GraphQL on Fly with Hasura

Warning: This document is old! It is likely wrong in some important way.

So you have a PostgreSQL database somewhere in the cloud and you want to modernize how you access it. GraphQL is likely on your list of things you want to implement, but tools like Hasura - which take care of the backend complexity - need to be installed on a server somewhere. Fly can help by putting that server where it needs to be and making API requests faster than ever.

Let's assume you have a Postgres database somewhere...

We made life easy and have our own sample app FlyDictionary with its own Deploy to Heroku button. Click that to create your own Heroku app and a Postgres backend.

And that you can get a URL with PostgreSQL user credentials to connect to that database.

For Heroku users, you can find those credentials by going to the Data view and selecting the database you want to use in the list. Then select the Settings tab and click on View Credentials. This will reveal a panel of various settings; the one you want is labelled URI.

Let's say that we got something like:

```
postgres://user:pass@serverhost.com:5432/databasename
```

We're now ready to deploy Hasura.

Create an App

```
$ flyctl apps create
```

If you don't have `flyctl`, you'll need to install it. Check out our Install Flyctl guide. If you haven't got a Fly account yet, run `flyctl auth signup` to get your own free account.

You will be asked for an app name at this point. Hit return to have one auto-generated for you. You'll see the app name when the create has been completed like this:

```
❯ flyctl apps create
? App Name (leave blank to use an auto-generated name)
? Select organization: demo (demo)
New app created
  Name     = still-dust-14
  Owner    = demo
  Version  = 0
  Status   =
  Hostname = <empty>

Wrote config file fly.toml
```

So, now we have a Fly app named `still-dust-14`.

Set the Stage

Now, before we deploy Hasura, there's one more step for a smooth deployment. We are going to set some environment variables and secrets in the app. In particular, we're going to pass over our database URL to the soon to be deployed application. For this we use `flyctl secrets set`.

```
$ flyctl secrets set \
  HASURA_GRAPHQL_DATABASE_URL="postgres://user:pass@serverhost.com:5432/datab
```

The Admin secret helps secure the Hasura GraphQL endpoint and console. For finer tuned authentication and role based security, consult the Hasura documentation.

The `HASURA_GRAPHQL_ENABLE_CONSOLE` environment variable, set to true, will allow you to log in to Hasura's interactive console where you can create queries.

Now, technically this whole step could take place after the app had deployed but that would mean allowing the app to spin its wheels looking for a database until we did set the values. This way, Hasura can start running properly as soon as it is deployed.

Fly Deploy

We can now deploy Hasura. We'll use the -i option of `flyctl deploy` which allows you to install publicly available images from Docker hub.

```
$ flyctl deploy -i hasura/graphql-engine:v1.1.0
```

The Fly platform will now go and deploy the Hasura docker image on one of the Fly nodes around the world.

When the deployment process has completed, running `flyctl info` will give the details of our Hasura instance:

```
❯ flyctl info
App
  Name     = still-dust-14
  Owner    = dj
  Version  = 2
  Status   = running
  Hostname = still-dust-14.fly.dev

Services
  TASK   PROTOCOL   PORTS
```

```
IP Addresses
  TYPE   ADDRESS                                CREATED AT
  v4     77.83.141.82                           1h16m ago
  v6     2a09:8280:1:ad2e:fa50:9986:a226:99d3   1h16m ago
```

The important part here is the App hostname - here it's `still-dust-14.fly.dev`.

Connecting to Hasura

Open your browser and go to the `/console` path on that host - `still-dust-14.fly.dev/console`. You'll be prompted for the admin secret we set in the secrets and once entered, you'll be in Hasura's GraphQL console. From there, you'll be able to create new tables in PostgreSQL, track existing tables and model your GraphQL API. Once you've done that, you can then create GraphQL queries. Of course, this wouldn't be how you would use Hasura day to day. You'd likely be using it as a GraphQL proxy for your applications.

```
❯ curl -X POST -H  "Content-Type: application/json" -H "x-hasura-admin-secret:O
{"data":{"example":[{"id":1,"name":"Fred"}, {"id":2,"name":"Wilma"}, {"id":3,"n
```

Once you do have your API configured, you can shut down the Hasura console by changing the console secret:

```
$ flyctl secrets set HASURA_GRAPHQL_ENABLE_CONSOLE=false
```

Fly will automatically redeploy Hasura with the console disabled, while still responding to your GraphQL API requests.