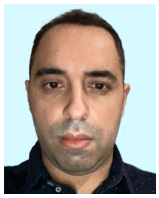
Seguridad en entornos IaaS públicos

PID_00286169

Miguel Ángel Flores Terrón
Jordi Guijarro Olivares

Tiempo mínimo de dedicación recomendado: 3 horas

Universitat Oberta de Catalunya



Miguel Ángel Flores Terrón

Ingeniero en Informática por la Universitat Oberta de Catalunya (UOC), ingeniero técnico en Informática de Sistemas por la Facultad de Informática de Barcelona (FIB-UPC) y máster en Seguridad de la TIC por la UOC. Desde 2020, profesor colaborador en la UOC como tutor de trabajos final de máster en el ámbito de la ciberseguridad. Actualmente, administrador de sistemas en la unidad de Operaciones y Cibeserguridad del Consorci de Serveis Universitaris de Catalunya (CSUC).

Jordi Guijarro Olivares

Ingeniero en Informática por la Universitat Oberta de Catalunya (UOC) y máster en Gestión de las TIC por la Universidad Ramon Llull (URL). Director de Innovación en Ciberseguridad en el Centro de Investigación en Internet (i2CAT, <http://www.i2cat.net>). Experto en *cloud computing* y ciberseguridad, participa en proyectos de investigación e innovación del programa Horizon 2020 de la Comisión Europea.

El encargo y la creación de este recurso de aprendizaje UOC han sido coordinados por el profesor: Josep Jorba Esteve

Primera edición: febrero 2022

Av. Tibidabo, 39-43, 08035 Barcelona
Autoría: Miguel Ángel Flores Terrón, Jordi Guijarro Olivares
Producción: FUOC

Índice

Introducción

La seguridad de la infraestructura es clave para operar de forma segura en entornos de computación en la nube. Como ya se ha podido ver en módulos anteriores, la «infraestructura» es la unión de servidores y redes sobre la que se construyen los servicios que permitirán la ejecución de las diferentes cargas de trabajo (*workload*).

La computación en la nube se basa fundamentalmente en la agrupación de recursos, y la virtualización es la tecnología utilizada para convertir la infraestructura fija en estos recursos agrupados. La virtualización proporciona la abstracción necesaria para los grupos de recursos, que luego se gestionan mediante la orquestación.

En este camino, es fundamental comprender los impactos de la virtualización en la seguridad para diseñar e implementar correctamente la seguridad en la nube. Los activos virtuales aprovisionados desde un grupo de recursos pueden parecerse a los activos físicos que reemplazan, pero esa apariencia es realmente una herramienta para ayudarnos a comprender y administrar mejor esos recursos.

En este módulo veremos prácticas de seguridad aplicadas a entornos IaaS públicos. En concreto, veremos el caso de un gran *hyperscaler* como Amazon Web Services y otro más especializado como el de Digital Ocean.

Para finalizar, este módulo también incluye aspectos de seguridad en entornos PaaS basados en contenedores Docker, donde veremos desde la seguridad del *host* hasta módulos de computación segura. Al final de este apartado, se proporciona al alumno una lista de comprobación de prácticas de seguridad en Docker.

Objetivos

Con los materiales de este módulo didáctico, se busca que el estudiantado aprenda y desarrolle los conocimientos y habilidades siguientes:

1. Detectar y estudiar las amenazas de dos de los principales *cloud* públicos: AWS y Digital Ocean.

2. Evaluar las responsabilidades de cada una de las partes en el modelo de *cloud* público.

3. Trabajar los procesos de gestión de la seguridad a través de diferentes casos de uso.

4. Establecer mecanismos de mitigación frente a los posibles incidentes de seguridad.

1. Amazon Web Services

En el caso de Amazon Web Services (AWS), veremos brevemente los aspectos más importantes relativos a la seguridad que se deben tener en cuenta, para centrarnos en detalle en cómo mitigar los ataques de tipo DDoS combinando diferentes servicios del proveedor de IaaS.

1.1. Modelo de responsabilidad compartida

Los asuntos relacionados con la seguridad y la conformidad son una responsabilidad compartida entre AWS y la organización.

El modelo de responsabilidad compartida permite aliviar la carga operativa de la organización, ya que AWS opera, administra y controla los componentes del sistema operativo *host* y la capa de virtualización hasta la seguridad física de las instalaciones en las que funcionan los servicios.

La organización asume la responsabilidad y la administración del sistema operativo invitado (incluidas las actualizaciones y los parches de seguridad), de cualquier otro software de aplicaciones asociado y de la configuración del cortafuegos del grupo de seguridad que ofrece AWS. Cada organización debe pensar detenidamente en los servicios que elige, ya que las responsabilidades varían en función de los servicios que utilicen, de la integración de estos en su entorno de TI, y de la legislación y los reglamentos correspondientes. La naturaleza de esta responsabilidad compartida también ofrece la flexibilidad y el control por parte de la organización que permite concretar la implementación. Como se describe a continuación, la diferenciación de responsabilidades se conoce normalmente como **seguridad «de» la nube** y **seguridad «en» la nube** (figura 1):

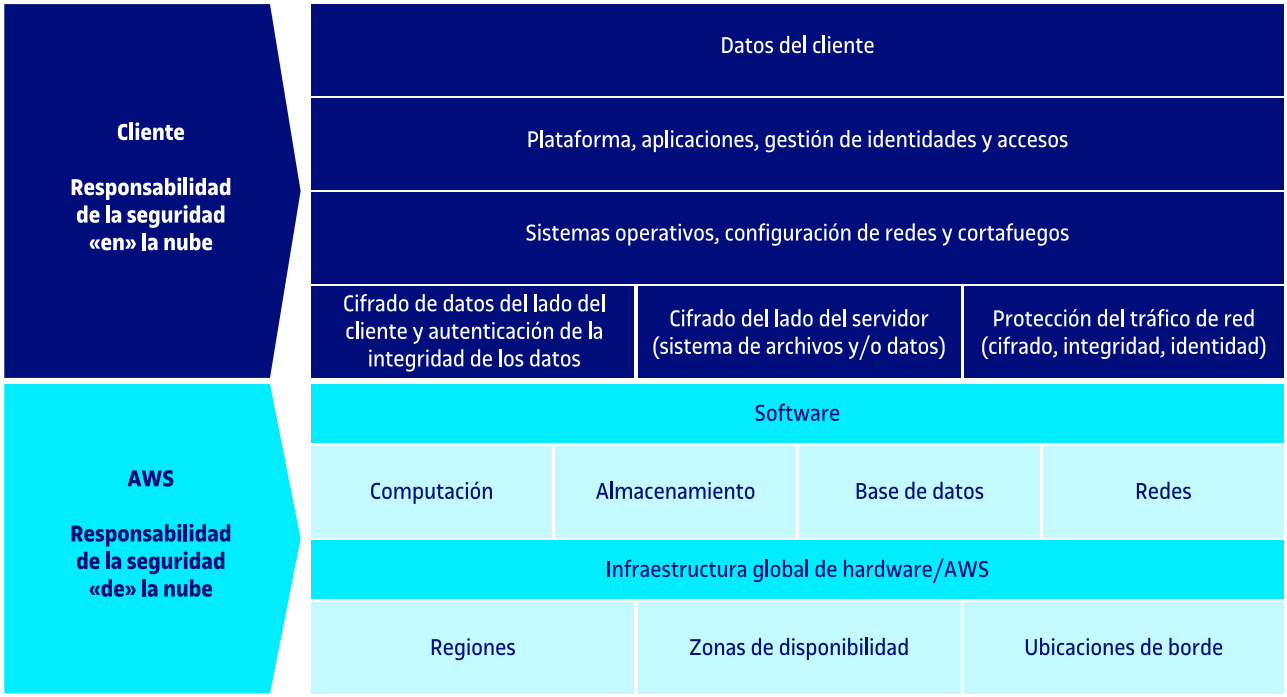
- **Responsabilidad de AWS en relación con la «seguridad de la nube»**: AWS es responsable de proteger la infraestructura que ejecuta todos los servicios provistos en la nube de AWS. Esta infraestructura está conformada por el hardware, el software, las redes y las instalaciones que ejecutan los servicios de la nube de AWS.

- **Responsabilidad de la organización en relación con la «seguridad en la nube»**: la responsabilidad de la organización estará determinada por los servicios de la nube de AWS que la organización utilice. Esto determina el alcance del trabajo de configuración a cargo de la organización como parte de sus responsabilidades de seguridad.

Amazon EC2, Amazon S3 y Amazon DynamoDB

Un servicio como Amazon Elastic Compute Cloud (Amazon EC2) se clasifica como infraestructura como servicio (IaaS) y, como tal, requiere que el cliente realice todas las tareas de administración y configuración de seguridad necesarias. En el caso de los servicios extraídos, como Amazon S3 y Amazon DynamoDB, AWS maneja la capa de infraestructura, el sistema operativo y las plataformas, mientras que la organización accede a los puntos de enlace para recuperar y almacenar los datos. La organización es la responsable de administrar sus datos (incluidas las opciones de cifrado), clasificar sus recursos y utilizar las herramientas de IAM para solicitar los permisos correspondientes.

Figura 1. Modelo de responsabilidad compartida de AWS



Fuente: elaboración propia a partir de <https://aws.amazon.com/es/compliance/shared-responsibility-model/>

1.2. Gestión de acceso de identidad, S3 y políticas de seguridad

Amazon ofrece herramientas para cuidar de la seguridad. La autenticación a través de múltiples factores en las cuentas de AWS es muy recomendable mediante el servicio **AWS Multi-Factor Authentication**.

Se pueden crear cuentas separadas para los usuarios de Amazon con el fin de no compartir contraseñas. Hay que asegurarse también de que no se utilicen cuentas *root* y que las cuentas solo tengan los privilegios necesarios, por ejemplo, las cuentas de desarrollador. **AWS Identity and Access Management (IAM)** permite definir cuentas de usuarios individuales con permisos en los recursos de AWS.

Es recomendable utilizar las herramientas de Amazon para administrar claves privadas y almacenarlas posteriormente de forma segura. Por último, hay que supervisar las actividades sospechosas, como las llamadas inesperadas a la API e inicios de sesión inusuales.

