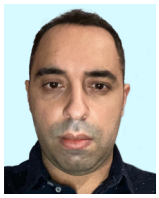
Fundamentos de DevSecOps

PID_00286171

Miguel Ángel Flores Terrón

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 Universitat Oberta de Catalunya (UOC). Desde 2020 es profesor colaborador en la UOC como tutor de trabajos final de máster en el ámbito de la ciberseguridad. Actualmente, trabaja como administrador de sistemas en la unidad de Operaciones y Ciberseguridad del Consorci de Serveis Universitaris de Catalunya (CSUC).

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

Primera edición: febrero 2022
© de esta edición, Fundació Universitat Oberta de Catalunya (FUOC)
Av. Tibidabo, 39-43, 08035 Barcelona
Autoría: Miguel Ángel Flores Terrón
Producción: FUOC
Todos los derechos reservados

Ninguna parte de esta publicación, incluido el diseño general y la cubierta, puede ser copiada, reproducida, almacenada o transmitida de ninguna forma, ni por ningún medio, sea este eléctrico, mecánico, óptico, grabación, fotocopia, o cualquier otro, sin la previa autorización escrita del titular de los derechos.

Índice

Introducción

En el ámbito del desarrollo ágil de software, la seguridad de las aplicaciones adquiere un papel cada vez más importante, y tanto la integración como las entregas aspiran a ser continuas, lo que se conoce como **integración continua** (*continuous integration*), **entrega continua** (*continuous delivery*) o CI/CD. Por este motivo, cada vez más organizaciones están empezando a adoptar una cultura DevOps, que entrelaza desde el inicio los procesos de los equipos de desarrollo y operaciones, y lo amplían con distintos componentes de seguridad: de ahí el acrónimo DevSecOps, también conocido como SecDevOps.

DevSecOps es una filosofía de desarrollo de software que promueve la adopción de la seguridad en todo el ciclo de vida del desarrollo de software (SDLC); también implica automatizar algunas tareas de seguridad para impedir que se ralentice el flujo de trabajo de DevOps.

Los mecanismos de seguridad están integrados en el proceso ya desde el inicio del desarrollo. Esta es una de las claras diferencias entre el sistema DevSecOps y los enfoques convencionales, en los que los equipos de seguridad suelen aplicar las medidas correspondientes una vez que se ha finalizado el producto o aplicación en sí. El propósito y la intención de DevSecOps es construir sobre la mentalidad de que «**todos son responsables de la seguridad**».

Objetivos

Con los materiales de este módulo didáctico se busca que aprendáis y desarrolléis los conocimientos y las habilidades siguientes:

1. Entender el concepto de DevOps y DevSecOps, y conocer los principios y las prácticas relacionadas con DevOps.

2. Conocer y entender las diferentes fases del ciclo de DevOps.

3. Identificar las fases del ciclo de DevOps donde podemos aplicar medidas de seguridad.

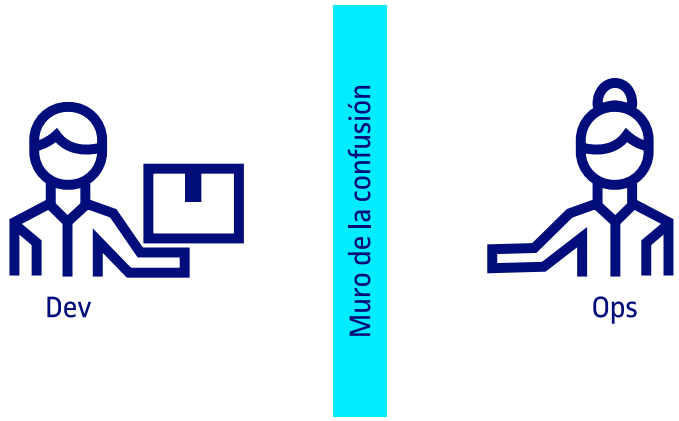
4. Ver un ejemplo concreto de un flujo DevSecOps (*pipeline*) y conocer las diferentes herramientas de software libre que podemos utilizar para cada una de las fases que componen el *pipeline*.

1. Introducción a DevOps

El término *DevOps* surge a finales de los años 2000, en un contexto en el que las metodologías de desarrollo ágiles cada vez adquirían mayor relevancia en la industria del desarrollo de software. La velocidad a la que se desarrollaban nuevas funcionalidades o se corregían defectos en el software (*bugs*) distaba mucho de la velocidad a la que se realizaban los despliegues de estos cambios, con lo que el ciclo de desarrollo del software se veía gravemente ralentizado. Esta ralentización habitualmente era resultado de la falta de comunicación existente entre los equipos de desarrollo y el de operaciones.

A esta falta de colaboración es lo que en DevOps suele denominarse el «**muro de la confusión**» (figura 1). Este muro metafórico existe cuando un equipo que forma parte de una cadena de valor considera que ha terminado su trabajo al pasárselo al siguiente equipo en la cadena, sin considerar los problemas que estos puedan tener y la ayuda que podrían necesitar de estos para resolverlos.

Figura 1. Representación del «muro de la confusión»



Fuente: elaboración propia a partir de Marvin López (2020, 30 de octubre). «El muro de la confusión» [en línea]. *Marvin López M.* <https://www.imarv.in/el-muro-de-la-confusion>

Por su puesto, la falta de colaboración tampoco es exclusiva de los equipos de desarrollo y operaciones, sino que pueden existir muros en los diferentes departamentos de una compañía (figura 2).

Figura 2. Representación de los diferentes «muros» que puede haber en una empresa

Fuente: elaboración propia

Aunque resulta complicado ofrecer una única definición del término *DevOps*, se suele describir como una cultura de trabajo que incluye una serie de principios y prácticas que sirven como base a las organizaciones a la hora de planificar y organizar el desarrollo de un producto.

> Lo que se pretende con la aplicación de la filosofía DevOps es la creación de **equipos multidisciplinares y autónomos**, bien comunicados entre sí, y que tengan responsabilidad y conocimiento del ciclo de vida de desarrollo al completo.

Entre las principales ventajas de adoptar prácticas de DevOps se encontrarían las siguientes:

- Mejora de la productividad y el rendimiento.
- Reducción del tiempo de entrega (*time-to-market*).
- Mayor agilidad para responder a cambios o incorporar nuevas funcionalidades en el software.
- Entornos de funcionamiento más estables.
- Mejor escalabilidad y disponibilidad.
- Mayor innovación.
- Mayor automatización.

1.1. Principios y buenas prácticas

Como se mencionó con anterioridad, al no existir una definición concreta de DevOps, tampoco hay unos principios claramente definidos como tales. Se pueden encontrar diferentes modelos de diversos autores que han propuesto

su definición de lo que serían los pilares básicos. Quizá uno de los más populares es el propuesto por Damon Edwards y John Willis, conocido por sus siglas CAMS (*culture*, *automation*, *measurement*, *sharing*) (figura 3).

Dicho modelo identifica como base de DevOps los cuatro puntos siguientes:

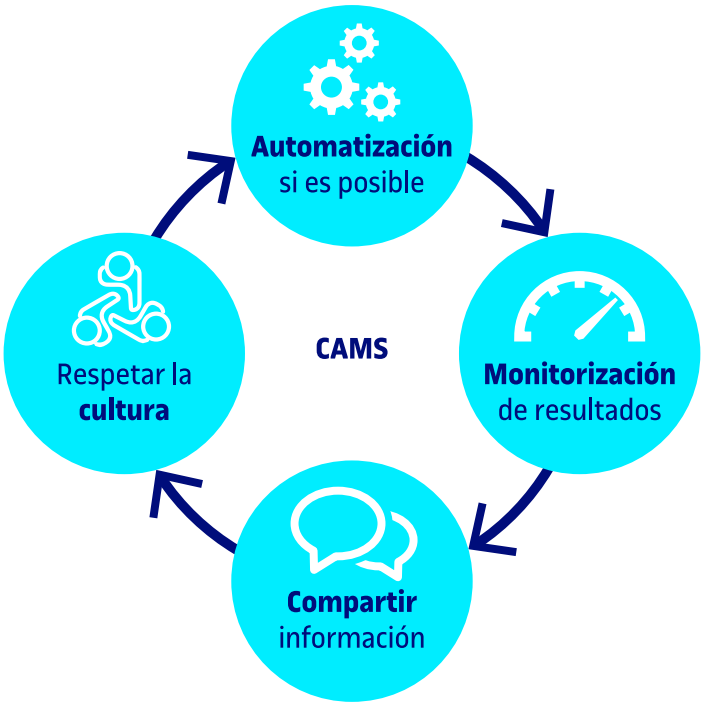
1) Cultura. Aplicar DevOps significa adoptar una nueva cultura de trabajo. Esto supone un cambio en la mentalidad; es necesario aprender a trabajar de modo diferente.

2) Automatización. Automatizar siempre que se pueda, especialmente aquellas tareas repetitivas que supongan una pérdida de tiempo útil.

3) Monitorización. Aplicar monitorización a lo largo de todo el proceso. Analizar la información obtenida para ver qué funciona bien y qué cosas se pueden mejorar.

