

UNIVERSIDAD INTERNACIONAL DE LA RIOJA (UNIR)

La Universidad en Internet

Contenedores

(MEXDEVOPS)

**Maestría en Desarrollo y Operaciones de Software
2025**

Actividad 1: Creación de aplicativos en contenedores
con diferentes lenguajes de programación

Luis Enrique Méndez Cantero

Índice

Creación de aplicativos en contenedores con diferentes lenguajes de programación	4
Objetivos	4
Creación de una contenedor con Ruby	4
Aplicación básica	4
Dockerfile	5
Creación de la imagen Ruby	5
Ejecución de la imagen Ruby	6
Publicación de la imagen Ruby	6
Creación de un contenedor con PHP	7
Aplicación básica.....	7
Dockerfile	7
Creación de la imagen PHP	7
Ejecución de la imagen PHP	8
Publicación de la imagen PHP	8
Aplicación básica.....	8
Dockerfile	9
Creación de la imagen Perl.....	9
Ejecución de la imagen Perl.....	9
Publicación de la imagen Perl.....	10
Creación de un contenedor con Elixir	10
Aplicación básica.....	10
Dockerfile	10
Creación de la imagen Elixir	10
Ejecución de la imagen Elixir	11
Publicación de la imagen Elixir	11

Creación de una aplicación con Docker Compose	12
Servicio db: Base de datos Postgres.....	13
Servicio backend: API escrita en Go.....	13
Dockerfile	13
Servicio proxy: Servidor Nginx.....	13
Implementar con Docker Compose	13
Parar y remover los contenedores	14
Conclusión	15
Referencias	15

Creación de aplicativos en contenedores con diferentes lenguajes de programación

Objetivos

- Aplicar los conocimientos adquiridos sobre contenedores mediante la creación de aplicaciones en diferentes lenguajes de programación.
- Desarrollar imágenes personalizadas de Docker para 4 diferentes lenguajes de programación.
- Publicar las imágenes a un repositorio remoto como parte del flujo de trabajo DevOps.
- Utilizar Docker Compose para implementar una aplicación multicontenedor.

Creación de una contenedor con Ruby

Aplicación básica

Para la creación de esta imagen primero crearemos el directorio donde se encontrarán y ejecutarán los archivos y comandos necesarios.

Nombre: ruby-app

Luego se creará el archivo de la aplicación básica que sólo mostrará un saludo.

Nombre: app.rb

Este será un servidor web mínimo que usa Sinatra, un framework simple en Ruby.

En el caso de esta aplicación de Ruby también necesitamos el archivo Gemfile el cual le dice a Ruby qué dependencias usar.

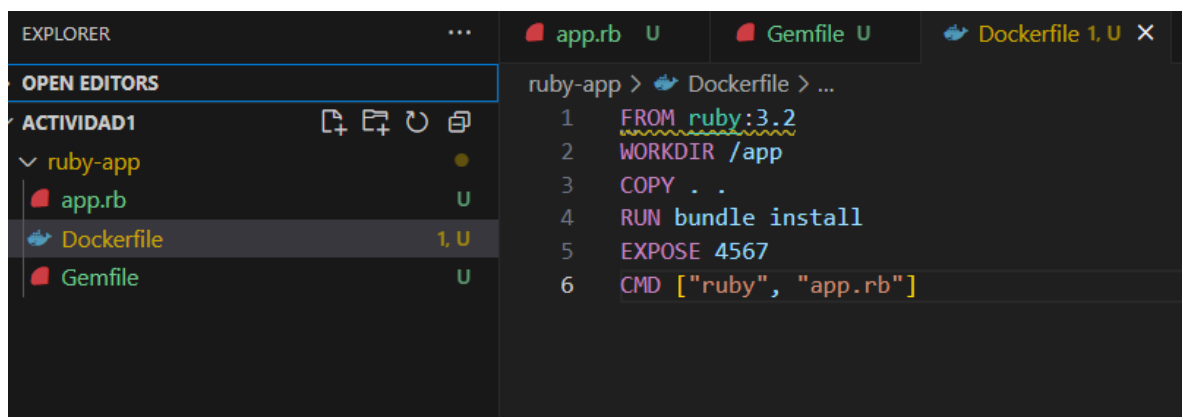
Nombre: Gemfile

Dockerfile

Para la creación de la imagen será necesario un Dockerfile para poder construir la imagen paso por paso.

