

Seguridad en Docker

Beatriz González Hernández

Máster interuniversitario en Seguridad de las Tecnologías de la Información y
Comunicaciones

Ad Hoc

Pau del Canto Rodrigo

Victor Garcia Font

06/2018



Esta obra está sujeta a una licencia de Reconocimiento-NoComercial-SinObraDerivada [3.0 España de Creative Commons](https://creativecommons.org/licenses/by-nc-nd/3.0/es/)

FICHA DEL TRABAJO FINAL

Título del trabajo:	<i>Seguridad en Docker</i>
Nombre del autor:	<i>Beatriz González Hernández</i>
Nombre del consultor/a:	<i>Pau del Canto Rodrigo</i>
Nombre del PRA:	<i>Victor Garcia Font</i>
Fecha de entrega (mm/aaaa):	<i>06/2018</i>
Titulación:	<i>Máster interuniversitario en Seguridad las de las Tecnologías de la Información y Comunicaciones</i>
Área del Trabajo Final:	<i>Ad Hoc</i>
Idioma del trabajo:	<i>Español</i>
Palabras clave	<i>Docker, seguridad, aplicación</i>
<p>Resumen del Trabajo (máximo 250 palabras): <i>Con la finalidad, contexto de aplicación, metodología, resultados i conclusiones del trabajo.</i></p>	
<p>El proyecto consiste en el desarrollo de una guía de seguridad para aplicaciones web simples desplegadas en Docker así como de un documento de políticas de empresa que permita garantizar el compromiso de la misma con la seguridad de las aplicaciones.</p> <p>Debido al reciente crecimiento de las aplicaciones desplegadas en contenedores Docker se ha considerado importante realizar una revisión de la seguridad que se puede aportar ya que es un apartado que ha sido olvidado o postergado por muchos hasta tal punto que la mayoría de documentación disponible en la red hablan de Docker sin ni siquiera mencionarla.</p> <p>Se ha realizado un análisis de las fases y medidas de seguridad necesarias para el despliegue de la aplicación así como de las herramientas de seguridad disponibles actualmente. En cada fase se encuentran explicados los pasos a realizar y se han probado las medidas necesarias para aportar seguridad a la misma.</p> <p>Se han descartado algunas herramientas comerciales ya que sus funcionalidades exceden los objetivos para el despliegue de la aplicación. También se han descartado herramientas debido a que la complejidad de instalación no encajaban con la intención del proyecto de obtener una guía simple y efectiva.</p> <p>Finalmente, se ha creado una guía para aplicaciones en Docker que unifica las medidas, herramientas y consejos de seguridad que se han considerado adecuados.</p> <p>Además se ha constituido un documento de políticas de empresas con los</p>	

puntos a tener en cuenta y revisar en la misma para un correcto funcionamiento de las medidas de seguridad implantadas.

Abstract (in English, 250 words or less):

The project consists in the development of a security guide for simple web applications deployed with Docker as well as a document of company policies that allows us to guarantee the company's obligation with the application's security.

Due to the recent growth of the applications deployed with Docker's containers it has been considered important to review the security that can be provided because it is a section that has been forgotten or postponed by many sites. Most of the documentation available on the network is about Docker without even mentioning security issues.

The project has carried out an analysis of the phases and security measures necessary for the application's deployment as well as an analysis of the security tools currently available. The steps to perform the deployment are explained in each phase and many necessary measures to provide security to it has been tested.

Some commercial tools has been discarded because their features exceed the objectives for the application's deployment. Some tools have also been discarded because the installation complexity did not fit with the project's goal to obtain a simple and effective guide.

Finally, a guide for applications with Docker's container has been created that unifies the measures, tools and safety tips that have been considered appropriate.

In addition, a company policies' document has been established with the points to be taken into account and reviewed in order to ensure the proper functioning of the security measures implemented.

Índice

1. Introducción	1
1.1 Contexto y justificación del Trabajo	1
1.2 Objetivos del Trabajo	3
1.3 Enfoque y método seguido.....	3
1.4 Planificación del Trabajo	3
1.5 Breve resumen de productos obtenidos	6
2. Introducción Docker	7
3. Preparación del equipo	11
3.1 Entorno de trabajo.....	11
3.2 Seguridad en el Kernel.....	12
4. Instalación y configuración de Docker	22
5. Imágenes Docker	25
6. Ejecución contenedor.....	35
7. Monitorización de contenedores.....	37
8. Conclusiones.	44
9. Glosario.	45
10. Bibliografía	46
Anexo I.....	47
Anexo II.....	51
Anexo III.....	57

Lista de figuras

Ilustración 1 Esquema Contenedor Docker	2
Ilustración 2 Planificación temporal	1
Ilustración 3 Gráfica planificación temporal	1
Ilustración 4 Esquema contenedor	9
Ilustración 5 Esquema VM	9
Ilustración 6 Docker Hub	9
Ilustración 7 Docker Store	10
Ilustración 8 Estadística de sistemas operativos	11
Ilustración 9 Creación repositorio Quay	27
Ilustración 10 Quay Repositories	27
Ilustración 11 Cabecera análisis Quay	27
Ilustración 12 Detalle análisis Quay	28
Ilustración 13 Descripción vulnerabilidad Quay	28
Ilustración 14 Búsqueda Anchore	28
Ilustración 15 Repositorio jboss/wildfly Anchore	29
Ilustración 16 Cabecera análisis Anchore	29
Ilustración 17 Cabecera análisis Anchore	29
Ilustración 18 Aplicación	30
Ilustración 19 Fichero Dockerfile	30
Ilustración 20 Dockerfile actualizado	31
Ilustración 21 Repositorios privados Anchore	33
Ilustración 22 Repositorios y credenciales Anchore	33
Ilustración 23 Cabecera my-app Anchore	33
Ilustración 24 Detalle my-app Anchore	33
Ilustración 25 Eliminación del repositorio Quay	34
Ilustración 26 Cabecera my-app Quay	34
Ilustración 27 Detalle my-app Quay	34
Ilustración 28 Twistlock	42
Ilustración 29 Aqua	43
Ilustración 30 StackRox	43

1. Introducción

1.1 Contexto y justificación del Trabajo

El desarrollo de software ha ido evolucionando con los años hacia los desarrollos ágiles^[1] ya que fomentan la participación tanto del cliente como de todo el equipo de trabajo obteniendo múltiples ventajas.

Se caracterizan por el desarrollo iterativo e incremental de la aplicación produciéndose entregas de forma rápida y continuada. Esto contribuye a la rápida detección de errores o malentendidos, de manera que se pueden solventar en una fase temprana y conseguir una mayor eficiencia y velocidad de desarrollo. Todo esto se traduce en una mayor satisfacción del cliente y un ahorro de costes para la empresa.

Con la misma filosofía han surgido los modelos de integración continua y entrega continua. La integración continua^[2] consiste en la automatización de la fase de integración: descarga del código desde la herramienta de control de versiones como puede ser GIT, Mercurial o Subversión entre otras, la compilación del programa, la ejecución de pruebas y la generación de informes. Con ello se agiliza más la detección de errores en las primeras fases del desarrollo pero no involucra la participación del cliente por lo que es importante el concepto de entrega continua.

La entrega continua^[3] consiste en desarrollar el software de manera modular en ciclos cortos asegurando que se puede hacer una entrega estable y confiable en cualquier momento. Si además se automatiza esta entrega y se realiza de manera constante conseguimos un despliegue continuo. De esta forma el cliente puede ver y validar la aplicación fácilmente y con frecuencia.

Para realizar el despliegue continuo con éxito es importante que las máquinas con los servidores de desarrollo y producción tengan las mismas librerías, las mismas variables, mismas herramientas, etc. Si no estuvieran en sincronía se perdería mucho tiempo solventando errores que no se detectan en los servidores de desarrollo. Para solucionar dicho problema son útiles herramientas como Docker.

Docker^{[4][5]} es una plataforma de código abierto para el desarrollo, montaje y ejecución de aplicaciones. Permite empaquetar las aplicaciones y ejecutarlas en un entorno aislado denominado contenedor. Dichos contenedores además contendrán todos los elementos necesarios para que la aplicación pueda ejecutarse sin problemas.

Los contenedores son más ligeros que las máquinas virtuales ya que no necesitan de la carga extra de un hypervisor sino que se ejecutan directamente en el kernel de la máquina host. Con ello se consigue una mayor capacidad de recursos para las aplicaciones y permite un mayor número de contenedores que de máquinas virtuales.

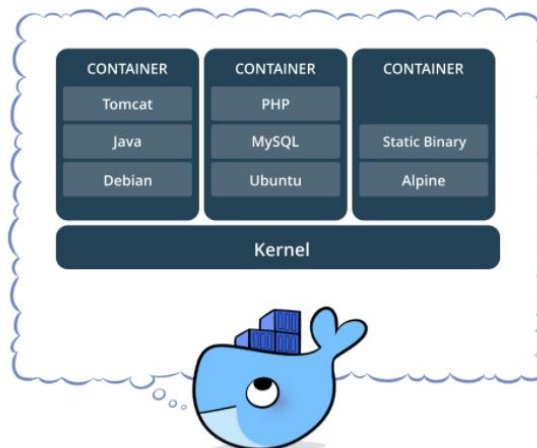


Ilustración 1 Esquema Contenedor Docker

Por otra parte, Docker proporciona la capacidad de trabajar la infraestructura de la misma manera que se manejan las aplicaciones, añadiendo las instrucciones de construcción de la imagen en un fichero «Dockerfile» y manteniendo un control de versiones de las imágenes. Además, su capacidad de escalabilidad y portabilidad permite que se pueda aumentar o disminuir los recursos y ejecutar en diferentes plataformas con una gran facilidad según lo necesite el negocio.

Aunque los contenedores no son un concepto nuevo Docker ha realizado una implementación que lo simplifica de manera que resulta muy fácil de implantar. Sin embargo esa simplicidad y la falta de información en la mayoría de los tutoriales hace que se corra el riesgo de que se instale sin tener en cuenta los aspectos de seguridad que le conciernen. Por ello, que es de vital importancia una mayor labor divulgativa y que existan guías que incluyan las buenas prácticas a tener en cuenta en dicha materia.

La seguridad en Docker abarca muchos frentes. Desde la correcta configuración del host y de Docker hasta la seguridad en las operaciones de los contenedores y la monitorización de los mismos. Además al trabajar con imágenes es importante tener en cuenta que estas pueden contener vulnerabilidades.

Actualmente se pueden encontrar libros y abstracts específicos sobre seguridad en Docker o libros sobre Docker en general con algún apartado de seguridad. Vía Web existen numerosos artículos y vídeos sobre instalación de Docker, como ejecutar contenedores con aplicaciones, puntos a tener en cuenta sobre seguridad en Docker, herramientas de auditoría, etc. Sin embargo, dicha información se encuentra por separado por lo que no sería difícil saltarse algún aspecto importante dejando el sistema vulnerable.

Este trabajo final del Máster interuniversitario en Seguridad de las Tecnologías de la Información y Comunicaciones consiste en la creación de una guía de implantación de Docker para una aplicación Java que se ejecuta en Wildfly teniendo en cuentas los puntos de seguridad más relevantes, desde la seguridad en el Kernel hasta la ejecución de los contenedores, y como realizar una monitorización una vez implantado. Además, se procederá a la creación de un documento que sirva para completar la política y normas de

empresa aportando los puntos con referencia a Docker de manera que establezca políticas de uso y actualización para minimizar los riesgos.

1.2 Objetivos del Trabajo

El trabajo tiene como objetivo principal la creación de una guía de despliegue de aplicaciones Java en servidores WildFly con Docker de manera segura y un documento de políticas y normas de empresa a definir con la implantación de Docker en la misma.

Además, tiene como objetivos generales la adquisición de conocimiento en las siguientes áreas:

- Contenedores Docker.
- Seguridad del Kernel de GNU/Linux.
- Seguridad en Docker.
- Monitorización en Docker.

1.3 Enfoque y método seguido

El trabajo consistirá de una parte de investigación y análisis que se irá entremezclando con el despliegue de una aplicación Java en un servidor WildFly mediante un contenedor Docker siempre teniendo en especial consideración la seguridad del mismo.

Se ha optado por un enfoque teórico y práctico, realizando una investigación de los diferentes apartados del proyecto a la vez que se ponen en práctica y se documentan para obtener una guía más visual y ordenada.

1.4 Planificación del Trabajo

Los recursos necesarios serán un ordenador de mesa o portátil con una distribución GNU/Linux instalada.

Las tareas a realizar para la finalización del proyecto son:

- Seguridad Kernel: Análisis de los distintos modos de añadir seguridad al Kernel e instalación de los que se considere oportunos. Se obtendrá una lista de condiciones a aplicar al contenedor durante el paso de ejecución si fuera necesario.
- Instalación y configuración Docker: Instalación de Docker y configuración del mismo realizando una investigación de seguridad y aplicando las modificaciones necesarias.
- Análisis de imágenes: Búsqueda de la imagen base a utilizar y análisis de las diferentes herramientas de detección de vulnerabilidades aplicándolas a dicha imagen.
- Creación de imagen - Buenas prácticas Dockerfile: Creación del contenedor para la aplicación teniendo en cuenta las buenas prácticas para la creación de ficheros Dockerfile.
- Ejecución contenedor de la aplicación: Investigación sobre la seguridad en la ejecución de contenedores Docker y ejecución del Docker creado en la tarea anterior .

- Monitorización del contenedor: Investigación de las diferentes herramientas de monitorización de contenedores y aplicación de las más relevantes.
- Políticas y normas de empresa: Creación de un documento con las políticas y normas de empresas a tener en cuenta al añadir despliegues con contenedores Docker.
- Finalización de memoria: Añadir conclusiones y finalizar memoria del proyecto.
- Presentación/Vídeo: Creación de una presentación y vídeo resumen del proyecto.

En todas las tareas hasta la monitorización del contenedor se procederá a escribir en un documento guía los pasos que se irán realizando para desplegar la aplicación Java con Docker teniendo en cuenta los diferentes aspectos de seguridad.

Nombre	Duración
PEC1 - Plan de trabajo	20 days
PEC2	28 days
Seguridad Kernel	14 days
Instalación y configuración Docker	14 days
PEC3	28 days
Análisis imágenes	14 days
Creación imagen - Buenas prácticas Dockerfile	9 days
Ejecución contenedor aplicación	5 days
PEC4	28 days
Monitorización del contenedor	18 days
Políticas de empresa	5 days
Finalización de memoria	5 days
PEC5	7 days
Presentación/Vídeo	7 days
PEC6	5 days
Defensa TFM	5 days

Ilustración 2 Planificación temporal

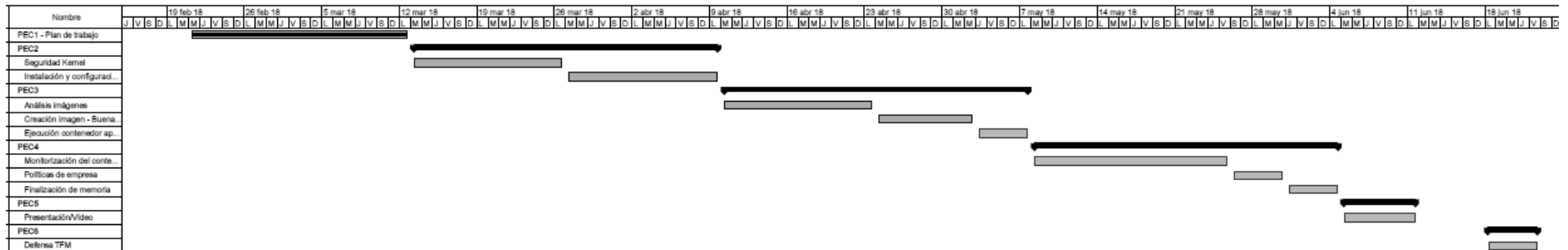


Ilustración 3 Gráfica planificación temporal

1.5 Breve resumen de productos obtenidos

- Guía Docker. Guía de instalación de aplicaciones Java en servidores WildFly con Docker que incluirá:
 - Pautas para añadir seguridad al Kernel.
 - Herramientas para comprobar vulnerabilidades en las imágenes Docker.
 - Buenas prácticas en Dockerfiles.
 - Buenas prácticas en ejecución de contenedores.
 - Herramientas de monitorización y auditoría de contenedores Docker.
- Documento de políticas y normas de empresa. Políticas y normas de empresa a tener en cuenta con la implantación de Docker.

2. Introducción Docker

Docker es una plataforma creada para el desarrollo, despliegue y ejecución de aplicaciones con contenedores por parte de programadores y administradores de sistemas.

Fue liberado en marzo de 2013 y en abril de 2015 ya se había convertido en uno de los proyectos con mayor número de estrellas en GitHub contando con contribuyentes como IBM, Google y Cisco, siendo el mayor aportador Red Hat.

Docker empezó como un proyecto basado en LXC (Linux Container) pero finalmente fue sustituido por su propia biblioteca, libcontainer escrita en el lenguaje de programación Go^[6]. Consigue así funcionalidades con mayor potencia y mejoras:

- **Portabilidad.** Permite la portabilidad de los despliegues entre diferentes máquinas. Docker define un formato de empaquetado de la aplicación y todas sus dependencias dentro de un único objeto llamado contenedor. Los contenedores pueden ser ejecutados con la garantía de que el entorno de ejecución proporcionado para la aplicación es exactamente el mismo en todos los entornos que suelen conformar el ciclo de desarrollo de software (desarrollo, pruebas, producción, etc.). Por su parte, LXC implementa un aislamiento de procesos que aun siendo un requisito importante para la portabilidad de los despliegues no es suficiente para conseguirla. Las aplicaciones que se transfieren con LXC están ligadas a la configuración específica de la máquina original: red, almacenamiento, logs, etc. Docker define una abstracción de dichas características de la máquina. El mismo contenedor Docker puede ejecutarse, sin sufrir ninguna modificación, en diferentes máquinas con diferentes configuraciones.
- **Focalizado en las aplicaciones.** Docker está optimizado para el despliegue de aplicaciones en lugar de centrarse en las máquinas. Esto queda reflejado en su API, su interfaz de usuario, la filosofía de su diseño y su documentación. En contraste, los scripts automatizados de LXC se focalizan en los contenedores como máquinas ligeras, básicamente como servidores que se inician más rápido y que consumen menos RAM.
- **Versionado.** Docker incluye capacidades similares a Git^[7] para el seguimiento continuo de versiones de los contenedores, inspeccionando las diferencias entre las versiones, permitiendo realizar commit de las nuevas versiones, restaurar, etc. El historial también incluye cuando fue ensamblado el contenedor y por quién de manera que se mantiene un traza completa en todo momento. Docker además implementa las funcionalidades de subida y bajada incrementales de las versiones de los contenedores, de manera similar a los git pull, enviando solamente las diferencias.
- **Reutilización.** Cada imagen de un contenedor puede ser utilizado como una especie de imagen padre para crear componentes más

especializados. Se puede realizar manualmente o como parte de su construcción automática.

- **Compartición.** Docker tiene acceso a un registro público en Docker Hub donde miles de personas han subido imágenes de interés general, desde imágenes con PostgreSQL a varias distribuciones GNU/Linux. El registro también incluye una sección oficial con contenedores muy útiles mantenidos por el equipo de Docker. El propio registro es open-source, de forma que cualquiera puede desplegar su propio registro para almacenar y transferir contenedores privados, por ejemplo para servidores internos de desarrollo
- **Herramientas.** Docker define una API para la automatización y personalización de la creación y despliegue de contenedores. Existe un gran número de herramientas integradas con Docker para ampliar su capacidades: herramientas para despliegues del tipo PaaS^[8] (Dokku, Deis, Flynn), paneles de control (docker-ui, Openstack, Horizon, Shipyard), gestores de configuración (Chef, Puppet), integración continua (Jenkins, Strider, Travis), etc. Docker se ha establecido rápidamente como un estándar para las herramientas basadas en contenedores.

Otras ventajas del uso de contenedores son:

- **Flexibilidad:** Incluso las aplicaciones más complejas pueden ejecutarse en un contenedor.
- **Ligeros:** Los contenedores comparten el mismo kernel del host. Esto los hace más ligero que las máquinas virtuales como se verá más adelante
- **Intercambiables:** Se pueden desplegar actualizaciones y aplicarle mejoras sobre la marcha.
- **Escalabilidad.** Permite incrementar y distribuir automáticamente replicas de los contenedores.
- **Apilable:** Se pueden apilar servicios verticalmente y sobre la marcha.

El uso de los contenedores aporta múltiples ventajas con respecto a los sistemas tradicionales y las máquinas virtuales (VM). Los contenedores se ejecutan de forma nativa en Linux y comparten el kernel de la máquina host con otros contenedores. Ejecutan un discreto proceso que no consume más memoria que cualquier otro ejecutable, de manera que es muy ligero. Al contrario las máquinas virtuales ejecutan un sistema operativo completo con acceso virtual a los recursos del host mediante un hypervisor^[9]. Además, en general las máquinas virtuales proporcionan un entorno con más recursos de los que la mayoría de las aplicaciones necesitan.

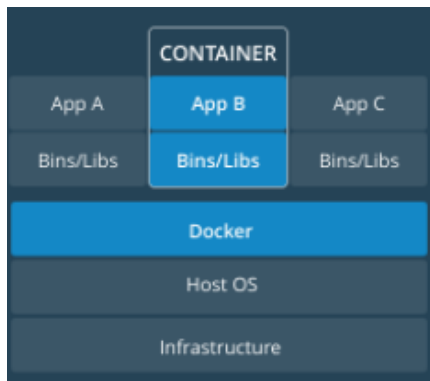


Ilustración 4 Esquema contenedor

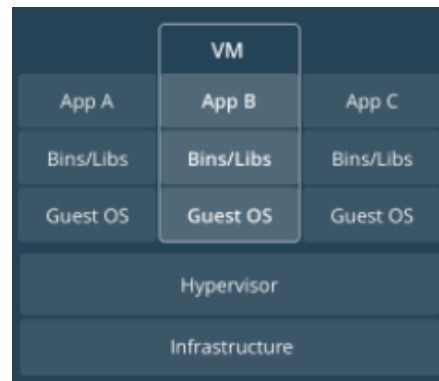


Ilustración 5 Esquema VM

Los contenedores se lanzan ejecutando imágenes. Una imagen es un paquete ejecutable que incluye todo lo necesario para ejecutar una aplicación: el código, librerías, variables de entorno, ficheros de configuración, etc. Por ello los contenedores se pueden definir como una instancia en ejecución de una imagen.

