Platformatic Runtime API with Standalone DB APIs One Database POC

Objective

Thinking through a SaaS platform that offers fully featured apps like a Customer Management, Inventory Management and Invoices. The requirements would be:

- Each application has to be able to standalone and deployed independently.
- Each app should offer a set of APIs specific to that app only.
- A platform can be built where each of these apps are offered as plugins or paid services with a full APIs to support the platform. Think Zoho.

I played around with elixir and the 'umbrella' app concept using a domain driven design. Ran across Plaformatic and the Runtime setup thought it was real darn close to what I wanted to accomplish. The monolith that allowed me to build a domain-driven architecture with an underlying service-mesh tying it all together or picking and choosing which APIs to expose using the Platformatic Composer got me excited.

So a couple of the Platformatic Runtime as presented didn't quite fit:

- I needed to use a single database. At the end of the day it should be a multi-tenant architecture and there could be many different domains (Billing (Invoicing, Recurring Payment, BNPL...) Order Management (Carts, Shipping, Orders...), Product Management (Stock Management, Digital Downloads, Vendors/Suppliers, Inventory Control, Warehouse...), Customers (Customer Management, Leads, Underwriting/Approvals, Chats/Messages, company/orgs/contacts relationships, addresses, emails, phones...), Payments (Processing, history, refunds, voids, chargebacks...) and so on. I am too inexperienced/scared to manage the complexity of tenancy and relationships and back-up/recovery and all that comes with managing many different databases. I am also considering row-level security and encryption so add that to my fear of managing multiple 'microservice' databases - one for each domain.
- Becuase I wanted each domain app (Platformatic DB app) in the Runtime to operate and be deployed independently I needed to control the port assignments. Runtime (as presented) requires I select an 'entryPoint' DB (domain) app and assigns random port numbers to all the other domain apps.

I fiddled around for a bit and figured out how to accomplish what I need.

> There is a little extra work involved (not much) and I hope the Platformatic team can review this, see some value and bridge the gap.

CAVEAT

I just got all this working and did some preliminary testing using the REST API. I did not venture into the graphql not have I extensively tested the REST APIs. What I did test worked as expected so I wanted to get this POC up then do the deep dive on what may someday be my unicorn 😀. I am not a - what do

they call it these days? I am not a 10x Programmer and there may be some obvious flaws in my thinking or this setup in general. I hope to get some friendly feedback in the Issues one way or the other.

At the end of the day this setup should meet my needs. I'll have standalone apps exposing APIs that are only relevant to the app and a full set of all apis in the runtime exposed for Composer where I can pick what I need to model my platform app.

The POC

This repository consists of a Platformatic Runtime environment with 4 DB apps:
1 Customers (domain standalone app)
2 Products (domain standalone app)
3 Invoices (domain standalone app)
4 Shared (tables/apis that are not intended to be a standalone app but provide supporting tables/APIs to associate with the standalone domain apps.)

- Each app has a Plugin example endpoint
- Each app only presents APIs that are necessary for the app.
- The Customers app has fk relationships with the shared addresses, emails and phones tables
- The Invoices app has an invoices table with a fk relationship to customer and a line_items table with fk relationship to invoices and products
- Each app has been assigned a static PORT and can be run independently with access to Platformatic splash page, swagger docs and graphql specific to the app domain
- The Runtime (root) app is assigned it's own PORT to a Platformatic splash page with swagger docs and graphql to all tables/APIs
- The migrations are setup to run from the Runtime root to a single database (sqlite). None of the apps have a database or migration requirement, they just work.
- The plugins and routes for each app reference the global.d.ts types at the root (all schemas, entities, hooks) so you should be able to build plugins that can access other apps (have not tried this yet)

Setup

I commented out the gitignore for the sqlite db and env files so you can clone repo and after package installs be ready to run.

- Clone repository and use your favorite package tool npm, yarn, pnpm or whatever else is out there. I use pnpm.
- cd to clone and `pnpm i` the Runtime and `pnpm run start`. Open browser to http://localhost:4040 to Platformatic start page. You can visit the swagger docs to see all available API endpoints.
- Open a terminal in each of the apps in the 'services' folder and `pnpm i` then `pnpm run start`. Open the browser for each app at their respective ports and you should get the Platformatic start page with swagger docs specific for each of the domains APIs.

