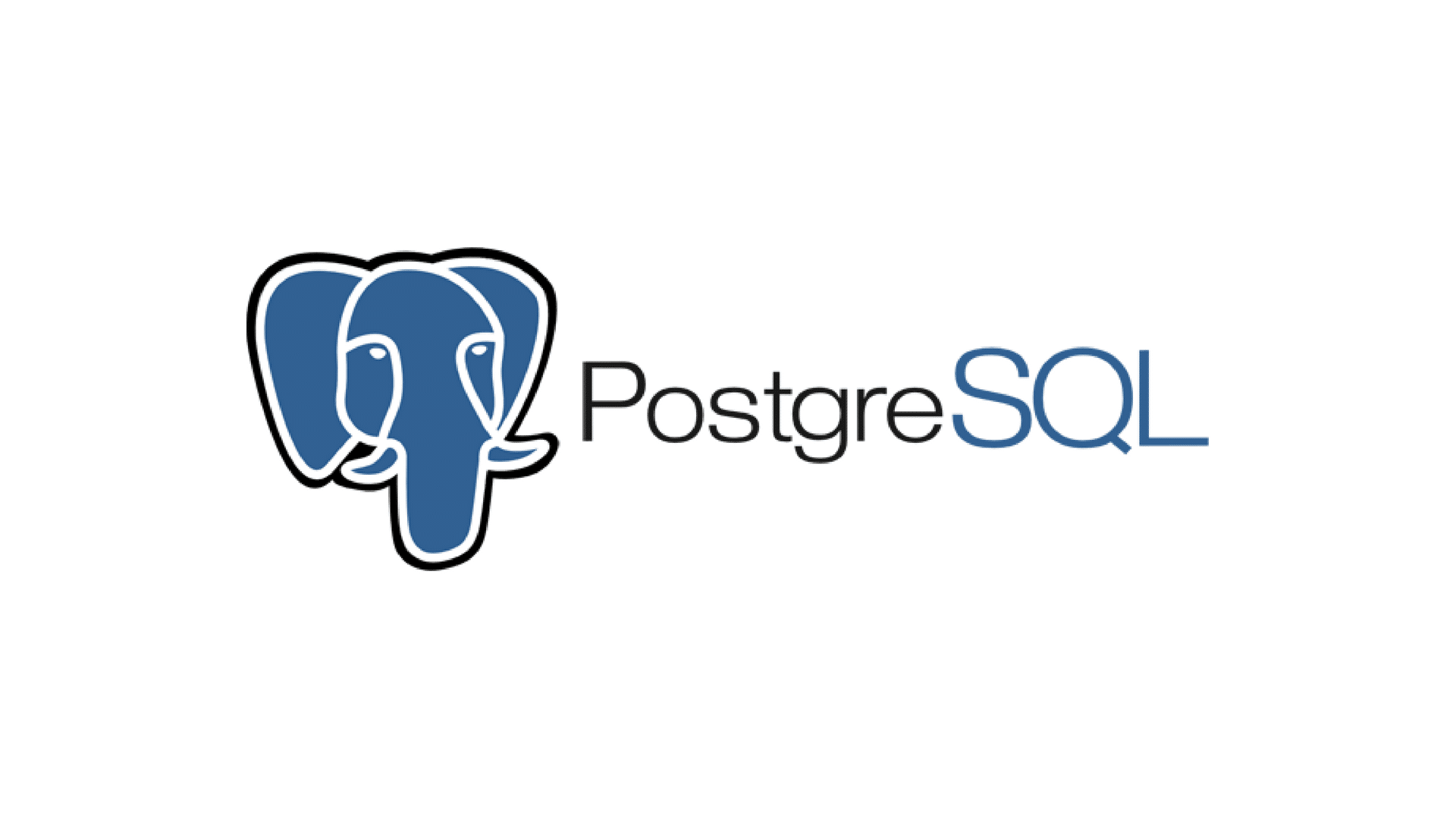
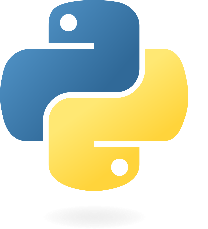
For this case and since we are managing a **high volume of data**, the ideal solution should be focusing on controlling the business structure and repetitive tasks at specific hours by using a relational database and separating the Python code as a Middleware for managing dayparting, inserting expenses, report generation and exposing the data as restful services for consuming frontends.

