



Contenedores para Desarrollo de Software

Diseño y Programación de Software Multiplataforma DPS104
Docente: Alexander Alberto Siguenza Campos




Integrantes

Yensy Alejandra Cruz Barahona
Diego Fernando Mancía Hernández
Johan Anthony Menjivar Girón
Javier Ernesto Pérez Joaquín
Alinson Javier Meléndez Torres

CB121442
MH212532
MG182330
PJ211152
MT191530

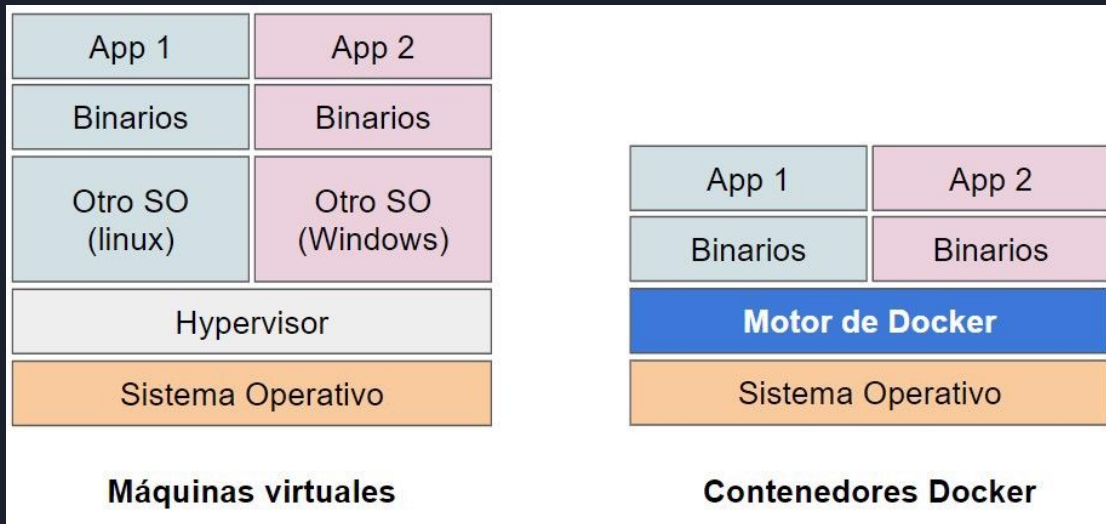
¿Qué son los
contenedores?





Un contenedor es una forma de virtualización del sistema operativo pensado para poder ejecutar cualquier cosa como un microservicio, un proceso de software (ej. API REST) o una aplicación de mayor tamaño (ej. motor de base de datos).

Además, es mucha más ligero que una máquina virtual ya que cada contenedor no necesitan tener una imagen del SO para poder funcionar.

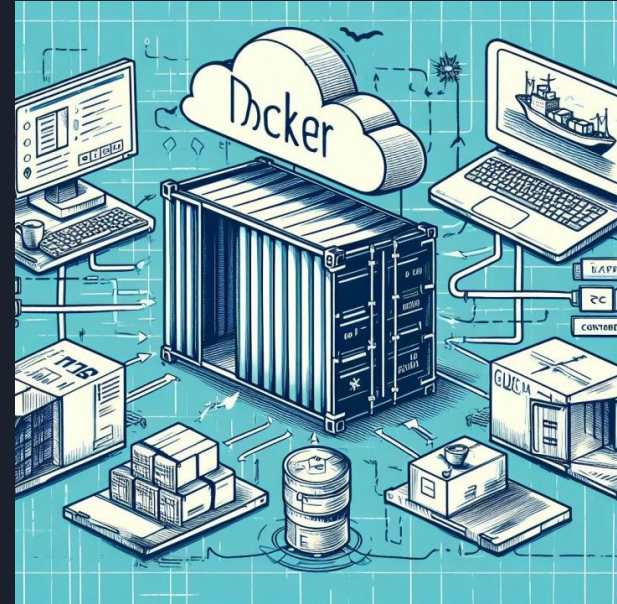


Funcionamiento de los contenedores



Los contenedores operan mediante la virtualización a nivel del sistema operativo, usando:

- **Aislamiento de procesos:** Cada contenedor tiene su propio espacio de usuario, sin interferir con otros.
- **Namespaces:** Aíslan recursos (sistema de archivos, redes, usuarios) entre contenedores.
- **Cgroups:** Controlan y limitan el uso de recursos (CPU, memoria, ancho de banda) para cada contenedor.
- **Sistema de archivos por capas:** Permite compartir capas comunes entre contenedores, ahorrando espacio y haciéndolos ligeros.



