



Introducción a Docker

Noviembre 2019

Desde cualquier país marque la extensión 3840 (Call Center)
o contáctenos vía email: mercadeo@gbm.net

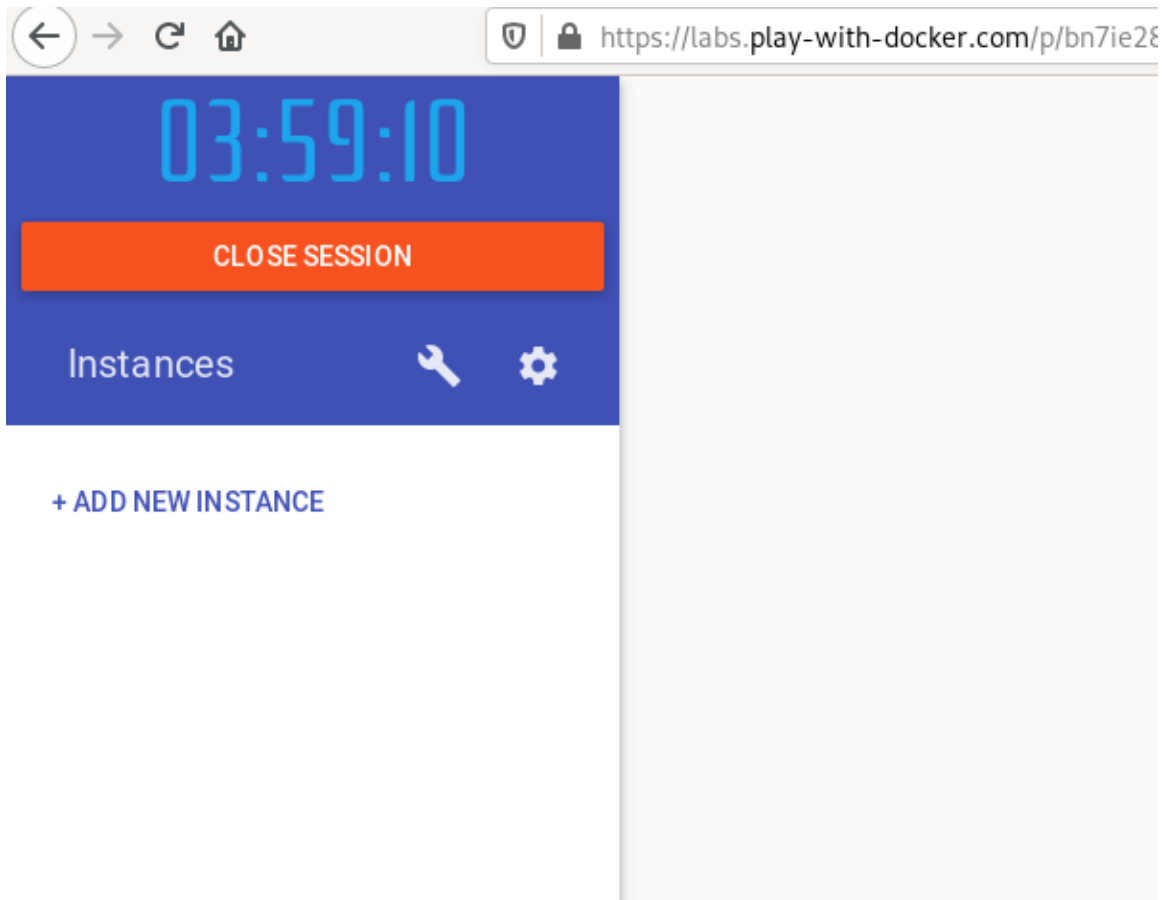
• Guatemala (502) 2424-2222 • El Salvador (503) 2505-9600 • Honduras (504) 2232-2319/09 •
• Nicaragua (505) 2255-6630 • Costa Rica (506) 2284-3999 • Panamá (507) 300-4800 •
República Dominicana • (809) 566-5161
www.gbm.net



1. Trabajando con docker

Los siguientes ejercicios se van a ejecutar en un ambiente de docker de la página "<https://labs.play-with-docker.com/>", solamente es necesario crear una cuenta de usuario en dockerhub, página "<https://hub.docker.com/>"

1. Ingresar a la página <https://labs.play-with-docker.com/>, darle en "Start", loguearse con el usuario y password creados en dockerhub, debería de verse la pantalla algo así.



