Docker 2023

Diego Saavedra

Dec 4, 2023

Table of contents

1 Curso de Docker

¡Bienvenidos al Curso de Docker!

Este curso te guiará a través de un viaje desde los fundamentos hasta el dominio de Docker, la plataforma de contenedores líder en la industria.

1.1 ¿Qué es este Curso?

Este curso exhaustivo te llevará desde los conceptos básicos de Docker hasta la implementación práctica de aplicaciones y servicios en contenedores. A través de una combinación de teoría y ejercicios prácticos, explorarás cada aspecto diseñado para fortalecer tus habilidades en el uso de Docker. Desde la instalación inicial hasta la orquestación de contenedores con Docker Compose, este curso te proporcionará las herramientas y conocimientos necesarios para desarrollar, desplegar y escalar aplicaciones con eficiencia.

1.2 ¿A quién está dirigido?

Este curso está diseñado tanto para aquellos que están dando sus primeros pasos en Docker como para aquellos que desean profundizar en sus conocimientos. Ya seas un estudiante, un profesional en busca de nuevas habilidades o alguien apasionado por la tecnología de contenedores, este curso te brindará la base necesaria para trabajar de manera efectiva con Docker en cualquier entorno.

1.3 ¿Cómo contribuir?

Valoramos tu participación en este curso. Si encuentras errores, deseas sugerir mejoras o agregar contenido adicional, ¡nos encantaría recibir tus contribuciones! Puedes contribuir a través de nuestra plataforma en línea, donde puedes compartir tus comentarios y sugerencias. Juntos, podemos mejorar continuamente este recurso educativo para beneficiar a la comunidad de usuarios de Docker.

Este curso ha sido creado con el objetivo de proporcionar acceso gratuito y universal al conocimiento de Docker. Estará disponible en línea para que cualquiera, sin importar su ubicación o circunstancias, pueda acceder y aprender a su propio ritmo.

¡Esperamos que disfrutes este emocionante viaje de aprendizaje y descubrimiento en el mundo de Docker y los contenedores!

Part I

Curso Docker

2 1. Comandos Básicos y Atajos

2.1 Conceptos:

Docker: Es una plataforma para desarrollar, enviar y ejecutar aplicaciones en contenedores. Un contenedor es una instancia ejecutable de una imagen.

Contenedor: Es una instancia de una imagen que se ejecuta de manera aislada. Los contenedores son ligeros y portátiles, ya que incluyen todo lo necesario para ejecutar una aplicación, incluidas las bibliotecas y las dependencias.

2.2 Ejemplos:

Descargar una imagen:

```
docker pull docker/getting-started
```

Este comando descarga la imagen **getting-started** desde el registro público de Docker.

Correr un contenedor en el puerto 80:

```
docker run -d -p 80:80 docker/getting-started
```

Este comando ejecuta un contenedor desenlazado en segundo plano (-d) y mapea el puerto 80 de la máquina host al puerto 80 del contenedor (-p 80:80).

2.3 Práctica:

Descarga la imagen “nginx”:

```
docker pull nginx
```

Descarga la imagen de Nginx desde el registro público.

Crea y ejecuta un contenedor en el puerto 8080:

```
docker run -d -p 8080:80 nginx
```

- Crea y ejecuta un contenedor de Nginx en el puerto 8080.
- Detén y elimina el contenedor creado:
- Utiliza los comandos para detener y eliminar un contenedor.

Resolución de la Actividad Práctica

Descarga la imagen de **Nginx**.

```
docker pull nginx
```

Crea y ejecuta un contenedor en el puerto 8080:

```
docker run -d -p 8080:80 nginx
```

Detén y elimina el contenedor creado:

```
docker stop $(docker ps -q)
docker rm $(docker ps -a -q)
```

> Tip
>
> Combinar banderas mejora la eficiencia en la ejecución de comandos.

2.4 ¿Qué Aprendimos?

- Aprendimos a descargar imágenes, correr contenedores y gestionarlos básicamente.
- Entendimos la importancia de las banderas en los comandos Docker.

3 Actividad Práctica

3.1 Objetivo:

Familiarizarse con los comandos básicos de Docker y atajos para gestionar contenedores de manera eficiente. Instrucciones:

- Utilizando el comando docker run, inicia un contenedor de la imagen “nginx” en segundo plano, mapeando el puerto 8080 del host al puerto 80 del contenedor.
- Detén y elimina el contenedor recién creado utilizando comandos Docker.
- Crea un nuevo contenedor con la imagen “alpine” y ejecuta un terminal interactivo dentro de él.
- Desde el contenedor alpine, instala el paquete curl utilizando el gestor de paquetes apk.
- Crea una imagen llamada “alpine-curl” a partir de este contenedor modificado.

