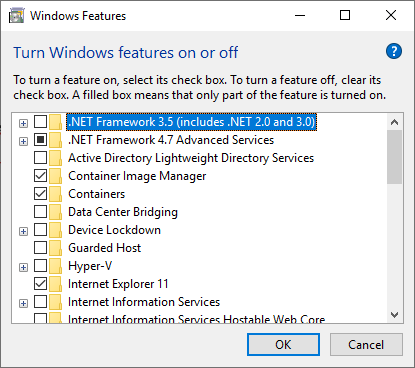
Pasos para iniciar Docker

**1.- Desactivar Hiper-V**

Se entra a la sección *Turn Windows features on or off*

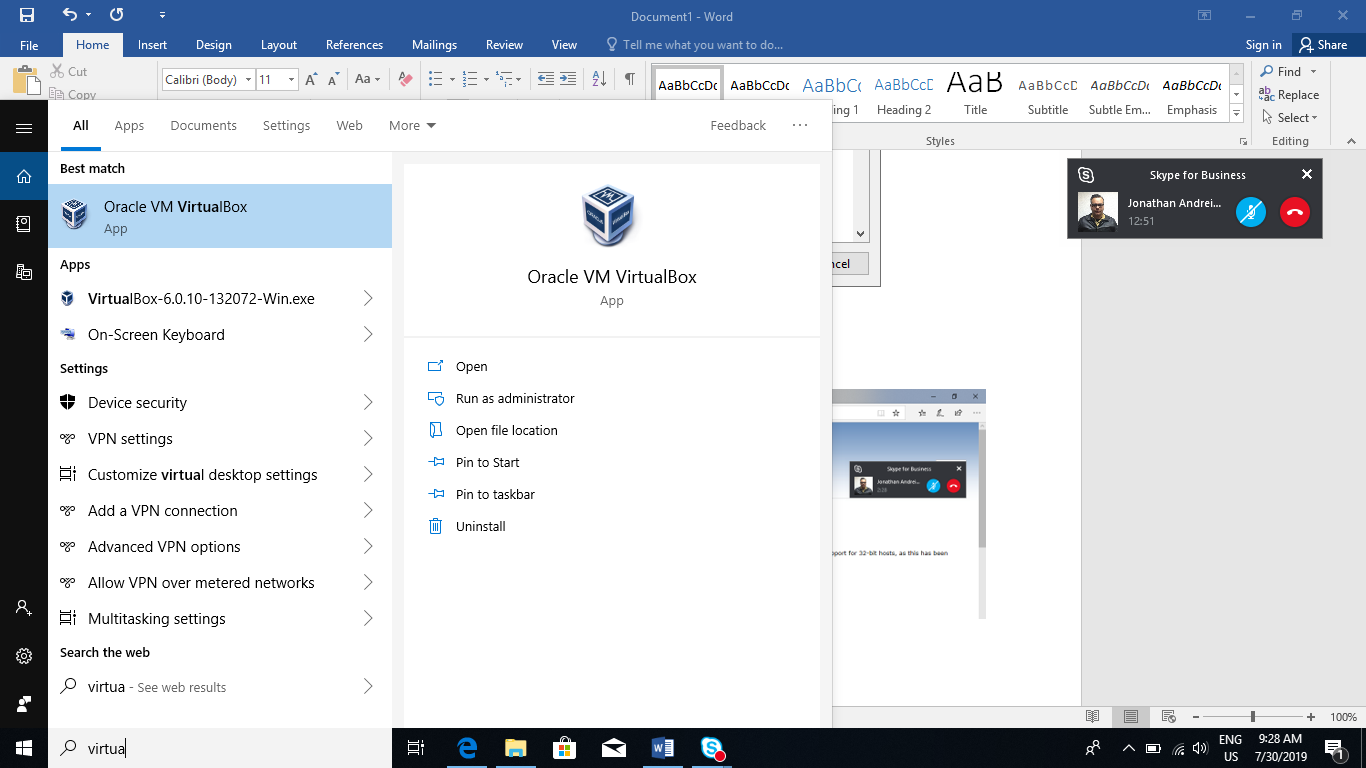


**2.- Instalar Oracle Virtual Box**

Descargar el instalador de Virtual Box para Windows desde el siguiente enlace:

*https://www.virtualbox.org/wiki/Downloads*

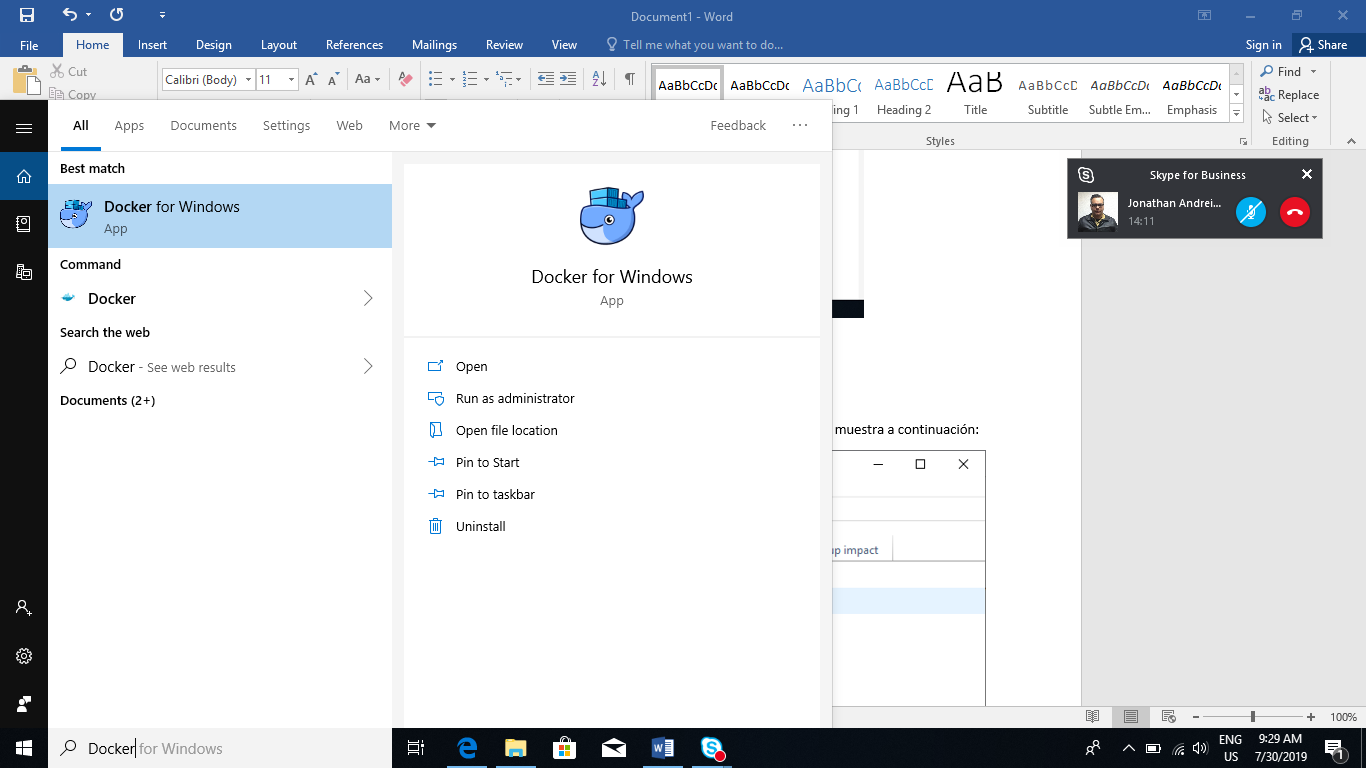




**3.- Instalar Docker**

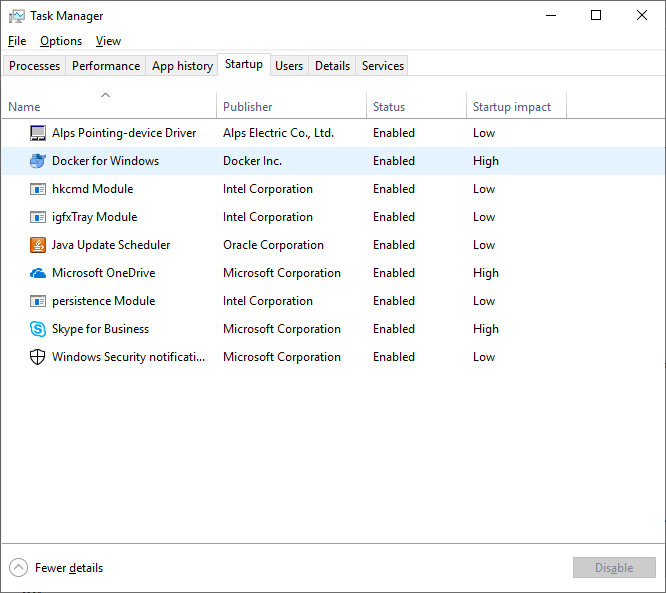
Descargar Docker para Windows desde la siguiente liga e instalarlo usando la opción default de contenedores para Linux.

*https://www.docker.com/products/docker-desktop*

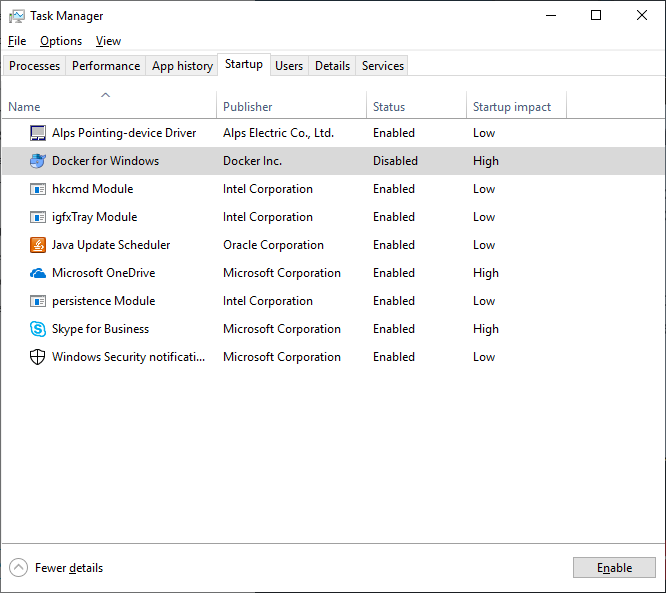


**4.- Desactivar Docker del StartUp**

Entrar al administrador de tareas y luego entrar a la sección de Starup como se muestra a continuación:

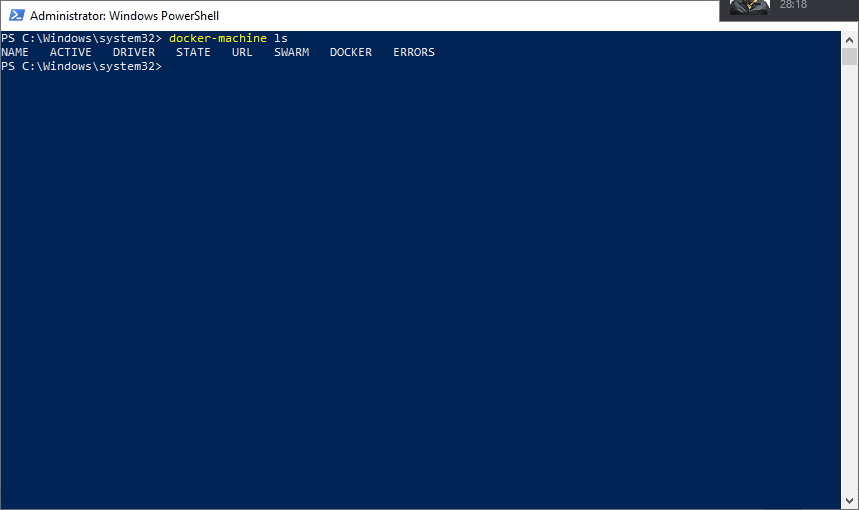


Desactivarlo seleccionando Disabled



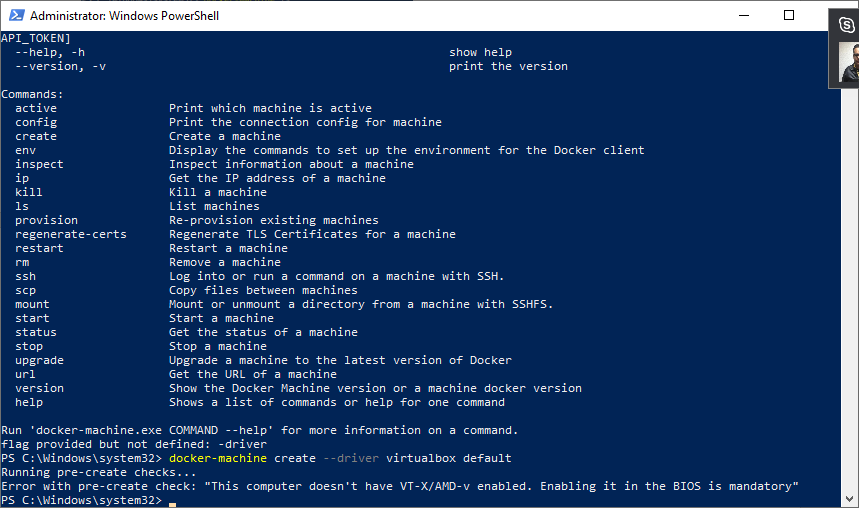
**5.- Abrir Power Shell**

Se abre el power Shell como administrador y ejecutamos el comando *docker-machine ls* para enlistar los contenedores



**6.- Crear un Docker Machine**

Al tratar de crear el docker machine inicialmente mostro un error como el siguiente:



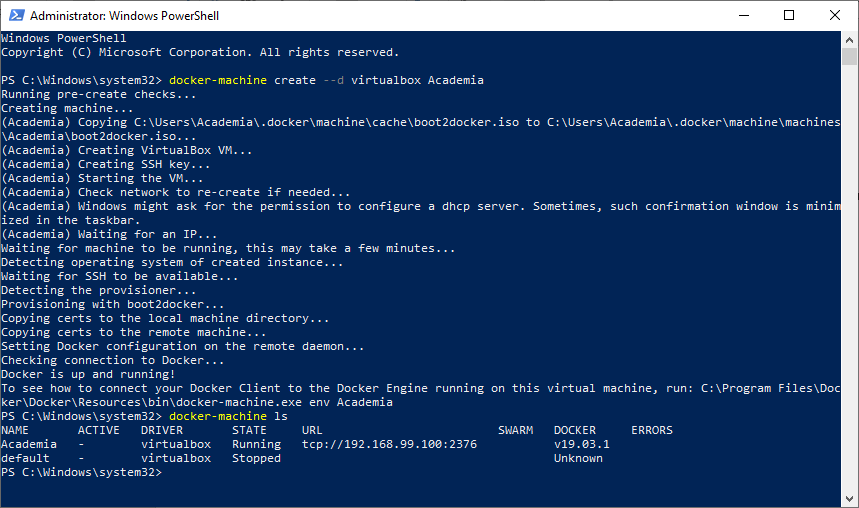
Se entró al BIOS para verificar que la virtualización este activada y se continuo con el error.