4) Compartir, principalmente información. Es necesario que exista una buena comunicación entre equipos, de tal manera que el conocimiento circule de uno a otro con facilidad y exista una retroalimentación del conocimiento aprendido.

Figura 3. Pilares básicos de DevOps (CAMS)



Fuente: elaboración propia a partir de Telehouse (2016, 1 de marzo). «DevOps: how a culture of empathy creates massive productivity» [en línea]. *Telehouse.* <https://www.telehouse.com/2016/03/devops-how-a-culture-of-empathy-creates-massive-productivity>

Complementariamente a los principios definidos con anterioridad, existe una serie de buenas prácticas que es necesario aplicar para una correcta implementación de DevOps (figura 4). Las más recomendables son las siguientes:

1) Producto mínimo viable. Entregar el producto o la aplicación viable más pequeño posible y luego mejorarlo con lanzamientos rápidos posteriores, en función de los comentarios y entregando valor de manera incremental.

2) Equipos dedicados. Formar equipos dedicados y multifuncionales, evitar la especialización excesiva y asegurarse de que no se vean interrumpidos por proyectos paralelos.

3) Arquitectura de acoplamiento flexible. Utilizar un acoplamiento de arquitectura flexible dentro de las aplicaciones y entre estas para reducir la complejidad y permitir la entrega en pequeños incrementos.

4) Minimizar las transferencias, maximizar el flujo. Eliminar los pasos innecesarios, los retrasos y la fricción entre los pasos para aumentar el flujo de trabajo.

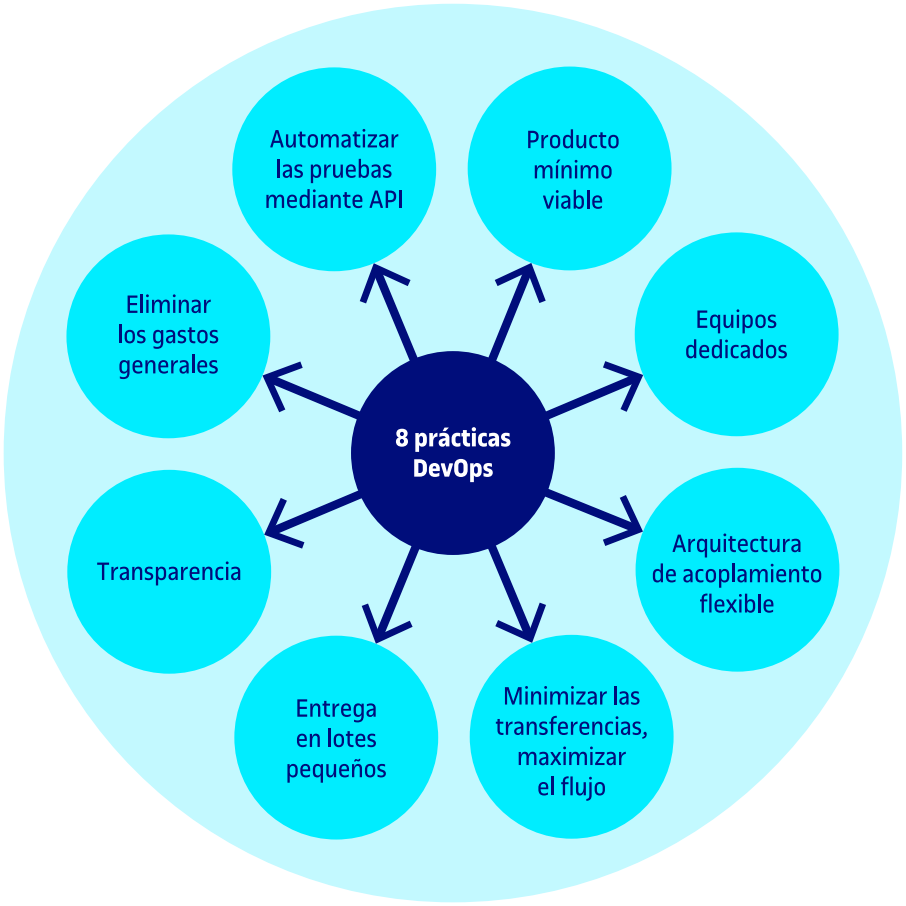
5) Entrega en lotes pequeños. La entrega en lotes más pequeños ayuda a exponer primero los problemas más inciertos y obtener a continuación comentarios sobre los modelos de uso más valiosos antes.

6) Transparencia. La transparencia en el progreso ayuda a todos a ver dónde se encuentran en cualquier momento y también a visibilizar el riesgo sin disminuir la productividad.

7) Eliminar los gastos generales. Eliminar los gastos generales de los informes de estado periódicos y manuales y las reuniones de estado, que consumen un tiempo excesivo y no ayudan en la entrega de valor.

8) Automatizar las pruebas mediante API. Reducir las pruebas manuales para mejorar la velocidad de entrega, la calidad y la precisión de las pruebas, y así reducir también los costes.

Figura 4. Buenas prácticas en la implementación de DevOps



Fuente: elaboración propia a partir de Contributor (2014, 22 de octubre). «Eight critical DevOps practices: innovate, deliver, repeat» [en línea]. *DevOps.com*. <https://devops.com/eight-critical-devops-practices-innovate-deliver-repeat>

1.2. Ciclo de vida

A diferencia del tradicional modelo de desarrollo en cascada (*waterfall*), que seguía un flujo lineal, el ciclo de vida en la cultura DevOps adquiere forma de **bucle**. Dicho ciclo se va repitiendo de manera constante y en periodos cortos de tiempo, a lo largo de todo el proceso de desarrollo. En cada una de las iteraciones del ciclo se realiza una entrega en el producto final.

El ciclo de vida DevOps está dividido en ocho etapas claramente definidas, en cada una de las cuales se realiza una tarea específica (figura 5). Las fases que podemos diferenciar son las siguientes:

1) Planificar. Se define el conjunto mínimo de funcionalidades que aportan valor en cada iteración y los criterios de aceptación que deben cumplirse. Es clave la comunicación entre el equipo de negocio y el equipo de desarrollo.

2) Codificar. Se comienza a desarrollar el código fuente del proyecto. Estos desarrollos avanzan en pequeños procesos en el ciclo de desarrollo. Por parte de operaciones, se empieza la construcción de las automatizaciones necesarias.

3) Construir. El objetivo es construir la aplicación con la integración de diversas piezas de código provenientes de la fase anterior. Una vez construida la aplicación, es el momento de crear el artefacto final que compone el software, incluidas las nuevas funcionalidades.

En esta fase del ciclo de vida de DevOps, el desarrollo de software es continuo e iterativo; cada parte avanza en el proceso. Esto es beneficioso para el equipo, ya que acelera el proceso y asegura la entrega.

4) Verificar. El equipo de calidad o QA identifica y corrige errores en la nueva pieza de código; al mismo tiempo, se empiezan a definir las pruebas que se deberán realizar al código fuente para asegurar que cumple con la especificación funcional.

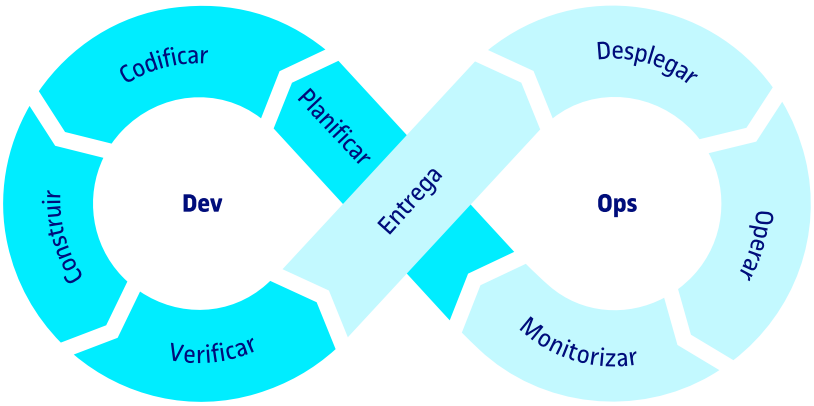
5) Entrega (*release*). Una vez que nuestra aplicación ha pasado todas las pruebas, podemos crear una versión de nuestro software. Esta versión se marca para indicar que ha pasado todas las validaciones previas y se podría poner a disposición de los usuarios en un futuro.

6) Desplegar. El proceso de implementación se lleva a cabo de manera continua. Se realiza de tal modo que cualquier cambio realizado en cualquier momento en el código no debe afectar al funcionamiento de la aplicación o producto existente.

7) Operar. El equipo de operaciones mide los recursos disponibles y los ajusta en función de las nuevas necesidades del software. Se elaboran nuevas métricas con las que supervisar el sistema y se configura la infraestructura para asegurar que todo funciona correctamente (gestión de escalado, balanceo de carga etc.).

8) Monitorizar. Por último, se establecen cuáles serán los parámetros que hay que supervisar de la aplicación. Se recopila toda la información recogida a lo largo de un periodo de tiempo para realizar los ajustes necesarios en la siguiente fase de planificación. Por su parte, el equipo de operaciones define las métricas para monitorizar y controlar el estado de salud de las aplicaciones y su infraestructura.

