www.ismsforum.es

DevSecOps

Guía de iniciación en la Seguridad aplicada al DevOps

Una iniciativa de

isms forum

Mayo 2023

DevSecOps

Guía de iniciación en la Seguridad aplicada al DevOps

AUTORES

COORDINADORES Francisco Lázaro

SUBCOORDINADORES David Moreno

Enrique Cervantes

PARTICIPANTES Adrian Prieto

Jorge Cerezo

Jorge Pardeiro

Jorge Valenzuela

Juan José Fonseca

Margarita Sanz

Usama Alanbari

GESTIÓN DE PROYECTO Beatriz García

DISEÑO/MAQUETACIÓN Cynthia Rica

CONTENIDOS

1. ¿Qué es DevOps? 08

2. ¿DevOps es un modelo valido para todas las empresas? 14

3. Cambio cultural 16

4. Gestión de proyectos DevSecOps 18

- 4.1. Metodologías de gestión de proyectos 18
- 4.2. El manifiesto Agile y la CI/CD 19
- 4.3. Modelos de Madurez 22
- 4.4. Más allá de “Kanban” o “Scrum” 24

5. La seguridad en DevOps 26

- 5.1. Seguridad desde el diseño y por defecto 26
- 5.2. Equipos de desarrollo DevSecOps 30

6. Marcos de desarrollo seguro, formativos y buenas practicas 40

- 6.1. ¿Por qué ayudarse de marcos de desarrollo seguro? 40
- 6.2. OWASP 41
- 6.3. MITRE – Metodología SecDevOps 47
- 6.4. Otros marcos de desarrollo seguro 49

7. Herramientas 50

7.1. Introducción 50

7.2 Herramientas & Clasificación 51

7.3. Adopción progresiva de las herramientas 55

8. Automatización “sec” 56

8.1. Introducción 56

8.2. Pipelines 56

8.3. Ciberseguridad en los pipelines 58

8.4. Inventariado y monitorización 59

8.5. Bloqueo VS Alerta 61

9. Cloud Infraestructura como código, contenedores, Kubernetes y microservicios 62

9.1 Breve introducción a la infraestructura como 62

9.2. Seguridad en infraestructura como código 64

9.3. Contenedores, clústers y microservicios 65

9.4. Seguridad en entornos dockerizados 70

CONTENIDOS

10. Hoja de ruta para adoptar DevSecOps **72**

10.1. Niveles de madurez **72**

10.2. Desafíos **75**

10.3. ¿Cómo empezar? **76**

10.4. Medir el éxito de la implantación **77**

11. Referencias **80**

1 ¿QUÉ ES DEVOPS?

Tradicionalmente, la creación de nuevos sistemas en el ámbito de las tecnologías de la información ha involucrado a perfiles con necesidades y objetivos muy diferentes. Por un lado, tenemos a los/las responsables de mantener las infraestructuras operativas (personal de comunicaciones, arquitectura, sistemas o seguridad entre otros) y en buena forma, siempre disponibles de acuerdo con el nivel de servicio necesario y dimensionadas para unas determinadas cargas establecidas. Y, por otro, están los perfiles expertos en el desarrollo de software, que tienen que responder a las necesidades del negocio en plazos ajustados y no siempre con unos alcances y definiciones funcionales claros y ajustados a la necesidad empresarial.

La dependencia mutua es completa, desde el momento en que, para que una nueva aplicación funcione requiere de una infraestructura específica; y para que esa infraestructura esté optimizada y ajustada, necesita conocer cómo se ha desarrollado la aplicación que va a ser ejecutada sobre ella.

Sin embargo, los intereses entran muchas veces en conflicto debido a que, mientras que los equipos de desarrollo necesitan desplegar sus soluciones de forma rápida y, a veces, sin tener en cuenta las limitaciones de infraestructura, los equipos de operaciones tienen la obligación de dimensionar dicha infraestructura de una forma sostenible y controlada; con unos tiempos de aprovisionamiento más lentos e inversiones y costes que controlar desde el punto de vista financiero.

Desafortunadamente la solución a este conflicto de intereses no pasa por juntar ambos roles en un único equipo, no es tan sencillo.

Para entender el porqué de la necesidad de utilizar metodologías DevOps en las empresas actuales, analicemos cómo se han venido gestionando los proyectos de software hasta hace algunos años.

Los equipos de desarrollo han venido trabajando en un modelo de gestión de proyectos llamado "en cascada" o waterfall. Esta metodología consiste en dividir las fases de desarrollo en pequeñas tareas relacionadas entre sí, definiendo hitos parciales y estimando tiempos de desarrollo, en base a la experiencia del propio equipo y la carga de trabajo, entre otros aspectos.

Esta forma de trabajar es altamente ineficiente por varios motivos:

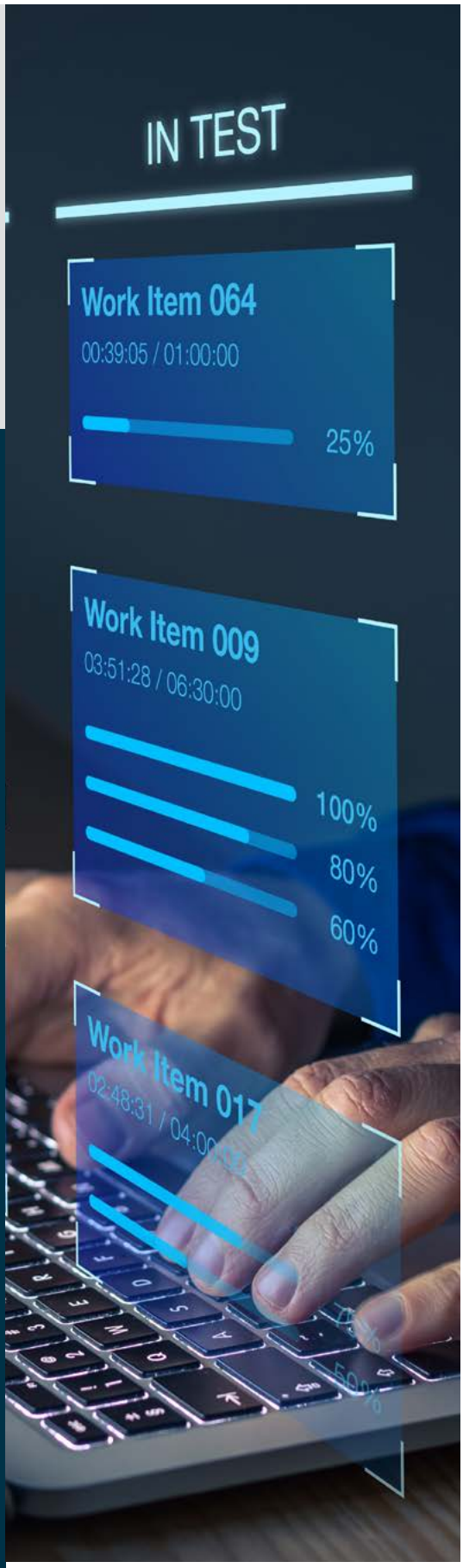
- Si existe un mecanismo de retroalimentación y las necesidades del solicitante cambian, es necesario reajustar todas las tareas para acomodarlas, con el consiguiente impacto en costes y recursos. En estas situaciones, es habitual intentar mantener los plazos de ejecución y los costes iniciales, lo que repercute en una menor calidad en el producto final; es algo inevitable.

