Imagen con Apache y PHP

Tabla de versiones

version	Autor	Comentarios
0.0	Sergi García Barea	Versión original de la práctica de Sergi García Barea
Curso 24/25	José Muñoz Jimeno	Apuntes de Github
Curso 24/25	Laura Agut	Unificar prácticas de ambos autores en un único documento

1. Introducción

En este caso práctico vamos a crear una imagen partiendo de la imagen base “alpine”, una de las imágenes más reconocidas como “imágen ligera”. A esta imagen le instalaremos Apache 2, PHP y pondremos un simple <?php phpinfo(); ?> obtenido desde un fichero de Internet.

Ese fichero lo hemos obtenido de https://gist.github.com/SyntaxC4/5648247

2. Preparando el Dockerfile y creando la imagen

Crearemos el siguiente “Dockerfile”:

```
admin@ip-172-31-91-112:~$ mkdir Practica2_3alpine
admin@ip-172-31-91-112:~$ cd Practica2_3alpine/
admin@ip-172-31-91-112:~/Practica2_3alpine$ nano Dockerfile
```

```
FROM alpine
```

```
MAINTAINER Laura <l.agutmanrique@edu.gva.es>

#Actualizamos e instalamos paquetes con APK para Alpine

RUN apk update && apk add apache2 php php-apache2 openrc tar

#Copiamos script para lanzar Apache 2

ADD ./start.sh /start.sh

#Descargamos un ejemplo de <?php phpinfo(); ?> por enseñar como bajar algo de
Internet

#Podria haber sido simplemente

#RUN echo "<?php phpinfo(); ?>" > /var/www/localhost/htdocs/index.php

ADD

https://gist.githubusercontent.com/SyntaxC4/5648247/raw/
94277156638f9c309f2e36e19bff378ba7364907/info.php
/var/www/localhost/htdocs/index.php

# Si quisiéramos algo como Wordpress haríamos

#ADD http://wordpress.org/latest.tar.gz

/var/www/localhost/htdocs/wordpress.tar.gz

#RUN tar xvzf /var/www/localhost/htdocs/wordpress.tar.gz && rm -rf

/var/www/localhost/htdocs/wordpress.tar.gz

# Usamos usuario y grupo www-data. El grupo lo crea Apache, pero si quisiéramos
crear grupo

# Grupo www-data RUN set -x && addgroup -g 82 -S www-data

# Creamos usuario www-data y lo añadimos a ese grupo

RUN adduser -u 82 -D -S -G www-data www-data

# Hacemos todos los ficheros de /var/www propiedad de www-data

# Y damos permisos s esos ficheros y a start.sh

RUN chown -R www-data:www-data /var/www/ && chmod -R 775 /var/www/ && chmod
755 /start.sh

#Indicamos puerto a exponer (para otros contenedores) 80

EXPOSE 80

#Comando lanzado por defecto al instalar el contenedor

CMD /start.sh
```

El funcionamiento del propio “Dockerfile” está definido por sus propios comentarios.

Antes de crear la imagen, vemos que el Dockerfile contiene la línea ***ADD .start.sh /start.sh***, por lo que hay que crear en el mismo directorio donde está el Dockerfile un archivo **start.sh** con el siguiente contenido:

```
#/bin/sh
#Lanzamos servicio Apache2 en segundo plano
/usr/sbin/httpd -D FOREGROUND
```

```
admin@ip-172-31-91-112:~/Practica2_3alpine$ nano start.sh
admin@ip-172-31-91-112:~/Practica2_3alpine$
```

```
  GNU nano 7.2
#/bin/sh
#Lanzamos servicio Apache2 en segundo plano
/usr/sbin/httpd -D FOREGROUND
```