Figura 5. Etapas del ciclo de vida de DevOps



Fuente: elaboración propia a partir de Irma Harlann (2017, 28 de abril). «DevOps is a culture, not a role!» [en línea]. *neonrocket.medium.com*. <https://neonrocket.medium.com/devops-is-a-culture-not-a-role-be1bed149b0>

1.3. Metodologías

Existen varias metodologías en DevOps comunes que las organizaciones utilizan para acelerar y mejorar el desarrollo y las publicaciones de productos. Normalmente, se presentan como prácticas y metodologías de desarrollo de software. Entre las más populares encontramos las siguientes:

- **Scrum**: define el modo como los miembros de un equipo deben colaborar para conseguir entre todos acelerar los proyectos de desarrollo y control de calidad. Las prácticas de Scrum incluyen flujos de trabajo principales y terminología específica (reunión de planificación de *sprint*, *scrum* diario, revisión del *sprint*, retrospectiva del *sprint*) y roles designados (*scrum master*, propietario del producto o *product owner*).

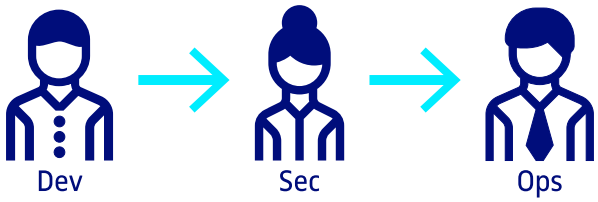
- **Kanban**: se originó a partir de las eficiencias que se alcanzaron en la fábrica de Toyota. Kanban prescribe que el estado «en curso» (WIP, del inglés *work in progress*) de un proyecto de software debe controlarse en un tablero Kanban.

- **Agile**: los anteriores métodos de desarrollo de software Agile siguen teniendo una gran influencia en las herramientas y las prácticas de DevOps. Muchos de estos métodos, incluidos Scrum y Kanban, han incorporado elementos de la programación Agile. Algunas de estas prácticas están asociadas a una mayor capacidad de respuesta a los continuos cambios en requisitos y necesidades, los requerimientos de documentación en forma de casos prácticos, la realización de reuniones diarias para ponerse al día y la incorporación de comunicación continua con los clientes para conocer sus opiniones. En Agile también se estipulan ciclos de desarrollo de software más cortos, en lugar de los tradicionales métodos de desarrollo en cascada que se prolongaban en el tiempo.

2. Seguridad en DevOps

Actualmente, en el marco de trabajo en colaboración de DevOps, la seguridad es una responsabilidad compartida e integrada durante todo el proceso. Puesto que es un enfoque tan importante, se estableció el término *DevSecOps* para enfatizar la necesidad de crear una base de seguridad en las iniciativas de DevOps (figura 6).

Figura 6. DevSecOps: seguridad en DevOps



Fuente: elaboración propia a partir de <https://www.redhat.com/es/topics/devops/what-is-devsecops>

Además de los beneficios que aporta DevOps, DevSecOps introduce la **automatización de la seguridad**, así como nuevos procesos que se incluyen dentro del flujo de DevOps.

DevSecOps no solo añade elementos de seguridad, sino que, como hemos comentado anteriormente, hace de la seguridad una parte integral dentro de todo el proceso, desde el inicio hasta el final e implicando a todos los equipos.

Como consecuencia, el equipo de seguridad se compromete mucho más con el resto de los equipos que intervienen en el SDLC, incluidos desarrollo y operaciones (figura 7). Esto elimina fricciones, ya que la tensión natural que existe entre velocidad y seguridad se comparte por todos los equipos.

Figura 7. DevSecOps: integración de la seguridad en todo el proceso

Fuente: elaboración propia a partir de Sumit Siddharth (s. f.). «Herramientas open-source para adoptar DevSecOps» [en línea]. *Claranet*. <https://www.claranet.es/blog/herramientas-open-source-para-adoptar-devsecops>

Consecuentemente, estamos **desplazando la seguridad más hacia la izquierda** en la canalización o *pipeline* de DevOps. Dicho de otra manera, DevSecOps es la metodología para integrar herramientas de seguridad en el proceso DevOps de manera automatizada. Asimismo, lo que también es importante es poseer los conocimientos necesarios para usar estas herramientas. Esto lleva necesariamente a un cambio cultural en el normal funcionamiento de DevOps: los equipos deben ser formados para que entiendan qué herramientas tienen a su disposición, qué pueden lograr y cómo funcionan, lo que permite colaborar entre equipos de manera más eficiente, hasta crear una «cultura de la seguridad» más robusta. Como resultado, este entorno de **seguridad automatizada multicultural, multidisciplinar y transversal** provoca que la seguridad sea un tema que compete a todos y no solo a un único equipo. Este es uno de los motores principales del DevSecOps.

Entre las principales ventajas de incorporar la seguridad desde la fase de desarrollo se encontrarían las siguientes:

- El **proceso** apenas se alarga, ya que las cuestiones de seguridad también pueden someterse a la automatización y a los mecanismos de seguimiento.

- Los distintos **equipos** encargados del desarrollo y de las operaciones se familiarizan con los factores de seguridad y los aplican desde el inicio, con lo que se prevén posibles brechas de seguridad.

- Versiones de **software** seguras y estables, que pueden pasar directamente al entorno de producción de la organización.

- La seguridad también se automatiza, lo que facilita el **desarrollo** ágil.

DevSecOps implica integrar la seguridad a lo largo de todo el ciclo de vida DevOps, empezando desde las fases iniciales. Para ello, se puede dividir la tarea en cuatro etapas, en las que se llevarán a cabo las prácticas necesarias (figura 8). Dichas etapas se detallan a continuación.

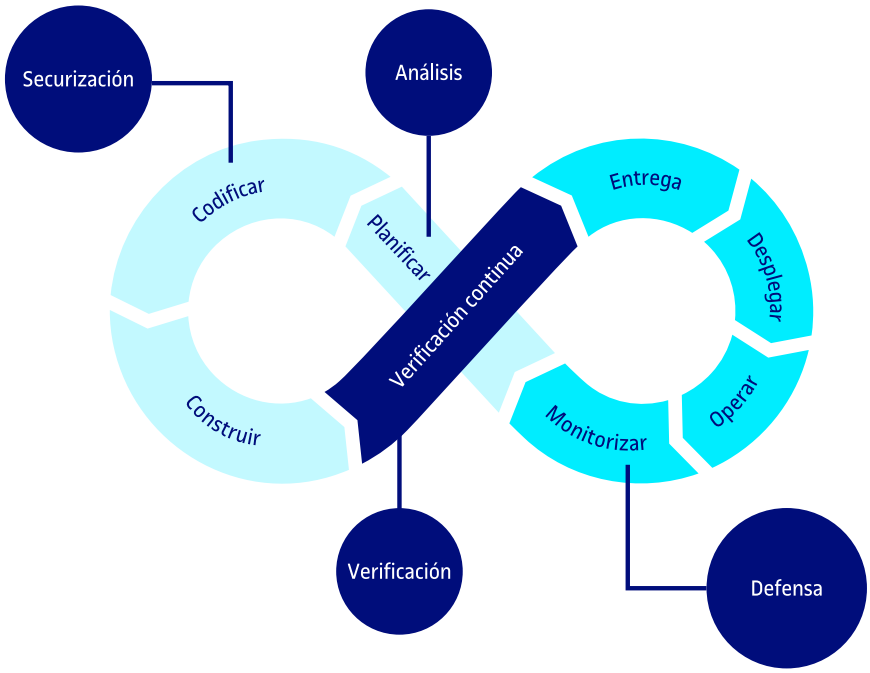
1) Análisis: consiste en la planificación y el análisis de los riesgos existentes. Se identifican las amenazas y vulnerabilidades y se priorizan aquellas más críticas.

2) Securización: se implementa una estrategia de securización y se toman las medidas necesarias a la hora de desarrollarla para reducir al máximo los posibles riesgos, en función de los resultados del análisis realizado en la etapa anterior.

3) Verificación: se basa en verificar las medidas de seguridad implementadas y detectar posibles vulnerabilidades en el código. Para ello, se utilizan herramientas automatizadas de análisis de código, tanto estáticas como dinámicas, así como pruebas de penetración.

4) Defensa: una vez desplegada nuestra aplicación, la última etapa consiste en utilizar herramientas de monitorización en tiempo de ejecución que permitan detectar a tiempo posibles ataques. Es común el uso de alertas que se activen en el caso de un comportamiento anómalo de la aplicación.

Figura 8. Etapas del ciclo de vida de DevSecOps



Fuente: elaboración propia

3. Caso de estudio: DevSecOps *pipeline*

En este apartado se presenta un caso de estudio detallado basado en la guía ***OWASP DevSecOps pipeline***, donde se van añadiendo una serie de herramientas que permiten ir mejorando la seguridad de nuestro flujo de trabajo en DevOps.

Enlace recomendado

Podéis consultar la guía *OWASP DevSecOps* en el enlace siguiente:

<https://owasp.org/www-project-devsecops-guideline>

> La guía *OWASP DevSecOps* se centra en explicar cómo podemos implementar una canalización o *pipeline* segura, utilizar las mejores prácticas e introducir herramientas que podamos utilizar en este asunto.

