terça-feira, 1 de abril de 2025 07:32

- CppRestSDK is not an option anymore, no support for years and the github page recommend not to use it
   Pistache

- May lack documentation.... but it's something I can contribute... they are looking for voluntiers
  - Ceems like a good option, has support (github page s active) <a href="https://github.com/CrowCpp/Crow">https://github.com/CrowCpp/Crow</a>
    Looks like there are some documentation arround it <a href="https://crowcpp.org/master/getting\_started/setup/linux/">https://crowcpp.org/master/getting\_started/setup/linux/</a>
    GPT gave a solid answer... that is nice

### 2 Structuring the C++ API Project

A well-structured project makes it easier to maintain and expand. Below is the recommended folder structure:



#### 3. Running the API in Development Mode

Now, you can run your API without rebuilding the image every time.

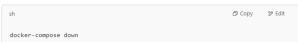
#### Start the API

sh	₫ Сору	≫ Edit
docker-compose upbuild		

#### This will-

- Build the container (if needed).
- Mount your local source code inside the container.
- Compile the code ( cmake . && make ).
- Run the API.

#### Stop the API



### **Recompile Without Restarting**

If you change your C++ code, you don't need to restart Docker. Instead, inside the running container, recompile:

```
docker exec -it cpp-api-dev bash # Open a shell inside the container
cmake . && make
                               # Recompile the API
./api_server
                                # Restart it
```

#### 🔥 Summary: Fast Development Workflow

Action	Command
Start API (auto-recompiling)	docker-compose upbuild
Stop API	docker-compose down
Recompile code inside container	docker exec -it cpp-api-dev bash $\rightarrow$ cmake . && make && ./api_server
Restart API after code changes	./run_dev.sh

### 1 Build & Start Container

docker-compose up --build

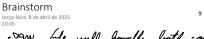
#### 2 Manually Rebuild (if needed)

docker exec -it cpp-api-dev bash cd build && cmake .. && make -j\$(nproc) ./api\_server # Run manually

# 3 Stop Container

docker-compose down

## Definição





Just a draft

#### 1. Development Dockerfile (docker/dev.Dockerfile)

Dockerfile

Dockerfile
CopyEdit
FROM ubuntu:22.04
# Install build dependencies
RUN apt\_eet update && apt\_get install -y\
g++ cmake git libboost-all-dev libasio-dev\
&& rm -ff /xar/libpat/rlists/\*
# Install Crow
RUN git clone --branch v1.2.1.2 https://sithub.com/CrowCpp/Crow.git /Crow\
&& cd /Crow && mkdir build && cd build\
&& cmake ... -DBUILD\_SHARED\_UBS=ON && make -jS(nproc) && make install

Described to include # 3. Instalar GoogleTest e GoogleMock

# 3. Instalar Google lest e GoogleMock RUN git clone https://github.com/google/googletest.git /tmn/poopletest && \

```
- src/
   └─ main.cpp
   build/
                               # Ignored in .gitignore, generated during bu

    dockerignore

L TEST L MY-TEST-CAP
```

```
yaml
                                                          🗗 Сору
                                                                  ⁰ Edit
     dockerfile: docker/dev.Dockerfile
    container_name: cpp-api-dev
   working_dir: /app
   ports:
      - "8080:8080"
     bash -c "mkdir -p build && cd build && cmake .. && make -j$(nproc) &&
     dockerfile: docker/prod.Dockerfile
    container_name: cpp-api-prod
      - "8080:8080"
    depends_on:
```

g++ cmake git libboost-all-lev libasio-dev\
&& rm -rf /var/lib/apt/lists/\*
# install Crow
RUN git clone -branch v1.2.1.2 https://github.com/CrowCpp/Crow.git /Crow\
&& cd /Crow && mkdir build && cd build\
&& cmake .- DBUILD\_SHARED\_LIBS=ON && make -j\$(nproc) && make install WORKDIR/app

Dockerfile CopyEdit FROM ubuntu:22.04 FROM ubuntu:22.04
# Install runtime-only dependencies
RUN apt-get update && apt-get install -y \
libboost-all-dev libasio-dev \
&& rm -ff /var/lib/apt/lists/\*
COPY build/api\_server /app/api\_server
WORKDIR /app
EXPOSE 8080

# 3. Instalar GoogleTest e GoogleMock RUN git clone nun gir cione
https://github.com/google/googletest.git
/tmp/googletest && \
cd /tmp/googletest && \
cmake - 5. - B build && \
cmake - -build build - j (Inproc) && \
cmake --install build && \
cmake --install build && \ rm -rf /tmp/googletest

יי שמשים --

#### 2. Production Dockerfile (docker/prod.Dockerfile)

CMD ["./api\_server"]

# 4. Development Workflow

**Run Dev Environment:** docker-compose up --build ☐ Copy 🍪 Edit bash docker-compose up cpp-dev

#### Recompile after code change (automatically with volume):

- Change code in src/main.cpp
- Inside the container ( cpp-dev ), rerun:

© Copy ७ Edit bash cd build && make

docker-compose up --build

# 5. Running the API (Dev or Prod)

• From host (Windows or WSL), call:

bash ☐ Copy 🍪 Edit curl -X POST http://localhost:8080/api/analyze -d '{"code":"class A {}"}'

#### Or test with Postman:

- POST URL: http://localhost:8080/api/analyze
- Body (JSON):

```
⁰ Edit
 "code": "class A {}"
}
```