En la columna de la izquierda le damos click en "+ADD NEW INSTANCE" y nos debería salir una imagen como la siguiente, con esto tenemos el ambiente listo para utilizarse.

Verificamos la versión de docker utilizada para el lab, esto no es especialmente importante, solamente nos indica

```
$ docker -v
Docker version 19.03.4, build 9013bf583a
```

2. Como todas las cosas nuevas en computación, es acostumbrado iniciar con "helloworld".

```
$ docker run hello-world
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:

1. The Docker client contacted the Docker daemon.
2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
(amd64)
3. The Docker daemon created a new container from that image which runs the executable that produces the output you are currently reading.
4. The Docker daemon streamed that output to the Docker client, which sent it to your terminal.

To try something more ambitious, you can run an Ubuntu container with:

```
$ docker run -it ubuntu bash
```

Share images, automate workflows, and more with a free Docker ID:

<https://hub.docker.com/>

For more examples and ideas, visit:

<https://docs.docker.com/get-started/>

Revise el mensaje. Primero van a ver que la imagen se descargó automáticamente. Segundo la versión :latest fue agregada al nombre de la imagen. Cuando se usa :latest no se especifica la versión.

3. Volver a ejecutar "helloworld" Verifique que la imagen no se descargó de Nuevo porque ya estaba localmente, después de esto corre de forma normal.

```
$ docker run hello-world
```

Hello from Docker!

This message shows that your installation appears to be working correctly.

4. Ya existe localmente, para verificar esto ejecutamos `docker images | grep hello-world` y vemos la imagen.

```
$ docker images | grep hello-world
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
hello-world	latest	fce289e99eb9	4 months ago	1.84kB

5. ¿Desde dónde la imagen hello-world se hizo pull? Vamos a la URL https://hub.docker.com/_/hello-world/ ahí pueden leer sobre esta imagen. Dockerhub es un repositorio que contiene las imagines de docker que se pueden usar. Dockerhub no es el único repositorio.
6. Cuando una imagen está corriendo se le llama contenedor. Vamos a correr una imagen típica; la imagen contiene la base de datos NOSQL "couchdb".

```
$ docker run -d couchdb
Unable to find image 'couchdb:latest' locally
latest: Pulling from library/couchdb
743f2d6c1f65: Already exists
951b83aff526: Pull complete
4691fbd29f92: Pull complete
f92ef7427d1d: Pull complete
fde962543341: Pull complete
c7bb84a3961b: Pull complete
```

En la captura anterior se muestra mientras se descarga la imagen desde dockerhub. Cuando la imagen se descarga, no se ve nada del contenedor como con helloworld. Lo que se ve es el ID del contenedor. Ejemplo:
81848690dd57fbf65ed80ebd5203e962eaccdd347de7691074648a42e8e405e1.

7. Ahora vamos a ver el contenedor corriendo o iniciado. Solamente la primera parte del ID del contenedor se muestra. Con lo que se muestra del ID del contenedor en la salida del comando `docker ps | grep couchdb` es suficiente para iniciarlo, detenerlo etc. Se puede especificar lo que andamos buscando si tenemos muchos contenedores corriendo con el siguiente comando `docker ps | grep couchdb`

```
$ docker ps | grep couchdb
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
81848690dd57	couchdb	"tini -- /docker-ent..."	14 minutes ago	Up 14 minutes	4369/tcp, 5984/tcp, 9100/tcp	vigorous_gauss

8. Una imagen puede correr múltiples veces. De esta forma se ejecuta otro contenedor para la imagen de couchdb.