Cómo se ha explicado en un punto anterior, existe un registro público en Docker Hub^[10] que contiene múltiples imágenes, oficiales y no oficiales, que se pueden descargar para su utilización. También es posible subir imágenes propias al mismo registro, pudiendo declararlas públicas o privadas (solo una privada por cuenta gratuita).

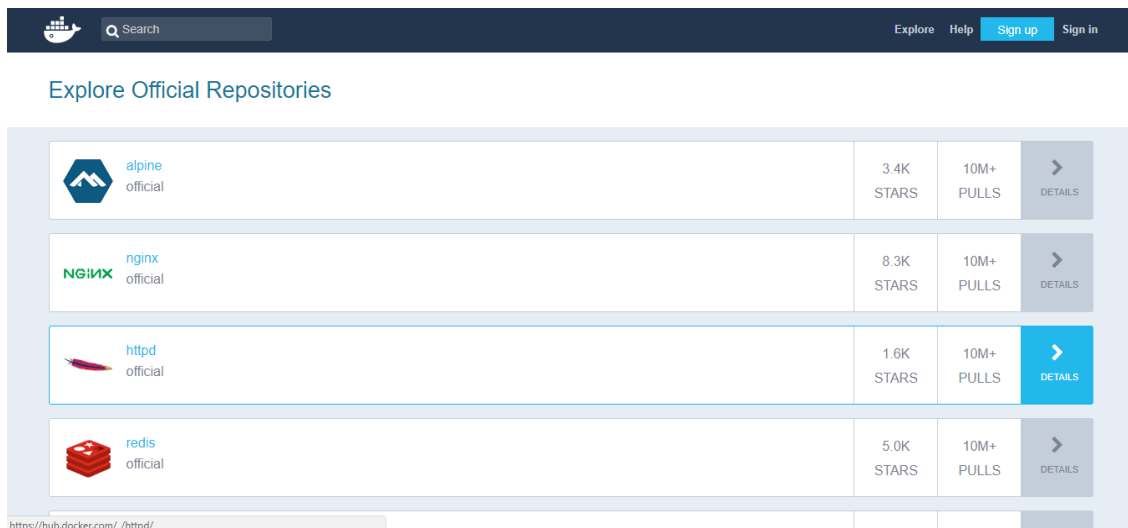


Ilustración 6 Docker Hub

Además Docker también proporciona una tienda, Docker Store^[11], para comercializar los contenedores. Dispone un sistema de búsqueda más sofisticado y permite buscar también en Docker Hub.

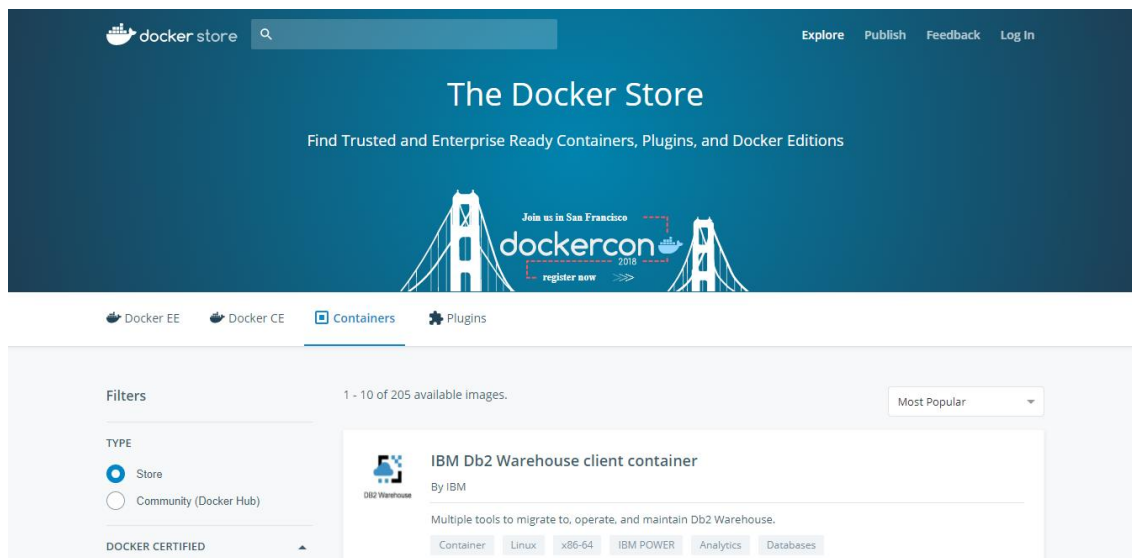


Ilustración 7 Docker Store

Actualmente, para ejecutar Docker se puede utilizar las siguientes plataformas:

- GNU/Linux. En cualquier distribución que ejecute una versión 3.10 o superior del Kernel. Se pueden encontrar diferentes instrucciones específicas para la mayor parte de distribuciones (Ubuntu, Suse, Red Hat, Debian, etc.)
- Windows. Se puede utilizar en Windows Server 2016 y Windows 10. Es necesario activar Hyper-V.

Para las versiones antiguas de Mac y Windows que no cumplen los requisitos mínimos del sistema se puede utilizar la herramienta Docker Toolbox que utiliza el software de virtualización VirtualBox para ejecutar Docker. Se recomienda no utilizar este sistema a no ser que sea estrictamente necesario.

Existen dos tipos de ediciones de Docker:

- Community Edition (CE): Esta edición es apropiada para desarrolladores y equipos pequeños. Es una versión gratuita que incluye el motor, funciones de red y seguridad.
- Enterprise Edition (EE): Está diseñada para el desarrollo en las empresas y equipos IT que necesitan gestionar aplicaciones críticas en producción. Aporta mayores funcionalidades como plugins, contenedores ISV, gestión de imágenes y contenedores, escaneo de seguridad de imágenes, etc.

3. Preparación del equipo

En este apartado se definirá el entorno de trabajo necesario. Se describirá la elección del sistema operativo utilizado así como del equipo en el que se realizará la parte práctica del proyecto. Además, se preparará el equipo con la instalación y configuración de Docker añadiendo seguridad al Kernel.

3.1 Entorno de trabajo

Tras valorar los diferentes sistemas operativos a utilizar se ha considerado como mejor opción una distribución GNU/Linux. En primer lugar, Docker empezó utilizando los LXC (posteriormente sustituido por libcontainer) como base y sigue haciendo uso por defecto de los contenedores Linux. Actualmente se pueden utilizar contenedores Docker de Windows pero solamente con la edición Enterprise. Además, para instalar Docker en Windows se debe activar la virtualización Hyper-V y solo en las ediciones Microsoft Windows 10 Professional o Enterprise 64-bit. Para versiones anteriores es necesario utilizar la herramienta Docker Toolbox que hace uso de VirtualBox.

Por otra parte, observando las estadísticas de uso de los diferentes sistemas operativos en servidores observamos una clara tendencia de servidores Linux.

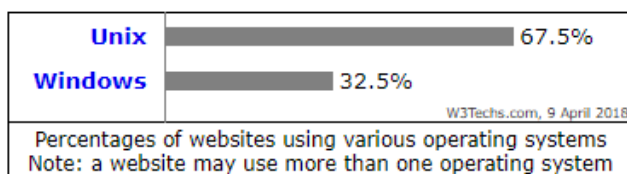


Ilustración 8 Estadística de sistemas operativos

No se han considerado los sistemas operativos Mac ya que no se dispone de ningún equipo para las pruebas y los servidores que lo usan según la estadística son menos de un 0.1%. Además, tanto en el caso de Mac como de Windows se requeriría de un desembolso económico adicional importante para comprar una nueva licencia y montar el nuevo equipo limpio.

Considerando el aspecto académico del proyecto se ha decidido el uso del sistema operativo Debian^[12] 9.4 de 64 bits ya que es un sistema open-source^[13] de gran estabilidad.

Debian es una distribución de GNU/Linux basada en software libre cuyos inicios se remontan al año 1993, sacando su primera versión estable en 1996. Su metodología de desarrollo basada en ramas aporta gran estabilidad al sistema. Los paquetes comienzan en las ramas inestable o experimental y pasan a la rama de pruebas cuando se reducen los fallos significativamente. En la rama de pruebas se detectan los errores y cuando llega a un nivel aceptable de ellos se pasa a la rama de congelación para centrarse en corregirlos y sacar una versión estable. Además, Debian sigue unas directrices de software libre^[14] que garantizan que su distribución principal es completamente open source y dispone de una comunidad muy amplia con lo que se puede encontrar soporte rápido y fiable antes cualquier problema.

Otro aspecto que se ha tenido que valorar es el uso de un sistema físico o en la nube. Se ha valorado positivamente la capacidad de los sistemas en la nube

para reinstalar el sistema operativo rápidamente si fuera necesario y se ha seleccionado la opción más básica en la plataforma Digital Ocean^[15] ya que ofrece un servicio muy económico más que suficiente para el objetivo del proyecto.

3.2 Seguridad en el Kernel

Los contenedores Docker aprovechan la habilidad del kernel de Linux para crear entornos aislados que comparten el kernel con el host anfitrión. Una de las características del Kernel que se aprovecha son las «capabilities». Se dividen las capacidades de root de forma se pueda otorgar a cada proceso las capacidades que realmente requiere. Existen capacidades para casi todas las áreas donde se necesitan privilegios de root. De esta manera se consigue evitar la sobreutilización de usuarios root y utilizar una política de permisos mínimo que aporta una mayor seguridad al sistema. Si un intruso consigue escalar a root dentro del contenedor le será más complicado realizar acciones dañinas o conseguir escalar al host.

En la mayoría de los casos Docker no necesita privilegios root y se podrían reducir las capacidades denegando entre otras:

- Las operaciones «mount»
- El acceso a los socket raw (para prevenir ataques de suplantación)
- El acceso a las operaciones del sistema de ficheros como crear nuevos nodos de dispositivos, modificar el propietario de los ficheros o alterar los atributos.
- La carga de módulos.

Por defecto, Docker utiliza una lista blanca. Deniega todas las capacidades excepto las especificadas y soporta la eliminación e inserciones de nuevas capacidades a la lista. Es importante revisar las que son requeridas realmente por nuestro proceso para establecer la configuración más segura ya que las otorgadas inicialmente a los contenedores crean un aislamiento incompleto. Se pueden ver las capacidades existentes en «<http://man7.org/linux/man-pages/man7/capabilities.7.html>».

Además de las capacidades, existen más funcionalidades de seguridad disponibles para el kernel que son compatibles con las mismas. Algunos sistemas conocidos para añadir una capa extra de seguridad son TOMOYO, AppArmor, SELinux, GRSECURITY, etc.

Se han analizado dos de las herramientas conocidas que se pueden encontrar en las versiones actuales de Debian: SELinux y AppArmor.

3.2.1 SELinux

Security-Enhanced Linux (SELinux)^{[16][17][18]} es un módulo de seguridad para el kernel Linux. Añade al kernel un control de acceso obligatorio fuerte y flexible. De forma general, se definen reglas que definen si un proceso puede acceder a determinados ficheros o recursos de red.

Se puede obtener y activar en Debian mediante el comando «apt-get install selinux-basics selinux-policy-default». El script «selinux-activate» se encarga de automatizar el etiquetado de los ficheros y activar SELinux en el modo

permissive que se detallará más adelante. Una vez reiniciado el equipo SELinux se encontrará en funcionamiento.

A cada usuario y proceso se le asigna un contexto que consiste en la identidad, rol, dominio y nivel MCS («Multi-Category Security», seguridad multicategoría). Se puede observar con la opción Z del comando ps:

```
root@debian-docker-se:~# ps axZ
LABEL                                PID TTY          STAT TIME COMMAND
system_u:system_r:init_t:s0         1 ?            Ss   0:12 /sbin/init
  identidad      rol      dominio nivel MCS
```

Se aplica lo mismo a los ficheros. Se pueden ver utilizando la opción -Z del comando ls:

```
root@debian-docker-se:~# ls -Z prueba.txt
unconfined_u:object_r:user_home_t:s0 prueba.txt
```

Todos los accesos están controlados mediante los dominios y niveles MCS por lo que todos los procesos y ficheros deben estar etiquetados correctamente y se deben crear las reglas necesarias. El comando «semanage» permite modificar los contextos de los ficheros y usuarios..

SELinux proporciona un paquete «selinux-policy-default» que contiene un conjunto de reglas estándar para facilitar su configuración inicial. Además, existen multitud de módulos ya creados que se pueden activar según corresponda.

Los módulos se pueden ver con el comando «semodule -l»

```
root@debian-docker-se:~# semodule -l
accountsd
acct
ada
afs
aiccu
aide
aisexec
alsa
```

Para añadir los módulos se utiliza el comando «semodule -i *modulo*» pero es necesarios habilitarlos con «semodule -e *modulo*». De igual forma se deshabilitan con «semodule -d *modulo*» y se eliminan con «semodule -r *modulo*».

En el paquete «selinux-policy-doc» se pueden encontrar ejemplos de cómo se componen los módulos y que pueden servir como plantillas. En resumen, se dispone de tres tipos de archivos con extensiones «.te», «.fc» y «.if» y un makefile.

Los ficheros «.fc» se corresponden con el fichero que define los tipos de los archivos.

```
myapp executable will have:
# label: system_u:object_r:myapp_exec_t
# MLS sensitivity: s0
# MCS categories: <none>

/usr/sbin/myapp      --      gen_context(system_u:object_r:myapp_exec_t,s0)
```

Los ficheros «.if» permiten definir interfaces que contiene simplemente funciones que generarán el conjunto de reglas al compilar.

```
## <summary>Myapp example policy</summary>
## <desc>
##     <p>
##         More descriptive text about myapp. The desc
##         tag can also use p, ul, and ol
##         html tags for formatting.
##     </p>
##     <p>
##         This policy supports the following myapp features:
##         <ul>
##             <li>Feature A</li>
##             <li>Feature B</li>
##             <li>Feature C</li>
##         </ul>
##     </p>
## </desc>
#

#####
## <summary>
##     Execute a domain transition to run myapp.
## </summary>
## <param name="domain">
##     <summary>
##         Domain allowed to transition.
##     </summary>
## </param>
#
interface('myapp_domtrans', `
    gen_require(`
        type myapp_t, myapp_exec_t;
    `)
    domtrans_pattern($1,myapp_exec_t,myapp_t)
,)
```

Los ficheros «.te» declaran los tipos de fichero y definen las reglas que se aplicarán en el sistema.

```
policy_module(myapp,1.0.0)

#####
#
# Declarations
#

type myapp_t;
type myapp_exec_t;
domain_type(myapp_t)
domain_entry_file(myapp_t, myapp_exec_t)

type myapp_log_t;
logging_log_file(myapp_log_t)

type myapp_tmp_t;
files_tmp_file(myapp_tmp_t)

#####
#
# Myapp local policy
#

allow myapp_t myapp_log_t:file { read_file_perms append_file_perms };

allow myapp_t myapp_tmp_t:file manage_file_perms;
files_tmp_filetrans(myapp_t,myapp_tmp_t,file)
```

Como se puede ver en el fichero las reglas se componen por:

- allow: directiva base para autorizar la operación.

- dominio: el primer parámetro es el dominio del proceso al que se le permite ejecutar la operación.
- tipo y clase: el tercer parámetro está formado por el tipo de SELinux que va a poder ser manipulado por el dominio anterior y la clase de fichero (archivo, directorio, tubería, etc.). Debe ser de la forma «tipo:clase»
- permisos: el último parámetro describe las operaciones permitidas.

SELinux se puede activar en dos modos diferentes: permissive y enforcing. El modo enforcing comprueba las reglas y hace que se cumplan forzosamente (se activa en la sesión con «setenforce 1»). El modo permissive solamente deja registro de las reglas que no se han cumplido (se activa en la sesión con «setenforce 1»). Para modificar el modo permanentemente se debe editar la variable SELINUX (enforcing, permissive o disable) en el fichero /etc/selinux/config. Los accesos que incumplan las reglas se puede ver en Debian con el comando «cat /var/log/messages | grep 'avc'»

```
root@debian-docker-se:~# cat /var/log/messages | grep 'avc'
Mar 24 22:31:35 debian-docker-se kernel: [ 8.003632] audit: type=1400 audit(1521930690.792:3): avc: denied { read } for pid=445 comm="systemd-detect-" name="/enviro
n" dev="proc" ino=7644 scontext=system_u:system_r:systemd_detect_virt_t:s0 tcontext=system_u:system_r:init_t:s0 tclass=file permissive=1
Mar 24 22:31:35 debian-docker-se kernel: [ 8.010404] audit: type=1400 audit(1521930690.804:4): avc: denied { open } for pid=445 comm="systemd-detect-" path="/proc
/1/environ" dev="proc" ino=7644 scontext=system_u:system_r:systemd_detect_virt_t:s0 tcontext=system_u:system_r:init_t:s0 tclass=file permissive=1
Mar 24 22:31:35 debian-docker-se kernel: [ 8.018987] audit: type=1400 audit(1521930690.812:5): avc: denied { getattr } for pid=445 comm="systemd-detect-" path="/p
roc/1/environ" dev="proc" ino=7644 scontext=system_u:system_r:systemd_detect_virt_t:s0 tcontext=system_u:system_r:init_t:s0 tclass=file permissive=1
Mar 24 22:31:35 debian-docker-se kernel: [ 8.049827] audit: type=1400 audit(1521930690.844:6): avc: denied { read } for pid=445 comm="systemd-detect-" name="cmdlin
e" dev="proc" ino=4026531980 scontext=system_u:system_r:systemd_detect_virt_t:s0 tcontext=system_u:object_r:proc_t:s0 tclass=file permissive=1
Mar 24 22:31:35 debian-docker-se kernel: [ 8.061453] audit: type=1400 audit(1521930690.832:7): avc: denied { open } for pid=445 comm="systemd-detect-" path="/proc
/cmdline" dev="proc" ino=4026531980 scontext=system_u:system_r:systemd_detect_virt_t:s0 tcontext=system_u:object_r:proc_t:s0 tclass=file permissive=1
Mar 24 22:31:35 debian-docker-se kernel: [ 8.069433] audit: type=1400 audit(1521930690.860:8): avc: denied { getattr } for pid=445 comm="systemd-detect-" path="/p
roc/cmdline" dev="proc" ino=4026531980 scontext=system_u:system_r:systemd_detect_virt_t:s0 tcontext=system_u:object_r:proc_t:s0 tclass=file permissive=1
Mar 24 22:31:35 debian-docker-se kernel: [ 10.476123] audit: type=1400 audit(1521930693.264:9): avc: denied { read } for pid=516 comm="systemd-detect-" name="enviro
n" dev="proc" ino=7644 scontext=system_u:system_r:systemd_detect_virt_t:s0 tcontext=system_u:system_r:init_t:s0 tclass=file permissive=1
Mar 24 22:31:35 debian-docker-se kernel: [ 10.481466] audit: type=1400 audit(1521930693.272:10): avc: denied { open } for pid=516 comm="systemd-detect-" path="/pro
c/1/environ" dev="proc" ino=7644 scontext=system_u:system_r:systemd_detect_virt_t:s0 tcontext=system_u:system_r:init_t:s0 tclass=file permissive=1
Mar 24 22:31:35 debian-docker-se kernel: [ 10.486720] audit: type=1400 audit(1521930693.280:11): avc: denied { getattr } for pid=516 comm="systemd-detect-" path="/
proc/1/environ" dev="proc" ino=7644 scontext=system_u:system_r:systemd_detect_virt_t:s0 tcontext=system_u:system_r:init_t:s0 tclass=file permissive=1
Mar 24 22:31:35 debian-docker-se kernel: [ 12.820271] audit: type=1400 audit(1521930695.408:26): avc: denied { create } for pid=661 comm="nscd" name="nscd" scontext
=system_u:system_r:nscd_t:s0 tcontext=system_u:object_r:rvar_run_t:s0 tclass=dir permissive=1
Mar 24 22:31:35 debian-docker-se kernel: [ 12.853197] audit: type=1400 audit(1521930695.444:27): avc: denied { setattr } for pid=661 comm="nscd" name="nscd" dev="t
mpfs" ino=12904 scontext=system_u:system_r:nscd_t:s0 tcontext=system_u:object_r:rvar_run_t:s0 tclass=dir permissive=1
Mar 24 22:31:35 debian-docker-se kernel: [ 12.790936] audit: type=1400 audit(1521930695.580:28): avc: denied { write } for pid=661 comm="nscd" path="pipe:[13133]"
 dev="pipefs" ino=13133 scontext=system_u:system_r:nscd_t:s0 tcontext=system_u:system_r:nscd_t:s0 tclass=fifo_file permissive=1
Mar 24 22:31:35 debian-docker-se kernel: [ 12.863875] audit: type=1400 audit(1521930695.652:29): avc: denied { write } for pid=692 comm="ntpd" path="/run/lock/nspd
ate" dev="tmpfs" ino=13288 scontext=system_u:system_r:ntpd_t:s0 tcontext=system_u:object_r:ntpd_lock_t:s0 tclass=file permissive=1
Mar 24 22:31:35 debian-docker-se kernel: [ 12.967562] audit: type=1400 audit(1521930695.756:30): avc: denied { write } for pid=661 comm="nscd" path="pipe:[13438]"
 dev="pipefs" ino=13438 scontext=system_u:system_r:nscd_t:s0 tcontext=system_u:system_r:nscd_t:s0 tclass=fifo_file permissive=1
Mar 24 22:31:36 debian-docker-se kernel: [ 13.824491] audit: type=1400 audit(1521930696.612:31): avc: denied { read } for pid=709 comm="systemd-detect-" name="envi
ron" dev="proc" ino=7644 scontext=system_u:system_r:systemd_detect_virt_t:s0 tcontext=system_u:system_r:init_t:s0 tclass=file permissive=1
Mar 24 22:31:36 debian-docker-se kernel: [ 13.832492] audit: type=1400 audit(1521930696.624:32): avc: denied { open } for pid=709 comm="systemd-detect-" path="/pro
c/1/environ" dev="proc" ino=7644 scontext=system_u:system_r:systemd_detect_virt_t:s0 tcontext=system_u:system_r:init_t:s0 tclass=file permissive=1
Mar 24 22:31:36 debian-docker-se kernel: [ 13.843402] audit: type=1400 audit(1521930696.632:33): avc: denied { getattr } for pid=709 comm="systemd-detect-" path="/p
roc/1/environ" dev="proc" ino=7644 scontext=system_u:system_r:systemd_detect_virt_t:s0 tcontext=system_u:system_r:init_t:s0 tclass=file permissive=1
Mar 24 22:31:36 debian-docker-se kernel: [ 13.852440] audit: type=1400 audit(1521930696.644:34): avc: denied { read } for pid=709 comm="systemd-detect-" name="cmdl
ine" dev="proc" ino=4026531980 scontext=system_u:system_r:systemd_detect_virt_t:s0 tcontext=system_u:object_r:proc_t:s0 tclass=file permissive=1
```