Nombre: Dockerfile

Dentro del archivo se utiliza la versión ruby:3.2, se hace un cambio al directorio de trabajo /app y después se hace una copia de todos los archivos del directorio local hacia el WORKDIR definido en el paso anterior, luego se instalan las dependencias necesarias del Gemfile, se expone el puerto 4567 y se ejecuta el comando ruby app.rb para poder iniciar la aplicación.

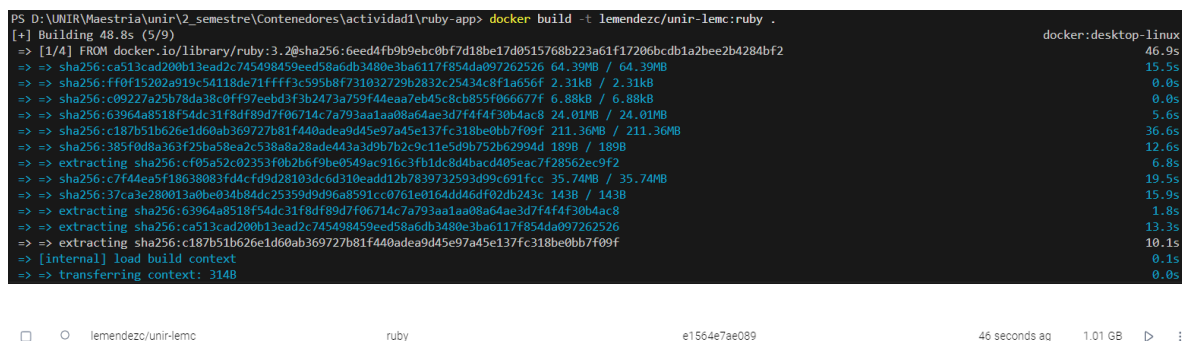


```
EXPLORER
...
OPEN EDITORS
ACTIVIDAD1
  ruby-app
    app.rb
    Dockerfile
    Gemfile
ruby-app > Dockerfile > ...
1 FROM ruby:3.2
2 WORKDIR /app
3 COPY . .
4 RUN bundle install
5 EXPOSE 4567
6 CMD ["ruby", "app.rb"]
```

Creación de la imagen Ruby

Para iniciar el proceso de construcción de la imagen es necesario usar el comando docker build, pero en este caso también queremos asignar una tag a la imagen, esto con el fin de poder hacer uso de mi repositorio de Docker Hub el comando final se vería así:

ruby-app> docker build -t lemenдец/unir-lemc:ruby .



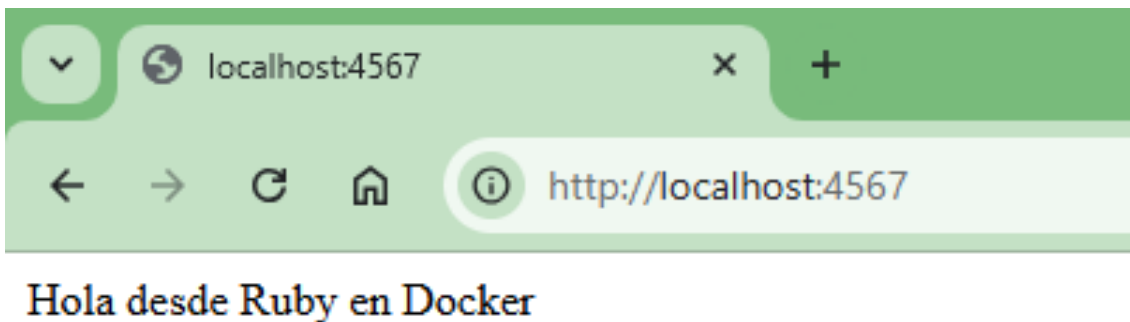
```
PS D:\UNIR\Maestria\unir\2_semestre\Contenedores\actividad1\ruby-app> docker build -t lemenдец/unir-lemc:ruby .
[+] Building 48.8s (5/9)
=> [1/4] FROM docker.io/library/ruby:3.2@sha256:6eed4fb9b9ebc0bf7d18be17d0515768b223a61f17206bcd1a2bee2b4284bf2
=> sha256:ca513cad200b13ead2c745498459eed58a6db3480e3ba6117f854da097262526 64.39MB / 64.39MB
=> sha256:ff0f15202a919c54118de71ffff3c595b8f731032729b2832c25434c8f1a656f 2.31kB / 2.31kB
=> sha256:c09227a25b78da38c0ff97eebd3f3b2473a759f44eaa7eb45c8cb855f066677f 6.88kB / 6.88kB
=> sha256:63964a8518f54dc31f8df89d7f06714c7a793aa1aa08a64ae3d7f4f4f30b4ac8 24.01MB / 24.01MB
=> sha256:c187b51b626e1d60ab369727b81f440adea9d45e97a45e137fc318be0bb7f09f 211.36MB / 211.36MB
=> sha256:385f0d8a363f25ba58ea2c538a8a28ade443a3d9b7b2c9c11e5d9b752b62994d 189B / 189B
=> extracting sha256:cf05a52c02353f0b2b6f9be9549ac916c3fb1dc8d4bacd405eac7f28562ec9f2 6.8s
=> sha256:c7f44ea5f18638083fd4cf9d28103dc6d310eadd12b7839732593d99c691fcc 35.74MB / 35.74MB
=> sha256:37ca3e280013a0be034b84dc25359d9d96a8591cc0761e0164dd46df02db243c 143B / 143B
=> extracting sha256:63964a8518f54dc31f8df89d7f06714c7a793aa1aa08a64ae3d7f4f4f30b4ac8 1.8s
=> extracting sha256:ca513cad200b13ead2c745498459eed58a6db3480e3ba6117f854da097262526 13.3s
=> extracting sha256:c187b51b626e1d60ab369727b81f440adea9d45e97a45e137fc318be0bb7f09f 10.1s
=> [internal] load build context
=> transferring context: 314B
```

