Backend development Class 03, Series 03

Hasura

CLASS 03

01 What is Hasura?

02 Hasura Features

03 Hasura Architecture

04 How can we use Hasura?

05 How does Hasura work?

What is Hasura?

Hasura is a backend as a service technology which enables one to provide an API directly from SQL databases.

Hasura features

Hasura comes with a lot of features that help you adopt GraphQL and a 3 factor architecture (read more about 3 factor & event-driven programming) quickly:

- **Make powerful queries:** Built-in filtering, pagination, pattern search, bulk insert, update, delete mutations.
- **Realtime:** Convert any GraphQL query to a live query by using subscriptions.
- **Merge remote schemas:** Access custom GraphQL schemas for business logic via a single GraphQL Engine endpoint.

Hasura features

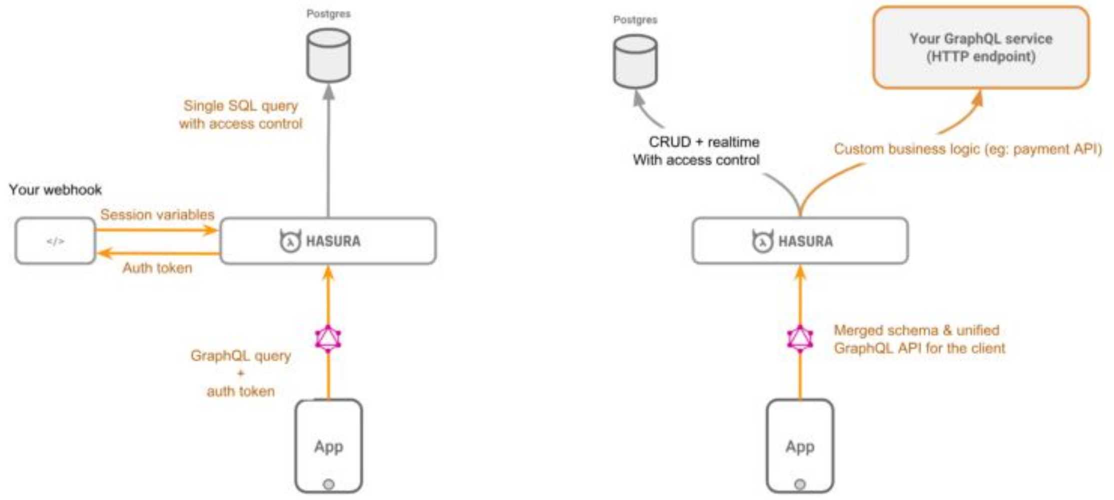
- **Trigger webhooks or serverless functions:** On Postgres insert/update/delete events.

- **Works with existing, live databases:** Point it to an existing Postgres database to instantly get a ready-to-use GraphQL API.

- **Fine-grained access control:** Dynamic access control that integrates with your auth system (eg: auth0, firebase-auth)

Hasura architecture

The Hasura GraphQL Engine fronts a Postgres database instance and can accept GraphQL requests from your client apps.


Postgres
Single SQL query with access control
Your webhook
Session variables
Auth token
HASURA
GraphQL query + auth token
App
Postgres
Your GraphQL service (HTTP endpoint)
CRUD + realtime With access control
Custom business logic (eg: payment API)
HASURA
Merged schema & unified GraphQL API for the client
App

How can I use Hasura?

There is Hasura Core and Hasura Cloud. Hasura Core is the open source GraphQL engine you can use for self-hosting. Hasura Cloud is a managed, hosted service.

In both cases you've to integrate Hasura with at least one SQL database (data layer).

How does Hasura work?

Hasura generates GraphQL API queries, mutations and subscriptions from a single or several database(s).

The database is connected using a database **URL**. It’s possible to merge the API generated from several databases into a single one.

Migrations & Metadata (CI/CD)

It is a typical requirement to export an existing Hasura **"setup"** so that you can apply it on another instance to reproduce the same setup. For example, to achieve a dev -> staging -> production environment promotion scenario.

Hasura needs **2** pieces of information to recreate your GraphQL API, the underlying PG database schema and the Hasura metadata which is used to describe the exposed GraphQL API.

Database migration files

The state of your PG database is managed via incremental SQL migration files. These migration files can be applied one after the other to achieve the final DB schema.

DB migration files can be generated incrementally and can by applied in parts to reach particular checkpoints. They can be used to rollback the DB schema as well.

Schema generation

The Hasura GraphQL engine automatically generates GraphQL schema components when you track a table/view in Hasura and create relationships between them.