```
$ docker run -d couchdb
bb89e93ad6fb9a46f285a6e0b40408fd61ad21987aefa886f04ec538cf0980f6
```

9. Ahora vemos 2 contenedores corriendo imagines de la base de datos couchdb. Como ya la imagen de couchdb estaba descargada de forma local no se necesitó descargar nuevamente la imagen desde el dockerhub. Ejecutamos nuevamente **docker ps | grep couchdb** esto para ver los contenedores que están corriendo o podemos ejecutar tambien **docker ps | grep couchdb**

```
$ docker ps | grep couchdb
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
bb89e93ad6fb	couchdb	"tini -- /docker-ent..."	2 minutes ago	Up 2 minutes	4369/tcp, 5984/tcp, 9100/tcp	cranky_blackwell
81848690dd57	couchdb	"tini -- /docker-ent..."	21 minutes ago	Up 21 minutes	4369/tcp, 5984/tcp, 9100/tcp	vigorous_gauss

10. Los contenedores lucen similares, pero tienen nombres únicos y IDs únicos. Detenga el contenedor más reciente y vuelva a ejecutar **docker ps | grep couchdb** para ver que está corriendo.

```
$ docker stop bb89e93ad6fb
```

```
bb89e93ad6fb
```

```
$ docker ps | grep couchdb
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
81848690dd57	couchdb	"tini -- /docker-ent..."	30 minutes ago	Up 30 minutes	4369/tcp, 5984/tcp, 9100/tcp	vigorous_gauss

11. Detenga el otro contenedor y revise que está corriendo.

```
$ docker stop 81848690dd57
```

```
81848690dd57
```

```
$ docker ps | grep couchdb
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS	NAMES
--------------	-------	---------	---------	--------	-------	-------

12. Revisemos que la imagen de couchdb todavía existe. Esto lo hacemos ejecutando **docker images | grep couchdb**

```
$ docker images | grep couchdb
couchdb      latest      3c889959dd0c   3 weeks ago   201MB
```

13. Vamos a eliminar la imagen de couchdb.

```
$ docker rmi couchdb:latest
Error response from daemon: conflict: unable to remove repository reference
"couchdb:latest" (must force) - container 81848690dd57 is using its referenced image
3c889959dd0c
```

No se puede eliminar una imagen de docker sin antes eliminar el contenedor.
Ejecute **docker ps -a | grep couchdb** esto mostrara todos los contenedores existentes y no solo los contenedores iniciados.

```
$ docker ps -a | grep couchdb
bb89e93ad6fb   couchdb      "tini -- /docker-ent..." 25 minutes ago   Exited (0) 13 minutes ago
cranky_blackwell
81848690dd57   couchdb      "tini -- /docker-ent..." 44 minutes ago   Exited (0) 10 minutes ago
vigorous_gauss
```

14. Elimine el contenedor de couchdb, elimine la imagen de couchdb y verifique si se eliminó correctamente.

Eliminar los contenedores:

```
$ docker rm bb89e93ad6fb
```

```
bb89e93ad6fb
```

```
$ docker rm 81848690dd57
```

```
81848690dd57
```

Eliminar imagen:

```
$ docker rmi couchdb
```

```
Untagged: couchdb:latest
```

```
Untagged:
```

```
couchdb@sha256:dbcb1b32f356aca9415ade8eccaca7c925c3ed4c2c52f113f9b3753994eee226
```

```
Deleted: sha256:3c889959dd0c9ca37f7fcd18ebbe430f935d3fd6c6d53179b8c515d89c46c7c
```

```
Deleted: sha256:b4cd668dbd720b7685d2926508099c6ec69b7f6b03e651ad480fe432d28b5a8e
```

```
Deleted: sha256:c682feb5d4f2dd6f19afe0c6e2da4029bdf902bbd1aab8ff2e8bfc3b375c771c
```

```
Deleted: sha256:5e4ec9d294ce29e9bc30bba6321dd5e82c767d1de2e7043a28d5d716c3f52aac
```

```
Deleted: sha256:5c88ea0dad38236fb68618694619738e9f614cedcbc30824871f124352f7f60f
```

```
Deleted: sha256:bac99d99225f78ddaf25d53a7214d37366790c917ae5f1510dc711e247165495
```

```
Deleted: sha256:b834f379103b4793a7b3f79cc387726258e694902a2b400fff1ed3ad9d737ab
```

```
Deleted: sha256:c657bc038affab5a16e3bf6a9d6627d22e696b71b612f5720b28cbdf069d9777
```

```
Deleted: sha256:771c5347f8c78fa7b5dda32c2df6f43b322c1eb1fe8451cd43ed24643e8106b7
```

```
Deleted: sha256:a89bc6fbab700b7f4b79c47d2ba97340b0776c7a1bfa397e0c073b83561f7270
```

```
Deleted: sha256:dfd172c570a58f71a51b7630b793779e972d92a430152da48e092aad45dd5818
```

```
Deleted: sha256:14a9d65c1bfa387f7aadb1cc1173740b7d958e79a4bf0a2811331d77c684044
```