3.2 Entregables:

- Documento explicando los comandos utilizados.
- Imagen Docker “alpine-curl” disponible localmente.

3.3 Rubrica de Evaluación:

- Correcta ejecución de comandos: 6 puntos
- Clara documentación: 4 puntos
- Imagen “alpine-curl” creada correctamente: 10 puntos

Resolución de la Actividad Práctica

- Iniciar un contenedor Nginx:

```
docker run -d -p 8080:80 --name my-nginx nginx
```

Detener y eliminar el contenedor Nginx:

```
docker stop my-nginx
docker rm my-nginx
```

Crear un contenedor Alpine interactivo:

```
docker run -it --name my-alpine alpine /bin/sh
```

Instalar el paquete curl desde el contenedor Alpine:

```
apk add --no-cache curl
```

Crear una nueva imagen “alpine-curl”:

```
docker commit my-alpine alpine-curl
```

4 2: Trabajando con Volúmenes

4.1 Conceptos:

Volumes: Son mecanismos que permiten persistir datos más allá del ciclo de vida de un contenedor.

4.2 Ejemplos:

Crear un nuevo volumen:

```
docker volume create my_volume
```

Este comando crea un volumen llamado “my_volume”.

Usar un volumen al correr un contenedor:

```
docker run -v my_volume:/app -d node:18-alpine
```

Este comando ejecuta un contenedor de Node.js y vincula el volumen “my_volume” al directorio “/app” dentro del contenedor.

4.3 Práctica:

- Crea un volumen llamado **data_volume**.
- Crea un volumen llamado **data_volume**.
- Ejecuta un contenedor de MySQL, utilizando el volumen creado.

Resolución de la Actividad Práctica

Crea un volumen:

```
docker volume create data_volume
```

Ejecuta el contenedor MySQL con el volumen creado:

```
docker run -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=mydb -v data_volume:/var/lib/m
```

> Tip
>
> **Recuerda la regla de oro:** Si dos contenedores están en la misma red, podrán comunicarse.

4.4 ¿Qué Aprendimos?

- Ahora entendemos cómo trabajar con volúmenes para mantener la persistencia de datos en Docker.
- Aprendimos sobre Named Volumes y cómo vincularlos a contenedores.

5 Actividad Práctica

5.1 Objetivo:

Practicar el uso de volúmenes en Docker para persistir datos entre contenedores.

5.2 Instrucciones:

- Crea un nuevo volumen llamado “mydata”.
- Inicia un contenedor de la imagen “nginx” y vincula el volumen “mydata” al directorio “/usr/share/nginx/html” dentro del contenedor.
- Crea un archivo HTML dentro del volumen “mydata” con el mensaje “Hola, este es un archivo HTML persistente”.
- Inicia otro contenedor de la imagen “nginx” y vincula el mismo volumen “mydata” al directorio “/usr/share/nginx/html” dentro de este segundo contenedor.
- Verifica que ambos contenedores comparten el mismo archivo HTML creado en el paso 3.

5.3 Entregables:

- Documento explicando los comandos utilizados.
- Capturas de pantalla que demuestren la persistencia de datos entre contenedores.

5.4 Rubrica de Evaluación:

- Correcta creación y vinculación de volúmenes: 6 puntos
- Creación y persistencia de archivos en el volumen: 8 puntos
- Verificación exitosa de la persistencia entre contenedores: 6 puntos

Resolución de la Actividad Práctica

Crear un nuevo volumen:

```
docker volume create mydata
```

Iniciar el primer contenedor Nginx con el volumen:

```
docker run -d -p 8080:80 --name nginx-1 -v mydata:/usr/share/nginx/html nginx
```

Crear un archivo HTML dentro del volumen:

```
docker exec -it nginx-1 sh -c "echo 'Hola, este es un archivo HTML persistente' > /usr/
```

Iniciar el segundo contenedor Nginx con el mismo volumen:

```
docker run -d -p 8081:80 --name nginx-2 -v mydata:/usr/share/nginx/html nginx
```

Verificar la persistencia del archivo HTML:

- Acceder a http://localhost:8080 en el navegador.
- Acceder a http://localhost:8081 en el navegador.

6 3: Dockerfile y Docker Compose

6.1 Conceptos:

Dockerfile: Es un archivo de texto que contiene instrucciones para construir una imagen Docker. Es como un plano para la construcción de imágenes.