Schema generation - Tables

When you track a table in the Hasura GraphQL engine, it automatically generates the following for it:

- A GraphQL type definition for the table
- A query field with where, order_by, limit and offset arguments
- A subscription field with where, order_by, limit and offset arguments
- An insert mutation field with on_conflict argument that supports upsert and bulk inserts
- An update mutation field with where argument that supports bulk updates
- A delete mutation field with where argument that supports bulk deletes

Schema generation - Views

When you track a view in Hasura GraphQL engine, it automatically generates the following for it:

- A GraphQL type definition for the view
- A query field with where, order_by, limit and offset arguments
- A subscription field with where, order_by, limit and offset arguments

Essentially the Hasura GraphQL engine does the same thing it would do for a table, but without creating the insert, update and delete mutations.

Schema generation - Relationships

When you create a relationship between a table/view with another table/view in the Hasura GraphQL engine, it does the following:

- Augments the type of the table/view by adding a reference to the nested type to allow fetching nested objects.
- Augments the where and order_by clauses to allow filtering and sorting based on nested objects.

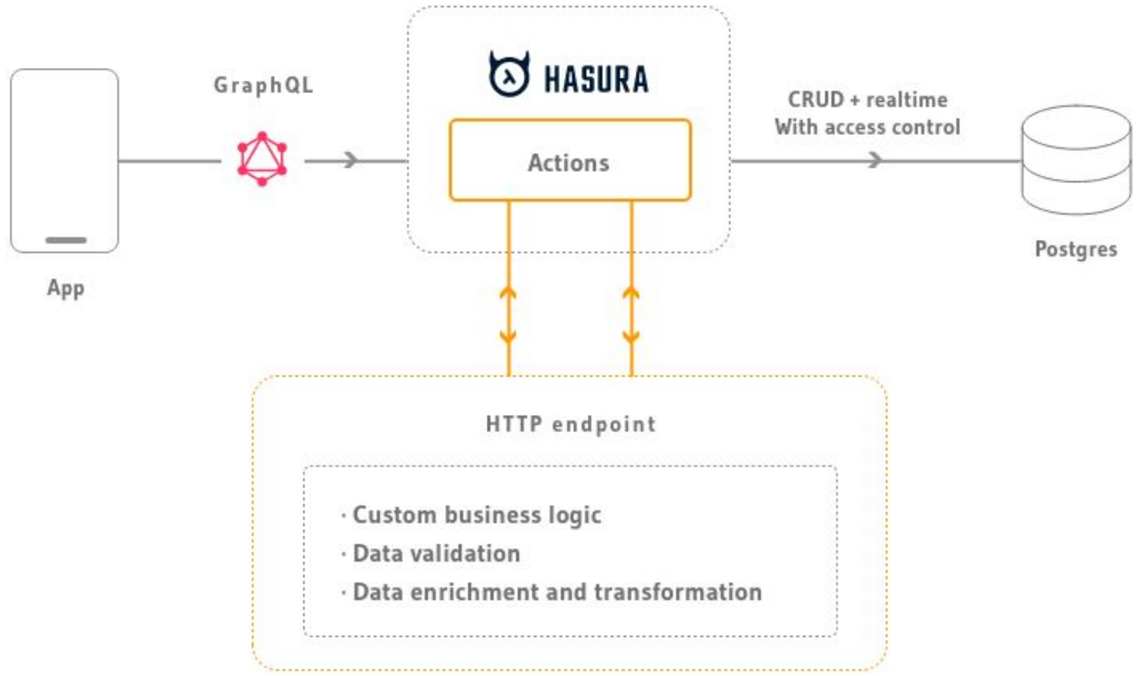
Hasura metadata files

The state of Hasura metadata is managed via snapshots of the metadata. These snapshots can be applied as a whole to configure Hasura to a state represented in the snapshot.

Hasura metadata can be exported and imported as a whole.