Después de eso nos dimos cuenta de un comando para desactivar otras funcionalidades de Hiper-V que estaban impidiendo el proceso y eso se logró con el siguiente comando:

*bcdedit /set hypervisorlaunchtype off*

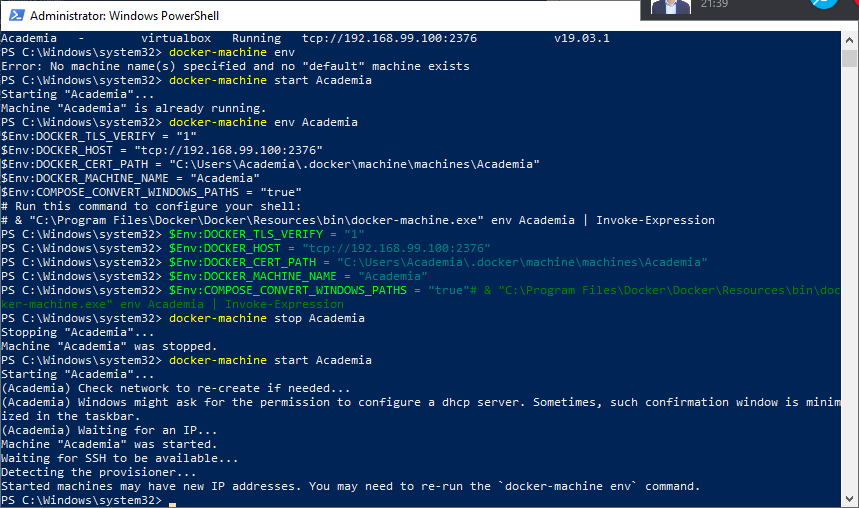
Despues de ese comando se reinicia la computadora y se ejecuta el comando para crear la máquina virtual:

*Docker-machine –d virtualbox <Nombre>*

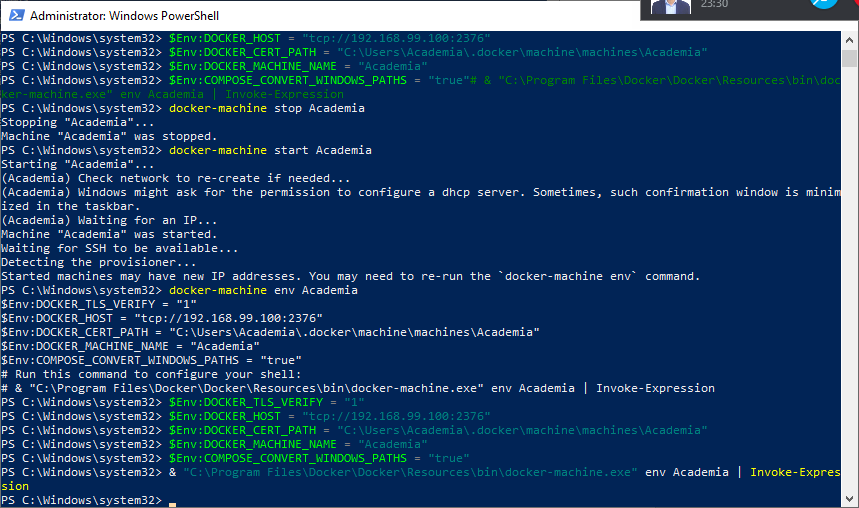


**7.- Iniciando Docker Machine**

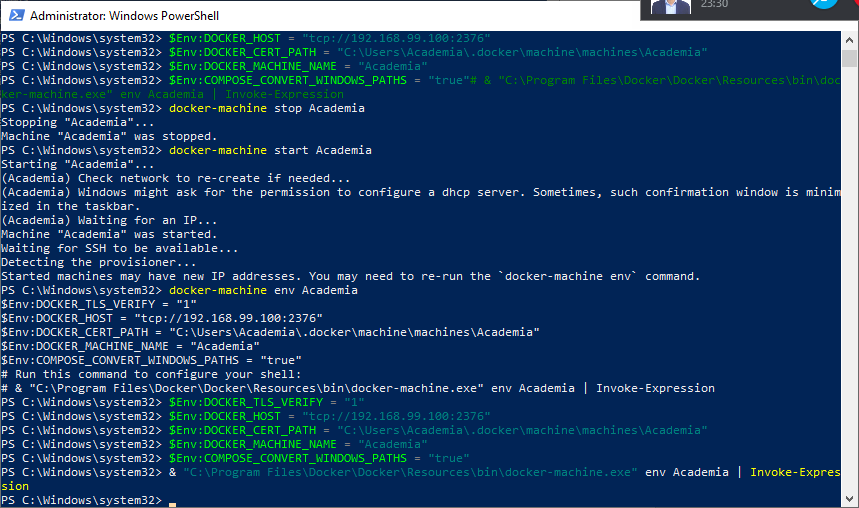
Primero se necesita echar a andar la maquina con el comando *docker-machine start*



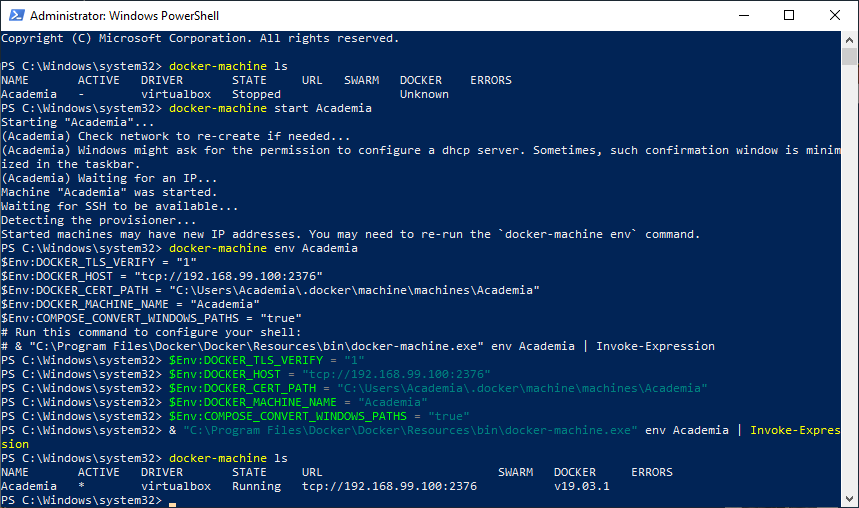
Después se utiliza el comando *docker-machine env* y mostrara unas líneas, las cuales se copian las primeras 5 y la última desde el signo “&”.



Después estas líneas se pegan en la consola como se muestra a continuación:

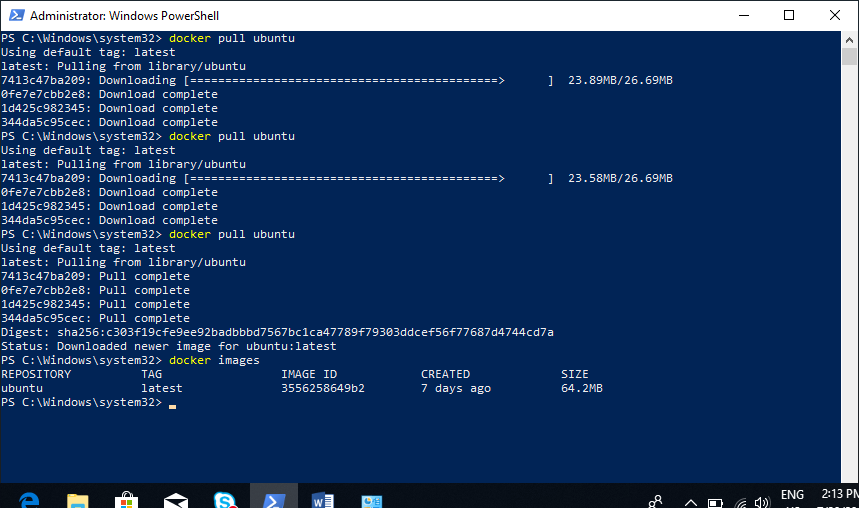


Para corroborar que la maquina inicio correctamente es listando de nuevo mostrando un asterisco en la columna de ACTIVE.

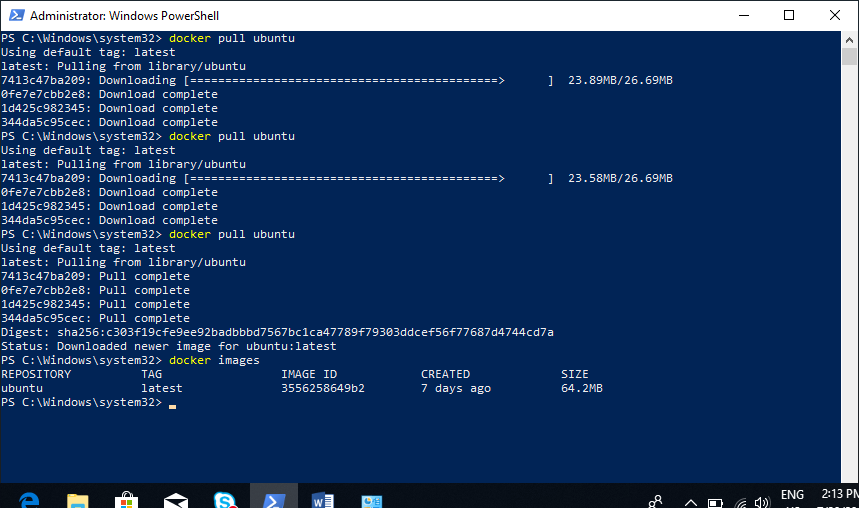


**8.- Descarga de imágenes en Docker**

Con el parámetro de pull podemos hacer una descarga de una imagen del repositorio de Docker

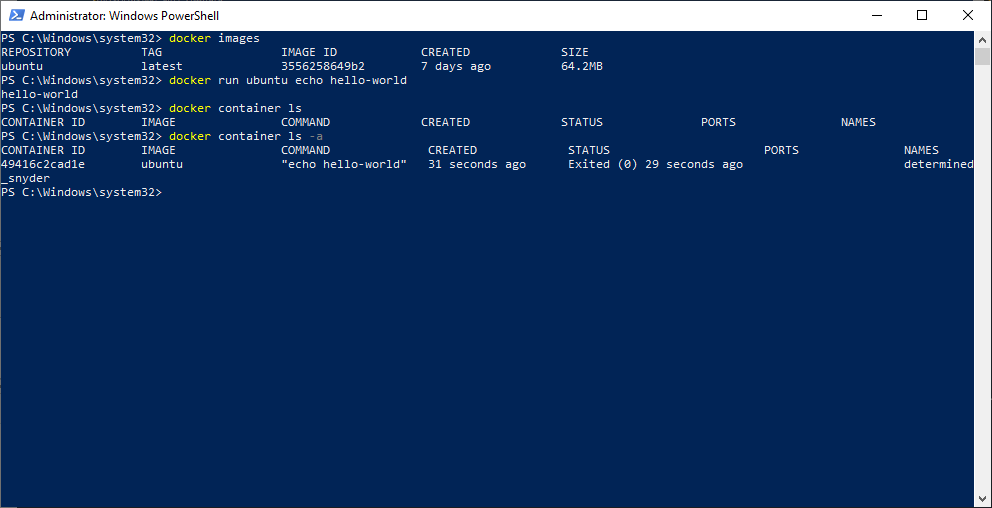


Para comprobar si la imagen se ha descargado se utiliza el parámetro *images* o *image ls*



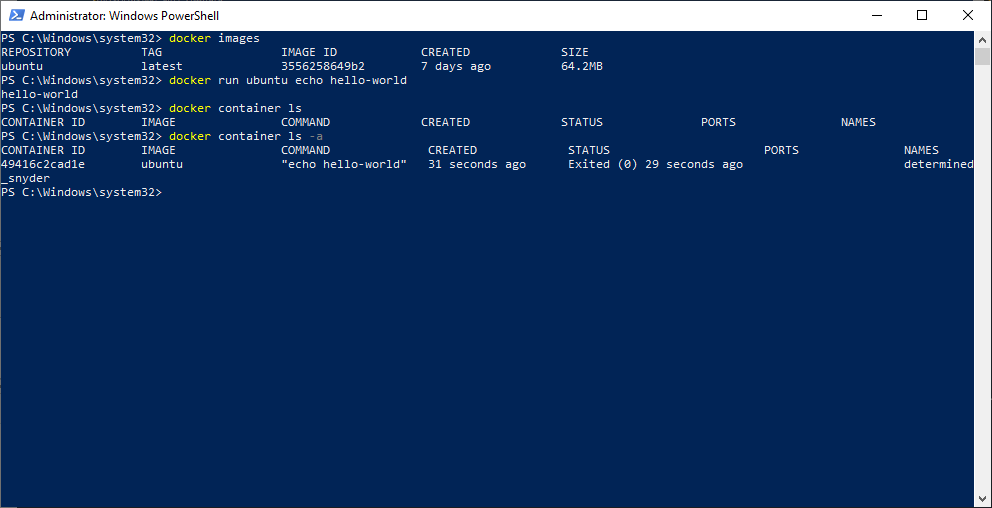
**9.- Correr un contenedor**

Para ejecutar una instrucción en un contenedor se utiliza el comando de docker run. En este caso solo imprimiremos un mensaje con la imagen de Ubuntu que descargamos.

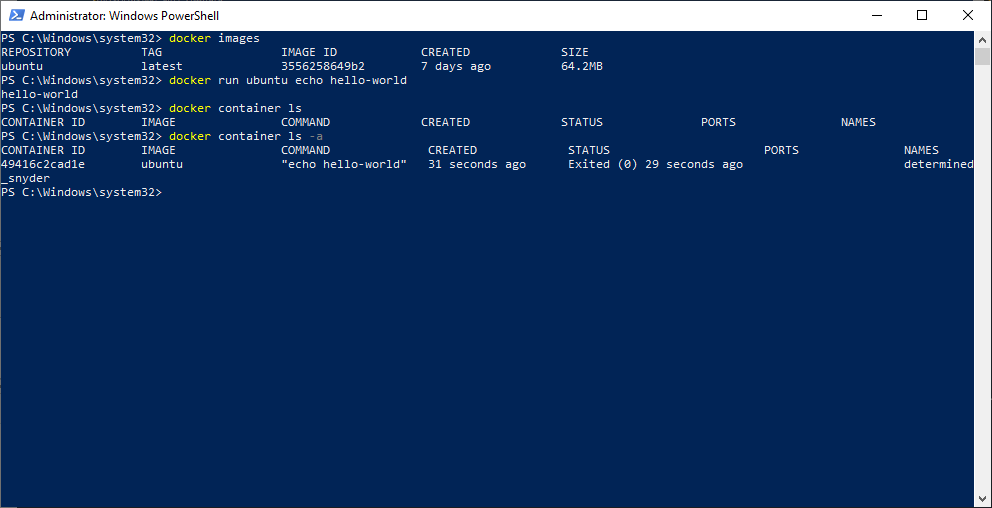


**10.- Visualizar contenedores**

Para visualizar los contenedores que se encuentran activos se utiliza el comando *docker container ls*



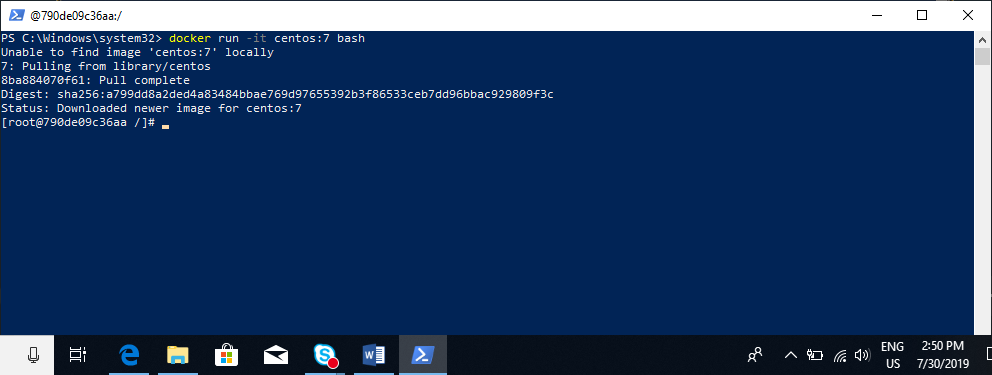
En este caso no se encuentra ningún activo ya que el existente ya termino su tarea. Para visualizar el existente se agrega el parámetro *–a*



**Actividad**

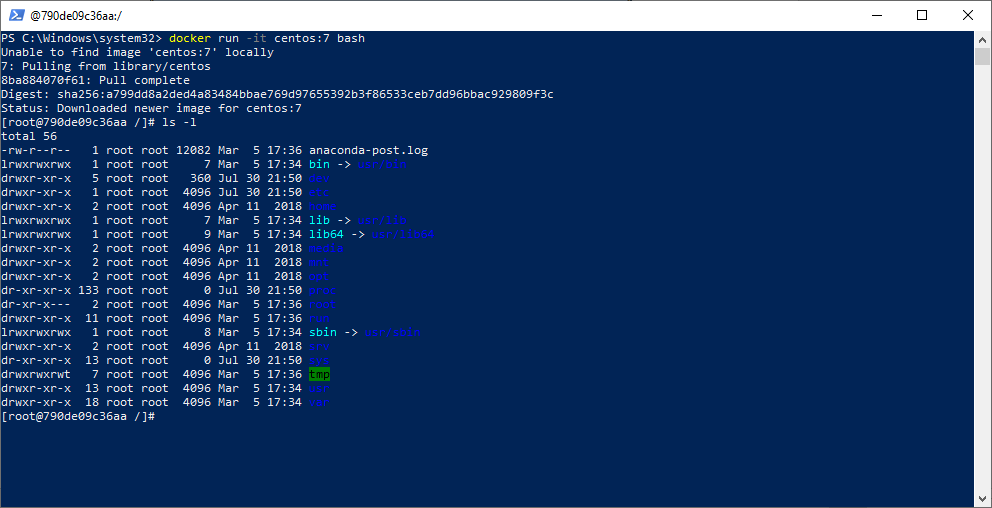
1. **Correr un nuevo contenedor con la imagen de Centos:7 sin usar el comando pull**

Al no tener la imagen ya descargada y se ejecuta el comando *run* con docker, este se dará cuenta que la imagen no está existente en el host local y bajará la Imagen automáticamente.