In my solution, I created a dockerized solution within the same network: **PostgreSQL Database**, a **PgAdmin client** for the DB and a container for the **Python Middleware**

On a real-life scenario and only in case the business is too large the Database should be divided into two clusters: a DB Cluster and a Job Cluster to avoid overloading the operation. Besides that, a microservices architecture should be considered to run the Python instances as separated services in a Kubernetes cluster (either on premise or cloud solutions).

This way, our services can be exposed to providers as Rest applications for frontend microservices to consume. In a real-life scenario, each Python method should be separated as a Microservice (Pod) in a Kubernetes architecture.



Network

1. **Pseudo-code:**
2. **Database Structure:**

**Tables:**

**Tspend:**

ID: Unique spend id

Campaign\_ID: Each spend is associated to a brand

Spend: Cost of the expense

**Tbrand:**

Id: Unique brand id

Name: Brand Name

Daily\_budget: For set up purposes

Monthly\_budget: For set up purposes

Spent\_today: Total spenses for the current day

Spent\_this\_month: Total spenses for the current day

Debt: New field created to manage activation or deactivation of campaigns after surpassing daily or monthly budget

**Tcampaign:**

Id: Unique campaign id

Brand\_id: ID to the owner brand

Status: Active or Not

Start time: To control dayparting

End time: To control dayparting

Activation\_Cost: Assuming there is a needed cost for creating a campaign

**Triggers:**

1. **Trigger for managing expenses and budgets (handle\_spend\_insert):** Every time you add **expenses** associated to a brand this trigger updates **monthly** and **daily expenses** for the brand in table tbrand. After the operation is performed, campaigns will be **deactivated** in case there is no available **budget**, and it will sum the remaining as debt in the table tbrand. This trigger allows to efficiently manage expenses, budget, campaign status and debt the same time. For example, as soon as debt is detected for a given brand all its campaigns will be stopped automatically.
2. **Trigger for managing campaigns (handle\_campaign\_insert):** Every campaign will have an associated starting costs, so there will be a trigger that updates the daily and monthly expenses for the associated brand.
3. **Middleware Structure Functions:**
4. **DB Connection:** Managing DB connection: In a real-life project a new user with limited roles and profiles to avoid DBA or special users to transact against the DB through microservices. For this scenario, **environmental variables** are passed when creating the containers.

**Def db\_conn():**

**User, pass,host,dbname = get env variables**

**Psycopg2.connect(user, pass,host,dbname)**

1. **Get Monthly Budget: Method to list monthly budget for every brand**

**@get**

**Def get\_monthly\_budget():**

**SELECT brand\_name, monthly\_budget FROM tbrand;'**

**Return brand\_name, monthly\_budget**

1. **Reset Spend Today: Method for setting expenses to 0 for current day**

**@post**

**def reset\_spent\_today():**

**brand\_id = select \* from tbrand**

**for each brand\_id:**

**set expend\_today = 0**

1. **Reset Spend Monthly: Method for setting expenses to 0 for current month**

**@post**

**def reset\_spent\_monthly():**

**brand\_id = select \* from tbrand**

**for each brand\_id:**

**set expend\_monthly = 0**

**@post**

1. **Handling dayparting: Method for deactivating campaigns out of running hours**

**Def deactivate\_campaigns\_out\_of\_time():**

**Update campaigns set status=inactive if start\_time <now NAND end\_time > now**

1. **Activates campaigns: if there is no debt for the associated brand**

**@post**

**Def activate\_campaigns\_if\_no\_debt():**

**Select \* from tbrand by id where debt = 0**

**For each campaign:**

**If bran\_id = id**

**Update campaing set status= active**

1. **Registering spends: Method for registering expense. The rest of the budget related logic is manged by the trigger handle\_spend\_insert**

**Def add\_spend(id, brand\_id, spend\_amount,spend\_date):**

**get params from request(id, brand\_id, spend\_amount, spend\_date)**

**if params not found:**

**return error**

**else:**

**insert into tspend(id, brand\_id, spend\_amount, spend\_date)**