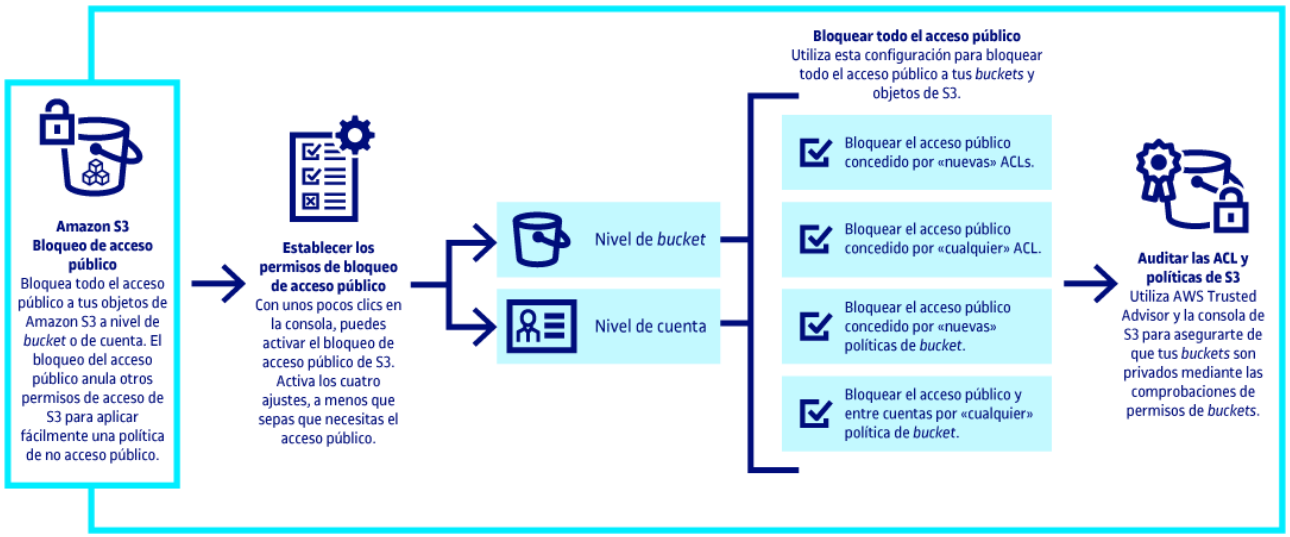
1.2.1. Amazon S3

De forma predeterminada, Amazon S3 ofrece acceso a los recursos únicamente a los usuarios que han creado el objeto. Para conceder acceso a otros usuarios, debemos utilizar una de las siguientes características de administración de acceso o una combinación de ellas (figura 2):

- **AWS Identity and Access Management (IAM)** para crear usuarios y administrar su correspondiente acceso.
- **Listas de control de acceso (ACL)** para conceder acceso a objetos individuales a los usuarios autorizados.
- **Políticas de *bucket*** a fin de configurar permisos para todos los objetos de un único *bucket* de S3.
- Autenticación por cadena de consulta para conceder acceso a otros usuarios por tiempo limitado con **direcciones URL temporales**.

Amazon S3 también admite registros de auditoría, que enumeran las solicitudes realizadas a los recursos de S3 para obtener total visibilidad de quién obtiene acceso a los distintos datos.

Figura 2. Diagrama para bloquear todo el acceso público a Amazon S3



Fuente: elaboración propia

1.2.2. Amazon Cognito

Amazon Cognito permite incorporar el registro, el inicio de sesión y el control de acceso de usuarios a las aplicaciones web y móviles. Amazon Cognito puede ser escalado para millones de usuarios y admite el inicio de sesión mediante

proveedores de identidad social, como Apple, Facebook, Google y Amazon, así como con proveedores de identidad empresarial a través de SAML 2.0 y OpenID Connect.

1.3. Registro (*logs*) y monitorización

AWS CloudTrail monitoriza y registra la actividad de una determinada cuenta en toda la infraestructura de AWS, lo que le permite controlar las acciones de almacenamiento, análisis y reparación. Este servicio puede sernos de mucha utilidad para los casos de uso siguientes:

- **Controlar la actividad**: monitorizar, almacenar y validar los eventos de actividad para determinar su autenticidad. También nos permite generar de forma sencilla los informes de auditoría que exigen las políticas internas y las normativas externas.

- **Identificar incidentes relativos a la seguridad**: detectar el acceso no autorizado mediante la información correspondiente a «quién», «qué» y «cuándo» que aparece en los eventos de CloudTrail. Podremos responder con alertas de EventBridge basadas en reglas y automatizar flujos de trabajo.

- **Solucionar problemas operativos**: monitorizar continuamente el historial de uso de las API mediante modelos de aprendizaje automático (ML, *machine learning*) para detectar cualquier actividad inusual en las cuentas de AWS y determinar su causa.

Amazon CloudWatch es un servicio de monitorización y observación pensado para DevOps, desarrolladores, ingenieros de fiabilidad de sitio (SRE) y administradores de TI. CloudWatch ofrece datos e información procesable para monitorizar las aplicaciones, responder a cambios de rendimiento que afectan a todo el sistema, optimizar el uso de recursos y lograr una vista unificada del estado de las operaciones. Este servicio puede resultar interesante para los casos de uso siguientes:

- **Monitorización y solución de problemas de infraestructuras**: monitorizar métricas y registros clave, visualizar la pila de infraestructuras y aplicaciones, crear alarmas, y correlacionar métricas y registros para comprender y resolver la causa que está en la raíz de los problemas de rendimiento en los recursos de AWS.

- **Mejora del tiempo medio de resolución**: ayuda a correlacionar, visualizar y analizar métricas y registros para poder actuar con rapidez a fin de solucionar problemas y combinarlos con datos de rastreo de AWS X-Ray para lograr una visibilidad completa. También se pueden analizar las soli-

citudes de los usuarios para contribuir a acelerar la solución de problemas y la depuración, además de reducir el tiempo medio de resolución (MTTR).

- **Optimización de recursos proactiva**: las alarmas de CloudWatch supervisan los valores de las métricas con respecto a umbrales especificados por el usuario o creados por CloudWatch mediante modelos de aprendizaje automático a fin de detectar comportamientos anómalos. Si se activa una alarma, CloudWatch puede llevar a cabo una acción automáticamente para habilitar Auto Scaling de Amazon EC2 o detener una instancia, por ejemplo, lo cual permite automatizar la capacidad y la planificación de recursos.

- **Monitorización de aplicaciones**: monitorizar las aplicaciones que se ejecutan en AWS (en Amazon EC2, en contenedores y sin servidor) o localmente. CloudWatch recopila datos en todas las capas de la pila de rendimiento, incluidas métricas y registros, en paneles automáticos.

- **Análisis de registros**: explorar, analizar y visualizar los registros para abordar problemas operativos y mejorar el rendimiento de las aplicaciones.

AWS GuardDuty

Este servicio permite a los usuarios supervisar la cuenta de AWS en busca de comportamientos inusuales e inesperados mediante el análisis de los registros de eventos de AWS CloudTrail, los registros de flujo de VPC y los registros de DNS. A continuación, utiliza los datos de los registros y los evalúa frente a múltiples fuentes de seguridad y detección de amenazas, buscando anomalías y fuentes maliciosas conocidas, como direcciones IP y URL.

1.4. Seguridad de la infraestructura

Amazon Web Services proporciona varias capacidades y servicios de seguridad para mejorar la privacidad y controlar el acceso de redes. Entre ellos, se incluyen los cortafuegos de red integrados en Amazon VPC, y las capacidades de *firewall* para aplicaciones web existentes en AWS WAF permiten crear redes privadas y controlar el acceso a las instancias y las aplicaciones. También se incluye el cifrado en tránsito con TLS en todos los servicios. Y, por último, existen las opciones de conectividad que permiten conexiones privadas o dedicadas desde la oficina o entorno *on-premise*.

Como podemos ver, AWS en sí ya implementa diferentes medidas de seguridad en cuanto a lo que se refiere a seguridad de la infraestructura, lo que permite a los usuarios de AWS centrarse en otros aspectos de la seguridad.

1.4.1. Cifrado de datos

AWS ofrece la posibilidad de añadir una capa de seguridad adicional a los datos en reposo en la nube, lo que proporciona características de cifrado a los servicios de base de datos y almacenamiento, como EBS, S3, Glacier, Oracle RDS, SQL Server RDS y Redshift.

También podemos utilizar servicios de administración de claves, como **AWS Key Management Service**, que permiten elegir si AWS administra las claves de cifrado o si mantenemos el control sobre las claves.

AWS también proporciona diversas API para que pueda integrar el cifrado y la protección de los datos con cualquiera de los servicios desarrollados en un entorno de AWS.

1.4.2. Trazabilidad de cambios

Mediante el uso de herramientas de administración de inventario y configuración, como AWS Config, que identifica los diferentes recursos de AWS, se puede realizar el seguimiento de los cambios en dichos recursos a lo largo del tiempo. Esto ayuda a construir una trazabilidad sobre los cambios de configuración, de forma que incluso es posible volver atrás en el tiempo hacia un escenario más seguro.

1.4.3. Escaneo de vulnerabilidades

AWS ofrece un servicio de evaluación de la seguridad, Amazon Inspector, que evalúa las aplicaciones automáticamente para detectar vulnerabilidades o desviaciones de las prácticas recomendadas. Dicho escaneo incluye las redes, el sistema operativo y el almacenamiento.

1.4.4. Web Application Firewall (WAF)

AWS WAF es un cortafuegos para aplicaciones web que ayuda a protegerlas de los ataques más comunes que podrían afectar a la disponibilidad de la aplicación, comprometer la seguridad o consumir recursos excesivos. AWS WAF permite controlar el tráfico que se admite o se bloquea en las aplicaciones web mediante la definición de **reglas de seguridad personalizables**. Se encarga de bloquear los ataques que ocurran sobre la capa de aplicación, y además puede ser utilizado para bloquear las peticiones teniendo en cuenta la cabecera del User-Agent.

En el presente, cada vez son más las aplicaciones que son hospedadas en la nube y renuncian a los tradicionales servidores. Esto se debe principalmente a la gran **escalabilidad** que ofrecen plataformas como AWS, que disponen de un gran abanico de servicios que permiten la escalabilidad de la aplicación. Algunas de las capacidades que ofrece el WAF son las siguientes:

- Control de acceso integrado basado en roles en todos los servicios de AWS (IAM).
- Seguridad y registro de auditoría de la API en todos los servicios (CloudTrail).
- Integración con otros servicios de AWS.
- Seguridad integrada con la forma de desarrollar aplicaciones.