Si se tiene instalado el paquete «auditd» también se pueden ver con el comando «ausearch -m avc»

```
root@debian-docker-se:~# ausearch -m avc
----
time-->Sat Mar 24 22:38:55 2018
type=PROCTITLE msg=audit(1521931135.215:59): proctitle=737368643A20726F6F74205B707269765D
type=SYSCALL msg=audit(1521931135.215:59): arch=c000003e syscall=82 success=yes exit=0 a0=7faca500bcdf a1=7faca500bcad a2=0 a3=8 items=0 ppid=673 pid=1434 uid=0 uid=0
gid=0 euid=0 suid=0 fsuid=0 egid=0 fsgid=0 tty=(none) ses=3 comm="sshd" exe="/usr/sbin/sshd" subj=system_u:system_r:sshd_t:s0-t0:c0:c1023 key=(null)
type=AVC msg=audit(1521931135.215:59): avc: denied { unlink } for pid=1434 comm="sshd" name="mod.dynamic" dev="tmpfs" ino=15402 scontext=system_u:system_r:sshd_t:s0-
s0:c0:c1023 tcontext=system_u:object_r:rvar_run_t:s0 tclass=file permissive=1
type=AVC msg=audit(1521931135.215:59): avc: denied { rename } for pid=1434 comm="sshd" name="mod.dynamic.new" dev="tmpfs" ino=20704 scontext=system_u:system_r:sshd
_t:s0-s0:c0:c1023 tcontext=system_u:object_r:rvar_run_t:s0 tclass=file permissive=1
----
time-->Sat Mar 24 22:38:55 2018
type=PROCTITLE msg=audit(1521931135.215:60): proctitle=737368643A20726F6F74205B707269765D
type=SYSCALL msg=audit(1521931135.215:60): arch=c000003e syscall=2 success=yes exit=5 a0=65366dd2c65 a1=0 a2=0 a3=8 items=0 ppid=673 pid=1434 uid=0 uid=0 gid=0 euid=0
suid=0 fsuid=0 egid=0 fsgid=0 tty=(none) ses=3 comm="sshd" exe="/usr/sbin/sshd" subj=system_u:system_r:sshd_t:s0-t0:c0:c1023 key=(null)
type=AVC msg=audit(1521931135.215:60): avc: denied { open } for pid=1434 comm="sshd" path="/run/mod.dynamic" dev="tmpfs" ino=20704 scontext=system_u:system_r:sshd_t
:s0-s0:c0:c1023 tcontext=system_u:object_r:rvar_run_t:s0 tclass=file permissive=1
type=AVC msg=audit(1521931135.215:60): avc: denied { read } for pid=1434 comm="sshd" name="mod.dynamic" dev="tmpfs" ino=20704 scontext=system_u:system_r:sshd_t:s0-s
0:c0:c1023 tcontext=system_u:object_r:rvar_run_t:s0 tclass=file permissive=1
----
time-->Sat Mar 24 22:38:55 2018
type=PROCTITLE msg=audit(1521931135.215:61): proctitle=737368643A20726F6F74205B707269765D
type=SYSCALL msg=audit(1521931135.215:61): arch=c000003e syscall=5 success=yes exit=0 a0=5 a1=7fede775f0 a2=7fede775f0 a3=8 items=0 ppid=673 pid=1434 uid=0 uid=0 g
id=0 euid=0 suid=0 fsuid=0 egid=0 fsgid=0 tty=(none) ses=3 comm="sshd" exe="/usr/sbin/sshd" subj=system_u:system_r:sshd_t:s0-t0:c0:c1023 key=(null)
type=AVC msg=audit(1521931135.215:61): avc: denied { getattr } for pid=1434 comm="sshd" path="/run/mod.dynamic" dev="tmpfs" ino=20704 scontext=system_u:system_r:ssh
d_t:s0-s0:c0:c1023 tcontext=system_u:object_r:rvar_run_t:s0 tclass=file permissive=1
----
time-->Sat Mar 24 22:39:01 2018
type=AVC msg=audit(1521931141.427:66): avc: denied { write } for pid=661 comm="nscd" path="pipe:[20904]" dev="pipefs" ino=20904 scontext=system_u:system_r:nscd_t:s0
tcontext=system_u:system_r:nscd_t:s0 tclass=fifo_file permissive=1
----
time-->Sun Mar 25 06:27:17 2018
type=AVC msg=audit(1521939237.856:945): avc: denied { write } for pid=661 comm="nscd" path="pipe:[320891]" dev="pipefs" ino=320891 scontext=system_u:system_r:nscd_t:
s0 tcontext=system_u:system_r:nscd_t:s0 tclass=fifo_file permissive=1
----
time-->Sun Mar 25 12:07:48 2018
type=PROCTITLE msg=audit(1521979668.855:1062): proctitle=737368643A20726F6F74205B707269765D
type=SYSCALL msg=audit(1521979668.855:1062): arch=c000003e syscall=82 success=yes exit=0 a0=7f2971882cbf a1=7f2971882cad a2=0 a3=8 items=0 ppid=673 pid=4295 uid=0 uid=
0 gid=0 euid=0 suid=0 fsuid=0 egid=0 fsgid=0 tty=(none) ses=19 comm="sshd" exe="/usr/sbin/sshd" subj=system_u:system_r:sshd_t:s0-s0:c0:c1023 key=(null)
type=AVC msg=audit(1521979668.855:1062): avc: denied { unlink } for pid=4295 comm="sshd" name="mod.dynamic" dev="tmpfs" ino=20704 scontext=system_u:system_r:sshd_t:
s0-s0:c0:c1023 tcontext=system_u:object_r:rvar_run_t:s0 tclass=file permissive=1
type=AVC msg=audit(1521979668.855:1062): avc: denied { rename } for pid=4295 comm="sshd" name="mod.dynamic.new" dev="tmpfs" ino=534033 scontext=system_u:system_r:ss
hd_t:s0-s0:c0:c1023 tcontext=system_u:object_r:rvar_run_t:s0 tclass=file permissive=1
```

Para crear reglas según los logs que ha dejado es muy útil el paquete «audit2allow». Permite ver las reglas que se crearían para solucionar los errores con el comando «audit2allow -w -a».

```
root@debian-docker-se:~# audit2allow -a

#===== fsadm_t =====
allow fsadm_t initrc_tmp_t:file { getattr ioctl write };
allow fsadm_t root_t:dir { add_name remove_name write };
allow fsadm_t root_t:file { create open read unlink write };

#===== mount_t =====

#!!!! This avc can be allowed using the boolean 'allow_mount_anyfile'
allow mount_t initrc_tmp_t:dir mounton;
allow mount_t iso9660_t:filesystem getattr;

#===== nscd_t =====
allow nscd_t self:fifo_file write;
allow nscd_t var_run_t:dir { create setattr };

#===== ntpd_t =====
allow ntpd_t initrc_lock_t:file write;

#===== sshd_t =====
allow sshd_t var_run_t:file { getattr open read rename unlink };

#===== systemd_detect_virt_t =====
allow systemd_detect_virt_t init_t:file { getattr open read };
allow systemd_detect_virt_t proc_t:file { getattr open read };
```

Además, audit2allow permite generar un módulo con dichas reglas con el comando «audit2allow -a -M *modulo*». Al ser reglas generadas automáticamente deben ser revisadas ya que en su mayoría suelen otorgar más permisos de los que son realmente necesarios.

```
root@debian-docker-se:~# audit2allow -a -M debianInicial
***** IMPORTANT *****
To make this policy package active, execute:

semodule -i debianInicial.pp
```

Una vez creado el módulo se debe insertar y activar con los siguiente comandos:

```
| semodule -i debianInicial.pp
| semodule -e debialnicial
```

Para configurar SELinux correctamente se requiere de conocimientos muy avanzados del funcionamiento del sistema operativo. Una mala configuración puede conllevar tanto la ineficacia con respecto a la seguridad del sistema como al bloqueo de funcionalidades legítimas por lo que se considera una herramienta complicada para los usuarios medios.

3.2.2 AppArmor

AppArmor^{[19][20][21][22][23]} al igual que SELinux provee al kernel de un control de acceso obligatorio. Comenzó como un componente de la antigua plataforma Immunix Linux y fue liberado en 2006. Se considera una alternativa a SELinux, siendo más sencilla de entender, instalar y mantener, de manera que puede ser usada por un usuario medio.

AppArmor trabaja con rutas en lugar de etiquetas de forma que realiza pocas modificaciones en el sistema de fichero. Esto le da más flexibilidad a la hora de trabajar con diferentes tipos de sistemas de ficheros.

Al igual que SELinux dispone de un modo que genera logs de incidencias sin bloquear el funcionamiento y permite observar el comportamiento habitual de los procesos mientras se utilizan. En este caso, el «enforce» es el modo que obliga a cumplir los permisos mientras que «complain» solo registra las incidencias.

En Debian se puede instalar mediante el comando «apt install apparmor apparmor-utils». En este caso no se dispone de script de activación por lo que se deben ejecutar los siguientes comandos manualmente para modificar el grub:

```
mkdir -p /etc/default/grub.d
$ echo 'GRUB_CMDLINE_LINUX_DEFAULT="$GRUB_CMDLINE_LINUX_DEFAULT \
apparmor=1 security=apparmor"' | sudo tee
/etc/default/grub.d/apparmor.cfg
$ update-grub
```

Una vez reiniciado el sistema se encontrará en funcionamiento.

Para comprobar los perfiles cargados se puede utilizar el comando «aa-status».

```
root@debian-docker:~# aa-status
apparmor module is loaded.
1 profiles are loaded.
1 profiles are in enforce mode.
  /usr/sbin/ntpd
0 profiles are in complain mode.
1 processes have profiles defined.
1 processes are in enforce mode.
  /usr/sbin/ntpd (714)
0 processes are in complain mode.
0 processes are unconfined but have a profile defined.
```

El comando «ps auxZ | grep -v '^unconfined'» mostrará todos los procesos ejecutados que están siendo afectados por el perfil de AppArmor.

```
root@debian-docker:~# ps auxZ | grep -v '^unconfined'
LABEL          USER          PID %CPU %MEM    VSZ   RSS TTY      S
TAT START    TIME COMMAND
/usr/sbin/ntpd (enforce)  ntp          714  0.0  0.3  95760  3064 ?        S
sl  16:52    0:00 /usr/sbin/ntpd -p /var/run/ntpd.pid -g -u 105:109
```

Todos los perfiles activos se encuentran en «/etc/apparmor.d ».

```
root@debian-docker:/etc/apparmor.d# ls
abstractions  cache  disable  force-complain  local  tunables  usr.sbin.ntpd
```

El sistema solo dispone del perfil ntpd así que para cargar más perfiles predefinidos que pueden servir de ayuda, se utiliza el comando «apt-get apparmor-profiles». Una vez instalados se pueden ver en «/etc/apparmor.d ».

```

root@debian-docker:/etc/apparmor.d# ls
abstractions          usr.lib.dovecot.config          usr.lib.dovecot.ssl-params
apache2.d             usr.lib.dovecot.deliver        usr.sbin.avahi-daemon
bin.ping              usr.lib.dovecot.dict           usr.sbin.dnsmasq
cache                 usr.lib.dovecot.dovecot-auth   usr.sbin.dovecot
disable               usr.lib.dovecot.dovecot-lda    usr.sbin.identd
force-complain        usr.lib.dovecot.imap           usr.sbin.mdnisd
local                 usr.lib.dovecot.imap-login     usr.sbin.nmbd
sbin.klogd            usr.lib.dovecot.lmtp           usr.sbin.nscd
sbin.syslogd          usr.lib.dovecot.log            usr.sbin.ntpd
sbin.syslog-ng        usr.lib.dovecot.managesieve    usr.sbin.smbd
tunables              usr.lib.dovecot.managesieve-login
usr.lib.dovecot.anvil usr.lib.dovecot.pop3           usr.sbin.smbldap-useradd
usr.lib.dovecot.auth  usr.lib.dovecot.pop3-login     usr.sbin.traceroute

```

Los nuevos perfiles se cargan con modo «complain» como se puede observar en la siguiente captura:

```

root@debian-docker:/etc/apparmor.d# aa-status
apparmor module is loaded.
34 profiles are loaded.
1 profiles are in enforce mode.
  /usr/sbin/ntpd
33 profiles are in complain mode.
  /usr/lib/dovecot/anvil
  /usr/lib/dovecot/auth
  /usr/lib/dovecot/config
  /usr/lib/dovecot/deliver
  /usr/lib/dovecot/dict
  /usr/lib/dovecot/dovecot-auth
  /usr/lib/dovecot/dovecot-lda
  /usr/lib/dovecot/dovecot-lda//usr/sbin/sendmail
  /usr/lib/dovecot/imap
  /usr/lib/dovecot/imap-login
  /usr/lib/dovecot/lmtp
  /usr/lib/dovecot/log
  /usr/lib/dovecot/managesieve

```

Para cambiar el modo de los perfiles se utilizan los comandos «aa-complain *my-profile*» y «aa-enforce *my-profile*».

Para crear un perfil nuevo para una aplicación se pueden seguir los siguientes pasos:

1. Crear el fichero vacío con «aa-autodep *app*» siendo *app* el nombre de la aplicación. La aplicación debe estar instalada en el sistema.
2. Activar el perfil en modo complain «aa-complain *app*».
3. Utilizar la aplicación de forma normal para que queden registrados su funcionamiento habitual.
4. Actualizar los perfiles según el fichero de auditoría «aa-logprof»
5. Editar el fichero, que se encuentra en `/etc/apparmor.d`, para revisar las reglas añadidas y editar las que correspondan.
6. Recargar los perfiles «`/etc/init.d/apparmor reload`» para asegurar que los cambios tengan efecto.

Otra opción consiste en ejecutar «aa-genprof *app*» para crear el perfil de forma interactiva como se muestra a continuación.

Para la prueba se ha creado un script sencillo que crea y borra un fichero «*myapp.sh*».


```

root@debian-docker:~# aa-genprof myapp.sh
Writing updated profile for /root/myapp.sh.
Setting /root/myapp.sh to complain mode.

Before you begin, you may wish to check if a
profile already exists for the application you
wish to confine. See the following wiki page for
more information:
http://wiki.apparmor.net/index.php/Profiles

Please start the application to be profiled in
another window and exercise its functionality now.

Once completed, select the "Scan" option below in
order to scan the system logs for AppArmor events.

For each AppArmor event, you will be given the
opportunity to choose whether the access should be
allowed or denied.

Profiling: /root/myapp.sh

[(S)can system log for AppArmor events] / (F)inish

```

Seleccionamos «Scan» después de iniciar la aplicación en otra ventana y ejecutarla con normalidad:

```

Reading log entries from /var/log/audit/audit.log.
Updating AppArmor profiles in /etc/apparmor.d.

Profile: /root/myapp.sh
Execute: /bin/touch
Severity: unknown

(I)nherit / (C)hild / (N)amed / (X)ix On / (D)eny / Abo(r)t / (F)inish

```

Nos permite seleccionar alguna de estas opciones:

- Inherit: El hijo hereda el perfil del padre y se ejecuta con las mismas restricciones.
- Child: El hijo tendrá su propio perfil dentro de los perfiles del proceso padre.
- Named: Utiliza un perfil ya existen que se deberá especificar.
- Xix On: El hijo puede tener su propio perfil pero usará el del padre si no existe.
- Deny: No ejecuta el programa solicitado.
- Abort: Aborta la ejecución de aa-genprof y se perderán las reglas introducidas. El perfil no será modificado.
- Finish: Termina la ejecución de aa-genprof. Se crean las reglas introducidas hasta el momento y se modifica el perfil. Tiene efecto una vez recargado apparmor.

En este caso seleccionamos «I».

```

Profile: /root/myapp.sh
Execute: /bin/rm
Severity: unknown

(I)nherit / (C)hild / (N)amed / (X)ix On / (D)eny / Abo(r)t / (F)inish

```

Seleccionamos «I» de nuevo.

```

Profile: /root/myapp.sh
Path: /dev/tty
New Mode: rw
Severity: 9

[1 - #include <abstractions/consoles>]
 2 - /dev/tty rw,
(A)llow / [(D)eny] / (I)gnore / (G)lob / Glob with (E)xtension / (N)ew / Audi(t)
 / Abo(r)t / (F)inish
Adding #include <abstractions/consoles> to profile.

```

Las opciones para las nuevas reglas son:

- Allow: Permite el acceso al fichero o directorio.
- Deny: Deniega el acceso al fichero o directorio.
- Ignore: Se salta la regla de manera que volverá a aparecer la próxima vez que se ejecute.
- New: Pedirá que se proporcione una nueva ruta que será añadida a lista de opciones.
- Glob: Reemplaza la última parte con un comodín, creando una nueva entrada en la lista de opciones.
- Glob with Extension: Como la anterior pero se mantendrá la extensión.
- Abort: Aborta la ejecución de aa-genprof y se perderán las reglas introducidas. El perfil no será modificado.
- Finish: Termina la ejecución de aa-genprof. Se crean las reglas introducidas hasta el momento y se modifica el perfil. Tiene efecto una vez recargado apparmor.

Seleccionamos la opción «A» ya que el script necesita acceso a la interfaz TTY para escribir en consola. Nos preguntará por más reglas que hemos respondido con la misma opción para esta prueba.

```

= Changed Local Profiles =

The following local profiles were changed. Would you like to save them?

[1 - /root/myapp.sh]
(S)ave Changes / Save Selec(t)ed Profile / [(V)iew Changes] / View Changes b/w
(C)lean profiles / Abo(r)t

```

Finalmente nos muestra que ha terminado nos da la opción de guardar, ver, limpiar o abortar los cambios.

```

Writing updated profile for /root/myapp.sh.

Profiling: /root/myapp.sh

[(S)can system log for AppArmor events] / (F)inish
Setting /root/myapp.sh to enforce mode.

Reloaded AppArmor profiles in enforce mode.

Please consider contributing your new profile!
See the following wiki page for more information:
http://wiki.apparmor.net/index.php/Profiles

Finished generating profile for /root/myapp.sh.

```

Ya tenemos realizados los cambios y podemos observar nuestro nuevo perfil:

```

root@debian-docker:/etc/apparmor.d# ls
abstractions          usr.lib.dovecot.imap-log
apache2.d             usr.lib.dovecot.lmtp
bin.ping              usr.lib.dovecot.log
cache                 usr.lib.dovecot.managesi
disable               usr.lib.dovecot.managesi
force-complain        usr.lib.dovecot.pop3
local                 usr.lib.dovecot.pop3-log
root.myapp.sh         usr.lib.dovecot.ssl-para
shim_klogd            usr.shim.avahi-daemon

```

Si modificamos el script para que intente escribir el fichero en nuevo directorio veremos el siguiente error al ejecutar.

```

root@debian-docker:~# ./myapp.sh
Ejecutando mi aplicacion
touch: cannot touch 'prueba/myFile.txt': Permission denied
Fichero creado
rm: cannot remove 'prueba/myFile.txt': No such file or directory
Fichero eliminado

```

Podemos realizar los pasos de la creación de perfiles a partir del punto 4 para añadir nuevas reglas y permitir los nuevos comportamientos de las aplicaciones.

Los logs se almacenan en «/var/log/audit/audit.log» por lo que para ver las incidencias se puede ejecutar el comando «cat /var/log/audit/audit.log | grep DENIED».

```

root@debian-docker:~# cat /var/log/audit/audit.log | grep DENIED
type=AVC msg=audit(1523141914.098:2070): apparmor="DENIED" operation="mknod" pro
file="/root/myapp.sh" name="/root/prueba/myFile.txt" pid=4225 comm="touch" reque
sted mask="c" denied mask="c" fsuid=0 ouid=0

```

Finalmente, se ha decidido activar AppArmor por su menor complejidad. Además, Docker permite asignar el perfil por defecto para Docker de AppArmor o perfiles previamente definidos para cada contenedor al ejecutarlos de la forma «docker run --rm -it --security-opt apparmor=*my-profile aplicacion*».

4. Instalación y configuración de Docker

Docker recomienda que en el servidor se ejecute Docker de forma exclusiva y añadir los servicios necesarios dentro de los contenedores. Como se está usando un sistema nuevo con los servicios mínimos no es necesario realizar ninguna acción adicional.

La versión de Docker a utilizar es la Community Edition ya que es gratuita y contiene las funcionalidades suficientes para cumplir el objetivo del proyecto.

Existen tres formas de instalar Docker en Debian:

- Mediante los repositorios de Docker. Facilita la instalación y las tareas de actualización. Es la forma recomendada.
- Descargando los paquetes DEB^[24] e instalándolos y actualizándolos manualmente. Es útil cuando se debe instalar en sistemas sin acceso a internet.
- Utilizando scrips automatizados para instalar versiones de pruebas de una manera rápida y no interactiva. No se recomienda para entornos de producción.