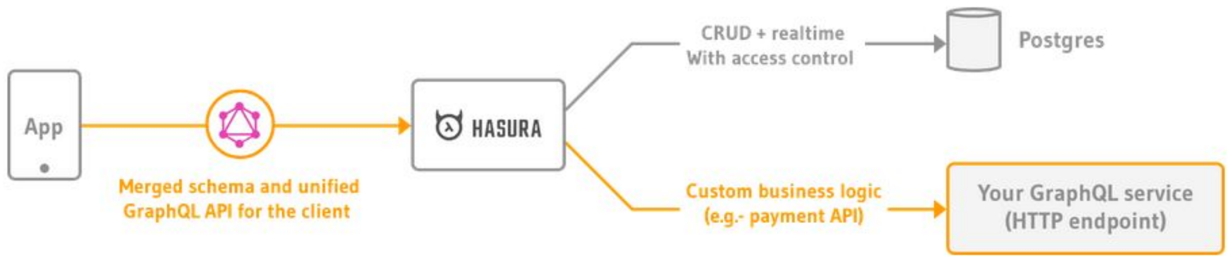
Actions

Actions are a way to extend Hasura's schema with custom business logic using custom queries and mutations.



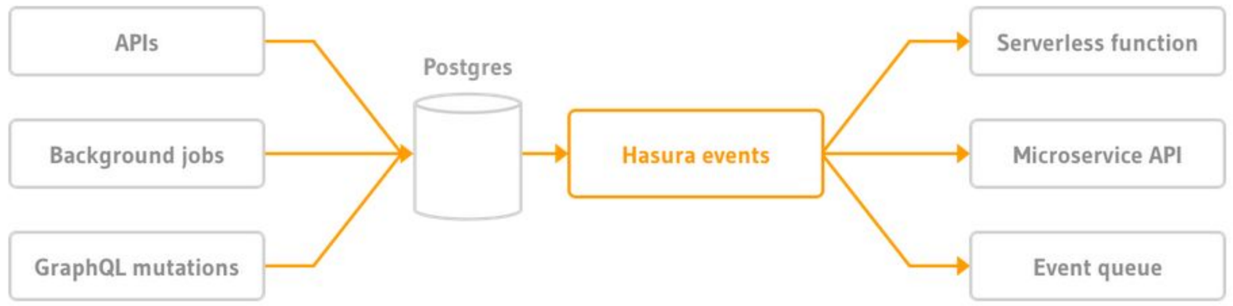
Remote schemas

Remote schemas are GraphQL schemas which are served externally from Hasura and are merged into the GraphQL schema generated from Hasura using schema stitching.


App
Merged schema and unified GraphQL API for the client
HASURA
CRUD + realtime With access control
Postgres
Custom business logic (e.g.- payment API)
Your GraphQL service (HTTP endpoint)

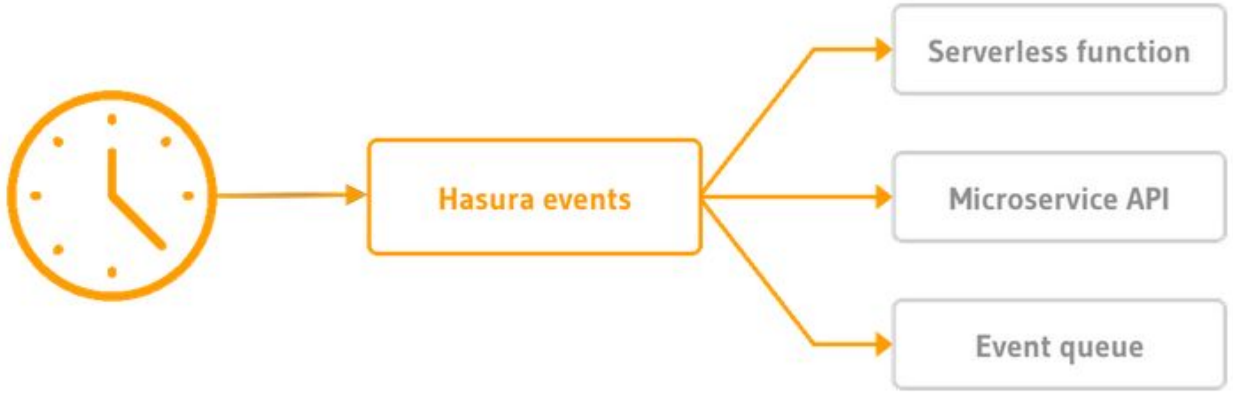
Event triggers

"Event triggers reliably capture events on specified (database) tables and invoke webhooks to carry out any custom logic." Events can be coupled to **INSERT, UPDATE, DELETE** and manually on a row (via CLI or API).



Scheduled triggers

"Scheduled triggers are used to execute custom business logic at specific points in time." The business logic may be implemented in serverless functions, microservices accessible via API or event queues.



MINAB IT SOLUTIONS

Thank you!

HahuJobs

ለሀገር ልጅ በሀገር ልጅ !

Endriyas Yeshidniber

yendriyas@gmail.com