	太郎	財務部	aa1
2	次郎	法務部	bki4
3	三郎	総務部	csu27
4	四郎	法務部	dte256
5	五郎	財務部	eno3125
6	六郎	財務部	fha46656

Role	説明
manager	全てのユーザー情報にアクセスできるRole
user	自らのユーザー番号と一致するデータにのみアクセスできるRole
finance	財務部に所属するユーザーのpassword以外の属性情報を取得できるRole

# 4-1. 検証環境の構築

認証・認可サーバにはKeycloakを使用し、Keycloakから取得した各RoleのHasuraにおける挙動を確認する



## 4-2. Keycloakの設定

“Clients”として”hasura”を作成し、“hasura”内の“Mapper”でJWTのclaimに組み込むfieldを指定する

Hasura 

Settings Keys Roles Client Scopes **Mappers** Scope Revocation Sessions Offline Access Installation

Search... 

Name	Category	Type
x-hasura-default-role	Token mapper	Hardcoded claim
x-hasura-user-num	Token mapper	User Attribute
x-hasura-allowed-roles	Token mapper	User Client Role

Mapper名	説明
x-hasura-allowed-roles(必須)	認証したユーザーに割り当てるRole一覧をセットするfield
x-hasura-default-role(必須)	HasuraへのリクエストヘッダでRoleを指定しない場合に割り当てるRoleを設定するfield
x-hasura-user-num	Roleとは関係ないですが、HasuraでSession Variablesとして使用するfield

# 4-2. Keycloakの設定

次に”hasura”内の”Roles”でHasuraに渡すRoleを作成  
(今回は”manager”、”user”、”finance”を作成)

Hasura 

Settings Keys **Roles** Client Scopes ⓘ Mappers ⓘ Scope ⓘ Revocation Sessions ⓘ Offline Access ⓘ Installation ⓘ



Role Name	Composite	Description
<a href="#">finance</a>	False	
<a href="#">manager</a>	False	
<a href="#">user</a>	False	

## 4-2. Hasuraの設定

Role”manager”はDBの全てのユーザー情報にアクセスできるように  
Permissionには何も設定をしない

Close
Role: manager
Action: select

▼ Row select permissions ⓘ - *without any checks*

Allow role **manager** to select rows:

Without any checks

1 | {}

With custom check: ⓘ

Limit number of rows:  ⓘ

▼ Column select permissions ⓘ - *all columns*

Allow role **manager** to access columns: Toggle All

user\_id    name    department    password

> Aggregation queries permissions ⓘ - *disabled*

Save Permissions
Delete Permissions

> Clone permissions ⓘ

## 4-2. Hasuraの設定

Role”user”は自らのユーザー番号と一致するデータにのみアクセスできるようにしたいので、JWT中のX-Hasura-User-Numとテーブルのuser\_numが一致する行のみ取得できるように”Row select permissions”を設定

Close Role: user Action: select

▼ Row select permissions ⓘ - with custom check

Allow role **user** to select rows:

Without any checks

With custom check: ⓘ

```

1 [{"user_num":{"_eq":"X-Hasura-User-Num"}}]

{
  "user_num": {
    "_eq": "X-Hasura-User-Num" [X-Hasura-User-Id]
  }
}

```

Limit number of rows:  ⓘ

▼ Column select permissions ⓘ - all columns

Allow role **user** to access columns:

user\_num  name  department  password

> Aggregation queries permissions ⓘ - disabled

> Clone permissions ⓘ

## 4-2. Hasuraの設定

Role”finance”は財務部に所属する人のpassword以外の情報を取得するよう  
 ”Row select permissions”はdepartmentが財務部に一致する行み、  
 ”Column select permissions”ではpassword以外の列を設定

Close
Role: finance
Action: select

▼ Row select permissions ⓘ - with custom check

Allow role **finance** to select rows:

Without any checks  
 With custom check: ⓘ

```
1 [{"department":{"_eq":"財務部"}}]
{
  " department  ▼ ": {
    " _eq  ▼ ": " 財務部 " [X-Hasura-User-Id]
  }
}
```

Limit number of rows:  ⓘ

▼ Column select permissions ⓘ - partial columns

Allow role **finance** to access columns: Toggle All

user\_id   
  name   
  department   
  password

> Aggregation queries permissions ⓘ - disabled

Save Permissions
Delete Permissions

> Clone permissions ⓘ

## Contents

---

1. GraphQLとHasuraの概要
2. Hasuraの主要機能
3. Hasuraの認証・認可機能
4. Keycloakを使用したHasuraの認可・認証環境の構築
- 5. まとめ**

## 5. まとめ

- Hasuraを使用することで対象をDBに限定されるもののGraphQL APIを容易に開発できる
  - Hasura上でテーブルを作成するとGraphQLスキーマを自動生成
- Role, Table, CRUD毎にPermissionと呼ばれるアクセスルールを設定することで、DBの行単位、列単位での制御を実施
  - JWT認証ではクライアントがアクセストークンを取得するのでクライアント側のAppによってRoleを切り替えることも可能

# Trademarks

- HASURAは、HASURA, INC.の登録商標です。
- PostgreSQLはPostgreSQL Community Association of Canadaの登録商標です。
- GraphQLはGraphQL Foundationが管理する商標です。商標およびロゴの使用は、LF Projectsの商標ポリシーに従います
- その他記載の会社名、製品名などは、それぞれの会社の商標もしくは登録商標です。

**HITACHI**  
Inspire the Next 