Ejecución de la imagen Ruby

Para correr un contenedor basado en mi imagen podemos usar el comando `docker run`, para no quedar “bloqueados” ejecutaremos el comando en modo detached (en segundo plano), y finalmente mapearemos el puerto local al del contenedor. El comando se vería así:

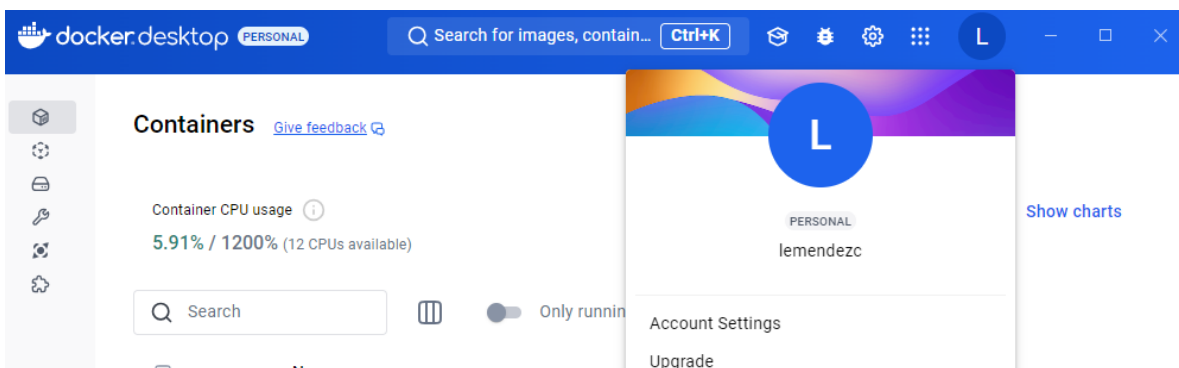
```
ruby-app> docker run -d -p 4567:4567 lemendezc/unir-lemc:ruby
```

Para comprobar que funciona podemos usar el navegador que queramos (Chrome en mi caso) y acceder a la siguiente url `http://localhost:4567`



Publicación de la imagen Ruby

Para subir la imagen en Docker Hub usaremos el comando `docker push`, con el nombre exacto de la imagen que subiremos. (Es importante iniciar sesión en la aplicación de Docker Desktop)



El comando para publicar la imagen es el siguiente:

```
ruby-app> docker push lemendezc/unir-lemc:ruby
```

```
PS D:\UNIR\Maestria\unir\2_semestre\Contenedores\actividad1\ruby-app> docker push lemendezc/unir-lemc:ruby
The push refers to repository [docker.io/lemendezc/unir-lemc]
bfc1f06bb5bd: Pushed
d7342ae65fee: Pushed
5d457769caf4: Pushed
0bd41cf713d9: Mounted from library/ruby
c6735347c80a: Mounted from library/ruby
e4950f7b4f89: Mounted from library/ruby
bf9c09fb6f3a: Mounted from library/ruby
fcbb8c0ae5d6: Mounted from library/ruby
8ce3e08e661a: Mounted from library/ruby
247fffb7158d: Mounted from library/ruby
ruby: digest: sha256:cc1137d07ff41e557e2e105c015c84799858d97ade0d6ff3da4e07da1237b3cc size: 2416
```