- No disponemos de un producto consumible hasta el final del proyecto y nuestro usuario no podrá saber si los requerimientos iniciales están bien implementados. O peor aún, es muy común que esos requerimientos sean muy vagos o poco realistas, lo que genera una sensación de frustración y pérdida de tiempo en todos los involucrados, usuarios y equipo de TI.

La solución a esta realidad vino de la mano de la introducción de dos metodologías nuevas, la primera heredada de la industria automovilística (LEAN) y la segunda creada por un grupo de desarrolladores que idearon el llamado Agile Manifesto (Manifiesto Ágil), que veremos más adelante, pero que introduciremos brevemente en este punto.

El Manifiesto Ágil establece doce principios que promueven la calidad del software, ciclos continuados de mejora, la colaboración más estrecha entre personas con el objetivo común de la entrega continua de valor.

A medida que esta metodología se ha ido imponiendo en los equipos de software, convirtiéndolos en más productivos y permitiendo incorporar una retroalimentación del solicitante de forma continua en el proyecto, el cuello de botella se ha trasladado a los equipos de operaciones, que no son capaces de responder a las necesidades de los desarrolladores a la velocidad necesaria.



Para solucionar este punto, se necesitan introducir nuevos mecanismos de aprovisionamiento, automatización de tareas mediante código y nuevos paradigmas de gestión, monitorización y respuesta a requerimientos, sin perder de vista la disponibilidad y escalabilidad de las plataformas.

DevOps nace precisamente de la fusión de los equipos de desarrollo y de operaciones con metodologías LEAN y Agile.

Los equipos DevOps están formados por miembros multidisciplinares (arquitectos, ingenieros de sistemas, desarrolladores, testeadores, responsables de producto -producto owners-, entre otros) que son capaces de actuar de forma aislada sin casi ninguna dependencia de otros equipos; disponen del conocimiento y las herramientas necesarias para gestionar el ciclo de vida completo de una aplicación: pueden diseñar una nueva característica, programarla, probarla, generar los entornos necesarios, y desplegar la versión definitiva en producción, mientras evalúan continuamente cómo mejorar y ser más eficientes.

Sin embargo, en la realidad, implementar una cultura DevOps en los equipos no es una tarea sencilla ya que poner en práctica un modelo como el descrito presenta retos culturales, procedimentales, de aprendizaje, tecnológicos y de resistencia al cambio.

Empezaremos describiendo el reto que, habitualmente, es de los más sencillos de solucionar: el tecnológico. Aunque es cierto que no son imprescindibles en fases iniciales, sí que es recomendable contar con plataformas que permitan la automatización de tareas, que descarguen por un lado al equipo de actividades que aporten poco o ningún valor y, por otro, eliminar los errores humanos. En este sentido, existen actualmente en el mercado soluciones solventes que permiten definir infraestructura como código; es decir, desplegar infraestructura que está definida en instrucciones de código y facilita su automatización, en lugar de crearse de forma manual, cuyo uso está extendido en empresas de todos los tamaños.

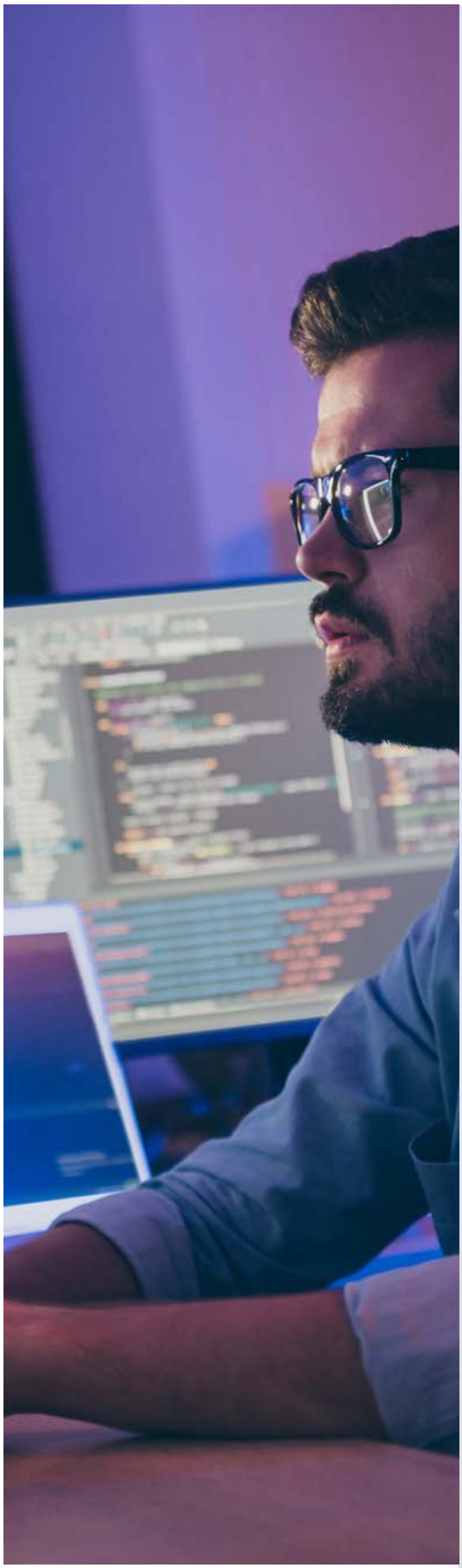
Este punto abre la discusión sobre el aprendizaje necesario para adquirir el conocimiento y capacitar en la utilización de estas nuevas tecnologías, así como las metodologías ágiles que son la base de este cambio de enfoque.

Es cierto que es común para el personal técnico mantener actualizadas sus competencias, adquiriendo de forma continua nuevos conocimientos; y en ese sentido, estas nuevas herramientas no deben suponer un problema.

Sin embargo, la adopción de las nuevas metodologías, que cambian drásticamente la forma de trabajar, presenta en múltiples ocasiones un proceso de resistencia al cambio. Los equipos pasan de un modelo, donde tienen el control completo de su ámbito de actuación, a otro donde la colaboración, el intercambio de información, la crítica continua, la búsqueda de la eficiencia y las métricas son la base fundamental del método.

Muchas personas ven esto no como una oportunidad sino una fiscalización continua de su trabajo y un menosprecio a su experiencia. Nada más lejos de la realidad, DevOps está enfocado a facilitar el trabajo de los equipos, a poner en mayor valor y dar visibilidad a su impacto en el negocio y a ayudar a la corporación en sus objetivos de negocio. Ayuda a dar a conocer a los equipos de Tecnologías de la Información (TI, en adelante) más allá de las paredes del centro de datos. Es en este punto donde padeceremos la mayor resistencia al cambio y será uno de los puntos más conflictivos a la hora de implementar estos nuevos modelos de gestión.