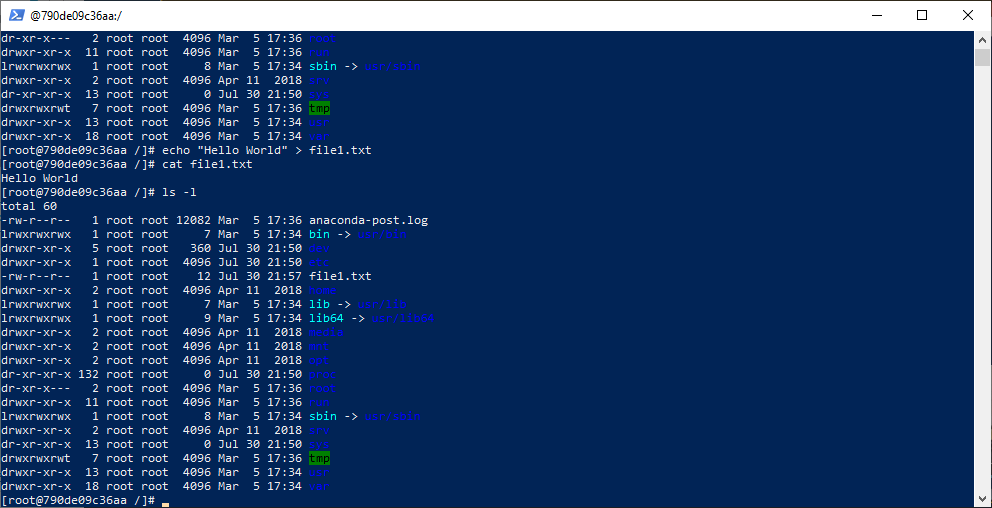
**2.- Listar la ubicación actual**

Para realizar esto se utiliza el comando *ls* y si se quiere mostrar más información como los permisos de los archivos que se encuentran se utiliza el parámetro *–l*

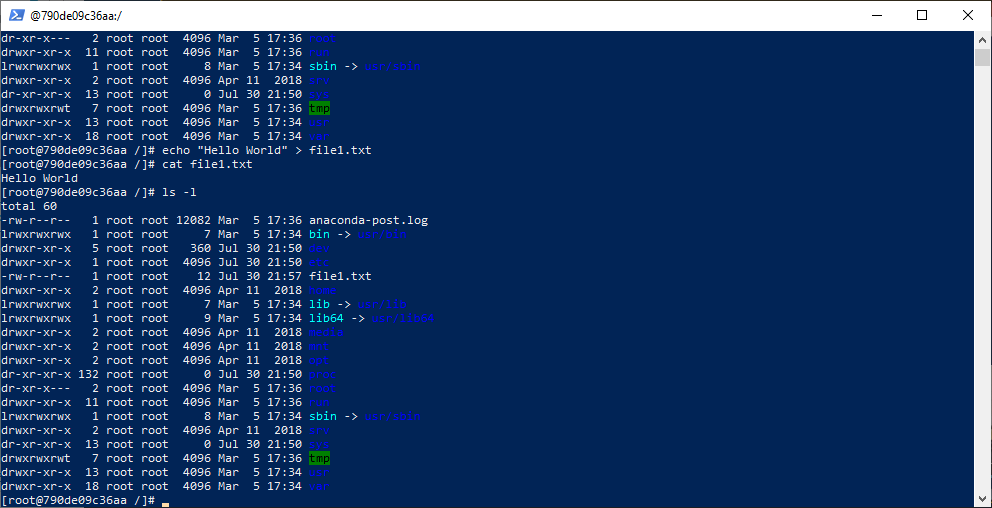


**3.- Crear un nuevo archivo llamado file1 con una frase “Hello World”**

Creando el archivo con el mensaje utilizando la instrucción echo con el mensaje e indicando que lo guarde en un archivo llamado *file1.txt*. Para ver que se guarde el contenido en el archivo se utiliza el comando cat.



Para corroborar que el archivo existe se enlistan de nuevo los archivos

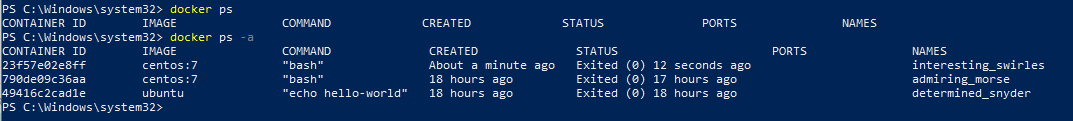


**4.- Salir del contenedor**

Para salir del contenedor solamente se utiliza el comando *exit*

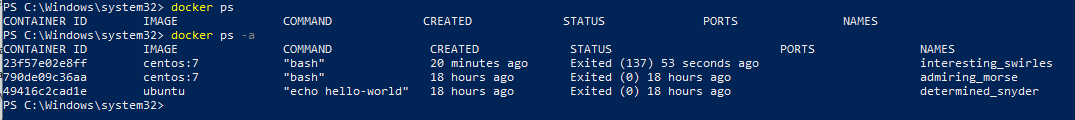


Para corroborar que nos salimos del contenedor usamos el comando de docker *ps* y nos muestra los contenedores activos, en este caso no salió ninguno ya que el contenedor ya lo cerramos. Pero con el comando *docker ps –a* nos muestra todos los contenedores que se han creado.

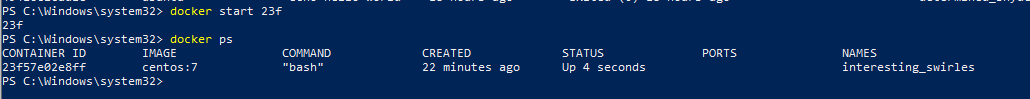


**5.- Iniciar un Contenedor con ID**

Primero verificamos cuales contenedores están activos y cuales tenemos con el comando *docker ps* y *docker ps –a*

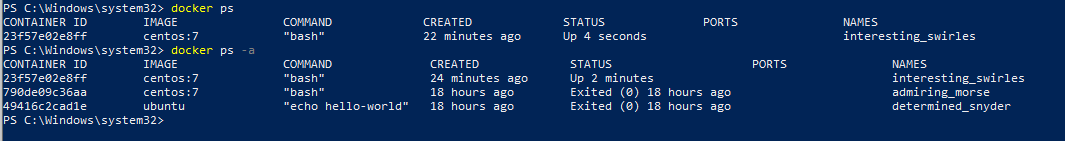


Para iniciar un contenedor con su ID se utiliza el comando *docker start* y se brinda el ID, basta con poner una parte de la cadena del ID única para iniciar el contenedor. Después para corroborar podemos utilizar el comando *docker ps* para ver los contenedores activos y se puede visualizar que el primer contenedor con la cadena 23f en su ID se inició.

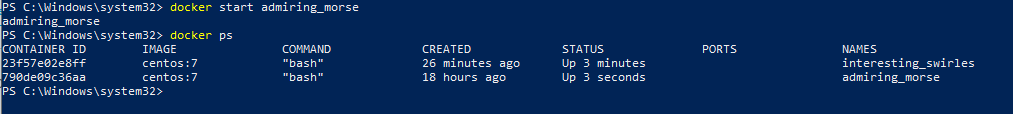


**6.- Iniciar un contenedor por su nombre**

Ahora se iniciará otro contenedor por medio de su nombre, en la imagen siguiente se muestra los contenedores activos y los que existen, se levantara el admiring\_morse

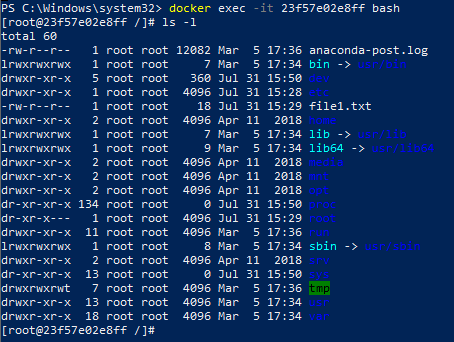


Con el comando *docker start admiring\_morse* se puso en marcha el contenedor con ese nombre y para corroborarlo se utiliza el comando *docker ps* para ver los contenedores activos.



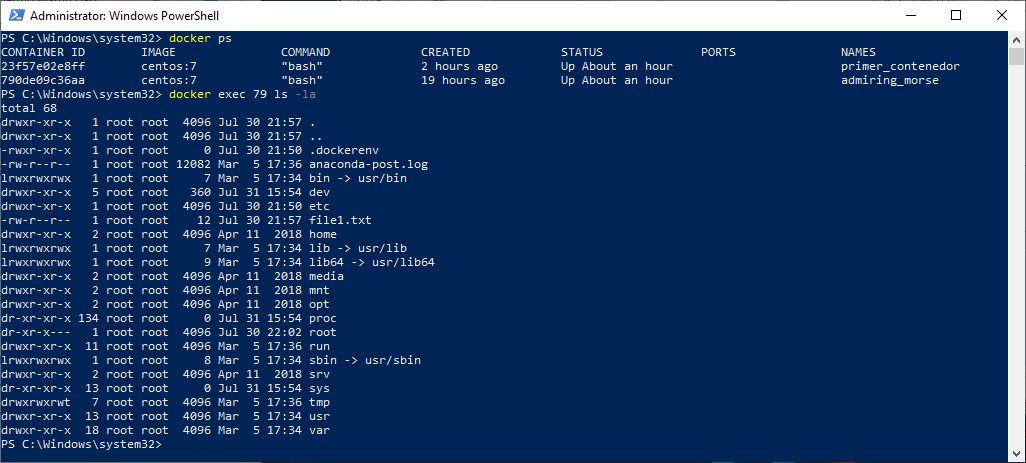
**7.- Entrar a un contenedor que se encuentra activo**

Después de activar los contenedores, para poder entrar a ellos o utilizarlos es con el comando *docker exec –it <ID>*



**8.- Ejecutar un comando en un contenedor sin entrar en el**

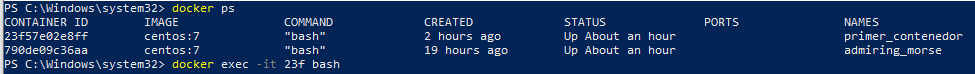
Podemos ejecutar comandos del contenedor Linux que existe utilizando el comando *docker exec <ID> <comando>* así como en la imagen donde se enlistan los archivos del contenedor sin entrar en él.



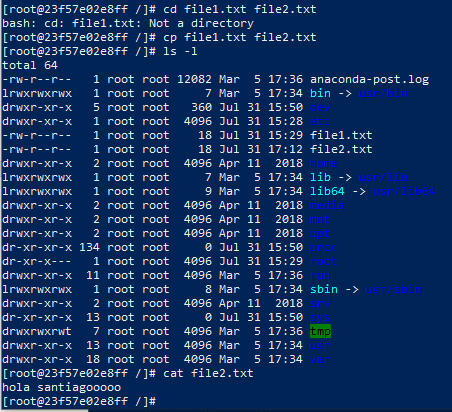
**9.- Copiar el archivo file1.txt que se encuentra dentro del contenedor**

El primer paso es entrar al contenedor con *exec –it* para entrar al contenedor y posteriormente usar el comando de copy *(cp)*.

Entrando a el contenedor con la cadena en el ID de 23f:



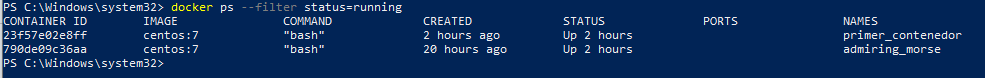
Copiando el archivo file1.txt y listando para visualizar que se hizo correctamente:



**10.- Listar los contenedores con filtros**

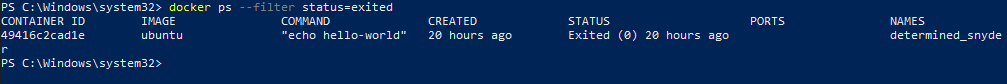
Se pueden fiktrar los contenedores utilizando diferentes parámetros, el primero es para visualizar los que tengan un status de ejecución:

*docker ps –filter status=running*



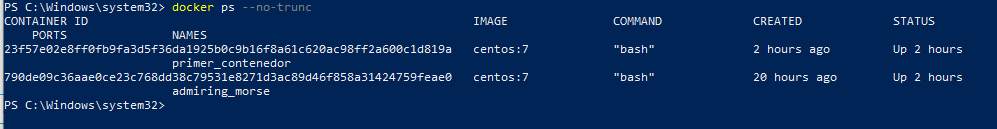
Ahora los que se encuentren cerrados:

*docker ps –filter status=exited*



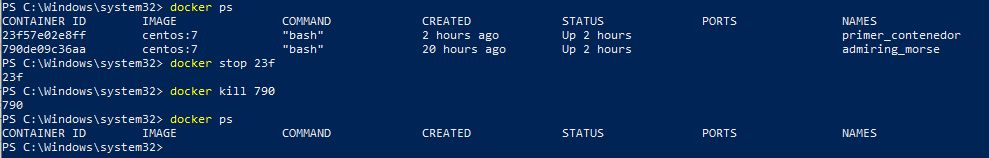
Ahora mostraremos todo el ID de los contenedores sin truncar:

*docker ps –no-trunc*



**11.- Parar los contenedores**

Existen dos formas de detener los procesos de los contenedores, una es con el comando *stop* y otra con el comando *kill.* En la imagen se ven los contenedores activos y como después ya no se muestran en la lista con *ps* después de utilizar los comandos para detenerlos.



**12.- Ejecutar un contenedor de modo interactivo y dejarlo corriendo en el background**

Para realizar este paso primero entramos a un contenedor de forma interactiva con el comando *run* y una vez dentro del contenedor se teclea:

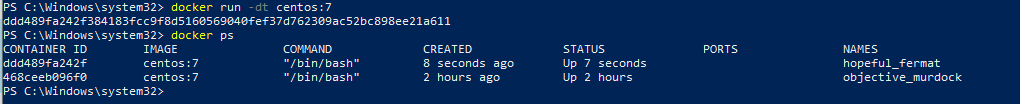
*Ctrl + P + Ctrl Q*



De esta forma se sale del contenedor, pero una vez fuera se observa que el contenedor se encuentra activo y está funcionando desde el background.

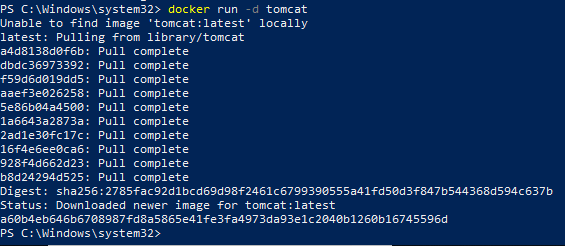
**13.- Ejecutar un contenedor y dejarlo corriendo en el background sin entrar en modo interactivo**

Para realizar esto se utiliza el comando *docker run* con los parámetros *–dt* y el sistema y versión para crear el contenedor y mandarlo al background.

