UNIVERSIDAD DON BOSCO

Facultad de Ingeniería

Escuela de Computación

Investigación Aplicada 2:

Contenedores para Desarrollo de Software

Docente:

Ing. Alexander Sigüenza

Integrantes:

Mejía Ortiz	Karla Lissette	MO190663
Chávez Flores	Jeanluca	CF190725
Cartagena Rivera	José Ángel	CR190362
Sánchez Mangandi	Angel Guillermo	SM192656

1. Definir qué son los contenedores

Los contenedores son una tecnología que permite empaquetar aplicaciones y sus dependencias en unidades estandarizadas, lo que facilita su uso en varios entornos. Los contenedores brindan un entorno aislado donde las aplicaciones pueden ejecutarse de manera consistente, independientemente de la infraestructura subyacente, en el contexto del desarrollo de software.

a. Explicar el concepto de contenedores en el contexto del desarrollo de software.

Los contenedores empaquetan las aplicaciones junto con todas las dependencias, como bibliotecas, configuraciones y otros recursos, necesarias para su ejecución. Este empaquetado se realiza a través de una imagen de contenedor, que es un archivo que contiene todos los elementos que la aplicación necesita. Los archivos de configuración como Dockerfiles, que proporcionan instrucciones para instalar dependencias y configurar el entorno de la aplicación, se utilizan para crear imágenes de contenedor.

Este método garantiza que las aplicaciones sean portátiles y funcionen de manera consistente en cualquier sistema que soportes contenedores, lo que elimina los problemas causados por cambios en las configuraciones ambientales. Además, las imágenes de contenedor se crean de manera eficiente mediante capas, lo que permite que los contenedores reutilicen componentes comunes entre diferentes aplicaciones, reduciendo el tamaño de las imágenes y acelerando su despliegue.

b. Diferenciar entre contenedores y máquinas virtuales.

Características	Contenedores	Máquinas Virtuales (VM)
Definición	Es un paquete de software que incluye el código de la aplicación, sus bibliotecas y todas las dependencias necesarias para ejecutar la aplicación en cualquier entorno.	Una máquina virtual es una copia digital de una máquina física que incluye su propio sistema operativo y permite dividir el hardware físico en varios entornos independientes.
Virtualización	Virtualizan el sistema operativo, lo que permite ejecutar aplicaciones de manera independiente	Virtualizan la infraestructura física subyacente, proporcionando un

	sobre cualquier plataforma.	entorno completo con su propio sistema operativo.
Encapsulación	Solo se encapsula la capa de software necesaria para ejecutar una aplicación o un componente en los contenedores.	Se encapsula el sistema operativo completo junto con todas sus capas de software y aplicaciones instaladas.
Tecnología	Utilizan un motor o tiempo de ejecución, como Docker, que coordina los recursos necesarios con el sistema operativo subyacente.	Emplean un hipervisor que coordina el uso de los recursos físicos entre el sistema operativo del host y el sistema operativo invitado.
Tamaño	Son significativamente más ligeros porque solo contienen los componentes necesarios para la ejecución de una aplicación, y su tamaño generalmente se indica en megabytes (MB).	Las máquinas virtuales con el sistema operativo completo son mucho más grandes y ocupan muchos gigabytes (GB).
Flexibilidad	Pueden migrar fácilmente entre entornos locales y en la nube, lo que los hace más fáciles de transportar y desplegar.	Las máquinas virtuales no son tan adaptables y migrar a diferentes entornos, especialmente en la nube, puede ser difícil y requerir mayor esfuerzo.

2. Describir el funcionamiento de los contenedores

a. Analizar cómo los contenedores empaquetan aplicaciones y sus dependencias.

Los contenedores empaquetan una aplicación con todas las dependencias necesarias, incluidas bibliotecas, configuraciones y otros recursos, necesarios para su ejecución. El empaquetado se realiza a través de una imagen de contenedor, que es un archivo que contiene todos los elementos y que la aplicación

requiere. Estas imágenes de contenedor se construyen a partir de los archivos de configuración llamados Dockerfiles, que incluyen instrucciones para la instalación de dependencias y la configuración del entorno de la aplicación.

Este enfoque asegura que las aplicaciones sean portables, ya que se ejecutarán de manera coherente en cualquier sistema compatible con contenedores sin problemas derivados de las diferencias en las configuraciones. Además, se mantiene un alto grado de eficiencia a través de capas con la reutilización de imágenes, lo que ahorra tiempo durante el despliegue.

b. Explicar el proceso de aislamiento y ejecución de contenedores en el sistema operativo.

Aislamiento de contenedores: Los contenedores están diseñados para ejecutar aplicaciones de manera aislada en un único host, esto implica que los procesos, redes y sistemas de archivos de un contenedor no interfieren en los otros contenedores. Esto se logra a través de dos componentes puntuales:

Namespaces (Espacios de nombres): Separan los recursos del sistema operativo, como los procesos y redes. Cada contenedor tiene sus propios recursos, por lo que no puede ver ni interactuar directamente con los recursos de otro contenedor. Esto es esencial para la independencia operativa y la seguridad de cada contenedor.

Cgroups (Grupos de Control): Gestionan el uso de los recursos del sistema (CPU, memoria, red, etc.) por parte de los contenedores, asegurando que un contenedor no acapare más recursos de los asignados, lo que mantiene el rendimiento. del sistema equilibrado.

3. Explorar las ventajas de utilizar contenedores

a. Discutir los beneficios en términos de portabilidad, eficiencia y consistencia.

El uso de contenedores en el desarrollo de software ofrece múltiples ventajas que han transformado la manera en que las aplicaciones se crean, prueban y despliegan. Entre los beneficios más destacados se encuentran la portabilidad, la eficiencia y la consistencia.

Portabilidad: Una de las principales ventajas de los contenedores es su capacidad para ejecutarse en cualquier entorno que cuente con un motor de contenedores compatible, como Docker. Debido a que los contenedores encapsulan todas las

dependencias de una aplicación, incluyendo bibliotecas, configuraciones y archivos de sistema, se garantiza que la aplicación se ejecutará de manera idéntica en distintos entornos, como máquinas locales, servidores de desarrollo, entornos de prueba o implementaciones en la nube. Esto elimina las fricciones relacionadas con la compatibilidad entre diferentes sistemas operativos y entornos de infraestructura, un problema común en la entrega de software tradicional. Esta portabilidad también facilita la migración entre plataformas, lo que es especialmente útil en entornos de nubes híbridas o multi-nube, donde las organizaciones buscan flexibilidad para mover cargas de trabajo entre diferentes proveedores de servicios de nube.

Eficiencia: A diferencia de las máquinas virtuales, los contenedores son mucho más ligeros en cuanto al uso de recursos. Mientras que las máquinas virtuales incluyen un sistema operativo completo, lo que implica un mayor consumo de CPU, memoria y almacenamiento, los contenedores comparten el kernel del sistema operativo del host. Esto significa que múltiples contenedores pueden ejecutarse en la misma máquina física sin incurrir en la sobrecarga de múltiples instancias de sistemas operativos. Además, los contenedores pueden iniciarse en cuestión de segundos o incluso milisegundos, en comparación con el arranque de una máquina virtual, que suele ser considerablemente más lento. Esta rapidez en el inicio y cierre de contenedores facilita la escalabilidad automática y la gestión dinámica de recursos en entornos de producción.