Por último, tenemos los cambios culturales, aunque más adelante, en esta misma guía, hablaremos de ellos, tanto en el ámbito de los departamentos de TI como de negocio, por lo que no ahondaremos en ellos ahora.



Como hemos visto anteriormente, DevOps surge como respuesta a la necesidad de mejorar los tiempos de puesta en producción de nuevas soluciones de software. Las corporaciones, ya sean públicas o privadas, se han visto envueltas en procesos de digitalización y transformación muy agresivos, que han obligado a poner en marcha iniciativas para poder mantener la competitividad en un escenario altamente variable, responder a movimientos de mercado o necesidades del cliente, crecer o, sencillamente, sobrevivir.

Todos estos procesos de digitalización han repercutido en una mayor intensidad de respuesta rápida en los equipos de TI, especialmente en aquellos que entregan valor en forma de soluciones de software a medida o adaptaciones de productos de mercado.

De la misma manera, la adopción de la nube como paradigma de consumo de servicios TI ha sido en gran medida detonante de la necesidad de implementar metodologías ágiles, por varios motivos:

- La sencillez a la hora de desplegar infraestructura nueva y ponerla en producción, frente a modelos más rígidos de infraestructura on-premise (en sus propias instalaciones), con modelos de inversión y plazos de amortización que eliminan las barreras de entrada y fomentan la innovación permitiendo validar ideas y proyectos sin asumir grandes riesgos (aunque en según cómo, en mayores gastos).

- La flexibilidad y la escalabilidad, que permiten ajustar muy fácilmente nuestra infraestructura a las necesidades puntuales, sin que tengamos la necesidad de sobredimensionarla para poder responder a picos máximos.

- La independencia, al Negocio, para poder consumir servicios en un modelo de pago por uso, sin necesidad de contar con TI en muchos casos (el llamado Shadow IT o TI en la sombra –sistemas no identificados/gestionados por el área de TI).

Evidentemente, fueron las empresas desarrolladoras de productos en nube de carácter masivo como Flickr, Netflix, Google, Microsoft, etc. los primeros que implementaron estos modelos ágiles, para poder ofrecer productos competitivos en tiempo récord. Nos acostumbramos a disponer de soluciones avanzadas en cuestión de dos o tres semanas y esto se trasladó a los equipos de desarrollo de las empresas con modelos más tradicionales. La presión del negocio para que el tiempo de comercialización o time to market fuera lo más reducido posible, ya tenía ejemplos reales y no son pocos los ejemplos de corporaciones donde ha sido el propio consejo de administración el que ha abanderado el proyecto de migración a la nube o adopción de metodologías ágiles como medio para competir en un mercado cada vez más exigente.

La nube es parte fundamental de estos modelos y, seguramente, sin su existencia y rápida adopción, la implementación real de los mismos sería costosa y poco viable.

Hay un punto de unión intrínseco entre cómo consumimos los servicios en nube y cómo los desarrollamos, cómo se trasladan los beneficios de la nube a las necesidades de las empresas; la nube ha supuesto, posiblemente, una de las revoluciones de mayor impacto en las tecnologías de la información. Hoy en día no es viable gestionar un departamento de TI sin que se tenga en cuenta la nube, en cualquiera de sus modalidades (Iaas, PaaS, SaaS, SECaaS, DRaaS, etc.) y se realicen ejercicios de viabilidad y casos de negocio que evalúen la conveniencia o no de utilizar la nube. Esto no quita que no todo es un camino de rosas en este sentido y que también se requieren importantes esfuerzos económicos, de procesos y humanos para adoptar la nube como modelo de crecimiento de las TI. Sin contar con el reto que la nube representa en el ámbito de la seguridad de la información, aspecto este que se tratará más adelante, en esta misma guía.

2 ¿DEVOPS ES UN MODELO VÁLIDO PARA TODAS LAS EMPRESAS?

La respuesta a esta pregunta es no, DevOps no se puede implementar en cualquier corporación. O, mejor dicho, se puede implementar, pero no es necesariamente la mejor decisión para cualquier tipo de organización.

Como hemos visto anteriormente, el concepto DevOps está ligado a las metodologías ágiles en el desarrollo de producto y no tiene sentido una sin las otras. Por lo tanto, es necesario implementar primero un modelo de trabajo basado en Scrum, Kanban (ambos son marcos de gestión de proyectos) o una mezcla de ambas (recordemos que agile es un marco de trabajo abierto, alejado de los dogmas) antes de plantearse llevar DevOps a un departamento de TI.

También debemos tener en cuenta que ciertos lenguajes de programación más antiguos tienen difícil encaje en un modelo agile y, por tanto, DevOps, sobre todo porque no son compatibles con las herramientas modernas. Tampoco existen mecanismos sencillos de automatización en esos entornos, por lo que deberemos optar por una gestión de tipo cascada clásica o bien plantear una evolución de la aplicación hacia arquitecturas y frameworks (marcos de trabajo) más actuales que permitan trabajar con ellos en un modelo cultural DevOps.

Otros ámbitos donde es recomendable no usar DevOps es en las fases de planificación y diseño, ya que trabajar de forma sistemática sin iteraciones permite a los arquitectos crear un diseño más robusto y con menos fallos. En estas fases es importante pensar detenidamente en cómo se implementará la aplicación de la mejor manera posible; aplicando DevOps aquí hace que pensemos menos y automaticemos más. El resultado si no se realiza correctamente podría ser un diseño no óptimo.

La realidad es que DevOps es muy poderoso y soluciona un buen número de los problemas habituales entre las áreas de desarrollo y operaciones. Pero no se puede dejar la planificación de un producto a un grupo de desarrolladores sin controlar lo que va a salir una vez finalicen su tarea. Evitar esto reduce las sorpresas y permite tener un mayor control sobre todo el flujo.

Como con cualquier otra tecnología actual, no estamos obligados a usar DevOps siempre solo porque esté de moda, lo importante es conocer sus ventajas, inconvenientes y limitaciones para poder decidir con conocimiento de causa si se adapta a nuestras necesidades.

También hay que huir de los dogmatismos a la hora de implementar DevOps. Como se ha dicho anteriormente, son marcos de trabajo que, si bien describen procesos claros y definidos, deja al implementador la decisión de ser más o menos flexible en determinados momentos. Llevar al extremo la aplicación de estas guías puede convertir en una pesadilla para los equipos lo que supuestamente ha venido para hacerles la vida más fácil.

3 CAMBIO CULTURAL

La introducción de metodologías ágiles y DevOps en una organización presenta muchos retos y oportunidades, algunos de ellos más fácilmente gestionables que otros.