En el proyecto se ha instalado con la primera opción, ejecutando los siguientes comandos para preparar el sistema:

```
sudo apt-get update
sudo apt-get install apt-transport-https ca-certificates curl
gnupg2 software-properties-common
```

Añadimos la clave GPG oficial de Docker y verificamos la huella buscando por los 8 últimos caracteres:

```
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo
apt-key add -
apt-key fingerprint 0EBFCD88
```

```
root@debian-docker:~# curl -fsSL https://download.docker.com/linux/debian/gpg |
apt-key add -
OK
root@debian-docker:~# apt-key fingerprint 0EBFCD88
pub  rsa4096 2017-02-22 [SCEA]
     9DC8 5822 9FC7 DD38 854A E2D8 8D81 803C 0EBF CD88
uid  [ unknown] Docker Release (CE deb) <docker@docker.com>
sub  rsa4096 2017-02-22 [S]
```

Y añadimos el repositorio estable:

```
add-apt-repository \
    "deb [arch=amd64] https://download.docker.com/linux/debian \
    $(lsb_release -cs) \
    stable"
```

Para la instalación de Docker CE se deben ejecutar los siguientes comando (incluye la actualización apt):

```
apt-get update
apt-get install docker-ce
```

Verificamos que se ha instalado correctamente:

```
systemctl status docker
```

```

root@debian-docker:~# systemctl status docker
Excess arguments.
root@debian-docker:~# systemctl status docker
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: e
   Active: active (running) since Sun 2018-04-08 15:58:05 UTC; 5min ago
     Docs: https://docs.docker.com
   Main PID: 11363 (dockerd)
   CGroup: /system.slice/docker.service
           └─11363 /usr/bin/dockerd -H fd://
             └─11368 docker-containerd --config /var/run/docker/containerd/contain
Apr 08 15:58:05 debian-docker dockerd[11363]: time="2018-04-08T15:58:05.15930865
Apr 08 15:58:05 debian-docker dockerd[11363]: time="2018-04-08T15:58:05.16112572

```

Se debe crear un usuario para el contenedor y asignarle el grupo de Docker correspondiente para que pueda ejecutarlo ya que no es una buena práctica trabajar con root. Sin embargo, hay que tener en cuenta que al añadir al usuario el grupo Docker le da la capacidad de ejecutar contenedores lo que implica que podría obtener privilegios root. Se puede añadir el directorio / al contenedor de manera que podrá alterar el sistema de ficheros host sin ninguna restricción (si no se dispone de ninguna herramienta de seguridad más como SELinux o AppArmor).

Con los siguientes comandos podemos ver el grupo y asignarlo al usuario que deseamos.

```

| cd /var/run
| ls -al docker.sock
| usermod -aG docker user

```

Se ha añadido además el perfil de AppArmor para el demonio de Docker Engine que se proporcionan en el proyecto Moby^[25] (Anexo I) para evitar que tenga acceso ilimitado ya que se ejecuta como root.

Siguiendo las pautas de «CIS Docker 1.13.0 Benchmark»^[26] se han tenido en cuenta los siguientes aspectos.

- Es importante auditar correctamente el sistema para lo que usaremos el paquete auditd. Se deberá crear el fichero «/etc/audit/audit.d/docker.rules» y añadir las siguientes reglas para cubrir el funcionamiento de docker :

- Para el demonio Docker:

```

| -w /usr/bin/docker -k docker

```

- Para los ficheros y directorios Docker:

```

| -w /var/lib/docker -k docker
| -w /etc/docker -k docker
| -w /etc/default/docker -k docker
| -w /etc/docker/daemon.json -k docker
| -w /usr/bin/docker-containerd -k docker
| -w /usr/bin/docker-runc -k docker

```

- Si se usarán docker.service y docker.socket se deberán añadir también las siguientes reglas:

```

| -w /usr/lib/systemd/system/docker.service -k docker
| -w /usr/lib/systemd/system/docker.socket -k docker

```

Para que funcionen las nuevas reglas se debe reiniciar el servicio con el comando «service auditd restart».

- En temas red, por defecto Docker permite la comunicación entre todos los contenedores del mismo host. Es recomendable desactivar dicha opción y si fuera necesaria se especificaría la conexión para cada contenedor en concreto. Utilizaremos el fichero de configuración del demonio «/etc/docker/daemon.json» (lo crearemos si no existe):

```
{
  "icc": false
}
```

- La opción ulimit en los contenedores permite controlar los recursos a los que va a poder acceder un usuario o proceso. Por ejemplo, nfiles (número de ficheros) nproc (número de procesos por usuario). Es recomendable establecer estos parámetros por defecto para que afecten a todos los contenedores pero existe un bug a la hora de configurarlo en el daemon.json así que hasta que se resuelva se deberá o ejecutar el demonio con dicha opción «dockerd --default-ulimit nproc=1024:2408 --default-ulimit nfile=100:200» o asegurarnos de añadir dicha opción a cada contenedor.
- Añadiremos los drivers de logs por defecto en el fichero de configuración del demonio «daemon.json» aunque se pueden especificar diferentes para cada contenedor. El fichero de configuración queda de la siguiente forma:

```
{
  "icc": false,
  "log-driver": "syslog"
}
```

- Para evitar que se interrumpa la ejecución del contenedor si el servicio del demonio no está disponible se debe activar el parámetro «live-restore» en el fichero «daemon.json» como se muestra a continuación. De esta forma además será más fácil actualizar el demonio sin provocar paradas en los servicios de los contenedores.

```
{
  "icc": false,
  "log-driver": "syslog",
  "live-restore": true
}
```

- Se recomienda utilizar "hairpin NAT" en lugar de los proxy de usuario para redirigir los puertos del host a los contenedores. Esto mejora el rendimiento ya que utiliza la funcionalidad nativa de iptables de Linux en lugar de componentes adicionales.

```
{
  "icc": false,
  "log-driver": "syslog",
  "live-restore": true,
  "userland-proxy": false
}
```

En el documento «CIS Docker 1.13.0 Benchmark» se pueden observar más consejos de seguridad para el uso de Docker Swarm, si se requiere que Docker sea accesible a través de la red, etc. No se han tenido en cuenta ya que el proyecto no lo utiliza pero será necesario revisar la guía cada vez que se introduzca una funcionalidad nueva.

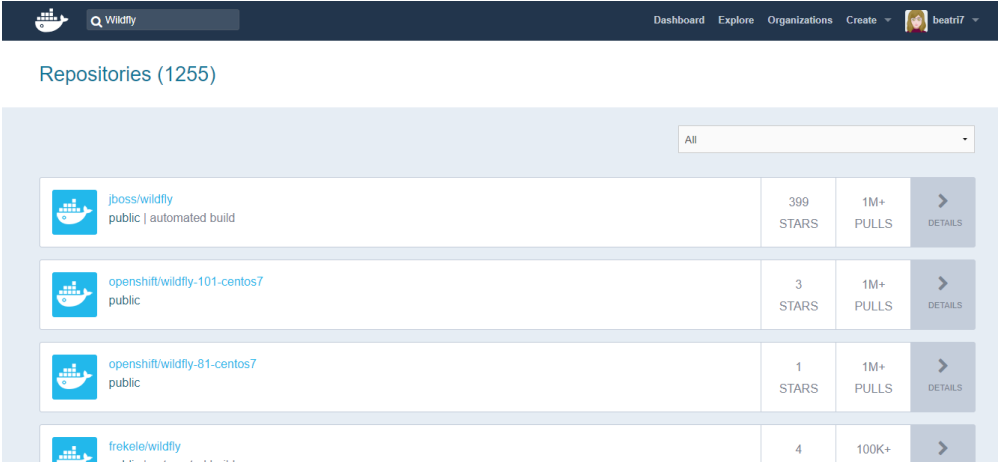
5. Imágenes Docker

Para crear nuestra imagen vamos a utilizar una imagen base de las almacenadas en Docker Hub. Es importante no confiar en cualquier imagen que se encuentre en internet ni en cualquier repositorio. Docker patrocina un equipo dedicado exclusivamente a revisar y publicar contenido en los repositorios oficiales. Existen repositorios oficiales para los casos de uso más comunes que proporcionan documentación clara y promueven las buenas prácticas por lo que es una buena estrategia partir de alguno de ellos.

Para el proyecto se busca una imagen base para servidores Wildfly con el comando de búsqueda de Docker.

```
user@debian-docker:~$ docker search wildfly
NAME                DESCRIPTION                STARS    OFFICIAL    AUTOMATED
jboss/wildfly       WildFly application server image    399      [OK]
apiman/on-wildfly10 The full apiman quickstart running on WildFL... 8        [OK]
openshift/wildfly-8-centos DEPRECATED - see openshift/wildfly-81-centos7 6        [OK]
piegsaj/wildfly     WildFly 10.0.0.Final on Java 8      6        [OK]
wildflyext/wildfly-camel WildFly with Camel Subsystem        5
aerogear/unifiedpush-wildfly 4        [OK]
frekele/wildfly     docker run --rm --name wildfly -p 8080:8080 ... 4        [OK]
openshift/wildfly-101-centos7 A Centos7 based WildFly v10.1 image for use ... 3
jboss/keycloak-adapter-wildfly 2        [OK]
bitnami/wildfly     Bitnami WildFly Docker Image        2
openshift/wildfly-110-centos7 A Centos7 based WildFly v11.0 image for use ... 1
ceagan/wildfly      JBoss Wildfly with XMLStarlet       1        [OK]
openshift/wildfly-100-centos7 A Centos7 based WildFly v10.0 image for use ... 1
openshift/wildfly-81-centos7 A Centos7 based WildFly v8.1 image for use w... 1
rhche/wildfly-swarm 1
surveysampling/wildfly-base base wildfly image                   0
caltha/wildfly      Self contained WildFly application server    0        [OK]
splatform/wildfly-resource 0
openshift/wildfly-120-centos7 A Centos7 based WildFly v12.0 image for use ... 0
cfje/wildfly-resource 0
fksaito/wildfly     Modificações em imagens wildfly para expor p... 0
dthuilot/wildfly-admin wildfly with admin console enabled.    0        [OK]
devcomb/keycloak-adapter-wildfly Similar to 'jboss/keycloak-adapter-wildfly' ... 0
openshift/wildfly-90-centos7 A Centos7 based WildFly v9.0 image for use w... 0
devbeta/wildfly     0
```

También se pueden ver en Docker Hub o Docker Store:




The screenshot shows the Docker Hub search results for 'wildfly'. The search bar contains 'wildfly' and the user 'beatriz' is logged in. The results show a list of repositories with columns for name, description, stars, pulls, and details. The top result is 'jboss/wildfly' with 399 stars and 1M+ pulls. Other results include 'openshift/wildfly-101-centos7', 'openshift/wildfly-81-centos7', and 'frekele/wildfly'.

Repository	Description	Stars	Pulls	Details
jboss/wildfly	public automated build	399	1M+	>
openshift/wildfly-101-centos7	public	3	1M+	>
openshift/wildfly-81-centos7	public	1	1M+	>
frekele/wildfly	public automated build	4	100K+	>

Observamos que no existe ninguna versión oficial por lo que se deben extremar las precauciones a la hora de utilizarlos. La primera opción tiene 399 estrellas y revisando el detalle vemos que su propietario es JBoss Developer.

Esta organización es la que utiliza JBoss para publicar sus imágenes.

Repo Info	Tags	Dockerfile	Build Details
Short Description WildFly application server image		Docker Pull Command <pre>docker pull jboss/wildfly</pre>	
Full Description WildFly Docker image This is an example Dockerfile with WildFly application server .		Owner  jboss	
Usage To boot in standalone mode <pre>docker run -it jboss/wildfly</pre>		Source Repository JBoss-Dockerfiles/wildfly	

Un aspecto importante a tener en cuenta al descargar o subir imágenes es que se está usando un medio no seguro como es internet. Por esto es importante disponer de mecanismos que nos aseguren la integridad y la autenticidad de las imágenes. Docker provee la funcionalidad «Content trust», esta opción fuerza que se firmen y verifiquen los tags de las imágenes en el lado del cliente. Viene desactivado por defecto por lo que debemos ejecutar el comando «export DOCKER_CONTENT_TRUST=1».

```
root@debian-docker:~# export DOCKER_CONTENT_TRUST=1
root@debian-docker:~# docker pull jboss/wildfly:latest
Error: remote trust data does not exist for docker.io/jboss/wildfly: notary.docker
.io does not have trust data for docker.io/jboss/wildfly
root@debian-docker:~# docker pull --disable-content-trust jboss/wildfly:latest
latest: Pulling from jboss/wildfly
469cfcc7a4b3: Pull complete
05677e4d61f0: Pull complete
a9520f492457: Pull complete
4d201219d6b1: Pull complete
70114b0e83fa: Pull complete
Digest: sha256:d74585a74bd4f6bc16feb62ec48156183a8e95d0a5b2221b3cbbcd7c94dalc39
Status: Downloaded newer image for jboss/wildfly:latest
```

Observamos que no se encuentra firmada por lo que se debe usar la opción «--disable-content-trust». Siempre es mejor tenerlo activado por defecto y utilizar el parámetro de desactivación si es necesario para que seamos conscientes de que no está firmada y debemos revisarla mejor.

En cualquier caso y más en los casos que no se usen imágenes oficiales o no se encuentre firmadas, se deben revisar las imágenes para evitar la introducción de software dañino en nuestro contenedor.

Para ayudarnos a inspeccionar las imágenes podemos utilizar herramientas de auditoría o aprovechar los servicios de almacenamiento de imágenes que suelen incluir un escaneo de las mismas.

Entre los servicios de almacenamiento en la nube se han probado los servicios Quay^[27] y Anchore^[28].

Quay es un servicio de almacenamiento en la nube que además permite automatizar la construcción y despliegue de los contenedores. Además, tiene integración con otros servicios como GitHub y Bitbucket. Es un servicio de pago pero tiene una versión de prueba de 30 días que se ha utilizado para realizar el análisis.

Para utilizarlo se debe crear un repositorio público (privado solo en la versión de pago) e iniciar sesión en Docker.

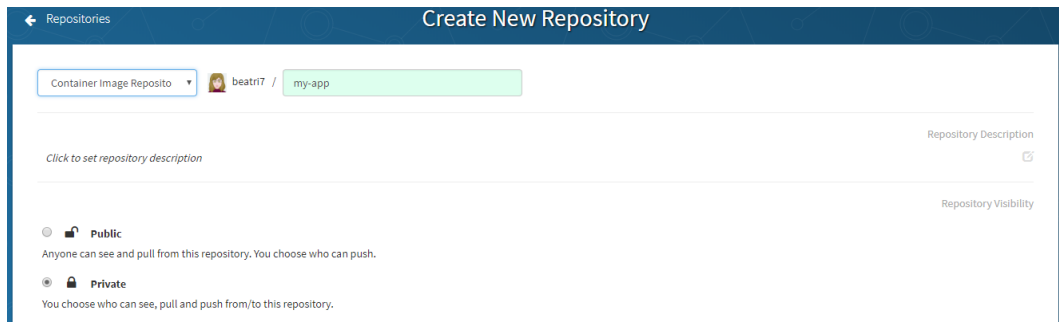


Ilustración 9 Creación repositorio Quay

```
> docker login quay.io
Username: beatr17
Password:
Login Succeeded
```

Una vez iniciada la sesión añadiremos el tag «beatr17/my-app» a la imagen wildfly base para poder analizarla y realizaremos el push:

```
| docker tag jboss/wildfly quay.io/beatr17/my-app
> docker push quay.io/beatr17/my-app:latest
The push refers to repository [quay.io/beatr17/my-app]
b29862dd4117: Layer already exists
46b3016cc5ee: Layer already exists
25392e8f9f5a: Layer already exists
0c8237d7452a: Layer already exists
d9e554ca876f: Layer already exists
43e653f84b79: Layer already exists
latest: digest: sha256:5f9c24dd75d386223ddc1b3f090ed385d75d4992616e8f6ddbc75804d4197e4b size: 14840
```

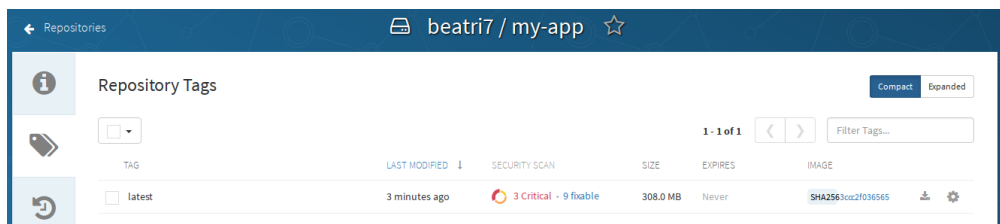


Ilustración 10 Quay Repositories

Seleccionando el tag, en el análisis de seguridad de la web podemos ver que hay 3 vulnerabilidades críticas, 4 de nivel medio y 2 bajas:

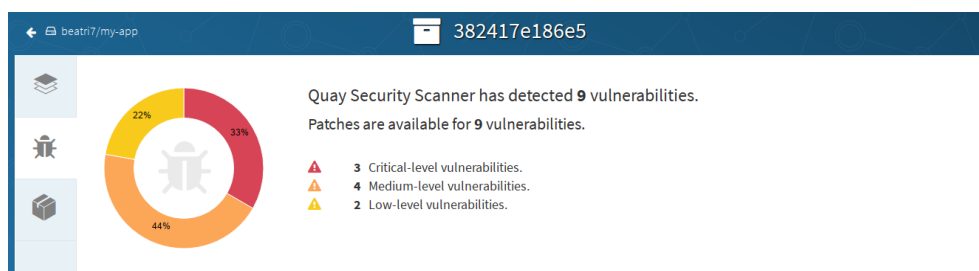


Ilustración 11 Cabecera análisis Quay

CVE	SEVERITY	PACKAGE	CURRENT VERSION	FIXED IN VERSION	INTRODUCED IN IMAGE
RHSA-2018:1191	Critical	java-1.8.0-openjdk-headless	1:1.8.0.161-0.b14.el7_4	1:1.8.0.171-7.b10.el7	RUN yum -y install java-1.8.0-op...
RHSA-2018:1191	Critical	java-1.8.0-openjdk	1:1.8.0.161-0.b14.el7_4	1:1.8.0.171-7.b10.el7	RUN yum -y install java-1.8.0-op...
RHSA-2018:1191	Critical	java-1.8.0-openjdk-devel	1:1.8.0.161-0.b14.el7_4	1:1.8.0.171-7.b10.el7	RUN yum -y install java-1.8.0-op...
RHSA-2018:0998	Medium	openssl-libs	1:1.0.2k-8.el7	1:1.0.2k-12.el7	ADD file:f755805244a649eccae3a3e...
RHSA-2018:0805	Medium	glibc	2.17-196.el7_4.2	0:2.17-222.el7	ADD file:f755805244a649eccae3a3e...
RHSA-2018:0805	Medium	glibc-common	2.17-196.el7_4.2	0:2.17-222.el7	ADD file:f755805244a649eccae3a3e...
RHSA-2018:0666	Medium	krb5-libs	1.15.1-8.el7	0:1.15.1-18.el7	ADD file:f755805244a649eccae3a3e...
RHSA-2018:0849	Low	libgcc	4.8.5-16.el7_4.2	0:4.8.5-28.el7	ADD file:f755805244a649eccae3a3e...
RHSA-2018:0849	Low	libstdc++	4.8.5-16.el7_4.2	0:4.8.5-28.el7	ADD file:f755805244a649eccae3a3e...

Ilustración 12 Detalle análisis Quay

Son vulnerabilidades introducidas debidas a la versión de Java y las diferentes librerías. Todas han sido encontradas en el mismo mes en el que se ha hecho el análisis por lo que se realizará un seguimiento para comprobar si se actualizará la imagen base de Wildfly o debemos tomar otras medidas.

Además de la vulnerabilidad y su nivel se puede ver la versión en la que se ha solucionado la vulnerabilidad, el punto de la imagen donde se encuentra dicha vulnerabilidad y una descripción de la misma (si pulsamos encima):

CVE	SEVERITY	PACKAGE	CURRENT VERSION	FIXED IN VERSION	INTRODUCED IN IMAGE
RHSA-2018:1191	Critical	java-1.8.0-openjdk-headless	1:1.8.0.161-0.b14.el7_4	1:1.8.0.171-7.b10.el7	RUN yum -y install java-1.8.0-open...

DESCRIPTION

The java-1.8.0-openjdk packages provide the OpenJDK 8 Java Runtime Environment and the OpenJDK 8 Java Software Development Kit. Security Fix(es): * OpenJDK: incorrect handling of Reference clones can lead to sandbox bypass (Hotspot, 8192025) (CVE-2018-2814) * OpenJDK: unrestricted deserialization of data from JCEKS key stores (Security, 8189997) (CVE-2018-2794) * OpenJDK: insufficient consistency checks in deserialization of multiple classes (Security, 8189977) (CVE-2018-2795) * OpenJDK: unbounded memory allocation during deserialization in PriorityBlockingQueue (Concurrency, 8189981) (CVE-2018-2796) * OpenJDK: unbounded memory allocation during deserialization in TabularDataSupport (JMX, 8189985) (CVE-2018-2797) * OpenJDK: unbounded memory allocation during deserialization in Container (AWT, 8189989) (CVE-2018-2798) * OpenJDK: unbounded memory allocation during deserialization in NamedNodeMapImpl (JAXP, 8189993) (CVE-2018-2799) * OpenJDK: RMI HTTP transport enabled by default (RMI, 8193833) (CVE-2018-2800) * OpenJDK: unbounded memory allocation during deserialization in StubIORImpl (Serialization, 8192757) (CVE-2018-2815) * OpenJDK: incorrect merging of sections in the JAR manifest (Security, 8189969) (CVE-2018-2790) For more details about the security issue(s), including the impact, a CVSS score, and other related information, refer to the CVE page(s) listed in the References section. Note: If the web browser plug-in provided by the icedtea-web package was installed, the issues exposed via Java applets could have been exploited without user interaction if a user visited a malicious website.

Ilustración 13 Descripción vulnerabilidad Quay

Anchore es otro servicio en la nube que permite encontrar, inspeccionar en profundidad y analizar las imágenes. Permite un repositorio privado con todas las funcionalidades de forma gratuita solo registrándose.