1.5. Caso de uso: protección contra ataques DDoS

En el caso de un ataque DDoS (*Distributed Denial of Service*), el atacante utiliza múltiples fuentes que pueden haber sido comprometidas o controladas por un colaborador para orquestar un ataque contra un objetivo. En un ataque DDoS, cada uno de los colaboradores o *hosts* comprometidos participa en la acción, lo que genera una inundación de paquetes o peticiones con el fin de que un servicio o recurso sea inaccesible a los usuarios legítimos.

1) Defensa de la capa de infraestructura

En un entorno de centro de datos tradicional, se puede proteger la capa de infraestructura frente a ataques DDoS utilizando técnicas como el sobreprovisionamiento, desplegando sistemas de mitigación de DDoS o la limpieza de tráfico con la ayuda de servicios de mitigación de DDoS. En AWS existen opciones que permiten diseñar una aplicación para poder escalar y absorber mayores volúmenes de tráfico sin recurrir a una complejidad innecesaria.

a) Tamaño de instancia. La capacidad de cómputos es un recurso escalable en Amazon EC2. Se puede escalar horizontalmente añadiendo instancias a la aplicación, según sea necesario, o bien verticalmente usando instancias más grandes. Algunos tipos de instancia constan de interfaces de red de 10 gigabits que mejoran la capacidad para manejar grandes volúmenes de tráfico. Esto ayuda a evitar la congestión de la interfaz para cualquier tráfico que llegue a la instancia de Amazon EC2. Estas instancias tienen mayor rendimiento de E/S, lo que implica una menor utilización de la CPU.

b) Elección de región. Muchos servicios de AWS, como Amazon EC2, están disponibles en múltiples ubicaciones en todo el mundo. Estas áreas geográficamente separadas se denominan «regiones AWS». En el momento de diseñar la arquitectura de la aplicación, se pueden elegir una o más regiones dependiendo de los requisitos. En cada región, AWS proporciona acceso a un conjunto único de conexiones a Internet y relaciones de *peering* que permiten una latencia adecuada a los usuarios finales que se encuentran situados cerca de la

región. También es importante considerar la elección de la región en términos de flexibilidad DDoS. Muchas regiones están más cerca de los grandes intercambios de Internet. Muchos ataques DDoS se orquestan internacionalmente, por lo que es útil estar cerca de los puntos de intercambio, ya que esto ayuda a los usuarios finales a alcanzar la aplicación.

c) Balanceo de carga. Los ataques DDoS más grandes pueden superar el tamaño de una única instancia de Amazon EC2. Al mitigar estos ataques, hay que considerar opciones para balancear la carga. Con Elastic Load Balancing (ELB) se puede reducir el riesgo de sobrecarga de la aplicación mediante la distribución de tráfico entre muchas instancias de *backend*. ELB puede hacer el balanceo automáticamente, lo que le permite gestionar grandes volúmenes de tráfico, como los volúmenes resultantes de los ataques DDoS.

Excepciones de ELB

ELB solo acepta conexiones TCP bien formadas. Esto significa que muchos de los ataques DDoS, como inundaciones SYN o ataques de reflexión UDP, no serán aceptados por ELB. Cuando detecta estos tipos de ataques, escala automáticamente para absorber el tráfico adicional.

d) Entrega a escala mediante AWS Edge Locations. El acceso a conexiones de Internet muy escaladas y diversas aumenta la capacidad para optimizar la latencia y el rendimiento a los usuarios finales, absorber ataques DDoS y aislar las fallas mientras se minimiza el impacto de la disponibilidad. AWS Edge constituye una capa adicional de infraestructura de red que proporciona beneficios a las aplicaciones web que utilizan también Amazon CloudFront y Amazon Route 53.

Con estos servicios, el contenido se sirve y las consultas DNS se resuelven desde ubicaciones que a menudo están más cerca de sus usuarios finales. **Amazon CloudFront** es un servicio de red de distribución de contenido (CDN) que se puede utilizar para entregar todo el sitio web, incluyendo el contenido estático, dinámico, de transmisión e interactivo. Las conexiones TCP persistentes y el tiempo de vida variable (TTL) se pueden utilizar para acelerar la entrega de contenido, incluso si no se puede almacenar en caché. Esto permite usar Amazon CloudFront para proteger una aplicación web, incluso si no se está sirviendo contenido estático. Amazon CloudFront solo acepta conexiones bien formadas para evitar que muchos ataques DDoS comunes, como inundaciones SYN y ataques de reflexión UDP, lleguen a origen. Los ataques DDoS están geográficamente aislados cerca de la fuente, lo que evita que el tráfico afecte a otras ubicaciones. Estas utilidades pueden mejorar considerablemente la capacidad para seguir sirviendo tráfico a los usuarios finales durante ataques DDoS mayores. Se puede utilizar Amazon CloudFront para proteger un origen en AWS o en cualquier otro lugar de Internet.

e) Resolución de nombres de dominio. Amazon Route 53 es un servicio de nombres de dominio (DNS) altamente disponible y escalable que se puede utilizar para dirigir tráfico a una aplicación web. Incluye muchas característi-

cas avanzadas, como flujo de tráfico, enrutamiento basado en latencia, Geo DNS, comprobaciones de estado y supervisión. Estas características permiten controlar cómo responde el servicio a las peticiones DNS para optimizar la latencia. Amazon Route 53 usa *shuffle sharding* y *anycast striping* para que los usuarios finales puedan acceder a la aplicación, incluso si el servicio DNS es atacado por un ataque DDoS:

- Con ***shuffle sharding***, cada servidor de nombres en su conjunto de delegaciones corresponde a un conjunto único de ubicaciones y rutas de Internet. Esto proporciona una mayor tolerancia a fallos y minimiza la superposición entre clientes. Si un servidor de nombres en el conjunto de delegaciones no está disponible, los usuarios finales pueden volver a intentarlo y recibir una respuesta de otro servidor de nombres en una ubicación diferente.

- ***Anycast striping*** se utiliza para que cada petición de DNS se sirva desde la ubicación más rápida. Esto tiene el efecto de extender la carga y reducir la latencia del DNS, lo que permite a los usuarios finales recibir una respuesta más rápida.

Además, Amazon Route 53 puede detectar anomalías en la fuente y el volumen de las consultas DNS, y priorizar las peticiones de los usuarios que se sabe que son fiables.

2) Defensa de la capa de aplicación

a) Detectar y filtrar peticiones web malintencionadas. Los cortafuegos de aplicaciones web (WAF) se utilizan para protegerlas contra ataques que intentan explotar una vulnerabilidad en la aplicación. Los ejemplos más comunes incluyen la inyección SQL o el *cross-site scripting*. También se puede utilizar un WAF para detectar y mitigar los ataques DDoS de la capa de aplicación web.

En AWS, se puede utilizar Amazon CloudFront y AWS WAF para defender una aplicación contra estos ataques. Amazon CloudFront permite almacenar en caché estática el contenido y servirlo desde AWS Edge Locations. Además, Amazon CloudFront puede cerrar automáticamente conexiones de lectura lenta o ataques de escritura lenta. Se puede utilizar la restricción geográfica de Amazon CloudFront para bloquear ataques que se originen desde ubicaciones geográficas donde no se espera servir a los usuarios finales.

Puede ser difícil identificar la firma de un ataque DDoS o identificar las direcciones IP que están participando en dicha acción. Se puede utilizar la consola AWS WAF para ver una muestra de las peticiones que Amazon CloudFront ha enviado a AWS WAF. Algunos ataques consisten en tráfico web que se disfraza para parecer que procede de un usuario final. Para mitigar este tipo de acción,

se puede utilizar una función AWS Lambda para implementar listas negras basadas en una tasa. Si un *bot* o *crawler* excede este límite, se puede utilizar AWS WAF para bloquear automáticamente futuras peticiones.

b) Escala para absorber. Otra forma de hacer frente a los ataques en la capa de aplicación es escalar. En el caso de las aplicaciones web, se puede utilizar ELB para distribuir instancias EC2 sobreprovisionadas o configuradas para escalar automáticamente con el objetivo de atender los aumentos de tráfico que pueden ser resultado de un ataque DDoS en la capa de aplicación. Las alarmas de Amazon CloudWatch se utilizan para iniciar el Auto Scaling, que escala automáticamente el número de instancias de Amazon EC2 en respuesta a eventos previamente definidos. Esto protege la disponibilidad de las aplicaciones incluso cuando se está tratando un volumen inesperado de peticiones. Mediante el uso de Amazon CloudFront o ELB, la negociación SSL es manejada por el balanceador de carga, lo que puede impedir que las instancias se vean afectadas por ataques que exploten vulnerabilidades SSL.

3) Ofuscación de los recursos AWS

Otra consideración importante al diseñar en AWS es limitar las oportunidades de que un atacante pueda aprender de la aplicación. Por ejemplo, si no se espera que un usuario final interactúe directamente con ciertos recursos, estos no deben estar accesibles desde Internet.

Del mismo modo, si no se espera que los usuarios finales o las aplicaciones externas se comuniquen con la aplicación en ciertos puertos o protocolos, ese tipo de tráfico debería no ser aceptado. Este concepto se conoce como ***reducción de la superficie de ataque***.

Para muchas aplicaciones, los recursos AWS no necesitan estar completamente expuestos a Internet.

Accesibilidad de los recursos AWS

Puede que no sea necesario que las instancias de Amazon EC2 detrás de un ELB estén accesibles al público. En este escenario, lo idóneo es que los usuarios finales solo puedan acceder al ELB en ciertos puertos TCP y que solo el ELB pueda comunicarse con las instancias de Amazon EC2. Esto puede lograrse configurando grupos de seguridad y listas de control de acceso a la red (NACL) dentro de Amazon Virtual Private Cloud (VPC). Amazon VPC permite proveer una sección lógicamente aislada de la nube de AWS donde lanzar recursos de AWS en una red virtual definida.