En general, el cambio en la cultura y la gestión al cambio son los más complejos desde el punto de vista de la gestión de personas. Como con cualquier otra disrupción que afecte a cómo se hacen las cosas, las responsabilidades de las personas, las relaciones interdepartamentales y las herramientas, nos enfrentaremos a una resistencia natural del ser humano a salir de su zona de confort. En el caso de las metodologías ágiles, por ser una tendencia que nace fundamentalmente de las áreas de desarrollo, se puede encontrar que dicho rechazo es más acusado en los equipos de operaciones, ya que ven como una intrusión en su forma de trabajar e, incluso, como un "triunfo" la implantación de estos modelos de trabajo o cuando no, que precisamente las áreas de desarrollo que habitualmente han estado alejadas de la sensibilidad por la optimización de los recursos máquina o de las buenas prácticas de arquitectura de sistemas o de seguridad de la información, sean ahora las que desplegaran esos recursos de infraestructura. En contraposición, desde una perspectiva positiva y en determinados ámbitos, puede ser visto como una oportunidad de mejorar las competencias y puede ayudar en la retención de talento de las compañías.

Pero estos cambios no solo afectan al departamento de TI, sino que el negocio también deberá enfrentarse a una mayor implicación en los procesos de toma de decisión. Figuras como el Product Owner (el responsable de producto, en adelante PO) no existían anteriormente, el usuario se limitaba a trasladar sus necesidades al equipo correspondiente y a esperar la entrega del resultado final; con suerte, participaba en el proceso de control de calidad o de aceptación. Ahora el escenario es bien distinto, no solo se necesita definir claramente qué necesita el usuario, sino que se acompaña a los equipos de desarrollo en procesos que anteriormente eran completamente ajenos y opacos para el usuario. En este sentido, se busca por un lado una mayor implicación del negocio en los procesos de creación, haciéndole colaborador de las decisiones que se toman y, por otro lado, permite un mayor acercamiento de los equipos de desarrollo a las necesidades y a los cambios que puedan producirse durante el desarrollo del proyecto. El PO es uno más del equipo y participa en muchas de las dinámicas de trabajo.

Precisamente, esas dinámicas de trabajo, con reuniones diarias donde se exponen los principales problemas que tiene el equipo para desarrollar sus funciones; las retrospectivas, donde se analiza cómo ha evolucionado el proyecto, dónde se ha fallado y se identifican puntos de mejora a aprovechar en el futuro; la introducción de métricas de control que evidencian el rendimiento de los equipos y sus miembros, son fuente habitual de conflicto en las primeras fases de implementación de los modelos.

Es necesario que los equipos partan de una posición de colaboración y entendimiento de por qué se hacen las cosas, qué sentido tienen las metodologías y se sientan cómplices del éxito. En este punto es importante contar con figuras denominadas "champions", que actúen como impulsores del cambio entre sus compañeros y sean modelos que imitar.

Otro elemento de nueva creación que las organizaciones deben asumir son los MVP, acrónimo de Minimum Viable Product o Mínimo Producto Viable. El MVP representa el entregable con la mínima funcionalidad posible que aporte valor al negocio, de tal manera que podemos centrarnos en ciclos de desarrollo más cortos, podemos ir iterando con el PO para entender si se va por el buen camino y se puede reaccionar con mayor velocidad antes cambios de alcance.

Se puede entender qué es el MVP mediante el siguiente ejemplo:

> *Dos equipos de desarrollo de producto reciben la orden de construir una motocicleta para cubrir sus necesidades de movilidad.*
>
> *Un equipo utiliza metodologías en cascada y, el otro, metodologías ágiles.*
>
> *El primero realiza una labor de planificación completa y determina que entregará la motocicleta en el plazo de 3 meses.*
>
> *El segundo equipo acuerda con el usuario un MVP que consiste en la construcción de un monopatín. Este es entregado en un plazo de 15 días. 15 días después se entrega un patinete, 15 días después, una bicicleta y, finalmente, 15 días después se entrega la motocicleta.*
>
> *Con el primer modelo, el usuario no podrá cubrir sus necesidades hasta que le entreguen la motocicleta; no podrá determinar si ese producto es el que realmente quiere y cualquier cambio de alcance implicará rehacer toda la planificación.*
>
> *Con el segundo modelo, el usuario dispone desde la primera iteración de un producto que sí cubre sus necesidades, aunque no sean completas; podrá determinar en las siguientes iteraciones si el equipo va por el buen camino y se podrán introducir cambios de alcance sin mucho esfuerzo. Incluso puede que el usuario descubra que no necesita una motocicleta sino un patinete eléctrico.*

Evidentemente, las ventajas son claras y el equipo que usa el segundo modelo es mucho más eficiente al usar su tiempo y recursos.

Pero para llegar a esto, tiene que existir voluntad, primero para entender la filosofía que hay detrás y, segundo, comprender que el fin de cualquier departamento de TI es aportar valor a la compañía creando producto de calidad en el menor tiempo posible y que respondan a las necesidades marcadas.

Por otra parte, las organizaciones se implican mucho más en la toma de decisión, no perciben a TI como una caja negra donde nadie sabe qué pasa dentro, se da mayor visibilidad a la función y a cómo impacta en la consecución de objetivos y, por descontado, en su posicionamiento de mercado.

Es interesante resaltar también que las metodologías ágiles pueden ser aplicadas en procesos distintos a los tecnológicos; en general, funcionan bien en todos aquellos destinados a la creación de un producto con unas fases definidas; marketing, diseño industrial, planificación, fabricación, etc.

4 GESTIÓN DE PROYECTOS DEVSECOPS

4.1. Metodologías de gestión de proyectos

Existen varias metodologías de gestión de proyectos, "waterfall" (o de cascada), "agile" (o ágil), crítica, e híbrida. Sin embargo, podríamos agrupar todas ellas en dos grandes categorías, las orientadas a gestión de proyectos y las orientadas a gestión de productos.

Primero entendamos cuál es la diferencia entre un proyecto y un producto. La principal característica que los diferencia es que un producto no tiene una definición concreta sobre qué se necesita entregar ni la fecha en la que debe estar listo. Un producto es algo vivo, que evoluciona en la misma medida en la que evoluciona la empresa puesto que cubre unas necesidades fundamentales que pueden ir cambiando de forma con el tiempo. Un proyecto busca solucionar una problemática muy acotada y definida que, una vez cubierta, podría no volver a darse nunca.

Los proyectos se financian en función de los "beneficios proyectados en un caso empresarial" y "se dotan de personal de una reserva de talento" con el mandato de "construir o mejorar algún sistema o aplicación y seguir adelante".

En cambio, el modo de producto utiliza equipos duraderos, dirigidos por profesionales con visión y criterio, que trabajan en un problema empresarial persistente. El modo producto permite a los equipos reorientarse rápidamente, reducir la duración de su ciclo integral y validar los beneficios potenciales reales mediante iteraciones cortas, manteniendo al mismo tiempo la integridad arquitectónica de su software para preservar su eficacia a largo plazo.

La financiación de los equipos en modo producto significa que se les financia para que trabajen en un problema empresarial concreto o en una oferta durante un periodo de tiempo determinado; la naturaleza del trabajo viene definida por un problema empresarial que hay que abordar más que por un conjunto de funciones que hay que ofrecer. Llamamos a esta forma de trabajar "modo-producto".