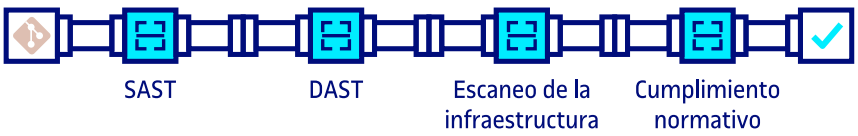
El proyecto intenta ayudar a promover la cultura de seguridad de cambio a la izquierda en nuestro proceso de desarrollo.

Esta guía puede ayudar a cualquier organización que ya disponga de un flujo de desarrollo o, en otras palabras, de un flujo de DevOps.

En la figura 9 se pueden apreciar las diferentes etapas o pasos que debemos considerar implementar en una canalización básica:

1) Gestión de secretos y credenciales en los repositorios.
2) SAST (prueba de seguridad de aplicaciones estáticas).
3) DAST (prueba de seguridad de aplicaciones dinámicas).
4) Escaneo de la infraestructura.
5) Cumplimiento normativo.

Figura 9. Etapas de una canalización básica



Fuente: elaboración propia a partir de <https://github.com/OWASP/DevSecOpsGuideline>

Podemos personalizar las etapas de nuestro *pipeline* de acuerdo con nuestro ciclo de vida de desarrollo de software (SDLC) o arquitectura de software y agregar automatización progresivamente.

CI/CD es una ventaja para DevSecOps, ya que es un punto perfecto para aplicar medidas y controles de seguridad. Sin embargo, cuando utilicemos herramientas de CI/CD para proporcionar automatización, deberemos tener en cuenta que también habrá que aplicar controles de seguridad en el software de construcción, implementación y automatización.

3.1. Gestión de secretos y credenciales en los repositorios

Respecto a la gestión de secretos, diferenciaremos dos fases diferentes:

1) Cuando **se filtra por error información sensible** (credenciales de nube pública, como claves de AWS o Azure, *tokens* de acceso o claves SSH) a través de repositorios de código fuente públicos debido a subidas de código (*commits*) accidentales, es decir, información confidencial que no debería formar parte ni siquiera de nuestro flujo de trabajo dentro de DevSecOps. Para evitar esta situación, podemos aplicar algunas comprobaciones previas a la subida del código al repositorio, para lo que disponemos de las herramientas de código abierto siguientes:

- **Talisman**: https://github.com/thoughtworks/talisman
- **Git Hooks**: https://githooks.com
- **Git Secret**: https://git-secret.io
- **Pre Commit**: https://pre-commit.com
- **Detect Secrets**: https://github.com/Yelp/detect-secrets
- **Git Hound**: https://github.com/ezekg/git-hound
- **Truffle Hog**: https://github.com/dxa4481/truffleHog

Truffle Hog

En este ejemplo de la aplicación Truffle Hog se puede apreciar que se debería evitar subir la clave secreta de la cuenta de una nube pública presente en el código fuente de la aplicación (figura 10).

Figura 10. Ejemplo de uso de la herramienta Truffle Hog

```
Date: 2014-04-21 18:46:21
Branch: master
Commit: Removing aws keys

@@ -57,8 +57,8 @@ public class EurekaEVCacheTest extends AbstractEVCacheTest {
             //

             props.setProperty("      datacenter", "cloud");
-            props.setProperty("      awsAccessId", "<aws access id>");
-            props.setProperty("      awsSecretKey", "<aws secret key>");
+            props.setProperty("      awsAccessId", "AKIAJCK2WUHJ2653GNBQ");
+            props.setProperty("      awsSecretKey", "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");
             props.setProperty("      .appinfo.validateInstanceId", "false");

             props.setProperty("      .discovery.us-east-1.availabilityZones", "us-east-1c,us-east-1d,us-east-1e");
```

Fuente: <https://github.com/trufflesecurity/truffleHog>

2) Etapa marcada por la **automatización**, donde almacenar credenciales en archivos de configuración y variables de entorno para acceder a los diferentes microservicios es una práctica habitual que siguen muchos desarrolladores y administradores cuando se trabaja mediante un flujo de trabajo de DevOps. No obstante, almacenar credenciales en archivos o configuraciones puede llevar a exponer nuestras credenciales a un **público inadecuado**. Para solventar esta problemática, también existen diferentes herramientas que se encargan de la gestión de secretos de una manera segura; a continuación, se enumeran algunas de estas:

- **Hashicorp Vault**: https://www.vaultproject.io
- **Keywhiz**: https://square.github.io/keywhiz
- **EnvKey**: https://www.envkey.com

- **Confidant**: https://github.com/lyft/confidant
- **AWS Secrets Manager**: https://aws.amazon.com/secrets-manager

Vault

Vault es una herramienta para acceder a secretos de manera segura. Un secreto es cualquier cosa a la que se desee restringir el acceso, como claves API, contraseñas, certificados, etc. Vault proporciona una interfaz unificada para cualquier secreto (figura 11), a la vez que proporciona un estricto control de acceso y genera un registro de auditoría detallado. Las características principales de Vault son las siguientes:

- Almacenamiento seguro de secretos.
- Secretos dinámicos.
- Cifrado de datos.
- Reserva y renovación de secretos.
- Revocación de secretos.

Figura 11. Ejemplo de creación de un secreto con Vault

```
~
❯ vault kv put secret/db username=root
Key              Value
---              -----
created_time     2021-05-04T14:04:48.297067Z
deletion_time    n/a
destroyed        false
version          1
```

Fuente: <https://www.vaultproject.io>

3.2. Pruebas de análisis estático o SAST

Las pruebas de seguridad de aplicaciones estáticas (SAST, *static application security testing*), o análisis estático, son una metodología de prueba que analiza el código fuente para encontrar vulnerabilidades de seguridad que provocan que las aplicaciones de una organización sean susceptibles de ser atacadas. SAST escanea una aplicación **antes de que se compile el código**. A este tipo de pruebas también se las conoce como *pruebas de caja blanca*.

SAST tiene lugar al inicio del ciclo de vida del desarrollo de software (SDLC), ya que no requiere una aplicación funcional y puede tener lugar sin que se ejecute código fuente.

SAST ayuda a los desarrolladores a identificar vulnerabilidades en las etapas iniciales de desarrollo y a resolver problemas rápidamente sin romper compilaciones o transmitir vulnerabilidades a la versión final de la aplicación.

Las herramientas SAST aportan a los desarrolladores **retroalimentación en tiempo real** mientras codifican, lo que los ayuda a solucionar problemas antes de pasar el código a la siguiente fase del SDLC. Esto evita que los problemas relacionados con la seguridad aparezcan de manera tardía. Algunas herramientas indican incluso la ubicación exacta de dichas vulnerabilidades y resaltan el código de riesgo. Las herramientas también son capaces de proporcionar una

guía detallada sobre cómo solucionar problemas y el mejor lugar en el código donde hacerlo, por lo que no se requiere mucha experiencia en el campo de la seguridad por parte del desarrollador.

Los desarrolladores también pueden crear los **informes personalizados** que necesiten con las herramientas SAST. Es importante tener en cuenta que las herramientas SAST deben ejecutarse en la aplicación de manera periódica, por ejemplo, durante las compilaciones diarias o mensuales, o cada vez que se registra el código o durante la publicación de un código.

A continuación, la tabla 1 muestra diferentes herramientas de código abierto que pueden utilizarse para pruebas SAST. Como comprobaréis, estas también dependen del lenguaje de programación utilizado.

Tabla 1. Herramientas de código abierto para pruebas SAST

Herramienta	Java	.NET	C, C++	Python	Golang	Ruby/Rails	PHP
Bandit	X	X	X	√	X	X	X
Brakeman	X	X	X	X	X	√	X
Codesake Dawn	X	X	X	X	X	√	X
FindBugs	√	X	X	X	X	X	X
FindSecBugs	√	X	X	X	X	X	X
Flawfinder	X	X	√	X	X	X	X
Graudit	√	√	X	√	X	√	√
LGTM	√	X	√	√	X	X	X
Progpilot	X	X	X	X	X	X	√
PreFast (Microsoft)	X	X	√	X	X	X	X
Puma Scan	X	√	X	X	X	X	X
.NET Security Guard	X	√	X	X	X	X	X
RIPS	X	X	X	X	X	X	√
phpcs-security-adit	X	X	X	X	X	X	√
SonarQube	√	√	√	√	√	√	√
VisualCodeGrepper (VCG)	X	X	√	X	X	X	√

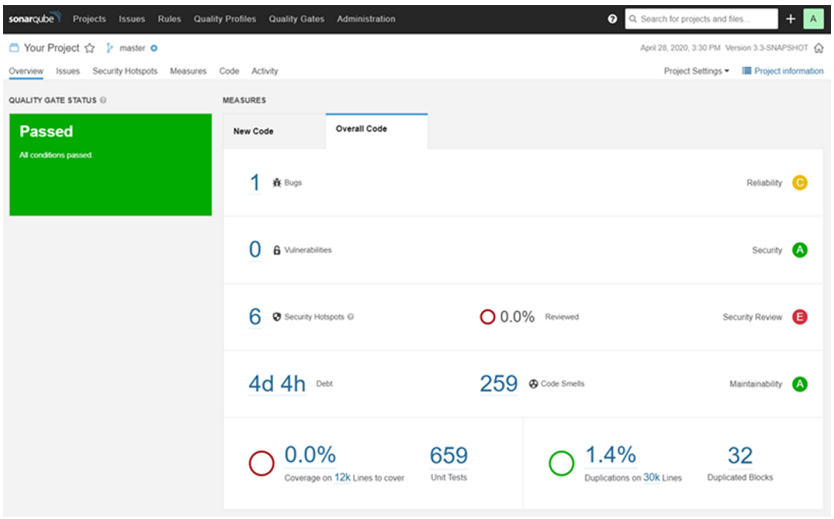
Fuente: <https://owasp.org/www-community/Source_Code_Analysis_Tools>

Entre las herramientas más populares que podemos utilizar para realizar un análisis estático de nuestro código fuente se encuentra la aplicación SonarQube. A continuación, se detallan las principales características de este software.

