Project Configuration

Use of a Package Manager

Introducción

- Hasta ahora habíamos utilizado el sistema AIO de Visual Studio
- Gestión de dependencias manual
- ¡Existen otras alternativas!

Como pasamos de un cpp a un ejecutable

- Build System “entrega” los .cpp y .h al compilador + configuraciones
- Compilador convierte cada .cpp en un .o / .obj (ensamblador)
- Linker unifica los obj y resuelve las funciones externas (librerias / otros cpp)
- Si todo ha ido bien tendremos un ejecutable!

Como funciona MSBuild

- Proyecto basado en una solución y uno o varios subproyectos (.vcproj)
- La configuración del sistema en archivos xml (editados mediante UI)
- Compilador = MSVC

Como configurar un proyecto en Visual Studio

- Crear una solución con mínimo un proyecto
- Decirle donde puede encontrar la fuente (include directories)
- Decirle el directorio de trabajo (desde donde parten todas las direcciones relativas)
- Por último, decirle al linker donde puede encontrar librerías externas ya compiladas (dynamic libraries (.dll / .so) o static libraries (.lib / .a))

¡Existen Alternativas!

Build Systems

CMake

MSBuild

Build2

SCons

Compilers

MSVC

GCC

LLVM

ZIG

¡Existen Alternativas!

Build Systems

Build2

SCons

Compilers

ZIG

Que es son los Build Systems?

- Son herramientas que automatizan el proceso de construcción de un proyecto.
- Te permiten gestionar directorios, bibliotecas externas, etc.
- Suelen ser muy flexibles.

CMake

- Muy flexible y organizado
- Multiplataforma y open source
- Compatible con la mayoria de IDE’s
- El estándar de la industria
- Más complejo de configurar

Como configurar un proyecto de CMake

- Necesita un archivo llamado **CmakeLists.txt**
- En este se configura todo lo necesario para “compilar” un proyecto
- Utiliza un lenguaje llamado “**CMake Language**” (basado en macros)

Ejemplo de Cmake file

```
cmake_minimum_required(VERSION 3.0)  # Versión mínima de CMake requerida

project(MyProject)  # Nombre del proyecto

# Agrega el archivo fuente main.cpp al proyecto
add_executable(my_executable main.cpp)
```

Vcpkg Dependency Manager

- Gestor de dependencias para C/C++ creado por Microsoft
- Open Source
- Multiplataforma
- Soportado por los IDE’s más populares

Introducción a vcpkg

- Vpkg tiene dos modos: Classic y **Manifest**
- Requiere un archivo vcpkg.json (manualmente o con comandos)
- Se puede utilizar en muchos **build systems** pero nativamente soportado por **Cmake** y **MSBuild.**

https://learn.microsoft.com/ca-es/vcpkg/

Configuración en MSBuild

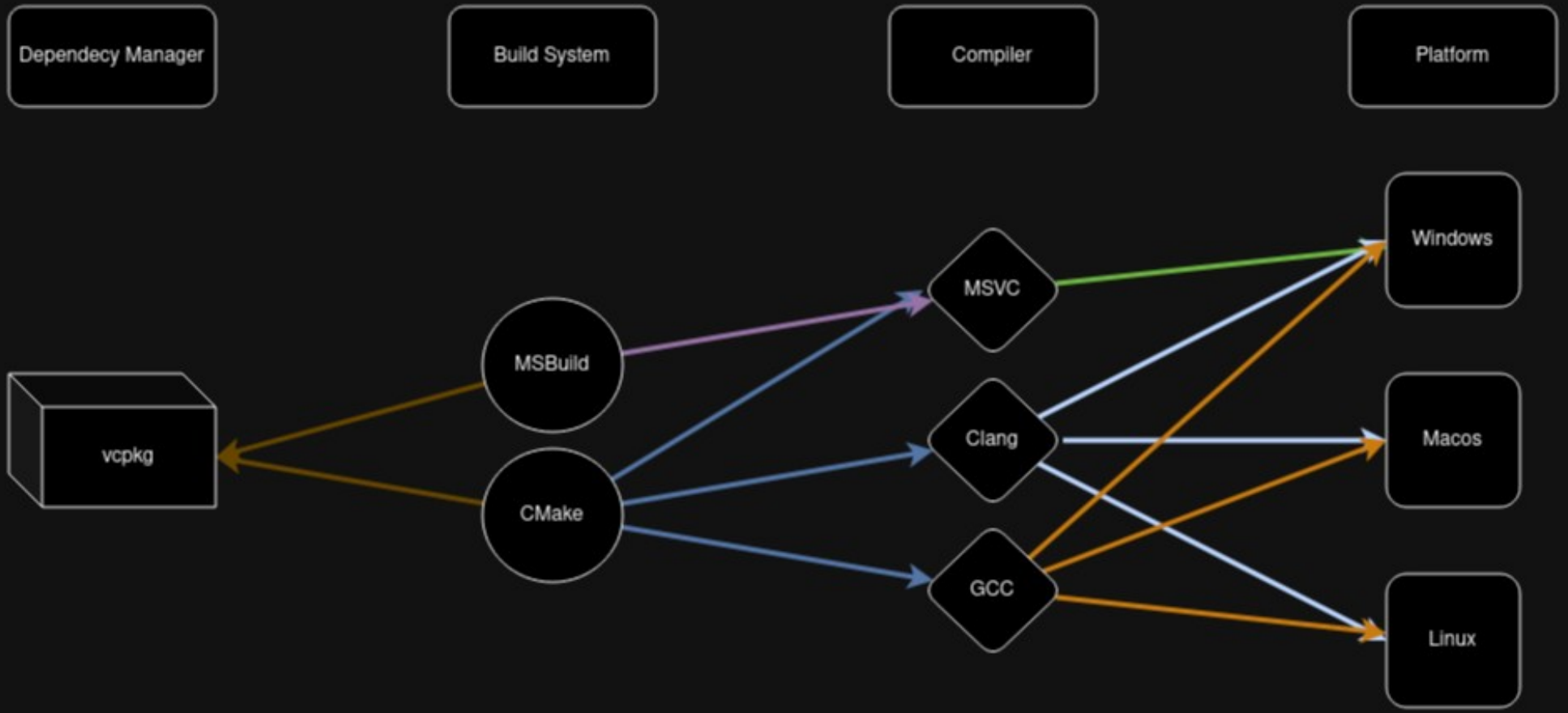
- Abrir la **Developer Command Prompt** o **Developer PowerShell** y introducir *vcpkg integrate install*
- Activar en la config del proyecto *Use Vcpkg Manifest”* a *“Yes”*

https://devblogs.microsoft.com/cppblog/vcpkg-is-now-included-with-visual-studio/

Configuración con **CMake**

- Visual Studio 2022
  - Abrir o crear un proyecto CMake
  - Añadir el vcpkg.json
- Clion
  - Abrir o crear un proyecto
  - Abrir la ventana de vcpkg (nativo)
  - Añadir el vcpkg.json en la pentaña usando el “+”
  - Refrescar proyecto CMake
- Visual Studio Code
  - Tener instalada la extension de CMake
  - Añadir un CMakePresets.json con la toolchain (mas info en el pie de página)
  - Configurar el preset
  - Refrescar el proyecto de CMake

https://learn.microsoft.com/ca-es/vcpkg/get_started/get-started?pivots=shell-cmd


Dependecy Manager
Build System
Compiler
Platform
vcpkg
MSBuild
CMake
MSVC
Clang
GCC
Windows
Macos
Linux

Repositorio de Ejemplo

https://github.com/HugoPlacer/CMake-Vcpkg-Example