Consistencia: Los contenedores garantizan que una aplicación funcione de manera consistente en diferentes entornos. Dado que los contenedores incluyen todo lo necesario para ejecutar la aplicación, eliminan los problemas asociados a las configuraciones específicas de cada entorno o diferencias en las versiones de las dependencias. Esta consistencia es particularmente valiosa en los flujos de trabajo de DevOps, donde los equipos de desarrollo, operaciones y pruebas necesitan garantizar que lo que se desarrolla y prueba sea exactamente lo que se desplegará en producción. Al evitar el conocido problema de "en mi máquina funciona", los contenedores simplifican la integración y entrega continua (CI/CD), mejorando la calidad y confiabilidad del software desplegado.

b. Ejemplificar cómo los contenedores facilitan el desarrollo, pruebas y despliegue de aplicaciones.

Desarrollo: Durante la etapa de desarrollo, los contenedores permiten a los desarrolladores trabajar en entornos idénticos a los que se utilizarán en producción. Esto elimina las diferencias entre el entorno de desarrollo y el de despliegue, lo que resulta en un proceso de desarrollo más fluido y eficiente. Los entornos de desarrollo pueden crearse y destruirse rápidamente, lo que permite a los equipos de desarrollo experimentar con diferentes configuraciones y probar rápidamente nuevos enfoques. Además, los contenedores permiten realizar desarrollos modulares, donde diferentes partes de la aplicación (por ejemplo, bases de datos, servicios web) se ejecutan en contenedores separados, lo que facilita la colaboración entre equipos que trabajan en diferentes componentes de una misma aplicación.

Pruebas: En la fase de pruebas, los contenedores proporcionan un entorno controlado y reproducible. Las pruebas automáticas pueden ejecutarse en entornos idénticos a los de producción, asegurando que los resultados de las pruebas sean confiables y no estén influenciados por diferencias en la configuración del entorno. Además, la capacidad de ejecutar múltiples contenedores simultáneamente permite la ejecución paralela de pruebas, lo que acelera considerablemente el proceso de validación de la aplicación. Los contenedores también facilitan las pruebas de integración continua, donde las nuevas versiones de la aplicación pueden probarse automáticamente en entornos aislados antes de su despliegue.

Despliegue: Uno de los mayores beneficios de los contenedores es su capacidad para facilitar el despliegue continuo. Los contenedores pueden ser empaquetados una vez y desplegados en cualquier entorno sin modificaciones adicionales, lo que simplifica enormemente el proceso de entrega continua. Herramientas de orquestación como Kubernetes permiten gestionar millas de contenedores simultáneamente, escalando dinámicamente la aplicación según sea necesario. El despliegue de nuevas versiones de la aplicación puede realizarse sin interrumpir el servicio mediante técnicas como el Rolling Update o el blue-green development , donde se despliegan nuevas versiones de la aplicación en contenedores mientras los usuarios siguen accediendo a la versión anterior, lo que minimiza. el tiempo de inactividad.

Ejemplo: Una organización que utiliza contenedores puede desarrollar una aplicación web en un entorno local donde todas las dependencias están incluidas en un contenedor. Los desarrolladores trabajan en este contenedor localmente,

mientras que los equipos de prueba ejecutan pruebas automáticas en instancias de contenedores idénticos en un entorno de CI/CD. Posteriormente, la aplicación se despliega en producción mediante un sistema de orquestación, asegurando que el mismo contenedor se ejecute en un entorno de nube sin modificar la configuración o el código base. Esto no solo reduce los errores, sino que mejora significativamente la velocidad y confiabilidad del despliegue de la aplicación.

4. Presentar herramientas populares en el ecosistema de contenedores

a. Describir y comparar Docker, Kubernetes y otras herramientas relevantes.

Docker debido al funcionamiento que posee de poder ejecutar aplicaciones en contenedore, los cuales son entornos ligeros y portátiles, con una usabilidad bastante versátil para el tipo de aplicaciones que puede desplegar, sin embargo, en el mercado existen competidores que pueden mejorar el ecosistema de contenedores significativamente por la necesidad de buscar aplicaciones más flexibles y escalables. Kubernetes es una herramienta conocida para la creación y despliegue de portátiles y diseñada para automatizar la implementación, escalado y operación de aplicaciones desplegadas; este se ha convertido en estándar para la gestión de contenedores en la nube y entornos locales.

Podman es una herramienta de gestión de contenedores que ofrece una alternativa ligera y segura a Docker, sin necesidad de ejecutar un servicio de fondo o "daemon" en segundo plano. Desarrollado por Red Hat, Podman está diseñado para ejecutar, gestionar y construir contenedores y sus imágenes, brindando una mayor flexibilidad y seguridad, especialmente en sistemas que requieren contenedores "rootless" (sin privilegios de superusuario), esto quiere decir que cada contenedor se ejecuta como un proceso hijo del proceso que lo inició. Esto mejora la seguridad y facilita la depuración. Además de que Podman es compatible con las mismas imágenes y comandos que Docker, lo que facilita la transición. La mayoría de los comandos de Docker. Podman es una opción poderosa para quienes buscan una alternativa más segura y flexible a Docker, especialmente en entornos que necesitan gestión de contenedores sin daemon y ejecución sin root.

Helm es una herramienta de código abierto que facilita la gestión de instalación de paquetes para Kubernetes, este simplifica el proceso de desplegar y actualizar aplicaciones complejas a través de plantillas predefinidas llamadas Charts, los cuales son colecciones empaquetadas de todos los archivos y configuraciones necesarias para desplegar una aplicación en Kubernetes. Un chart puede describir

una aplicación compleja que consista en múltiples servicios o un simple microservicio. Helm es una herramienta poderosa que mejora la experiencia de gestionar aplicaciones en Kubernetes, proporcionando una forma estructurada y sencilla de desplegar, actualizar y gestionar aplicaciones complejas. Con su capacidad de reutilización, personalización y gestión de versiones, es esencial para los equipos que buscan optimizar el despliegue de aplicaciones en entornos de Kubernetes.

Prometheus es un sistema de monitoreo y alerta de código abierto diseñado para aplicaciones y servicios distribuidos, actualmente se ha convertido en un estándar en la monitorizaion de aplicaciones y recolección de datos y métricas, especialmente en entornos nativos de la nube. El servicio de métricas recolecta datos en intervalos regulares de las aplicaciones que monitoriza, este funciona de forma que periódicamente consulta los endpoints, de los cuales estos datos se pueden crear alertas, gráficos e informes.

b. Mostrar casos de uso y ejemplos prácticos de cada herramienta.