SonarQube

SonarQube es una de las herramientas más populares de revisión automática de código fuente para detectar errores, vulnerabilidades y malos olores de código (*code smells*). Se trata de software libre y utiliza diversas herramientas de análisis estático de código fuente, como Checkstyle, PMD o FindBugs, para obtener métricas que pueden ayudar a mejorar la calidad del código de un programa (figura 12). Soporta múltiples lenguajes de programación y permite configurar diferentes tipos de reglas en función de los requisitos de cada caso. Se integra perfectamente con Jenkins.

Figura 12. Pantalla principal de SonarQube



Fuente: <https://www.sonarqube.org>

Si queréis profundizar en la herramienta SonarQube, a continuación os proporcionamos un fichero `docker-compose.yml` que facilita la ejecución de SonarQube en contenedores Docker a través de Docker Compose:

```
version: "3"

services:
  sonarqube:
    image: sonarqube
    container_name: sonarqube
    hostname: sonarqube
    ports:
      - 9000:9000
    environment:
      - sonar.jdbc.username=sonar
      - sonar.jdbc.password=password
      - sonar.jdbc.url=jdbc:postgresql://db:5432/sonar
    volumes:
      - /opt/sonarqube/logs:/opt/sonarqube/logs
      - /opt/sonarqube/data:/opt/sonarqube/data
      - /opt/sonarqube/extensions:/opt/sonarqube/extensions

  db:
    image: postgres
    container_name: db
    hostname: db
    environment:
      - POSTGRES_USER=sonar
      - POSTGRES_PASSWORD=password
    volumes:
      - pg_db:/var/lib/postgresql
      - pg_data:/var/lib/postgresql/data
    ulimits:
      nofile:
        soft: 65536
        hard: 65536
```

```
volumes:
  pg_db:
    driver: local
  pg_data:
    driver: local
```

Enlace recomendado

Para saber más sobre Docker Compose y su instalación, podéis consultar el enlace siguiente: <https://docs.docker.com/compose>

3.3. Pruebas de análisis dinámico o DAST

La prueba de seguridad de aplicaciones dinámicas (DAST, *dynamic application security testing*) es el proceso de analizar una aplicación web a través de su frontal web (o *front-end*) para encontrar vulnerabilidades mediante **ataques simulados**. Este tipo de enfoque evalúa la aplicación desde «afuera hacia adentro» atacando una aplicación, como lo podría hacer un usuario malintencionado. Una vez que un analizador DAST realiza estos ataques, busca resultados que no forman parte del conjunto de los resultados esperado e identifica vulnerabilidades de seguridad. También se conoce como *prueba de caja negra*.

Los escáneres de aplicaciones web son una parte importante de la evaluación de vulnerabilidades de una aplicación web. La mayoría de estos tiene acceso vía API o CLI, que se pueden aprovechar para iniciar análisis en las aplicaciones objetivo.

Algunas de las herramientas de código abierto para el testeo de seguridad de análisis dinámico son las siguientes:

- **OWASP ZAP**: https://www.zaproxy.org
- **Arachni Scanner**: http://www.arachni-scanner.com
- **Nikto**: https://cirt.net/Nikto2

Enlace recomendado

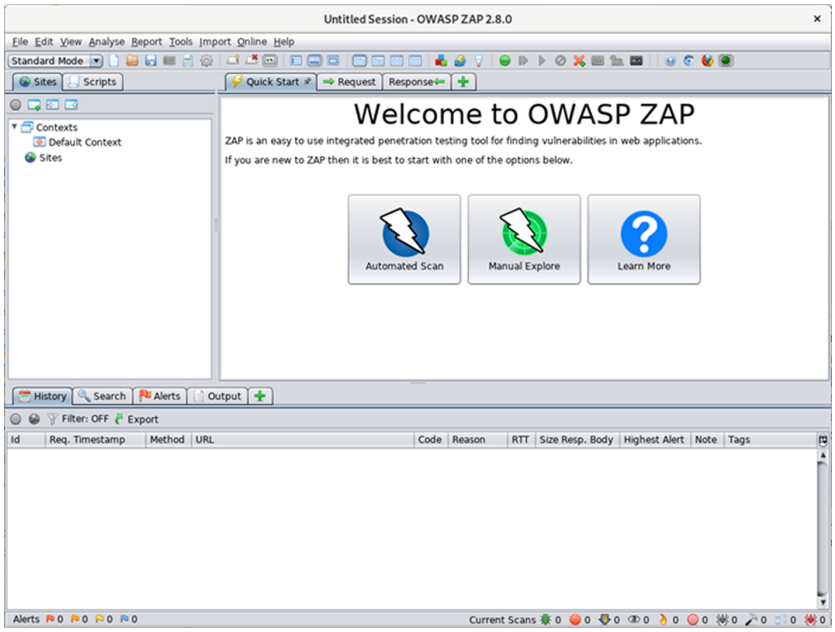
Para saber más sobre el funcionamiento de OWASP ZAP en Docker, podéis consultar el enlace siguiente:
<https://www.zaproxy.org/docs/ docker/about>

OWASP ZAP

OWASP ZAP (Zed Attack Proxy) es el escáner web de vulnerabilidades más popular y utilizado; se trata de una herramienta de código abierto y multiplataforma (figura 13). Es uno de los proyectos más activos de OWASP. Entre las características disponibles se encuentran las siguientes:

- Servidor *proxy* de interceptación.
- Rastreadores web tradicionales y por AJAX.
- Escáner automatizado.
- Escáner pasivo.
- Navegación forzada.
- *Fuzzer*.
- Soporte para WebSocket.
- Lenguajes de *scripting* y compatibilidad con Plug-n-H.

Figura 13. Aplicación OWASP ZAP



Fuente: <https://www.zaproxy.org>

Para profundizar en la herramienta OWASP ZAP, podéis probar su funcionamiento mediante Docker con el siguiente comando:

```
docker run --detach --name zap -u zap -v "/opt/dast/reports":/zap/reports/:rw
-i owasp/zap2docker-stable zap.sh -daemon -host 0.0.0.0 -port 8080   -config
api.addrs.addr.name=.* -config api.addrs.addr.regex=true   -config
api.disablekey=true
```

Una vez que tenemos la herramienta ZAP corriendo en Docker podemos realizar un análisis dinámico de nuestra aplicación con el siguiente comando:

```
docker exec zap zap-cli --verbose quick-scan http://ipaddress.app -l Medium
```

3.4. Pruebas de análisis interactivo o IAST

Las pruebas de seguridad de aplicaciones interactivas (IAST, *interactive application security testing*) combinan los beneficios de las metodologías de caja negra y caja blanca. También se conocen como *pruebas de caja gris*. IAST es una tecnología relativamente nueva para encontrar fallos de seguridad en aplicaciones. Al igual que las herramientas DAST, las herramientas IAST analizan las aplicaciones **en ejecución**.

Desde un punto de vista de la arquitectura, las herramientas IAST se instalan como parte de la infraestructura que soporta las aplicaciones web, ya que se instalan como un complemento del servidor de aplicaciones. A este enfoque se le denomina ***instrumentación***, y se realiza mediante un componente conocido como ***agente***.

Una vez que el agente está instalado, incorpora sensores automáticos de seguridad en puntos críticos de la ejecución de la aplicación. Estos sensores monitorizan las peticiones y respuestas al servidor, las librerías externas que la aplicación utiliza, el acceso a recursos del servidor, y operaciones de datos como el acceso a bases de datos. Esta **cobertura de amplio espectro** supera con creces la visibilidad que nos proporcionan las herramientas SAST y DAST.

El enfoque IAST ofrece numerosos beneficios, que se describen a continuación:

- **Cobertura completa**, ya que permite analizar toda la aplicación: tanto el código propio como el código externo, también conocido como *dependencias*.

- **Flexibilidad**, dado que una única herramienta se adapta a todos los entornos: desarrollo, aseguramiento de calidad o *testing* y producción.

- **Alta precisión**, ya que la combinación de visibilidad estática y dinámica permite encontrar más vulnerabilidades, sin caer en la trampa de los falsos positivos.

- **Información detallada de cada vulnerabilidad**: estática (en lo referente al código fuente) y dinámica (en lo referente a cómo el código responde a las peticiones de los usuarios).

