|  |
| --- |
| **Guía de Laboratorio Docker** |
| Gerencia de Tecnología.  **Documento de Técnico.**  Elaborado por:  Julian Rivera-Pineda  Analista de Servicios  [jurivera@promerica.com.sv](mailto:jurivera@promerica.com.sv) |
| **Banco Promerica de El Salvador**  **San Salvador 28 de septiembre de 2022** |

Historial de revisiones

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fecha** | **Versión** | **Descripción** | **Autor** | **Revisado por** |
| 27/09/2022 | 1 | Creación de documento | Julian Rivera-Pineda |  |

Contenidos

[Objetivo General. 2](#_Toc115257325)

[Listado de Materiales a utilizar. 2](#_Toc115257326)

[Procedimiento. 3](#_Toc115257327)

[Parte 1. Configurando el ambiente Play with Docker 3](#_Toc115257328)

[Parte 2. Ejecutando Microservicio con Payara Micro 14](#_Toc115257329)

[Parte 3. Ejecutando Microservicio con Angular 20](#_Toc115257330)

[Parte 4. Ejecutando Microservicio con .Net 7 25](#_Toc115257331)

[Parte 5. Ejercicio de Desafío 1 27](#_Toc115257332)

# Objetivo General.

Conocer el ambiente de contenedores Docker y sus principales comandos para gestionar cargas de trabajo empleando el laboratorio online [Play with Docker](https://labs.play-with-docker.com/).

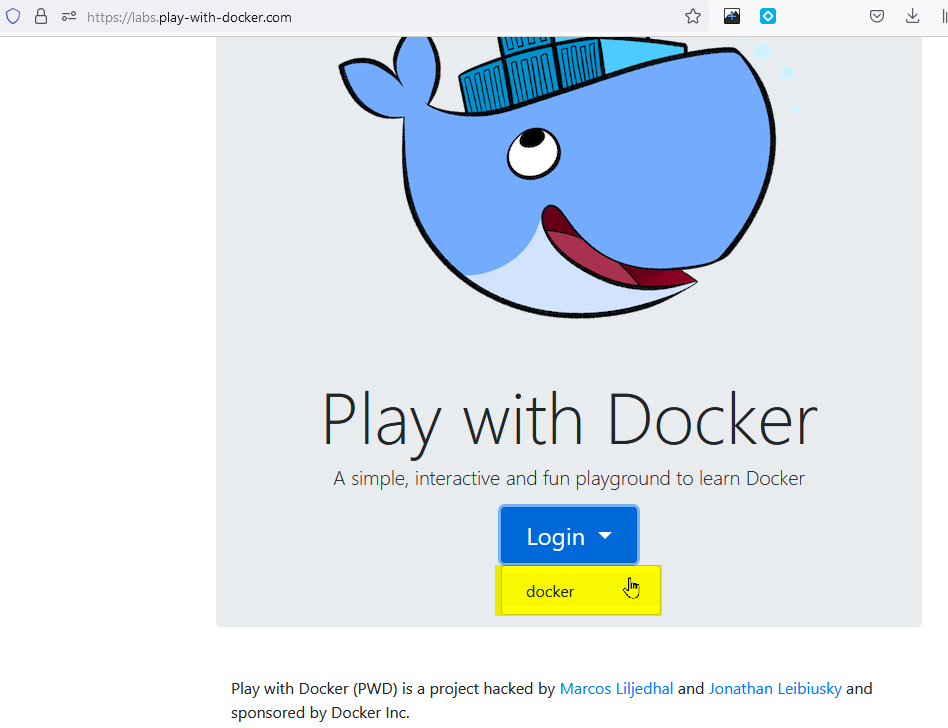
# Listado de Materiales a utilizar.

* Navegador de Internet (Firefox, Chrome o Brave)
* Guía de Laboratorio (Esta guía)
* Usuario de Docker hub (Registrarse en el siguiente enlace <https://hub.docker.com/signup> )

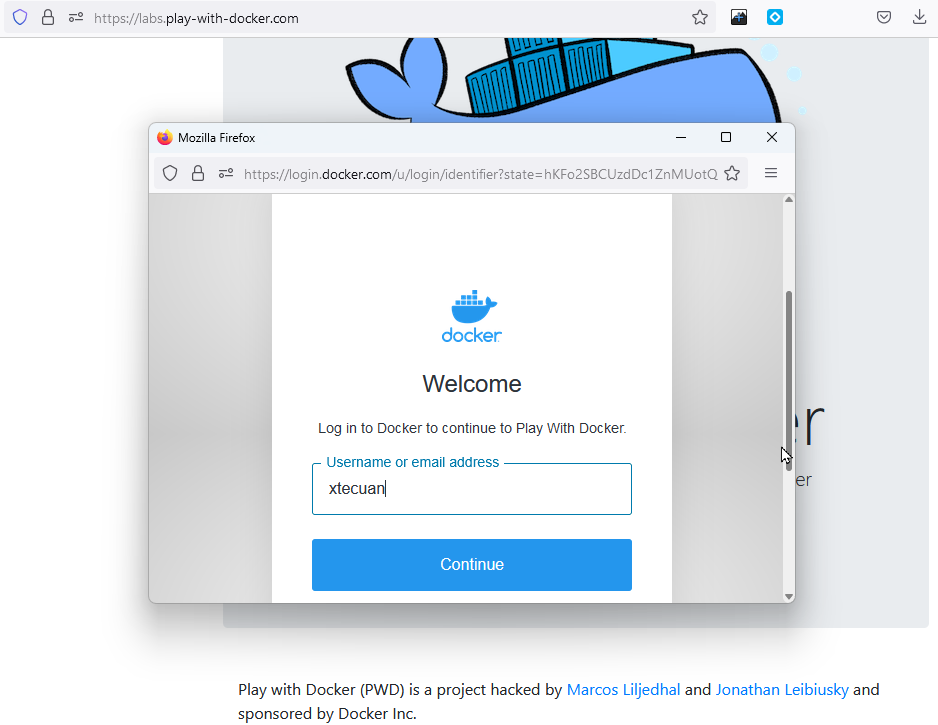
# Procedimiento.