Roll your own

Following are the steps needed to create or add to this repo.

1. Create a Platofrmatic Runtime app `pnpm create platformatic@latest`
2. While going through the wizard create at least one DB app. Reply 'Y' to run pnpm install on Runtime.
3. When creating your service, select your DB of choice, 'y, to the connection string, 'y' to default migrations, 'n' to apply migrations, 'y' to create plugin, 'y' if for TS if you want to drop productivity by 30%. Create as many DBs (domains) as you want. I did not try to setup services yet but seems to me they should behave as expected in this setup.
4. Select any app for entry, Set any port number - we will be editing these.

Install deps and verify apps are working

1. cd into your Runtime app, `pnpm run start` and open the Runtime app in a browser make sure it is working. It should have at least one 'example' endpoint in the api It will create a schema.lock and tell you no tables yet. No biggie go ahead and ctrl+c close it
2. Open a terminal in each of your apps and `pnpm i` to install packages then `pnpm run start` each app and make sure it runs also. Each app should have an 'example' endpoint api. ctrl+c Close the app.

Runtime configuration

1. In any one of your apps, copy the migrations folder, the plugins folder, the routes folder and the platformatic.db.json folder and past them in the Runtime (root) folder. The following steps will be working in the Runtime (root) folder only
2. Open the plugins/example.js file and set the /// to `/// <reference path="../global.d.ts" />`. You can change the 'foobar' to 'runtime' so we have a good reference when testing.
3. Open the routes/root.js file and set the /// to `/// <reference path="../global.d.ts" />`. Change the '/example' endpoint path to 'runtime/example' so we have a good reference when testing.
4. In the .env variable, set the PORT variable to the port you want tuse to open the Runtime app with all the API references. Change the name for the DB variable and set value to `PLT_DATABASE_URL=sqlite://./db.sqlite`
5. In your platformatic.db.json set the db.connectionString to `{PLT_DATABASE_URL}` and add a new "server" block to set HOST and PORT values. It should end up like this:

```
{
  "$schema": "https://platformatic.dev/schemas/v1.2.0/db",
  "db": {
    "connectionString": "{PLT_DATABASE_URL}",
    "graphql": true,
    "openapi": true,
    "schemalock": true
  },
  "watch": {
    "ignore": [
      "*.sqlite",
      "*.sqlite-journal"
    ]
  },
```

```
  "migrations": {
    "dir": "migrations"
  },
  "server": {
    "hostname": "127.0.0.1",
    "port": "{PORT}"
  },
  "plugins": {
    "paths": [
      {
        "path": "./plugins",
        "encapsulate": false
      },
      {
        "path": "./routes"
      }
    ]
  },
  "types": {
    "autogenerate": true
  }
}
```

6. In the platformatic.runtime.json set the entryPoint to `"."`. Not sure what effect this has overall as it seemed to work no matter what I put in there but the intent is to use itself as an entry point.
7. Make sure all files are saved then start the Runtime app `pnpm start`. The server should start on the port assigned in the .env and the swagger docs should provide the endpoint you set in step 3.
8. Close the app ctrl+c and `pnpm i @platformatic/db` install the Platformatic db package.
9. In the 'package.json' file add migrations to your scripts block `"migrate": "platformatic db migrations apply"`
10. The migrations folder should contain the default 'movies' SQL in the '001.do.sql' file. Let's change 'movies' to 'runtime' in the do and undo migration files for testing.
11. Make sure all files are saved then apply the migration `pnpm run migrate`.
12. After migrate finishes start your app and check the swagger docs to make sure the 'runtime' CRUD endpoints are there along with the plugin endpoint from step 3.
13. All of your database migrations for all apps should be crafted in the Runtime root mograions folder. Setup a new migration and add at least one table for each of your apps for testing with `pnpm platformatic db migrations create`. This should create '002.do.sql' and 002.undo.sql'. Open the '002.do.sql' and create a table for each of your db apps:

```
CREATE TABLE IF NOT EXISTS app1 (
  id INTEGER PRIMARY KEY,
  title TEXT NOT NULL
);

CREATE TABLE IF NOT EXISTS app2 (
  id INTEGER PRIMARY KEY,
```

```
  title TEXT NOT NULL
);
```

14. Add the table drops in '002.undo.sql'
15. Save your files and run `pnpm run migrate` to create these tables.
16. Start your runtime app again and check the swagger endpoints. You should see the new app endpoints there.
17. Check the plugins/example.js and routes/root.js `/// <reference path="../global.d.ts" />` make sure it can find the 'global.d.ts' file that was created after our migrations.