Ventajas de utilizar contenedores



Beneficios

- **Portabilidad:** Al encapsular la aplicación junto con sus dependencias en un contenedor, se tiene una única instancia con lo necesario para ejecutarla.
- **Eficiencia:** Al ser más ligeras que una máquina virtual se pueden tener varias instancias de un contenedor en un servidor consumiendo menos recursos.
- **Consistencia:** El contenedor garantiza que la aplicación se ejecutará de manera consistente en diferentes entornos para evitar el conocido “en mi maquina funciona”.



docker



Áreas que se benefician

- **Desarrollo:** Gracias a la portabilidad y consistencia de los contenedores se puede ejecutar la aplicación en un ambiente local pero similar al ambiente productivo, logrando asegurar que la aplicación se ejecutará sin problemas.
- **Pruebas:** Similar a la facilitación en el desarrollo se puede tener un ambiente de pruebas similar al ambiente productivo para realizar un testeo exhaustivo, así como, crear pruebas automatizadas (ej. con Gherkin).
- **Despliegue:** Los contenedores facilitan desplegar una aplicación a un ambiente productivo, ya que todo lo necesario para ejecutar la aplicación ya se encuentra en una única instancia. Además, gestionar diferentes contenedores de una misma imagen por medio de una herramienta facilita manejar un balanceo de carga y evitar que el sitio tenga cuellos de botella.

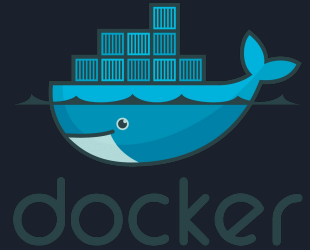
Herramientas populares en el ecosistema de contenedores






Kubernetes: Es el motor para orquestar contenedores más popular que existe, es de código abierto que se utiliza para automatizar la implementación, el escalado y la administración de aplicaciones en contenedores.

Docker: Proporcionando una manera sencilla para ejecutar su código, Docker es el sistema operativo para contenedores más popular, de manera similar que una máquina virtual podemos crear imágenes de este sistema tanto local como en la nube.

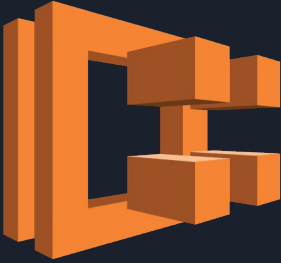




HashiCorp Nomad: Es una herramienta capaz de planificar y desplegar todas las aplicaciones dentro de contenedores, es conocido por la masividad de acciones que este puede realizar, conocido por la gran cantidad de despliegues que es capaz de hacer en poco tiempo.



Portainer.io: Es una interfaz gráfica que permite la gestión de servicios como Docker o Kubernetes de manera visual y más interactiva, permite administrar los recursos desde una interfaz web.




AWS ECS

Amazon ECS: Es un administrador y orquestador de Contenedores que ayuda a desplegar y escalar las aplicaciones en los contenedores, tiene la ventaja de contener una robusta seguridad dado la naturaleza de los servicios AWS pero tiene una desventaja principal y es que no existe el soporte para ejecutar contenedores fuera de EC2