4.2. El manifiesto Agile y la CI/CD

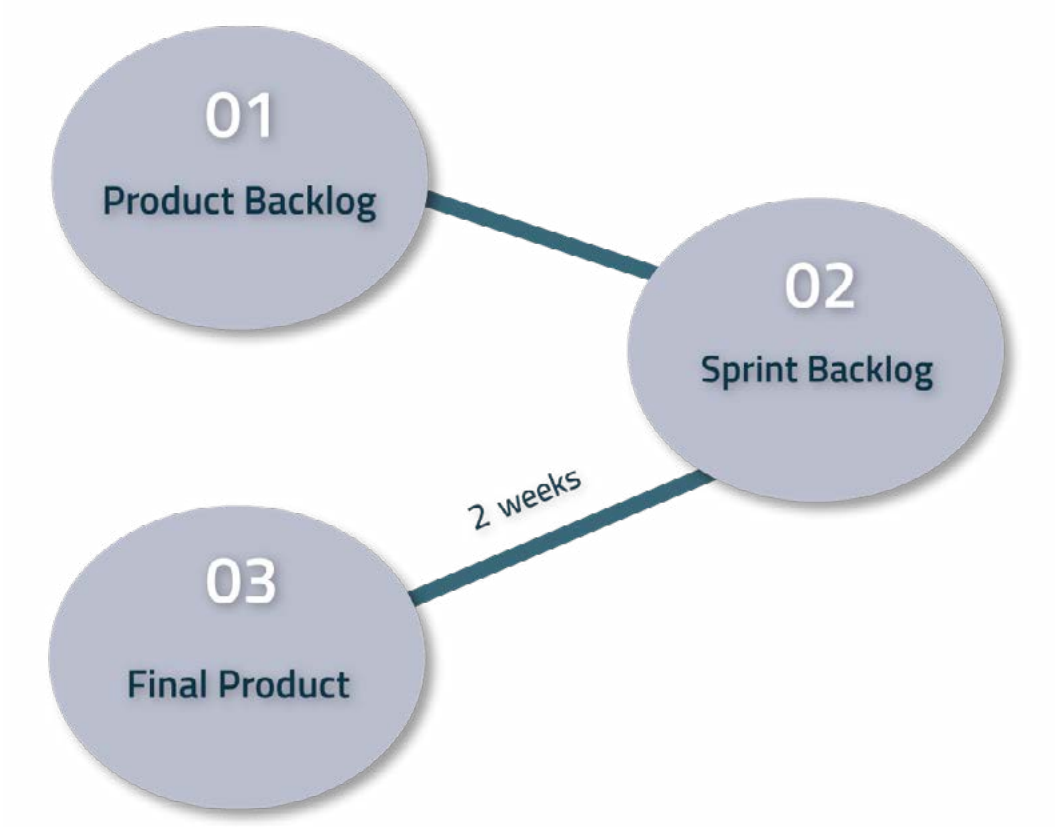
Agile va a aportar en el proceso del DevSecOps para que no existan silos entre Desarrollo, Prueba y Operaciones. Cuánto más trabajo en equipo e intercambio de información exista, se conseguirá una mejor integración a lo largo del ciclo de vida.

El desarrollo e implementación serán iterativos de forma que al diiseñar, desarrollar, probar e implementar cambios incrementales o cambios para usuarios comerciales serán mucho más rápidos.

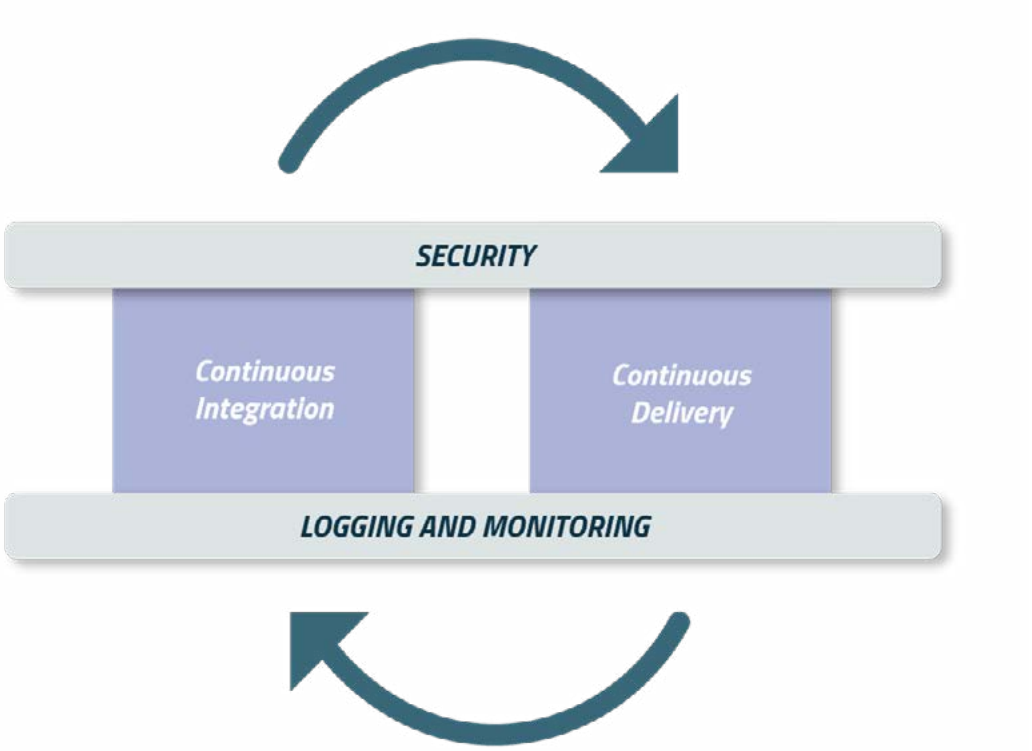
Pero lo que de verdad es la clave es la automatización, tanto como sea posible, porque ayudará a reducir el tiempo de entrega, mejorar la calidad y la seguridad, intentando eliminar el error humano.

Los procesos optimizados, repetibles y rutinarios harán que los ciclos de entrega sean cada vez más rápidos y esto supondrá que los clientes estén más satisfechos. Además, promoviendo una cultura de desarrollo seguro, con el uso de prácticas y herramientas que ayuden a la automatización con metodología Agile, permitirá que los equipos estén capacitados para aprovechar al máximo las tecnologías idóneas y lograr un resultado de calidad y con prácticas seguras.

La metodología Agile en el que se desarrollan funciones en base a las necesidades del usuario reflejados en la siguiente imagen, con sprint, backlog y la creación de un producto final.



Se integra en la canalización DevSecOps con la automatización de compilación, protección/prueba, implementación y supervisión, dónde la seguridad es parte del proceso completo, desde el inicio (fase de preproducción) hasta las Operaciones de Producción:



El gráfico muestra un ciclo continuo de CI & CD (Continuous Integrarion & Continuous Delivery). Con la integración continua (CI) se consigue el ciclo de retroalimentación que garantiza la corrección de errores y la reparación de vulnerabilidades en cada etapa de la canalización de DevSecOps. Por otro lado, la entrega continua (CD) promueve el software de trabajo desde entornos inferiores a entornos más altos después de que se cumplan las pruebas y la seguridad.