Creación de un contenedor con PHP

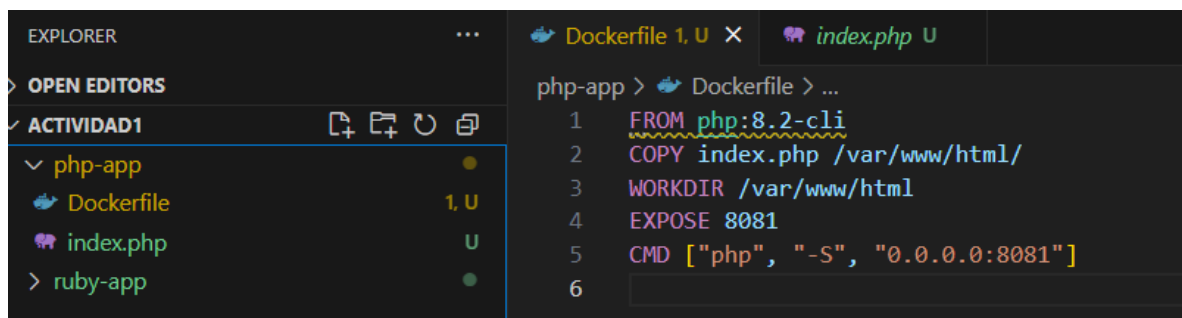
Aplicación básica

Nombre del directorio: php-app

Aplicación básica que solo mostrará un saludo: index.php

Dockerfile

Dentro del archivo se utiliza la imagen oficial de PHP versión 8.2-cli, después se copia el archivo index.php al directorio /var/www/html del contenedor. Luego se establece ese mismo directorio como WORKDIR, se expone el puerto 8081 y se ejecuta el servidor embebido de PHP para atender peticiones en todas las interfaces (0.0.0.0).



The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane shows a project named 'php-app' containing a 'Dockerfile' and an 'index.php' file. The main editor area shows the 'Dockerfile' with the following content:

```
php-app > Dockerfile > ...
1 FROM php:8.2-cli
2 COPY index.php /var/www/html/
3 WORKDIR /var/www/html
4 EXPOSE 8081
5 CMD ["php", "-S", "0.0.0.0:8081"]
6
```

Creación de la imagen PHP

php-app> docker build -t lemendezc/unir-lemc:php .

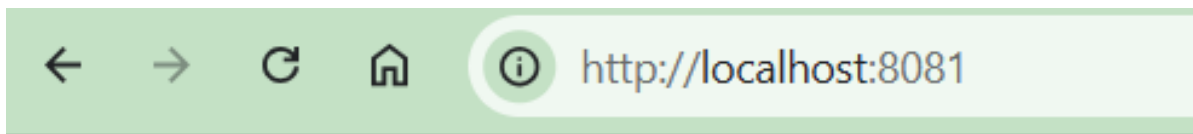
```
PS D:\UNIR\Maestria\unir\2_semestre\Contenedores\actividad1\php-app> docker build -t lemendezc/unir-lemc:php .
[*] Building 33.4s (9/9) FINISHED
[+] Building 33.4s (9/9) FINISHED
=> [internal] load build definition from Dockerfile
=> => transferring dockerfile: 445B
=> [internal] load metadata for docker.io/library/php:8.2-cli
=> [auth] library/php:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [internal] load build context
=> => transferring context: 79B
=> [1/3] FROM docker.io/library/php:8.2-cli@sha256:ed4385b854a7ef4aeee1108c75333443d64c937faaf7c7d28bf63a436df06428
=> => resolve docker.io/library/php:8.2-cli@sha256:ed4385b854a7ef4aeee1108c75333443d64c937faaf7c7d28bf63a436df06428
=> => sha256:8dd9d3fbc7dc7d907d4d90e1dd09f5aef2b3144f7973a8de1d5153181481908 2.69kB / 2.69kB
=> => sha256:254e724d77862dc53abbd3bf0e27f9d2f64293909cdd3d0aad6a8fe5a6680659 28.23MB / 28.23MB
=> => sha256:ed4385b854a7ef4aeee1108c75333443d64c937faaf7c7d28bf63a436df06428 10.35kB / 10.35kB
=> => sha256:33b6dc95c47acfc8dc9a08124d9012a97a909d779b08bb9549e7beede43e5055 8.42kB / 8.42kB
```