Una vez preparado, crearemos la imagen con la siguiente línea, en la que indicamos que creamos la imagen “alpineapache” basándonos en el fichero “Dockerfile” del directorio actual.

```
docker build -t alpineapache ./
```

```
admin@ip-172-31-91-112:~/Practica2_3alpine$ docker build -t alpineapache ./
Sending build context to Docker daemon  4.096kB
Step 1/9 : FROM alpine
latest: Pulling from library/alpine
43c4264eed91: Pull complete
Digest: sha256:beefdbd8a1da6d2915566fde36db9db0b524eb737fc57cd1367effd16dc0d06d
Status: Downloaded newer image for alpine:latest
 ---> 91ef0af61f39
Step 2/9 : MAINTAINER Laura <l.agutmanrique@edu.gva.es>
 ---> Running in 59616fd88ade
Removing intermediate container 59616fd88ade
 ---> 807fabde6fb7
Step 3/9 : RUN apk update && apk add apache2 php php-apache2 openrc tar
 ---> Running in ab84c67e26bb
fetch https://dl-cdn.alpinelinux.org/alpine/v3.20/main/x86_64/APKINDEX.tar.gz
fetch https://dl-cdn.alpinelinux.org/alpine/v3.20/community/x86_64/APKINDEX.tar.gz
v3.20.3-227-gb0ce744230b [https://dl-cdn.alpinelinux.org/alpine/v3.20/main]
v3.20.3-229-g589343ed480 [https://dl-cdn.alpinelinux.org/alpine/v3.20/community]
OK: 24165 distinct packages available
(1/21) Installing libuuid (2.40.1-r1)
(2/21) Installing apr (1.7.5-r0)
(3/21) Installing libexpat (2.6.3-r0)
(4/21) Installing apr-util (1.6.3-r1)
(5/21) Installing pcre2 (10.43-r0)
```

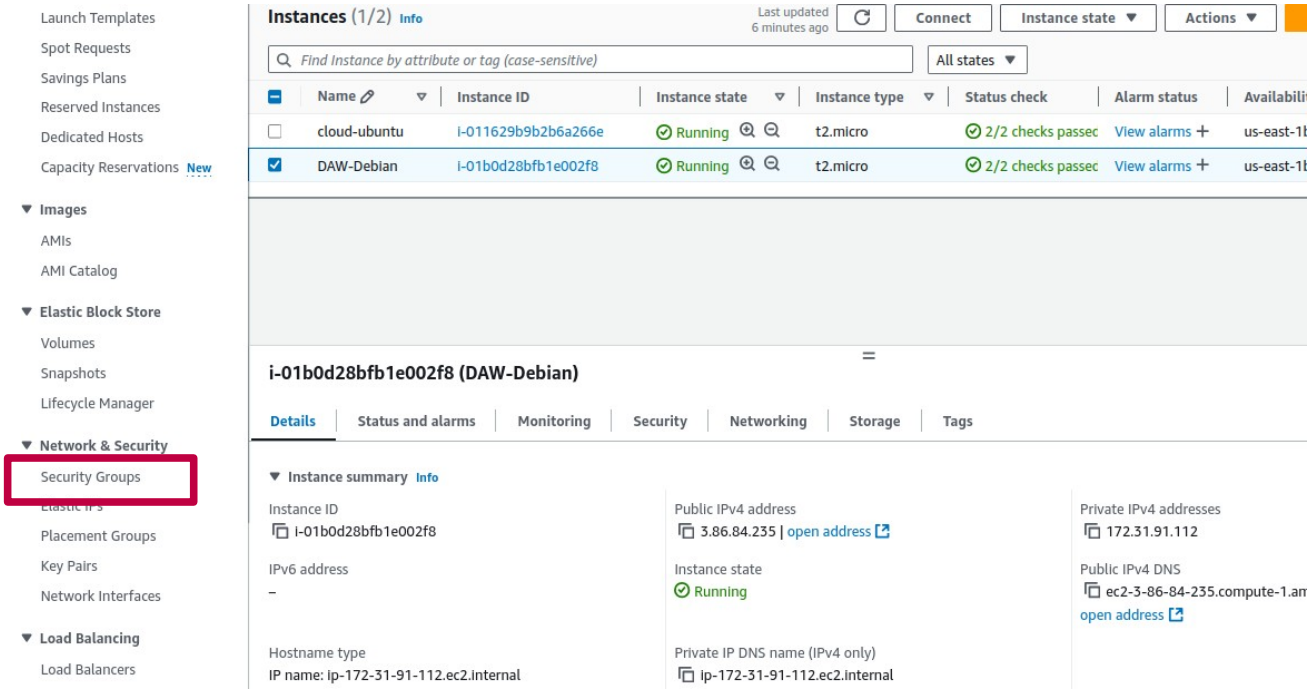
Comprobamos que se ha creado la imagen alpineapache:

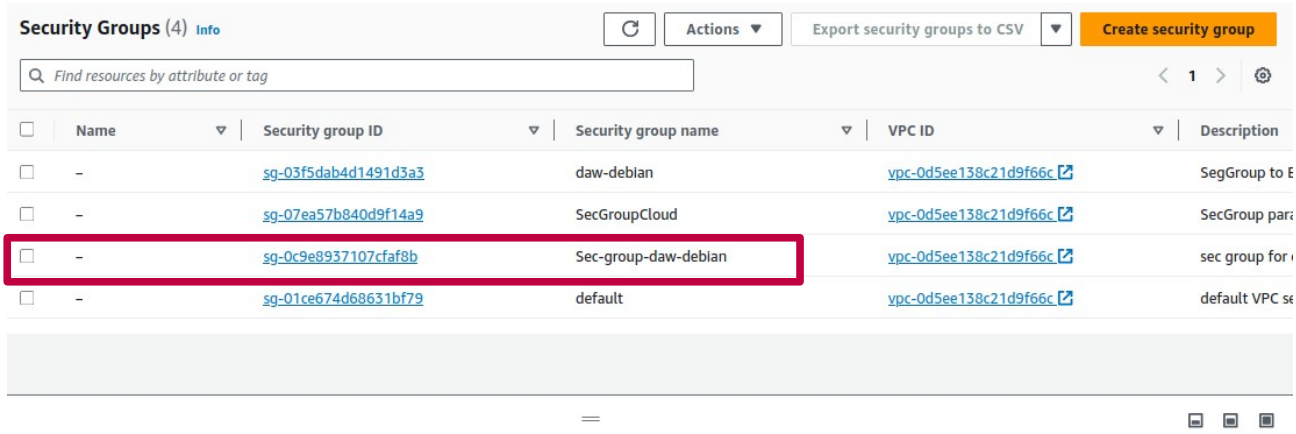
```
admin@ip-172-31-91-112:~/Practica2_3alpine$ docker images
REPOSITORY              TAG       IMAGE ID       CREATED          SIZE
alpineapache            latest    1cbdc343ae46   28 seconds ago   31.1MB
laura/ubuntu_pruebas    25        37f99ac4cab5   2 hours ago      125MB
ubuntunano              latest    64dcc985ae37   4 hours ago      79.3MB
laura/ubuntu_pruebas    24        28c74947969a   23 hours ago     125MB
laura/ubuntu_pruebas    latest    28c74947969a   23 hours ago     125MB
ubuntu                  latest    59ab366372d5   2 weeks ago      78.1MB
busybox                 latest    27a71e19c956   4 weeks ago      4.27MB
alpine                  latest    91ef0af61f39   7 weeks ago      7.8MB
alpine                  3.10      e7b300aee9f9   3 years ago      5.58MB
```