Las metodologías ágiles alcanzan su mayor potencial cuando se trabaja en un modelo orientado a producto en lugar de un modelo orientado a proyecto. Algunos de los principales beneficios son:

- Los equipos de producto empiezan a producir valor antes. Los proyectos tradicionales aportan todo su valor al final, cuando ya se ha hecho todo el trabajo y el equipo se ha disuelto. Incluso a veces, las empresas nunca saben si el proyecto se ha llegado a terminar o si el equipo se ha disuelto antes, porque no utilizan indicadores para medir sus casos de negocio y responsabilizar conjuntamente a la empresa y a IT de los resultados. Por diseño, cada entrega más pequeña en un enfoque de producto ágil aporta valor. Puedes empezar a medir el valor entregado haciendo un seguimiento de las métricas en el resultado empresarial. Mientras trabajas en la siguiente pieza de valor, deberías empezar a obtener pruebas de que la primera funcionó. Si, por alguna razón, no ves una mejora, puedes encontrar la causa raíz y solucionarlo rápidamente.

- Las organizaciones orientadas a producto son más eficientes y reducen más en gastos. La transformación hacia un modelo más ágil y orientada a producto se basa en una transformación en la eficiencia de los equipos de IT. Al igual que las fábricas y almacenes de los años 80 y 90, el resultado es una organización más plana, con menos gastos generales y más personas dedicadas directamente al flujo de valor. El equipo ágil estándar tiene desarrolladores, un scrum master y un propietario del producto que representa a la empresa. Un equipo de operaciones apoya la automatización y producción DevOps. Además, un arquitecto ágil trabaja en todos los equipos para asegurarse de que se utilizan enfoques estándar y se cumplen los requisitos no funcionales. Las iniciativas de mayor envergadura también pueden contar con un gestor de producto que elabore la hoja de ruta del producto a largo plazo.

- Los equipos de producto se pueden adaptar en función de la demanda. Esta ventaja es controvertida, pero muchas organizaciones acaban teniendo que reasignar recursos de vez en cuando. El listón debe estar alto, porque un equipo que ha pasado meses averiguando cómo trabajar juntos y equilibrando las habilidades necesarias para hacer el trabajo puede verse gravemente perturbado por un cambio. Los equipos estables son deseables, pero eso no significa que el tamaño y la composición del equipo no puedan cambiar. Si cada equipo cuenta con todas las competencias necesarias para trabajar en su producto, se puede cambiar de personal ocasionalmente, cuando se requiera menos trabajo a medida que el producto madura, o cuando el equipo de un producto de alta prioridad necesite más capacidad. Como los miembros del equipo son ingenieros full-stack, se puede lograr un cambio equilibrado.

- Los productos minimizan las pérdidas de calidad. Los productos ágiles se basan en la idea de lanzamientos frecuentes, pero más pequeños. Esto no va a servir de nada a menos que se reduzca el traspaso del desarrollo a las operaciones. Esto se consigue mediante DevOps y la creación de una cadena de herramientas automatizada. En lugar de realizar las pruebas y el proceso de producción manualmente, el equipo de operaciones automatiza el proceso y supervisa el resultado de cada traspaso. Reducir el número de cambios en cualquier versión reduce la posibilidad de que los errores se cuelen en la producción por falta de pruebas. En los proyectos, el tiempo de pruebas al final siempre es escaso. La integridad arquitectónica es otra de las grandes ventajas del modelo centrado en el producto para garantizar la calidad. Dado que los miembros del equipo permanecen en él durante un largo periodo de tiempo, cuando tomen decisiones de arquitectura, es mucho más probable que tengan en cuenta los objetivos a largo plazo, ya que son ellos quienes vivirán con las consecuencias.

- Los proyectos ayudan a que las partes interesadas prioricen el trabajo. En los proyectos, con demasiada frecuencia se asigna el dinero al proyecto, pero nunca hay suficientes personas para dotarlos de todo el personal necesario. La gente pasa constantemente de un proyecto a otro para avanzar, perdiendo hasta un 40% de su productividad debido al cambio de contexto. En lugar de tener un alcance, un presupuesto y un calendario fijos, los productos ágiles fijan esencialmente la capacidad que se dedica a un producto o productos. Las partes interesadas no pueden exigir un calendario irracional, pero sí responder a la pregunta "¿Qué es lo siguiente que quieres?". Esto obliga a las partes interesadas a priorizar el trabajo y ayuda a garantizar que los elementos en los que el tiempo es un factor crítico se encuentran al principio del proceso. De este modo, el departamento de IT o el comité de gobierno no tienen que tomar decisiones detalladas sobre las prioridades.

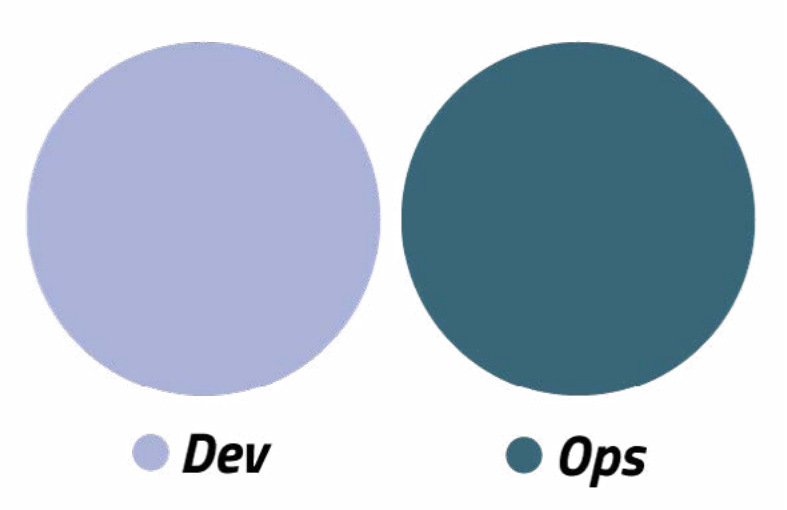
4.3. Modelos de Madurez

Existen distintos modelos de madurez, también conocidos como patrones y anti-patrones. A continuación, vamos a comentar los más comunes dentro de las empresas.

Es importante tener en cuenta que, aunque el modelo preferible es el modelo DevOps, no en todos los casos es posible pasar de un modelo cascada a un modelo ágil. Dependiendo de la estructura de la organización puede ser muy beneficioso realizar pasos intermedios que garanticen que, culturalmente, se entienden y adoptan los principios de una forma más natural. Además, es esta cultura organizacional la que va a predecir la forma en la que la información fluye entre las distintas organizaciones y por tanto, la manera en la que los distintos equipos se relacionen.