## Parte 1. Configurando el ambiente Play with Docker

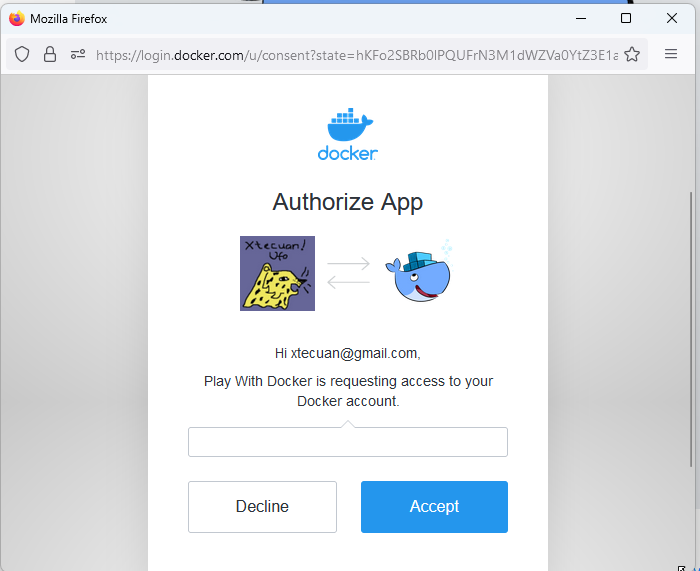
1. Ingresar a la página web de **Play with Docker** <https://labs.play-with-docker.com/> :



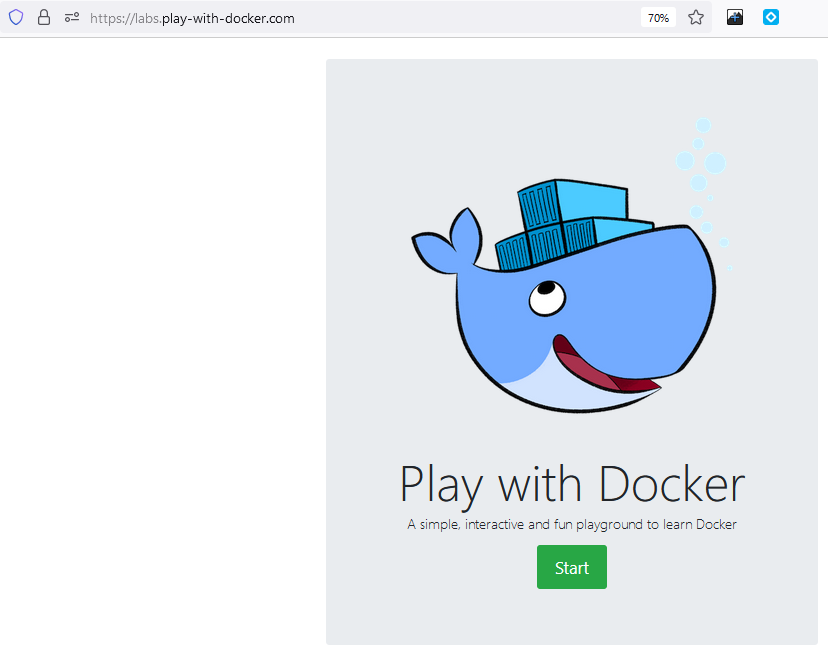
1. Seleccionar Login con docker (Se abrirá una ventana emergente, proceder a ingresar a Docker hub):



1. Autorizar a Play with Docker desde Docker Hub dando click en Accept:



1. Iniciar la sesión de Play with Docker dando click en Start:



1. Se iniciará la sesión que tiene tiempo límite de 4h:

Interfaz de usuario gráfica

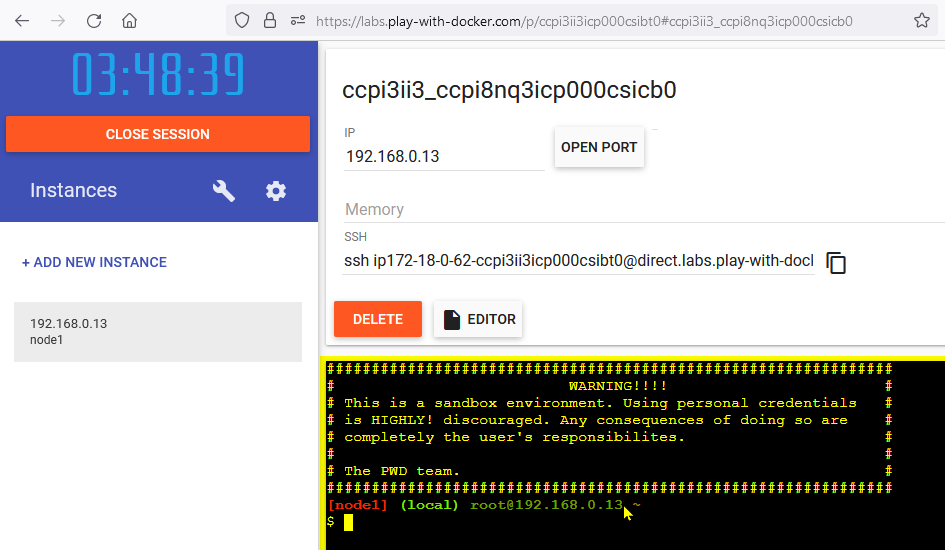
Descripción generada automáticamente con confianza baja

1. Presionar el enlace que dice ADD NEW INSTANCE:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. Aparecerá una terminal con el runtime de Docker listo para trabajar:

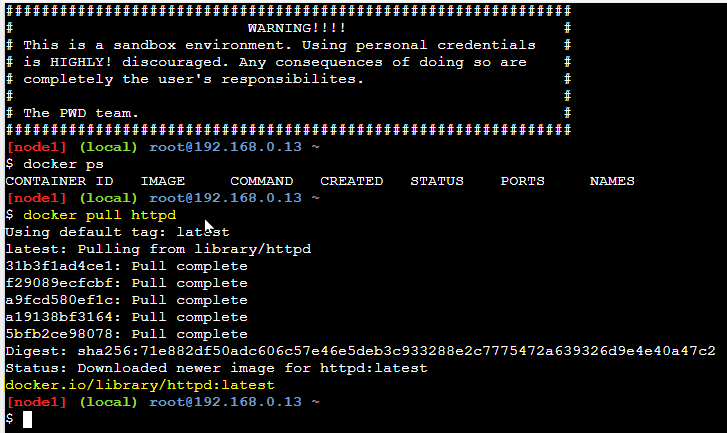


1. Verificar los contenedores que están siendo ejecutados con el comando << ***docker ps*** >>:

Pantalla de computadora con letras

Descripción generada automáticamente con confianza media

1. Como se puede ver al inicio la instancia de docker no muestra ningún contenedor ejecutándose. Hacer pull de una imagen de Apache HTTPD Server con el siguiente comando << ***docker pull httpd*** >>



1. Crear la carpeta de Projects << ***mkdir $HOME/projects*** >> donde crearemos nuestros archivos temporales para trabajar en este ambiente Docker, ingresar a la carpeta usando << ***cd $HOME/Projects*** >>:

Texto

Descripción generada automáticamente

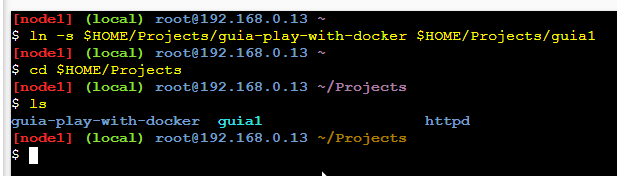
1. Clonar repositorio de la guía usando el comando:

<< ***git clone*** [***https://github.com/xtecuan/guia-play-with-docker.git***](https://github.com/xtecuan/guia-play-with-docker.git) >>

Texto

Descripción generada automáticamente

1. Crear un link simbólico para accesar rápidamente a los recursos del repositorio git con el siguiente comando << ***ln -s $HOME/Projects/guía-play-with-docker $HOME/Projects/guia1*** >>



1. Crear la carpeta httpd con el siguiente comando << ***mkdir $HOME/Projects/httpd*** >> y luego << ***cd $HOME/Projects/httpd*** >>:

Texto

Descripción generada automáticamente

1. Copiar el archivo **index.html** a la carpeta del contenedor httpd con el siguiente comando << ***cp $HOME/Projects/guia1/httpd/index.html $HOME/Projects/httpd/*** >> luego moverse a la carpeta httpd con el siguiente comando << ***cd $HOME/Projects/httpd*** >>

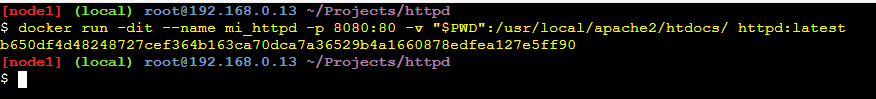
Texto

Descripción generada automáticamente

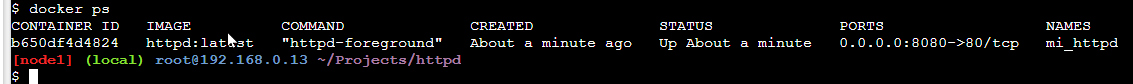
1. Ejecutar el contenedor **httpd** con el siguiente comando:

<<

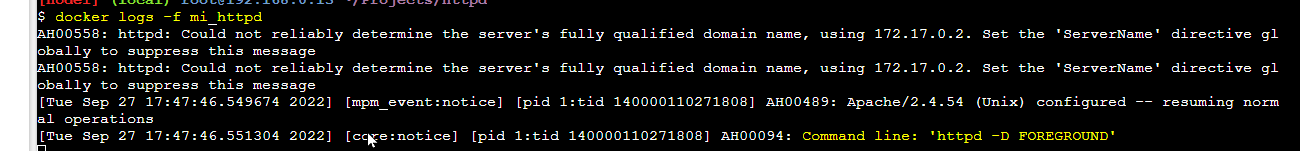
***docker run -dit --name mi\_httpd -p 8080:80 -v "$PWD":/usr/local/apache2/htdocs/ httpd:latest***