- **Reducción del plazo de verificación** de la seguridad de las aplicaciones, por lo que acelera el desarrollo de aplicaciones seguras.

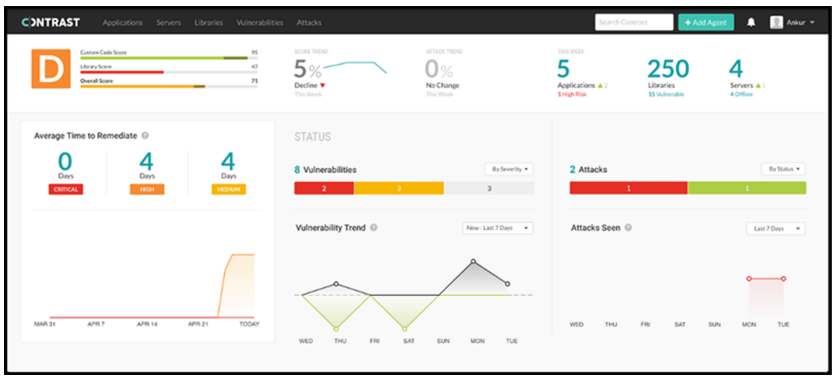
- **Adaptación perfecta a las nuevas metodologías** de desarrollo, incluidas las metodologías ágiles y DevSecOps, dado que favorece la introducción de automatismos y reduce la dependencia en operaciones manuales.

Una de las herramientas destacadas que podemos utilizar para la realización de pruebas interactivas sería Constrast Community Edition.

Constrast Community Edition

Contrast es una plataforma DevOps-Native ideada para mejorar la seguridad de las aplicaciones web (figura 14). La versión Community Edition ofrece acceso casi completo a los productos comerciales de Contrast (Assess, OSS y Protect), y los desarrolladores pueden realizar pruebas de seguridad de aplicaciones interactivas (IAST) y análisis de composición de software (SCA), así como disponer de soluciones de autoprotección de aplicaciones en tiempo de ejecución (RASP). El principal inconveniente de la versión Community Edition es que los desarrolladores solo pueden instrumentar y proteger una única aplicación y con la limitación de los siguientes lenguajes de programación: Java o .NET Core.

Figura 14. Pantalla principal de Constrast Community Edition



Fuente: <https://www.contrastsecurity.com/contrast-community-edition>

3.5. Escaneo de la infraestructura

La **infraestructura como código (IaC)** permite gestionar y preparar la infraestructura con código, en lugar de hacerlo mediante procesos manuales.

Con este tipo de infraestructura, se crean archivos de configuración que contienen las especificaciones que esta necesita, lo cual facilita la edición y la distribución de las configuraciones. Asimismo, garantiza que siempre se prepare el mismo entorno.

La IaC es una parte fundamental de la implementación de las prácticas de DevOps y de la integración y distribución continuas (CI/CD).

> La IaC libera a los desarrolladores de tener que realizar la mayor parte del trabajo de preparación, ya que solo deben ejecutar un *script* y la infraestructura estará lista para funcionar.

Las implementaciones de aplicaciones no necesitan esperar a la infraestructura, y los administradores de sistemas no tienen que gestionar procesos manuales que consumen mucho tiempo.

Si además queremos ofrecer un enfoque DevSecOps, también deberemos preocuparnos de la **seguridad de la infraestructura**. Las herramientas de esta categoría detectan y reparan automáticamente varias vulnerabilidades de seguridad y problemas de configuración para varios aspectos de los entornos en la nube. Van desde la automatización basada en eventos hasta la gestión de la configuración, la infraestructura como código (IaC) y las herramientas de gestión de la configuración en la nube, como las plataformas de protección de carga de trabajo en la nube (CWPP).

Una de las herramientas más populares actualmente en el mercado para el uso de infraestructura como código (IaC) es Terraform. A continuación, se detallan algunas de sus características principales.

Enlace recomendado

Para saber más sobre Terraform, podéis consultar el enlace siguiente:

<https://www.terraform.io/intro/ index.html>

Terraform

Terraform es una herramienta de código abierto desarrollada por HashiCorp. Se utiliza para definir y aprovisionar la infraestructura completa utilizando un lenguaje declarativo.

Es una herramienta de aprovisionamiento de infraestructura en la que se puede almacenar la configuración de su infraestructura en la nube como código. Es muy similar a herramientas como CloudFormation, utilizada para automatizar la infraestructura de AWS. Con Terraform, podemos utilizarla para otras plataformas también en la nube.

A continuación, se muestran algunos de los beneficios de usar Terraform:

- Realiza orquestación, no solo gestión de configuración.
- Admite múltiples proveedores de *cloud*, como AWS, Azure, GCP, Digital Ocean y otros.
- Proporciona una infraestructura inmutable donde la configuración cambia sin problemas.
- Utiliza un lenguaje fácil de entender, HCL (lenguaje de configuración de HashiCorp).
- Es fácilmente portátil a cualquier otro proveedor.
- Admite arquitectura de solo cliente, por lo que no es necesario administrar la configuración adicional en un servidor.

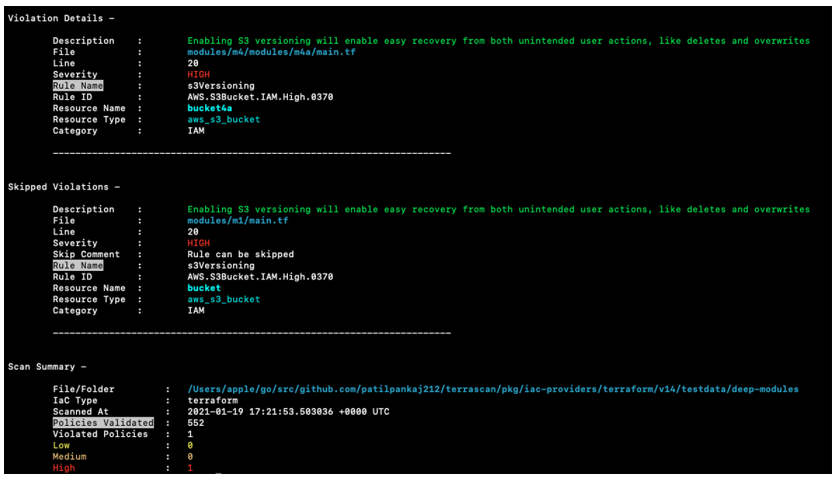
El uso de la infraestructura como código (IaC) también implica riesgos de seguridad que hay que tener presente. En concreto, para Terraform disponemos de herramientas como Terrascan, que se encargan de analizar el código que hemos creado para implementar la infraestructura en busca de posibles brechas de seguridad.

Terrascan

Se trata de un analizador de código estático para infraestructura como código (figura 15). Terrascan permite:

- Escanear de forma sencilla la infraestructura como código para detectar errores de configuración.
- Supervisar la infraestructura de nube aprovisionada para detectar cambios de configuración y permite volver a una posición segura.
- Detectar vulnerabilidades de seguridad y violaciones de cumplimiento.
- Mitigar los riesgos antes de aprovisionar la infraestructura nativa de la nube.
- Flexibilidad para ejecutarse localmente o integrarse con su CI/CD.
- Disponer de más de quinientas políticas para las mejores prácticas de seguridad.
- Escanear ficheros de Terraform (HCL2), Kubernetes (JSON/YAML), Helm v3, Kustomize y Dockerfiles.
- Soporte para AWS, Azure, GCP, Kubernetes, Dockerfile y GitHub.
- Integrarse con el escaneo de vulnerabilidades de imágenes de Docker para registros de contenedores de AWS, Azure y GCP.

Figura 15. Ejemplo de uso de la herramienta Terrascan



Fuente: <https://github.com/accurics/terrascan>

3.5.1. Gestión de la configuración

En los entornos empresariales actuales podemos encontrar distintas formas de operar:

- Algunos poseen máquinas físicas que ejecutan el rol de servidor y ofrecen acceso a las aplicaciones desarrolladas. Esta forma de disposición de servidores es una visión clásica y simple, muy utilizada durante años, con obvias dificultades durante procesos habituales de migraciones de entorno y cambios o ampliaciones exigidos en el hardware utilizado.

- Una visión más moderna con una perspectiva más amplia sobre los sistemas intentaría desvincular las máquinas físicas de los entornos de despliegue de servidores; estos sistemas resultan más flexibles al utilizar herramientas de virtualización como VMware, Xen, KVM y VirtualBox, entre otras.

Hemos descrito distintas visiones sobre los entornos de producción y cómo se pueden presentar, pero ¿y los equipos de los entornos de desarrollo? Imaginaos un equipo en el que cada miembro trabaja en su propio ordenador, algunos utilizando portátiles, otros con equipos de sobremesa, con distintas configuraciones, incluso con sistemas operativos distintos, ¿no cabría esperar problemas derivados de esta diversidad?

Para la filosofía DevOps, los **entornos de despliegue**, tanto en desarrollo como en producción, adquieren una especial importancia. No solo es importante poseer un entorno de servidor funcional en el que desplegar una aplicación; como hemos visto, muchos de los problemas durante la puesta en producción del software no se derivan de fallos durante el proceso de desarrollo, sino que se presentan durante el despliegue en entornos donde estos difieren en especificaciones: distintos sistemas operativos, versiones de software y paquetería, configuraciones de seguridad, etc.