Ejecución de la imagen PHP

php-app> docker run -d -p 8081:8081 lemendezc/unir-lemc:php

```
PS D:\UNIR\Maestria\unir\2_semestre\Contenedores\actividad1\php-app> docker run -d -p 8081:8081 lemendezc/unir-lemc:php
dbff4d7183dc962222551f3e2b29d1adeab07146090c9ebc81a733ec53583065
```

Comprobación de que funciona correctamente abriremos un navegador y accederemos a la siguiente dirección <http://localhost:8081>



Hola desde PHP en Docker

Publicación de la imagen PHP

php-app> docker push lemendezc/unir-lemc:php

```
PS D:\UNIR\Maestria\unir\2_semestre\Contenedores\actividad1\php-app> docker push lemendezc/unir-lemc:php
The push refers to repository [docker.io/lemendezc/unir-lemc]
5f70bf18a086: Pushed
4ea183575435: Pushed
9bb3450f825a: Mounted from library/php
e7f464f96498: Mounted from library/php
ee543aa70305: Mounted from library/php
a286b4514a65: Mounted from library/php
e94c9e736c38: Mounted from library/php
2f4a7818e002: Mounted from library/php
2d0d47490161: Mounted from library/php
75550490b486: Mounted from library/php
6c4c763d22d0: Mounted from library/php
php: digest: sha256:cd2825b83d147161ab6824d34a4c20c4f48cd0fe67b28e63b4076291c7183ecc size: 2615
PS D:\UNIR\Maestria\unir\2_semestre\Contenedores\actividad1\php-app>
```

Creación de un contenedor con Perl

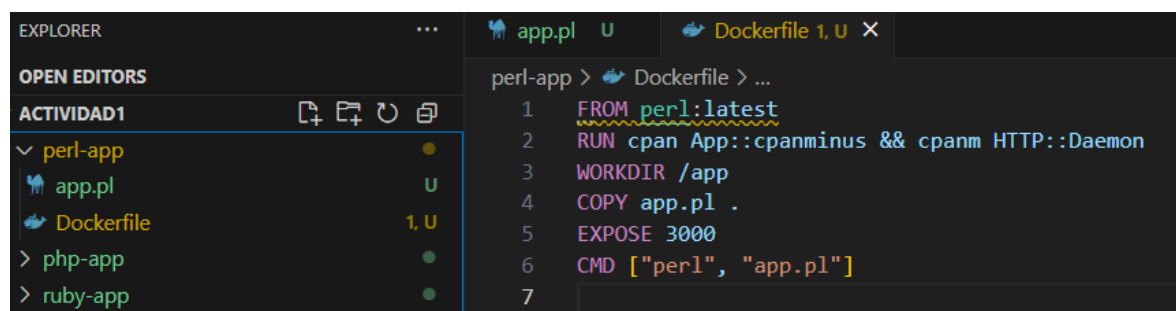
Aplicación básica

Nombre del directorio: perl-app

Aplicación básica que solo mostrará un saludo: app.pl

Dockerfile

Dentro del archivo se utiliza la imagen oficial de Perl. Luego se instalan los módulos necesarios mediante cpan y cpanm, en especial Dancer2. Posteriormente, se copia el archivo de la aplicación al directorio /app del contenedor, se expone el puerto 3000 y finalmente se ejecuta el archivo app.pl como punto de entrada de la aplicación.