—El modelo clásico

El modelo clásico es el modelo en el que se encuentran las empresas que no han empezado aún su transformación digital/DevOps. Este es el caso más común. Lo que podemos ver es que el mundo de desarrollo y el mundo de las operaciones se encuentran completamente aislados, convirtiéndose en silos entre ellos. La principal consecuencia de este modelo es una falta de comunicación entre ambos equipos que impacta directamente en la productividad, eficiencia y cantidad de recursos necesarios para sacar cualquier nuevo producto a producción.

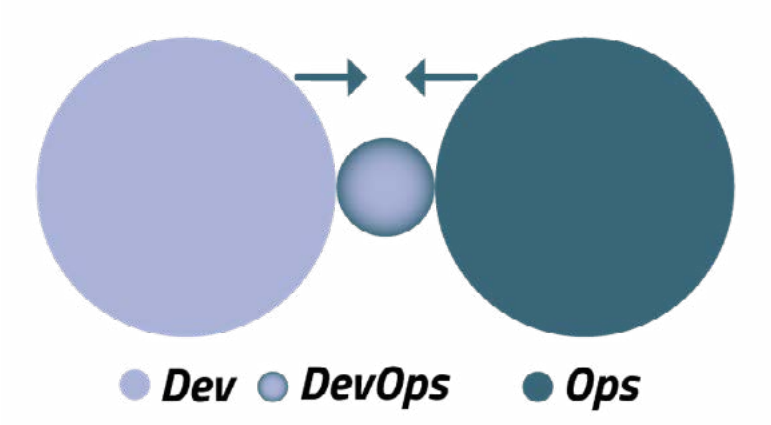


—Equipo DevOps temporal

Este modelo propone crear un nuevo equipo de DevOps que colabore con los equipos existentes de desarrolladores y operaciones. Tiene algunos beneficios:

- El equipo temporal de DevOps tiene como objetivo acercar ambos equipos.
- Este equipo puede ser útil a la hora de comunicar mensajes, ayudar en la implantación del nuevo modelo y servir de guía para el resto.

Sin embargo, hay que tener en cuenta la inversión estratégica que debe hacer la empresa para implantar este modelo de forma temporal.

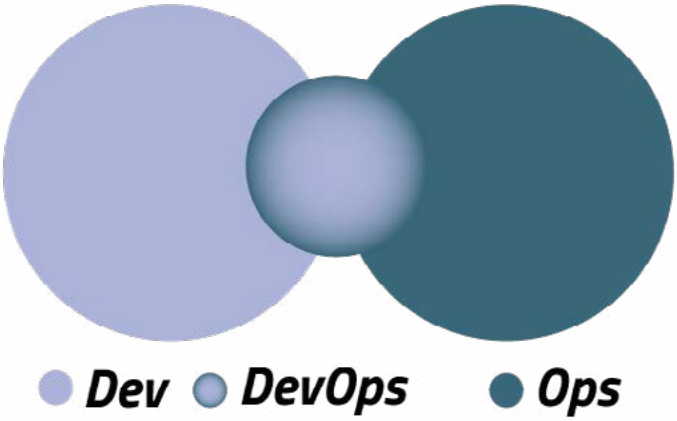


—Equipo evangelista de DevOps

Entre los principales beneficios de este modelo se encuentran:

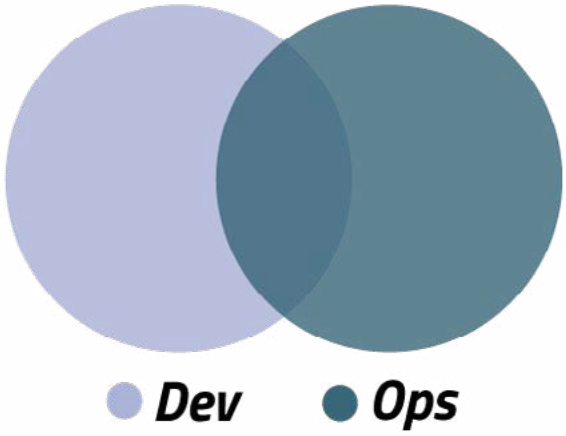
- Es muy útil en aquellos contextos en los que la separación entre los desarrolladores y los equipos de operaciones es muy grande.
- Empieza a construir equipos híbridos donde el equipo evangelista ya no solamente traslada mensajes, sino que colabora en las tareas del día a día.

De nuevo, como principal contrapunto está el hecho de la inversión que supone la implantación de un modelo así.



—El modelo DevOps

Este es el modelo óptimo donde ambos equipos trabajan de forma conjunta al tener perfiles híbridos que permiten incluir ambas perspectivas tanto en el desarrollo de los productos como en la operación de estos. Este modelo se enriquece incluso más cuando también se incluye al equipo de seguridad.



4.4. Más allá de “Kanban” o “Scrum”

Como resumen práctico de lo expuesto en este apartado de la guía, se pondrá el ejemplo de cómo DevSecOps es una filosofía que va más allá de la metodología de gestión de proyectos que se utilice. Si se habla de dos de los métodos más populares de gestión, como son Kanban y Scrum, se puede observar, que aunque son dos métodos que tienen diferencias entre ellos, caben dentro de DevSecOps de una manera perfecta y completa. Se hará la comparativa en los pilares más comunes de ambas estrategias:

—4.4.1. Enfoque de entrega

Mientras Kanban es un marco de trabajo que se centra en la entrega continua, sin plazos definidos y atendiendo a las necesidades del producto, Scrum se centra en la entrega iterativa incremental en los llamados “sprints”, entregando evolutivos del producto.

Desde el punto de vista de DevSecOps, mientras las seguridad vaya integrada dentro del ciclo de vida del desarrollo desde el principio y hasta el final, como se podrá observar en esta guía, ambas metodologías caben dentro de la filosofía de una manera completa.

—4.4.2. Roles y Responsabilidades

En Kanban, no encontramos roles definidos. Todos los miembros del equipo tienen las mismas responsabilidades y trabajan juntos para completar las tareas. En cambio, en Scrum sí que hay roles definidos (Product Owner, Scrum Master y Equipo de Desarrollo) con responsabilidades específicas en el proyecto.

Desde el punto de vista de DevSecOps, el enfoque sobre las responsabilidades no tienen que ver con el rol, sino con el individuo en sí mismo, ya que todos los componentes del equipo tienen la responsabilidad común de que la seguridad debe estar integrada en todo el ciclo de vida del producto.

—4.4.3. Planificación y seguimiento

En Kanban, la planificación y el seguimiento se realizan mediante el uso de un tablero Kanban. El tablero muestra las tareas pendientes, en progreso y completadas. En Scrum, la planificación y el seguimiento se realizan mediante el uso de reuniones diarias, de revisión y retrospectivas.

Con DevSecOps, la planificación y el seguimiento son una parte integral de todo el ciclo de vida del desarrollo de software. Los equipos deben planificar y realizar un seguimiento de la seguridad en todas las fases, desde el diseño hasta la implementación y el mantenimiento. De manera que es irrelevante cómo se gestiona esa planificación, mientras cubra la seguridad en todas sus fases.

—4.4.4. Mejora continua

En Kanban, la mejora continua es un proceso constante en el que los equipos trabajan en pequeñas mejoras continuas en el proceso de entrega. En Scrum, la mejora continua se realiza en la reunión de retrospectiva después de cada sprint.