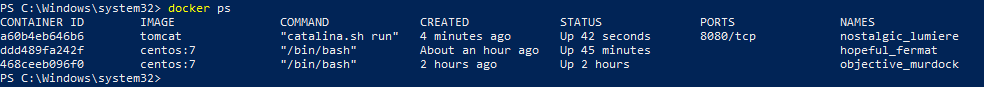


**Actividad 2**

**Correr un contenedor tomcat y dejarlo en el background**



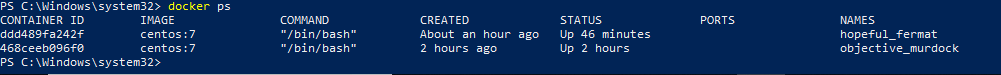
**Listando los contenedores donde se ve que se está ejecutando el nuevo contenedor de tomcat.**



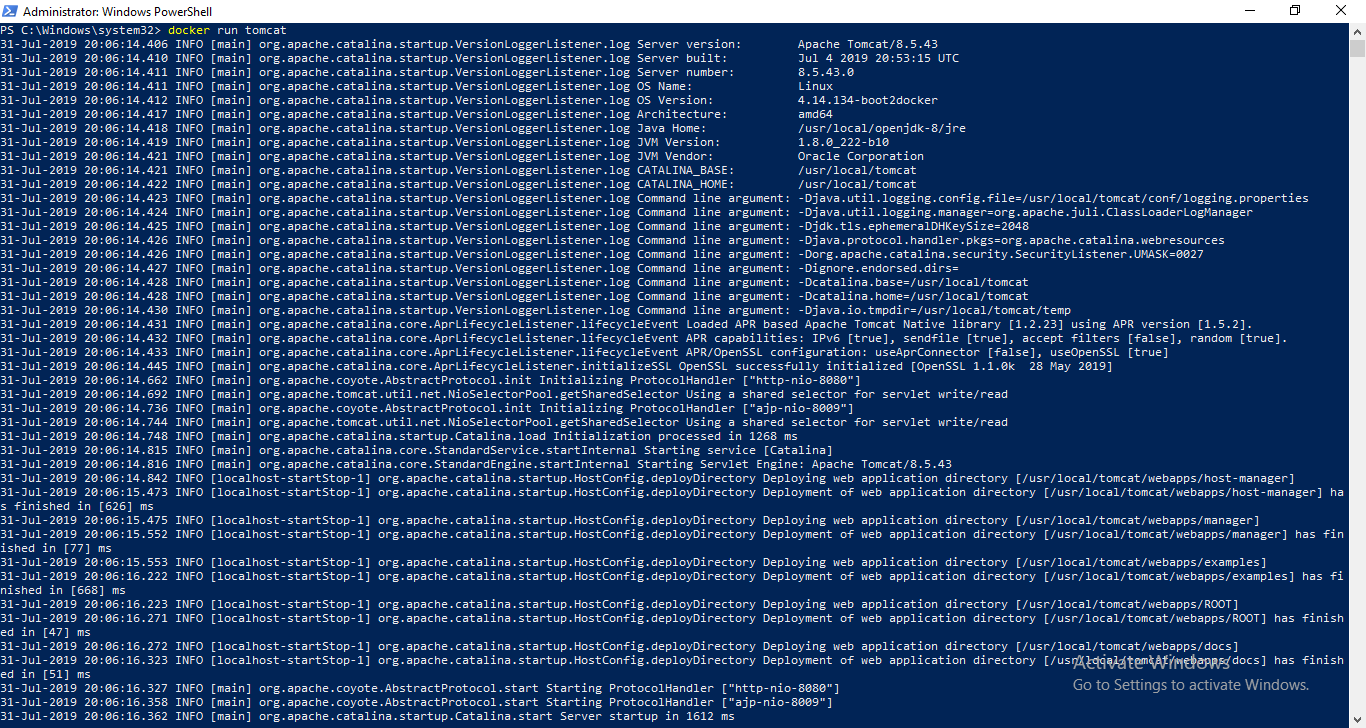
**Deteniendo el contenedor con Stop**



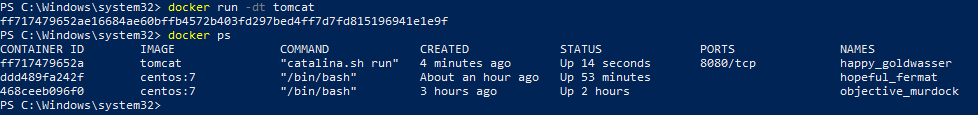
**Verificando que el contenedor ha sido detenido correctamente**



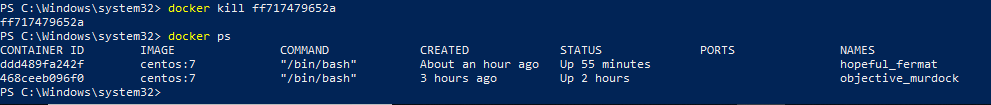
**Correr el contenedor tomcat y entrar**



**Reiniciar el contenedor y correrlo en el background**



**Detener el contenedor con Docker Kill**

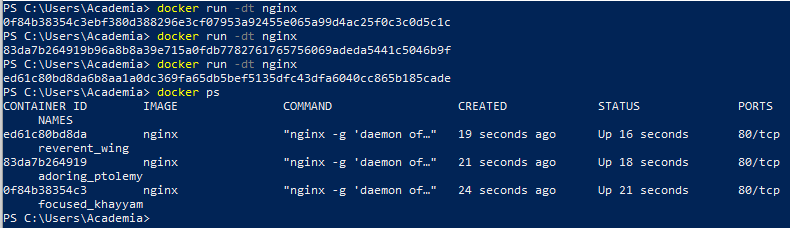


**Diferencia entre *docker stop* y *docker kill***

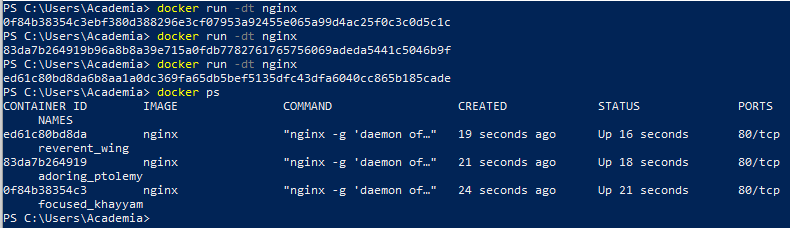
Con *docker stop* se manda un trigger que informa al contenedor de que sera detenido, de esta forma este detiene sus procesos de la forma debida y *docker kill* simplemente detiene los procesos sin importar que.

Actividad 3

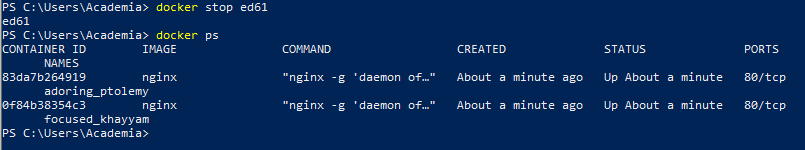
Iniciar tres contenedores en el background con la imagen de nginx



Detener el primer contenedor



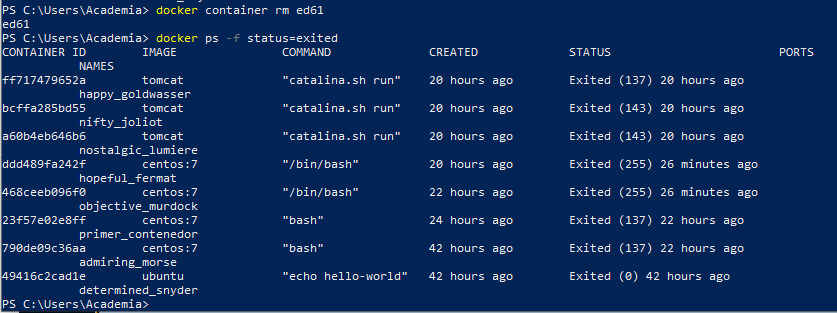
Con el comando *docker stop <ID>* se detendrá el primer contenedor, que en este caso su ID comienza con los caracteres únicos de ed61.



Listar los contenedores con el estado de *exited* utilizando los filtros

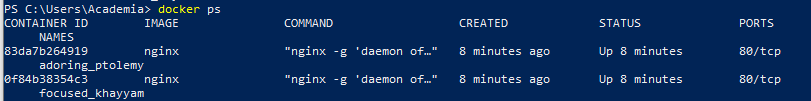


Eliminar el contenedor nginx que se detuvo con el comando *docker container rm*

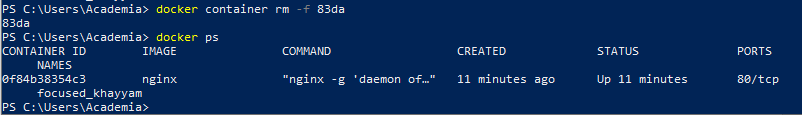


Eliminar el segundo contenedor creado con el comando *docker container rm –f <ID>*

Primero listamos los contenedores que quedan activos para revisar el ID



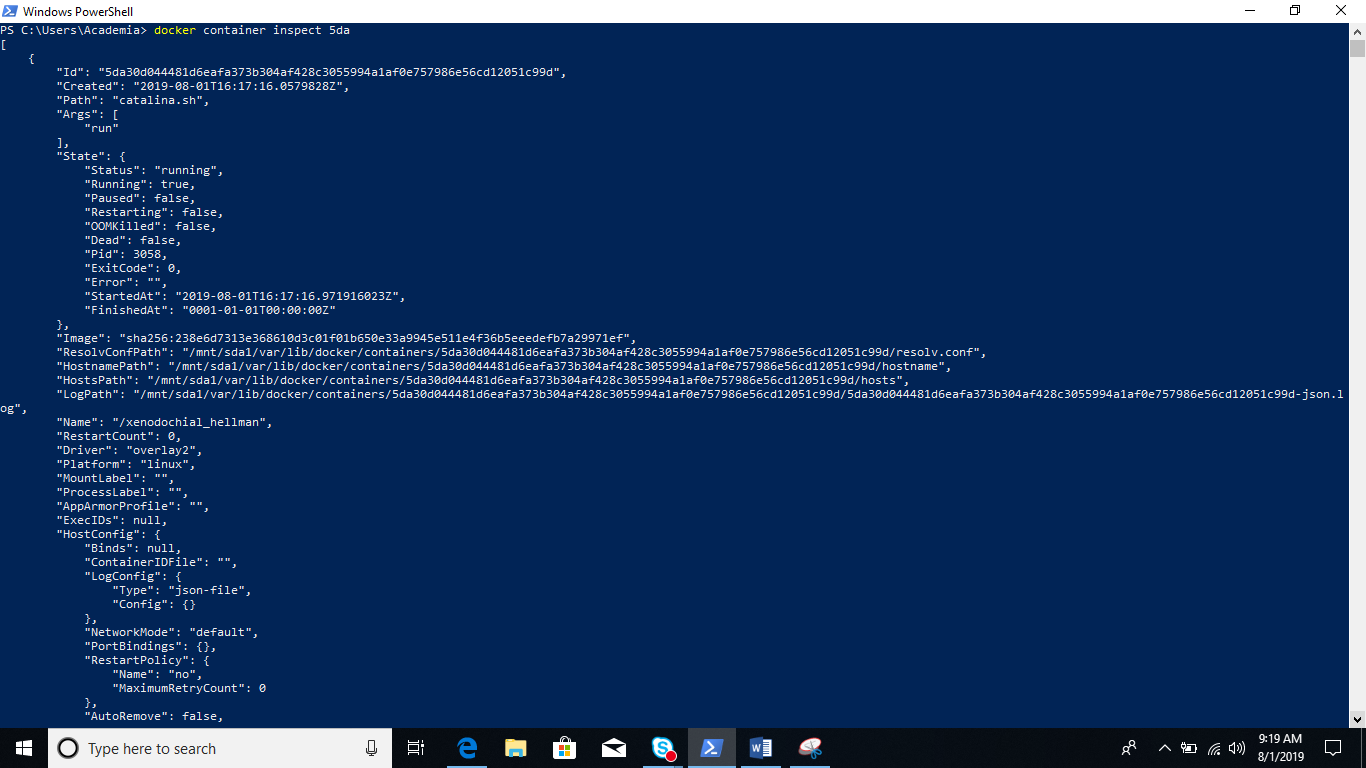
Y ahora como el contenedor se encuentra activo se utiliza el parámetro *–f* para forzar la eliminación, ya que sin el necesitaríamos detenerlo con *stop* y después borrarlo



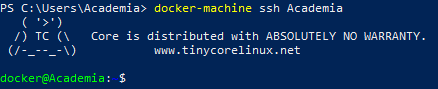
**Inspeccionar**

Iniciar un servidor tomcat

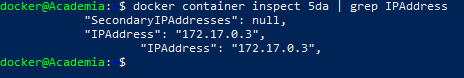


Inspeccionar los detalles del contenedor usando el comando *docker container inspect <CID>* esto nos regresara como resultado la información en formato Json.

(Buenas practicas: iniciar con ssh)



Inspeccionar el contenedor tomcat filtrando con *grep* y buscando la IP Address



Filtrar el contenedor con el grep buscando la información de Cmd. En este caso no muestra nada.



Utilizar una bandera para lograr mostrar

Utilizando el parámetro de *–format* se indica la información que se quiere buscar, ya que con grep no se puede acceder a la información. Para acceder a ella se le indica la búsqueda que se quiere hacer indicando que la información que queremos está dentro del argumento Config y queremos la información de Cmd.

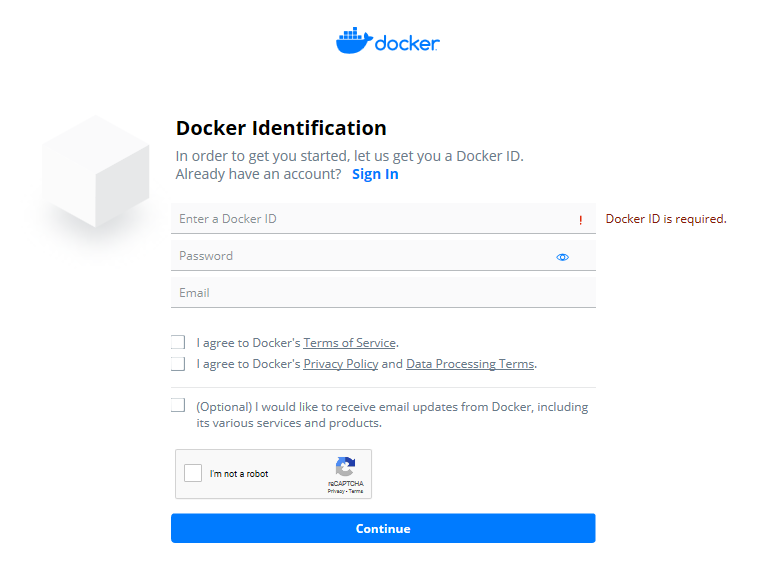


**Imagenes**

Creación de imagines, manejo y registro.