- Docker: Algunos casos de usos en los que se puede mencionar el uso de Docker puede ser un entorno en el que varios desarrolladores necesiten diferentes versiones de lenguaje de programación y dependencias, la aplicación de Docker se encargaría que cada desarrollador cree un contenedor de forma local y así asegurando que cada uno trabaje en las mismas versiones del software y con las mismas configuraciones, asi también levantar entornos de prueba, con una fluidez de forma que no afecte la eficiencia del trabajo. De la misma forma, con Docker se pueden desplegar aplicaciones en la nube en plataformas populares como Azure, AWS, Google cloud, etc.
- Kubernetes: Esta herramienta es ideal para aquellas aplicaciones que consisten en varios servicios pequeños y desacoplados que pueden ser gestionados independientemente. Ademas dentro de Kubernetes se es capaz de crear aplicaciones de machine Learning el cual sirve para ejecutar cargas de trabajo de Inteligencia Artificial y aprendizaje automatico, esto es posible ya que los contenedores creados aqui son capaces de desplegar aplicaciones complejas con gran demanda de recursos de computación.
- Podman: Este al tener una estructura similar a Docker, puede ejecutar aplicaciones de manera eficiente, pero al centrarse más en la seguridad ofrece algunas ventajas interesantes como lo son el despliege de aplicaciones en contenedores sin el uso de permisos de administrador, lo cual deriva en la reducción del impacto de posibles vulnerabilidades dentro de los contenedores, ya que no se ejecutan con permisos elevados, asimismo el

usuario final puede acceder a los contenedores sin necesidad de los permisos que normalmente se necesitan para interactuar con el sistema.

- Helm: Como se ha especificado, helm utiliza plantillas predefinidas llamadas Charts para el facilitamiento de despliegue de aplicaciones en el entorno de Kubernetes, el cual optimiza la configuracion, actualizacion y mantenimiento de las aplicaciones ejecutadas en dicha plataforma. Helm gestiona el ciclo de vida de las aplicaciones desplegando distintos componentes que requieran estar interconectados dentro del entorno, un ejemplo de esto seria Una plataforma de comercio electrónico que incluye servicios de frontend, backend, base de datos y almacenamiento se puede desplegar y gestionar fácilmente con un único chart de Helm.
- Prometheus: Tal como se ha dicho, esta herramienta funciona de tal forma que es posible monitorear la actividad y los datos de las aplicaciones desplegadas, de esta forma es posible ver el uso del CPU por parte de la aplicación; una visibilidad en tiempo real en caso de cuelgues y bajones de rendimiento o simplemente su disponibilidad en cualquier momento. Además, Prometheus está integrado en Kubernetes, lo cual permite la mejor eficiencia y monitorización de las aplicaciones asegurándose de que se ejecuten en el entorno sin problemas y con posibles soluciones a problemas futuros.

5. Discutir los desafíos y consideraciones al usar contenedores

a. Identificar posibles limitaciones y problemas comunes.

• Limitaciones

Persistencia de Datos: Los contenedores no guardan la información por siempre, ya que al ser eliminados se pierde toda la información almacenada.

Tiempo de Inicio: Algunos contenedores pueden ser lentos para iniciar, sobre todo si la información almacenada por ejemplo imágenes, tiene dependencias pesadas.

Redes complejas: La configuración de redes entre contenedores puede ser compleja, especialmente en entornos distribuidos.

Orquestación y escalabilidad: Gestionar una gran cantidad de contenedores de forma manual no es sencillo, es por eso que herramientas como kubernetes son necesarias para escalarlos adecuadamente.

• Problemas comunes

Seguridad: Los OS compartidos en los contenedores pueden aumentar la probabilidad de ataque si se compromete el host o las imágenes no están en un lugar seguro.

Gestión de recursos: Cuando no se gestionan bien los recursos, un contenedor consume más de lo esperado, afectando al resto.

Depuración: Depurar aplicaciones dentro de contenedores puede ser más complejo que hacerlo desde el OS anfitrión, requiriendo herramientas adicionales.

Sobrecarga de Red: Cuando se despliega a gran escala, la comunicación entre varios contenedores puede sobrecargar la red si no se gestiona correctamente.

b. Proponer soluciones y mejores prácticas para abordar estos desafíos

• Limitaciones

Persistencia de Datos: Implementar volúmenes de datos persistentes (Persisent Volumes, PV) y Persistent Volume Claims (PVC) en Kubernetes, o utilizar servicios externos como bases de datos gestionadas y almacenamiento en la nube, de este modo sea seguro que los datos no se perderán al momento de eliminar el contenedor.

Tiempo de inicios: Minimizar el tamaño de las imágenes de todos los contenedores, eliminando dependencias innecesarias y utilizando imágenes base más ligeras. Algunos ejemplos son: Photon OS, Distroless, Alpine Linux, etc.

Redes Complejas: Utilizar herramientas como Weave o Calico para simplificar la gestión de redes, y emplear políticas de red para gestionar el tráfico entre contenedores. Además, habilitar la segmentación de red entre pods para mejorar el aislamiento.

Orquestación y Escalabilidad: Implementar Kubernetes para automatizar la gestión y escalado de contenedores en lugar de hacerlo manualmente. Asegurarse de configurar el autoescalado y supervisar el uso de recursos en tiempo real para evitar sobrecargas.

• Problemas comunes

Seguridad: Mantener los contenedores aislados adecuadamente del sistema operativo host mediante el uso de espacios de nombres y cgroups. Utilizar soluciones como kata Containers para mejorar el aislamiento de los contenedores.

Gestión de recursos: Implementar límites y cuotas de recursos para cada contenedor en Kubernetes o Docker para evitar que un contenedor consuma más de lo esperado. Se debe configurar los límites en CPU, memoria y almacenamiento.

Depuración: Desactivar el modo debug en entornos de producción. Este modo puede exponer información sensible y ser una puerta de entrada para cualquier atacante.

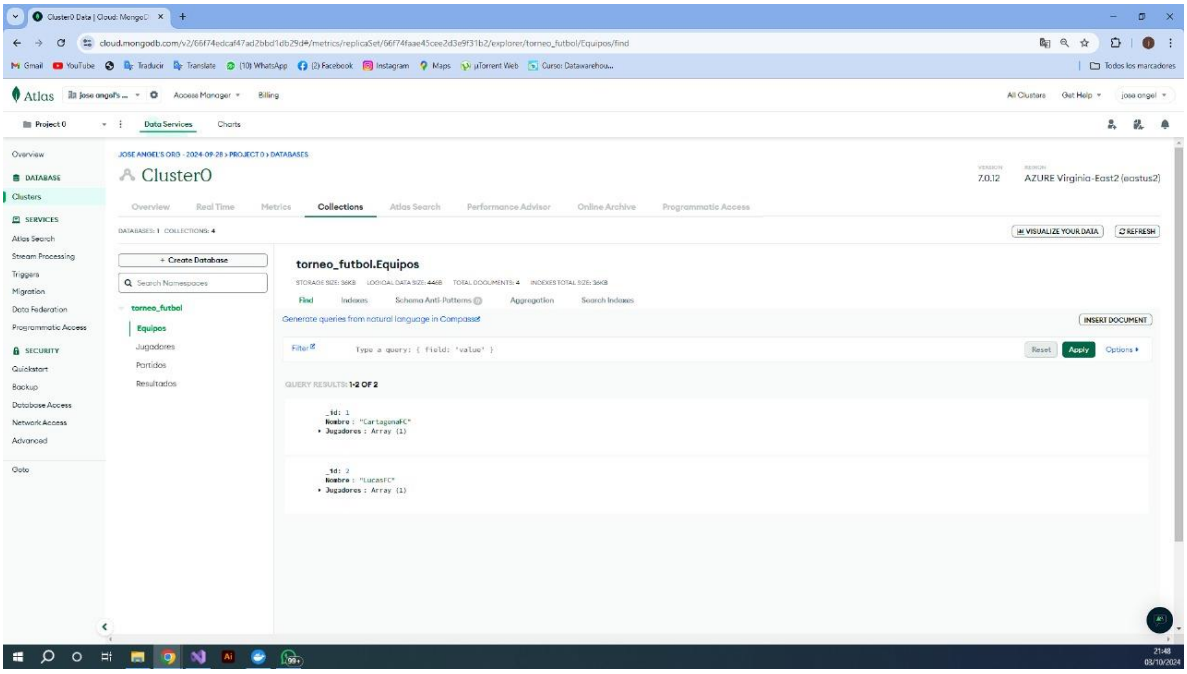
Sobrecarga de Red: Implementar soluciones de monitoreo y auditoría continua para detectar actividades sospechosas o vulnerabilidades en los contenedores. Herramientas como Prometheus o Sysdig pueden alterar y proporcionar métricas de seguridad en tiempo real.