>>

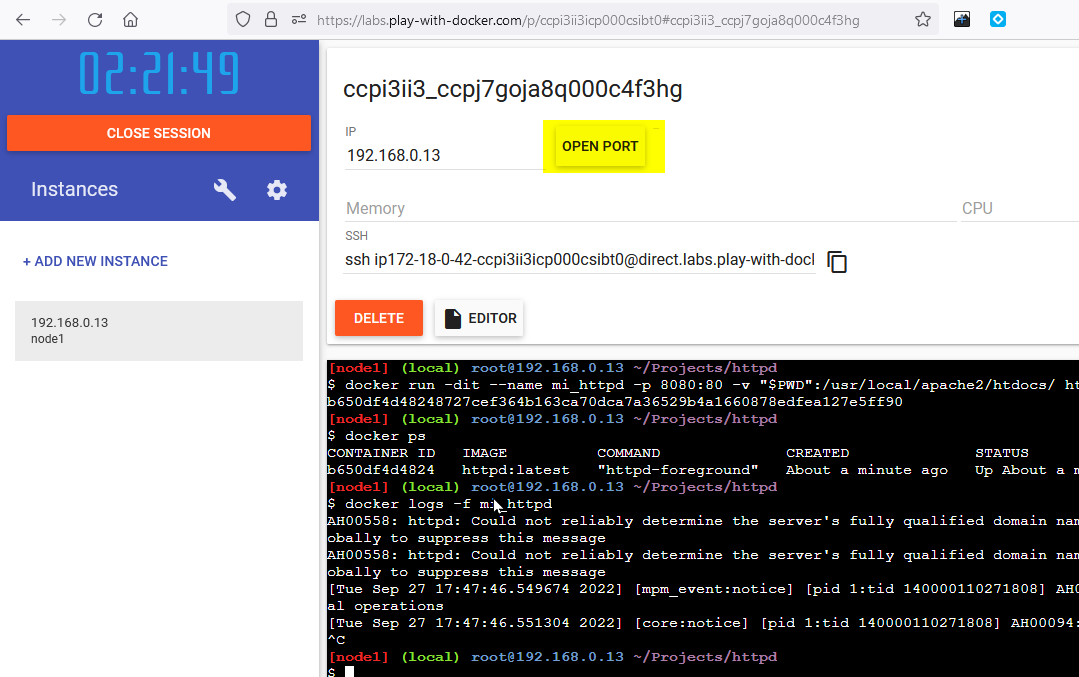
1. Hacer << ***docker ps*** >> para ver que el contenedor se está ejecutando:



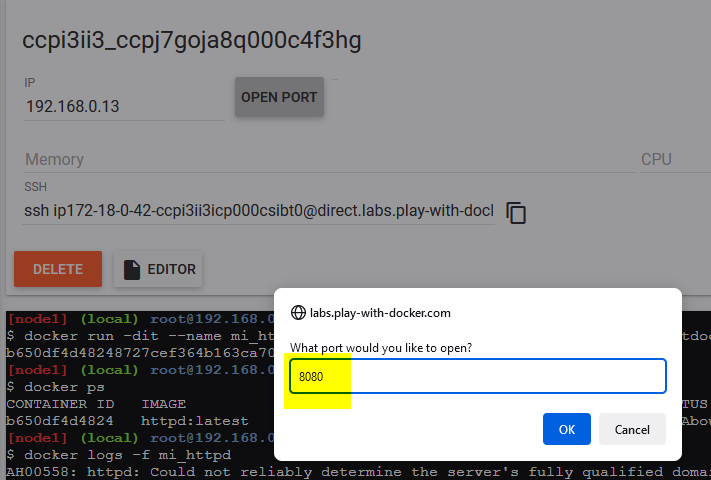
1. Revisar el log del contenedor con <<***docker logs -f mi\_httpd*** >>



1. Presionar **Ctrl+C** para abandonar la vista de logs de docker
2. Presionar el botón **OPEN PORT** para ver nuestro **httpd** corriendo en Internet:



1. Ingresar el puerto 8080 y presionar **OK**:



1. El resultado será la página web siguiente:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

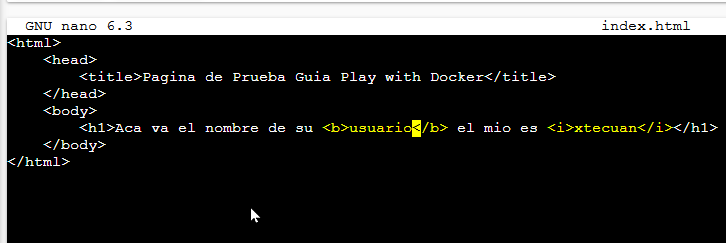
1. Modifique la pagina Web con nano, pero primero instale nado usando el comando

<< ***apk add nano zip unzip*** >> (Se instalarán como extra zip y unzip):

Texto

Descripción generada automáticamente

1. Hacer << ***nano index.html*** >> cambiar donde dice usuario y xtecuan, escribir lo que quiera:



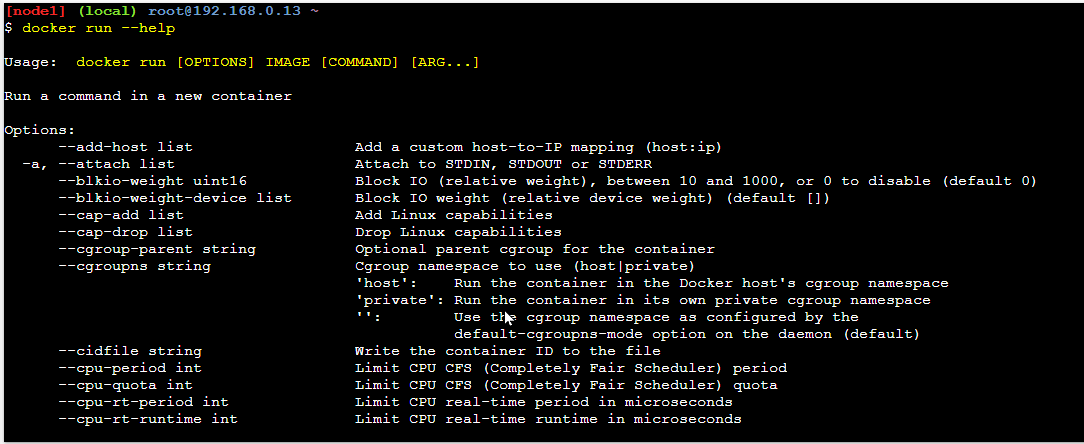
**Nota 1:** ***Guardar con Ctrl+x Yes y Enter***