1.- Primero se debe crear una cuenta en Docker en el enlace

*<https://hub.docker.com/signup?next=%2F%3Fref%3Dlogin>*

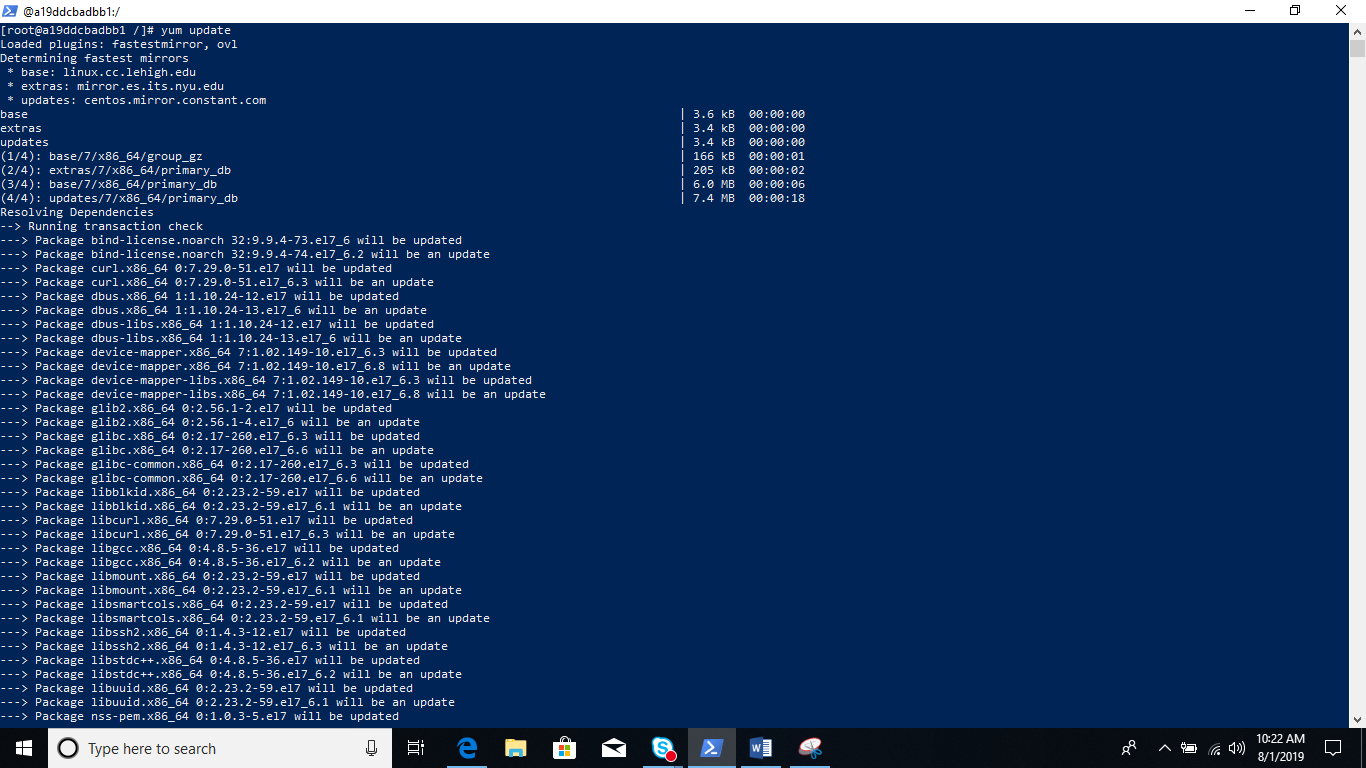


2.- Iniciar un contenedor de centOS con *docker container run –it centos:7 bash*

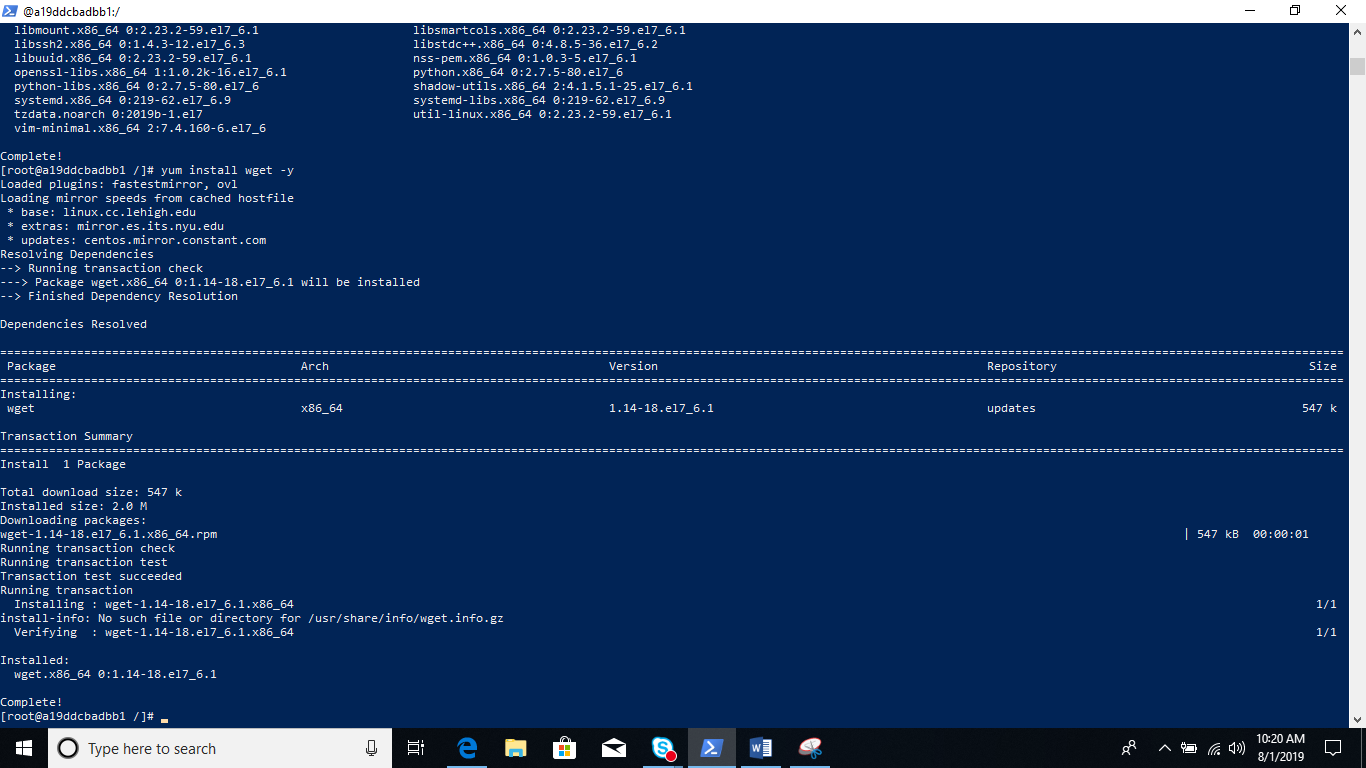


3.- Instalar vim y wget

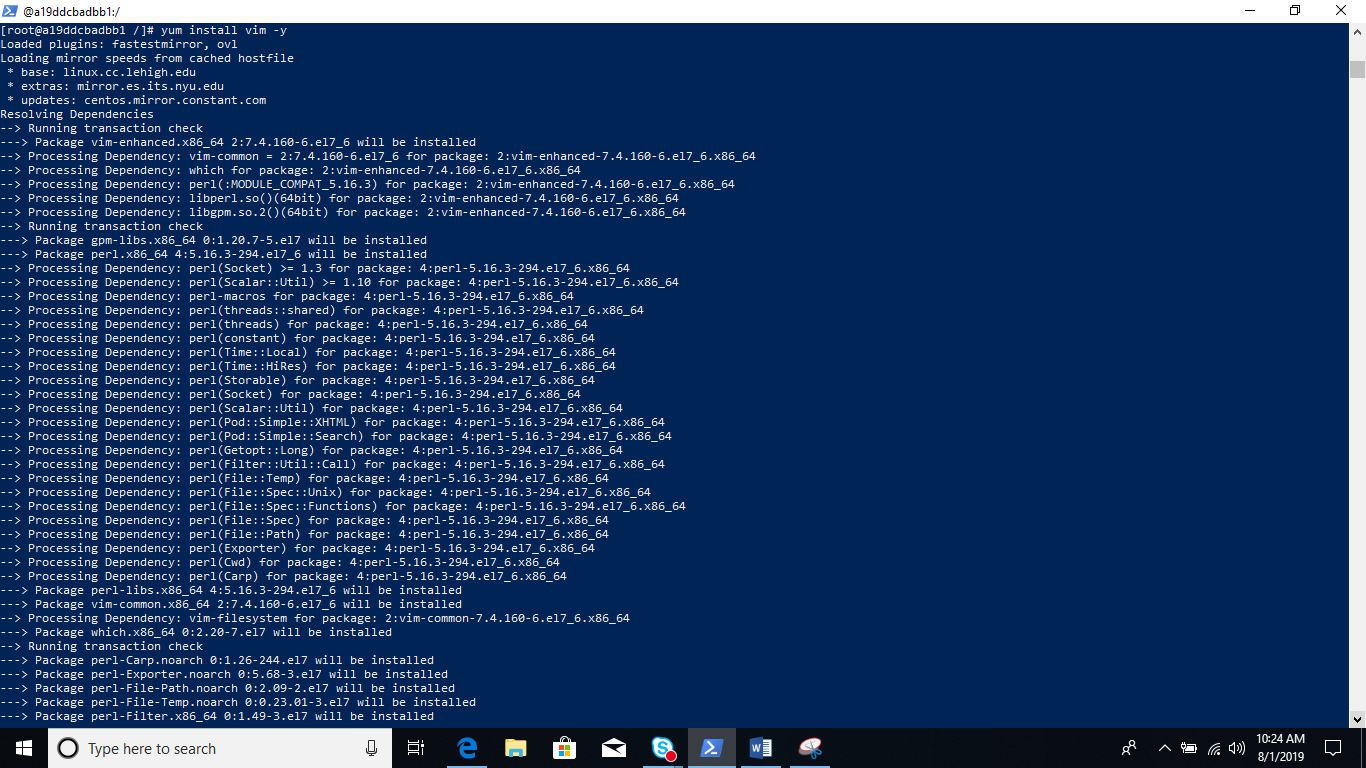
Antes de comenzar con la isntalacion se requiere actualizar el sistema con un *yum update*



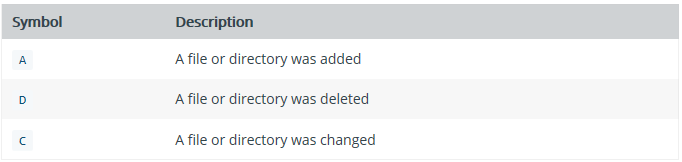
Instalando wget con el comando *yum install wget –y*



Instalando vim con el comando *yum install vim –y*

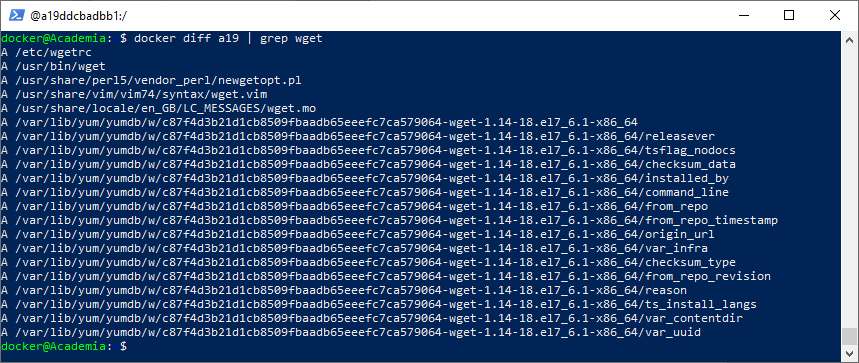


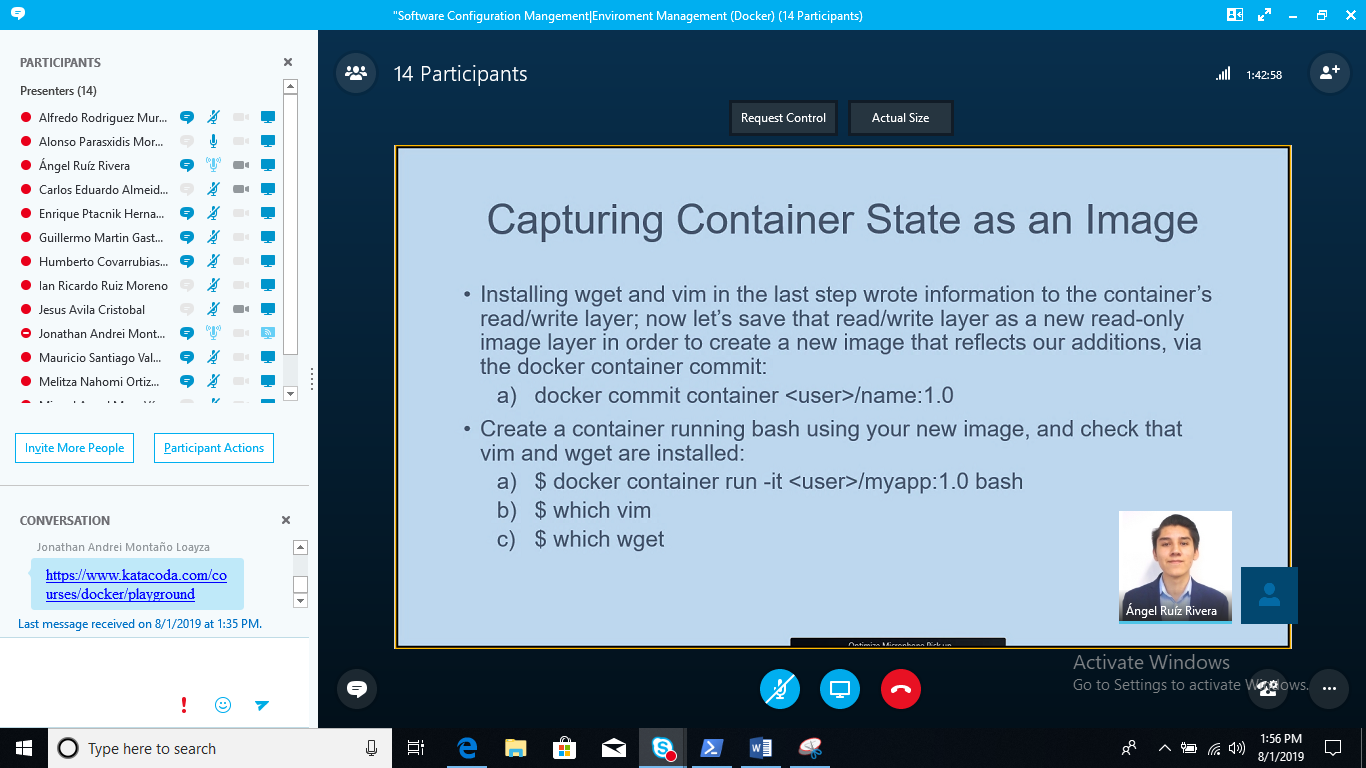
Visualizar los cambios del contenedor con *docker container diff <ID>* mostrando una lista con todos los cambios donce cada directorio muestra una letra al principio que significa lo siguiente según la documentación de Docker:





Se pueden realizar filtros en la búsqueda de modificaciones con el comando *grep*

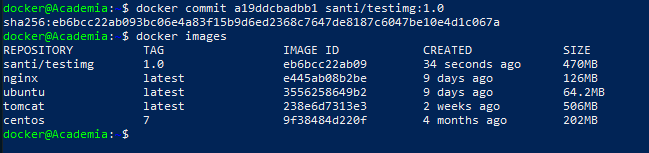




Una vez instalados vim y wget, estos son creados en la capa de lectura y escritura del contenedor. El siguiente paso es crearlo solo capa de lectura para crear copias de este contenedor para que hereden lo que este contenedor ya tiene, como los programas que se instalaron. Esto se logra con el siguiente comando

*docker commit a19ddcbadbb1 santi/testimg:1.0*

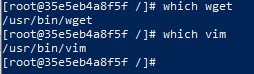
De esta forma ya creamos una imagen en base al contenedor ya creado y con los programas instalados, como se muestra a continuación.



Ahora se crea un contenedor en base a la nueva imagen creada con el comando *docker container run –it <user>/myapp:1.0 bash*



Para corroborar que se realizó correctamente este debería tener los programas instalados en el contenedor anterior, para esto se utiliza el comando *which* para localizar la carpeta donde se ubican los programas *wget* y *vim*.

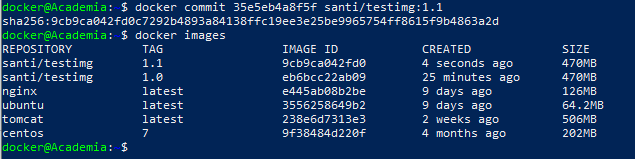


Crear un archivo dentro del contenedor

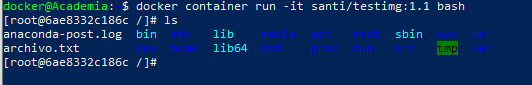


Hacer este contenedor una nueva imagen con el tag 1.1

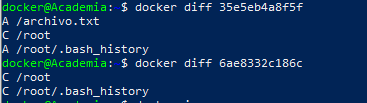
*docker commit 35e5eb4a8f5f santi/testimg:1.1*



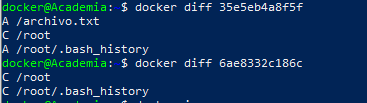
Creando un nuevo contenedor con la nueva imagen y verificando que este el archivo



Revisando los cambios del contenedor utilizando *docker diff* primero del contenedor con la imagen 1.0 donde en este contenedor se creó un archivo y como se observa a comparación de la imagen con la que se creó el archivo es la diferencia.



Después de eso se creó el contenedor con la imagen 1.1 donde ya se encuentra el archivo y usando el comando no muestra ningún cambio, solamente cambios en el bash ya que se entró al contenedor y se modificó el history del bash.



**Dockerfile**

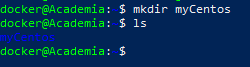
Una imagen se genera a través de un dockerfile y con un comando llamado docker build. Un docker file es un archivo de texto que genera una imagen. Con un docker file contiene todas las instrucciones necesarias para generar imágenes personalizadas brindando un mayor control y orden sobre una imagen.

Comandos

* **From:** Primera línea que debe tener el dockerfile. Este es la base de la imagen.
* **Maintainer:** Comando opcional. Provee el nombre de quien da mantenimiento a la imagen.
* **Run:** Este se utiliza para ejecutar instrucciones durante el proceso para generar la nueva imagen.
* **Copy/Add:** Copian un archivo o directorio a la imagen.
* **ENV:** Define la variable de entorno.
* **WORKDIR:** Situa el directorio de trabajo
* **CMD:** Instrucción que se va a leer cuando se ejecuta un contenedor de esa imagen.
* **ENTRYPOINT:**

**Actividad**

1.- Crear un nuevo folder llamado myCentos



2.- Crear un nuevo archivo llamado Dockerfile



3.- Dentro del archivo agregar las instrucciones

1. Crear una nueva imagen con Centos 7



1. Actualice el sistema



1. Instalar wget



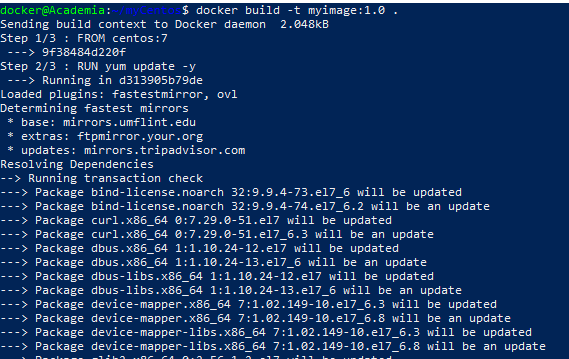
Contenido del archivo Dockerfile



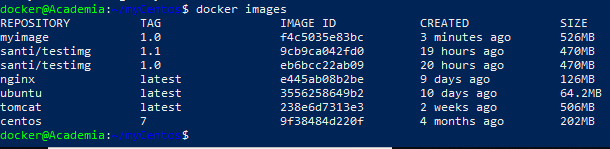
4.- Crear la imagen con el comando *docker build –t myimg*

Primero se entra en la carpeta creada llamada myCentos y se ejecuta el siguiente comando donde el último argumento es el URL donde se encuentra el archivo Dockerfile y de esta forma solo el path es con un punto indicando que el archivo esta donde nos encontramos.