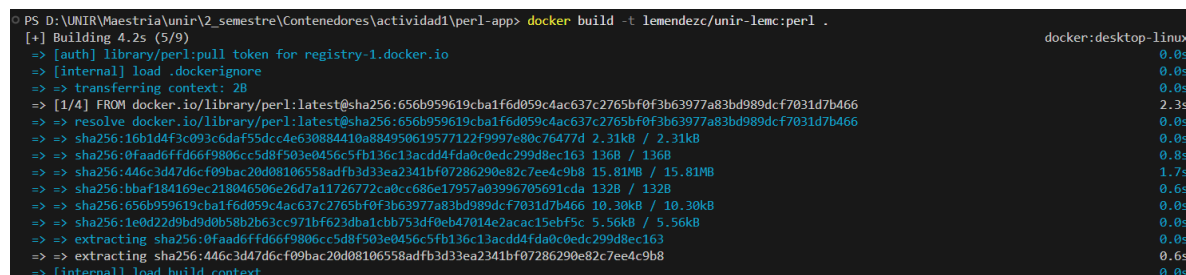


The screenshot shows the Visual Studio Code interface. On the left, the Explorer pane shows a project named 'perl-app' with files 'app.pl' and 'Dockerfile'. The Dockerfile is open in the editor, showing the following content:

```
1 FROM perl:latest
2 RUN cpan App::cpanminus && cpanm HTTP::Daemon
3 WORKDIR /app
4 COPY app.pl .
5 EXPOSE 3000
6 CMD ["perl", "app.pl"]
```

Creación de la imagen Perl

perl-app> docker build -t lemenandezc/unir-lemc:perl .

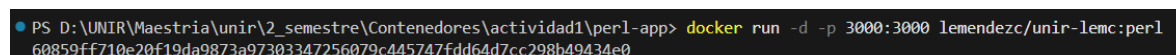


The screenshot shows a terminal window with the following output:

```
PS D:\UNIR\Maestria\unir\2_semestre\Contenedores\actividad1\perl-app> docker build -t lemenandezc/unir-lemc:perl .
[+] Building 4.2s (5/9)
=> [auth] library/perl:pull token for registry-1.docker.io
=> [internal] load .dockerignore
=> => transferring context: 2B
=> [1/4] FROM docker.io/library/perl:latest@sha256:656b959619c8a1f6d059c4ac637c2765bf0f3b63977a83bd989dcf7031d7b466
=> => resolve docker.io/library/perl:latest@sha256:656b959619c8a1f6d059c4ac637c2765bf0f3b63977a83bd989dcf7031d7b466
=> => sha256:16b1d4f3c093c6daf55d4e630884410a884950619577122f9997e80c76477d 2.31kB / 2.31kB
=> => sha256:0faad6ffdf66f9806cc5d8f503e0456c5fb136c13acdd4fda0c0edc299d8ec163 136B / 136B
=> => sha256:446c3d47d6cf09bac20d08106558adfb3d33ea2341bf07286290e82c7ee4c9b8 15.81MB / 15.81MB
=> => sha256:bbaf1847d6ec218046506e26d7a11726772ca0cc686e17957a03996705691cda 132B / 132B
=> => sha256:656b959619c8a1f6d059c4ac637c2765bf0f3b63977a83bd989dcf7031d7b466 10.30kB / 10.30kB
=> => sha256:1e0d22d9bd9d0b58b2b63cc971bf623d8a1cbb753df0eb47014e2acac15ebf5c 5.56kB / 5.56kB
=> => extracting sha256:0faad6ffdf66f9806cc5d8f503e0456c5fb136c13acdd4fda0c0edc299d8ec163
=> => extracting sha256:446c3d47d6cf09bac20d08106558adfb3d33ea2341bf07286290e82c7ee4c9b8
=> [internal] load build context
```

Ejecución de la imagen Perl

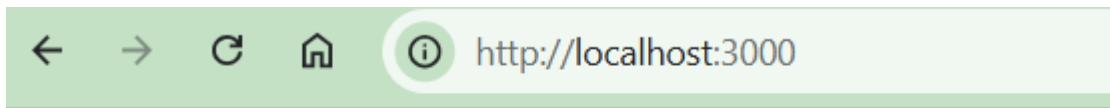
perl-app> docker run -d -p 3000:3000 lemenandezc/unir-lemc:perl



The screenshot shows a terminal window with the following output:

```
PS D:\UNIR\Maestria\unir\2_semestre\Contenedores\actividad1\perl-app> docker run -d -p 3000:3000 lemenandezc/unir-lemc:perl
60859ff710e20f19da9873a97303347256079c445747fdd64d7cc298b49434e0
```

Comprobación de que funciona correctamente abriremos un navegador y accederemos a la siguiente dirección <http://localhost:3000>



Hola desde Perl en Docker

Publicación de la imagen Perl

perl-app> docker push lemendezc/unir-lemc:perl

```
PS D:\UNIR\Maestria\unir\2_semestre\Contenedores\actividad1\perl-app> docker push lemendezc/unir-lemc:perl
The push refers to repository [docker.io/lemendezc/unir-lemc]
01e081489780: Pushed
f010f5428a24: Pushed
b4f1bfab5832: Pushed
dcb199bbb40f: Mounted from library/perl
f71c51bfc705: Mounted from library/perl
4a649d5ce4a0: Mounted from library/perl
bf9c09fb6f3a: Layer already exists
fcbb8c0ae5d6: Layer already exists
8ce3e08e661a: Layer already exists
247fffb7158d: Layer already exists
perl: digest: sha256:182401a310ec4cee949525042e1d5115001aac141669468608377834a28ffac5 size: 2417
```