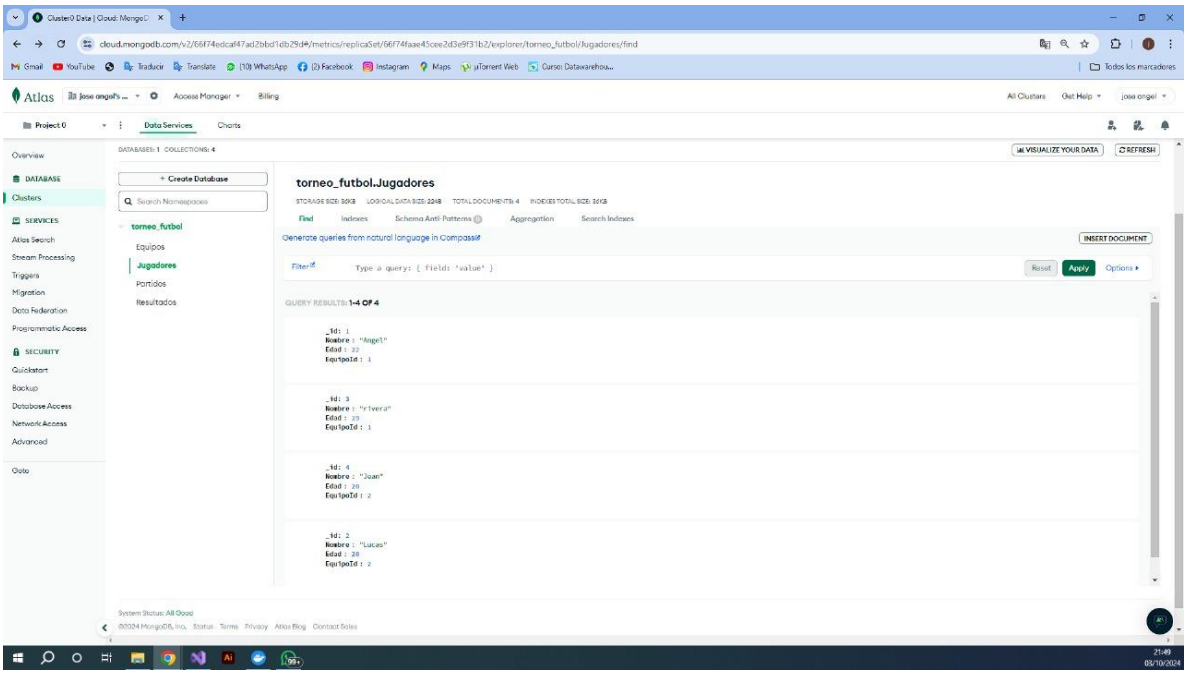
https://kinsta.com/es/blog/tecnologia-de-contenedores-aislados/

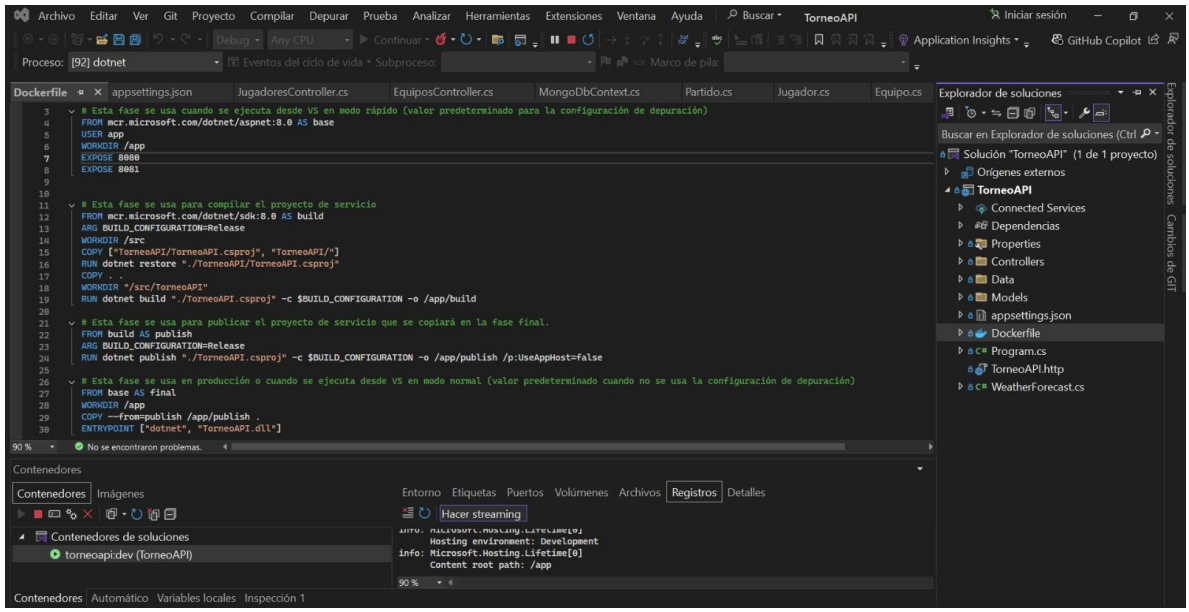
https://aws.amazon.com/es/compare/the-difference-between-containers-and-virtual-machines/

