

- 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 volunteers
- Crow
  - Seems like a good option, has support ( github page s active ) <https://github.com/CrowCpp/Crow>
  - Looks like there are some documentation around it [https://crowcpp.org/master/getting\\_started/setup/linux/](https://crowcpp.org/master/getting_started/setup/linux/)
  - 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:

```
bash
/c++-api
├── /src
│   ├── main.cpp      # Entry point of the API
│   ├── handler.cpp    # Logic to process requests
│   ├── handler.h      # Header file for the handler
│   ├── CMakeLists.txt # CMake build configuration
│   ├── Dockerfile     # Instructions to containerize the API
│   ├── .dockerignore  # Ignore unnecessary files in Docker
│   └── README.md      # Documentation
```

3. Running the API in Development Mode

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

Start the API

```
sh
docker-compose up --build
```

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

```
sh
docker-compose down
```

Recompile Without Restarting

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

```
sh
docker exec -it c++-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)	<code>docker-compose up --build</code>
Stop API	<code>docker-compose down</code>
Recompile code inside container	<code>docker exec -it c++-api-dev bash → cmake . &amp;&amp; make &amp;&amp; ./api_server</code>
Restart API after code changes	<code>./run_dev.sh</code>

1 Build & Start Container

`docker-compose up --build`

2 Manually Rebuild (if needed)

`docker exec -it c++-api-dev bash`  
`cd build && cmake . && make -j$(nproc)`  
`./api_server` # Run manually

3 Stop Container

`docker-compose down`

- Definição
- Rotas
  - Entidades
  - Recursos
  - Domínios

Brainstorm

*API file will handle both containers*

```
Implementation/
├── docker/
│   ├── dev.Dockerfile      # Image for development (build tools, live c
│   └── prod.Dockerfile     # Image for production (run only)
├── docker-compose.yml     # Multi-container setup (dev + prod)
├── CMakeLists.txt
├── src/
│   ├── main.cpp
├── build/                  # Ignored in .gitignore, generated during bu
├── .dockerignore
└── ...
```

*Just a draft*

1. Development Dockerfile (docker/dev.Dockerfile)

```
Dockerfile
CopyEdit
FROM ubuntu:22.04
# Install build dependencies
RUN apt-get update && apt-get install -y \
    g++ cmake git libboost-all-dev 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
```

*Also need to include G-tests in the installation*

# 3. Instalar GoogleTest e GoogleMock  
RUN git clone <https://github.com/google/googletest.git> /tmp/gtest && \

```
└─ src/
└─   └─ main.cpp
└─ build/
└─ .dockerignore
└─ TEST
└─ my-test.cpp
```

```
g++ cmake git libboost-all-dev 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
```

```
# 3. Instalar GoogleTest e GoogleMock
RUN git clone
https://github.com/google/googletest.git
cd /tmp/googletest && \
cmake -S -B build && \
cmake --build build -j$(nproc) && \
cmake --install build && \
rm -rf /tmp/googletest
```

```
yaml Copy Edit

version: "3.8"
services:
  cpp-dev:
    build:
      context: .
      dockerfile: docker/dev.Dockerfile
      container_name: cpp-api-dev
    volumes:
      - ./app
    working_dir: /app
    ports:
      - "8080:8080"
    command: >
      bash -c "mkdir -p build && cd build && cmake .. && make -j$(nproc) && ..."

  cpp-api:
    build:
      context: .
      dockerfile: docker/prod.Dockerfile
      container_name: cpp-api-prod
    ports:
      - "8080:8080"
    depends_on:
      - cpp-dev
```

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

```
Dockerfile
CopyEdit
FROM ubuntu:22.04
# Install runtime-only dependencies
RUN apt-get update && apt-get install -y \
  libboost-all-dev libasio-dev \
  && rm -rf /var/lib/apt/lists/*
COPY build/api_server /app/api_server
WORKDIR /app
EXPOSE 8080
CMD ["/api_server"]
```

## 4. Development Workflow

### Run Dev Environment:

docker-compose up --build

```
bash Copy Edit

docker-compose up cpp-dev
```

Run tests manually in dev container:

cd build  
ctest --output-on-failure

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

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

```
bash Copy Edit

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):**

```
json Copy Edit

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

## ✓ Summary

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

- Usar o tunder client ou postman?
- Pensar no versionamento das apis e como testar elas
  - Rodar uma bateria de testes na API a cada versão
    - Uma aplicação que comunica com uma API e faz avaliação das respostas eseradas
- Camada de segurança ( não vou preocupar com isso agora
  - Permitir aquela origem
- Receber um endereço no github, baixar os arquivos e processar eles