Creación de un contenedor con Elixir

Aplicación básica

Nombre del directorio: elixir-app

Aplicación básica que solo mostrará un saludo: hello_web.ex

Archivos extra necesarios: mis.exs, /lib/hello_web/router

Dockerfile

En este contenedor se ha desarrollado una aplicación web básica utilizando Plug y Cowboy, dos bibliotecas ligeras y ampliamente utilizadas dentro del ecosistema Elixir. Se expondrá su ejecución en el puerto 4000.

Creación de la imagen Elixir





elixir-app> docker build -t lemendezc/unir-lemc:elixir .

Tags

 DOCKER SCOUT INACTIVE

[Activate](#)

This repository contains 3 tag(s).

Tag	OS	Type	Pulled	Pushed
 elixir		Image	less than 1 day	less than a minute
 perl		Image	less than 1 day	19 minutes
 php		Image	less than 1 day	about 1 hour
 ruby		Image	less than 1 day	about 1 hour

Creación de una aplicación con Docker Compose

Haciendo uso de los ejemplos proporcionados en <https://github.com/docker/awesome-compose/tree/master> podemos hacer uso de uno de los ejemplos de aplicaciones creadas con diferentes contenedores gracias a Docker Compose, como por ejemplo:

Nginx + Golang + Postgres

Estructura del proyecto:

```
.
├── backend
│   ├── Dockerfile
│   ├── go.mod
│   ├── go.sum
│   └── main.go
├── db
│   └── password.txt
├── compose.yaml
├── proxy
└── └── nginx.conf
```

La aplicación está compuesta por tres servicios principales que están definidos en el archivo `compose.yaml`:

1. db (base de datos)
2. backend (servidor de aplicación en Go)
3. proxy (servidor web con Nginx)

Servicio db: Base de datos Postgres

Este servicio usa la imagen oficial de PostgreSQL el cual configura una base de datos llamada example y cuya contraseña del usuario postgres se lee desde un secreto (/run/secrets/db-password) para mayor seguridad. Se monta un volumen (db-data) para persistir los datos, incluso si el contenedor se reinicia o se destruye, es importante utilizar healthcheck para verificar que la base de datos esté lista antes de que otros servicios puedan conectarse a ella.

Servicio backend: API escrita en Go

Se construye a partir de un Dockerfile, y hace uso de mux como router HTTP y se conecta a PostgreSQL usando el driver lib/pq. El backend lee la contraseña de la base de datos desde el secreto (/run/secrets/db-password) y expone un endpoint HTTP en el puerto 8000. Al iniciar, ejecuta la función prepare() que espera a que la base de datos esté lista, crea la tabla blog si no existe e inserta 5 entradas de ejemplo: "Blog post #0" hasta "Blog post #4".

Dockerfile

Usa una imagen base de Alpine con Go1.18 y se hace uso de un build stage optimizado con caché para descargas de módulos Go.

Servicio proxy: Servidor Nginx

Este servicio usa la imagen oficial de Nginx el cual monta una configuración personalizada (proxy/nginx.conf) donde se redirige todas las solicitudes del puerto 80 del host hacia el servicio backend en el puerto 8000, esto permite acceder a la aplicación desde un navegador simplemente entrando a <http://localhost>