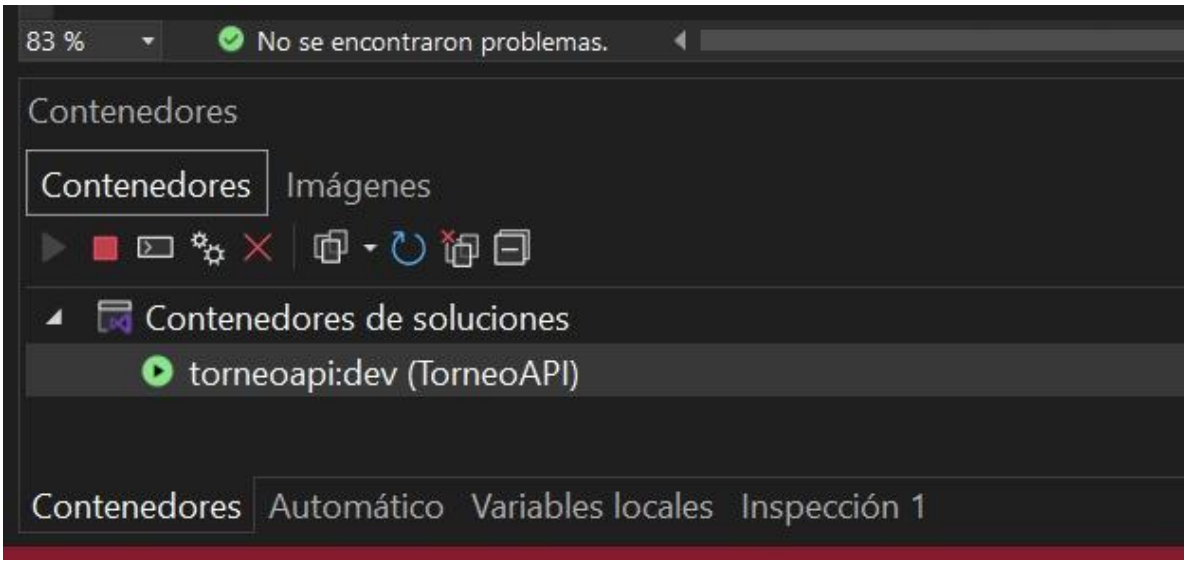
PARTE II

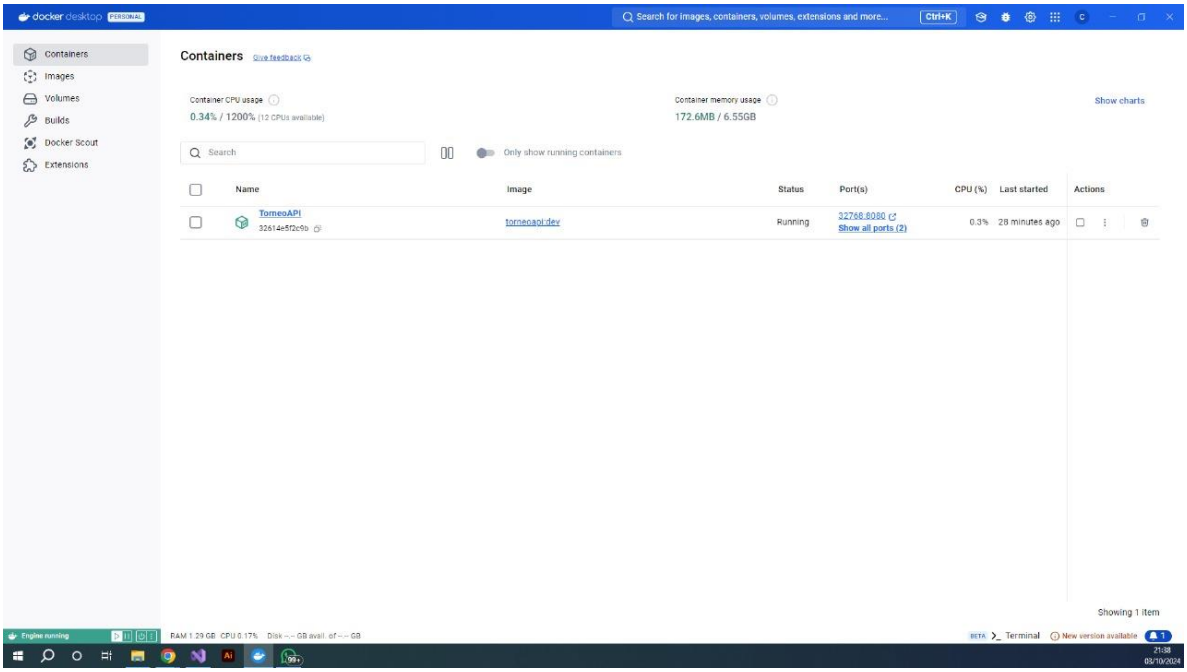
Podemos observar cómo se ven las tablas desde la base de datos “MongoDB”