Para solucionar esta necesidad de homogeneización de entornos aparecen herramientas como Chef y Puppet, que nos permitirán que los sistemas utilizados por el equipo de desarrollo sean exactamente iguales a los que se utilizarán *a posteriori* durante el proceso de despliegue en entornos de producción. De este modo se eliminará cualquier error de incompatibilidades y los entornos utilizados habrán sido testados durante las primeras fases del desarrollo del proyecto.

Las soluciones de **homogeneización de entornos** implican un cambio de concepto de todos los equipos implicados en el desarrollo y despliegue de un proyecto, ya que la coordinación para la construcción de un sistema inicial será una tarea compartida entre los equipos de diseño de software y los equipos de operaciones encargados de la gestión de máquinas y sistemas.

Existe una multitud de herramientas especializadas en la gestión de la configuración donde los DevOps pueden apoyarse. Las más conocidas son Puppet, Chef y Ansibleen. A continuación, describiremos Ansible como ejemplo de la utilización de estas herramientas enfocadas a la disposición de entornos homogéneos de desarrollo-producción.

Enlace recomendado

Para saber más sobre Ansible, podéis consultar el enlace siguiente:
<https://www.ansible.com/how-ansible-works>

Ansible

Ansible es una plataforma de software libre (GPLv3) desarrollada en Python y ofrecida comercialmente por AnsibleWorks. Ofrece una forma simple de automatización de procesos TI (por ejemplo, el aprovisionamiento de infraestructura, la gestión de la configuración, el despliegue de aplicaciones, la verificación de la seguridad y el cumplimiento normativo o la orquestación de despliegues). Puede configurar sistemas, realizar despliegues de software y llevar a cabo tareas avanzadas de despliegues continuos.

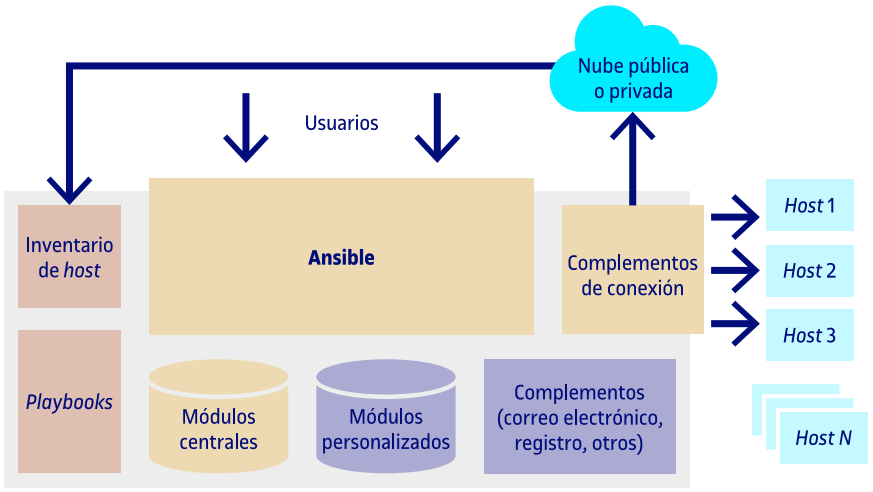
Estas son quizá sus características más importantes:

- Excelente rendimiento sin necesidad de instalación ni despliegues de agentes.
- Bajo coste operativo (*overhead* muy bajo).
- Método de autenticación por SSH (preferiblemente con claves) en paralelo.
- No necesita usuario *root* (permite la utilización de sudo).
- Permite utilizar comandos básicos.
- Acepta comandos en casi cualquier lenguaje de programación.

Ansible puede utilizarse en modo Ad-Hoc o mediante Playbook. El modo Ad-Hoc nos permitirá ejecutar comandos en una sola línea sobre todos los *hosts*. Este método es utilizado cuando se pretenden realizar tareas simples sobre todos nuestros *hosts*; para tareas más complejas, los *playbooks* resultarán mucho más cómodos (figura 16).

Los *playbooks* son ficheros escritos en formato YAML, que pueden presentarse en forma de un solo fichero o siguiendo un modelo estructurado; contienen todos los parámetros necesarios para realizar una determinada tarea sobre un grupo de servidores.

Figura 16. Flujo de datos en Ansible



Fuente: elaboración propia

3.5.2. Escaneo de imágenes

Los equipos de DevOps suelen implementar componentes mediante imágenes y contenedores de Docker. En un entorno de DevSecOps, una de las principales preocupaciones es **identificar vulnerabilidades en las imágenes de contenedores**, ya que es común que se extraigan de repositorios públicos o de otras fuentes no confiables, y porque las implementaciones de contenedores pueden escalar rápidamente, potencialmente escalando la superficie de ataque como bien.

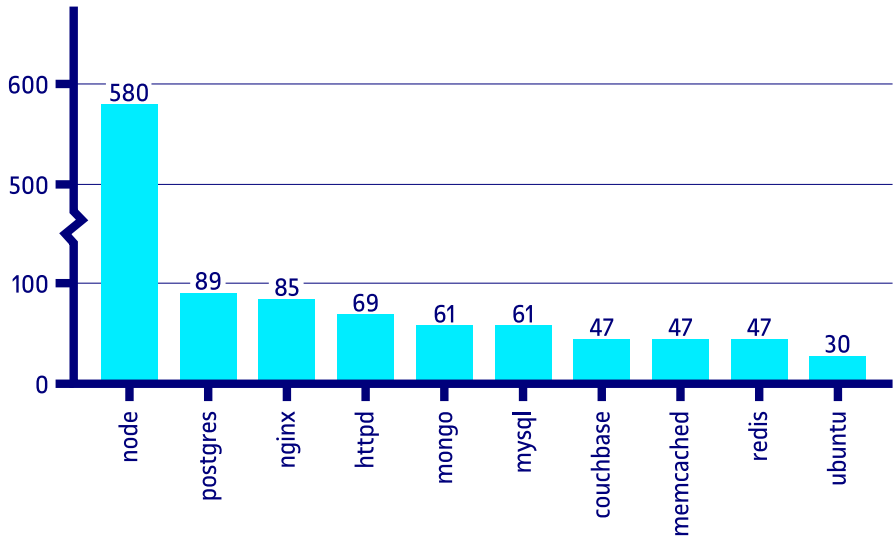
Las imágenes de Docker y las imágenes base en las que se fundamentan pueden contener muchos componentes de software que pueden estar desactualizados, sin parchear o con vulnerabilidades de seguridad.

> Los escáneres de imágenes de contenedores verifican que las imágenes contengan solo códigos y artefactos seguros y confiables, y que cumplan con las mejores prácticas de configuración segura.

Los procesos de DevSecOps que involucran contenedores deben incluir escaneo de imágenes y corrección en cada etapa de la canalización de CI/CD.

Por ejemplo, la figura 17 muestra las vulnerabilidades encontradas en algunas de las imágenes más utilizadas de Docker.

Figura 17. Número de vulnerabilidades en imágenes de Docker



Fuente: elaboración propia a partir de Liran Tal (2019, 26 de febrero). «Top ten most popular docker images each contain at least 30 vulnerabilities» [en línea]. *Snyk*. <https://snyk.io/blog/top-ten-most-popular-docker-images-each-contain-at-least-30-vulnerabilitie>

Para remediar este problema de seguridad, se proponen los siguientes escáneres de imágenes que permiten encontrar vulnerabilidades presentes en la imagen:

- **Clair**: https://github.com/coreos/clair
- **Anchore Engine**: https://github.com/anchore/anchore-engine
- **Trivy**: https://github.com/aquasecurity/trivy
- **Docker Scan**: https://docs.docker.com/engine/scan
- **Grype**: https://github.com/anchore/grype

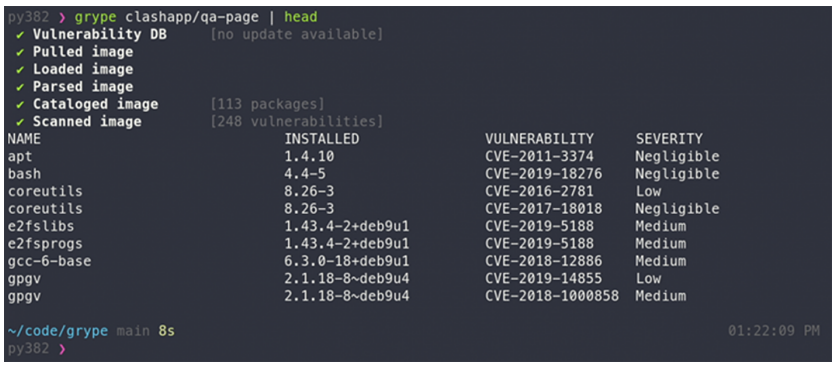
Finalmente, vamos a analizar con mayor detalle la herramienta de código abierto Grype.