1. Recargar la pagina para ver los cambios:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

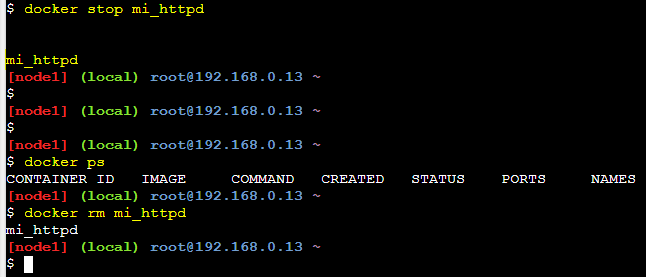
1. ¡Felicidades acaba de crear su primer contenedor Docker!
2. Explore en la ayuda del comando **docker run para ver** que flags/opciones han sido empleadas para levantar nuestro primer contenedor con el siguiente comando << ***docker run --help*** >>



|  |
| --- |
| Usage: docker run [OPTIONS] IMAGE [COMMAND] [ARG...]  Run a command in a new container  Options:  --add-host list Add a custom host-to-IP mapping (host:ip)  -a, --attach list Attach to STDIN, STDOUT or STDERR  --blkio-weight uint16 Block IO (relative weight), between 10 and 1000, or 0 to disable (default 0)  --blkio-weight-device list Block IO weight (relative device weight) (default [])  --cap-add list Add Linux capabilities  --cap-drop list Drop Linux capabilities  --cgroup-parent string Optional parent cgroup for the container  --cgroupns string Cgroup namespace to use (host|private)  'host': Run the container in the Docker host's cgroup namespace  'private': Run the container in its own private cgroup namespace  '': Use the cgroup namespace as configured by the  default-cgroupns-mode option on the daemon (default)  --cidfile string Write the container ID to the file  --cpu-period int Limit CPU CFS (Completely Fair Scheduler) period  --cpu-quota int Limit CPU CFS (Completely Fair Scheduler) quota  --cpu-rt-period int Limit CPU real-time period in microseconds  --cpu-rt-runtime int Limit CPU real-time runtime in microseconds  -c, --cpu-shares int CPU shares (relative weight)  --cpus decimal Number of CPUs  --cpuset-cpus string CPUs in which to allow execution (0-3, 0,1)  --cpuset-mems string MEMs in which to allow execution (0-3, 0,1)  -d, --detach Run container in background and print container ID  --detach-keys string Override the key sequence for detaching a container  --device list Add a host device to the container  --device-cgroup-rule list Add a rule to the cgroup allowed devices list  --device-read-bps list Limit read rate (bytes per second) from a device (default [])  --device-read-iops list Limit read rate (IO per second) from a device (default [])  --device-write-bps list Limit write rate (bytes per second) to a device (default [])  --device-write-iops list Limit write rate (IO per second) to a device (default [])  --disable-content-trust Skip image verification (default true)  --dns list Set custom DNS servers  --dns-option list Set DNS options  --dns-search list Set custom DNS search domains  --domainname string Container NIS domain name  --entrypoint string Overwrite the default ENTRYPOINT of the image  -e, --env list Set environment variables  --env-file list Read in a file of environment variables  --expose list Expose a port or a range of ports  --gpus gpu-request GPU devices to add to the container ('all' to pass all GPUs)  --group-add list Add additional groups to join  --health-cmd string Command to run to check health  --health-interval duration Time between running the check (ms|s|m|h) (default 0s)  --health-retries int Consecutive failures needed to report unhealthy  --health-start-period duration Start period for the container to initialize before starting health-retries countdown (ms|s|m|h) (default 0s)  --health-timeout duration Maximum time to allow one check to run (ms|s|m|h) (default 0s)  --help Print usage  -h, --hostname string Container host name  --init Run an init inside the container that forwards signals and reaps processes  -i, --interactive Keep STDIN open even if not attached  --ip string IPv4 address (e.g., 172.30.100.104)  --ip6 string IPv6 address (e.g., 2001:db8::33)  --ipc string IPC mode to use  --isolation string Container isolation technology  --kernel-memory bytes Kernel memory limit  -l, --label list Set meta data on a container  --label-file list Read in a line delimited file of labels  --link list Add link to another container  --link-local-ip list Container IPv4/IPv6 link-local addresses  --log-driver string Logging driver for the container  --log-opt list Log driver options  --mac-address string Container MAC address (e.g., 92:d0:c6:0a:29:33)  -m, --memory bytes Memory limit  --memory-reservation bytes Memory soft limit  --memory-swap bytes Swap limit equal to memory plus swap: '-1' to enable unlimited swap  --memory-swappiness int Tune container memory swappiness (0 to 100) (default -1)  --mount mount Attach a filesystem mount to the container  --name string Assign a name to the container  --network network Connect a container to a network  --network-alias list Add network-scoped alias for the container  --no-healthcheck Disable any container-specified HEALTHCHECK  --oom-kill-disable Disable OOM Killer  --oom-score-adj int Tune host's OOM preferences (-1000 to 1000)  --pid string PID namespace to use  --pids-limit int Tune container pids limit (set -1 for unlimited)  --platform string Set platform if server is multi-platform capable  --privileged Give extended privileges to this container  -p, --publish list Publish a container's port(s) to the host  -P, --publish-all Publish all exposed ports to random ports  --pull string Pull image before running ("always"|"missing"|"never") (default "missing")  --read-only Mount the container's root filesystem as read only  --restart string Restart policy to apply when a container exits (default "no")  --rm Automatically remove the container when it exits  --runtime string Runtime to use for this container  --security-opt list Security Options  --shm-size bytes Size of /dev/shm  --sig-proxy Proxy received signals to the process (default true)  --stop-signal string Signal to stop a container (default "SIGTERM")  --stop-timeout int Timeout (in seconds) to stop a container  --storage-opt list Storage driver options for the container  --sysctl map Sysctl options (default map[])  --tmpfs list Mount a tmpfs directory  -t, --tty Allocate a pseudo-TTY  --ulimit ulimit Ulimit options (default [])  -u, --user string Username or UID (format: <name|uid>[:<group|gid>])  --userns string User namespace to use  --uts string UTS namespace to use  -v, --volume list Bind mount a volume  --volume-driver string Optional volume driver for the container  --volumes-from list Mount volumes from the specified container(s)  -w, --workdir string Working directory inside the container |