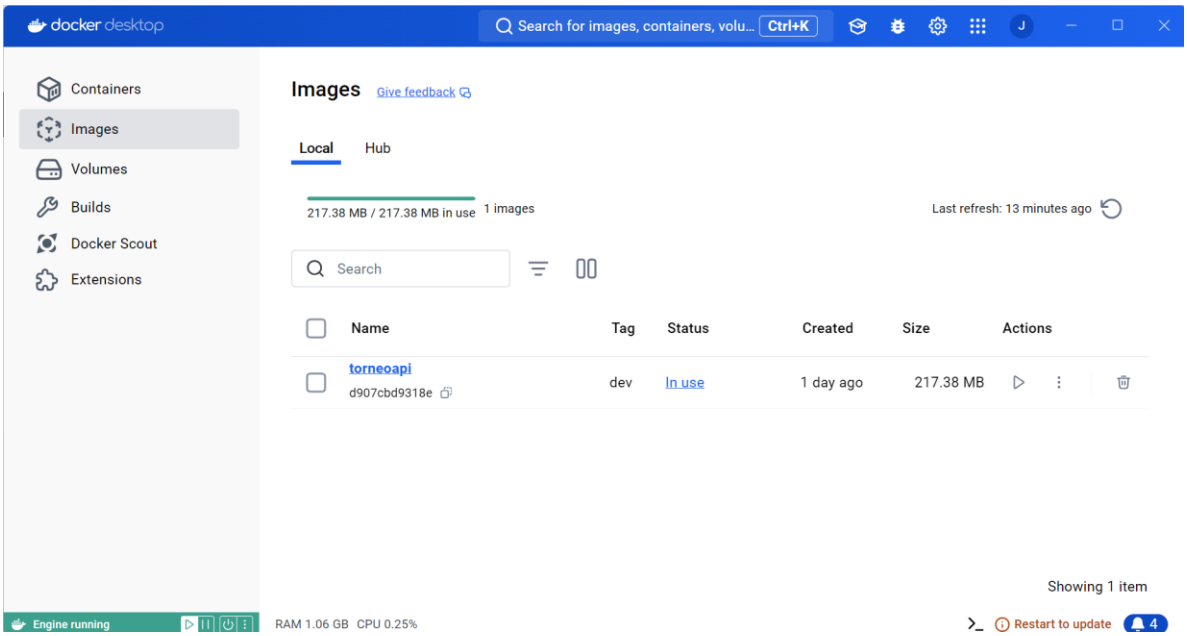




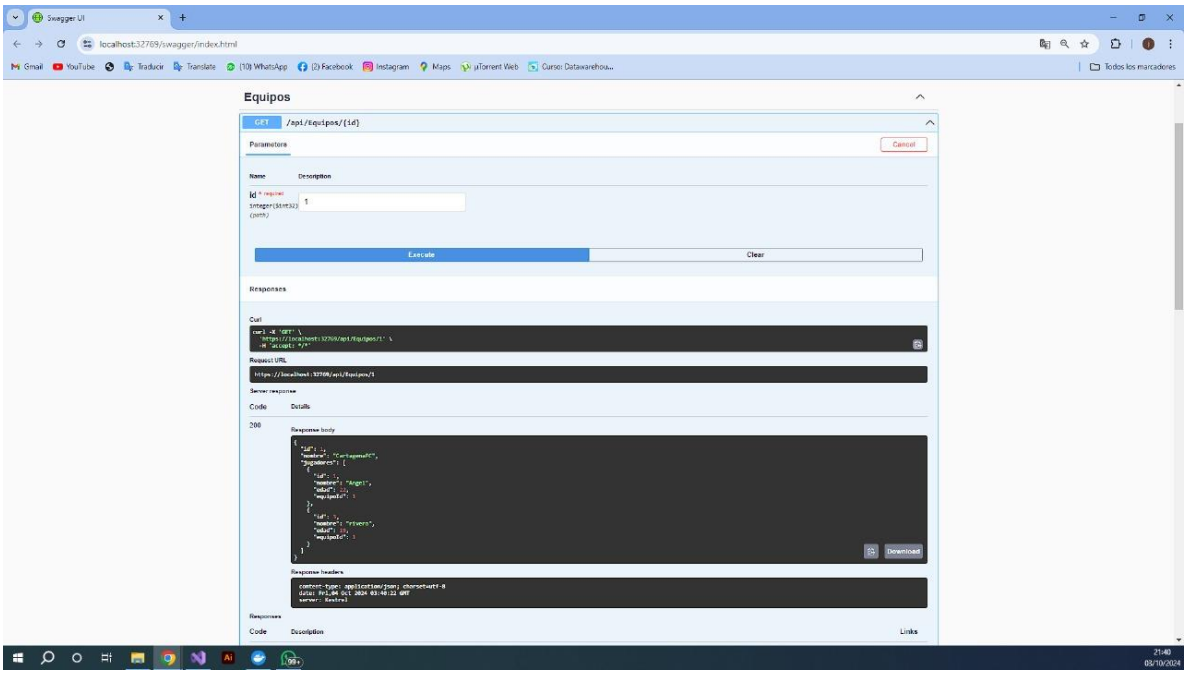

Archivo Editar Ver Git Proyecto Compilar Depurar Prueba Analizar Herramientas Extensiones Ventana Ayuda
Buscar
TorneoAPI
Iniciar sesión
Application Insights
GitHub Copilot
Proceso: [92] dotnet
Dockerfile
appsettings.json
JugadoresController.cs
EquiposController.cs
MongoDbContext.cs
Partido.cs
Jugador.cs
Equipo.cs
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 ["TorneoAPI/TorneoAPI.csproj", "TorneoAPI/"]
RUN dotnet restore "./TorneoAPI/TorneoAPI.csproj"
COPY . .
WORKDIR "/src/TorneoAPI"
RUN dotnet build "./TorneoAPI.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 "./TorneoAPI.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", "TorneoAPI.dll"]
No se encontraron problemas.
Explorador de soluciones
Buscar en Explorador de soluciones (Ctrl
Solución "TorneoAPI" (1 de 1 proyecto)
Orígenes externos
TorneoAPI
Connected Services
Dependencias
Properties
Controllers
Data
Models
appsettings.json
Dockerfile
Program.cs
TorneoAPI.http
WeatherForecast.cs
Explorador de soluciones
Cambios de GIT
Contenedores
Contenedores
Imágenes
Contenedores de soluciones
torneoapi:dev (TorneoAPI)
Entorno
Etiquetas
Puertos
Volúmenes
Archivos
Registros
Detalles
Hacer streaming
Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
Content root path: /app
Contenedores
Automático
Variables locales
Inspección 1

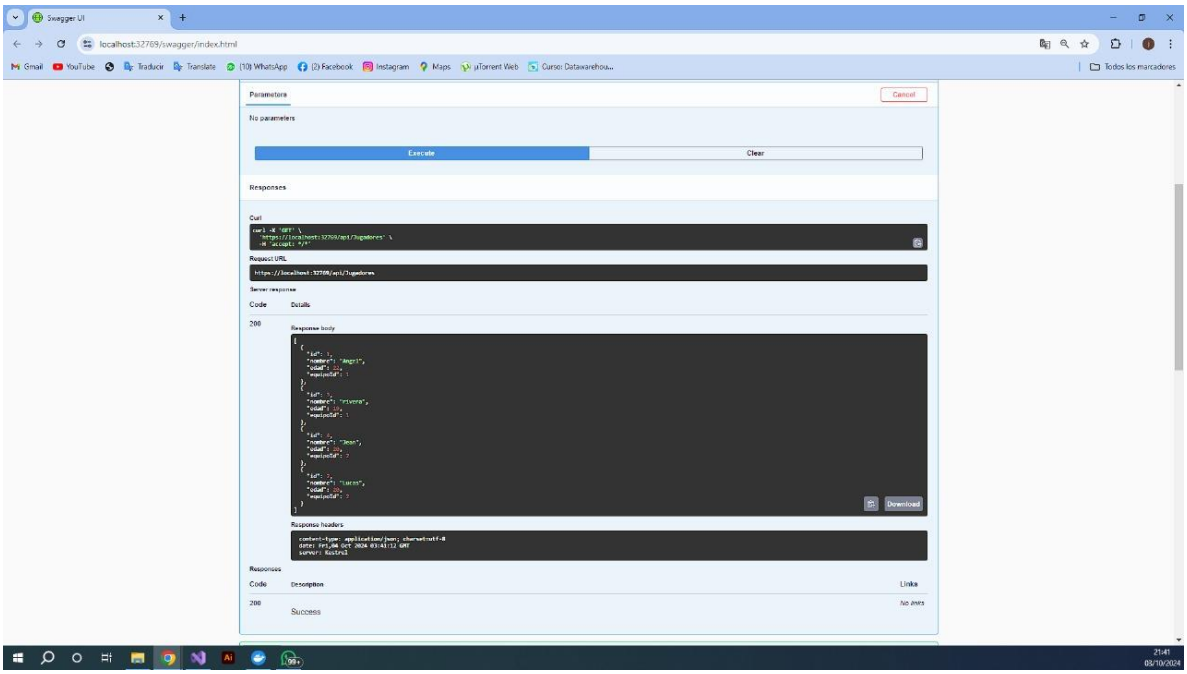

83 %
No se encontraron problemas.
Contenedores
Contenedores
Imágenes
Contenedores de soluciones
torneoapi:dev (TorneoAPI)
Contenedores
Automático
Variables locales
Inspección 1