Desafíos y consideraciones al usar contenedores



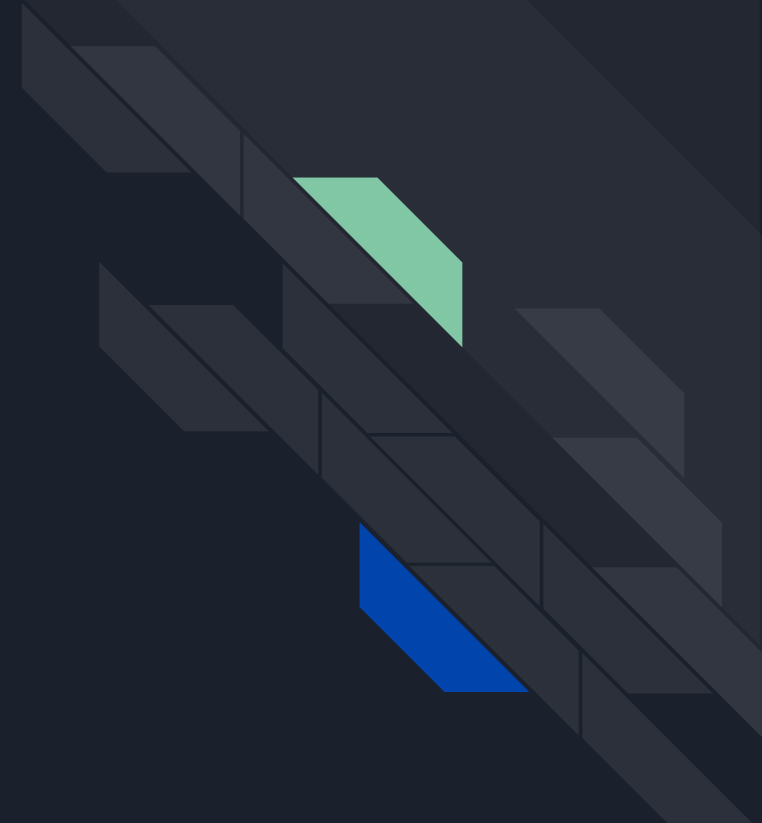


Como hemos venido hablando los contenedores son una forma muy versátil de tener un entorno con todas las dependencias necesarias ya establecidas y fácilmente ejecutable y migrable entre distintas plataformas pero usarlos presenta ciertos desafíos y consideraciones que hay que tomar en cuenta y citaremos a continuación:

- **Problemas con seguridad:** Los contenedores como todo sistema informático presente tienen vulnerabilidades presentes a los que atacantes pueden sobrepasar para robar información, pero en los contenedores existe un pequeño contratiempo las herramientas tradicionales de detección de vulnerabilidades pueden bloquear la ejecución de la integración y el despliegue de los contenedores por lo que las empresas tienden a no usarlas de manera correcta para evitar posibles cuellos de botella.
- **Problemas con configuraciones:** aproximadamente el 20% de los contenedores vulnerados es debido a una mala configuración por lo que para garantizar la solidez de un contenedor es necesario documentarse correctamente para realizar una configuración apropiada.

- 
- **Manejo de redes:** Como discutimos el en punto anterior una mala configuración de red puede dejar abierto el sistema para cualquiera, pero al momento de intentar comunicarse con un host pueden surgir errores desafiantes de corregir si no se tiene un buen control de la configuración de red, esto puede generar pérdidas de tiempo innecesarias.
 - **Constantes actualizaciones de imágenes:** Puede ser un poco obvio, pero se necesita actualizar periódicamente las imágenes de los contenedores para obtener las últimas actualizaciones de seguridad necesarias para el correcto funcionamiento de este.

Implementación de contenedores en una API REST



Lenguaje y Framework

 ASP.NET Core Web API
Una plantilla de proyecto para crear una API web RESTful utilizando controladores ASP.NET Core o API mínimas, con soporte opcional para OpenAPI y autenticación.