Verificar si hay contenedores de couchdb

```
$ docker ps -a | grep couchdb
```

CONTAINER ID	IMAGE	COMMAND	CREATED	STATUS	PORTS
NAMES					

Verificar si hay images de couchdb

```
$ docker images | grep couchdb
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
------------	-----	----------	---------	------

Crear una aplicación NGINX dentro de un docker y modificarle el contenido.

- 1- Vamos a crear una aplicación NGINX dentro de un docker y modificarle el contenido del html.

```
$ mkdir test-nginx
$ cd test-nginx

$ vi index.html

#### HTML File
<html>
<head>
<style>
h1 {color:black;text-align:left;font-family:helvetica;font-size:36}
p {color:tomato;text-align:left;font-family:helvetica;font-size:18}
</style>
</head>
<body>
<h1> Bienvenidos a nginx service! </h1>
<p> Este es un servicio custom de nginx para el workshop de
GBM. </p>
</body>
</html>
####
```

- 2- Una breve explicación de cada linea del Dockerfile:

FROM: Inicializa el build y setea la imagen base para los siguientes comandos.

LABEL: Agrega metadata a una imagen.

VOLUME: Instrucción para crear un punto de montaje en el contenedor.

COPY: Copia nuevos archivos o directorios desde la ruta que indiquemos al destino que se le indique dentro de contenedor.

EXPOSE: Le indica a docker que el contenedor escucha en el puerto especificado.

Para más detalles sobre dockerfile y usos:

<https://docs.docker.com/engine/reference/builder/#expose>

Creamos el docker file.

```
$ vim dockerfile

#### Dockerfile
FROM nginx:alpine
LABEL maintainer cliente
VOLUME ["/vol-nginx"]
COPY index.html /usr/share/nginx/html
EXPOSE 80
####
```


3- Realizamos el docker build

```
$ docker build --no-cache -f dockerfile -t my-nginx .
Sending build context to Docker daemon 3.072kB
Step 1/5 : FROM nginx:alpine
alpine: Pulling from library/nginx
e7c96db7181b: Pull complete
264026bbe255: Pull complete
a71634c55d29: Pull complete
5595887beb81: Pull complete
Digest:
sha256:57a226fb6ab6823027c0704a9346a890ffb0cacde06bc19bbc234c8720673555
Status: Downloaded newer image for nginx:alpine
--> dd025cdfe837
Step 2/5 : LABEL maintainer cliente
--> Running in d74ceeb59363
Removing intermediate container d74ceeb59363
--> b9ed8d0ae35f
Step 3/5 : VOLUME ["/vol-nginx"]
--> Running in 931919633f18
Removing intermediate container 931919633f18
--> 3a91965e41cf
Step 4/5 : COPY index.html /usr/share/nginx/html
--> 83e4844ca82b
Step 5/5 : EXPOSE 80
--> Running in 5f75437adf13
Removing intermediate container 5f75437adf13
--> 651e1d3217ca
Successfully built 651e1d3217ca
Successfully tagged my-nginx:latest
```

4- Verificamos si se creó la imagen.

```
$ docker images | grep nginx
```

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
my-nginx	latest	651e1d3217ca	3 minutes ago	16.1MB

5- Iniciamos el contenedor.

```
$ docker run -it -d --name nginx -p 8080:80 my-nginx  
fe3f9772ad0637e5ed82baae46c8695c40a8ee66ce0bffb2df73f9a5d8e674c
```

6- Ejecutamos un `docker ps -a | grep nginx` para ver el status del contenedor.