Los grupos de seguridad y las ACL de red son similares, ya que permiten controlar el acceso a los recursos de AWS dentro de una VPC. Los grupos de seguridad permiten controlar el tráfico entrante y saliente a nivel de instancia, y la ACL de red ofrece capacidades similares, pero a nivel de subred VPC.

a) Grupos de seguridad. Se pueden especificar grupos de seguridad al iniciar una instancia. Todo el tráfico desde Internet en un grupo de seguridad se niega implícitamente a menos que se cree una regla para permitir el tráfico.

Permisos de acceso

Si consideramos una aplicación web que consiste en un ELB y varias instancias Amazon EC2, se podría crear un grupo de seguridad para el ELB (grupo de seguridad del ELB) y uno para las instancias (grupo de seguridad del servidor de aplicaciones web). A continuación, se pueden crear permisos para permitir el tráfico desde Internet al grupo de seguridad del ELB y para permitir el tráfico desde este grupo al servidor de aplicaciones web. Como resultado, el tráfico en Internet no puede comunicarse directamente con sus instancias de Amazon EC2, lo que hace más difícil que un atacante aprenda acerca de una aplicación.

b) Listas de control de acceso a la red. Esta herramienta es útil cuando se responde a ataques DDoS porque puede permitir crear reglas para mitigar dicho ataque si se conoce la dirección IP de origen. Con las ACL de red, se pueden especificar las reglas para permitir y denegar. Esto es útil en caso de que se desee negar explícitamente ciertos tipos de tráfico a la aplicación.

Permitir y denegar accesos

Se pueden definir direcciones IP (como rangos CIDR), protocolos y puertos de destino, que se deben denegar para toda una subred. Si una aplicación utiliza solo tráfico TCP, se puede crear una regla para denegar todo el tráfico UDP o viceversa.

c) Protección basada en el origen de las peticiones. Mientras se usa Amazon CloudFront con origen dentro de una VPC, se puede utilizar una función AWS Lambda para actualizar automáticamente las reglas del grupo de seguridad asociado con la finalidad de permitir únicamente el tráfico de Amazon CloudFront. Esto obliga a las peticiones a ser procesadas por Amazon CloudFront y AWS WAF.

d) Protección de los puntos finales de una API. Normalmente, cuando tenemos la necesidad de exponer una API al público, existe el riesgo de que la interfaz API pueda ser atacada por un ataque DDoS. Amazon API Gateway es un servicio totalmente administrado que permite crear una API que actúa como una «puerta de entrada» a aplicaciones que se ejecutan en Amazon EC2, AWS Lambda o cualquier aplicación web.

Con **Amazon API Gateway** no es necesario que se ejecuten los propios servidores para la API, y también se pueden ocultar otros componentes de la aplicación al acceso público. Esto ayuda a evitar que los recursos de AWS sean atacados mediante un DDoS. Amazon API Gateway está integrado con Amazon CloudFront, lo que permite beneficiarse de la capacidad adicional de DDoS inherente a ese servicio. También se puede proteger el *backend* del exceso de tráfico configurando límites de velocidad estándar o de ráfaga para cada método en el caso de que sean API de tipo REST.

2. DigitalOcean

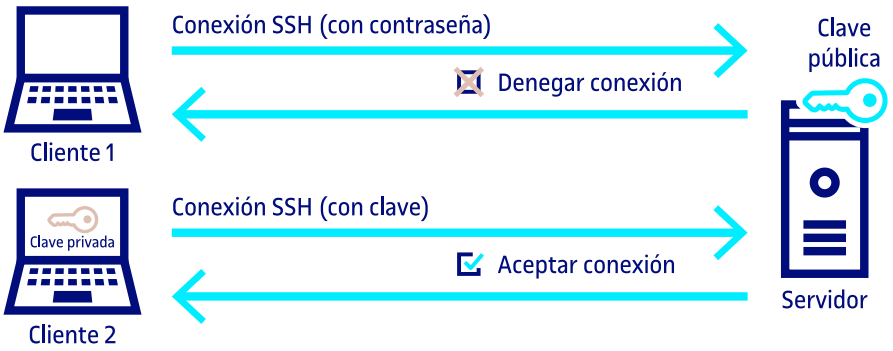
DigitalOcean es un proveedor estadounidense de **servidores virtuales privados**. La compañía alquila servidores de sus centros de datos, denominados *droplets*. Aprovechando que se trata de un proveedor de carácter simple, se explicarán los conceptos de seguridad básica práctica aplicada al entorno de DigitalOcean.

2.1. Claves SSH

Las claves SSH son un par de claves **criptográficas** con las que se puede autenticar contra un servidor SSH como alternativa a los inicios de sesión basados en contraseña. Con anterioridad a la autenticación, se crean un par de claves, una privada y la otra pública. La privada se mantiene secreta y asegurada por el usuario, mientras que la pública puede ser compartida con cualquiera.

Para configurar la autenticación SSH, hay que colocar la **clave pública** del usuario en el servidor en un directorio específico. Cuando el usuario se conecta al servidor, este comprobará que el cliente tiene la **clave privada asociada**, que deberá utilizar para probarlo. A partir de ahí, el servidor permitirá al cliente conectar sin el uso de contraseña (figura 3).

Figura 3. Ejemplo de intercambio de claves SSH



Fuente: elaboración propia

Mediante el uso de SSH, cualquier clase de autenticación, también la realizada con contraseña, es completamente encriptada. De todas formas, cuando el servidor permite el uso de contraseñas, un usuario malicioso podría intentar acceder al servidor por repetición. Con la capacidad actual de cómputo, es posible obtener acceso a un servidor mediante la automatización de los intentos hasta encontrar la contraseña correcta.

La instalación de la autenticación de claves SSH permite deshabilitar la autenticación basada en contraseñas.

Las claves SSH generalmente tienen muchos más bits de datos que una contraseña, lo que significa que hay muchas más combinaciones posibles y, por tanto, dificulta enormemente la labor de los atacantes para acceder al sistema.

Muchos algoritmos de clave SSH se consideran no *hackeables* por el hardware de computación actual, simplemente, porque requerirían demasiado tiempo para computar las diferentes posibilidades.

Las claves SSH son muy fáciles de instalar y es la forma recomendable de registrarse en cualquier entorno de servidor Linux o Unix de forma remota. El usuario puede generar el par de claves SSH en su máquina y luego transferir la clave pública a sus servidores en unos minutos mediante un gestor de configuración.

En el caso de tener que utilizar autenticación mediante contraseña, existe la posibilidad de implementar una solución como la aplicación Fail2ban en los servidores para limitar los intentos de acceso fraudulentos.

Enlace recomendado

Para saber más sobre la herramienta Fail2ban podéis consultar el enlace: <https://www.fail2ban.org>

2.2. Cortafuegos

Un cortafuegos es un software que controla qué servicios están expuestos a la red. Esto significa que bloquea o restringe el acceso a cada puerto, excepto a los que tengan que estar públicamente disponibles.

En un servidor típico, los servicios pueden ser categorizados en los grupos siguientes:

- **Servicios públicos** a los que puede acceder cualquier persona en Internet, a menudo de forma anónima. Un ejemplo es un servidor web.

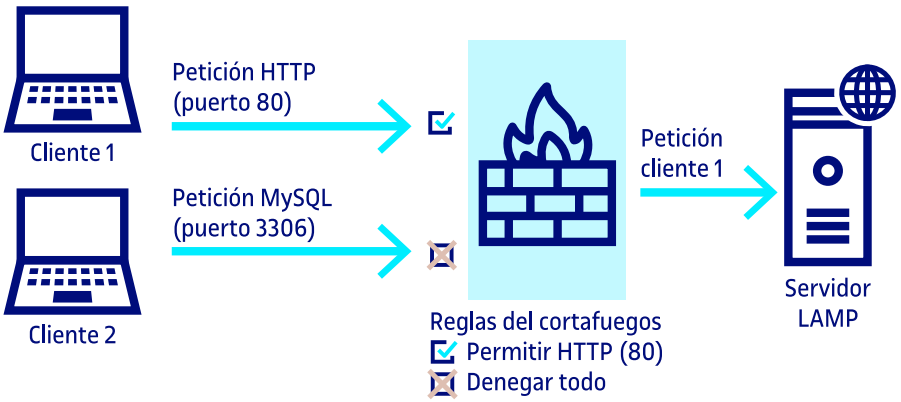
- **Servicios privados** a los que solo deberían acceder un grupo de cuentas autorizadas o sitios seguros. Un ejemplo puede ser un panel de base de datos.

- **Servicios internos** a los que se debería acceder solo desde dentro del propio servidor, sin exponer el servicio al mundo exterior. Por ejemplo, puede tratarse de una base de datos que solo acepta conexiones locales.

Un cortafuegos puede garantizar que el acceso a un determinado servicio esté restringido o no en función de las categorías mencionadas. Los servicios públicos pueden dejarse abiertos y estar disponibles para todos, mientras que los servicios privados pueden restringirse basándose en criterios diferentes (figura

4). Los servicios internos pueden ser completamente inaccesibles al mundo exterior. Para los puertos que no se utilizan, el acceso se bloquea completamente.

Figura 4. Ejemplo de funcionamiento del cortafuegos



Fuente: elaboración propia

El uso que le demos al cortafuegos es una parte esencial en la configuración de un servidor. Incluso si los servicios instalados implementan características de seguridad o están restringidos a las interfaces de red correspondientes, un cortafuegos provee de una capa adicional de protección.

Un cortafuegos correctamente configurado restringirá cualquier acceso, excepto a los servicios específicos que se necesita que permanezcan abiertos. Exponer solo unos pocos servicios reduce la superficie de ataque de un servidor, porque limita los componentes vulnerables a la explotación.

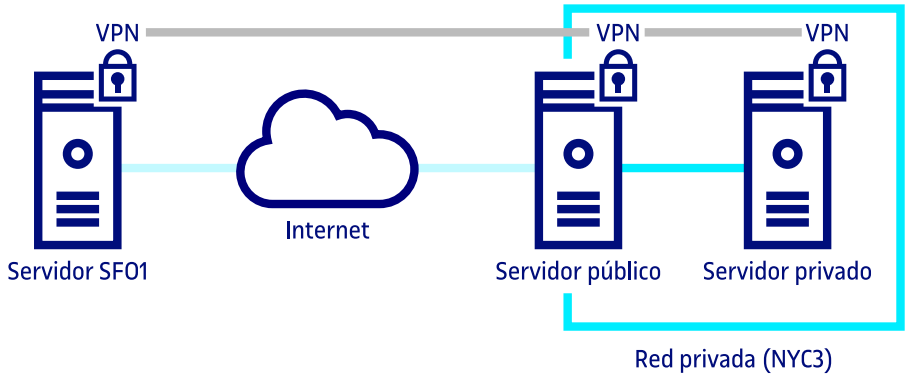
Hay muchos cortafuegos disponibles para sistemas Linux, algunos de los cuales tienen una curva de aprendizaje más pronunciada que otros. Por lo general, la instalación del cortafuegos solo debería tardar unos minutos y únicamente tendría que pasar por un servidor de configuración inicial o cuando se realicen cambios en los servicios.