C#

Linux

macOS

Windows

API

Nube

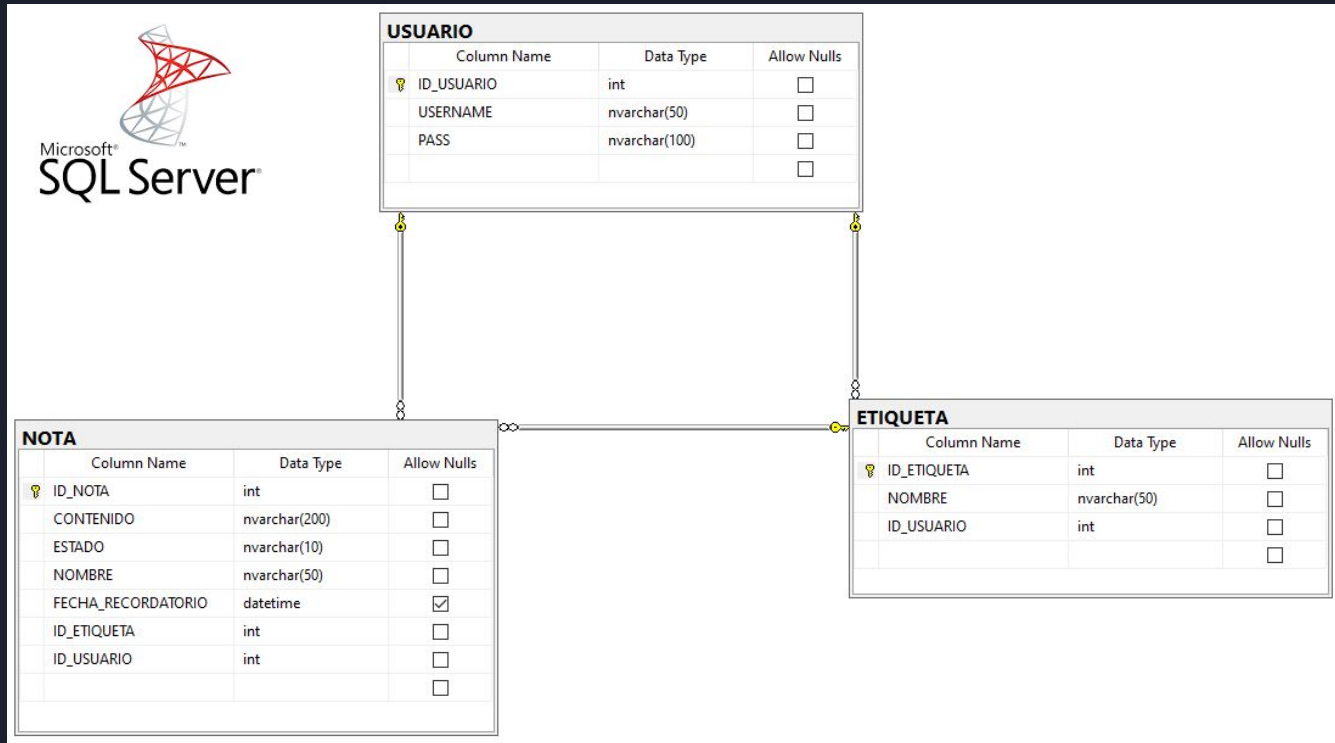
Servicio

Web

Web API



Base de datos



Dockerfile

```

# Esta fase se usa cuando se ejecuta desde VS en modo rápido (valor predeterminado para la configuración de depuración)
FROM mcr.microsoft.com/dotnet/aspnet:8.0 AS base
USER app
WORKDIR /app
EXPOSE 8080
EXPOSE 8081

# Esta fase se usa para compilar el proyecto de servicio
FROM mcr.microsoft.com/dotnet/sdk:8.0 AS build
ARG BUILD_CONFIGURATION=Release
WORKDIR /src
COPY ["API_REST_ADMIN_NOTAS.csproj", "."]
RUN dotnet restore "./API_REST_ADMIN_NOTAS.csproj"
COPY . .
WORKDIR "/src/."
RUN dotnet build "./API_REST_ADMIN_NOTAS.csproj" -c $BUILD_CONFIGURATION -o /app/build

# Esta fase se usa para publicar el proyecto de servicio que se copiará en la fase final.
FROM build AS publish
ARG BUILD_CONFIGURATION=Release
RUN dotnet publish "./API_REST_ADMIN_NOTAS.csproj" -c $BUILD_CONFIGURATION -o /app/publish /p:UseAppHost=false

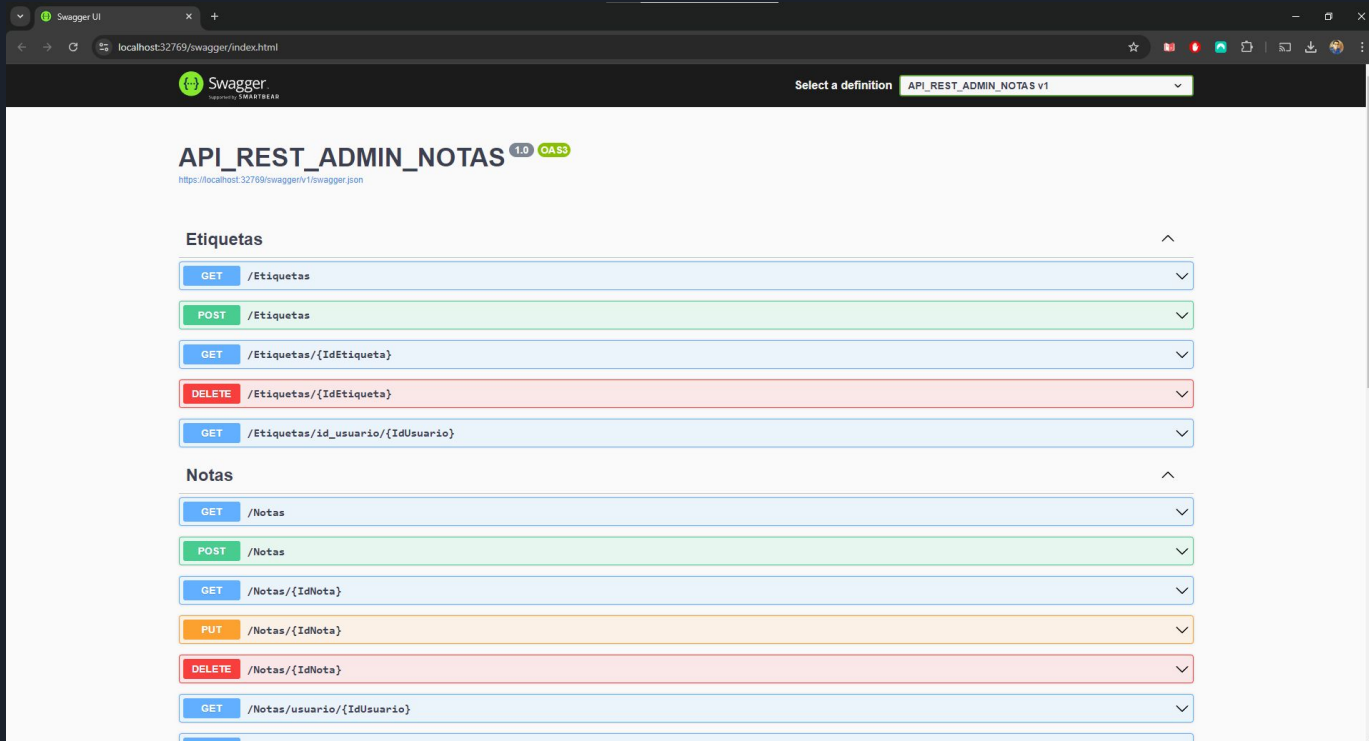
# Esta fase se usa en producción o cuando se ejecuta desde VS en modo normal (valor predeterminado
#cuando no se usa la configuración de depuración)
FROM base AS final
WORKDIR /app
COPY --from=publish /app/publish .
ENTRYPOINT ["dotnet", "API_REST_ADMIN_NOTAS.dll"]
```

LaunchSettings

schema: <http://json.schemastore.org/launchsettings.json>

```
1  {