*docker build -t myimage:1.0 .*

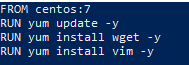


Comprobando si se creo la imagen con *docker images*

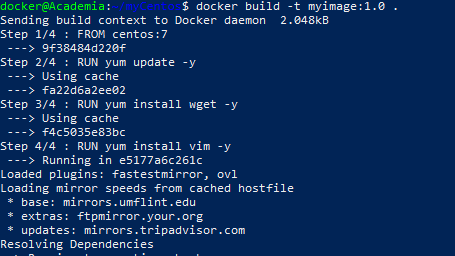




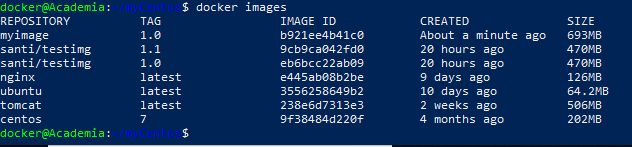
5.- Agregar instrucciones en el Dockerfile para isntalar vim



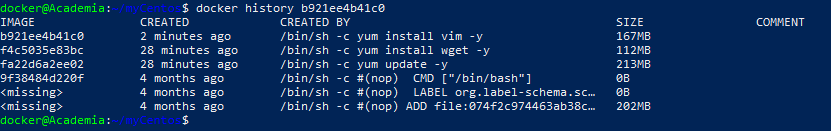
6.- Reconstruir la imagen



Comprobando que se creó correctamente con *docker images*

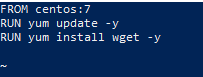


7.- Inspeccionar la imagen con el comando *history*

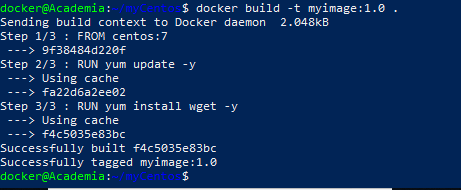


8.- Borrar el paso de la instalación de vim

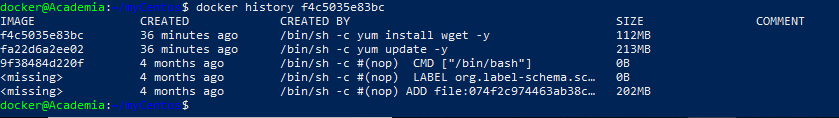
Borrando el comando RUN



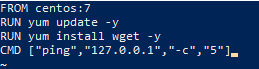
Recreando la imagen sin la instalación de vim



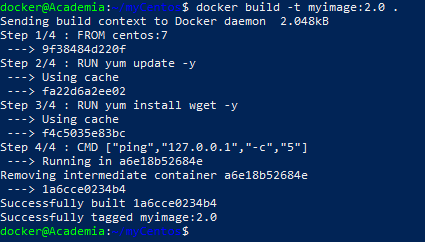
9.- inspeccionar el historial de la imagen



10.- Agregar la instrucción CMD [“ping”, “127.0.0.1”, “-c”, “5”]

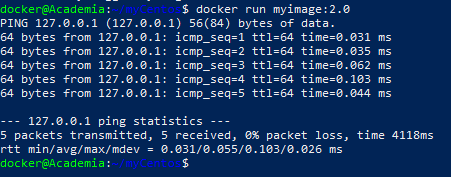


11.- Reconstruir la imagen



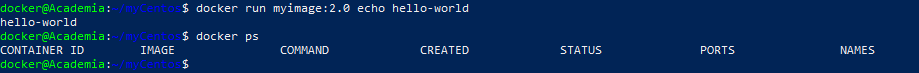
12.- Correr un contenedor con esa imagen con *docker run <nombreImagen>*

Al ejecutar el nuevo contenedor realiza un ping al localhost (127.0.0.1) y al terminar de hacer el ping termina su cometido y se cierra el contenedor.

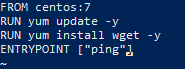


13.- Hacer que al ejecutar el contenedor se ejecute un comando de *echo “Hello World”*

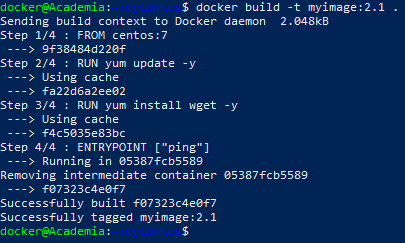
El poner en el comando al crear el contenedor este se sobrescribe. Solamente puede existir un comando CMD y se tomara en cuenta el ultimo, en este caso no tomo en cuenta el que se usó para el ping ya que el ultimo comando CMD era el *echo*.



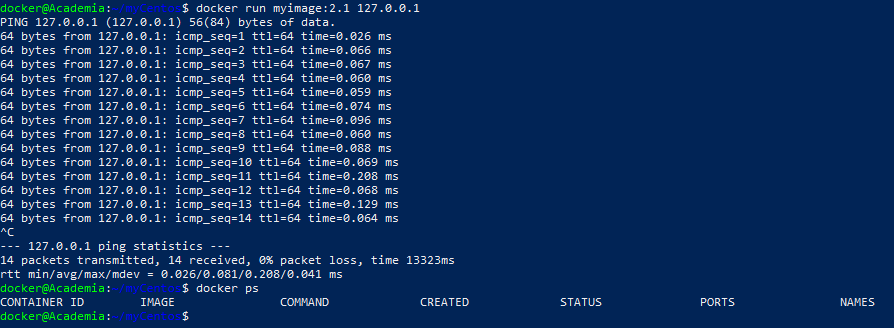
14.- Remplazar la línea en el Dockerfile de CMD por *ENTRYPOINT [“ping”]*



15.- Construir una imagen y correr un nuevo contenedor



16.- Probar levantando un contenedor y como último parámetro la dirección IP 127.0.0.1



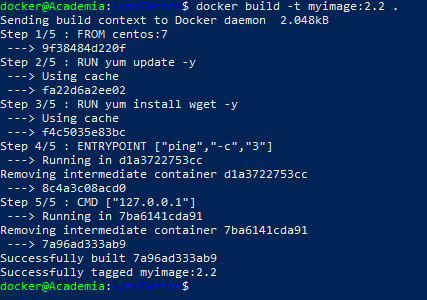
17.- Abir el Dockerfile y agregar los argumentos *ENTRYPOINT [“ping”,”-c”,”3”]*



18.- Agregar el comando CMD [“127.0.0.1”]

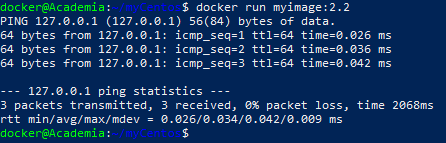


20.- Construir una nueva imagen

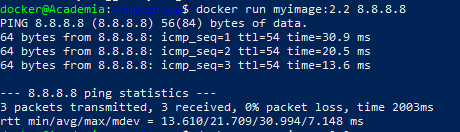


21.- Correr otro contenedor solo con *docker run <ImageName>*

Se creo un contenedor sin mandar un argumento al final de la instruccion y se hace ping a la IP especificada en el archivo Dockerfile en el comando de CMD y hace ping a esa direccion.



Pero si le mandamos como argumento al final de la instrucción otra dirección IP como a google que es 8.8.8.8, realizara el ping a la dirección puesta en el comando final ya que sobrescribe el ultimo comando CMD puesto.

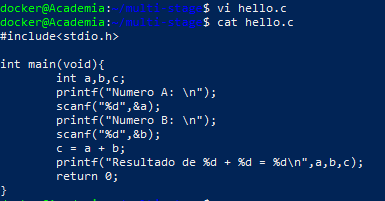


Actividad 2

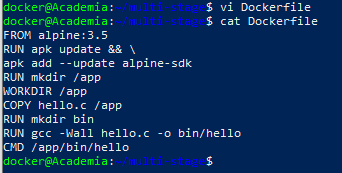
1.- Crear una nueva carpeta llamada multi-stage



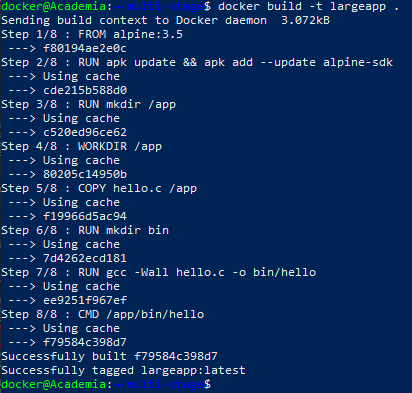
2.- Crear un archivo dentro en c llamado hello.c

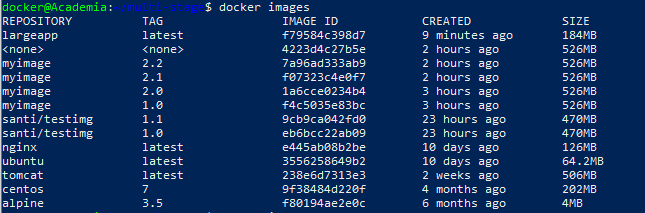


Crear un nuevo archivo Dockerfile:

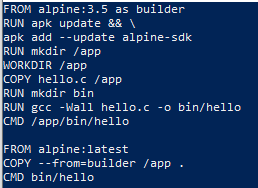


Generar la nueva imagen con el Dockerfile

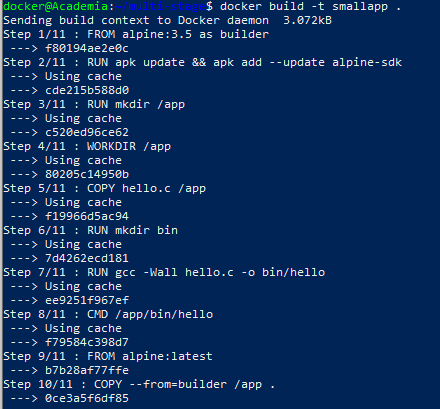




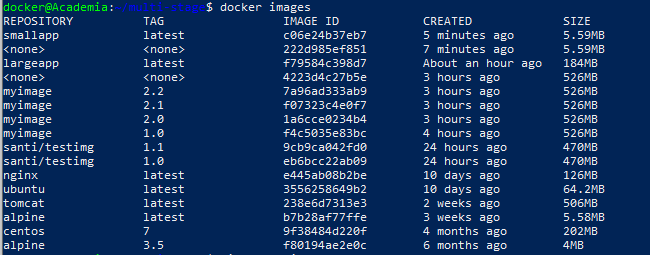
Nuevo Dockerfile con multi-stage



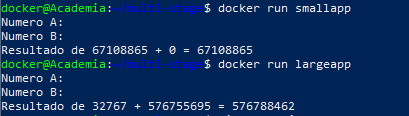
Creando la nueva imagen con el nuevo Dockerfile



Comparando las versiones y como se puede observar hay un cambio entre la versión *largeapp* que pesa 184MB y la *smallapp* que se redujo a solo 5.59MB ya que se remueve todo lo que no se indicó en el segundo stage



Comprobando que funcionan igual



**Docker Volumes y Mounts**

**Bind mounts:** A bind mount is a file or folder stored anywhere on the container host filesystem, mounted into a running container. The main difference a bind mount has from a volume is that since it can exist anywhere on the host filesystem, processes outside of Docker can also modify it.

**Volumes:** Volumes are the preferred way to store persistent data Docker containers create or use. The host filesystem also stores volumes, similar to bind mounts. However, Docker completely manages them and stores them under C:\ProgramData\docker\volumes by default.

La **diferencia** entre ellos dos es que con un bind podemos almacenar la información en la maquina host y manejarla nosotros, mientras que con volumes Docker es el que las maneja.

1.- Creando un nuevo volumen



2.- Listando los volúmenes existentes



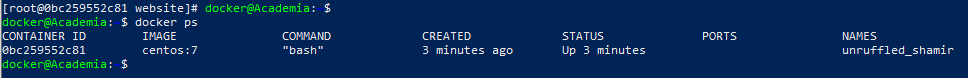
3.- Crear un nuevo contenedor con Centos 7 donde el maping se hara en la ubicación web de centos y ubicarnos en la ubicación del volumen



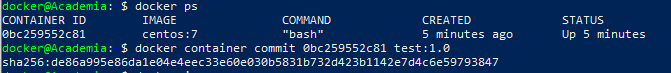
4.- Generar un archivo dentro de esa ubicación

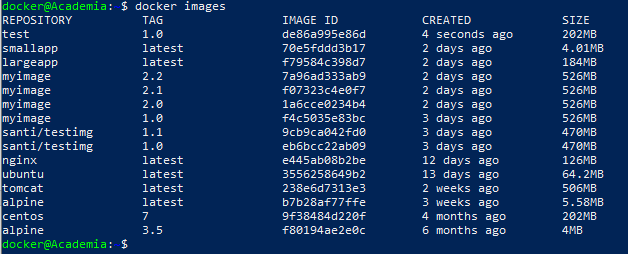


5.- Salirnos del contenedor dejándolo en el background utilizando Ctrl P + Ctrl Q

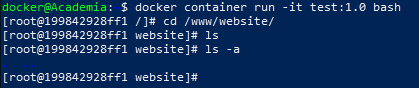


6.- Crear una nueva imagen en base al contenedor ya cerado de Centos 7



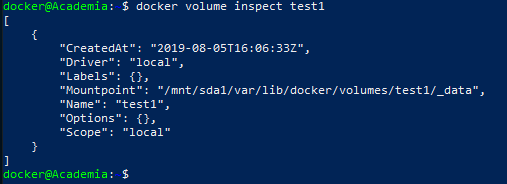


7.- Levantar un nuevo contenedor con la nueva imagen test que se creó y listar el contenido en /www/website/

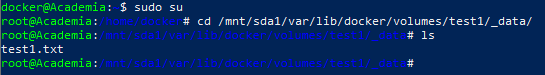


No muestra ningún archivo a pesar de haber creado anteriormente el archivo test1.txt. Esto es porque ese archivo se creó almacenado en el volumen creado y como a este contenedor no se le monto ningún volumen, entonces este no se muestra.

8.- Inspeccionar el volumen que se genero



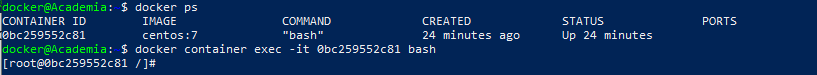
9.- Incrementar los permisos con *sudo su* a la dirección que muestra la información del volumen en *Mountpoint* y ver si se encuentra el archivo creado con *ls*.



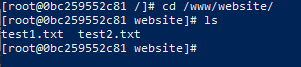
10.- Crear un archivo dentro de esa dirección llamado *test2.txt test2.txt* y salirse del super usuario con *exit*.



11.- Entrar al contenedor que se mantiene corriendo en el background



12.- Listar el contenido de la carpeta /www/website/



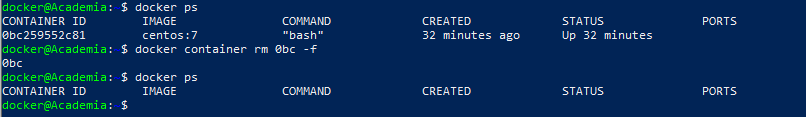
Muestra los dos archivos ya que se modificó el archivo directo ahí en el volumen

13.- Borrar un volumen con el comando *docker volumen rm <Nombre>*



No permite borrarlo ya que hay un contenedor existente utilizando ese volumen.