Se pueden buscar y analizar imágenes públicas con facilidad:

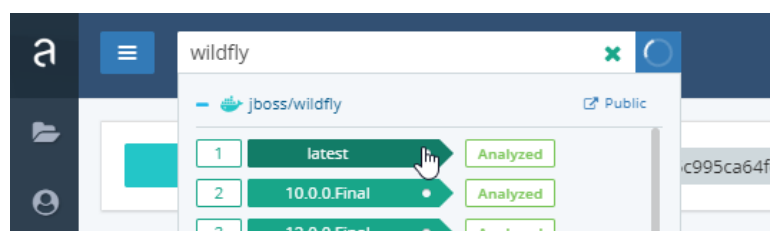


Ilustración 14 Búsqueda Anchore

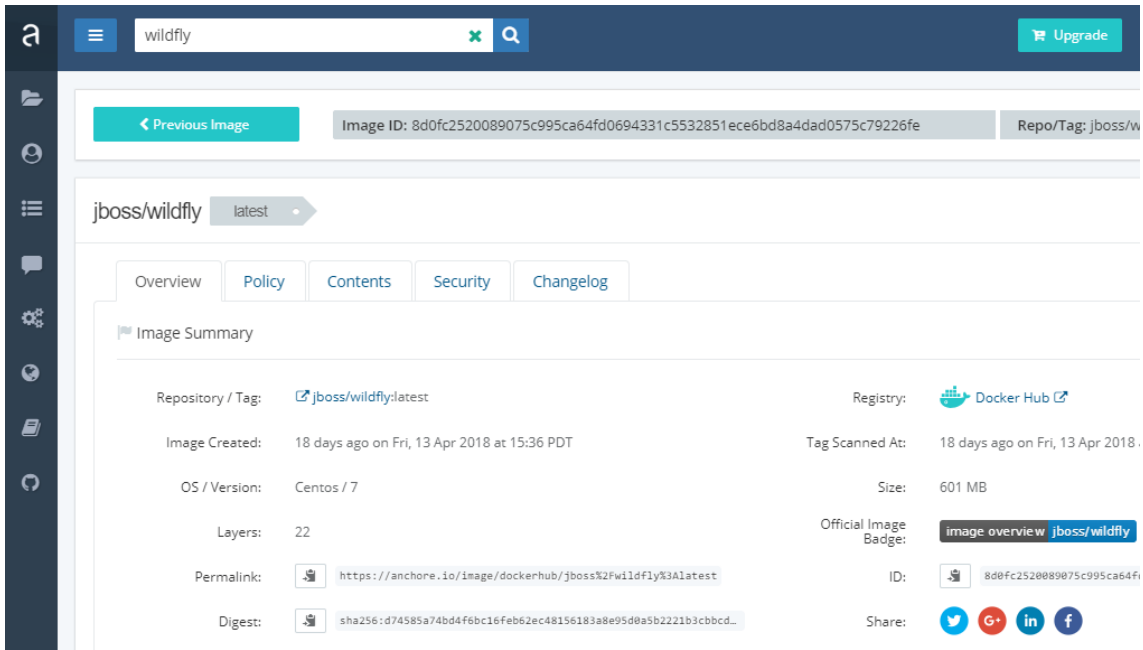


Ilustración 15 Repositorio jboss/wildfly Anchore

Las vulnerabilidades encontradas en Quay y Anchore son las mismas:

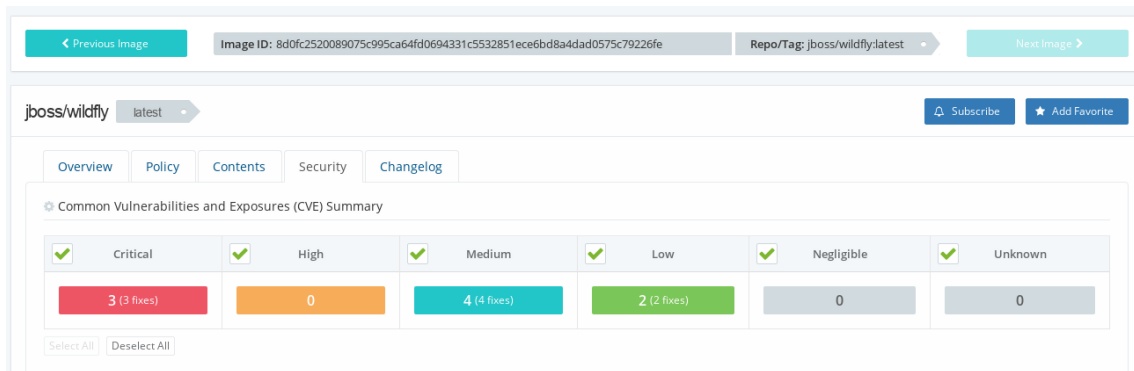


Ilustración 16 Cabecera análisis Anchore

CVE List

Show only CVEs with fixes Display CVEs Filter the CVE list: Previous Next

CVE ID	Severity	Vulnerable Package	Fix Available	URL
RHSA-2018:1191	CRITICAL	java-1.8.0-openjdk-1.8.0.161-0.b14...	1:1.8.0.171-7.b10.e17	https://access.redhat.com/errata/RHSA-2018:1191
RHSA-2018:1191	CRITICAL	java-1.8.0-openjdk-devel-1.8.0.161...	1:1.8.0.171-7.b10.e17	https://access.redhat.com/errata/RHSA-2018:1191
RHSA-2018:1191	CRITICAL	java-1.8.0-openjdk-headless-1.8.0.1...	1:1.8.0.171-7.b10.e17	https://access.redhat.com/errata/RHSA-2018:1191
RHSA-2018:0805	MEDIUM	glibc-2.17-196.e17_4.2	0:2.17-222.e17	https://access.redhat.com/errata/RHSA-2018:0805
RHSA-2018:0805	MEDIUM	glibc-common-2.17-196.e17_4.2	0:2.17-222.e17	https://access.redhat.com/errata/RHSA-2018:0805
RHSA-2018:0666	MEDIUM	krb5-libs-1.15.1-8.e17	0:1.15.1-18.e17	https://access.redhat.com/errata/RHSA-2018:0666
RHSA-2018:0998	MEDIUM	openssl-libs-1.0.2k-8.e17	1:1.0.2k-12.e17	https://access.redhat.com/errata/RHSA-2018:0998
RHSA-2018:0849	LOW	libgcc-4.8.5-16.e17_4.2	0:4.8.5-28.e17	https://access.redhat.com/errata/RHSA-2018:0849
RHSA-2018:0849	LOW	libstdc++-4.8.5-16.e17_4.2	0:4.8.5-28.e17	https://access.redhat.com/errata/RHSA-2018:0849

Ilustración 17 Cabecera análisis Anchore

Para crear la imagen de nuestra aplicación vamos a utilizar de base la imagen «jboss/wildfly» analizada.

La aplicación a utilizar es una aplicación web de prueba muy sencilla hecha en Java. Utiliza como punto de partida, para simplificar su creación, una aplicación de ejemplo de Mkyong^[29] y se modifica para que simplemente muestre un «Hello World» con el usuario escrito por pantalla.

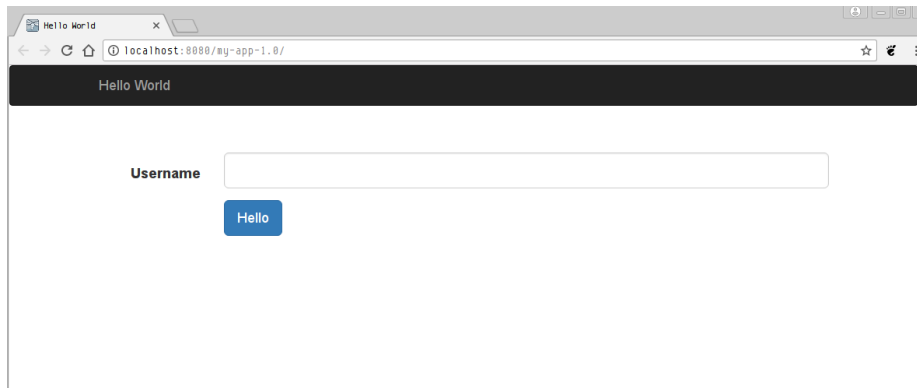


Ilustración 18 Aplicación

Para generar la imagen primero se debe situar el fichero WAR de la aplicación en una carpeta limpia junto al fichero Dockerfile. El Dockerfile será muy simple:

```
Dockerfile
FROM jboss/wildfly:latest
COPY my-app-1.0.war /opt/jboss/wildfly/standalone/deployments/
```

Ilustración 19 Fichero Dockerfile

Aunque el fichero Dockerfile utilizado es muy sencillo se deben tener en cuenta una serie de buenas prácticas en su construcción. Docker, Inc. proporciona una guía cuyo resumen se puede ver en el Anexo II.

Siguiendo dichas guías para la construcción de nuestra imagen se ha utilizado un carpeta que únicamente contiene el fichero Dockerfile y el WAR de la aplicación. Además, se añadió el fichero de la aplicación con la instrucción «COPY» en lugar de «ADD» ya que la primera es más transparente.

Una vez creado se construye con el comando «docker build --tag=beatr17/my-app»

```
> docker build --tag=beatr17/my-app .
Sending build context to Docker daemon 7.079MB
Step 1/3 : FROM jboss/wildfly:latest
----> 8d0fc2520089
Step 2/3 : MAINTAINER Beatriz Gonzalez
----> Using cache
----> 9b4dbea2abd0
Step 3/3 : COPY my-app-1.0.war /opt/jboss/wildfly/standalone/deployments/
----> d66fd5952c7c
Successfully built d66fd5952c7c
Successfully tagged beatr17/my-app:latest
```

Vamos a utilizar las mismas herramientas web para realizar un análisis de seguridad. Además, se va utilizar la herramienta de escritorio Lynis^{[30] [31]}.

Lynis es una herramienta de auditoría open source. Se utiliza para las defensas de seguridad de los sistemas Linux y los basados en Unix. Se ejecuta en el propio host de manera que realiza un escaneo de seguridad más profundo. Además provee de un escaneo específico de ficheros Dockerfiles.

Se puede instalar en el equipo de desarrollo para escanear las imágenes. En Debian se utiliza el siguiente comando apt-get:

```
| apt-get lynis
```

Para realizar la revisión de seguridad del Dockerfile se utiliza el comando «lynis audit Dockerfile *Dockerfile*»

```
[+] Helper: audit_dockerfile
-----
File to audit = ../my-app/Dockerfile

[+] Image
-----
Found image:                               [ jboss/wildfly:latest ]
Unknown image

[+] Basics
-----

[+] Software
-----

[+] Downloads
-----
No files seems to be downloaded in this Dockerfile

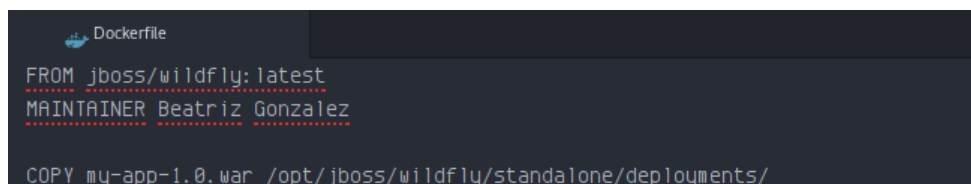
[+] Permissions
-----

=====

-[ Lynis 2.6.3 Results ]-
Warnings (1):
-----
./lynis: 111: /home/manz/bee/lynis/include/report: -F:: not found
! No maintainer found. Unclear who created this file. [dockerfile]
  https://cisofy.com/controls//

No suggestions
```

Nos avisa que no hemos añadido los datos de la persona encargada del mantenimiento de la imagen. Además, si la imagen fuera pública se debería añadir junto al usuario una dirección de correo electrónico o alguna forma de contacto. Lo añadimos al Dockerfile y volvemos a ejecutar Lynis.



```
Dockerfile
FROM jboss/wildfly:latest
MAINTAINER Beatriz Gonzalez
COPY my-app-1.0.war /opt/jboss/wildfly/standalone/deployments/
```

Ilustración 20 Dockerfile actualizado

```
[+] Helper: audit_dockerfile
-----
File to audit = ../my-app/Dockerfile

[+] Image
-----
Found image:                               [ jboss/wildfly:latest ]
Unknown image

[+] Basics
-----
Maintainer                                  [ MAINTAINER Beatriz Gonzalez ]

[+] Software
-----

[+] Downloads
-----
No files seems to be downloaded in this Dockerfile

[+] Permissions
-----
```

-[Lynis 2.6.3 Results]-

Great, no warnings

No suggestions

En esta ejecución no muestra warnings ni sugerencias así que procedemos a subir nuestra imagen a Docker Hub para almacenarla y poder analizarla con Anchore. Para ello creamos nuestro repositorio en Docker Hub.

Create Repository

En nuestro ordenador local debemos iniciar sesión para poder interactuar con Docker Hub:

```
cat ~/my_password.txt | docker login --username user --password-stdin
```

Al enviar la imagen a Docker Hub para firmar el tag nos pedirá que introduzcamos una clave para el repositorio.

```
I > docker push beatri7/my-app:latest
The push refers to repository [docker.io/beatri7/my-app]
83bbf96dc499: Pushed
46b3016cc5ee: Mounted from jboss/wildfly
25392e8f9f5a: Mounted from jboss/wildfly
0c0237d7452a: Mounted from jboss/wildfly
d9e554ca876f: Mounted from jboss/wildfly
43e653f84b79: Mounted from jboss/wildfly
latest: digest: sha256:e7e3760406f01678ead0a9f8f7b07560cecb8cf07d017e5efdaf9ac5d6210567 size: 1584
Signing and pushing trust metadata
Enter passphrase for repository key with ID a7bf68f:
Successfully signed docker.io/beatri7/my-app:latest
```

En Anchore debemos configurar nuestro repositorio privado Docker Hub y nuestra contraseña de acceso al mismo:

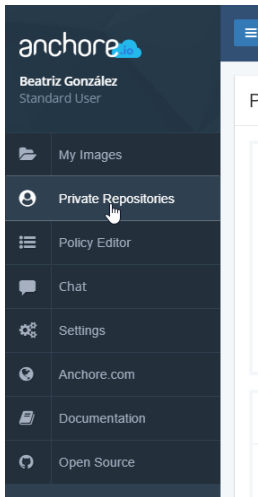


Ilustración 21 Repositorios privados Anchore

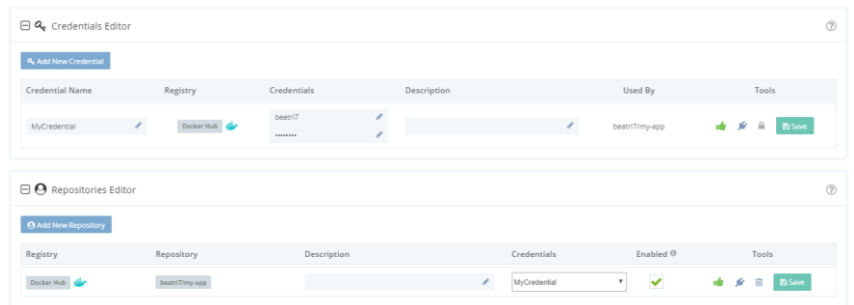


Ilustración 22 Repositorios y credenciales Anchore

Al observar el análisis en Anchore vemos que seguimos obteniendo las mismas vulnerabilidades que existían en la imagen base y que no se ha añadido ninguna vulnerabilidad adicional.

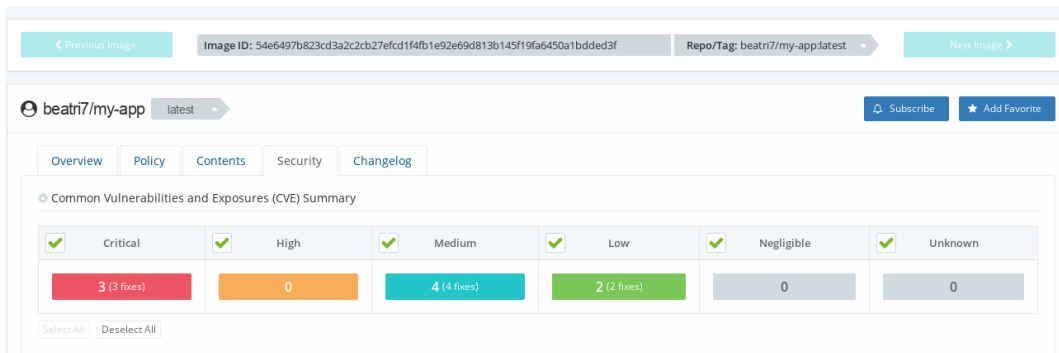


Ilustración 23 Cabecera my-app Anchore

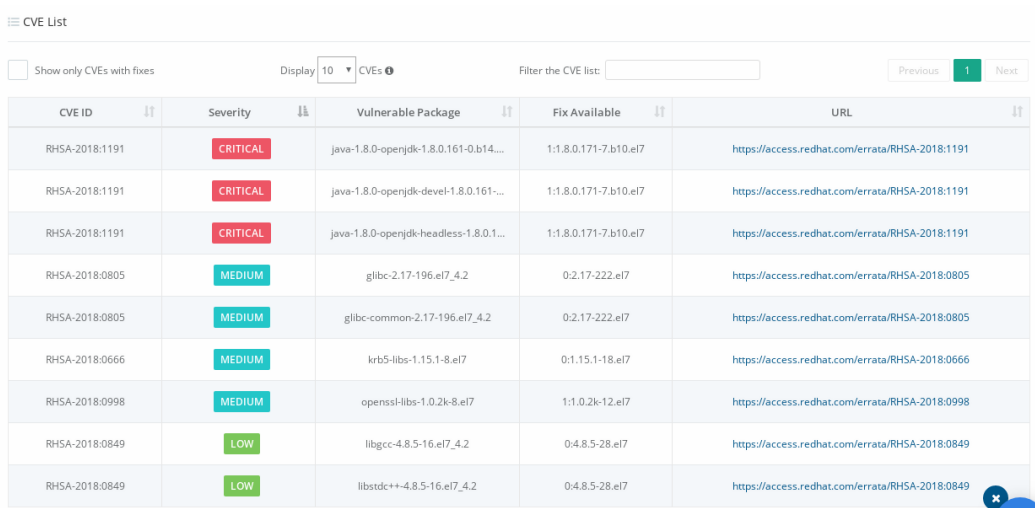


Ilustración 24 Detalle my-app Anchore

Para poder analizar nuestra imagen con Quay eliminamos el repositorio anterior que utilizamos de prueba y creamos uno nuevo donde subirla.

Delete Repository

Deleting a Repository **cannot be undone**. Here be dragons!

Delete Repository

Ilustración 25 Eliminación del repositorio Quay

Al igual que en Anchore las vulnerabilidades detectadas no varían.

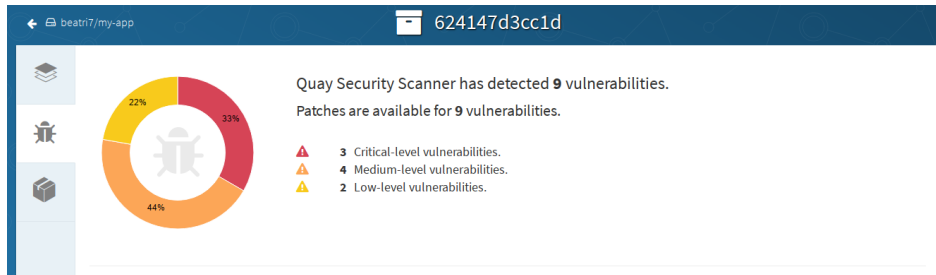


Ilustración 26 Cabecera my-app Quay

Image Vulnerabilities

Filter Vulnerabilities... Only show fixable

CVE	SEVERITY	PACKAGE	CURRENT VERSION	FIXED IN VERSION	INTRODUCED IN IMAGE
RHSA-2018-1191	Critical	java-1.8.0-openjdk	1:1.8.0.161-0.b14.el7_4	1:1.8.0.171-7.b10.el7	RUN yum -y install java-1.8.0-op...
RHSA-2018-1191	Critical	java-1.8.0-openjdk-headless	1:1.8.0.161-0.b14.el7_4	1:1.8.0.171-7.b10.el7	RUN yum -y install java-1.8.0-op...
RHSA-2018-1191	Critical	java-1.8.0-openjdk-devel	1:1.8.0.161-0.b14.el7_4	1:1.8.0.171-7.b10.el7	RUN yum -y install java-1.8.0-op...
RHSA-2018-0998	Medium	openssl-libs	1:1.0.2k-8.el7	1:1.0.2k-12.el7	ADD file:f755805244a649eccc3a3e...
RHSA-2018-0805	Medium	glibc-common	2.17-196.el7_4.2	0:2.17-222.el7	ADD file:f755805244a649eccc3a3e...
RHSA-2018-0666	Medium	krb5-libs	1.15.1-8.el7	0:1.15.1-18.el7	ADD file:f755805244a649eccc3a3e...
RHSA-2018-0805	Medium	glibc	2.17-196.el7_4.2	0:2.17-222.el7	ADD file:f755805244a649eccc3a3e...
RHSA-2018-0849	Low	libstdc++	4.8.5-16.el7_4.2	0:4.8.5-28.el7	ADD file:f755805244a649eccc3a3e...
RHSA-2018-0849	Low	libgcc	4.8.5-16.el7_4.2	0:4.8.5-28.el7	ADD file:f755805244a649eccc3a3e...

Contact Us

Ilustración 27 Detalle my-app Quay

Las imágenes de Docker permiten tratar el entorno como se trata el código añadiendo las instrucciones de construcción de la imagen mediante los ficheros Dockerfile. Esto implica que además se deben tener las mismas consideraciones para su mantenimiento:

- Comentar adecuadamente. Se utiliza el carácter # para comentar y se debe proporcionar información significativa que ayude a entender el código.
- Mantener actualizado. Se debe realizar un seguimiento del código y mantenerlo actualizado teniendo en cuenta especialmente las actualizaciones de seguridad que puedan tener los componentes de la imagen. Es importante proporcionar el nombre de la persona que esté manteniendo dicha imagen y proporcionar un método de contacto para cualquier problema o sugerencia.
- Control de versiones. Se debe seguir un control de versiones del Dockerfile como en cualquier otro código.

