Project Overview

Name: Tauri application builder
Description: Tauri application builder project aims to create a Dockerfile that compiles Tauri lightweight web apps into Windows executables.

Project requirements

Create a watertight "recipe" in the form of Dockerfile that can easily compile Tauri lightweight apps.

1. The build process should be done in isolated Docker environment
2. Docker container host OS during the build process is not limited to Windows
3. The output of build process should be Windows executable file

The project's primary goal is to create a watertight "recipe" for building lightweight web app clients that can be easily compiled using a Dockerfile. Builded container should be the factory that produces the executable file from Tauti application source code.

Objectives

Task is to compose Dockerfile that builds a Tauri application executable for Windows using project's source files. The project build process should be done locally, without using online services compilation tools or assuming that the host OS is Windows.

Deliverables

- A Dockerfile for compiling the lightweight web apps using Tauri framework
- Documentation on how to build the "factory" container and build a Tauri example application using that "factory" container.
- The source code for the project

Functional requirements

The system should be able to compile a Tauri lightweight web application executable for Windows using a Dockerfile despite the host OS.

Non-functional requirements

- The Tauri application compilation process should be done in an isolated container environment.
- The build process should be done locally and not assume that the host OS is Windows.

Design

Dockerfile produces an image based on the **rust:alpine** base image. The build tools, toolchain, cross-compiler and targets for compiling Tauri applications for the **windows_x86_64** platform are

installed during the image build process. The actual Tauri application compilation is done during container execution.

The interaction of the host OS with the container will occur through the shared volume mechanism. The folder with the source code of the application acts as “input” and the folder with the compilation result as “output”.

Testing

Will need to test the Dockerfile build image process alongside the executable compilation process using the builded container.

Build (image) and compilation (executable) tests will be performed on Windows, WSL, native Linux and macOS to ensure that there will be no issues.

Worklog

I need to pack all the steps for setting up the Rust cargo build environment and the compilation process of the executable file from the source code of the Tauri application into a Dockerfile.

The main difficulty is setting up a cross platform toolchain for compiling inside an isolated container environment. Compilation process itself is trivial.

Since I settled on the GNU MinGW compiler, I need to pre-install corresponding Rust build toolchain and target before starting the actual Tauri application compilation process.
After that, the build environment for Tauri application compilation will be encapsulated inside the image. That compilation environment will be used during container execution and will compile Tauri Tauri executable like any other Rust project.

Known limitations

Compiled Tauri executable depends on a binary library **WebView2Loader** from the WebView2 SDK. During the executable run you may encounter the following error: “The code execution cannot proceed because WebView2Loader.dll was not found”. This is due to GNU instead of MSVC compiler use. To fix that issue, put the WebView2Loader.dll file alongside the compiled exe file.

It is possible to ADD that missing DLL file during container build, but it is more like a workaround for problem solving.

To build an image Docker **linux** containers should be used because Rust docker images do not have windows/amd64 support.

Testing

Windows 11 (x64) environment is already tested by using it as a development environment. I've done it intentionally, to exclude possible cross-platform compilation issues, so the MSVC compiler was used.

Due to the high complexity of the build process, this took some time on my old machine. At this time I'am writing the document with "Initial thoughts" for other projects.

	Windows 11 (HDD)	Windows 11 (SSD)	WSL2	macOS Big Sur	Ubuntu 18.04
Image build time	71.1s	69.4s	67.7s	168.7s	78.6s
Compilation time	19m08s	11m48s	8m32s	48m10s	11m38s

Windows 11 and WSL2 are on one machine (4790K@4.8Ghz + 32GB) and macOS and Ubuntu are on another (MacBook Pro, late 2013, dual-core i5@2.4Ghz + 8GB). Windows tests are taken when files are located either on HDD or SSD (NVME). WSL2 filesystem is located on the same SSD.

Linux based systems showed the best performance compared to Windows and macOS. i5 with 2x2.4 Ghz cores on linux beats 8x4.8 Ghz on Windows.

Improvements

- Should be possible to pre-configure rust build targets during image build using Docker container arguments. Possible workaround is to install all possible Rust toolchains and targets, but it's costly in terms of final image size and its build times.
- BAT/BASH/Python scripts - the convenient way to to use that image