**Guía Laboratorio 2**

# **Ciclo de vida de contenedores, volúmenes y docker-compose**

# **Objetivo**

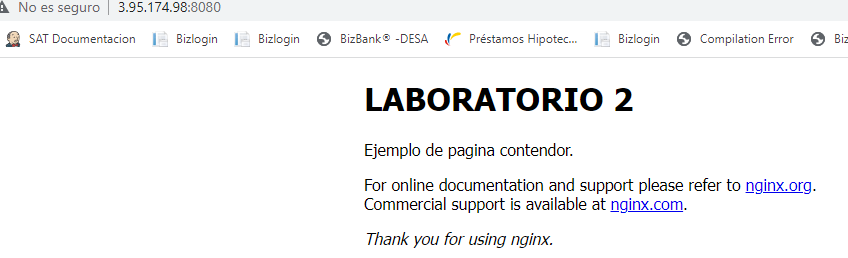
Comprender el funcionamiento de contendores, mediante prácticas con comandos básicos para el despliegue de aplicaciones completas.

# **Duración**

Ciento veinte (40) minutos.

**Actividad 1. Ciclo de vida y volúmenes**

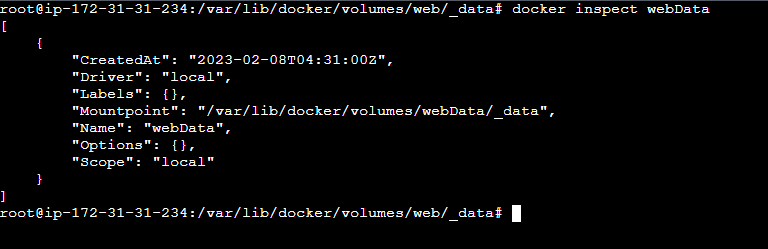
1. Cree un contenedor de la imagen NGINX (web server) usando la documentación oficial de DockerHub. NO use volúmenes. Ponga un nombre que identifique correctamente al contenedor, Ej. MyWebServer
2. Entre al contenedor y coloque una página html simple en el directorio correcto del contenedor. Adjunte una captura de pantalla del resultado desde el navegador.



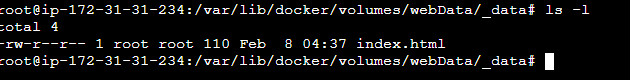
1. Reinicie la máquina virtual. Escriba los resultados a continuación:

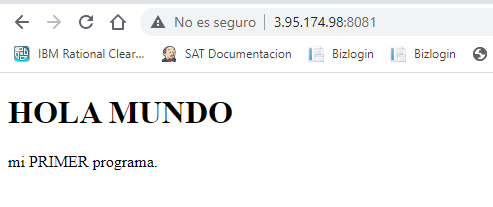
Luego de stop y start la miservidorweb1 , la pagina se muestra igual

1. Detenga el contenedor que creo en el paso 1.
2. Cree un nuevo contenedor de NGINX, con otro nombre y monte un volumen desde el directorio local /web hacia el directorio interno de nginx para servir html.
3. Cree un archivo html simple desde el directorio local (/web) y verifique que la página web los lee correctamente.
4. Realice varios cambios al archivo html, y visualice los cambios.
5. Cree un nuevo volumen llamado webData (docker volume create webData)
6. Corra el comando docker inspect webData. Adjunte captura de pantalla del resultado.



1. Localice el directorio local donde se ha montado el volumen creado en el paso 8. Cree un html para ser servido desde ahí. Cree un nuevo contenedor de nginx y monte el volumen del paso 8. Adjunte captura de pantalla del resultado.



****

**Actividad 2. Docker-compose**

1. Crear un directorio con el nombre DockerComposeTest
2. Dentro del directorio crear un archivo llamado **app.py** (nano app.py)
3. Copiar el siguiente código dentro del archivo:

|  |
| --- |
| from flask import Flask  from redis import Redis  app = Flask(\_\_name\_\_)  redis = Redis(host='redis', port=6379)  @app.route('/')  def hello():  redis.incr('hits')  return 'Hello World! I have been seen %s times.' % redis.get('hits')  if \_\_name\_\_ == "\_\_main\_\_":  app.run(host="0.0.0.0", debug=True) |

1. Dentro del directorio crear otro archivo llamado **requirements.txt** y pegar las siguientes líneas

|  |
| --- |
| flask  redis |

1. Crear un archivo llamado **Dockerfile** , sin extensión. Pegar el siguiente contenido.

|  |
| --- |
| FROM python:2.7  ADD . /code  WORKDIR /code  RUN pip install -r requirements.txt  CMD python app.py |