6. Ejecución contenedor

En primer lugar nos descargamos la imagen que hemos subido a Docker Hub.

```
root@debian-docker:/home/user# docker pull beatri7/my-app:latest
Pull (1 of 1): beatri7/my-app:latest@sha256:e7e3760406f01678ead0a9f8f7b07560cecb8cf07d017e5efdaf9ac5d6210567
sha256:e7e3760406f01678ead0a9f8f7b07560cecb8cf07d017e5efdaf9ac5d6210567: Pulling
from beatri7/my-app
469cfc7a4b3: Pull complete
05677e4d61f0: Pull complete
a9520f492457: Pull complete
4d201219d6b1: Pull complete
70114b0e83fa: Pull complete
c3d3fd8c8f3c: Pull complete
Digest: sha256:e7e3760406f01678ead0a9f8f7b07560cecb8cf07d017e5efdaf9ac5d6210567
Status: Downloaded newer image for beatri7/my-app@sha256:e7e3760406f01678ead0a9f8f7b07560cecb8cf07d017e5efdaf9ac5d6210567
Tagging beatri7/my-app@sha256:e7e3760406f01678ead0a9f8f7b07560cecb8cf07d017e5efdaf9ac5d6210567 as beatri7/my-app:latest
```

Para ejecutarla se utiliza el comando «run» y como argumento solo es necesario el nombre de la imagen y el mapeo del único puerto necesario 8080 con la opción «-p».

```
| docker run -p 8080:8080 beatri7/my-app
```

Comprobamos que se puede acceder desde nuestro navegador:



Sin embargo dicha ejecución no tiene en cuenta ninguna recomendación de seguridad, por lo que se procede a realizar un análisis y aplicar las modificaciones necesarias en la instrucción siguiendo las pautas de «CIS Docker 1.13.0 Benchmark»:

- Se eliminan todas las capacidades ya que no son necesarias para la aplicación. Se utiliza el argumento «--cap-drop=all»
- Se limita la memoria que va a utilizar el contenedor para controlar su consumo. Si no se controlará el contenedor podría hacer que el sistema se volviera inestable e imposible de utilizar. En principio se ha limitado a 512 MB ya que con 256 no era suficiente «--memory 512m».
- Se establece la prioridad a la hora de compartir la CPU del contenedor sobre otros. Esto permite garantizar que los contenedores con mayor prioridad son servidos mejor. En principio se asignará un 50% ya que no hay más contenedores «--cpu-shares 512»
- El sistema de ficheros root del contenedor debería ser solo de lectura y definir exactamente los volúmenes que se deberían escribir. Esto tiene diferentes ventajas como la inmutabilidad de la infraestructura, reduce el vector de ataque, permite usar backup solo de los volúmenes necesarios, etc. Como nuestra imagen necesita hacer cambios en la carpeta wildfly hemos añadido los argumentos «--read-only -volumen /opt/jboss/wildfly».
- Por defecto los contenedores pueden conectarse con el exterior pero no al revés. Cada conexión externa aparecerá con una de las IP de la propia máquina host. Solo debe permitirse que el contenedor se conecte mediante

una interfaz externa específica ya que si tenemos múltiples interfaces de red en nuestra máquina host podría aceptar conexiones en los puertos expuestos de cualquiera de las interfaces. En nuestro caso especificaremos el puerto y la IP de la máquina en DigitalOcean « -p 165.227.146.197:8080:8080».

- Se recomienda establecer la política de reinicio en caso de fallo a 5 con el argumento «--restart=on-failure:5». Si no se establece un valor y el contenedor sigue intentando reiniciarse podría causar problemas de denegación de servicios en el host. Siempre que el contenedor se termine se debe investigar las causas en lugar de volver a reiniciarlo sin más.
- Tal y como se explica en el punto 4 de instalación de Docker la opción ulimit en los contenedores permite controlar los recursos a los que va a poder acceder un usuario o proceso y debería estar establecida por defecto. Sin embargo, existe un bug a la hora de configurarlo en el daemon.json por lo que se añadirá temporalmente en la ejecución del contenedor la opción «--ulimit nproc=1024:2408 --ulimit nofile=100:4096». El valor 4096 de máximo de los ficheros es el recomendado por JBoss.
- Se debe impedir que los contenedores adquieran privilegios adicionales vía suid o sgit. Para ello el proceso puede fijar el valor del bit no_new_priv en el kernel. De esta forma muchas operaciones peligrosas lo son menos gracias a que no existe la posibilidad de modificar los privilegios. La opción utilizada es «--security-opt=no-new-privileges».
- Uno de los aspectos más importantes en la seguridad es la disponibilidad por lo que se debe usar una instrucción que la compruebe y realice las acciones necesarias en caso de que falle. Los parámetros a añadir en nuestro contenedor para que compruebe si nuestra página es accesible cada hora «--health-cmd='curl -f localhost:8080/my-app-1.0' -health-interval=60m». Se puede comprobar el estado con «docker ps»
- Se debe utilizar la opción «--pids-limit 100» para limitar el número de forks. Esto evita que los atacantes puedan lanzar una bomba fork con un comando dentro del contenedor que podría provocar el fallo de todo el sistema y necesitar el reinicio del host.

El comando de ejecución de nuestro contenedor queda finalmente de la siguiente forma:

```
docker run --cap-drop=ALL \  
  --memory 512m \  
  --read-only --volume /opt/jboss/wildfly \  
  --cpu-shares 512 \  
  -p 165.227.146.197:8080:8080 \  
  --restart=on-failure:5 \  
  --ulimit nproc=1024:2408 --ulimit nofile=100:4096 \  
  --security-opt=no-new-privileges \  
  --health-cmd=' localhost:8080/my-app-1.0' \  
  --health-interval=60m \  
  --pids-limit 100 \  
  beatri7/my-app
```

7. Monitorización de contenedores

Una vez ejecutado el contenedor es importante conocer el estado del mismo y el consumo que hace de los recursos del sistema. Para ello en este apartado se utilizarán diferentes herramientas que nos permitan monitorizar nuestros contenedores.

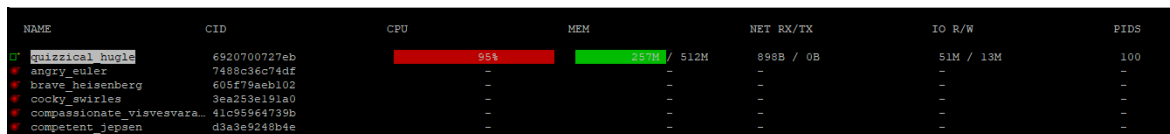
7.1 ctop^[32]

Se trata de una herramienta que permite obtener un resumen en tiempo real de las métricas de múltiples contenedores. ctop está preparado para soportar contenedores Docker y runC.

Para instalarlo se debe ejecutar:

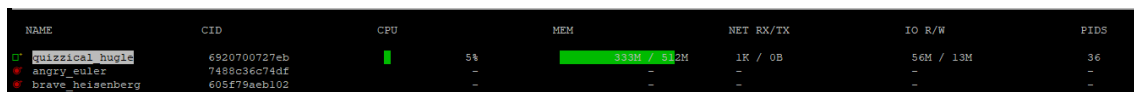
```
sudo wget
https://github.com/bcicen/ctop/releases/download/v0.7.1/ctop-
0.7.1-linux-amd64 -O /usr/local/bin/ctop
sudo chmod +x /usr/local/bin/ctop
```

Con nuestro contenedor ejecutándose probamos la herramienta ctop. Para ello solo se necesita ejecutar el comando «ctop»



NAME	CID	CPU	MEM	NET RX/TX	IO R/W	PIDS
quizzical_hugle	6920700727eb	95%	257M / 512M	898B / 0B	51M / 13M	100
angry_euler	7488c36c74df	-	-	-	-	-
brave_heisenberg	605f79aeb102	-	-	-	-	-
cocky_swirles	3ea253e191a0	-	-	-	-	-
compassionate_vivesvara...	41c959e4739b	-	-	-	-	-
competent_jepBen	d3a3e249b4e	-	-	-	-	-

Se puede observar que mientras se está desplegando la aplicación el uso de la CPU es elevado. Una vez desplegada y sin utilizar el uso de CPU es 0 o mínimo:

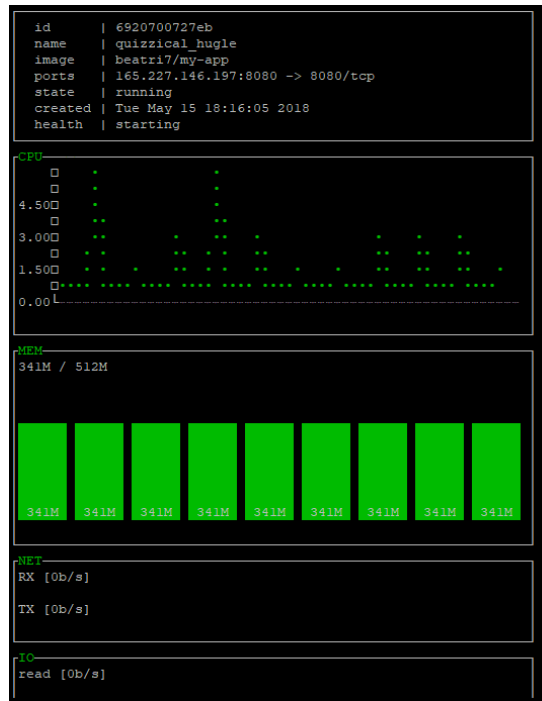


NAME	CID	CPU	MEM	NET RX/TX	IO R/W	PIDS
quizzical_hugle	6920700727eb	5%	333M / 512M	1K / 0B	56M / 13M	36
angry_euler	7488c36c74df	-	-	-	-	-
brave_heisenberg	605f79aeb102	-	-	-	-	-
cocky_swirles	3ea253e191a0	-	-	-	-	-

Otros parámetros que permite a la hora de ejecutar son:

- **-a** : muestra solo los contenedores activos.
- **-f «string»** : establece un filtro inicial.
- **-h** : muestra un dialogo de ayuda .
- **-i** : invierte los colores por defecto.
- **-r** : invierte el orden de los contenedores.
- **-s** : selecciona el campo de ordenación inicial.
- **-scale-cpu** : muestra la cpu como un porcentaje del total del sistema.
- **-v** : muestra la versión del programa y sale.

Además, permite inspeccionar un contenedor específico lo que es más útil en nuestro caso que solo tenemos uno. Debemos situarnos en el contenedor que deseamos inspeccionar y pulsar la tecla «o» o mediante la tecla «intro» seleccionar «single view».



Opciones de la aplicación:

- **«intro»** : abre el menú.
- **a** : alterna entre todos los contenedores y los que se están ejecutando.
- **f** : filtros (pulsar «esc» para salir).
- **H** : oculta o muestra la cabecera.
- **h** : abre el dialogo de ayuda.
- **s** : selecciona el campo de ordenación.
- **r** : invierte el orden de los contenedores.
- **o** : abre la vista de un único contenedor.
- **l** : ver el log de los contenedores.
- **S** : guardar la configuración actual en un fichero.
- **q** : parar ctop.

7.2 Sysdig Falco^[33]

Sysdig Falco es un monitor de actividades conductual diseñado para detectar las actividades anómalas de las aplicaciones. Funciona con la infraestructura de captura de llamadas del sistema de Sysdig. Permite monitorizar y detectar de forma continua contenedores, aplicaciones, hosts, actividad de red, etc. utilizando una única base de datos y un conjunto de reglas.

Falco puede detectar y alertar de cualquier comportamiento que involucre la realización de llamadas al sistema Linux. Gracias a la decodificación central de Sysdig y la funcionalidad de seguimiento de estados, las alertas de falco pueden ser disparadas por llamadas específicas del sistema, sus argumentos y por propiedades del proceso de llamada. Por ejemplo, se pueden detectar fácilmente eventos como:

- La ejecución de una terminal dentro de un contenedor.
- La generación de un proceso secundario inesperado por un proceso del servidor.
- Lectura inesperada de un fichero sensible.
- Escritura de un fichero que no es un dispositivo en /dev.
- Conexión saliente de red de un sistema binario estándar (como ls)

Se configura mediante un fichero de reglas que definen los comportamientos y eventos que debe vigilar, y un fichero general de configuración. Las reglas se expresan en un lenguaje de alto nivel entendible por el usuario. Por defecto contiene el fichero «/etc/falco/falco_rules.yaml» que puede ser modificado.

Una vez desplegado, falco utiliza el módulo del kernel Sysdig y las librerías de espacio de usuario para detectar los eventos que coincidan con alguna de las condiciones definidas en el fichero de reglas. Si se encuentra un evento coincidente se emite una notificación en la salida configurada.

Las salidas permitidas pueden ser:

- Escritura en la salida de errores estándar (stderr).
- Escritura en fichero.
- Escritura en syslog.
- Pasar la salida a un programa secundario como puede ser enviar cada notificación por email.

Para instalar falco se deben ejecutar las siguientes instrucciones:

- Confiar en la clave Draios GPG, configurar el repositorio apt y actualizar la lista de paquetes

```
curl -s https://s3.amazonaws.com/download.draios.com/DRAIOS-GPG-KEY.public | apt-key add -
curl -s -o /etc/apt/sources.list.d/draios.list
https://s3.amazonaws.com/download.draios.com/stable/deb/draios.list
apt-get update
```

- Instalar los fuentes del kernel.

```
| apt-get install linux-headers-$(uname -r)
```

- Instalar falco

```
| apt-get install falco
```

También se puede hacer uso del script que proveen en la web de la siguiente forma:

```
curl -s
https://s3.amazonaws.com/download.draios.com/stable/install-falco
| sudo bash
```

Falco se puede ejecutar como un servicio:

```
| service falco start
```

Los logs se pueden ver en «/var/log/syslog».

```
May 17 19:06:41 debian-docker systemd[1]: Stopping LSB: Falco syscall activity monitoring agent...
May 17 19:06:42 debian-docker kernel: [445201.821623] falco_probe: deallocating consumer ffff99930585c0c0
May 17 19:06:42 debian-docker kernel: [445201.831186] falco_probe: no more consumers, stopping capture
May 17 19:06:42 debian-docker systemd[1]: Stopped LSB: Falco syscall activity monitoring agent.
May 17 19:06:45 debian-docker systemd[1]: Starting LSB: Falco syscall activity monitoring agent...
May 17 19:06:45 debian-docker falco: Falco initialized with configuration file /etc/falco/falco.yaml
May 17 19:06:45 debian-docker falco: Loading rules from file /etc/falco/falco_rules.yaml:
May 17 19:06:45 debian-docker falco[1492]: Thu May 17 19:06:45 2018: Falco initialized with configuration file /etc/falco/falco.yaml
May 17 19:06:45 debian-docker falco[1492]: Thu May 17 19:06:45 2018: Loading rules from file /etc/falco/falco_rules.yaml:
May 17 19:06:46 debian-docker falco: Loading rules from file /etc/falco/falco_rules.local.yaml:
May 17 19:06:46 debian-docker falco[1492]: Thu May 17 19:06:46 2018: Loading rules from file /etc/falco/falco_rules.local.yaml:
May 17 19:06:46 debian-docker kernel: [445206.271132] falco_probe: adding new consumer ffff99933bef4080
May 17 19:06:46 debian-docker kernel: [445206.280831] falco_probe: initializing ring buffer for CPU 0
May 17 19:06:46 debian-docker kernel: [445206.309572] falco_probe: CPU buffer initialized, size=8388608
May 17 19:06:46 debian-docker kernel: [445206.315904] falco_probe: starting capture
May 17 19:06:46 debian-docker systemd[1]: Started LSB: Falco syscall activity monitoring agent.
```

Además, se puede ejecutar manualmente de manera que vemos los mensajes de manera instantánea con el comando «falco».

```
root@debian-docker:~# falco
Thu May 17 19:07:56 2018: Falco initialized with configuration file /etc/falco/falco.yaml
Thu May 17 19:07:56 2018: Loading rules from file /etc/falco/falco_rules.yaml:
Thu May 17 19:07:57 2018: Loading rules from file /etc/falco/falco_rules.local.yaml:
```

Ejecutando nuestro contenedor no muestra ninguna alerta por lo que se ha realizado una prueba simple abriendo una terminal en el contenedor.

```
user@debian-docker:~$ docker ps
CONTAINER ID        IMAGE               COMMAND
STATUS            PORTS              CREATED          NAMES
a050c8cacd8a       beatri7/my-app    "/opt/jboss/wildfly/..." 5 minutes ago
Up 5 minutes (health: starting) 165.227.146.197:8080->8080/tcp focused_jones
user@debian-docker:~$ docker exec -it focused_jones bash
```

La salida mostrada es la siguiente:

```
root@debian-docker:~# falco
Thu May 17 19:13:14 2018: Falco initialized with configuration file /etc/falco/falco.yaml
Thu May 17 19:13:14 2018: Loading rules from file /etc/falco/falco_rules.yaml:
Thu May 17 19:13:14 2018: Loading rules from file /etc/falco/falco_rules.local.yaml:
19:14:21.844229717: Notice A shell was spawned in a container with an attached terminal (user=debian focused_jones (id=a050c8cacd8a) shell=bash parent=<NA> cmdline=bash terminal=34816)
```

Si probamos a abrir un fichero con vi en «/dev» nos muestra el siguiente mensaje:

```
root@debian-docker:~# falco
Thu May 17 19:13:14 2018: Falco initialized with configuration file /etc/falco/falco.yaml
Thu May 17 19:13:14 2018: Loading rules from file /etc/falco/falco_rules.yaml:
Thu May 17 19:13:14 2018: Loading rules from file /etc/falco/falco_rules.local.yaml:
19:14:21.844229717: Notice A shell was spawned in a container with an attached terminal (user=debian focused_jones (id=a050c8cacd8a) shell=bash parent=<NA> cmdline=bash terminal=34816)
19:17:49.187382666: Error File below / or /root opened for writing (user=root command=vi prueba parent=bash file=/root/.vim/4913 program=vi)
```

Sin embargo, por nuestra configuración de seguridad no podemos llegar a guardarlo al igual que no podemos ver archivos sensibles como «/etc/shadow».

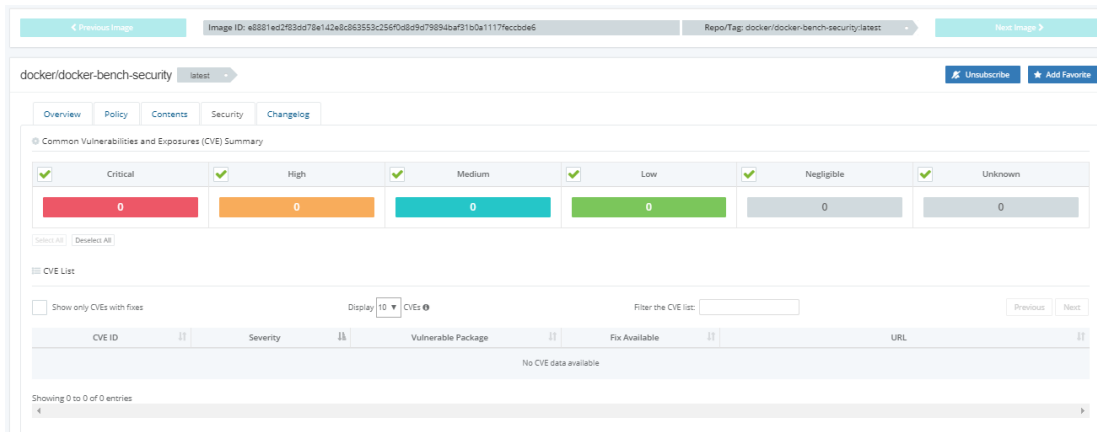
7.3 Docker Bench for Security^[34] [35]

Docker Bench for Security es un script que comprueba docenas de buenas prácticas comunes en el despliegue de contenedores Docker en producción. Realiza test automáticos que son inspirados por el CIS Docker Community Edition Benchmark que hemos visto en capítulos anteriores.

Es un proyecto open-source que está compuesto por un contenedor que escanea el host y los contenedores Docker. Es importante tener en cuenta que

dicho contenedor se ejecuta con muchos privilegios, se comparte el sistema de ficheros del host, pid y espacio de nombre de red. Se ejecutará para realizar la comprobación y se eliminará del sistema ya que una vez comprobado no es necesario siempre que no cambie nuestro sistema o contenedor o se añadan nuevas recomendaciones.

Se ha comprobado en Anchore que la imagen no tiene vulnerabilidades:



Para obtenerlo se ejecutará el siguiente comando. Hay que recordar que se debe tener activado el DOCKER_CONTENT_TRUST.

```
docker pull docker/docker-bench-security
```

Para ejecutarlo se utiliza el siguiente comando:

```
docker run -it --net host --pid host --userns host --cap-add
audit_control \
-e DOCKER_CONTENT_TRUST=$DOCKER_CONTENT_TRUST \
-v /var/lib:/var/lib \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /usr/lib/systemd:/usr/lib/systemd \
-v /etc:/etc --label docker_bench_security \
docker/docker-bench-security
```

Con el contenedor de nuestra aplicación ejecutándose obtenemos:

```
# -----
# Docker Bench for Security v1.3.4
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker containers
in production.
# Inspired by the CIS Docker Community Edition Benchmark v1.1.0.
# -----

Initializing Sun May 27 11:10:46 GMT 2018

[INFO] 1 - Host Configuration
[WARN] 1.1 - Ensure a separate partition for containers has been created
[NOTE] 1.2 - Ensure the container host has been Hardened
[INFO] 1.3 - Ensure Docker is up to date
[INFO] * Using 18.03.0, verify is it up to date as deemed necessary
[INFO] * Your operating system vendor may provide support and security main
tenance for Docker
[INFO] 1.4 - Ensure only trusted users are allowed to control Docker daemon
[INFO] * docker:x:999:user,root
```

En general nuestra aplicación pasa todas las pruebas aunque muestra algunos warnings. Algunos son consideraciones que realmente se cumplen pero tenemos que revisar nosotros, otros son pruebas que no aplican a nuestro proyecto, o como en el caso de las instrucciones HEALTHCHECK, se han solucionado a la hora de ejecutar la aplicación.