**Nota 2:** para detener el contenedor ejecutar << ***docker stop mi\_httpd*** >>

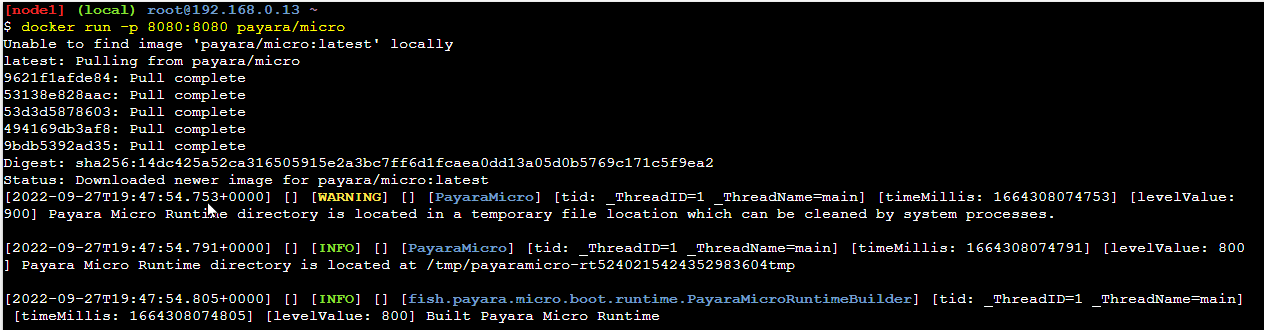
**Nota 3:** para eliminar el contenedor ejecutar << ***docker rm mi\_httpd*** >>



## Parte 2. Ejecutando Microservicio con Payara Micro

1. Para ejecutar un contenedor con Payara Micro sin ninguna aplicación publicada hacemos

<< ***docker run -p 8080:8080 payara/micro*** >>



1. Si hacemos OPEN PORT en el puerto 8080:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. El resultado sería el siguiente:

Interfaz de usuario gráfica, Texto, Aplicación

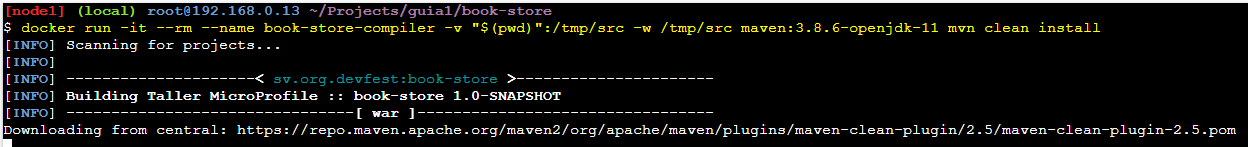
Descripción generada automáticamente

1. Presionamos **Ctrl + C** para interrumpir la ejecución del contenedor
2. Compilar código fuente de ejemplo de microservicio en Eclipse Microprofile usando Maven en Docker:

<< ***cd*** ***$HOME/Projects/guia1/book-store*** >>

<< ***docker run -it --rm --name my-maven-project -v "$(pwd)":/tmp/src -w /tmp/src maven:3.8.6-openjdk-11 mvn clean install***

>>



1. El resulta será el siguiente:

Interfaz de usuario gráfica

Descripción generada automáticamente con confianza media

1. Crear una imagen personalizada de Payara Micro para publicar el ejemplo ***book-store:***

<< ***cd*** ***$HOME/Projects/guia1/book-store*** >>

Texto

Descripción generada automáticamente

1. Hacer build de la imagen personalizada (Se requiere haber concluido con éxito los numerales del 5 y 6):

<< ***docker build -t mipayara .*** >>

Texto

Descripción generada automáticamente

1. Ejecutar la imagen personalizada de Payara Micro:

<< ***docker run -dit --name mipayara\_container -p 8080:8080 mipayara:latest*** >>

Texto

Descripción generada automáticamente

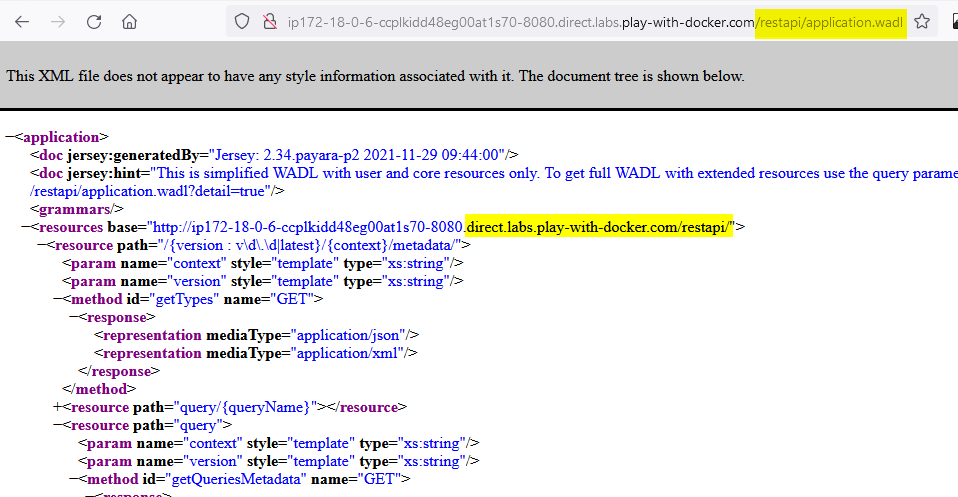
Texto

Descripción generada automáticamente

1. Automáticamente se abrirá el Puerto **8080** en el nodo de **Play with Docker** pudiendo ingresar a la pagina web del descriptor del servicio **RESTful** en el path /**restapi/application.wadl**