2.3. VPN y redes privadas

Las redes privadas solo son visibles dentro de la red que forman nuestros servidores (figura 5).

Para conectar equipos remotos a una red privada, se utiliza una red privada virtual (VPN). Es una forma de crear conexiones seguras entre equipos remotos y presentar la conexión como una red privada local.

Figura 5. Ejemplo de funcionamiento de una conexión VPN



Fuente: elaboración propia

El uso de una VPN es efectivamente una forma de mapear una red privada que solo los usuarios pueden ver. La comunicación será totalmente privada y segura. Otras aplicaciones se pueden configurar para pasar el tráfico a través de la interfaz virtual expuesta por el software VPN. De esta manera, los únicos servicios que se exponen públicamente son los que están destinados a ser accedidos por los usuarios a través de Internet.

El uso de redes privadas en un centro de datos que tenga esta capacidad es sencillo. Basta con habilitar la interfaz privada durante la creación de configuraciones de servidores y aplicaciones, y configurar el cortafuegos para usar la red privada.

En cuanto a la VPN, la configuración inicial es un poco más complicada, pero la seguridad aumenta considerablemente; de hecho, es la opción recomendada para la mayoría de los casos de uso. En una VPN, cada servidor necesita instalar y configurar los datos comparativos de seguridad y configuración necesarios para establecer la conexión segura. Una vez que la VPN esté en funcionamiento, las aplicaciones deben configurarse para usar el túnel VPN.

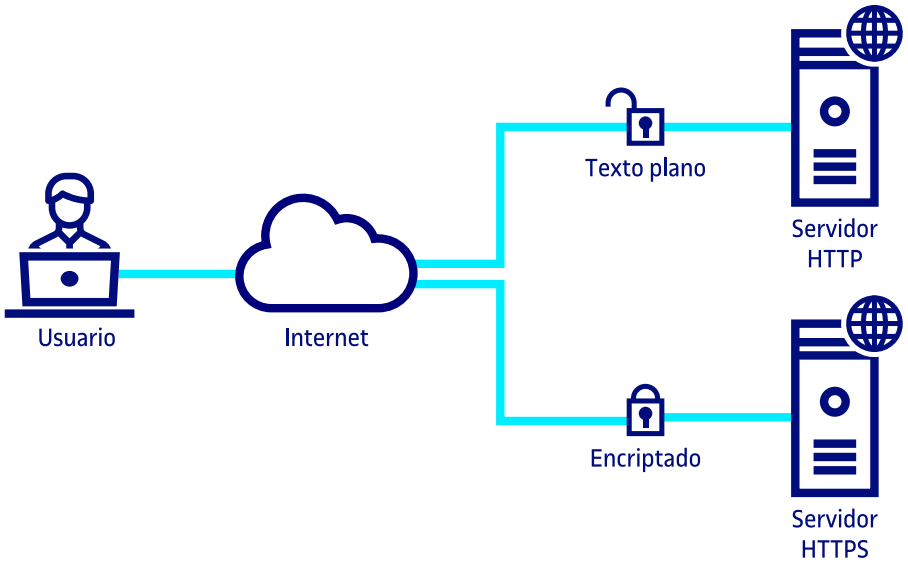
2.4. Infraestructura de clave pública y encriptación SSL/TLS

La **infraestructura de clave pública (PKI)** se refiere a un sistema diseñado para crear, dirigir y validar certificados con la finalidad de identificar comunicaciones individuales y de cifrado. Los **certificados SSL o TLS** pueden ser auténticos y diferentes. Después de la autenticación, la comunicación establecida también puede encriptarse usando los mismos certificados.

Se trata de establecer una autoridad de certificación y los certificados de los administradores de los servidores de cada entidad dentro de la infraestructura para validar la otra identidad de los miembros y cifrar el tráfico (figura 6).

Con la encriptación SSL/TLS se puede evitar el tipo de ataque a las comunicaciones en el que un atacante imita ser un servidor de la infraestructura para interceptar el tráfico.

Figura 6. Ejemplo de cifrado con SSL/TLS



Fuente: elaboración propia

Cada servidor puede configurarse para confiar en una **autoridad de certificación centralizada**. A continuación, se puede confiar implícitamente en cualquier certificado que señale la autoridad. La encriptación SSL/TLS es una manera de cifrar el sistema sin utilizar una VPN (que SSL también utiliza internamente).

Consiste en configurar la autoridad de certificación e instalar el resto de la infraestructura. Además, dirigir los certificados crea una nueva necesidad de gestión de los certificados que deben generarse, firmarse o revocarse.

Implementar una infraestructura de clave pública tiene sentido en infraestructuras grandes. Asegurar las comunicaciones entre los componentes que usan una VPN puede ser una buena estrategia hasta que se alcance el punto en el que la PKI valga la pena en compensación de los costos de administración adicionales.

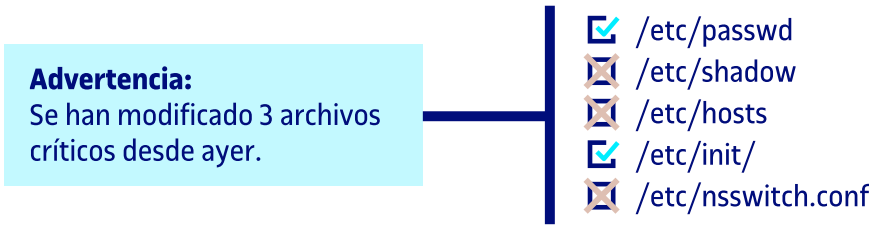
2.5. Auditoría de archivo y archivo de sistemas de detección

La auditoría de intrusión es el proceso que consiste en comparar el sistema actual de un registro de archivos y los archivos del sistema cuando se encontraban en buen estado. Por lo general, detecta cambios en el sistema que pueden no haber sido autorizados (figura 7).

La auditoría de archivo es una manera de asegurarse de que el sistema de archivos no ha sido alterado por ningún usuario o proceso.

Los intrusos a menudo quieren permanecer ocultos para seguir explotando el servidor durante un período prolongado de tiempo. Para ello, podrían reemplazar binarios con versiones comprometidas. Mediante una auditoría del sistema de archivos para saber si alguno ha sido alterado, se asegura la integridad del entorno del servidor.

Figura 7. Ejemplo de ficheros susceptibles de ser auditados



Fuente: elaboración propia

La realización de auditorías de archivo puede ser un proceso muy intenso. La configuración inicial implica la auditoría del sistema de cualquier cambio no estándar realizado en el servidor y la definición de rutas que deberían ser excluidas para crear una base de lectura.

También hace que las operaciones diarias sean más complicadas. Se trata de complicar los procedimientos de actualización cuando sea necesario para volver a comprobar el sistema antes de ejecutar actualizaciones y reconstruir la *baseline* después de ejecutar la actualización. También es necesario descargar los informes a otra ubicación para que un atacante no pueda alterar la auditoría.

Aunque esto puede aumentar la carga de administración, ser capaz de comprobar el sistema asegura que los archivos no hayan sido alterados. Algunos sistemas populares de detección son Tripwire y Aide.

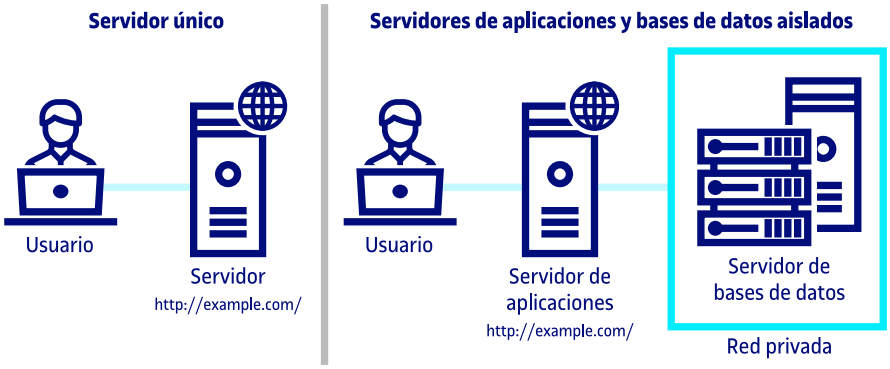
2.6. Entornos de ejecución aislada

Aislar entornos de ejecución se refiere a cualquier método por el que los componentes individuales se están ejecutando dentro de un espacio dedicado (figura 8).

Aislar entornos de ejecución puede significar que se han separado los componentes de aplicación en servidores propios o puede referirse a configurar los servicios para operar en entornos *chroot* o contenedores.

El nivel de aislamiento depende en gran medida de los requisitos de cada aplicación.

Figura 8. Ejemplo de entorno de ejecución aislada



Fuente: elaboración propia

Al aislar los procesos en entornos de ejecución **individual**, aumenta la capacidad de aislar cualquier problema de seguridad que pueda surgir. Separando los componentes individuales, se puede limitar el acceso de un intruso a otras piezas de la infraestructura.

En función del tipo de aislamiento, aislar una aplicación puede ser una tarea relativamente sencilla. Paquetizar los componentes individuales en contenedores es una manera de hacerlo; curiosamente, Docker no considera su aislamiento una medida de seguridad.

Si instalamos un ***chroot***, el entorno para cada pieza de software también puede proporcionar un cierto nivel de aislamiento, pero cabe recordar que hay maneras de romper el *chroot* incluso para aplicaciones que se encuentran dentro de este tipo de aislamiento.

