Container and C++

quinta-feira, 7 de novembro de 2024 20:21

Step 1: Understanding Dockerfile Commands

The Dockerfile defines how to create your Docker image, setting up the environment you need to compile and run your C++20 project.

1. Base Image Selection

dockerfile
FROM registry.access.redhat.com/ubi8/ubi:latest

- **Explanation**: FROM specifies the base image to build on. Here, ubi8 is Red Hat's Universal Base Image (UBI) version 8. This is essentially a minimal Linux operating system that Docker uses to start your environment. By using ubi:latest, you get the latest stable UBI version.

changed to 8.10-1132

2. Installing Development Tools

```
RUN apt-get update && \
  apt-get install -y build-essential cmake gcc g++ && \
  apt-get clean
```

- **RUN apt-get update**: apt-get is the package manager for Ubuntu. Running apt-get update updates the package list inside the container, allowing it to locate and install the latest versions of any software we specify.
- **apt-get install -y build-essential cmake gcc g++**:
  - **-y**: This flag automatically answers "yes" to any prompts during installation, ensuring the process is non-interactive.
  - **build-essential**: This package provides a suite of essential tools for building software on Ubuntu (compilers, libraries, and tools).
  - **cmake**: Installs CMake, which is a tool to manage the build process of C++ projects.
  - **gcc g++**: Installs the GCC and G++ compilers for C and C++.
- **apt-get clean**: This removes any cached packages that were downloaded but are no longer needed, which helps reduce the final size of the container.

Don't want this
Required

3. **Set env variables for c++20**

```
ENV CC=gcc
ENV CXX=g++
```

- **ENV**: Sets environment variables inside the container.
  - **CC=gcc**: Sets the CC environment variable to use gcc as the default C compiler.
  - **CXX=g++**: Sets the CXX environment variable to use g++ as the default C++ compiler.

4. **Create and set the working directory for the application**

```
WORKDIR /src
COPY ./src /src
```

Set the working directory and copy all the files there

5. **Run CMake to configure and build the project**

```
RUN cmake -Bbuild -H. && \
  cmake --build build
```

- **cmake -Bbuild -H.**:
  - **cmake**: Runs CMake, the build system.
  - **-Bbuild**: Specifies the output directory for build files. It will create a build folder within /app.
  - **-H.**: Sets the project source directory as the current location (.).
- **cmake --build build**: Compiles the project using CMake with the generated build files in the build folder.

6. **Default command to run your application**

```
CMD [ "tail", "-f", "/dev/null" ]
```

tail -f /dev/null is a common trick to keep the container running indefinitely. tail -f is a command that outputs the end of a file and keeps running until manually stopped. When pointing it to /dev/null, which is an empty, endless file, the process will run forever, effectively keeping the container open.