2    "profiles": {
3      "http": {
4        "commandName": "Project",
5        "launchBrowser": true,
6        "launchUrl": "swagger",
7        "environmentVariables": {
8          "ASPNETCORE_ENVIRONMENT": "Development"
9        },
10       "dotnetRunMessages": true,
11       "applicationUrl": "http://localhost:5066"
12     },
13     "https": {
14       "commandName": "Project",
15       "launchBrowser": true,
16       "launchUrl": "swagger",
17       "environmentVariables": {
18         "ASPNETCORE_ENVIRONMENT": "Development"
19       },
20       "dotnetRunMessages": true,
21       "applicationUrl": "https://localhost:7197;http://localhost:5066"
22     },
23     "IIS Express": {
24       "commandName": "IISExpress",
25       "launchBrowser": true,
26       "launchUrl": "swagger",
27       "environmentVariables": {
28         "ASPNETCORE_ENVIRONMENT": "Development"
29       }
30     },
31     "Container (Dockerfile)": {
32       "commandName": "Docker",
33       "launchBrowser": true,
34       "launchUrl": "{Scheme}://{ServiceHost}:{ServicePort}/swagger",
35       "environmentVariables": {
36         "ASPNETCORE_HTTPS_PORTS": "8081",
37         "ASPNETCORE_HTTP_PORTS": "8080"
38       },
39       "publishAllPorts": true,
40       "useSSL": true
41     }
42   },
43   "$schema": "http://json.schemastore.org/launchsettings.json",
44 }
```