3. Caso de uso: seguridad en entornos PaaS basados en contenedores

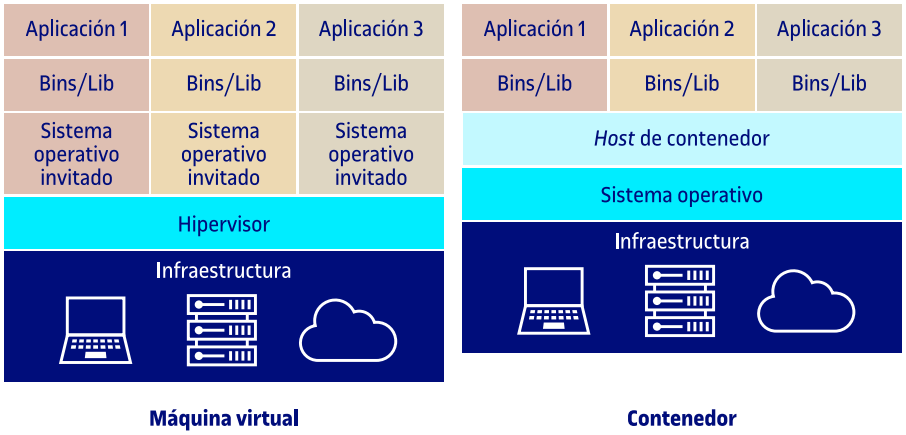
La inclusión en contenedores es un enfoque de desarrollo de software que consiste en que una aplicación o un servicio, incluyendo sus dependencias y su configuración (extraídos como archivos de manifiesto de implementación), se empaquetan como una **imagen de contenedor**. La aplicación en contenedor puede probarse como una unidad e implementarse como una instancia de imagen de contenedor en el sistema operativo (SO) *host*.

Del mismo modo que los contenedores de mercancías permiten su transporte por barco, tren o camión independientemente de la carga que contengan, los contenedores de software actúan como una unidad estándar de implementación de software que puede contener diferentes dependencias y código.

> La inclusión del software en contenedor permite a los desarrolladores y los profesionales de TI implementarlo en entornos con pocas modificaciones o ninguna en absoluto.

Los contenedores también aíslan las aplicaciones entre sí en un **sistema operativo compartido**. Las aplicaciones en contenedor se ejecutan sobre un *host* de contenedor que a su vez se ejecuta en el sistema operativo (Linux o Windows). Por lo tanto, los contenedores tienen una superficie significativamente menor que las imágenes de máquina virtual (VM) (figura 9).

Figura 9. Diferencias entre máquina virtual y contenedor



Fuente: elaboración propia
Bins/Lib: archivos binarios / biblioteca

La tabla 1 permite comparar las diferentes tecnologías de contenedores existentes en función de sus características más destacadas.

Tabla 1. Tecnologías de contenedores

	Docker	rkt	LXC	LXD
Tecnología de virtualización	En el nivel del sistema operativo	En el nivel del sistema operativo, hipervisor	En el nivel del sistema operativo	En el nivel del sistema operativo
Full system container	No	No	Sí	Sí
Contenedor de aplicaciones	Sí	Sí	No	No
Licencia	Apache 2.0	Apache 2.0	GNU LGPLv2.1+	Apache 2.0
Formato del contenedor	Docker Container	appc, Docker Container	Lux Container (LXC)	Lux Container (LXC)
Plataformas compatibles	Linux, Windows, macOS, Microsoft Azure, Amazon Web Services (AWS)	Linux, Windows, macOS	Linux	Linux

Fuente: elaboración propia

Para el caso de uso que nos ocupa, nos centraremos exclusivamente en la seguridad de contenedores basados en Docker.

3.1. Docker

Docker es una plataforma que nos permite construir y ejecutar aplicaciones distribuidas basadas en contenedores. Esta nueva forma de trabajar está cambiando el paradigma del desarrollo de aplicaciones, de manera que se ha pasado de aplicaciones monolíticas a aplicaciones basadas en microservicios. Muchas de las grandes empresas, como Google, Spotify e infinidad de empresas emergentes, recorren a esta arquitectura, afrontando nuevos retos en cuanto a seguridad se refiere.

Cada aplicación basada en Docker debe aislarse conceptualmente de cualquier otra, y para ello se limitan y controlan los recursos a los que estas pueden acceder mediante capacidades heredadas del *host*, como *namespaces* y *cgroups*.

Docker ofrece **seguridad basada en capas**. Por defecto, estas ya incorporan métodos de aislamiento, pero debemos recordar que Docker es una plataforma que puede ejecutarse en máquinas virtuales; en ese caso, podría añadirse una capa suplementaria si fuese necesario un aislamiento extra.

Para definir cómo asegurar la seguridad en un entorno *dockerizado*, no existe un protocolo infalible, sino que seguiremos un conjunto de recomendaciones para mitigar y dificultar cualquier ataque recibido.

Entre los ataques más comunes a los que deberemos prestar especial atención encontramos:

- ***Container scape***: escalado de privilegios en el *host*.
- ***Root/Kernel exploit***: *host* comprometido.
- **DOS** (*Denial of service*): pérdida de disponibilidad.
- ***Software & Crypto exploit***: pérdida de confidencialidad.

El objetivo de este apartado es proporcionar una lista con los errores más comunes de seguridad y una guía de buenas prácticas que nos ayudarán a proteger los contenedores basados en Docker. A continuación, exponemos las reglas a seguir.

1) Seguridad en el host. Para definir la seguridad de nuestras aplicaciones *dockerizadas*, deberemos primero centrarnos en la raíz, donde estas se ejecutarán.

Para evitar las vulnerabilidades conocidas, como el ***Container scape***, que generalmente acaban en un escalado de privilegios de *root*, es muy importante mantener tanto el *host* como Docker Engine y Docker Machine actualizados a la última versión.

Además, los contenedores (a diferencia de las máquinas virtuales) comparten el *kernel* con el *host*, es decir, las vulnerabilidades del *kernel* ejecutadas dentro del contenedor afectarán directamente al *kernel* del *host*.

2) Aislamiento del *socket* de Docker. El *socket* de Docker `/var/run/docker.sock` es el de UNIX donde está escuchando Docker. Este es el punto de entrada principal para la API de Docker. El propietario de este *socket* es *root*. Si un usuario malintencionado pudiera acceder, sería equivalente a darle acceso de *root* sin restricciones al *host*.

3) Establecimiento de un usuario sin privilegios. Configurar el contenedor para que lo use un usuario sin privilegios es la mejor manera de prevenir ataques de escalada de privilegios. Esto se puede lograr de diferentes maneras:

a) Durante el tiempo de ejecución, usando la opción `-u` del comando `docker run`. Por ejemplo:

```
docker run -u 4000 alpine
```

b) Durante la construcción de la imagen, simplemente agregando un usuario en `Dockerfile` y usándolo. Por ejemplo:

```
FROM alpine
RUN groupadd -r myuser && useradd -r -g myuser myuser
```

Exploit de escalada de privilegios

Un *exploit* de escalada de privilegios del *kernel* (como Dirty COW) ejecutado dentro de un contenedor bien aislado dará como resultado el acceso de *root* en un *host*.

```
<HERE DO WHAT YOU HAVE TO DO AS A ROOT USER LIKE INSTALLING PACKAGES ETC.>
USER myuser
```

c) Espacio de nombres (*namespaces*): los espacios de nombres son una característica del *kernel* de Linux que aísla y virtualiza los recursos del sistema de una colección de procesos.

Los contenedores Docker son muy similares a los contenedores LXC y comparten características de seguridad parecidas. Cuando se inicia un contenedor con Docker, se crean un conjunto de espacios de nombres y grupos de control para el contenedor.

Los espacios de nombres proporcionan la primera y más sencilla forma de aislamiento: los procesos que se ejecutan dentro de un contenedor no pueden ver los procesos que se ejecutan en otro contenedor o en el sistema *host*.

Para remapear nuestro usuario con otro usuario con su propio identificador dentro del contenedor, ejecutaremos:

```
docker daemon --userns-remap=<nombre_usuario>
```

Esta instrucción tomará el identificador del `<usuario>` dentro del *host* y lo remapeará dentro del contenedor siguiendo la fórmula:

```
FIRST_SUB_UID = 100000 + (UID - 1000) * 65536
```

Por ejemplo, si nuestro usuario en el *host* está definido por `usuario::100000:65536`, al ejecutar el remapeo podremos encontrar que dentro del contenedor se corresponderá con `usuario:1000:1`.

Seguiremos algunas recomendaciones que afectarán a distintos ámbitos del sistema anfitrión.

4) Uso de imágenes validadas. Durante la construcción de nuestros contenedores, es común el uso de imágenes de terceros como punto de partida. La recomendación más clara es la de usar repositorios validados. Cualquier fuente no confiable puede exponer nuestro contenedor a numerosas vulnerabilidades.

5) Securización del *kernel* de Linux. Es recomendable aplicar los parches de Grsecurity (<grsecurity.net>), que ofrecen una mejora de seguridad para el *kernel* de Linux, ya que incorporan mecanismos ante amenazas de seguridad a través de un control de acceso inteligente y mediante controles de prevención de vulnerabilidades basadas en corrupción de memorias.

6) Recursos del sistema con `ulimit`. Linux incorpora el comando `ulimit`, encargado de obtener y modificar los límites de los recursos del usuario. Con este comando, podremos limitar el número máximo de ficheros abiertos o el número máximo de procesos que los contenedores podrán manejar por usuario. Por otro lado, Docker también incorpora mecanismos para definir estos valores mediante el uso de parámetros con la misma nomenclatura (`ulimit`).

Nota

Los valores `ulimit` especificados para un contenedor sobrescriben los valores predeterminados establecidos para el demonio.

Con la limitación del número de recursos con `ulimit`, estableceremos valores máximos para los contenedores, pero cabe recordar que otras aplicaciones como Apache o Nginx disponen de parámetros extra para establecer distintos límites dentro de ellos.