Docker Compose: Es una herramienta que permite definir y compartir aplicaciones multi-contenedor. Con un solo archivo (docker-compose.yml), puedes configurar y ejecutar tus servicios.

6.2 Ejemplos:

- Crear un Dockerfile para una aplicación Node.js:

```
FROM node:14
WORKDIR /app
COPY . .
CMD ["npm", "start"]
```

Este Dockerfile configura una imagen de Node.js, establece el directorio de trabajo, copia los archivos locales al contenedor y define el comando para ejecutar la aplicación.

- Configurar Docker Compose para una aplicación Node.js:

```
version: '3'
services:
  myapp:
    build:
      context: .
      dockerfile: Dockerfile.node
    image: my-node-app
```

Este archivo docker-compose.yml define un servicio llamado “myapp” que construirá una imagen usando el Dockerfile “Dockerfile.node” y le asignará el nombre de “my-node-app”.

6.3 Práctica:

- Crea un Dockerfile para una aplicación Python simple:
- Configura un archivo docker-compose.yml para ejecutar la aplicación:

  Resolución de la Actividad Práctica
- Ejemplo de Dockerfile (nombre: Dockerfile.python):

```
FROM python:3.9
WORKDIR /app
COPY . .
CMD ["python", "app.py"]
```

Ejemplo de docker-compose.yml:

```
version: '3'
services:
  myapp:
    build:
      context: .
      dockerfile: Dockerfile.python
    image: my-python-app
```

Tip

Cuando trabajas con Docker Compose, es útil conocer el comando docker-compose up con la opción -d para ejecutar los contenedores en segundo plano. Esto permite liberar la terminal para otras operaciones mientras tus servicios continúan ejecutándose en el fondo.

```
docker-compose up -d
```

Este comando es especialmente útil en entornos de desarrollo donde deseas ejecutar múltiples servicios, pero aún así necesitas utilizar tu terminal para otras tareas. Además, puedes detener los servicios en segundo plano con:

```
docker-compose down
```

Esto ayudará a liberar los recursos utilizados por los contenedores sin afectar tu entorno de desarrollo principal.

6.4 ¿Qué Aprendimos?

- Hemos adquirido habilidades para crear imágenes personalizadas y gestionar aplicaciones multi-contenedor con Docker Compose.

- Ahora comprendemos la importancia de organizar nuestras aplicaciones en contenedores y cómo Docker Compose simplifica la orquestación.

7 Actividad Práctica

7.1 Objetivo:

Practicar la creación de imágenes personalizadas utilizando Dockerfile y la orquestación de servicios con Docker Compose.

7.2 Instrucciones:

- Crea un Dockerfile para una aplicación Python simple que imprima “Hola, Docker” al ejecutarse.
- Construye la imagen a partir del Dockerfile.
- Utilizando Docker Compose, define un servicio que utilice la imagen creada y exponga el puerto 5000.
- Inicia el servicio con Docker Compose.
- Accede a la aplicación en http://localhost:5000 y verifica que imprime “Hola, Docker”.

7.3 Entregables:

- Dockerfile para la aplicación Python.
- Archivo Docker Compose.
- Documento explicando los comandos utilizados.
- Capturas de pantalla que demuestren el acceso a la aplicación.

7.4 Rubrica de Evaluación:

- Correcta creación del Dockerfile: 6 puntos
- Imagen construida correctamente: 4 puntos
- Configuración adecuada en Docker Compose: 6 puntos
- Acceso exitoso a la aplicación: 4 puntos

Resolución de la Actividad Práctica

Dockerfile para la aplicación Python:

```
FROM python:3.9
CMD ["python", "-c", "print('Hola, Docker')"]
```

Construir la imagen:

```
docker build -t my-python-app .
```

Archivo Docker Compose (docker-compose.yml):

```
version: '3'
services:
  myapp:
    image: my-python-app
    ports:
      - "5000:5000"
```

Iniciar el servicio con Docker Compose:

```
docker-compose up -d
```

Verificar el acceso a la aplicación:

Acceder a http://localhost:5000 en el navegador.

8 4: Buenas Prácticas y Seguridad

8.1 Conceptos:

Escaneo de imagen: Después de construir una imagen, es buena práctica realizar un escaneo en busca de vulnerabilidades.

Capas de la imagen: Cada imagen de Docker se construye en capas, permitiendo un nivel de abstracción independiente.

Multi-Stage builds: Permiten separar dependencias necesarias para construir la aplicación de las necesarias para ejecutarla en producción, reduciendo el tamaño de la imagen final.

8.2 Ejemplos:

Escaneo de imagen con Snyk:

```
snyk container test <IMAGE_NAME:TAG>
```

Utilizando la herramienta Snyk, podemos escanear una imagen en busca de vulnerabilidades.

