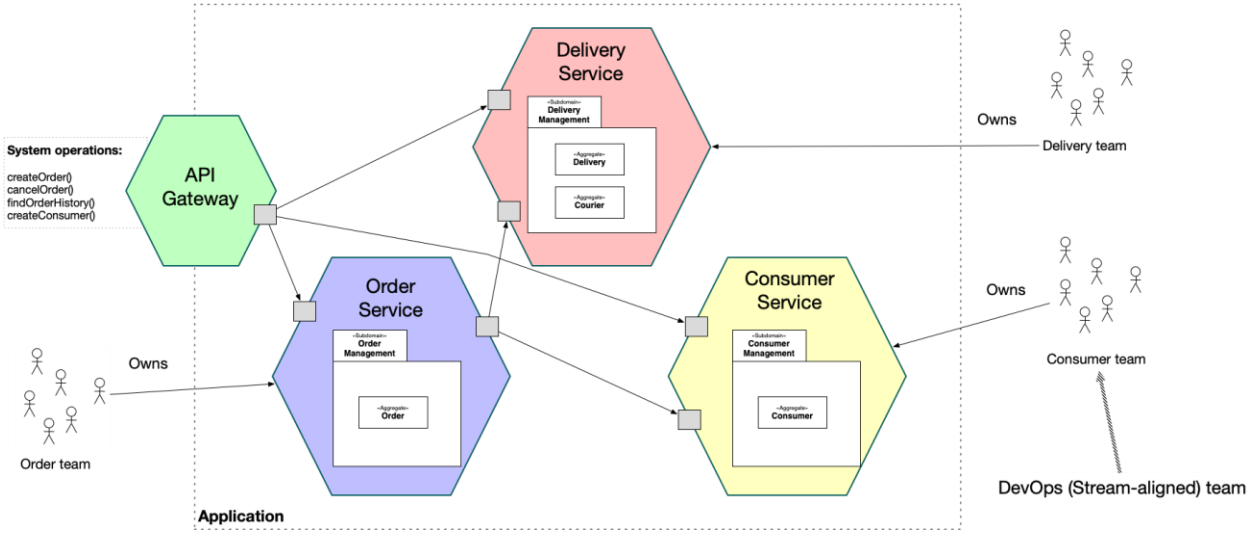
The Pokémon saga continues...

Basic microservice architecture:



In this part of the project, we are going to “break” the monolith to micro services.

1. Create a new service, analogue to the CRUD service created in this tutorial.
   The tutorial is creating containerized CRUD API application using Flask and MongoDB.
   Some points to be aware of:
   a. We are using fastAPI and MySQL DB
   b. The data model in the tutorial is general, but we have a specific data model, which we created for the Pokémon data.
   c. The CRUD API should give all the functionality to the “gateway” CRUD operations.

      Your API should support the following operations:

      1. Add new Pokémon species: adds a new Pokémon species with the following information: id, name, height, weight, types (all of them). You can get the information from the external API or add it yourselves.
      2. Get Pokémons by type: returns all Pokémons with the specific type
      3. Get Pokémons by trainer: get all the Pokémons of a given owner
      4. Get trainers of a Pokémon: get all the trainers of a given Pokémon
      5. delete Pokémon of trainer
      6. Add Pokémon to a trainer: when a trainer catches a Pokémon and train it the Pokémon become his.
      7. Evolve (Pokémon x of trainer y)

      but it doesn't need to overlap - for example Evolve can be implemented as add and delete

d. Pay attention to the connection string URL (you should use the host name you gave in the docker compose file) db_host in the example

```
version: "3"
services:
    app:
        build: .
        depends_on:
            - db_host
        ports:
            - "5001:5001"

    db_host:
        image: mongo:latest
        ports:
            - "27017:27017"
```

e. Pay attention to the port – you cannot have to servers (of any kind) running on the same machine with the same port number)

f. The user should be root without password
you should add in the docker compose file in the db section
environment:
MYSQL_ALLOW_EMPTY_PASSWORD: "yes"

g. Sometimes the mysql server needs time to raise – if you have connection refused problems – add a delay in the docker file:
CMD ["sh", "-c", "sleep 60]

2. Fix your API gateway (the original server) instead of the DB access layer – add a class of proper web requests that will "talk" with the DB.
3. Create a docker file for the fixed server (API gateway) and add to your docker compose.
4. Learn gridFS – how to save file in mongo DB
5. Create another service of a server and mongoDB, that will help us
this service will be used to save Pokémon images in the DB
    a. save images in the DB.
    b. Read images from the db
6. Add the new service to the API gateway

7. Fix the Migration process should now add the relevant images to the mongoDB new service.
   Since we don't use only relational DB (we have two different DB services)
   we need to take care of the consistency of the data, and verify it is whole (for each Pokémon species we need to have an image).
8. Fix external CRUD API and add a new one:
    a. (1) Add new Pokémon species – on top of the implemented functionality should also add the image of the Pokémon to the image db
    b. (7) Evolve - should also add the image of the evoloved Pokémon to the image db
    c. new API - Get Pokémon by species which will return the following:
       id, name, height, weight, types (all of them). And the **Pokémon image**