That's all for the Runtime configuration. If you want to add more apps or services you can run `pnpm create platformatic@latest` anytime and follow the instructions below.

Service Apps configuration

The following steps should apply to all DB apps in the 'services' folder.

1. Delete the 'migrations' folder. If there is a 'db.sqlite' file there delete that also.
2. Open the 'plugins/example.js' file and change '/// ' to point to the 'global.d.ts' file in the Runtime -> `/// <reference path="../../../global.d.ts" />`. Update 'foobar' to your app name for testing
3. Open the 'plugins/example.js' file and change '/// ' to point to the 'global.d.ts' file in the Runtime -> `/// <reference path="../../../global.d.ts" />`. Update '/example' route to to '< your app name >/example' for testing.
4. Open or create the '.env' file and change the DB to point to the same DB in the Runtime app. If you setup a PG database for example, the db connection string should be similar. If you setup sqlite in the Runtime, the db string should point to the db.sqlite file -> `sqlite://../..db.sqlite`. Do not change the env variable name, just the db value.
5. In the same '.env' file add a `PORT=xxxx` where xxxx is the port number you want to assign to the app. Save the file
6. Open the 'platformatic.db.json' file to add a 'server' block to specify your hostname and port and ensure the 'migrations' block is present. The json should look like this:

```
{
  "$schema": "https://platformatic.dev/schemas/v1.2.0/db",
  "db": {
    "connectionString": "{PLT_SYMPTOMATIC_STEWARDSHIP_DATABASE_URL}",
    "graphql": true,
    "openapi": true,
    "schemalock": true
  },
  "watch": {
    "ignore": [
      "*.sqlite",
      "*.sqlite-journal"
    ]
  },
  "server": {
```

```
    "hostname": "127.0.0.1",
    "port": "{PORT}"
  },
  "migrations": {
    "dir": "migrations"
  },
  "plugins": {
    "paths": [
      {
        "path": "./plugins",
        "encapsulate": false
      },
      {
        "path": "./routes"
      }
    ]
  },
  "types": {
    "autogenerate": true
  }
}
```

8. Complete the above steps for every app before proceeding.

Setting up API schemas

> This step is a little painful and error prone. Hoping if this POC has a place in the Platformatic stable their team can come save the day...

1. Make sure all the files are saved in each app. go to any one of the apps and delete the 'schema.lock' file if present. Start the app server with `pnpm start`. The server should start on the port specified in step 5.
2. Go to the site in your browser. You will see the swagger will show all the endpoints for all apps like it does when you visit the Runtime (root) swagger. This is not what we want, we only want the endpoints relevent for the app domain. Close your server with ctrl+c.
3. When you started the server it created a new schema.lock file that describes the schema for your database. Being we are using a single database and not a database per app the schema reflects all of the tables just like it does in the Runtime. To only show the APIs specific for your app you need to open the schema lock file and delete the tables that do not apply to the app. Once you do this, save the schema.lock file adn start your server you should only see the endpoints for the tables that remain in the schema.lock file.

NOTES

- IF your apps have relationships with other tables in other apps you need to make sure those tables are present in the schema.lock file also. See the 'customers' app in this repo for an example where the 'addresses', 'emails' and 'phones' tables from the shared app are included in the customer schema.lock file.

- If you update any of your apps tables in the Runtime migrations you need to close the app, delete the schema.lock and start the app apgain to create a new schema.lock with the schema and remove the tables you do not need in the app again. Conversely you can go to the Runtime schema.lock, find the schema object(s) that were just migrated, close your app server and add them to your app schema.lock file and restart.

> **PLATFORMATIC TEAM:** A suggestion to help out here if this POC is a viable configuration is to allow us to add a 'tables' block or array in the platformatic.db.json file where we can put the tables we want to include in our DB app achema. If present the `schema.lock` file can be generated from the tables in this file. If not present then business as usual.

THAT DOES IT!

Let me know how this works for you if interested and if you run into any problems or have any enahncements or suggestions.