Grype

Grype es un escáner de vulnerabilidades para imágenes de contenedores y sistemas de archivos (figura 18). La herramienta extrae una base de datos de vulnerabilidades derivadas del servicio Anchore Feed disponible públicamente. Esta base de datos se actualiza al comienzo de cada análisis, pero también se puede activar una actualización manualmente. Grype puede escanear una variedad de fuentes más allá de las que se encuentran en Docker. Las principales características de esta herramienta son las siguientes:

- Escaneo del contenido de una imagen de un contenedor o un sistema de archivos para encontrar vulnerabilidades conocidas.
- Búsqueda de vulnerabilidades para los principales paquetes de sistemas operativos (Alpine, BusyBox, CentOS/Red Hat, Debian y Ubuntu).
- Búsqueda de vulnerabilidades en librerías externas de los distintos lenguajes de programación (Ruby, Java, JavaScript, Python, etc.).
- Admisión de formatos de imagen Docker y OCI.

Figura 18. Ejemplo de uso del escáner de imágenes Grype



Fuente: elaboración propia a partir de <https://anchore.com/opensource>

La instalación de la herramienta Grype resulta muy sencilla, ya que únicamente necesitamos descargar el binario de la aplicación:

```
curl -sSfL https://raw.githubusercontent.com/anchore/grype/main/install.sh | sh
-s -- -b /usr/local/bin
```

A continuación, simplemente debemos actualizar la base de datos de vulnerabilidades de Grype con el comando:

```
grype db update
```

Podemos verificar el estado de la base de datos de vulnerabilidad ejecutando el comando:

```
grype db status
```

Finalmente, podemos realizar un análisis completo de una imagen de Docker con el comando:

```
grype -f critical tomcat:9.0.14-jre8
```

En este caso, establecemos la opción `-f` a `critical`. Esto significa que la salida del comando devolverá un error si la imagen analizada posee una vulnerabilidad catalogada como crítica.

3.6. Cumplimiento normativo

El cumplimiento normativo o *compliance* es un conjunto de procedimientos y buenas prácticas adoptados por las organizaciones para identificar y clasificar los **riesgos operativos y legales** a los que se enfrentan, y establecer así mecanismos internos de prevención, gestión, control y reacción frente a estos.

Dentro del marco normativo no han de considerarse únicamente las normas legales, como leyes y reglamentos, sino que también deberían incluirse las políticas internas, los compromisos con clientes, proveedores o terceros, y,

especialmente, los **códigos éticos** que la empresa se haya comprometido a respetar, pues existen multitud de casos en los que una actuación puede ser legal pero no ética.

Los proveedores de software de seguridad han creado herramientas para permitir que los usuarios finales, proveedores de servicios y otros creadores de productos evalúen más fácilmente el cumplimiento de la seguridad frente a todo el marco, en lugar de mirar cientos de controles diferentes. Exponemos a continuación una muestra de algunos de los estándares de cumplimiento comunes con los que nos podemos encontrar:

- El **Reglamento de protección de datos de la Unión Europea general (GDPR)** armoniza las leyes nacionales de protección de datos en la Unión Europea para proteger la privacidad de los ciudadanos de la UE.

- El **Pago de tarjeta de datos estándar de la industria de seguridad (PCI-DSS)** aumenta los controles alrededor de los datos de los titulares para reducir el fraude de tarjetas de crédito.

- La **ISO/IEC 27001** es un conjunto de normas internacionales sobre el manejo de seguridad de la información en las organizaciones que ayudan a hacer los activos de información que poseen más seguros.

- La **Health Insurance Portability y Accountability Act (HIPAA)** es una ley federal de Estados Unidos para proteger la información sensible de la salud y que sea divulgada sin el consentimiento del paciente.

Las organizaciones deben aplicar controles a su infraestructura TI para cumplir con las mejores prácticas del sector y regulaciones como PCI DSS, HIPAA, SOX o RGPD.

Con la infraestructura como código en DevOps, el entorno de producción no es estático, siempre se puede recrear de nuevo, de ahí que sea imperativo realizar una prueba sobre el entorno nuevo o actualizado una vez redesplegado. En la actualidad, existen un gran número de estándares y normativas de seguridad, muchos de obligado cumplimiento por parte de las organizaciones. Se pueden realizar las comprobaciones requeridas por estas normativas de una manera automatizada gracias a herramientas de código abierto para *Compliance as Code* como las siguientes.

- **Inspec**: https://www.inspec.io
- **Serverspec**: https://serverspec.org
- **DevSec Hardenning Framework**: https://dev-sec.io
- **Kitchen CI**: https://kitchen.ci

Chef Inspec

Chef InSpec es un marco abierto de código (*framework*) para probar y auditar aplicaciones e infraestructura. Chef InSpec funciona comparando el estado real del sistema con el estado deseado que se expresa en un código de Chef InSpec; también detecta infracciones y muestra los hallazgos en modo de informe, y todo esto sin perder el control de la remediación.

Bibliografía

Bolullo Pérez, Alejandro (2019, 30 de mayo). «DevSecOps: la evolución de la seguridad del software» [en línea]. *Paradigma*. <https://www.paradigmadigital.com/dev/devsecops-evolucion-seguridad-software>

Contributor (2014, 22 de octubre). «Eight critical DevOps practices: innovate, deliver, repeat» [en línea]. *DevOps.com*. <https://devops.com/eight-critical-devops-practices-innovate-deliver-repeat>

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

Harlann, Irma (2017, 28 de abril). «DevOps is a culture, not a role!» [en línea]. *neonrocket.medium.com*. <https://neonrocket.medium.com/devops-is-a-culture-not-a-role-be1bed149b0>

Hsiang-Chih Hsu, Tony (2018). *Hands-On Security in DevOps: Ensure continuous security, deployment, and delivery with DevSecOps*. Birmingham: Packt Publishing.

Hsiang-Chih Hsu, Tony (2019). *Practical Security Automation and Testing: Tools and techniques for automated security scanning and testing in DevSecOps*. Birmingham: Packt Publishing.

López, Marvin (2020, 30 de octubre). «El muro de la confusión» [en línea]. *Marvin López M.*. <https://www.imarv.in/el-muro-de-la-confusion>

OWASP Foundation (s. f.). «OWASP DevSecOps Guideline» [en línea]. *owasp.org*. <https://owasp.org/www-project-devsecops-guideline>

OWASP Foundation (s. f.). «OWASP Devsecops Maturity Model» [en línea]. *owasp.org*. <https://owasp.org/www-project-devsecops-maturity-model>

OWASP Foundation (s. f.). «Source Code Analysis Tools» [en línea]. *owasp.org*. <https://owasp.org/www-community/Source_Code_Analysis_Tools>

Red Hat (s. f.). «DevSecOps y la seguridad de DevOps» [en línea]. *Red Hat*. <https://www.redhat.com/es/topics/devops/what-is-devsecops>

Siddharth, Sumit (s. f.). «Herramientas open-source para adoptar DevSecOps» [en línea]. *Claranet*. <https://www.claranet.es/blog/herramientas-open-source-para-adoptar-devsecops>

Tal, Liran (2019, 26 de febrero). «Top ten most popular docker images each contain at least 30 vulnerabilities» [en línea]. *Snyk*. <https://snyk.io/blog/top-ten-most-popular-docker-images-each-contain-at-least-30-vulnerabilitie>

Telehouse (2016, 1 de marzo). «DevOps: how a culture of empathy creates massive productivity» [en línea]. *Telehouse*. <https://www.telehouse.com/2016/03/devops-how-a-culture-of-empathy-creates-massive-productivity>

Vehent, Julien (2018). *Securing DevOps-Safe services in the Cloud: Security in the Cloud*. Nueva York: Manning Publications.

Webgrafía

Anchore Engine: <https://github.com/anchore/anchore-engine>

Ansible: <https://www.ansible.com>

Arachni Scanner: <http://www.arachni-scanner.com>

AWS Secrets Manager: <https://aws.amazon.com/secrets-manager>

Clair: <https://github.com/coreos/clair>

Confidant: <https://github.com/lyft/confidant>

Constrast Community Edition: <https://www.contrastsecurity.com/contrast-community-edition>

Detect Secrets: <https://github.com/Yelp/detect-secrets>

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

Docker Compose: <https://docs.docker.com/compose>

Docker Scan: <https://docs.docker.com/engine/scan>

EnvKey: <https://www.envkey.com>

Git Hooks: <https://githooks.com>

Git Hound: <https://github.com/ezekg/git-hound>

Git Secret: <https://git-secret.io>

Grype: <https://github.com/anchore/grype>

Hashicorp Vault: <https://www.vaultproject.io>

Inspec: <https://www.inspec.io>

Keywhiz: <https://square.github.io/keywhiz>

Kitchen CI: <https://kitchen.ci>

Nikto: <https://cirt.net/Nikto2>

OWASP ZAP: <https://www.zaproxy.org>

Pre Commit: <https://pre-commit.com>

Serverspec: <https://serverspec.org>

SonarQube: <https://www.sonarqube.org>

Talisman: <https://github.com/thoughtworks/talisman>

Terraform: <https://www.terraform.io/intro/index.html>

Terrascan: <https://github.com/accurics/terrascan>

Trivy: <https://github.com/aquasecurity/trivy>

Truffle Hog: <https://github.com/dxa4481/truffleHog>