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente



1. Para probar la aplicación backend puede usar desde su máquina la colección de **POSTman** que esta alojada en el respositorio git de la guía en la URL:

<https://github.com/xtecuan/guia-play-with-docker/blob/main/postman/book-store.postman_collection.json>

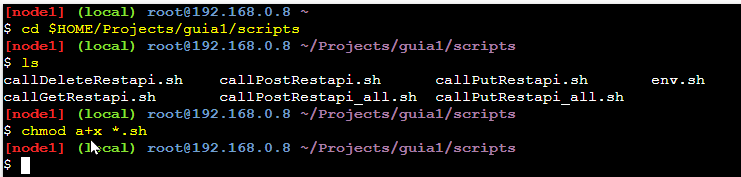
1. Emplee la URL del backend generada por el nodo que tiene el formato siguiente:

http://nombreautogenerado-puerto.**direct.labs.play-with-docker.com/restapi/application.wadl**

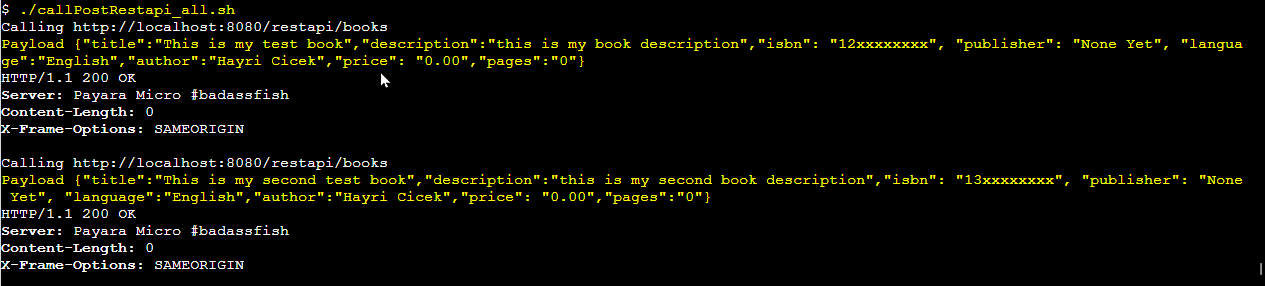
1. Otra forma de Probar la API de **book-store** es usando **curl** o empleando los scripts que están en el repositorio:

<< **cd $HOME/Projects/guia1/scripts** >>

<< ***chmod a+x \*.sh*** >>



1. Crear Libros:



1. Obtener Libros:

Un conjunto de letras negras en un fondo negro

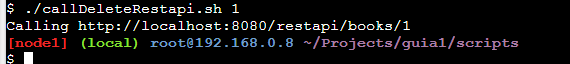
Descripción generada automáticamente con confianza media

1. Modificar Libros:

Imagen que contiene Texto

Descripción generada automáticamente

1. Borrar Libros:



1. ¡Felicidades ha completado la parte 2 de la guía empleando contenedores para la compilación y ejecución de cargas de trabajo!
2. Explore en la ayuda del comando **docker build** para ver que flags/opciones han sido empleadas para construir nuestra primera imagen con el siguiente comando << ***docker build --help*** >>

|  |
| --- |
| Usage: docker build [OPTIONS] PATH | URL | -  Build an image from a Dockerfile  Options:  --add-host list Add a custom host-to-IP mapping (host:ip)  --build-arg list Set build-time variables  --cache-from strings Images to consider as cache sources  --cgroup-parent string Optional parent cgroup for the container  --compress Compress the build context using gzip  --cpu-period int Limit the CPU CFS (Completely Fair Scheduler) period  --cpu-quota int Limit the CPU CFS (Completely Fair Scheduler) quota  -c, --cpu-shares int CPU shares (relative weight)  --cpuset-cpus string CPUs in which to allow execution (0-3, 0,1)  --cpuset-mems string MEMs in which to allow execution (0-3, 0,1)  --disable-content-trust Skip image verification (default true)  -f, --file string Name of the Dockerfile (Default is 'PATH/Dockerfile')  --force-rm Always remove intermediate containers  --iidfile string Write the image ID to the file  --isolation string Container isolation technology  --label list Set metadata for an image  -m, --memory bytes Memory limit  --memory-swap bytes Swap limit equal to memory plus swap: '-1' to enable unlimited swap  --network string Set the networking mode for the RUN instructions during build (default "default")  --no-cache Do not use cache when building the image  --pull Always attempt to pull a newer version of the image  -q, --quiet Suppress the build output and print image ID on success  --rm Remove intermediate containers after a successful build (default true)  --security-opt strings Security options  --shm-size bytes Size of /dev/shm  -t, --tag list Name and optionally a tag in the 'name:tag' format  --target string Set the target build stage to build.  --ulimit ulimit Ulimit options (default []) |

**Nota 4:** para ver el listado de imágenes presentes en el runtime de docker hacer << ***docker images*** >>

Texto

Descripción generada automáticamente

## Parte 3. Ejecutando Microservicio con Angular

1. Para esta parte emplearemos una conexión SSH hacia el nodo de Play with Docker, podemos utilizar una terminal ***cmd*** o ***powershell*** de Windows, en la pantalla de la sesión se obtiene el enlace de conexión:

Interfaz de usuario gráfica, Texto, Aplicación

Descripción generada automáticamente

1. Ejecutamos el comando:

<< ***ssh ip172-18-0-35-ccq5tg23icp000csjq9g@direct.labs.play-with-docker.com*** >> desde una interfaz ***cmd o powershell***

