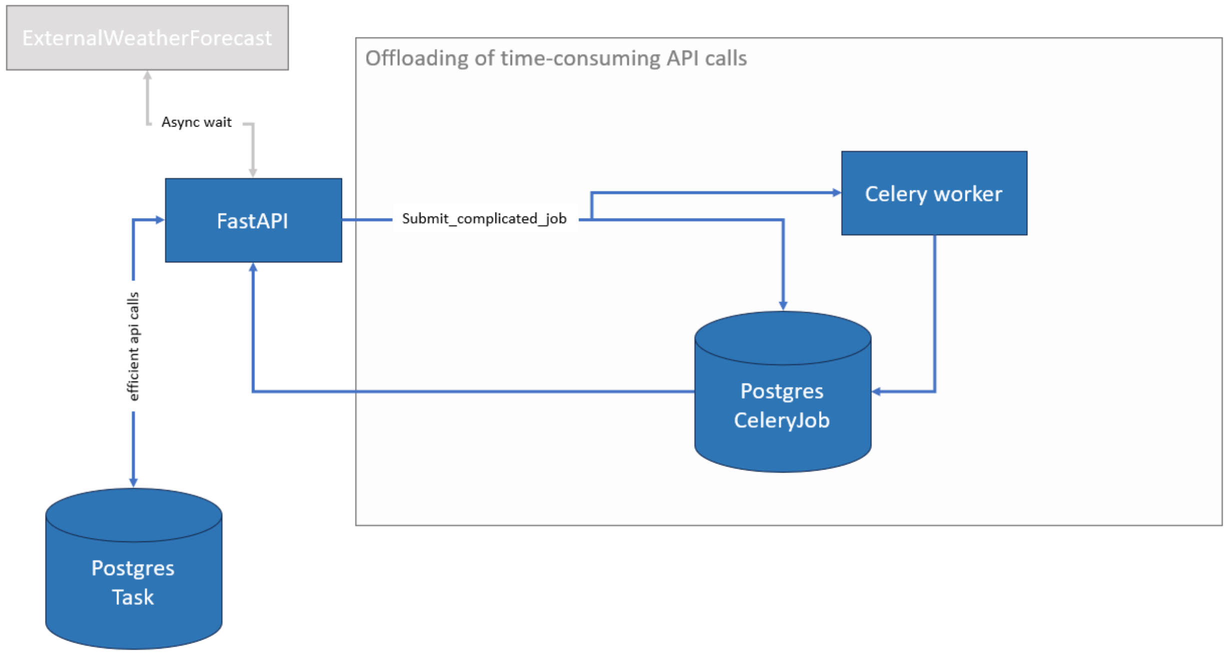
Asynchronous Processing

We don’t want http timeouts, so every task that is expected to have a longer duration than 10 seconds the API Schuld be designed as a “submit-task” type of call, where the result eventually is available in e.g. a database.

Typical API call:

Submit_complicated_job(params) -> 202



Notifications

Since we have the celery tasks in previous section, I will use it as an example here as well. The status of the task in the CeleryJob database will be changed by the celery worker to “Completed”. The frontend (or the client) keeps polling. Separate endpoint to be designed for returning status of task.

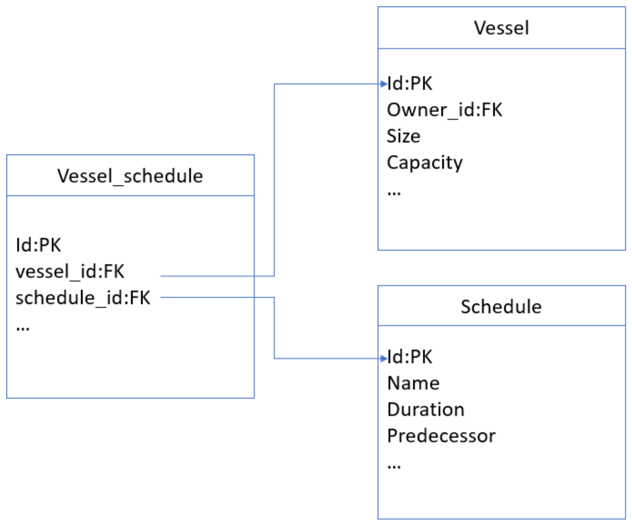
Data & modeling relations

Many to many relationships between multiple vessels and multiple schedules can be handled by separate relation table in this case named vessel_schedule, where vessel_id and schedule_id are foreign keys.

API calls that can be implemented:

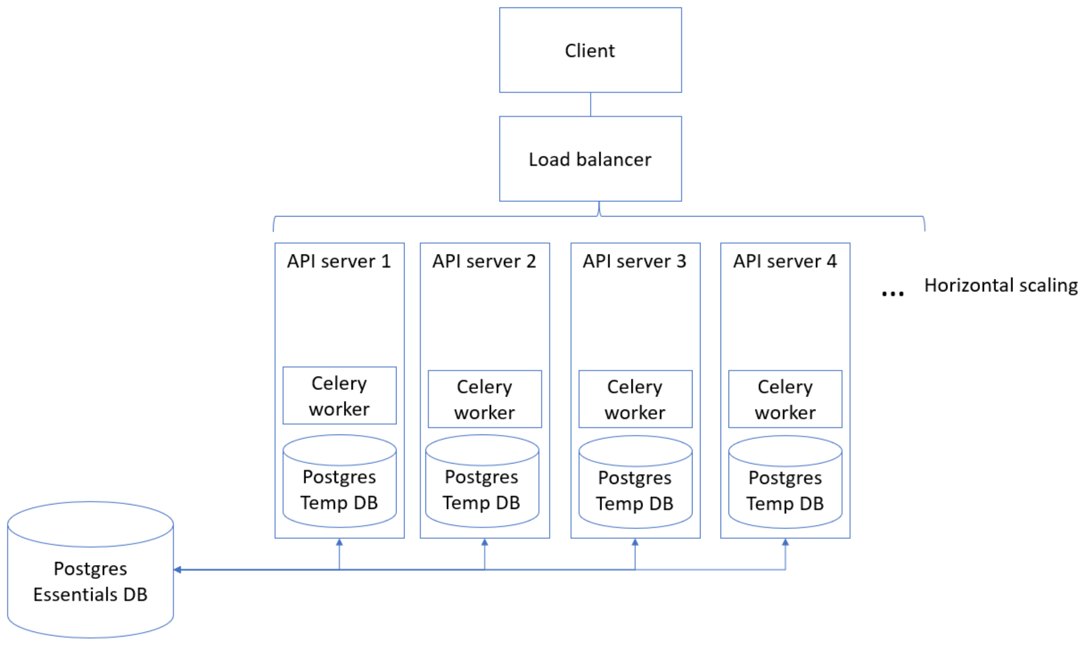
get all schedules associated with a certain vessel (or selection of vessels)

get all vessels associated with a certain schedule (or selection of schedules)



Scalability & robustness

- A load balancer distributes the api calls to api servers, and increase number of servers automatically if required
  - Makes the underlying servers appear as one single for the client
  - Pick one of the standard cloud solutions (AWS ELB, Google Cloud LB, Azzure application gateway)
- In horizontal scaling, each server instance must operate independently of the others — in other words, the service must be stateless.
  - Hence state-dependent information on external database (essentials DB)
- Challenges
  - Licensing cost/acquiring process of OrcaFlex license if a license is needed for every API server
    - Offload to external numerical cluster, Taylor?



Development Environment

One way could be to have a docker-compose file for local development that can be configured where localhost can be defined in environmental variables instead of production address.