Jennyfer Natalia Garcia Tiria
Karoll Daniela Fernandez
Cristian David Buitrago
Johan Schneider Avila

# CONFIGURACIÓN A NUESTRO ENTORNO DE DESARROLLO CON LA HERRAMIENTAS DOCKER Y WSL

1. Verificar el procesador de nuestro sistema preferiblemente que sea Intel.

    - **Procesador Intel:**
      En la BIOS habilitar las virtualizaciones (Intel Virtualization Technology)
    - **Procesador AMD:**
      En la BIOS habilitar las virtualizaciones (Secure Virtual Machine)
      App para obtener información de mi sistema: https://www.cpuid.com/

2. Habilitar Características de Windows:
    - Panel de control
    - Programas
    - Activar o desactivar las características de Windows
    - Habilitar (Plataforma de máquina virtual)
    - Habilitar (Subsistema de Windows para Linux)

**OPCIONAL:**
Habilitar Características por Líneas de Comando (Windows PowerShell)

**Habilitar la característica del WSL**

```
dism.exe /online /enable-feature /featurename:Microsoft-Windows-Subsystem-Linux /all /norestart
```

**Habilitar las características de Máquina Virtual en Windows**

```
dism.exe /online /enable-feature /featurename:VirtualMachinePlatform /all /norestart
```

3. **INSTALAR DOCKER**
   Vamos a la página de Docker: https://www.docker.com/
   **1.** Descargamos Docker
   **2.** Instalamos Docker
   **2.1** Configuración:
   Dejar habilitado (Use wsl 2 instead of Hyper-v)

**3**. Al finalizar la instalación ejecutamos Docker (Habilitar Aceptar términos y Aceptar)
**4.** Aparece una ventana que nos dice que necesitamos instalar WSL 2 (Clic en Cancelar)

**INSTALAR WSL 2**

**1.** Descargar paquete de actualización del kernel de Linux
https://wslstorestorage.blob.core.windows.net/wslblob/wsl_update_x64.msi

**2.** Instalamos el paquete de actualización que descargamos
**3.** Abrimos PowerShell y ejecutamos lo siguiente: wsl --set-default-version 2

...Volvemos a Docker (Abrir Docker)

* Debe estar habilitada la casilla (Use the WSL 2 based engine)

...Volvemos a Wsl 2 (Abrir PowerShell)

**1. Revisamos las opciones de configuración de WSL en:**
https://learn.microsoft.com/en-us/windows/wsl/install

**2. En PowerShell ejecutamos los comandos:**
**a)** wsl --install (Nos mostrará todos los comandos de ayuda debido a que WSL ya se encuentra instalado)
**b)** cls (limpiar pantalla)
**c)** wsl --list – online(Nos mostrará un listado de distribuciones de Linux que se pueden instalar)

**3**. wsl --install -d Ubuntu-20.04 (Se puede escoger cualquier versión de la lista)
**4.** Nos pedirá que ingresemos una clave y una contraseña (con este paso finaliza la instalación de la distro de Linux)

**...Volvemos a Docker**

**1**. Clic en Resources
**2.** Clic en WSL INTEGRATION
**3.** Habilitamos la versión de Linux que acabamos de instalar (Ubuntu-24.04)
**4.** Apply & Restart

**Abrir PowerShell…**

**1.** wsl --list --verbose (ejecutamos este comando que mostrará las versiones instaladas)
**Opcional:**
En caso de que en el anterior listado aparecen versiones de distribuciones de WSL 1, ejecutar el siguiente comando:
**a)** wsl --set-default-version 2

**b)** wsl --list --verbose (para volver a ver la lista anterior)

**Vamos al menú de Windows...**
**1**. Abrimos la consola que se ha creado de la distro de Linux instalada (Ubuntu 20.04 LTS)
**2**. Limpiamos la consola (clear)
**3.** Vamos a la instalación en la página de Laravel (https://laravel.com/docs/10.x/installation#getting-started-on-windows)

**4.** Copiar la siguiente línea y pegarla en la consola de Linux:

curl -s https://laravel.build/example-app | bash (empezara a crearse nuestro proyecto de Laravel)
**5.** Nos pedirá la contraseña que habíamos creado antes (Si todo está ok, nos mostrará unos comandos que nos sugiere Laravel en la documentación)
**6.** cd example-app (ejecutamos este comando para acceder a la carpeta del proyecto)
**7.** ./vendor/bin/sail up (con este comando ejecutamos nuestro proyecto)
**8.** Aparece una ventana solicitando permisos (Permitir acceso)
**9.** Vamos a localhost en nuestro navegador y tendremos nuestra aplicación corriendo...

Abrir el proyecto con VSCode…

1. En la consola de Ubuntu cerramos el proceso con el siguiente comando (Control + C)
2. code . (este comando inicializara nuestro proyecto en VSCode)
3. Confirmamos Autores
4. Opcional (instalar las extensiones recomendadas)

**Instalar Rust**

**1.** Descargue las últimas actualizaciones para la distribución de Ubuntu usando Ubuntu Herramienta de empaquetado avanzada (apt) ejecutando el siguiente comando:

sudo apt update

**2.** Agregue los paquetes necesarios para la distribución de Ubuntu ejecutando el siguiente comando:

sudo apt install --assume-yes git clang curl libssl-dev llvm libudev-dev make protobuf-compiler