Se puede ver la salida completa en el Anexo III.

También se puede descargar y ejecutar como script. Admite las siguientes opciones:

- **-h** : Muestra mensaje de ayuda.
- **-l file**: Vuelca el log de salida en un fichero.
- **-c check** : Permite añadir una lista con las revisiones específicas (separadas por comas) que se desea realizar.
- **-e check** : Permite añadir una lista con las revisiones (separadas por comas) que se desean excluir.
- **-x exclude** : Permite añadir una lista de patrones (separador por comas) para excluir contenedores de la revisión.

7.4 Otras aplicaciones.

Se han revisado aplicaciones comerciales que aportan unas interfaces de usuario más amigables y que además unifican las diferentes fases del despliegue de una aplicación en contenedores Docker. Sin embargo, en todos los casos se trata de plataformas completas en las que la complejidad que aportan por su gran número de funcionalidades exceden los objetivos de este proyecto

Algunos ejemplos son:

- **Twistlock^[36]**: Es una solución de seguridad para contenedores que aporta seguridad en todo su ciclo de vida. Permite la administración de vulnerabilidades en imágenes, aporta defensas en tiempo de ejecución, añade seguridad a los pipelines de integración continua, etc.



Ilustración 28 Twistlock

- **Aqua^[37]**: Herramienta que proporciona controles de seguridad para aplicaciones cloud-native. Proporciona escaneo de vulnerabilidades y

asegura la integridad de las imágenes, restringe el acceso de los contenedores al host y los recursos de red, detecta problemas de configuración, exploits y ataques, etc.

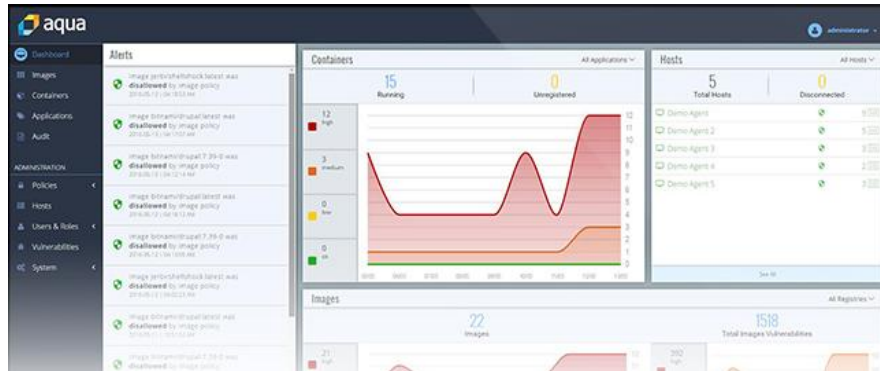


Ilustración 29 Aqua

- **StartRox^[38]**: Se trata de una plataforma de seguridad para aplicaciones en contenedores y cloud-native que previene las amenazas reduciendo los puntos de ataque con escáneres de vulnerabilidades, guías de referencia de seguridad, configuración de los datos de las aplicaciones, etc. Además, detecta los ataques supervisando las actividades de los contenedores en tiempo de ejecución, bloquea los ataques automatizando las acciones de protección, pone en cuarentena los sistemas comprometidos, etc.

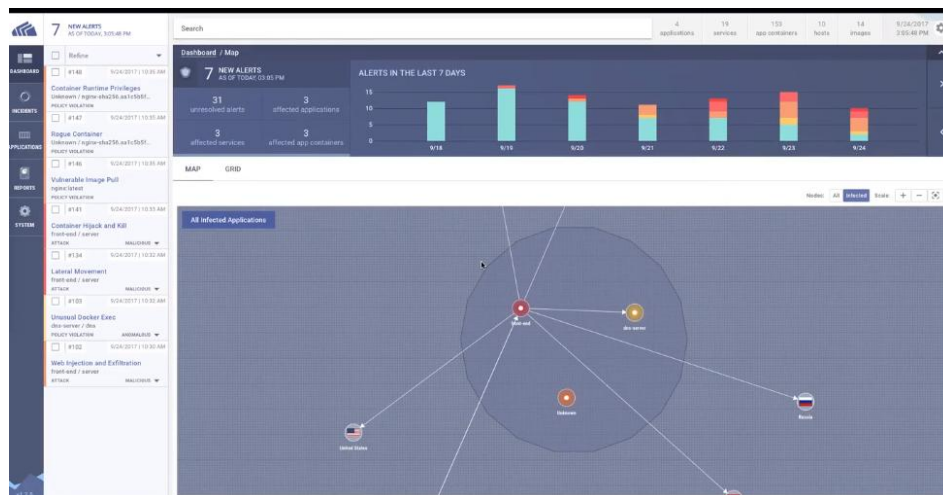


Ilustración 30 StackRox

Otra herramienta a mencionar es Dagda Docker. Se trata de un proyecto open-source que permite realizar un análisis estático de vulnerabilidades, troyanos, virus, malware y otras amenazas de las imágenes/contenedores Docker. Además, permite monitorizar el demonio Docker y los contenedores que se encuentran en ejecución para detectar actividades anómalas. Requiere de la instalación de la base de datos MongoDB, Python y la integración con Sysdig Falco. La configuración de la base de datos y la complejidad de instalación de Dagda y sus requerimientos, teniendo en cuenta por ejemplo que se debe añadir la seguridad de base de datos, ha derivado en que se descartara para este proyecto.

8. Conclusiones.

La implementación de los contenedores por parte Docker simplificándolos y consiguiendo una fácil implantación ha permitido que se extienda su uso ampliamente. Sin embargo, la información y la concienciación sobre una implementación correcta y segura del mismo no ha ido a la par.

Mediante este trabajo se ha detectado que Docker requiere de múltiples planteamientos en lo que se refiere a la seguridad que aplican a cada fase de la implantación de Docker. Sin embargo, dicha información se encuentra la mayoría muy dispersa y es escasa en comparación con la cantidad de información que se puede encontrar actualmente sobre Docker.

Por otra parte, al observar en detalle las distintas etapas que componen el despliegue de una aplicación con Docker de manera segura, se han encontrado bastantes diferencias de usabilidad y capacidad en las herramientas gratuitas disponibles. En la etapa de búsqueda y creación de nuestra imagen ha resultado sencillo encontrar sitios web y aplicaciones que las analicen y aporten un informe con vulnerabilidades y recomendaciones. Además, aportan una interfaz amigable y fácil de utilizar como es el ejemplo de Anchore. Sin embargo, las herramientas de monitorización gratuitas tienen interfaces de línea de comando o que escriben en los ficheros de logs por lo que son menos intuitivas y con diseños menos atractivos.

En cuanto a los mecanismos disponibles para añadir seguridad al kernel del host el problema que se ha encontrado es que se necesita un nivel de conocimiento del sistema bastante elevado por lo que crear perfiles específicos resulta muy costoso.

En general se ha observado que, pese a las facilidades que aporta Docker, a la hora de implantarlo de forma segura, se requiere de la colaboración entre los perfiles de desarrollo y los perfiles de sistemas para que sea lo más completo y eficaz posible.

Un avance en cuanto a unificación de soluciones y mejora de interfaces se puede encontrar en herramientas de comerciales como Twistlock, Aquasec, StarRox, etc. ya que aportan muchas funcionalidades y que cubren todas las fases de implantación los contenedores Docker. Sin embargo, resultan excesivas en componentes y complejidad para una aplicación simple como es el caso de este proyecto.

Finalmente se ha constatado la necesidad de una guía y un documento de políticas y normas a seguir, como las que se proponen en este proyecto, que recorren todas las fases de la implantación aportando herramientas, configuraciones y consejos de seguridad. Como posibles mejoras futuras a la misma se podría hacer más compleja la aplicación, incluyendo por ejemplo interacción con otros contenedores, de manera que se puedan explicar más conceptos de seguridad referentes a la interconexión. Por otro lado, se podría optar por añadir ejemplos de las aplicaciones comerciales mencionadas y de Docker Swarm y sus opciones de seguridad para la utilización de clusters.

Otros posibles proyectos derivados del análisis llevado a cabo es la creación de una herramienta que unifique las diferentes etapas con una instalación sencilla y una interfaz de usuario amigable.

9. Glosario.

- **GitHub:** Plataforma de desarrollo donde se pueden alojar, ver y gestionar el código de los proyectos que se han subido de forma pública o los proyectos propios que se han subido de forma privada.
- **Go:** Lenguaje de programación concurrente y compilado desarrollado originalmente por Google.
- **Git:** Sistema de control de versiones que permite llevar un control de cambios y coordinar el trabajo de todas las personas que modifican archivos compartidos.
- **Commit:** Comando de Git que permite confirmar los cambios realizados.
- **Git pull:** Comando de Git que permite descargar y añadir a la rama actual los cambios almacenados en el repositorio remoto.
- **Open-source:** Es un modelo de desarrollo basado en la colaboración abierta. Una de sus principales características es que el código, diseño y documentación se encuentre disponible al público.
- **PaaS:** Plataforma como servicio. Es una plataforma y entorno de desarrollo e implementación completo en la nube, con recursos que permiten desarrollar y publicar aplicaciones y servicios.
- **Hypervisor:** Es una plataforma que permite aplicar diversas técnicas de control de virtualización para utilizar, al mismo tiempo, diferentes sistemas operativos en una misma computadora.
- **Docker Toolbox:** Herramienta que permite utilizar Docker en los sistemas que no cumplen con los requisitos mínimos para la aplicación de Docker. Incluye el cliente de Docker para ejecutar Docker Engine, VirtualBox, Docker Machine para ejecutar los comandos desde la terminal de Windows, etc.
- **ISV:** Vendedores independientes de software. Compañías que se especializan en hacer o vender software especializado.
- **DEB:** Formato de paquetes de software de la distribución GNU/Linux Debian.
- **Hairpin Nat:** Comunicación entre dos host que permite acceder a tu dirección pública de internet (WAN) desde dentro de tu propia red (LAN).
- **WAR:** «Web Application Archive». Es un archivo de aplicación web JAR que contiene todas las librerías, páginas web, clases Java, etc. que constituyen la aplicación web.
- **Aplicaciones cloud-native^[39]:** Son una combinación de patrones de software ya existentes y nuevos. Los ya existentes que utiliza son de automatización, integración de API y arquitecturas orientadas a servicios. Los nuevos se componen por arquitecturas de microservicios, contenedores y gestión y orquestación distribuida.

10. Bibliografía.

1. **Web:** https://en.wikipedia.org/wiki/Agile_software_development
2. **Web:** https://es.wikipedia.org/wiki/Integraci%C3%B3n_continua
3. **Web:** https://es.wikipedia.org/wiki/Entrega_continua
4. **Web:** [https://es.wikipedia.org/wiki/Docker_\(software\)](https://es.wikipedia.org/wiki/Docker_(software))
5. **Web:** <https://docs.docker.com/engine/docker-overview/>
6. **Web:**
[https://es.wikipedia.org/wiki/Go_\(lenguaje_de_programaci%C3%B3n\)](https://es.wikipedia.org/wiki/Go_(lenguaje_de_programaci%C3%B3n))
7. **Web:** <https://es.wikipedia.org/wiki/Git>
8. **Web:** <https://www.interoute.es/what-paas>
9. **Web:** <https://es.wikipedia.org/wiki/Hipervisor>
10. **Web:** <https://hub.docker.com>
11. **Web:** <https://store.docker.com>
12. **Web:** <https://geekland.eu/motivos-para-debian-en-linux/>
13. **Web:** https://en.wikipedia.org/wiki/Open-source_model
14. **Web:**
https://es.wikipedia.org/wiki/Directrices_de_software_libre_de_Debian
15. **Web:** <https://www.digitalocean.com>
16. **Web:** https://en.wikipedia.org/wiki/Security-Enhanced_Linux
17. **Web:** <https://developers.redhat.com/jboss-docker/?referrer=jbd>
18. **Web:** <https://debian-handbook.info/browse/es-ES/stable/sect.selinux.html>
19. **Web:** <https://en.wikipedia.org/wiki/AppArmor>
20. **Web:** <https://www.digitalocean.com/community/tutorials/how-to-create-an-apparmor-profile-for-nginx-on-ubuntu-14-04>
21. **Web:** <https://medium.com/information-and-technology/so-what-is-apparmor-64d7ae211ed>
22. **Web:** <https://www.tecmint.com/mandatory-access-control-with-selinux-or-apparmor-linux/2/>
23. **Web:** <https://www.la-samhna.de/library/apparmor.html>
24. **Web:** <https://es.wikipedia.org/wiki/Deb>
25. **Web:** <https://github.com/moby/moby>
26. **Web:** <https://learn.cisecurity.org/benchmarks>
27. **Web:** <https://quay.io/>
28. **Web:** <https://anchore.com/cloud/>
29. **Web:** <http://www.mkyoung.com/spring-mvc/spring-4-mvc-ajax-hello-world-example/>
30. **Web:** <https://linux-audit.com/security-best-practices-for-building-docker-images/>
31. **Web:** <https://cisofy.com/lynis/>
32. **Web:** <https://github.com/bcicen/ctop>
33. **Web:** <https://github.com/draios/falco/wiki>
34. **Web:** <https://github.com/docker/docker-bench-security>
35. **Web:** https://docs.docker.com/compliance/cis/docker_ce/
36. **Web:** <https://www.twistlock.com>
37. **Web:** <https://www.aquasec.com>
38. **Web:** <https://www.stackrox.com/>
39. **Web:** <http://container-solutions.com/what-is-cloud-native>

Anexo I

```
profile /usr/bin/docker flags=(attach_disconnected,complain) {
  # Prevent following links to these files during container
  setup.
  deny /etc/** mkl,
  deny /dev/** kl,
  deny /sys/** mkl,
  deny /proc/** mkl,
  mount -> /var/lib/docker/**,
  mount -> /,
  mount -> /proc/**,
  mount -> /sys/**,
  mount -> /run/docker/netns/**,
  mount -> /.pivot_root[0-9]*/,
  / r,
  umount,
  pivot_root,
  signal (receive) peer=usr.bin.docker,
  signal (receive) peer=unconfined,
  signal (send),
  network,
  capability,
  owner /** rw,
  /var/lib/docker/** rwl,
  /var/lib/docker/linkgraph.db k,
  /var/lib/docker/network/files/boltddb.db k,
  /var/lib/docker/network/files/local-kv.db k,
  /var/lib/docker/[0-9]*.[0-9]*/linkgraph.db k,
  # For non-root client use:
  /dev/urandom r,
  /dev/null rw,
  /dev/pts/[0-9]* rw,
  /run/docker.sock rw,
  /proc/** r,
  /proc/[0-9]*/attr/exec w,
  /sys/kernel/mm/hugepages/ r,
  /etc/localtime r,
  /etc/ld.so.cache r,
  /etc/passwd r,
  ptrace peer=usr.bin.docker,
  ptrace (read) peer=docker-default,
  deny ptrace (trace) peer=docker-default,
  deny ptrace peer=/usr/bin/docker//bin/ps,
  /usr/lib/** rm,
  /lib/** rm,
  /usr/bin/docker pix,
  /sbin/xtables-multi rCx,
  /sbin/iptables rCx,
  /sbin/modprobe rCx,
  /sbin/auplink rCx,
  /sbin/mke2fs rCx,
  /sbin/tune2fs rCx,
  /sbin/blkid rCx,
```

```

/bin/kmod rCx,
/usr/bin/xz rCx,
/bin/ps rCx,
/bin/tar rCx,
/bin/cat rCx,
/sbin/zfs rCx,
/sbin/apparmor_parser rCx,
# Transitions
change_profile -> docker-*,
change_profile -> unconfined,
profile /bin/cat flags=(attach_disconnected,complain) {
    /etc/ld.so.cache r,
    /lib/** rm,
    /dev/null rw,
    /proc r,
    /bin/cat mr,
    # For reading in 'docker stats':
    /proc/[0-9]*/net/dev r,
}
profile /bin/ps flags=(attach_disconnected,complain) {
    /etc/ld.so.cache r,
    /etc/localtime r,
    /etc/passwd r,
    /etc/nsswitch.conf r,
    /lib/** rm,
    /proc/[0-9]*/** r,
    /dev/null rw,
    /bin/ps mr,
    # We don't need ptrace so we'll deny and ignore the error.
    deny ptrace (read, trace),
    # Quiet dac_override denials
    deny capability dac_override,
    deny capability dac_read_search,
    deny capability sys_ptrace,
    /dev/tty r,
    /proc/stat r,
    /proc/cpuinfo r,
    /proc/meminfo r,
    /proc/uptime r,
    /sys/devices/system/cpu/online r,
    /proc/sys/kernel/pid_max r,
    /proc/ r,
    /proc/tty/drivers r,
}
profile /sbin/iptables flags=(attach_disconnected,complain) {
    signal (receive) peer=/usr/bin/docker,
    capability net_admin,
}
profile /sbin/auplink flags=(attach_disconnected,complain) {
    signal (receive) peer=/usr/bin/docker,
    capability sys_admin,
    capability dac_override,
    /var/lib/docker/aufs/** rw,
    /var/lib/docker/tmp/** rw,
}

```

```

# For user namespaces:
/var/lib/docker/[0-9]*.[0-9]*/** rw,
/sys/fs/aufs/** r,
/lib/** rm,
/apparmor/.null r,
/dev/null rw,
/etc/ld.so.cache r,
/sbin/auplink rm,
/proc/fs/aufs/** rw,
/proc/[0-9]*/mounts rw,
}
profile /sbin/modprobe /bin/kmod
flags=(attach_disconnected,complain) {
    signal (receive) peer=/usr/bin/docker,
    capability sys_module,
    /etc/ld.so.cache r,
    /lib/** rm,
    /dev/null rw,
    /apparmor/.null rw,
    /sbin/modprobe rm,
    /bin/kmod rm,
    /proc/cmdline r,
    /sys/module/** r,
    /etc/modprobe.d{/,**} r,
}
# xz works via pipes, so we do not need access to the
filesystem.
profile /usr/bin/xz flags=(attach_disconnected,complain) {
    signal (receive) peer=/usr/bin/docker,
    /etc/ld.so.cache r,
    /lib/** rm,
    /usr/bin/xz rm,
    deny /proc/** rw,
    deny /sys/** rw,
}
profile /sbin/xtables-multi
flags=(attach_disconnected,complain) {
    /etc/ld.so.cache r,
    /lib/** rm,
    /sbin/xtables-multi rm,
    /apparmor/.null w,
    /dev/null rw,
    /proc r,
    capability net_raw,
    capability net_admin,
    network raw,
}
profile /sbin/zfs flags=(attach_disconnected,complain) {
    file,
    capability,
}
profile /sbin/mke2fs flags=(attach_disconnected,complain) {
    /sbin/mke2fs rm,
    /lib/** rm,
}

```

```

    /apparmor/.null w,
    /etc/ld.so.cache r,
    /etc/mke2fs.conf r,
    /etc/mtab r,
    /dev/dm-* rw,
    /dev/urandom r,
    /dev/null rw,
    /proc/swaps r,
    /proc/[0-9]*/mounts r,
}
profile /sbin/tune2fs flags=(attach_disconnected,complain) {
    /sbin/tune2fs rm,
    /lib/** rm,
    /apparmor/.null w,
    /etc/blkid.conf r,
    /etc/mtab r,
    /etc/ld.so.cache r,
    /dev/null rw,
    /dev/.blkid.tab r,
    /dev/dm-* rw,
    /proc/swaps r,
    /proc/[0-9]*/mounts r,
}
profile /sbin/blkid flags=(attach_disconnected,complain) {
    /sbin/blkid rm,
    /lib/** rm,
    /apparmor/.null w,
    /etc/ld.so.cache r,
    /etc/blkid.conf r,
    /dev/null rw,
    /dev/.blkid.tab r1,
    /dev/.blkid.tab* rw1,
    /dev/dm-* r,
    /sys/devices/virtual/block/** r,
    capability mknod,
    mount -> /var/lib/docker/**,
}
profile /sbin/apparmor_parser
flags=(attach_disconnected,complain) {
    /sbin/apparmor_parser rm,
    /lib/** rm,
    /etc/ld.so.cache r,
    /etc/apparmor/** r,
    /etc/apparmor.d/** r,
    /etc/apparmor.d/cache/** w,
    /dev/null rw,
    /sys/kernel/security/apparmor/** r,
    /sys/kernel/security/apparmor/.replace w,
    /proc/[0-9]*/mounts r,
    /proc/sys/kernel/osrelease r,
    /proc r,
    capability mac_admin,
}
}

```


Anexo II

Guía de buenas prácticas al trabajar con Dockerfiles.

Recomendaciones generales:

- Los contenedores deben ser efímeros.

El contenedor que se genera de las imágenes debe ser lo más efímero posible. Esto quiere decir que el contenedor debe poderse parar y destruir de manera que se pueda volver a construir y sustituir por uno nuevo con un mínimo de configuración requerida.

- Contexto de construcción.

Cuando se utiliza el comando «docker build» se considera como contexto de construcción el directorio de trabajo actual. Sin embargo, se puede modificar con el flag «-f» especificando un directorio diferente.

Se debe tener cuidado de no incluir en el contextos ficheros innecesarios ya que estos influirán en el tamaño de la imagen y por tanto en el tiempo de construcción y transferencia de la misma.

- Utilizar .dockerignore.

Se puede utilizar el fichero .dockerignore si deseamos excluir ficheros que no son relevantes para la construcción de la imagen sin tener que reestructurar nuestro repositorio. Este fichero soporta expresiones de exclusión similares a los de .gitignore.

- Utilizar construcciones multifase.

Para Docker 17.05 o superior se puede utilizar las construcciones multifase para reducir drásticamente el tamaño de la imagen final. La fase de la construcción deben contener muchas capas ordenadas de la menos frecuentemente modificada a la que sufre más cambios, por ejemplo:

- Instalación de las herramientas necesarias para la construcción de la aplicación.
- Instalación o actualización de las librerías de dependencias.
- Generación de la aplicación.