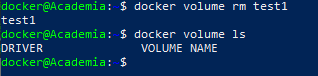
14.- Borrar el contenedor que está utilizando el volumen



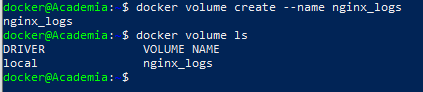
15.- Verificar si están los archivos en el volumen



16.- Borrar el volumen test1

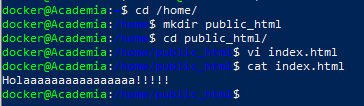


17.- Crear un nuevo volumen llamado *nginx\_logs*



18.- Crear una carpeta llamada *public\_html*

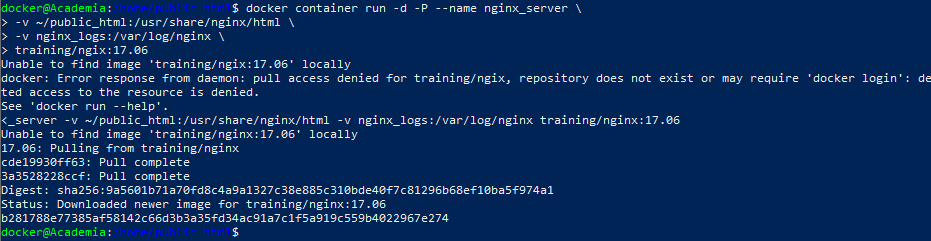
1. Dentro de esa carpeta crear un archivo llamado *index.html*
2. Ver el contenido del archivo



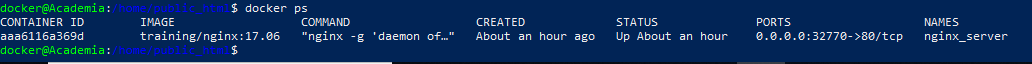
19.- Correr una imagen de *nginx* y mapear la carpeta creada de public\_html utilizando el siguiente comando:

docker container run -d -P --name nginx\_server \  
-v ~/public\_html:/usr/share/nginx/html \  
-v nginx\_logs:/var/log/nginx \  
training/nginx:17.06

docker container run -d -P --name nginx\_server -v ~/public\_html:/usr/share/nginx/html -v nginx\_logs:/var/log/nginx training/nginx:17.06

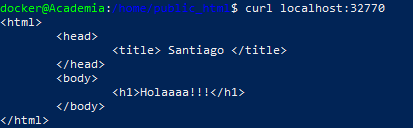


20.- Verificar que se levantó el contenedor y revisar el puerto donde fue mapeado que es asignado aleatoriamente y vinculado al puerto 80.

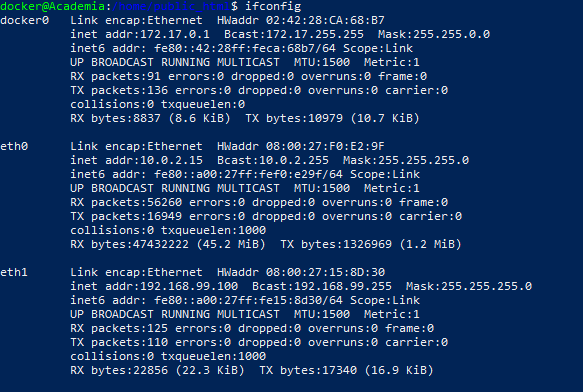


21.- Acceder al servidor nginx en el buscador

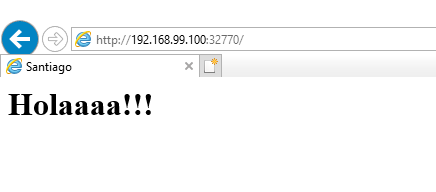
Para probar que funciona utilizando *curl*



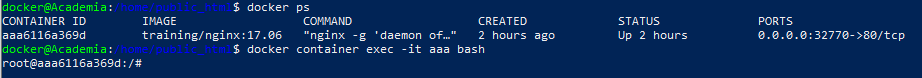
Para poder acceder se requiere indicar la dirección IP de la docker machine con *ifconfig*

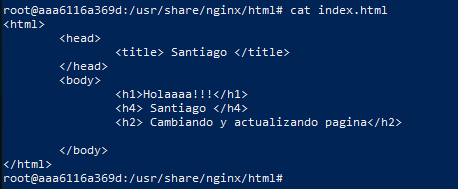


Ingresando en el navegador la dirección IP y el puerto se podrá visualizar el archivo index

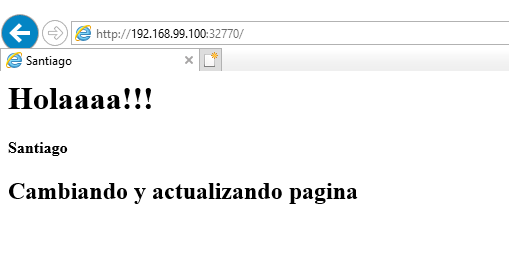


22.- Abrir el contenedor y agregar algo nuevo al archivo index.html del contenedor





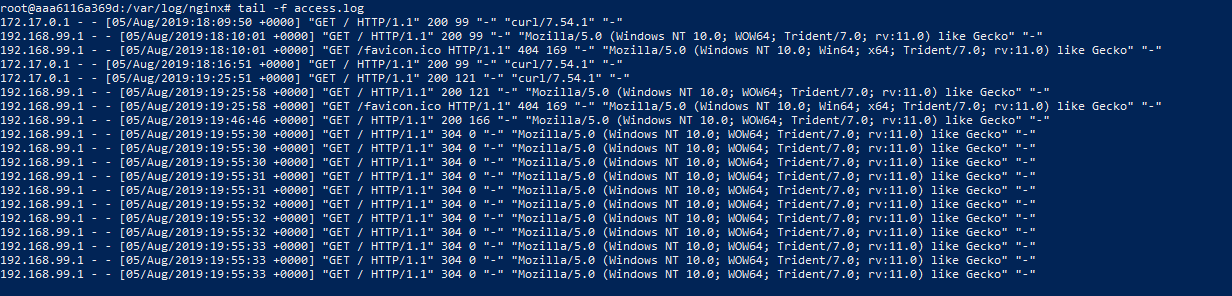
Verificando que se actualizo la pagina



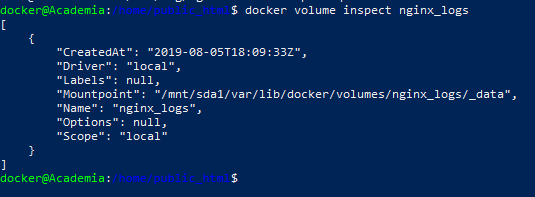
23.- Listar el contenido dentro de nginx



24.- Para revisar el uso del sitio se puede usar el comando tail donde nos muestra la hora y la dirección IP que ingreso al sitio.

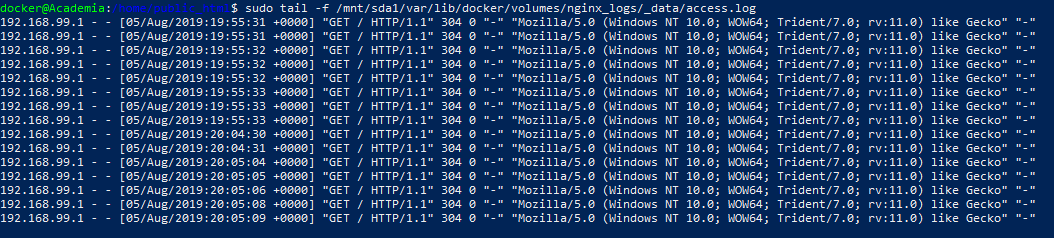


25.- Inspeccionar el contenedor para saber donde esta montado



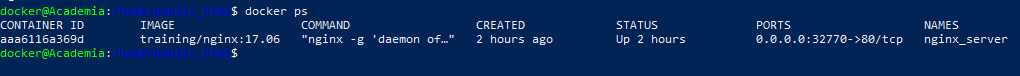
26.- Realizar lo mismo dentro de los archivos del volumen. Verificando si se registran los accesos dentro del archivo Access.log dentro del directorio donde está montado el volumen.





**Compartir volúmenes**

26.- Comprobar que siga corriendo el contenedor

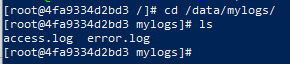


27.- Levantar otro contenedor



En este caso al final de nombrar el volumen tiene la extensión *:ro* esto significa *read only* lo que signidica que solo es de lectura y no podrá realizar cambios.

28.- Listar los archivos dentro de */data/mylogs*



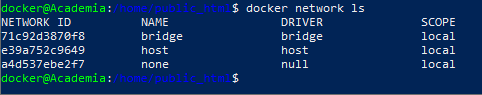
29.- Intentar crear un archivo dentro de ese directorio



Mostro un error ya que el volumen se montó de solo lectura y este contenedor no tiene el permiso para hacer modificaciones en el.

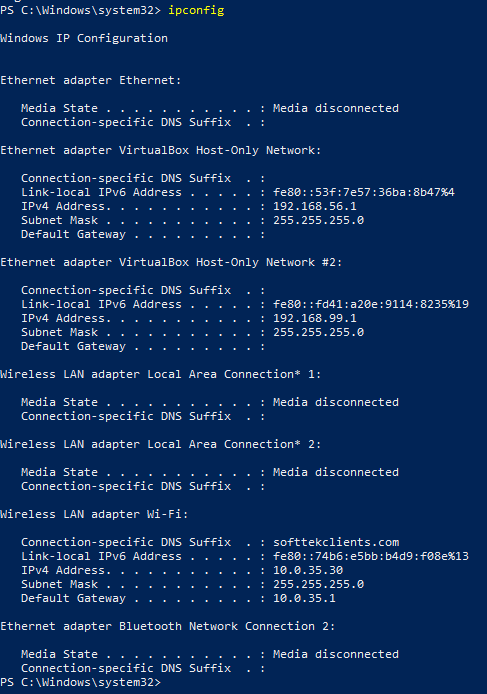
**Comunicación entre contenedores (Container Networking)**

1.- Revisando las redes o conexiones que se crean por default con Docker

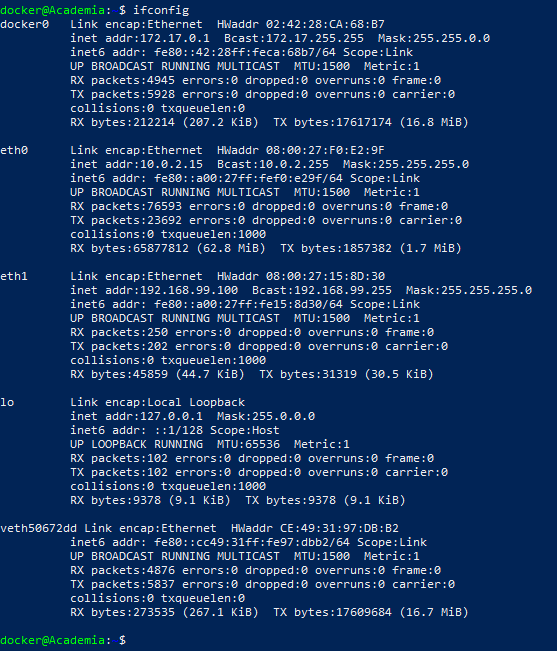


2.- utilizar el comando ifcongif dentro y fuera del docker machine

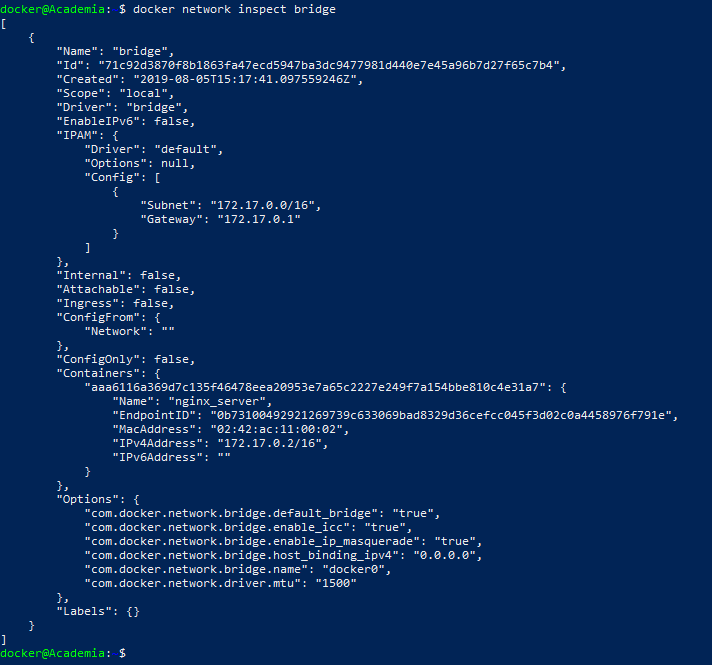
Fuera



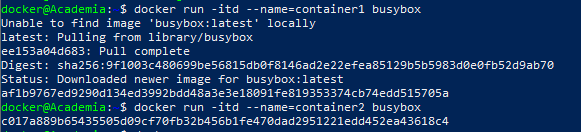
Dentro de la docker machine



3.- Inspeccionar la network bridge

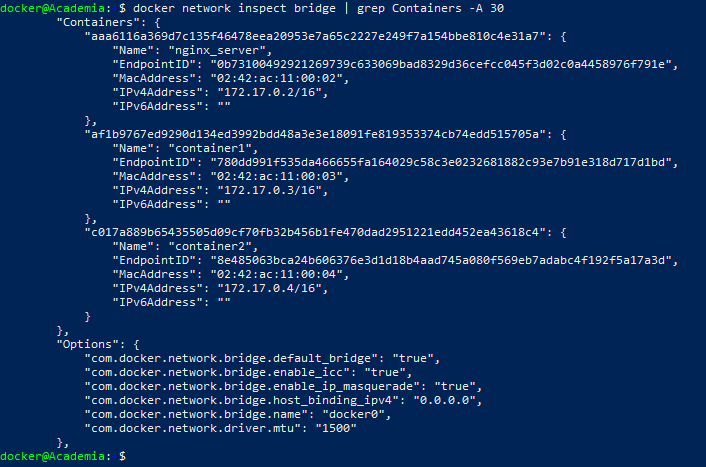


4.- Levantar dos contenedores de busybox el primero llamado contenedor 1 y el segundo contenedor2





5.- Inspeccionar la network bridge para verificar que estén ahí los contenedores creados

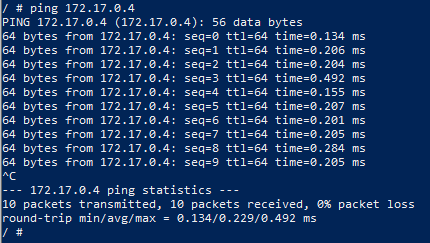


6.- realizar un docker attach al contenedor 1

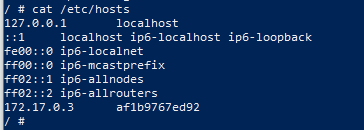


7.- Revisar si hay comunicación entre el contenedor 1 y 2 realizando un ping donde el ip del contenedor 2 es 172.17.0.4

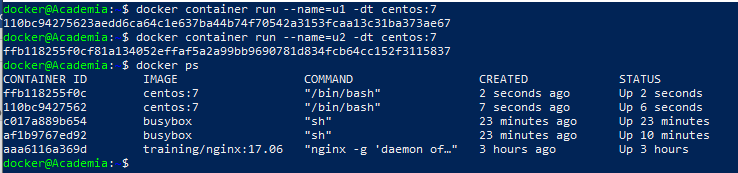




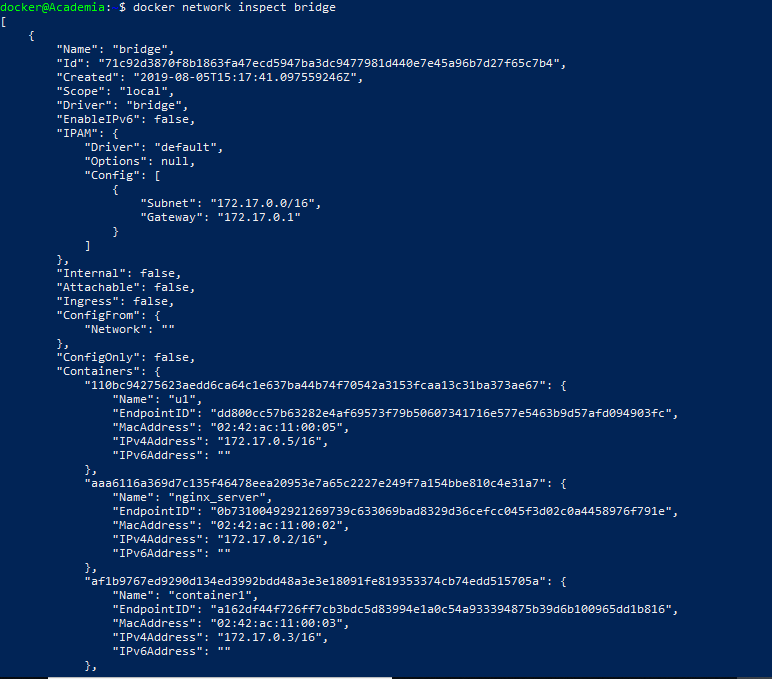
8.- Revisar el contenido del archivo hosts que son los hosts e IP address que el contenedor reconoce.