Podremos definir distintos tipos de limitaciones para contenedores de forma global o local. A continuación, mostraremos distintos ejemplos de configuración de máximo, tanto de ficheros abiertos como de procesos activos:

a) Estableciendo límites durante el arranque del demonio de Docker:

```
docker daemon --default-ulimit nofile=100:200
```

El parámetro `nofile` define los valores máximos de la forma siguiente: `<soft limit>[:<hard limit>]`. Diferenciaremos el `soft limit` y el `hard limit`: el primero es lo que realmente se aplica para una sesión o proceso y permite al administrador (o usuario) establecer el límite máximo requerido; el segundo es un campo opcional y define el límite máximo para el parámetro definido por el `soft limit`.

b) Modificando el fichero `/etc/security/limits.conf`:

```
docker soft nofile <max_soft_limit>
docker hard nofile <max_hard_limit>
```

c) Mediante la sobreescritura de los valores de límites por defecto en el momento de la ejecución de un contenedor específico: limitando el número de ficheros máximos abiertos mediante el parámetro `--ulimit nofile` o estableciendo el número máximo de procesos abiertos por el contendor.

```
docker run --name <nombre container> --ulimit nofile=<soft
limit>:<hard limit> ...
docker run --name <nombre container> --ulimit nproc=<soft
limit>:<hard limit> ...
```

d) Estableciendo las opciones oportunas dentro del fichero de configuración de Docker `/etc/sysconfig/docker`:

```
OPTIONS="--ulimit nofile=<soft limit>:<hard limit> --ulimit
nproc=<soft limit>:<hard limit>"
```

O en el fichero `/etc/systemd/system/docker.service.d`:

```
[Service]
Environment="OPTIONS=$OPTIONS \"--default-ulimit nofile=<soft limit>:<hard
limit>\""
Environment="OPTIONS=$OPTIONS \"--default-ulimit nproc=<soft limit>:<hard
limit>\""
```

Después de agregar o modificar un archivo, deberemos ejecutar el comando `systemctl daemon-reload` para indicar a `systemd` que vuelva a cargar la configuración del servicio.

7) *Capabilities*. Las *capabilities* definen un sistema de seguridad que permite «fragmentar» los privilegios de *root* en distintos componentes, que podrán ser asignados de forma individual a procesos que requieran privilegios especiales para recursos específicos sin necesidad de poseer privilegios de superusuario.

Enlace recomendado

Para saber más sobre *Linux kernel capabilities*, podéis ver el enlace siguiente:

<https://man7.org/linux/man-pages/man7/capabilities.7.html>

Docker utiliza esta granulación de privilegios para restringir los accesos a recursos, de forma que define qué capas del contenedor podrán tener acceso.

Para definir un acceso global a los recursos, aunque no sea recomendable, se utilizará la instrucción:

```
docker run --privileged=true...
```

A menudo es necesario exponer directamente los dispositivos a un contenedor. La opción que se debe utilizar será `--device`, que permitirá vincular el contenedor con dispositivos de almacenamiento específicos o dispositivos de audio sin poseer privilegios de *root*:

```
# Acceso a disco en modo lectura/escritura
docker run --device=/dev/sdd -i -t ubuntu

# Acceso a la tarjeta de sonido
docker run --device=/dev/snd:/dev/snd ...
```

Por defecto, el contenedor podrá leer (`read`), escribir (`write`) y `mknod` sobre estos dispositivos. Estos privilegios pueden ser sobreescritos mediante las opciones `:rwm` para cada etiqueta *device*:

```
#Permisos de lectura/escritura esplicito
docker run --device=/dev/sda:/dev/xvdc:rw --rm -it ubuntu fdisk
/dev/xvdc
```

8) Módulos de seguridad (AppArmor). Tanto AppArmor (Application Armor) y SeLinux son módulos de seguridad de Linux que protegen al sistema operativo y a sus aplicaciones contra amenazas de seguridad.

Para utilizar AppArmor, un administrador del sistema asocia un perfil de seguridad AppArmor para cada programa. Así, pues, Docker espera encontrar una política de AppArmor cargada y aplicada para limitar la aplicación *dockerizada*. El binario Docker instala un perfil predeterminado de AppArmor en el archivo `/etc/apparmor.d/docker`. Este perfil se utilizará en los contenedores, no en el Docker Daemon.

9) Computación segura (Seccomp). El modo de computación segura (Seccomp) es una característica del *kernel* de Linux. Se utiliza para restringir las acciones disponibles dentro del contenedor. La llamada al sistema `seccomp ()` opera en el estado de la llamada. Esta función se puede usar para restringir el acceso a la aplicación.

Lectura recomendada

Es aconsejable la lectura de la documentación oficial para la especificación de todas aquellas llamadas al sistema disponibles.

Esta función solo está disponible si Docker se ha creado con Seccomp y el núcleo está configurado con `CONFIG_SECCOMP` habilitado. Para comprobar si el *kernel* es compatible con Seccomp, ejecutaremos:

```
cat /boot/config-`uname -r` | grep CONFIG_SECCOMP=
CONFIG_SECCOMP=y
```

El perfil de Seccomp predeterminado deshabilita alrededor de 44 llamadas de sistema de más de 300 disponibles. Es moderadamente protector y es compatible con una amplia variedad de aplicaciones. El perfil predeterminado de Docker tiene un diseño JSON y puede sobrescribirse mediante:

```
docker run --rm -it --security-opt
seccomp=/path/to/seccomp/profile.json hello-world
```

Donde `profile.json` sería un fichero JSON que sigue la estructura siguiente:

```
{
```

```
    "defaultAction": "SCMP_ACT_ERRNO",
    "architectures": [
        "SCMP_ARCH_X86_64",
        "SCMP_ARCH_X86",
        "SCMP_ARCH_X32"
    ],
    "syscalls": [
        {
            "name": "accept",
            "action": "SCMP_ACT_ALLOW",
            "args": []
        },
        {
            "name": "accept4",
            "action": "SCMP_ACT_ALLOW",
            "args": []
        },
        ...
    ]
}
```

10) Limitación de CPU y RAM

a) Limitación de CPU. En algunos entornos, nos interesará la limitación del uso de la CPU del *host* para protegerlo frente a ataques que intenten vulnerar las capacidades del procesador anfitrión. Para ello, Docker dispone de opciones de configuración específicos:

- `--cpu-shares <int>`: la limitación relativa a la cantidad de procesadores disponibles se especifica en tanto por ciento.

- `--cpu-quota <int>`: en cuanto a la limitación del uso de la CPU en el contenedor, el valor 0 predeterminado permite que el contenedor tome la totalidad de un recurso de CPU (1 CPU). Para limitar el uso del procesador al 50 %, estableceremos un valor de cuota de 50000.

```
#Límite del uso de la CPU al 50%
docker run -ti --cpu-quota=50000 ubuntu:latest /bin/bash.
```

- `--cpu-period <int>`: esta opción se utiliza para establecer el período de CPU y limitar el uso de la CPU del contenedor. El período por defecto de CPU CFS es de 100 ms. Usualmente `--cpu-period` funciona juntamente con la opción `--cpu-quota`.

```
#Limitación del 50% de la CPU cada 50ms
```

```
docker run -it --cpu-period=50000 --cpu-quota=25000 ubuntu:14.04
/bin/bash
```

- `--cpuset-cpus <string>`: permite la restricción del uso de determinadas CPU de nuestro *host*. Estos se especificarán en formato `1,3` en el caso de querer utilizar solo 2 CPU (número 1 y 3) o con el formato `0-2` para indicar que usaremos un rango (uso de CPU 0,1 y 2).

b) Limitación de RAM. En Docker, o en cualquier otro sistema contenedor, la memoria es administrada por el *kernel* que ejecuta Docker y los contenedores. El proceso que se ejecuta en el contenedor sigue funcionando como un proceso normal, pero en un *cgroup* es diferente, porque cada *cgroup* posee sus propios límites, como la cantidad de memoria que el *kernel* entregará al espacio del usuario o, como ya hemos visto, qué dispositivos tendrá disponibles.

Para especificar las limitaciones oportunas, usaremos las opciones de ejecución de la familia `--memory` de Docker:

- `--memory <int>[<unidad (b, k, m o g)>]`: para definir la cantidad de memoria máxima que el contenedor tendrá disponible, se indicará en forma de número entero positivo la cantidad de memoria seguido de la unidad correspondiente (`b`, `k`, `m` o `g`). El mínimo de memoria es de 4M.

```
#Limitación de la memoria a 1Gb
docker run --memory 1G ubuntu
```

- `--memory-reservation <int>[<unidad (b, k, m o g)>]`: se utiliza normalmente en conjunción con la instrucción `--memory`. En el caso de especificar ambos parámetros, la memoria (especificada con opción `--memory`) debe ser mayor que la memoria reservada. Al especificar una cantidad de memoria reservada, el contenedor se iniciará con el valor reservado, de lo contrario utilizará el valor de la memoria asignada.
  Por ejemplo, si un contenedor normalmente utiliza 256 Mb de memoria, pero ocasionalmente requiere a 512 Mb por períodos cortos de tiempo, podemos indicar una reserva de memoria de 256 Mb y un límite de memoria de 600 Mb, de forma que establecemos los dos ámbitos donde la memoria del contenedor puede ubicarse. El mínimo de memoria reservada es de 4M.

```
#Reserva de memoria inicial de 256Mb con límite de 600Mb
docker run --memory 600M --memory-reservation 256M ubuntu
```

- `--memory-swap <int>[<unidad (b, k, m o g)>]`: los contenedores pueden hacer uso de la memoria SWAP, la cual amplía los límites de la

memoria RAM fijada por el parámetro `--memory` y establece el uso total de memoria como (memoria + memoria SWAP asignada).

- `--memory-swappiness <int>`: existe la posibilidad de ajustar la frecuencia con la que se utilizan los ficheros de intercambio del contenedor con valores de 0 a 100. Los valores bajos indicarán que no se utilizará memoria SWAP a menos que sea totalmente necesario; por otro lado, los valores altos indicarán un uso exhaustivo de este tipo de memoria.

11) Establecimineto del sistema de archivos y los volúmenes en solo lectura. Consiste en ejecutar los contenedores con un sistema de archivos de solo lectura usando la marca `--read-only`. Por ejemplo:

```
docker run --read-only alpine sh -c 'echo "whatever" > /tmp'
```

Si una aplicación dentro de un contenedor tiene que guardar algo temporalmente, se puede combinar la marca `--read-only` con `--tmpfs` de esta manera:

```
docker run --read-only --tmpfs /tmp alpine sh -c 'echo "whatever" >
/tmp/file'
```

12) Redes. Cada contenedor Docker tiene su propia pila de red, lo que significa que un contenedor no obtiene acceso privilegiado a los *sockets* o interfaces de otro contenedor. No obstante, esto será posible si el sistema *host* está configurado para ello, e incluso se podrá interactuar con *hosts* externos.