1. Explique línea a línea lo que significa cada uno de los comandos del paso anterior.

Se supone que son comandos del lenguaje python

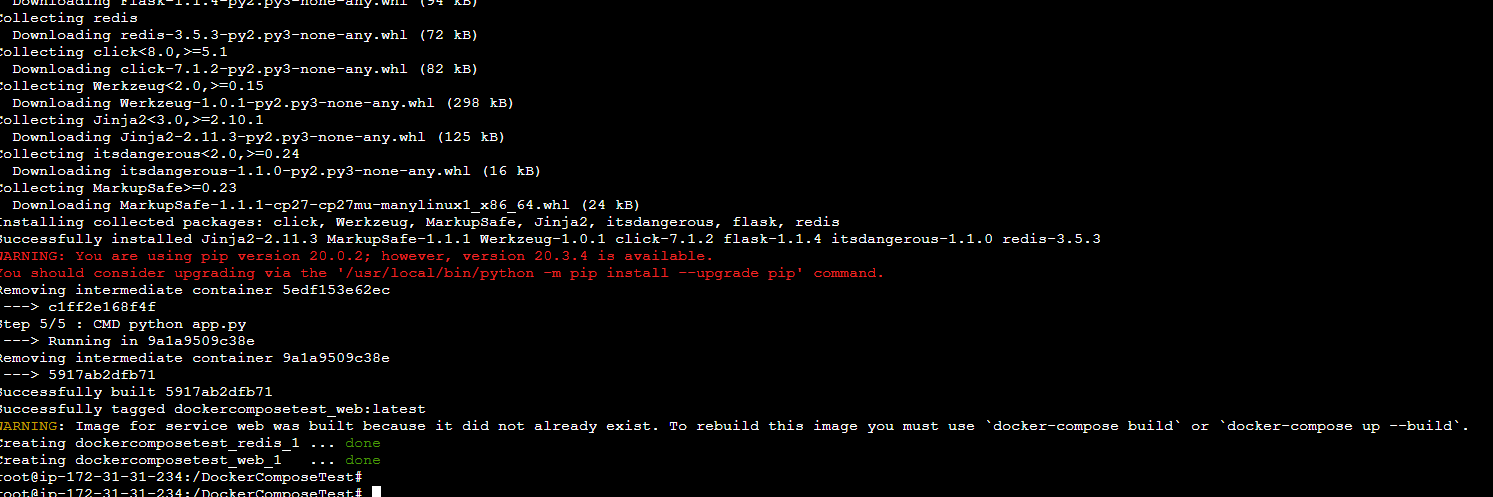
1. Crear un archivo llamado **docker-compose.yml** . Pegar el siguiente contenido.

|  |
| --- |
| web:  build: .  ports:  - "5000:5000"  volumes:  - .:/code  links:  - redis  redis:  image: redis |

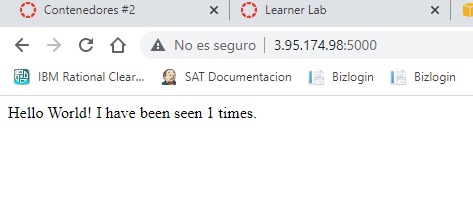
1. Explique línea a línea lo que significa cada uno de los comandos del paso anterior.

Son comandos para desplegar un sitio web según lo visto en clase

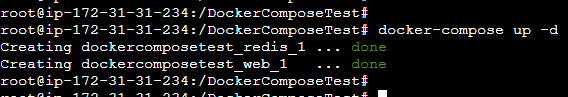
1. Levante los contenedores con el comando **docker-compose up -d**

****

1. En el navegador, abra la aplicación en la IP publica y el puerto correspondiente, adjunte captura de pantalla del resultado.

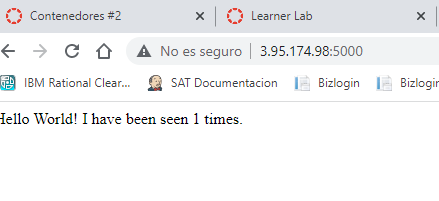
****

1. Detenga el contenedor con el comando docker-compose down y vuelva a iniciarlo. ¿Qué sucedió con el contador? Escriba el comportamiento y que cambios de hacer o incluir para que esta data no se pierda.



Se levanto mas rápidamente , al parecer la imagen ya estaba

Y el contador se reinicio a 1



Habría que crear algún volumen local para que persista la data