API REST ejecutándose



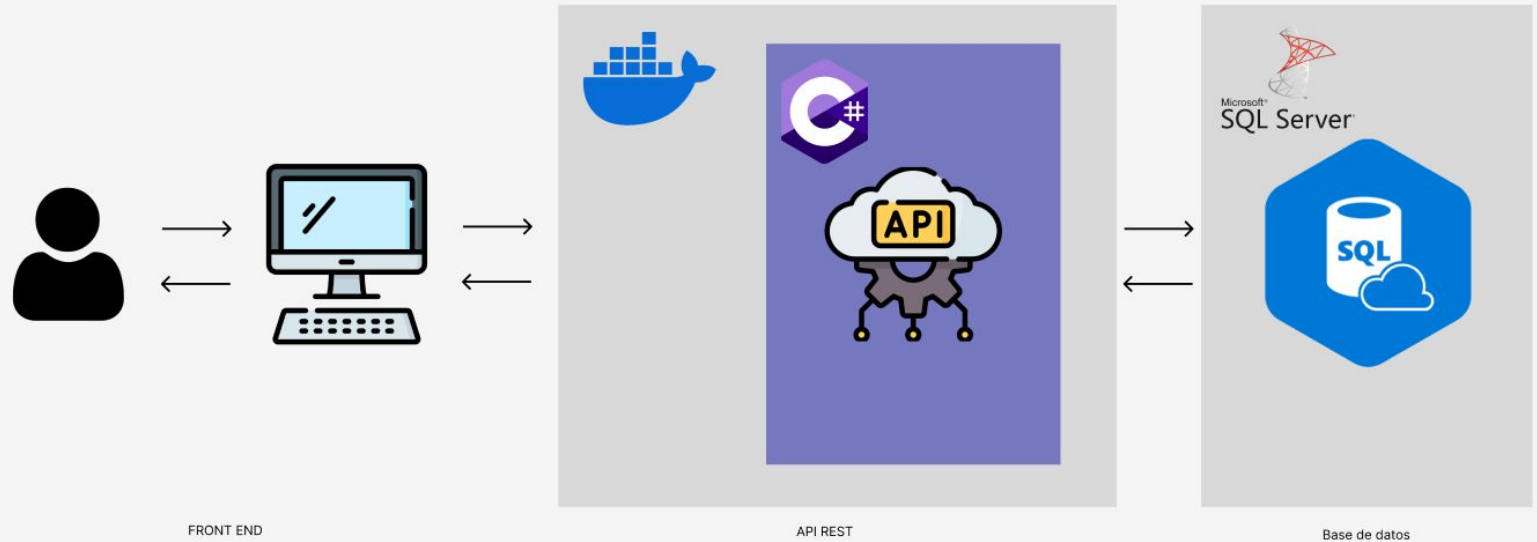
Contenedor ejecutándose

The screenshot shows the Docker Desktop application window. The left sidebar contains navigation options: Containers (selected), Images, Volumes, Builds, Docker Scout, and Extensions. The main area is titled 'Containers' and displays system metrics: 'Container CPU usage 0.79% / 800% (8 CPUs available)' and 'Container memory usage 132MB / 2.78GB'. A search bar and a toggle for 'Only show running containers' are present. A table lists the running containers:

| <input type="checkbox"/> | Name | Image | Status | Port(s) | CPU (%) | Last started | Actions |
|--------------------------|--|--|---------|--|---------|----------------|---------------------------|
| <input type="checkbox"/> | API_REST_ADMIN_NOTAS 99f025688cd2 | api-rest-admin-notas.dev | Running | 32768:8080 Show all ports (2) | 0.85% | 54 minutes ago | [Stop] [Refresh] [Delete] |

At the bottom, a status bar indicates 'Engine running' and system resources: 'RAM 1.21 GB · CPU 0.63% · Disk --- GB avail. of --- GB'. On the right, there are links for 'Terminal', 'New version available', and a notification icon with '2'.

Arquitectura de software



Repositorio

[https://github.com/DMancia03/API REST ADMIN NOTAS DPS](https://github.com/DMancia03/API_REST_ADMIN_NOTAS_DPS)





¡GRACIAS POR LA ATENCIÓN PRESTADA!