Uso de Multi-Stage builds:

```
FROM node:14 AS builder
WORKDIR /app
  COPY . .
  RUN npm install
  RUN npm run build

  FROM nginx:alpine
  COPY --from=builder /app/dist /usr/share/nginx/html
```

Este Dockerfile utiliza Multi-Stage builds para primero construir una aplicación Node.js y luego copiar solo los artefactos necesarios en una imagen más ligera de Nginx.

8.3 Práctica:

- Realiza un escaneo de vulnerabilidades en una imagen de tu elección:
- Utiliza Snyk para escanear una imagen de Docker.
- Implementa un Multi-Stage build en un Dockerfile:

Resolución de la Actividad Práctica

Escaneo de imagen con Snyk:

```
snyk container test my-image:my-tag
```

Ejemplo de Multi-Stage Dockerfile:

```
FROM node:14 AS builder
WORKDIR /app
  COPY . .
  RUN npm install
  RUN npm run build

  FROM nginx:alpine
  COPY --from=builder /app/dist /usr/share/nginx/html
```

> **Tip**
>
> Recuerda crear contenedores efímeros y desacoplar aplicaciones para mejorar el rendimiento y la eficiencia

8.4 ¿Qué Aprendimos?

- Entendemos la importancia de escanear imágenes en busca de vulnerabilidades.
- Ahora sabemos cómo utilizar Multi-Stage builds para optimizar el tamaño de las imágenes.

9 Actividad Práctica

9.1 Objetivo:

Aplicar buenas prácticas y medidas de seguridad al trabajar con Docker.

9.2 Instrucciones:

- Escanea la imagen “alpine:latest” en busca de vulnerabilidades utilizando la herramienta Snyk.
- Implementa una política de etiquetado para las imágenes Docker que siga las mejores prácticas.
- Utiliza la herramienta “docker-compose lint” para verificar la validez del archivo Docker Compose.
- Implementa una red de contenedores y asegúrate de que solo los contenedores necesarios tengan acceso a ella.
- Crea un archivo “.dockerignore” para excluir archivos y directorios innecesarios en la construcción de imágenes Docker.

9.3 Entregables:

- Capturas de pantalla del escaneo de vulnerabilidades.
- Política de etiquetado para imágenes Docker.
- Resultado de la verificación del archivo Docker Compose.
- Documento explicando la implementación de la red de contenedores.
- Archivo “.dockerignore”.

9.4 Rubrica de Evaluación:

- Escaneo de vulnerabilidades realizado con éxito: 4 puntos
- Correcta implementación de la política de etiquetado: 4 puntos
- Validación exitosa del archivo Docker Compose: 4 puntos
- Implementación adecuada de la red de contenedores: 4 puntos
- Correcta configuración del archivo “.dockerignore”: 4 puntos

Resolución de la Actividad Práctica

Escaneo de vulnerabilidades:

```
snyk container test alpine:latest
```

Política de etiquetado (ejemplo):

Se etiquetarán las imágenes con el formato "versión-año-mes-día" (ejemplo: 1.0-20230115).

Verificación del archivo Docker Compose:

```
docker-compose config
```

Implementación de una red de contenedores:

```
docker network create my-network
```

Archivo ".dockerignore" (ejemplo):

```
node_modules
.git
.env
```

10 5. DevContainers

10.1 Conceptos:

DevContainers: DevContainers es una característica de Visual Studio Code que permite definir, configurar y compartir fácilmente entornos de desarrollo con contenedores de Docker. Con DevContainers, puedes especificar las herramientas, extensiones y configuraciones necesarias para tu proyecto, garantizando que todos los miembros del equipo utilicen el mismo entorno de desarrollo, independientemente de su sistema operativo.

Stack MEAN: El stack MEAN (MongoDB, Express.js, Angular, Node.js) es un conjunto de tecnologías ampliamente utilizado para el desarrollo web moderno. MongoDB es una base de datos NoSQL, Express.js es un marco web para Node.js, Angular es un framework para construir aplicaciones web de una sola página (SPA), y Node.js es un entorno de ejecución para JavaScript en el lado del servidor.