Implementar con Docker Compose

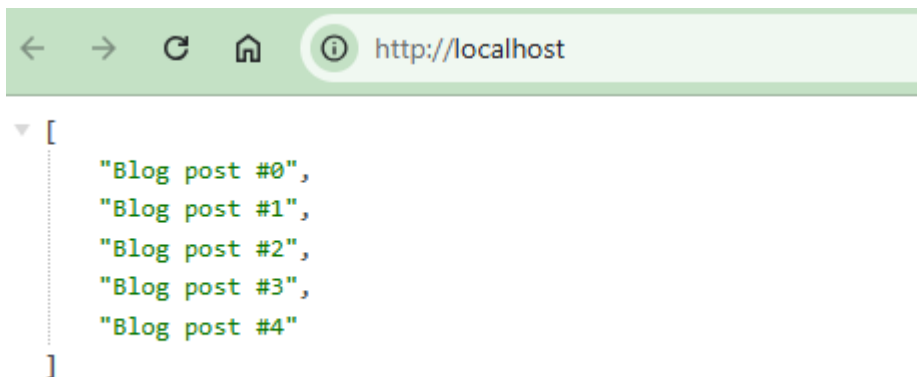
Para poder crear la aplicación haremos uso del comando docker compose up -d dentro del directorio donde se encuentra nuestro compose.yml.

```
PS D:\UNIR\Maestria\unir\2_semestre\Contenedores\actividad1\nginx-golang-postgres> docker compose up -d
[+] Running 17/2
  - proxy [██████████] 44MB / 44.16MB Pulling 15.5s
  - db [██████████] 104.6MB / 128.1MB Pulling 15.5s
```

Resultados:

<input type="checkbox"/>	<input checked="" type="checkbox"/>	nginx-golang-postgres	-	-	-	0.03%	45 seconds ago			
<input type="checkbox"/>	<input checked="" type="checkbox"/>	proxy-1	0764ce6cc643	nginx	80:80	0%	45 seconds ago			
<input type="checkbox"/>	<input checked="" type="checkbox"/>	backend-1	a08e08b42acb	nginx-golang-postgres-backend		0%	45 seconds ago			
<input type="checkbox"/>	<input checked="" type="checkbox"/>	db-1	41a4685a32d3	postgres		0.03%	56 seconds ago			

Podemos ver que los contenedores están agrupados con el nombre del directorio nginx-golang-postgres y dentro se encuentran los 3 servicios (contenedores) ahora solo queda comprobar que todo funciona correctamente accediendo a través de un navegador a la ruta `http://localhost`:



También podemos comprobar el estado de los procesos con el siguiente comando:

```
nginx-golang-postgres> docker compose ps
```

```
PS D:\UNIR\Maestria\unir\2_semestre\Contenedores\actividad1\nginx-golang-postgres> docker compose ps
```

NAME	IMAGE	COMMAND	SERVICE	CREATED	STATUS	PORTS
nginx-golang-postgres-backend-1	nginx-golang-postgres-backend	"/code/bin/backend"	backend	2 minutes ago	Up 2 minutes	
nginx-golang-postgres-db-1	postgres	"docker-entrypoint.s..."	db	2 minutes ago	Up 2 minutes (healthy)	5432/tcp
nginx-golang-postgres-proxy-1	nginx	"docker-entrypoint..."	proxy	2 minutes ago	Up 2 minutes	0.0.0.0:80->80/tcp

Parar y remover los contenedores

Para finalizar con este laboratorio podemos hacer uso del siguiente comando para parar y remover los contenedores creados previamente:

```
nginx-golang-postgres> docker compose down
```

```
PS D:\UNIR\Maestria\unir\2_semestre\Contenedores\actividad1\nginx-golang-postgres> docker compose down
[+] Running 4/4
✔ Container nginx-golang-postgres-proxy-1 Removed 0.7s
✔ Container nginx-golang-postgres-backend-1 Removed 0.4s
✔ Container nginx-golang-postgres-db-1 Removed 0.7s
✔ Network nginx-golang-postgres_default Removed 0.3s
```

Conclusión

La actividad permitió explorar el poder de Docker como herramienta para crear aplicaciones con contenedores en distintos lenguajes. A través del desarrollo y publicación de imágenes propias en Docker Hub, se comprendió el ciclo completo de construcción, prueba y distribución de contenedores. Además, el uso de Docker Compose para levantar una aplicación multicontenedor demostró como integrar múltiples servicios con facilidad, favoreciendo la modularidad, escalabilidad y mantenibilidad de los sistemas.

Este ejercicio no solo refuerza habilidades técnicas, sino que también prepara el camino para automatizar despliegues en entornos más complejos y reales, alineándose con los principios de DevOps.

Referencias

- Docker Inc. (2025). *Official Images Documentation*. Docker Hub. <https://hub.docker.com/search?q=&type=image>
- Docker Inc. (2024). *Docker Compose Overview*. Docker Docs. <https://docs.docker.com/compose/>
- Phoenix Team. (2025.). *Plug and Cowboy – Minimal Elixir Web Development*. Phoenix Framework. <https://hexdocs.pm/plug/readme.html>
- The Perl Foundation. (2025). *Dancer2 Documentation*. <https://metacpan.org/pod/Dancer2>