3. Probando la imagen

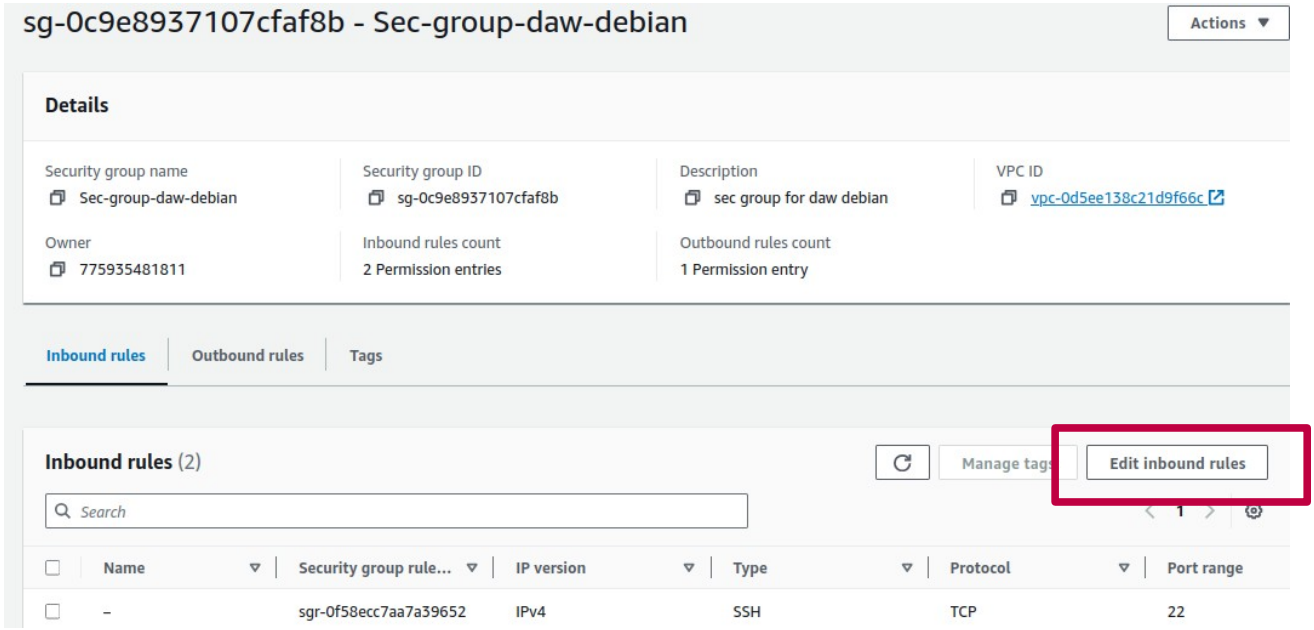
Aquí tenemos varias opciones.

Si estás trabajando sobre una máquina virtual en AWS Academy, deberás abrir el puerto 8080. Para ello hay que ir a «Security groups» y seleccionar el grupo de seguridad que creamos al crear nuestra instancia EC2.