10.2 Ejemplos:

Configuración de DevContainers para el stack MEAN:

Creación del archivo .devcontainer/devcontainer.json:

```
{
  "name": "MEAN Stack",
  "dockerComposeFile": "docker-compose.yml",
  "service": "app",
  "workspaceFolder": "/workspace",
  "extensions": [
    "ms-vscode.node-debug2",
    "dbaeumer.vscode-eslint"
  ],
  "postCreateCommand": "npm install"
}
```

Creación del archivo docker-compose.yml:

```
version: '3'
services:
  app:
    image: node:14
    command: /bin/sh -c "while sleep 1000; do :; done"
```

```
    volumes:
      - ..:/workspace
```

Estructura del proyecto:

```
.
   .devcontainer
      devcontainer.json
   docker-compose.yml
   src
       app.js
```

Archivo app.js (Ejemplo de aplicación Node.js):

```
const express = require('express');
const app = express();
const port = 3000;

app.get('/', (req, res) => res.send('Hola desde Express.js en un contenedor Docker!'));

app.listen(port, () => console.log(`Aplicación escuchando en http://localhost:${port}`)
```

10.3 Actividad Práctica:

Configurar un entorno de desarrollo MEAN con DevContainers en Visual Studio Code.

- Clona un repositorio base que contenga la estructura mencionada y el código de ejemplo.
- Abre el proyecto en Visual Studio Code.
- Visualiza y comprende los archivos .devcontainer/devcontainer.json y docker-compose.yml.
- Inicia el entorno de desarrollo con DevContainers.
- Accede a la aplicación en tu navegador utilizando el puerto especificado en el archivo app.js.
- Realiza modificaciones en el código de la aplicación y observa cómo se reflejan en tiempo real dentro del contenedor.

Resolución de la Actividad Práctica

Clonar el repositorio base:

```
git clone https://ejemplo-repositorio-mean.git
```

Abrir el proyecto en Visual Studio Code:

```
code ejemplo-repositorio-mean
```

Iniciar el entorno de desarrollo con DevContainers:

Visual Studio Code detectará automáticamente la configuración de DevContainers y te preguntará si deseas reabrir el proyecto en un contenedor.

Acceder a la aplicación en el navegador:

Visita http://localhost:3000 en tu navegador.

Realizar modificaciones en el código:

Abre el archivo src/app.js y realiza cambios en el mensaje de respuesta.

Observa cómo los cambios se reflejan en tiempo real dentro del contenedor.

> Tip
>
> Si experimentas problemas con la integración de DevContainers, asegúrate de tener Docker instalado y la extensión "Remote - Containers" habilitada en Visual Studio Code.
> Además, verifica que tu sistema cumple con los requisitos para DevContainers.

10.4 ¿Qué Aprendimos?

- Aprendimos a configurar DevContainers en Visual Studio Code para un entorno de desarrollo MEAN.
- Comprendimos cómo utilizar Docker Compose junto con DevContainers para definir la infraestructura del entorno de desarrollo.
- Exploramos la posibilidad de realizar cambios en el código de la aplicación de manera eficiente gracias a la integración de DevContainers con Visual Studio Code.

11 Actividad Práctica

11.1 Objetivo:

Aplicar buenas prácticas y medidas de seguridad al trabajar con Docker.

11.2 Instrucciones:

- Escanea la imagen “alpine:latest” en busca de vulnerabilidades utilizando la herramienta Snyk.
- Implementa una política de etiquetado para las imágenes Docker que siga las mejores prácticas.
- Utiliza la herramienta “docker-compose lint” para verificar la validez del archivo Docker Compose.
- Implementa una red de contenedores y asegúrate de que solo los contenedores necesarios tengan acceso a ella.
- Crea un archivo “.dockerignore” para excluir archivos y directorios innecesarios en la construcción de imágenes Docker.

11.3 Entregables:

- Capturas de pantalla del escaneo de vulnerabilidades.
- Política de etiquetado para imágenes Docker.
- Resultado de la verificación del archivo Docker Compose.
- Documento explicando la implementación de la red de contenedores.
- Archivo “.dockerignore”.

11.4 Rubrica de Evaluación:

- Escaneo de vulnerabilidades realizado con éxito: 4 puntos
- Correcta implementación de la política de etiquetado: 4 puntos
- Validación exitosa del archivo Docker Compose: 4 puntos
- Implementación adecuada de la red de contenedores: 4 puntos
- Correcta configuración del archivo “.dockerignore”: 4 puntos

Resolución de la Actividad Práctica

Escaneo de vulnerabilidades:

```
snyk container test alpine:latest
```

Política de etiquetado (ejemplo):

Se etiquetarán las imágenes con el formato "versión-año-mes-día" (ejemplo: 1.0-20230115).

Verificación del archivo Docker Compose:

```
docker-compose config
```

Implementación de una red de contenedores:

```
docker network create my-network
```

Archivo ".dockerignore" (ejemplo):

```
node_modules
.git
.env
```