**3**. Descargue el programa de instalación de Rustup y utilícelo para instalar Rust para distribución de Ubuntu ejecutando el siguiente comando:

curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh

**4**. Siga las indicaciones que se muestran para continuar con una instalación predeterminada.

**5.** Actualice su shell actual para incluir Cargo ejecutando lo siguiente dominio:

source ~/.cargo/env

**6**. Verifique su instalación ejecutando el siguiente comando:

rustc --version

cargo --version

**Instalar Framework Rocket**

**1.** Para instalar la última versión de Rust, recomendamos utilizar Rustup. Instalar Rustup siguiendo las instrucciones de su sitio web. Una vez instalado Rustup, asegúrese de que la última cadena de herramientas esté instalada ejecutando el comando:

rustup default stable

**2.** ¡Escribamos nuestra primera aplicación Rocket! Comience creando un nuevo archivo binario
Proyecto de carga y cambio al nuevo directorio:

cargo new hello-rocket --bin

cd hello-rocket

**3.** Abrir en Visual Studio Code:

code .

**4.** Ahora, agrega Rocket como dependencia en tu Cargo.toml:

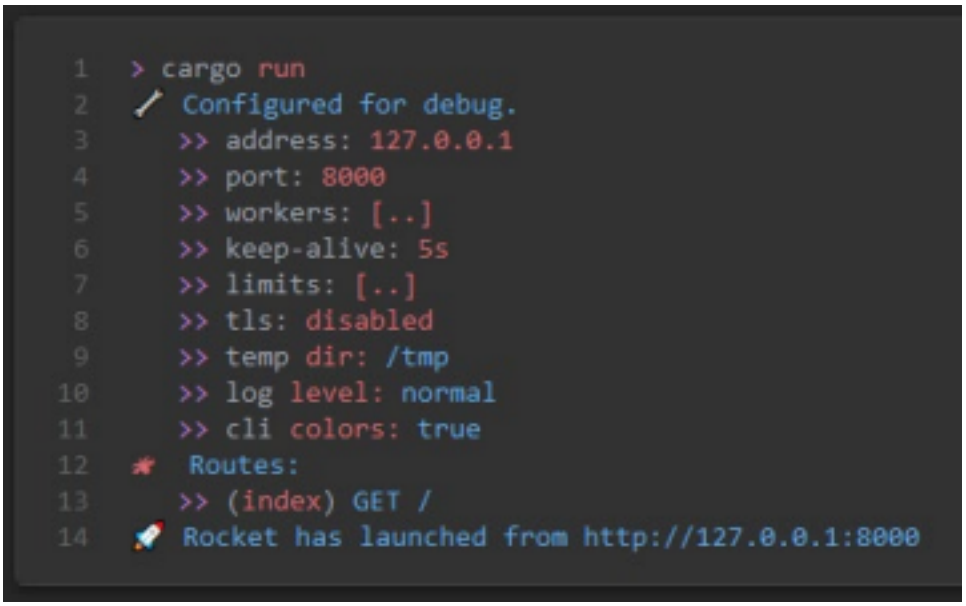
```
[dependencies]
rocket = "0.5.1"
postgres = "0.19"
serde = "1.0"
serde_json = "1.0"
serde_derive = "1.0"
mio = "1"
```

**5.** Modifique src/main.rs para que contenga el código del Rocket Hello, world! programa, reproducido a continuación:

```rust
#[macro_use] extern crate rocket;

#[get("/")]¿
fn index() -> &'static str {
    "Hello, world!"
}
#[launch]
fn rocket() -> _ {
    rocket::build().mount("/", routes![index])
}
```

**6.** No explicaremos exactamente qué hace el programa ahora; eso lo dejamos para el resto de la guía. En resumen, crea una ruta de índice, monta la ruta en /
ruta e inicia la aplicación. Compile y ejecute el programa con cargo run Deberías ver lo siguiente:




**7.** Visite http://localhost:8000 para ver su primera aplicación Rocket en acción

**8.** Creamos el archivo dockerfile y agregamos este código

```dockerfile
# Build stage
FROM rust:1.70.0-buster as builder
WORKDIR /app
```

```
# accept the build argument
ARG DATABASE_URL

ENV DATABASE_URL=$DATABASE_URL

COPY . .

RUN cargo build --release

# Production stage
FROM debian:buster-slim

WORKDIR /usr/local/bin

COPY --from=builder /app/target/release/rocket-project .

CMD ["./rocket-project"]
```

**9.** Creamos el archivo docker-compose.yml y agregamos este código teniendo en cuenta que no puede haber un solo espacio después de cada línea.

```
services:
  rustapp:
    container_name: rustapp
    image: test/rustapp:1.0.0
    build:
      context: .
      dockerfile: Dockerfile
      args:
        DATABASE_URL: postgres://postgres:postgres@db:5432/postgres
    ports:
      - '8000:8000'
    depends_on:
      - dbrust

  dbrust:
    container_name: dbrust
    image: postgres:15
    environment:
      POSTGRES_USER: postgres
      POSTGRES_PASSWORD: postgres
```

```
      POSTGRES_DB: postgres
    ports:
      - '5431:5432'
    volumes:
      - pgdata:/var/lib/postgresql/data

volumes:
  pgdata: {}
```

**10. Crear el archivo .dockerignore**

**/target

**11. Crear el archivo Rocket.toml**

[default]
address = "0.0.0.0"
log_level = "critical"

**12.** Luego ejecutamos el comando

docker compose up -d dbrust
docker compose up rustapp

Estos comandos nos crean los contenedores para los servicios dbrust y rustapp que están en el archivo docker-compose.yml