***Texto

Descripción generada automáticamente***

1. Ingresamos a la carpeta de la aplicación Angular con el siguiente comando

<< ***cd $HOME/Projects/guia1/my-store*** >>

Texto

Descripción generada automáticamente

1. Se construye la imagen del proyecto Angular ingresando el comando:

***<< docker build -t my-store-img .*** >>

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

1. La construcción de la imagen Docker tomará un tiempo considerable debido a que se usa una construcción de varias etapas como se muestra en el archivo Dockerfile:

Texto

Descripción generada automáticamente

**Nota 5:** El primer stage del proceso de build se hace en un contenedor con node version 16.10.0 luego los objetos generados por este proceso se copian al contenedor con nginx 1.15 para ser servidos en el path

***/my-store***

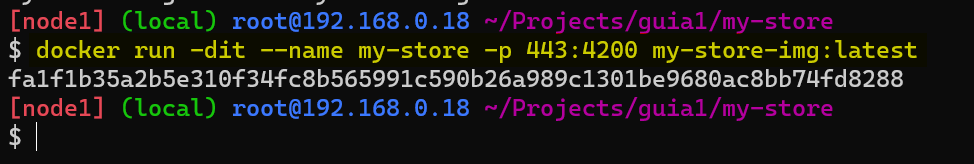
1. Si el proceso de construcción termina exitosamente se tiene la siguiente vista:

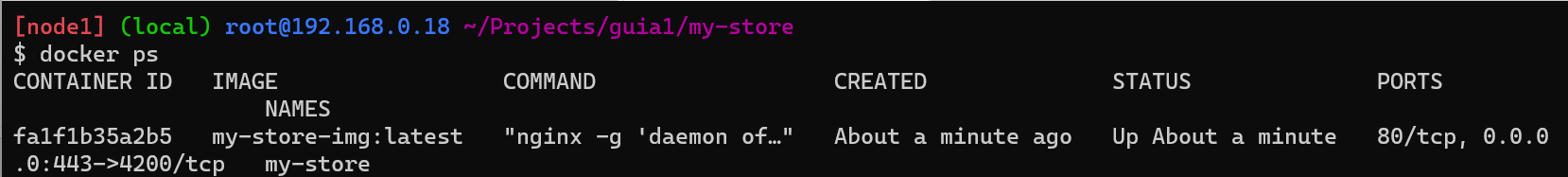
Texto

Descripción generada automáticamente

1. Se procede a ejecutar el contenedor con el siguiente comando:

***<< docker run -dit --name my-store -p 443:4200 my-store-img:latest >>***

******

******

1. Se tiene que abrir el puerto 443 desde la pagina web de Play with Docker:

Interfaz de usuario gráfica, Aplicación

Descripción generada automáticamente

1. Automáticamente la página abrirá una url base pero con el protocolo http, hay que cambiar a https y agregar el path ***/my-store/,*** a manera de ejemplo la url usa el siguiente formato:

***https://ip172-18-0-35-ccq5tg23icp000csjq9g-443.direct.labs.play-with-docker.com/my-store/***

Se tiene que aceptar el certificado auto firmado para lograr ver la página.

1. La vista de la aplicación Angular es la siguiente:

Interfaz de usuario gráfica, Aplicación, Word

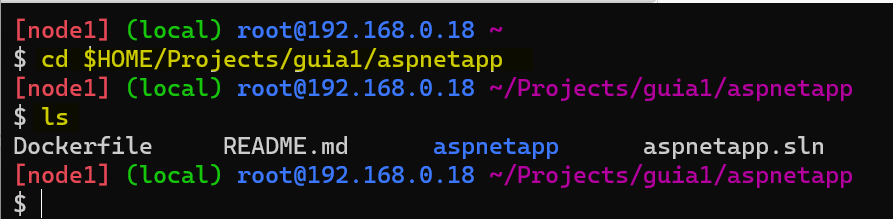
Descripción generada automáticamente

1. ¡Felicidades ha logrado construir y ejecutar un contenedor con Angular!

## Parte 4. Ejecutando Microservicio con .Net 7

1. Ingresamos al path ***aspnetapp*** dentro del repositorio con el siguiente comando:

<< ***cd $HOME/Projects/guia1/aspnetapp*** >>



1. Construimos la aplicación con el siguiente comando:

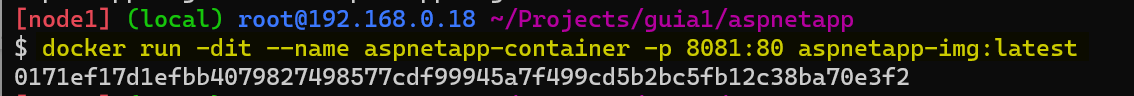
<< ***docker build -t aspnetapp-img .*** >>

Texto

Descripción generada automáticamente

1. Ejecutamos la aplicación con el siguiente comando:

<< ***docker run -dit --name aspnetapp-container -p 8081:80 aspnetapp-img:latest*** >>



1. Revisamos el log de la aplicación con el comando:

<< ***docker logs -f aspnetapp-container*** >>

Texto

Descripción generada automáticamente

1. Desde la página Web de Play with Docker abrimos el puerto 8081:

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

Descripción generada automáticamente

1. El Resultado sería el siguiente:

Interfaz de usuario gráfica, Tabla

Descripción generada automáticamente

1. ¡Felicidades ha logrado publicar la aplicación en .Net!

## Parte 5. Ejercicio de Desafío 1

1. Instalar un contenedor con mariadb o postgresql empleando Volumenes.
2. Crear una base de datos de ejemplo en cualquiera de las dos bases.
3. Ingresar datos de ejemplo en las tablas de la base de datos.
4. Crear una API RESTful que consuma los datos de ejemplo de la base de datos en cualquier tecnología de las que se han empleado en esta guía de laboratorio.