```
$ docker ps -a
```

CONTAINER ID	IMAGE	COMMAND	CREATED
STATUS	PORTS	NAMES	
fe3f9772ad06	my-nginx	"nginx -g 'daemon of...'"	About an hour ago
Up About an hour	0.0.0.0:8080->80/tcp	nginx	

7- Ejecutamos un `docker inspect` y lo formateamos de manera que nos muestre solamente la IP para probar el servicio.

```
$ docker inspect --format='{{range .NetworkSettings.Networks}}{{println  
.IPAddress}}{{end}}' fe3f9772ad06  
172.17.0.2
```

- 8- Probamos el site con el comando “curl” seguido de la IP del contenedor, que vimos en el comando anterior, nos debería mostrar un mensaje como el siguiente, que es el texto agregado en el archivo index.html, creando en el punto 1 de este ejercicio.

```
$ curl 172.17.0.2
#### HTML File
<html>
<head>
<style>
h1 {color:black;text-align:left;font-family:helvetica;font-size:36}
p {color:tomato;text-align:left;font-family:helvetica;font-size:18}
</style>
</head>
<body>
<h1> Bienvenidos a nginx service! </h1>
<p> Este es un servicio custom de nginx para el workshop de GBM. </p>
</body>
</html>
####
```

9- Ahora vamos a actualizar lo que dice en la página de nginx.

Verificamos el ID del contenedor de my-nginx mediante el siguiente comando:

```
$ docker ps -a | grep nginx
```

```
d63d843d6428      my-nginx          "nginx -g 'daemon of..." 34 seconds ago    Up 33 seconds
0.0.0.0:8080->80/tcp      nginx
```

Ingresamos al contenedor por medio de docker exec

```
$ docker exec -i -t d63d843d6428 sh
```

Se nos va a mostrar un signo de dolar indicanos que estamos dentro del contenedor, abrimos el index.html mediante vi

```
$ vi /usr/share/nginx/html/index.html
```

Modificamos el index.html y le ponemos en el h1 Workshop de GBM

```
<html>
<head>
<style>
h1 {color:black;text-align:left;font-family:helvetica;font-size:36}
p {color:tomato;text-align:left;font-family:helvetica;font-size:18}
</style>
</head>
<body>
<h1> Workshop de GBM </h1>
<p> Este es un servicio custom de nginx para el workshop de GBM. </p>
</body>
</html>
```

Guardamos cambios con :wq

Nos salimos del contenedor con exit

10-Verificamos el URL con el comando “curl” y la página contiene los cambios.

```
$ curl 172.17.0.2
#### HTML File
<html>
<head>
<style>
h1 {color:black;text-align:left;font-family:helvetica;font-size:36}
p {color:tomato;text-align:left;font-family:helvetica;font-size:18}
</style>
</head>
<body>
<h1> Workshop de GBM </h1>
<p> Este es un servicio custom de nginx para el workshop de GBM. </p>
</body>
</html>
####
```

11- Vamos a borrar el contenedor e iniciarlo de nuevo.

Verificamos que el contenedor esta iniciado.

```
$ docker ps -a | grep nginx
```

7174225956ad	my-nginx	"nginx -g 'daemon of..."	3 minutes ago	Up 3 minutes	0.0.0.0:8080->80/tcp	nginx
--------------	----------	--------------------------	---------------	--------------	----------------------	-------

Detenemos el contenedor

```
$ docker container stop 7174225956ad
```

```
7174225956ad
```

Borramos el contenedor

```
$ docker rm 7174225956ad
```

```
7174225956ad
```

Iniciamos el contenedor de nuevo.

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

```
8b198f7534e3d61e612ee21d9c9361404d5d9876f285d09bd22022494f08d607
```

Verificamos que el contenedor esta iniciado.

```
$ docker ps -a | grep nginx
```

8b198f7534e3	my-nginx	"nginx -g 'daemon of..."	48 seconds ago	Up 47		
seconds		0.0.0.0:8080->80/tcp		nginx		

- 1- Verificamos el URL con el comando “curl” y la página ya no contiene los cambios que habíamos realizado en el index.html.