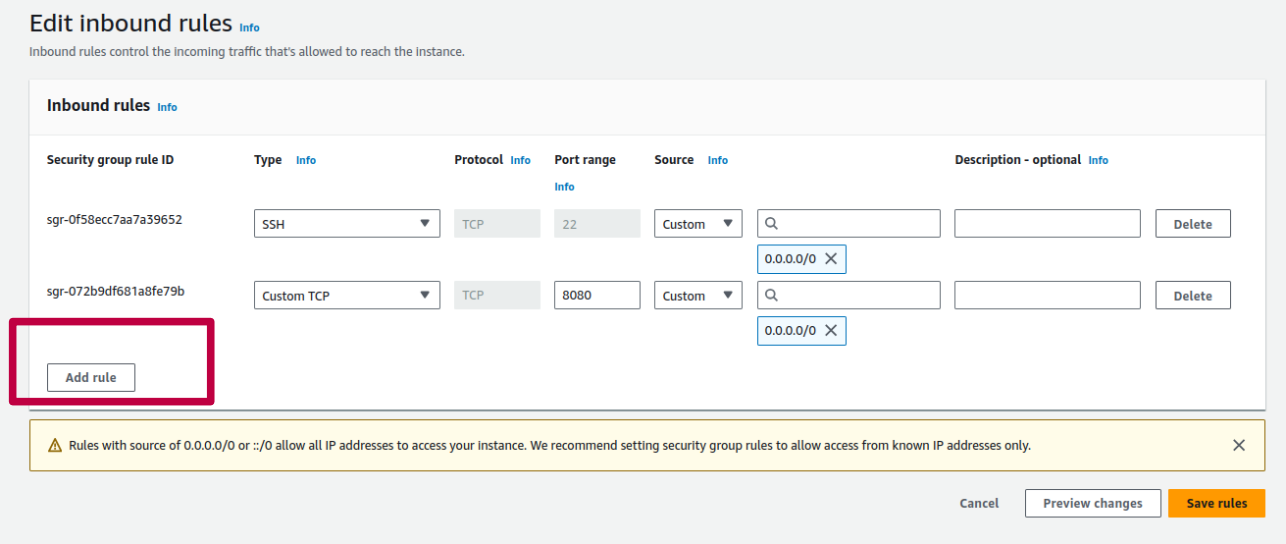




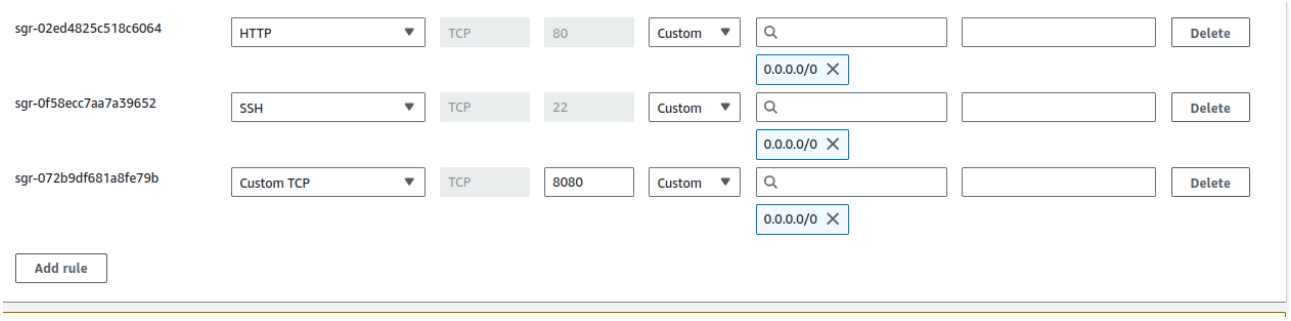
Una vez en el grupo de seguridad, tenemos que editar la reglas de entrada:



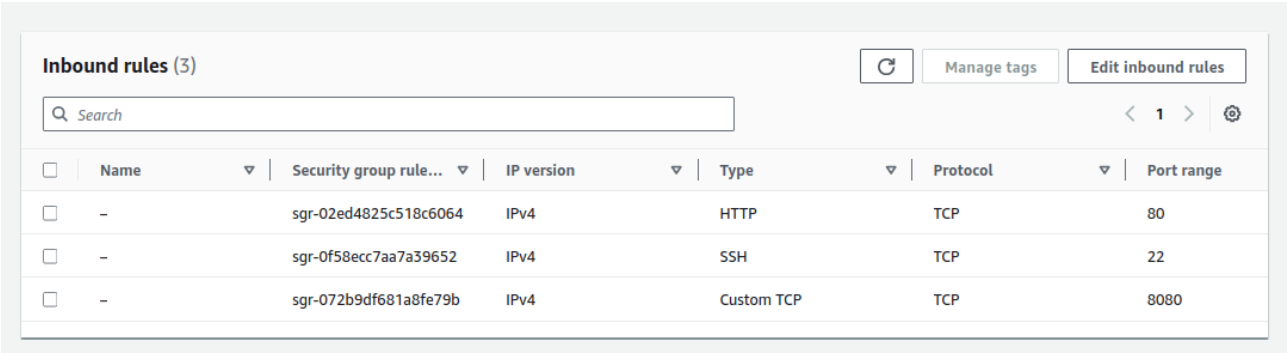
Pulsamos en «Añadir regla» y vamos a abrir el puerto TCP 80 para cualquier IP (0.0.0.0/0).



Lo lógico seria lanzar la aplicación sobre el puerto 80 de nuestra máquina. Pero si ese puerto ya estuviera usado por otra aplicación te podría dar problemas. Por eso hemos abierto el puerto 8080. Aunque aquí abajo tienes ambas reglas, la del 80 y la del 8080, por si quieres probar.



Así pues las reglas quedarían de la siguiente manera:



Con el siguiente comando, podremos lanzar la aplicación en el puerto 80 de nuestra máquina:

```
docker run -dp 80:80 alpineapache
```

Para lanzar la aplicación en el puerto 8080 usamos el siguiente comando:

```
docker run -d -p 8080:80 alpineapache
```

Una vez hecho esto, podremos probar la aplicación accediendo a **http://localhost:8080/index.html** y **http://localhost:8080/index.php**.

Si estás en AWS deberás sustituir localhost por la IP pública de esa máquina virtual.

```
admin@ip-172-31-91-112:~$ docker run -d -p 8080:80 alpineapache
c5dae44331fe74b76d9dbee428b73bc6dd8893c313e3d767407d194ca1fc42a4
admin@ip-172-31-91-112:~$
```

```
admin@ip-172-31-91-112:~$ docker run -dp 80:80 alpineapache
66c7d4efe2ffd532cd26e0219821e993687d31c59573ca4ad5dc555901f06f5a
admin@ip-172-31-91-112:~$
```