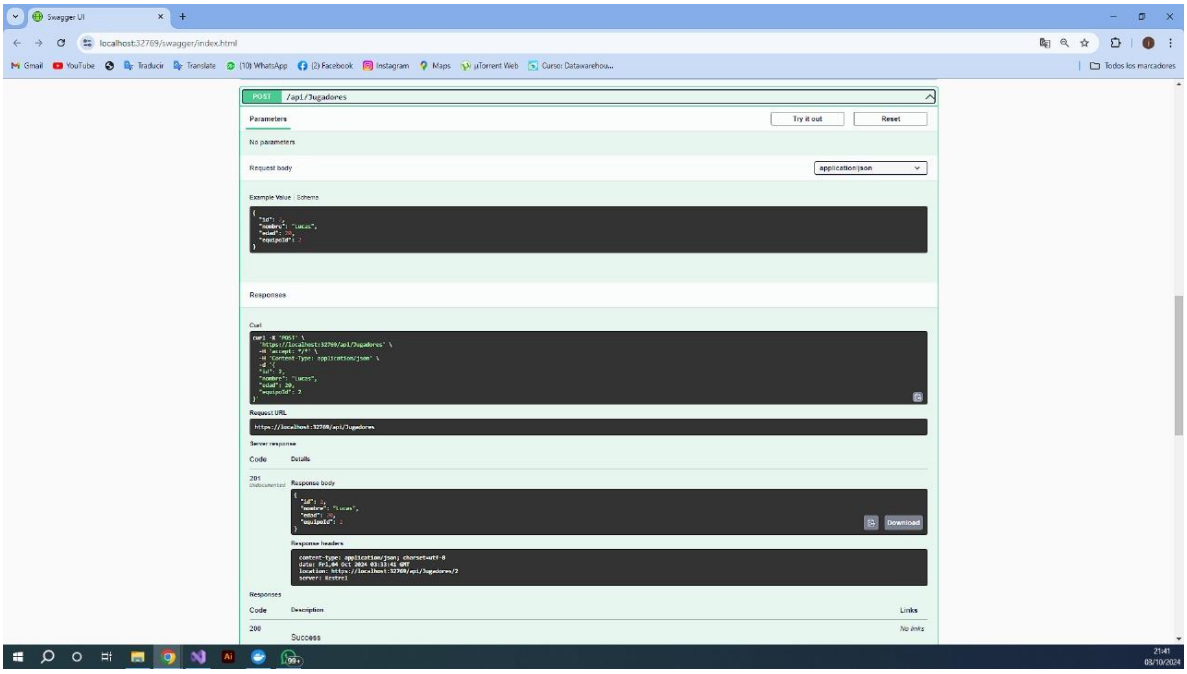

docker desktop
PERSONAL
Search for images, containers, volumes, extensions and more...
Ctrl+K
Containers
Images
Volumes
Builds
Docker Scout
Extensions
Containers
Give feedback
Container CPU usage
0.34% / 1200% (12 CPUs available)
Container memory usage
172.6MB / 6.55GB
Show charts
Search
Only show running containers
Name
Image
Status
Port(s)
CPU (%)
Last started
Actions
TorneoAPI
32614e5f2c9b
torneoapi:dev
Running
32768:8080
Show all ports (2)
0.3%
28 minutes ago
Showing 1 item
Engine running
RAM 1.29 GB CPU 0.17%
Terminal
New version available
21:38
03/10/2024