```
$ curl 172.17.0.2
#### HTML File
<html>
<head>
<style>
h1 {color:black;text-align:left;font-family:helvetica;font-size:36}
p {color:tomato;text-align:left;font-family:helvetica;font-size:18}
</style>
</head>
<body>
<h1> Bienvenidos a nginx service! </h1>
<p> Este es un servicio custom de nginx para el workshop de GBM. </p>
</body>
</html>
####
```

2- Vamos a volver a modificar el index.html.

Verificamos el ID del contenedor de my-nginx mediante el siguiente comando:

```
$ docker ps -a | grep nginx
```

```
8b198f7534e3    my-nginx    "nginx -g 'daemon of..." 6 minutes ago    Up 6 minutes    0.0.0.0:8080->80/tcp    nginx
```

Ingresamos al contenedor por medio de docker exec

```
$ docker exec -i -t 8b198f7534e3 sh
```

Se nos va a mostrar un signo de numero indicándonos que estamos dentro del contenedor, abrimos el index.html mediante vi.

```
# vi /usr/share/nginx/html/index.html
```

Modificamos el index.html y le ponemos en el h1 Workshop de GBM

```
<html>
<head>
<style>
h1 {color:black;text-align:left;font-family:helvetica;font-size:36}
p {color:tomato;text-align:left;font-family:helvetica;font-size:18}
</style>
</head>
<body>
<h1> Workshop de GBM </h1>
<p> Este es un servicio custom de nginx para el workshop de GBM. </p>
</body>
</html>
```

Guardamos cambios con :wq

Nos salimos del contenedor con exit

- 3- Verificamos el URL con el comando “curl” y la página contiene los cambios que habíamos realizado en el index.html.

```
$ curl 172.17.0.2
#### HTML File
<html>
<head>
<style>
h1 {color:black;text-align:left;font-family:helvetica;font-size:36}
p {color:tomato;text-align:left;font-family:helvetica;font-size:18}
</style>
</head>
<body>
<h1> Workshop de GBM </h1>
<p> Este es un servicio custom de nginx para el workshop de GBM. </p>
</body>
</html>
####
```

- 4- Realizamos el docker commit para que se mantenga los cambios realizados en el index.html aunque se borre el contenedor, el formato del comando docker commit es: **docker commit CONTAINER_ID IMAGE**

```
$ docker commit 8b198f7534e3 my-nginx
sha256:bfef66366d18c22d55b79023655a69dcc36bf736a58dd80b0526e8ebc86d64f8
```


- 5- Vamos a borrar el contenedor e iniciarlo de nuevo, esta vez los cambios se deberían de mantener cuando se pruebe la URL

Verificamos que el contenedor esta iniciado.

```
$ docker ps -a | grep nginx
```

```
8b198f7534e3    d0d22bb3845a    "nginx -g 'daemon of..." 17 minutes ago    Up 17 minutes    0.0.0.0:8080->80/tcp    nginx
```

Detenemos el contenedor

```
$ docker container stop 8b198f7534e3
```

```
8b198f7534e3
```

Borramos el contenedor

```
$ docker rm 8b198f7534e3
```

```
8b198f7534e3
```

Iniciamos el contenedor de nuevo.

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

```
c771047e17a91e9c5e28c1f7782ac3dda6bebf80af5159b3650ce0334beab021
```

Verificamos que el contenedor esta iniciado.

```
$ docker ps -a | grep nginx
```

```
c771047e17a9    my-nginx    "nginx -g 'daemon of..." 28 seconds ago    Up 27 seconds    0.0.0.0:8080->80/tcp    nginx
```

- 6- Como esta vez se hizo commit antes de borrar el contenedor, la imagen guarda los cambios realizados en el index.html y aunque se elimine el contenedor, al iniciar un contenedor nuevo de esta app, se van a mostrar los cambios realizados antes del commit. Revisamos la URL con el comando "curl". Y efectivamente la página cuenta con los cambios realizados.

```
$ curl 172.17.0.2
#### HTML File
<html>
<head>
<style>
h1 {color:black;text-align:left;font-family:helvetica;font-size:36}
p {color:tomato;text-align:left;font-family:helvetica;font-size:18}
</style>
</head>
<body>
<h1> Workshop de GBM </h1>
<p> Este es un servicio custom de nginx para el workshop de GBM. </p>
</body>
</html>
####
```