Cuando se especifica un puerto público para el contenedor o se utilizan vínculos, se permite el tráfico IP entre contenedores, es decir, pueden hacerse *ping* unos a otros, enviar y recibir paquetes UDP, y establecer conexiones TCP.

Desde el punto de vista de la arquitectura de red, todos los contenedores de un *host* Docker determinado están situados en interfaces de puente (*bridge*). Esto significa que son como máquinas físicas conectadas a través de un conmutador Ethernet común.

13) Uso de herramientas de análisis estático. Para detectar contenedores con vulnerabilidades conocidas, se pueden escanear imágenes utilizando herramientas de análisis estático:

- Clair
- Trivy
- Snyk (código abierto y opción gratuita disponible)
- anchore (código abierto y opción gratuita disponible)
- JFrog XRay

- Qualys

Para detectar secretos en imágenes:

- GitGuardian shield (ggshield) (código abierto y opción gratuita disponible)
- SecretScanner (gratis)

Para detectar configuraciones incorrectas en Kubernetes:

- kubeaudit
- kubesec.io
- kube-bench

Para detectar configuraciones incorrectas en Docker:

- inspec.io
- DevSec Hardening Framework

3.2. Buenas prácticas de seguridad

La seguridad de los sistemas es una cuestión bien tratada por los propios contenedores y Docker está dedicando cada vez más esfuerzos a mejorar los sistemas integrados de seguridad de los contenedores. De todas formas, para comprobar que nada quede en el descuido, incluimos un listado de noventa buenas prácticas desarrolladas por el Center for Internet Security (CIS) para Docker que se podrán seguir a modo de lista de comprobación.

Buenas prácticas para Docker

1) Configuración del *host*:

- Crear una partición separada para los contenedores.
- Uso del *kernel* de Linux actualizado.
- No utilizar herramientas de desarrollo en producción.
- Asegurar el *host* anfitrión del contenedor.
- Eliminar todos los servicios no esenciales del *host*.
- Mantener Docker actualizado.
- Permitir que solo los usuarios de confianza controlen el demonio de Docker.
- Aplicar técnicas de auditoría sobre el demonio de Docker.
- Auditar archivos y directorios de Docker: `/var/ lib/docker`.
- Auditar archivos y directorios de Docker: `/etc/docker`.
- Auditar archivos y directorios de Docker: `docker-registry.service`.
- Auditar archivos y directorios de Docker: `docker.service` (marcado).
- Auditar archivos y directorios de Docker: `/var/run/docker.sock`.
- Auditar archivos y directorios de Docker: `/etc/sysconfig/docker`.
- Auditar archivos y directorios de Docker: `/etc/sysconfig/docker-network`.
- Auditar archivos y directorios de Docker: `/etc/sysconfig/docker-registry`.
- Auditar archivos y directorios de Docker: `/etc/sysconfig/docker-storage`.
- Auditar archivos y directorios de Docker: `/etc/default/docker`.

2) Configuración del demonio de Docker:

- No utilizar el controlador de ejecución LXC.
- Limitar el tráfico de red entre contenedores.
- Establecer el nivel adecuado de *logging*.

- Permitir que Docker realice cambios en *Iptables*.
- No utilizar registros inseguros (sin TLS).
- Configurar un espejo del registro local.
- No utilizar Aufs como controlador de almacenamiento.
- No vincular Docker a otro puerto IP o un *socket* de Unix.
- Configurar la autenticación TLS para el demonio de Docker.
- Establecer el valor predeterminado Ulimit según corresponda.

3) Archivos de configuración del demonio Docker:

- Comprobar que la propiedad del archivo `docker.service` está establecida en `root:root`.
- Comprobar que los permisos del archivo `docker.service` están establecidos en 644 o más restrictivos.
- Comprobar que la propiedad del archivo `docker-registry.service` está establecida en `root:root`.
- Comprobar que los permisos de archivo `docker-registry.service` están establecidos en 644 o más restrictivos.
- Comprobar que la propiedad del archivo `docker.socket` está establecida en `root:root`.
- Comprobar que los permisos del archivo `docker.socket` están establecidos en 644 o más restrictivos.
- Comprobar que la propiedad del archivo de entorno Docker está establecida en `root:root`.
- Comprobar que los permisos del archivo de entorno de Docker se establecen en 644 o más restrictivos.
- Comprobar que la propiedad del archivo del entorno `docker-network` está establecida en `root:root`.
- Comprobar que los permisos de archivos de entorno de red Docker están configurados en 644 o más restrictivos.
- Comprobar que la propiedad del archivo del entorno `docker-registry` está establecida en `root:root`.
- Comprobar que los permisos de archivo del entorno de registro de base de datos se establecen en 644 o más restrictivos.
- Comprobar que la propiedad del archivo del entorno de almacenamiento acoplador está establecida en `root:root`.
- Comprobar que los permisos de archivo del entorno de almacenamiento acoplador se establecen en 644 o más restrictivos.
- Verificar que la propiedad del directorio `/etc/docker` está establecida en `root:root`.
- Verificar que los permisos del directorio `/etc/docker` están configurados en 755 o más restrictivos.
- Comprobar que la propiedad del archivo del certificado del `registry` se establece en `root:root`.
- Comprobar que los permisos de los archivos del certificado del `registry` se establecen en 444 o más restrictivos.
- Comprobar que la propiedad del archivo del certificado de la CA de TLS está establecida en `root:root`.
- Comprobar que los permisos de archivo del certificado de CA de TLS se establecen en 444 o más restrictivos.

- Verificar que la propiedad del archivo del certificado del servidor Docker está establecida en `root:root`.

- Verificar que los permisos de los archivos de certificados del servidor Docker estén establecidos en 444 o más restrictivos.

- Comprobar que la propiedad del archivo de clave de certificado del servidor Docker se establece en `root:root`.

- Verificar que los permisos de archivo de clave de certificado del servidor Docker estén establecidos en 400.

- Verificar que la propiedad del archivo de *socket* Docker está establecida en `root:docker`.

- Comprobar que los permisos de archivo de zócalo de Docker están establecidos en 660 o más restrictivos.

4) Imágenes de contenedor y archivo de generación:

- Crear un usuario para el contenedor.
- Utilizar imágenes de confianza para los contenedores.
- No instalar paquetes innecesarios en el contenedor.
- Regenerar las imágenes para incluir parches de seguridad.

5) Tiempo de ejecución del contenedor:

- Verificar el perfil de AppArmor.
- Verificar las opciones de seguridad SELinux.
- Verificar que los contenedores solo ejecutan un proceso principal.
- Restringir las capacidades del *kernel* de Linux dentro de los contenedores.
- No utilizar contenedores con privilegios.
- No montar directorios sensibles del *host* en los contenedores.
- No ejecutar SSH dentro de los contenedores.
- No asignar puertos privilegiados dentro de los contenedores.
- Abrir solo los puertos necesarios en el contenedor.
- No utilizar el modo *host network* en el contenedor.
- Limitar el uso de memoria para el contenedor.
- Establecer la prioridad de CPU del contenedor de forma adecuada.
- Montar el sistema de archivos raíz del contenedor como solo lectura.
- Vincular el tráfico de contenedores entrantes a una interfaz de *host* específica.
- Establecer la política de reinicio del contenedor en modo *on-failure*.
- No compartir el PID del proceso del *host* anfitrión con los contenedores.
- No compartir el espacio de nombres IPC del anfitrión.
- No exponer directamente los dispositivos del anfitrión con los contenedores.
- Sobrescribir el valor predeterminado Ulimit en tiempo de ejecución solo si es necesario.

6) Operaciones de seguridad de Docker:

- Realizar auditorías regulares de seguridad del sistema anfitrión y de los contenedores.
- Monitorizar el uso, el rendimiento y las métricas de los contenedores de Docker.
- Uso de *endpoint protection platform* (EPP) para contenedores.
- Hacer copias de respaldo de los contenedores.
- Utilizar un servicio de recolección de registros centralizado y remoto.
- Evitar el almacenamiento de imágenes obsoletas o sin etiquetas correctas.
- Evitar el almacenamiento de contenedores obsoletos o sin etiquetas correctas.

Abreviaturas

ACL *Access control list* (listas de control de acceso)

Amazon EC2 Amazon Elastic Compute Cloud

AWS Amazon Web Services

CDN *Content delivery network* (red de distribución de contenido)

CIS Center for Internet Security

DDoS *Distributed denial of service*

DOS *Denial of service*

DNS *Domain name system* (servicio de nombres de dominio)

ELB Elastic Load Balancing

EPP *Endpoint protection platform*

IaaS *Infrastructure as a service* (infraestructura como servicio)

IAM Identity and Access Management

LXC Lux Container

ML *Machine learning* (aprendizaje automático)

MTTR *Mean time to repair* (tiempo medio de resolución)

NACL *Network access-control list* (listas de control de acceso a la red)

PKI *Public key infrastructure* (infraestructura de clave pública)

Seccomp *Secure computing mode* (computación segura)

SO *Operating system* (sistema operativo)

SRE *Site reliability engineering* (ingenieros de fiabilidad de sitio)

TTL *Time to live* (tiempo de vida)

VM *Virtual machine* (máquina virtual)

VPC Virtual Private Cloud

VPN *Virtual private network* (red privada virtual)

WAF *Web application firewall* (cortafuegos de aplicaciones web)

Bibliografía

Anthony, Albert (2017). *Mastering AWS Security: Create and maintain a secure cloud ecosystem*. Birmingham, RU: Packt Publishing.

Dotson, Chris (2019). *Practical Cloud Security: A Guide for Secure Design and Deployment*. Sebastopol, CA: O'Reilly Media.

Guijarro Olivares, Jordi; Caparrós Ramírez, Joan; Cubero Luque, Lorenzo (2019). *DevOps y seguridad cloud*. Barcelona: Editorial UOC.

Webgrafía

AWS: «AWS Documentation». <https://docs.aws.amazon.com>

DevSec Hardening Framework. <https://dev-sec.io/>

DigitalOcean: «Recommended Security Measures to Protect Your Servers». <https://www.digitalocean.com>

Fail2ban. <https://www.fail2ban.org>

OWASP Cheat Sheet Series: «OWASP Cheat Sheet Series Project». <https://cheatsheetseries.owasp.org>

Linux: «capabilities(7) - Linux manual page». <https://man7.org/linux/man-pages/man7/capabilities.7.html>