- Evitar la instalación de paquetes innecesarios.

Se debe evitar la instalación de paquetes extras o innecesarios para reducir la complejidad, las dependencias, el tamaño de los ficheros y los tiempos de construcción.

- Cada contenedor debe tener un único propósito.

Desacoplar las aplicaciones utilizándolos por separado en múltiples contenedores hace mucho más fácil el escalado horizontal y la reutilización de los contenedores. Si los contenedores dependen unos de otros se puede utilizar «Docker container networks» para asegurar que dichos contenedores se pueden comunicar.

- Minimizar el número de capas.

Se debe tener cuidado ya que las versiones anteriores de Docker pueden crear múltiples capas. Docker ha ido introduciendo mejoras al respecto:

- A partir de Docker 1.10, solo las instrucciones «RUN», «COPY» y «ADD» crean capas. El resto de instrucciones crea imágenes temporales intermedias sin incrementar directamente el tamaño de la construcción.
- A partir de Docker 17.05, añade el soporte de las construcciones multifase, lo que permite copiar los artefactos necesarios en la imagen final. Esto permite incluir herramientas e información de debug en las fases intermedias sin incrementar el tamaño de la imagen final.

- Ordenar los argumentos multilínea.

Al utilizar argumentos multilínea, se recomienda ordenarlos por orden alfabético. Por ejemplo, en la actualización de paquetes es recomendable ordenar el nombre de los paquetes a actualizar. Esto ayuda a evitar duplicados y nos facilita la actualización de la lista de argumentos ya que son mucho más fáciles de leer y revisar.

- Construir usando la cache.

Docker utiliza imágenes cacheadas en lugar de construirlas de nuevo para cada paso de las instrucciones. Para desactivarlo se utiliza la opción «--no-cache=true». Si se utiliza hay que tener en cuenta que las reglas que utiliza para determinar si existe una imagen que puede utilizar son las siguientes:

- Empezando por la imagen padre que se encuentra actualmente en la cache, la siguiente instrucción se compara con las imágenes hijas derivadas de la imagen base para comprobar si alguna de ellas fue construida utilizando exactamente la misma instrucción. Si no se encuentra invalida la cache.
- En la mayoría de los casos es suficiente con comparar la instrucción en el Dockerfile con una de las imágenes hijas. Sin

embargo, algunas instrucciones requieren de un mayor examen y explicación.

- Para las instrucciones «ADD» y «COPY» se examina el contenido del fichero en la imagen y se calcula un checksum para cada fichero. No se tienen en cuenta los tiempo de última modificación ni acceso del fichero para el checksum. Si ha habido cualquier cambio en el fichero, como el contenido o metadatos, la cache es invalidada.
- A excepción del punto anterior, la comprobación no tiene en cuenta los ficheros dentro del contenedor para determinar si coinciden las imágenes.

Una vez se invalida la cache el resto de comandos del Dockerfile generarán una nueva imagen sin utilizar la cache.

Instrucciones Dockerfile:

- FROM

Siempre que sea posible se deben usar como base las imágenes de repositorios oficiales.

- LABEL

Añadir etiquetas para mejorar la organización las imágenes, almacenar información de licencias, ayudar en la automatización, etc. A pesar de que en las versiones anteriores a Docker 1.10 era necesario combinar las etiquetas para obtener solo una y evitar la creación de capas extras, actualmente ya no es necesario.

Para prevenir duplicados con la imágenes creadas por las organizaciones es recomendable seguir los siguientes puntos:

- Los autores de herramientas de terceros deben añadir el prefijo formado por el nombre de dominio invertido. Por ejemplo: com.example.some-label.
- No se deben utilizar nombres dominio sin el permiso del propietario.
- Los prefijos com.docker.*, io.docker.* y org.dockerproject.* están reservados para usos internos de Docker.
- Las etiquetas deben empezar y terminar con minúscula y solo deben contener caracteres alfanuméricos en minúscula, puntos «.», y guiones «-». No se pueden utilizar puntos y guiones consecutivos.

- El caracter punto separa los valores de los dominios. Las etiquetas sin espacio de nombre están reservadas para el uso en CLI. Esto permite al usuario de CLI interactuar con las etiquetas de Docker de manera más amigable.

- RUN

La instrucción «RUN» se debe dividir en múltiples líneas separadas por barras invertidas para que el Dockerfile sea más legible, entendible y mantenible.

- APT-GET

«RUN apt-get» es uno de los casos de uso más comunes. Se debe evitar el uso de «RUN apt-get upgrade» y «dist-upgrade» ya que la mayoría de los paquetes esenciales de las imágenes padres no se pueden actualizar dentro de un contenedor sin privilegios. Si encontramos paquetes obsoletos deberemos contactar con el encargado de su mantenimiento mientras que si el paquete necesita ser actualizado podemos usar «apt-get install -y *paquete*».

Se debe combinar siempre en la misma instrucción «apt-get update» con «apt-get install». Si se utiliza «apt-get update» de forma separada causa problemas de cache que hacen que la instrucción de instalación pueda fallar ya que utiliza la versión cacheada del «apt-get update» que no estaría realmente actualizada.

Se puede limpiar la cache eliminando «/var/lib/apt/lists». Esto reduce el tamaño de la imagen ya que la apt cache no se almacena en una capa.

- PIPES

Se debe tener en cuenta que el uso de tuberías permite que se cree una imagen a pesar de que el comando que proporcione la salida para la siguiente tubería falle. Si se desea que el comando falle si ocurre un error en cualquiera de las fases de la tubería se debe añadir antes «set -o pipefail &&».

- CMD

El comando «CMD» debe ejecutarse casi siempre de la forma «CMD [*ejecutable*, "param1", "param2"...]» y en la mayoría de los casos aportando un intérprete de comandos para que al ejecutar el contenedor de forma interactiva el usuario pueda utilizarlo.

- EXPOSE

Se deben exponer solo los puertos necesarios para la aplicación.

- ENV

Se debe usar «ENV» para actualizar el path en las variables de entorno para el software instalado en el contenedor de manera que se facilita la ejecución del nuevo software.

Hay que tener en cuenta que cada instrucción «ENV» crea una nueva capa intermedia por lo que si se intenta eliminar en una nueva capa seguirá persistiendo en la capa intermedia y podrá ser accesible. Si se quiere evitar dicho comportamiento se debe crear, usar y eliminar la variable en la misma instrucción «RUN»

- ADD o COPY

De manera general es preferible «COPY» a «ADD» ya que la primera es más transparente. «COPY» solo permite el copiado básico de ficheros locales a contenedores mientras que «ADD» tiene más funcionalidades como la extracción de ficheros tar. Se recomienda utilizar este último solo cuando es necesario extraer ficheros tar.

Si tenemos diferentes pasos dentro del Dockerfile que utilizan diferentes ficheros de nuestro contexto es mejor copiarlos individualmente en lugar de a la vez. Esto asegura que la cache de cada paso es invalidada solo si su fichero específico cambia.

- ENTRYPOINT

El mejor uso para la instrucción «ENTRYPOINT» es para indicar el comando principal de la imagen utilizando el comando «CMD» como flags. También permite utilizar un script en lugar de un comando.

- VOLUME

La instrucción «VOLUME» debe utilizarse para exponer el área de almacenamiento de la base de datos, el almacenamiento de configuración, los ficheros creado por el contenedor Docker o cualquier servicio variable de la imagen.

- USER

Si un servicio puede ejecutarse sin privilegios, podemos utilizar «USER» para cambiar a un usuario no root. Se debe evitar instalar o utilizar sudo ya que tiene comportamientos impredecibles que pueden causar problemas.

- WORKDIR

Para mayor claridad y estabilidad se debe usar siempre rutas absolutas para la instrucción «WORKDIR». Además, es preferible usar esta instrucción a utilizar la instrucción «RUN» con comandos del sistema que son más complicados de leer y mantener.

- ONBUILD

Las imágenes construidas desde la instrucción «ONBUILD» deben tener tag separados.

Se debe tener cuidado si se añaden las instrucciones «ADD» o «COPY» ya que la imagen construida en el «ONBUILD» fallará si el nuevo contexto de construcción no encuentra los recursos añadidos.

Anexo III

```
_[1;33m# -----
-----
# Docker Bench for Security v1.3.4
#
# Docker, Inc. (c) 2015-
#
# Checks for dozens of common best-practices around deploying Docker
containers in production.
# Inspired by the CIS Docker Community Edition Benchmark v1.1.0.
# -----
-----_[]m

Initializing Sun May 27 11:17:00 GMT 2018

_[1;34m[INFO]_[]m 1 - Host Configuration
_[1;31m[WARN]_[]m 1.1 - Ensure a separate partition for containers
has been created
_[1;33m[NOTE]_[]m 1.2 - Ensure the container host has been Hardened
_[1;34m[INFO]_[]m 1.3 - Ensure Docker is up to date
_[1;34m[INFO]_[]m * Using 18.03.0, verify is it up to date as
deemed necessary
_[1;34m[INFO]_[]m * Your operating system vendor may provide
support and security maintenance for Docker
_[1;34m[INFO]_[]m 1.4 - Ensure only trusted users are allowed to
control Docker daemon
_[1;34m[INFO]_[]m * docker:x:999:user,root
_[1;32m[PASS]_[]m 1.5 - Ensure auditing is configured for the Docker
daemon
_[1;32m[PASS]_[]m 1.6 - Ensure auditing is configured for Docker
files and directories - /var/lib/docker
_[1;32m[PASS]_[]m 1.7 - Ensure auditing is configured for Docker
files and directories - /etc/docker
_[1;34m[INFO]_[]m 1.8 - Ensure auditing is configured for Docker
files and directories - docker.service
_[1;34m[INFO]_[]m * File not found
_[1;34m[INFO]_[]m 1.9 - Ensure auditing is configured for Docker
files and directories - docker.socket
_[1;34m[INFO]_[]m * File not found
_[1;32m[PASS]_[]m 1.10 - Ensure auditing is configured for Docker
files and directories - /etc/default/docker
_[1;32m[PASS]_[]m 1.11 - Ensure auditing is configured for Docker
files and directories - /etc/docker/daemon.json
_[1;34m[INFO]_[]m 1.12 - Ensure auditing is configured for Docker
files and directories - /usr/bin/docker-containerd
_[1;34m[INFO]_[]m * File not found
_[1;34m[INFO]_[]m 1.13 - Ensure auditing is configured for Docker
files and directories - /usr/bin/docker-runc
_[1;34m[INFO]_[]m * File not found
```

```

_[1;34m[INFO]_[0m 2 - Docker daemon configuration
_[1;32m[PASS]_[0m 2.1 - Ensure network traffic is restricted between
containers on the default bridge
_[1;32m[PASS]_[0m 2.2 - Ensure the logging level is set to 'info'
_[1;32m[PASS]_[0m 2.3 - Ensure Docker is allowed to make changes to
iptables
_[1;32m[PASS]_[0m 2.4 - Ensure insecure registries are not used
_[1;32m[PASS]_[0m 2.5 - Ensure aufs storage driver is not used
_[1;34m[INFO]_[0m 2.6 - Ensure TLS authentication for Docker daemon
is configured
_[1;34m[INFO]_[0m * Docker daemon not listening on TCP
_[1;34m[INFO]_[0m 2.7 - Ensure the default ulimit is configured
appropriately
_[1;34m[INFO]_[0m * Default ulimit doesn't appear to be set
_[1;31m[WARN]_[0m 2.8 - Enable user namespace support
_[1;32m[PASS]_[0m 2.9 - Ensure the default cgroup usage has been
confirmed
_[1;32m[PASS]_[0m 2.10 - Ensure base device size is not changed until
needed
_[1;31m[WARN]_[0m 2.11 - Ensure that authorization for Docker client
commands is enabled
_[1;32m[PASS]_[0m 2.12 - Ensure centralized and remote logging is
configured
_[1;31m[WARN]_[0m 2.13 - Ensure operations on legacy registry (v1)
are Disabled
_[1;32m[PASS]_[0m 2.14 - Ensure live restore is Enabled
_[1;32m[PASS]_[0m 2.15 - Ensure Userland Proxy is Disabled
_[1;32m[PASS]_[0m 2.16 - Ensure daemon-wide custom seccomp profile is
applied, if needed
_[1;32m[PASS]_[0m 2.17 - Ensure experimental features are avoided in
production
_[1;31m[WARN]_[0m 2.18 - Ensure containers are restricted from
acquiring new privileges

_[1;34m[INFO]_[0m 3 - Docker daemon configuration files
_[1;34m[INFO]_[0m 3.1 - Ensure that docker.service file ownership is
set to root:root
_[1;34m[INFO]_[0m * File not found
_[1;34m[INFO]_[0m 3.2 - Ensure that docker.service file permissions
are set to 644 or more restrictive
_[1;34m[INFO]_[0m * File not found
_[1;34m[INFO]_[0m 3.3 - Ensure that docker.socket file ownership is
set to root:root
_[1;34m[INFO]_[0m * File not found
_[1;34m[INFO]_[0m 3.4 - Ensure that docker.socket file permissions
are set to 644 or more restrictive
_[1;34m[INFO]_[0m * File not found
_[1;32m[PASS]_[0m 3.5 - Ensure that /etc/docker directory ownership
is set to root:root
_[1;32m[PASS]_[0m 3.6 - Ensure that /etc/docker directory
permissions are set to 755 or more restrictive
_[1;34m[INFO]_[0m 3.7 - Ensure that registry certificate file
ownership is set to root:root

```



```

_[1;34m[INFO]_[]m      * Directory not found
_[1;34m[INFO]_[]m 3.8 - Ensure that registry certificate file
permissions are set to 444 or more restrictive
_[1;34m[INFO]_[]m      * Directory not found
_[1;34m[INFO]_[]m 3.9 - Ensure that TLS CA certificate file
ownership is set to root:root
_[1;34m[INFO]_[]m      * No TLS CA certificate found
_[1;34m[INFO]_[]m 3.10 - Ensure that TLS CA certificate file
permissions are set to 444 or more restrictive
_[1;34m[INFO]_[]m      * No TLS CA certificate found
_[1;34m[INFO]_[]m 3.11 - Ensure that Docker server certificate file
ownership is set to root:root
_[1;34m[INFO]_[]m      * No TLS Server certificate found
_[1;34m[INFO]_[]m 3.12 - Ensure that Docker server certificate file
permissions are set to 444 or more restrictive
_[1;34m[INFO]_[]m      * No TLS Server certificate found
_[1;34m[INFO]_[]m 3.13 - Ensure that Docker server certificate key
file ownership is set to root:root
_[1;34m[INFO]_[]m      * No TLS Key found
_[1;34m[INFO]_[]m 3.14 - Ensure that Docker server certificate key
file permissions are set to 400
_[1;34m[INFO]_[]m      * No TLS Key found
_[1;32m[PASS]_[]m 3.15 - Ensure that Docker socket file ownership is
set to root:docker
_[1;32m[PASS]_[]m 3.16 - Ensure that Docker socket file permissions
are set to 660 or more restrictive
_[1;32m[PASS]_[]m 3.17 - Ensure that daemon.json file ownership is
set to root:root
_[1;32m[PASS]_[]m 3.18 - Ensure that daemon.json file permissions are
set to 644 or more restrictive
_[1;32m[PASS]_[]m 3.19 - Ensure that /etc/default/docker file
ownership is set to root:root
_[1;32m[PASS]_[]m 3.20 - Ensure that /etc/default/docker file
permissions are set to 644 or more restrictive

_[1;34m[INFO]_[]m 4 - Container Images and Build File
_[1;32m[PASS]_[]m 4.1 - Ensure a user for the container has been
created
_[1;33m[NOTE]_[]m 4.2 - Ensure that containers use trusted base
images
_[1;33m[NOTE]_[]m 4.3 - Ensure unnecessary packages are not
installed in the container
_[1;33m[NOTE]_[]m 4.4 - Ensure images are scanned and rebuilt to
include security patches
_[1;31m[WARN]_[]m 4.5 - Ensure Content trust for Docker is Enabled
_[1;31m[WARN]_[]m 4.6 - Ensure HEALTHCHECK instructions have been
added to the container image
_[1;31m[WARN]_[]m      * No Healthcheck found: [beatr17/my-
app:latest]
_[1;31m[WARN]_[]m      * No Healthcheck found: [sysdig/falco:latest]
_[1;34m[INFO]_[]m 4.7 - Ensure update instructions are not use alone
in the Dockerfile

```

```

_[1;34m[INFO]_[0m      * Update instruction found: [beatr17/my-
app:latest]
_[1;34m[INFO]_[0m      * Update instruction found:
[sysdig/falco:latest]
_[1;33m[NOTE]_[0m 4.8  - Ensure setuid and setgid permissions are
removed in the images
_[1;34m[INFO]_[0m 4.9  - Ensure COPY is used instead of ADD in
Dockerfile
_[1;34m[INFO]_[0m      * ADD in image history: [beatr17/my-
app:latest]
_[1;34m[INFO]_[0m      * ADD in image history: [docker/docker-bench-
security:latest]
_[1;34m[INFO]_[0m      * ADD in image history: [sysdig/falco:latest]
_[1;33m[NOTE]_[0m 4.10 - Ensure secrets are not stored in Dockerfiles
_[1;33m[NOTE]_[0m 4.11 - Ensure verified packages are only Installed

_[1;34m[INFO]_[0m 5    - Container Runtime
_[1;32m[PASS]_[0m 5.1  - Ensure AppArmor Profile is Enabled
_[1;32m[PASS]_[0m 5.2  - Ensure SELinux security options are set, if
applicable
_[1;32m[PASS]_[0m 5.3  - Ensure Linux Kernel Capabilities are
restricted within containers
_[1;32m[PASS]_[0m 5.4  - Ensure privileged containers are not used
_[1;32m[PASS]_[0m 5.5  - Ensure sensitive host system directories are
not mounted on containers
_[1;32m[PASS]_[0m 5.6  - Ensure ssh is not run within containers
_[1;32m[PASS]_[0m 5.7  - Ensure privileged ports are not mapped
within containers
_[1;33m[NOTE]_[0m 5.8  - Ensure only needed ports are open on the
container
_[1;32m[PASS]_[0m 5.9  - Ensure the host's network namespace is not
shared
_[1;32m[PASS]_[0m 5.10 - Ensure memory usage for container is limited
_[1;32m[PASS]_[0m 5.11 - Ensure CPU priority is set appropriately on
the container
_[1;32m[PASS]_[0m 5.12 - Ensure the container's root filesystem is
mounted as read only
_[1;32m[PASS]_[0m 5.13 - Ensure incoming container traffic is binded
to a specific host interface
_[1;32m[PASS]_[0m 5.14 - Ensure 'on-failure' container restart policy
is set to '5'
_[1;32m[PASS]_[0m 5.15 - Ensure the host's process namespace is not
shared
_[1;32m[PASS]_[0m 5.16 - Ensure the host's IPC namespace is not
shared
_[1;32m[PASS]_[0m 5.17 - Ensure host devices are not directly exposed
to containers
_[1;32m[PASS]_[0m 5.18 - Ensure the default ulimit is overwritten at
runtime, only if needed
_[1;32m[PASS]_[0m 5.19 - Ensure mount propagation mode is not set to
shared
_[1;32m[PASS]_[0m 5.20 - Ensure the host's UTS namespace is not
shared

```

```

_[1;32m[PASS]_[] 5.21 - Ensure the default seccomp profile is not
Disabled
_[1;33m[NOTE]_[] 5.22 - Ensure docker exec commands are not used
with privileged option
_[1;33m[NOTE]_[] 5.23 - Ensure docker exec commands are not used
with user option
_[1;32m[PASS]_[] 5.24 - Ensure cgroup usage is confirmed
_[1;32m[PASS]_[] 5.25 - Ensure the container is restricted from
acquiring additional privileges
_[1;32m[PASS]_[] 5.26 - Ensure container health is checked at
runtime
_[1;34m[INFO]_[] 5.27 - Ensure docker commands always get the latest
version of the image
_[1;32m[PASS]_[] 5.28 - Ensure PIDs cgroup limit is used
_[1;34m[INFO]_[] 5.29 - Ensure Docker's default bridge docker0 is
not used
_[1;34m[INFO]_[] * Container in docker0 network:
flamboyant_leavitt
_[1;32m[PASS]_[] 5.30 - Ensure the host's user namespaces is not
shared
_[1;32m[PASS]_[] 5.31 - Ensure the Docker socket is not mounted
inside any containers

_[1;34m[INFO]_[] 6 - Docker Security Operations
_[1;34m[INFO]_[] 6.1 - Avoid image sprawl
_[1;34m[INFO]_[] * There are currently: 4 images
_[1;34m[INFO]_[] 6.2 - Avoid container sprawl
_[1;34m[INFO]_[] * There are currently a total of 8 containers,
with 2 of them currently running

_[1;34m[INFO]_[] 7 - Docker Swarm Configuration
_[1;32m[PASS]_[] 7.1 - Ensure swarm mode is not Enabled, if not
needed
_[1;32m[PASS]_[] 7.2 - Ensure the minimum number of manager nodes
have been created in a swarm (Swarm mode not enabled)
_[1;32m[PASS]_[] 7.3 - Ensure swarm services are binded to a
specific host interface (Swarm mode not enabled)
_[1;32m[PASS]_[] 7.5 - Ensure Docker's secret management commands
are used for managing secrets in a Swarm cluster (Swarm mode not
enabled)
_[1;32m[PASS]_[] 7.6 - Ensure swarm manager is run in auto-lock
mode (Swarm mode not enabled)
_[1;32m[PASS]_[] 7.7 - Ensure swarm manager auto-lock key is
rotated periodically (Swarm mode not enabled)
_[1;32m[PASS]_[] 7.8 - Ensure node certificates are rotated as
appropriate (Swarm mode not enabled)
_[1;32m[PASS]_[] 7.9 - Ensure CA certificates are rotated as
appropriate (Swarm mode not enabled)
_[1;32m[PASS]_[] 7.10 - Ensure management plane traffic has been
separated from data plane traffic (Swarm mode not enabled)

_[1;34m[INFO]_[] Checks: 104

```

| _[1;34m[INFO]_[0m Score: 52