#### Summary

Goal	Solution
Dev on Linux	Use cpp-dev container with volume mapping
Run API	Auto-run in dev or manually in cpp-api
CI/CD	GitHub Actions with docker build
Installer	Use cpack or makeself after build

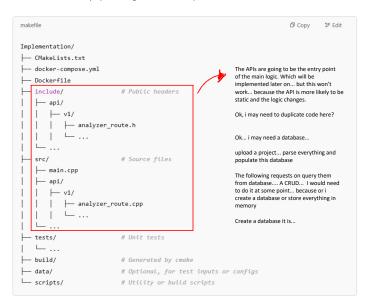
- osar o tunider cuent ou positifair?
  Pensar no versionamento das apis e como testar elas

  Rodar uma bateria de testes na API a cada versão

  Uma aplicação que comunca com uma API e faz avaliação das respostas eseradas
- Camada de segurança ( não vou preocupar com isso agora
- Permito aquela origem
   Reebe um endereço no github, baixa os arquivos e processa eles

#### Notes to consider developing the API

This folder structure will be very important to organize version of the implementations!!!!!



#### How to create APIs with crow in c++

- Recommended Folder Structure

  Follow the folder Structure for includes and source files

  Inside the V1 folder, organize by domains

  Analyze domain, database domain, retreve information domain....

  Looks like the API structure is like lambdas functions

  This is saying: "when someone does a GET / Status, run this small anonymous function and return the result."

  We need to define the route and the HTTP method required to be used there

  ROW (ROUTE fann. "\frac{\text{Status}}{\text{Status}}") methods (rouge: \text{MTPMethod} : GFT)
- CROW\_ROUTE(app, "/status").methods(crow::HTTP Why put the implementation in a .h file?
- in Crow, we use .h for route setup
  - . The lambdas are templates and must be visible during compilation. If you split them between .cpp and .h, the compiler won't know the lambda's exact type in another compilation unit.
- Town itself is mostly a header-only library (no.so or .a files to link). Your handlers behave better when defined the same way.
   You can use classes to group APIs by domain:
   organizing routes by domain in classes is a best practice to keep code clean and modular.
  - You group related endpoints (e.g., user, Jauth, /product) in their own files.
    Each class encapsulates its setup.
    Easier to reuse shared logic inside those classes
    We can add to the other folder and call the implementation, there is na example ->>>>

```
срр
#pragma once
#include <crow.h>
#include "../services/CalculatorService.h"
class CalculatorRoutes {
    static void init(crow::SimpleApp& app) {
        CROW_ROUTE(app, "/v1/calc/add").methods("POST"_method)
        ([](const crow::request& req) {
            auto body = crow::json::load(req.body);
           if (!body || !body.has("a") || !body.has("b"))
                return crow::response(400, "Missing param
            int a = body["a"].i();
            int b = body["b"].i();
            int result = CalculatorService::add(a, b);
            crow::json::wvalue res;
            res["result"] = result;
            return crow::response(res);
        });
};
```

```
#include <crow.h>
#include "routes/CalculatorRoutes.h"
int main() {
   crow::SimpleApp app;
   CalculatorRoutes::init(app);
   app.port(8080).multithreaded().run();
```

```
Istart on API
Setup (Basic Crow App)
                                                                           server
 #include "crow_all.h"
    main() {
crow::SimpleApp app;
 int main() {
     // Define routes her
     app.port(18880).multithreaded().run(); - Tun +6 opp
  GET - Fetch data
   Example: Basic route
    CROW_ROUTE(app, "/hello")
       return "Hello, Crow!";
    });
   Example: With parameters
                                               & Poronates defuntion flat com
                                                be used a +6 code bre
    CROW_ROUTE(app, "/user/<int>")
    ([](int userId){
       (int userId){
return "User ID: " + std::to_string(userId); a Cambolo function
    });
     POST - Submit data (like a form or JSON body)
                                              p specify that it's a post command
     Example: Receiving JSON
     CROW_ROUTE(app, "/user").methods("POST"_method)

([](const crow::requesta req){

auto body - crow::json::load(req);

if (Ibody) return crow::response(400, "Invalid J50N");

Hendle exceptions
        std::string name - body["name"].s(); } thus the grow info int age - body["age"].i(); } the respective type
         return crow::response("Received name: " + name + ", age: " + std::to_string(age
     Request Body (from Postman or curl):
                                                             © Copy ٷ Edit
      json
       "name": "Hugo", upul structure
"age": 30
  PUT – Update something
                                                            ord yest went to dange
something on t
   CROW_ROUTE(app, "/user/<int>").methods("PUT"_method)
   ([](const crow::request& req, int userId){
       auto body = crow::json::load(req);
       if (!body) return crow::response(400, "Invalid JSON");
       std::string newName = body["name"].s();
       return crow::response("User " + std::to_string(userId) + "renamed to " + new
   });
     ✓ DELETE – Remove something
                                                              ⊕ Copy % E
```

CROW ROUTE(app, "/user/<int>").methods("DELETE" method)

return crow::response("User " + std::to\_string(userId) + " deleted.");

([](int userId){

// delete user with ID

});

# ✓ DELETE – Remove something

## ✓ Sending JSON responses

```
CROW_ROUTE(app, "/json")
([](){
    crow::json::wvalue result;
    result["status"] = "ok";
    result["version"] = 1.0;
    return crow::response(result);
});
```

# Bonus: Reading query params

```
CROW_ROUTE(app, "/search")
([](const crow::request& req){
   auto query = crow::query_string(req.url_params);
   std::string q = query.get("q") ? query.get("q") : "default";
   return "You searched for: " + q;
});
```