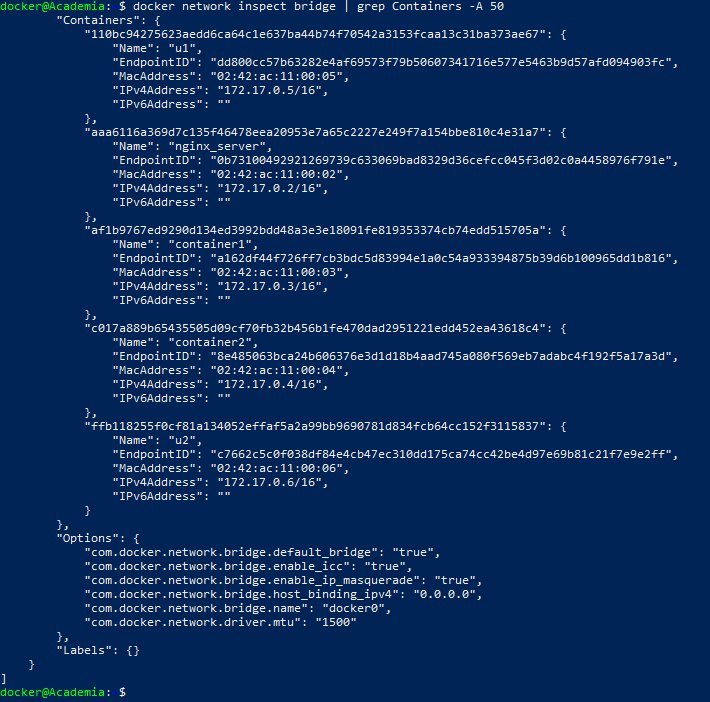
9.- Correr dos contenedores nuevos de Centos 7



10.- Inspeccionar la network bridge



Contenedores

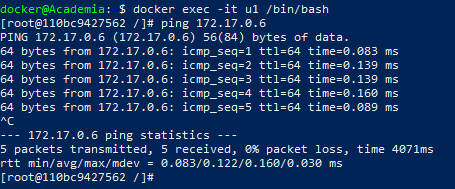


11.- Tratar de hacer ping a otro contenedor desde el u1

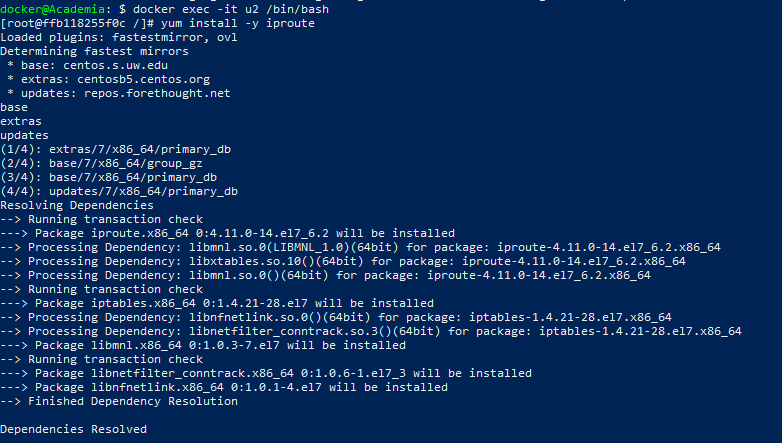
Intentar primero por el nombre y se ve que no se puede realizar la conexión



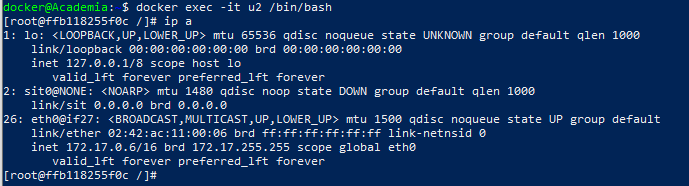
Pero por dirección IP esto si esposible.



12.- Instalar en el contenedor u2 iproute

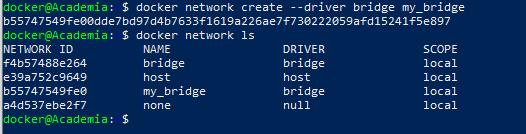


13.- Ejecutar el comando *ip a*

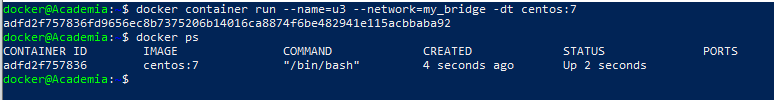


Crear nuestra propia network

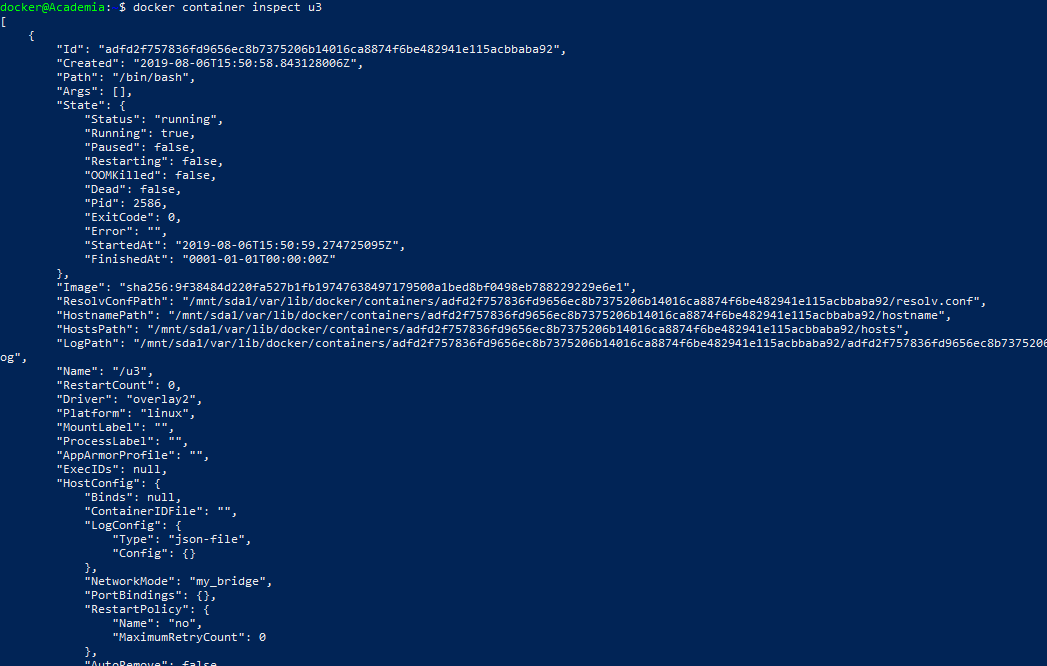
1.- Crear la nueva network



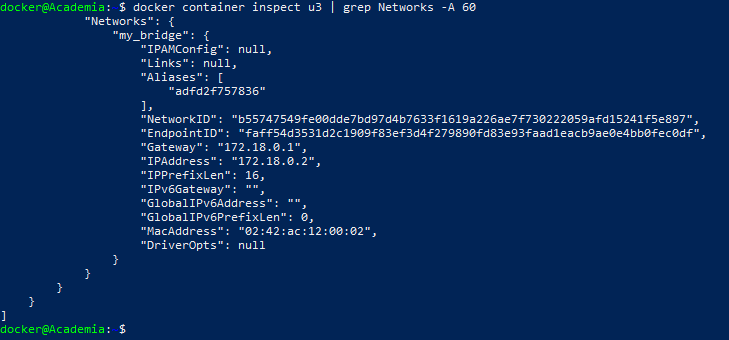
2.- Crear un contenedor y conectarlo a la network que se acaba de crear llamada *my\_bridge*



3.- Inspeccionar el nuevo contenedor creado llamado *u3*



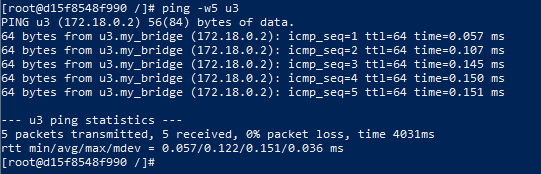
Utilizando grep podemos extraer solamente la información de Networks



4.- Crear un contenedor llamado u4 y entrar de forma interactiva con bash



5.- Hacer ping al contenedor u3 por su nombre



Cuando se crea una nueva network es posible realizar ping con el nombre del contenedor. Esto brinda la ventaja de aparte de ser más fácil de recordar nos evitamos el problema de realizar ping por IP y revisarla a cada momento ya que esta se forma dinámicamente y es muy cambiante.

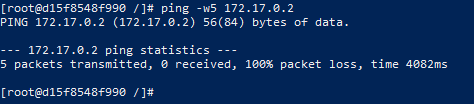
6.- Realizar un ping a otro contenedor ya sea U1 o U2.

Esto no es posible ya que estos dos contenedores forman parte de la red bridge y no la creada *my\_bridge.*

Información del contenedor u1:



Intentando hacer ping desde el contenedor u4

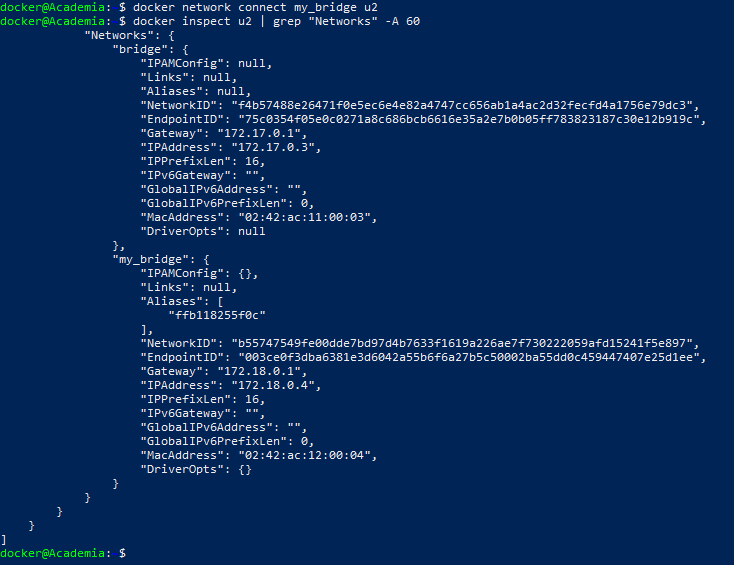


6.- Conectar el contenedor u2 a la nueva network

Inspect del contenedor antes:

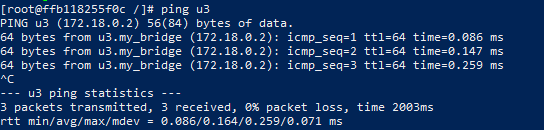


Conectandolo a la nueva network llamada *my\_bridge*

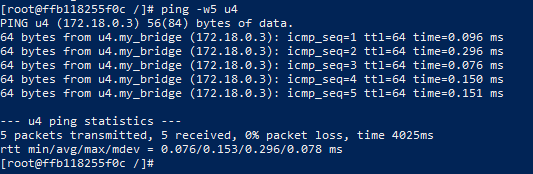


7.- Realizar ping desde el contenedor U2 al contenedor U3 o U4

1. Haciendo ping al contenedor u3 desde U2



1. Haciendo ping al contenedor U4 desde U2

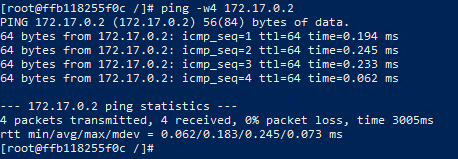


1. Haciendo ping al contenedor U1 desde U2 por Nombre



Esto pasa porque a pesar de que u2 está conectado a *my\_bridge* la conexión entre u1 y u2 es por medio de la red default que es *bridge* lo que no nos permite realizar el ping por nombre.

1. Haciendo ping al contenedor U1 desde U2 por IP



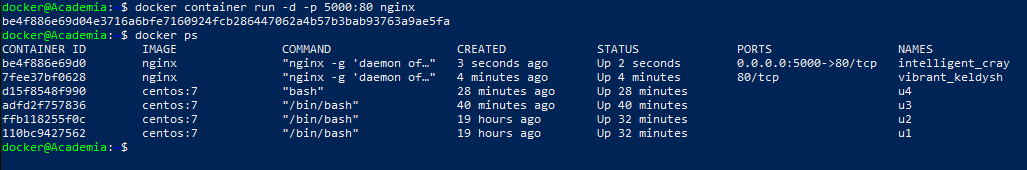
8.- Crear un contenedor de nginx



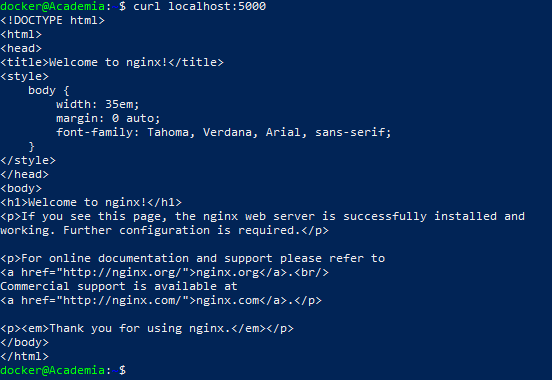
9.- Hacer un mapping al puerto 80 con el comando curl



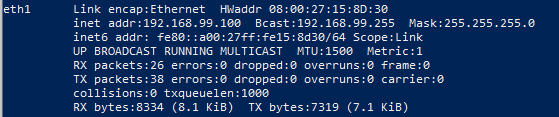
10.- Correr un nuevo contenedor nginx haciendo un maping del pueto 5000 que es del host y el puerto 80 que será del contenedor.

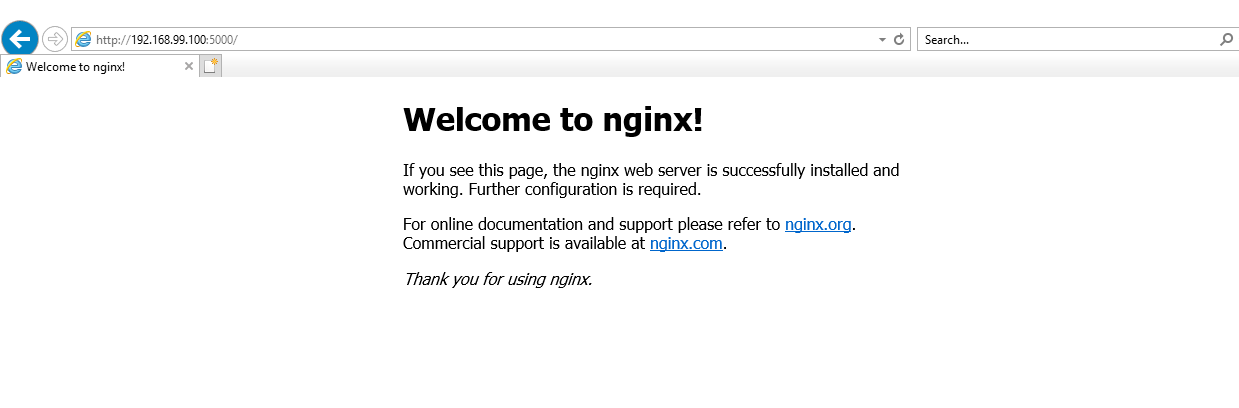


1. Verificar si puedo conectarme con curl y el nuevo puerto 5000

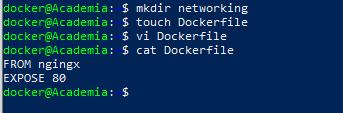


1. Revisar en el navegador si se puede realizar la conexión a ese puerto

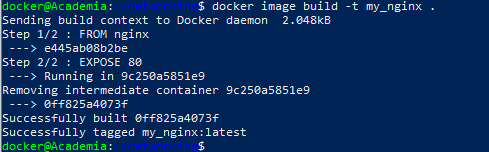




11.- Crear un Dockerfile para crear una nueva imagen y exponer el puerto 80 de nginx



12.- Crear una imagen con ese Dockerfile



13.- Levantar un contenedor con esa nueva imagen

Al especificar el parámetro –P indica que se asignara el puerto que esté disponible para hacer el mapeo al puerto 80. En este caso se le asignó al contenedor al puerto 32768.