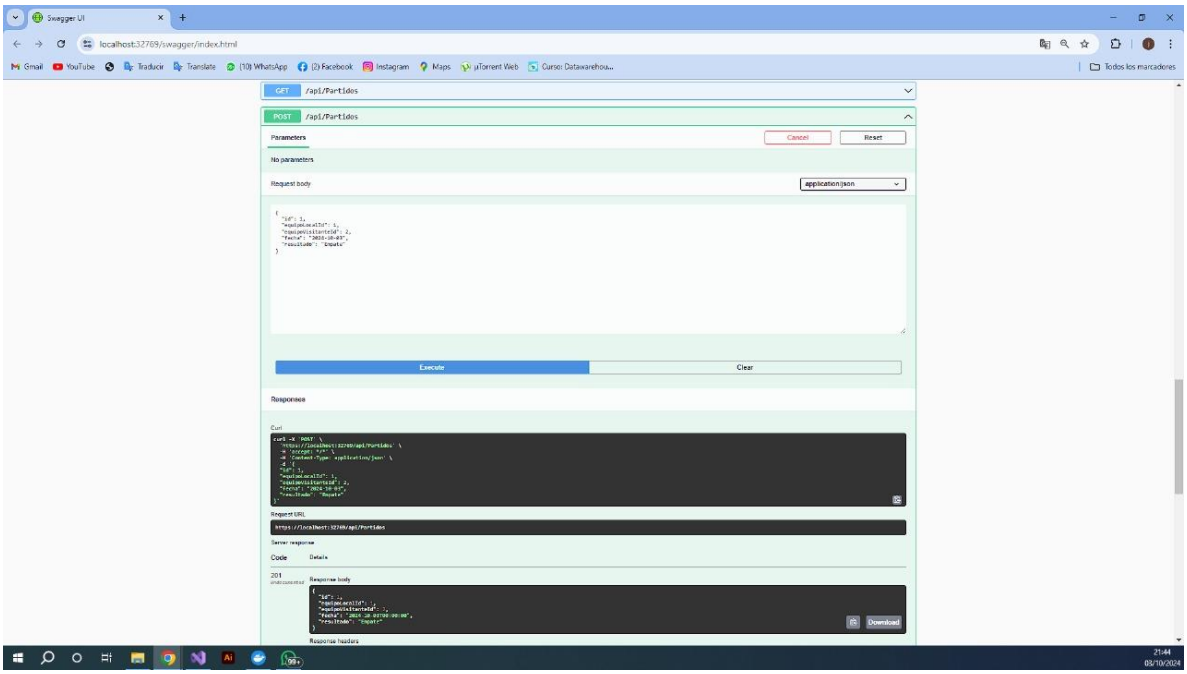


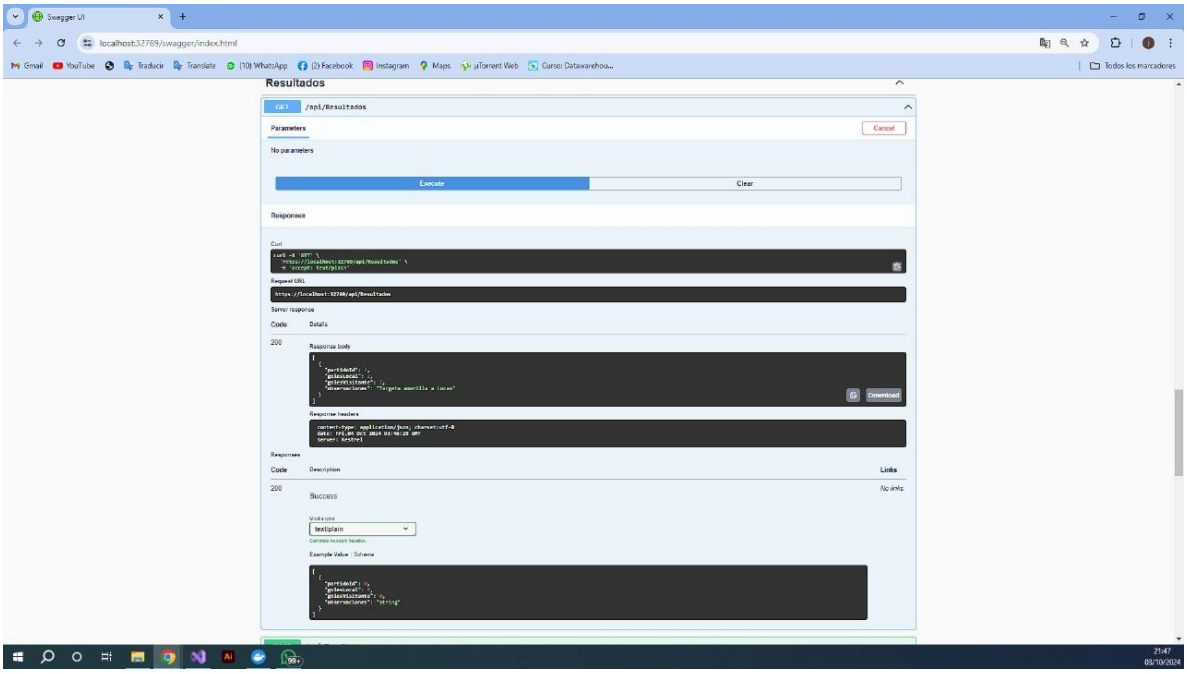
Funcionamiento de la API

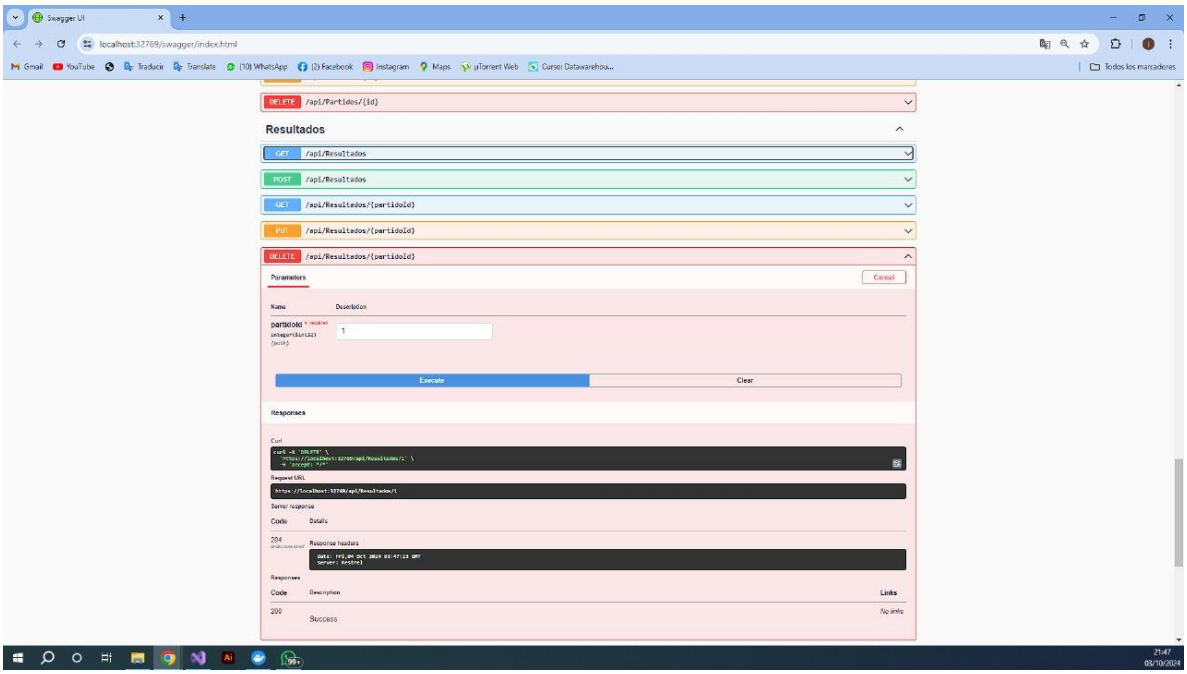



Swagger UI
localhost:32769/swagger/index.html
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
curl -X 'GET' \
'https://localhost:32769/api/Jugadores' \
-H 'accept: */*'
Request URL
https://localhost:32769/api/Jugadores
Server response
Code
Details
200
Response body
{
"id": 1,
"nombre": "Angel",
"edad": 25,
"equipoId": 1
},
{
"id": 2,
"nombre": "rivera",
"edad": 25,
"equipoId": 1
},
{
"id": 3,
"nombre": "Jean",
"edad": 25,
"equipoId": 2
},
{
"id": 4,
"nombre": "Lucas",
"edad": 25,
"equipoId": 2
}
Download
Response headers
content-type: application/json; charset=utf-8
date: Fri,04 Oct 2024 03:41:12 GMT
server: Kestrel
Responses
Code
Description
Links
200
Success
No links


Swagger UI
localhost:32769/swagger/index.html
POST
/api/Jugadores
Parameters
Try it out
Reset
No parameters
Request body
application/json
Example Value
Schema
{
"id": 2,
"nombre": "Lucas",
"edad": 25,
"equipoId": 2
}
Responses
Curl
curl -X 'POST' \
'https://localhost:32769/api/Jugadores' \
-H 'accept: */*' \
-H 'Content-Type: application/json' \
-d '{
"id": 2,
"nombre": "Lucas",
"edad": 25,
"equipoId": 2
}'
Request URL
https://localhost:32769/api/Jugadores
Server response
Code
Details
201
Response body
{
"id": 2,
"nombre": "Lucas",
"edad": 25,
"equipoId": 2
}
Download
Response headers
content-type: application/json; charset=utf-8
date: Fri,04 Oct 2024 03:33:41 GMT
location: https://localhost:32769/api/Jugadores/2
server: Kestrel
Responses
Code
Description
Links
200
Success
No links


Swagger UI
localhost:32769/swagger/index.html
GET /api/Partidos
POST /api/Partidos
Parameters
Cancel
Reset
No parameters
Request body
application/json
Execute
Clear
Responses
Curl
Request URL
https://localhost:32769/api/Partidos
Server response
Code
Details
201
Response body
Download
Response headers


Swagger UI
localhost:32769/swagger/index.html
Resultados
GET /api/Resultados
Parameters
Cancel
No parameters
Execute
Clear
Responses
Curl
Request URL
https://localhost:32769/api/Resultados
Server response
Code
Details
200
Response body
Download
Response headers
Responses
Code
Description
Links
Success
No links
Media type
text/plain
Example Value | Schema


Swagger UI
localhost:32769/swagger/index.html
DELETE /api/Partidos/{id}
Resultados
GET /api/Resultados
POST /api/Resultados
GET /api/Resultados/{partidoId}
PUT /api/Resultados/{partidoId}
DELETE /api/Resultados/{partidoId}
Parameters
Cancel
Name
Description
partidoId
Execute
Clear
Responses
Curl
Request URL
Server response
Code
Details
Response headers
Code
Description
Links
200
Success
No links