Con DevSecOps, la mejora continua es una parte integral de todo el ciclo de vida del desarrollo de software. Los equipos deben trabajar continuamente en la mejora de la seguridad en todas las fases del ciclo de vida del desarrollo de software. La ciberseguridad es una característica básica de la calidad de un desarrollo y, como tal, debe tenerse en cuenta en las diferentes iteraciones del producto.

5 LA SEGURIDAD EN DEVOPS

5.1. Seguridad desde el diseño y por defecto

Security by default, o seguridad por defecto, es uno de los principios de la seguridad de la información, el cual es exigido incluso en la regulación (citemos el Esquema Nacional de Seguridad, la directiva NIS2 o incluso la protección de datos, entre otras) y que también en DevSecOps aplica como enfoque para garantizar que la seguridad se incorpore en todo el proceso de desarrollo y operaciones de software.

Este enfoque busca establecer la seguridad como una característica predeterminada y prioritaria a lo largo de los desarrollos, en lugar de agregarla, como tradicionalmente se solía hacer, al final de los mismos. Además de ser un enfoque eficaz también lo es eficiente pues abordar la seguridad desde las fases tempranas es mucho más económico en términos de esfuerzo y coste que solucionar los problemas durante la implementación o la operación que podrían llegar a forzar a un rediseño.

Este principio no quiere decir que la seguridad sólo se contempla al principio del proceso, lo que verdaderamente significa es que la seguridad se considerará en todas las etapas del ciclo de vida del software, desde el diseño hasta la implementación, el mantenimiento y la operación. La seguridad es una responsabilidad compartida por todo el equipo, incluyendo a los desarrolladores, los operadores y los responsables de seguridad. Esta implicación y presencia a lo largo de todo el ciclo de vida del desarrollo debería ayudar a que la aplicación o software sea más seguro y confiable y consecuentemente a reducir la posibilidad de brechas de seguridad y caso de producirse a un menor impacto; en definitiva, a un menor riesgo asociado al desarrollo.

Existe mucha literatura acerca de metodologías de seguridad por diseño, por ejemplo, siguiendo los principios básicos tomados de la Guía de Desarrollo de OWASP (owasp.org) una estrategia sólida debería basarse en los siguientes pilares:

1. Minimizar la superficie de ataque. Cada vez que un programador agrega una función a su aplicación, aumenta el riesgo de una vulnerabilidad de seguridad. Este principio restringe las funciones a las que los usuarios tienen acceso para reducir posibles vulnerabilidades. Por ejemplo, podría codificar una función de búsqueda en una aplicación. Esa función de búsqueda es potencialmente vulnerable a ataques de inclusión de archivos y ataques de inyección SQL. El desarrollador podría limitar el acceso a la función de búsqueda, de modo que solo los usuarios registrados pudieran usarla, reduciendo así la superficie de ataque y el riesgo de un ataque exitoso.

2. Establecer valores predeterminados seguros. Este principio establece que la aplicación debe ser segura por defecto. Por ejemplo, debe haber reglas de seguridad sólidas para cómo se manejan los registros de usuarios, con qué frecuencia se deben actualizar las contraseñas, cuán complejas deben ser las contraseñas, la utilización de doble factor, etc. Los usuarios de la aplicación podrían llegar a desactivar algunas de estas características (si así se determina), pero deberían estar configuradas en un nivel de seguridad alto de forma predeterminada.

3. El principio de menor privilegio. Establece que un usuario debe tener el conjunto mínimo de privilegios necesarios para realizar una tarea específica. Se puede aplicar a todos los aspectos de una aplicación web, incluidos los derechos de usuario y el acceso a recursos. Por ejemplo, un usuario que se registra en una aplicación de blog como "autor" no debería tener privilegios administrativos que le permitan agregar o eliminar usuarios. Solo se les debería permitir publicar artículos en la aplicación.

4. El principio de defensa en profundidad establece que múltiples controles de seguridad que aborden riesgos de diferentes maneras son la mejor opción para asegurar una aplicación. Por ejemplo, en lugar de tener un control de seguridad para el acceso del usuario, tendría múltiples capas de validación, herramientas adicionales de auditoría de seguridad y herramientas de registro o en lugar de permitir que un usuario inicie sesión con solo un nombre de usuario y una contraseña, usaría una verificación de la dirección IP, un sistema de Captcha, registro de sus intentos de inicio de sesión, detección de fuerza bruta, y así sucesivamente.

5. Fallar de forma segura Hay muchas razones por las que una aplicación web podría fallar al procesar una transacción. Las aplicaciones deben fallar de manera segura. El fallo no debe dar al usuario privilegios adicionales ni mostrar información confidencial como consultas de base de datos o registros.

6. Gestionar correctamente las relaciones de confianza. En el panorama actual de creación de aplicaciones, lo normal es que una aplicación tenga múltiples interacciones con servicios externos. Hay que tener en cuenta siempre que estos actores dentro de la cadena de suministro software de las aplicaciones de una compañía pueden suponer problemas de seguridad y chequearlos de manera continua

7. Separación (segregación) de funciones como principal medida de seguridad ante posibles fraudes. Los tipos de usuario, permisos y roles deben estar diseñados para que no exista posibilidad de fraudes, por ejemplo, de un usuario sin privilegios que pueda ejecutar tareas administrativas o a nivel funcional; por ejemplo, que el usuario que realiza el pedido de un material en una gran empresa no sea el mismo que lo recepcione, gestione el inventario y autorice el pago.

8. Evitar seguridad por oscuridad. Deben implementarse los controles de seguridad para mantener las aplicaciones seguras sin necesidad de acudir a la seguridad por oscuridad. Para esto, cifrados, una correcta aplicación del mínimo privilegio y separación de funciones, serán clave.

9. Keep security simple. Los desarrolladores deben evitar el uso de arquitecturas muy sofisticadas al desarrollar controles de seguridad para sus aplicaciones. Tener mecanismos muy complejos puede aumentar el riesgo de errores y, por lo tanto, de vulnerabilidades de seguridad. En lugar de eso, se debe buscar una solución simple y efectiva que cumpla con los requisitos de seguridad.

10. Solucionar problemas de seguridad de manera adecuada. Si se identifica un problema de seguridad en una aplicación, los desarrolladores deben determinar la causa raíz del problema. Si la aplicación utiliza patrones de diseño, es probable que el error esté presente en múltiples sistemas. Por lo tanto, los programadores deben tener cuidado de identificar todos los sistemas afectados y tomar las medidas necesarias para solucionar el problema en todos ellos.

Hablando de una metodología DevSecOps, los conceptos de "seguridad por diseño y por defecto" encajan a la perfección, ya que, como se ha visto y se continuará exponiendo en este documento, de una manera muy básica: se trata de aplicar una serie de principios fundamentales de seguridad en cada una de las fases de un diseño y serán estos principios o requisitos los que, con su definición, acompañarán de manera automática la estrategia de seguridad por defecto de la compañía.

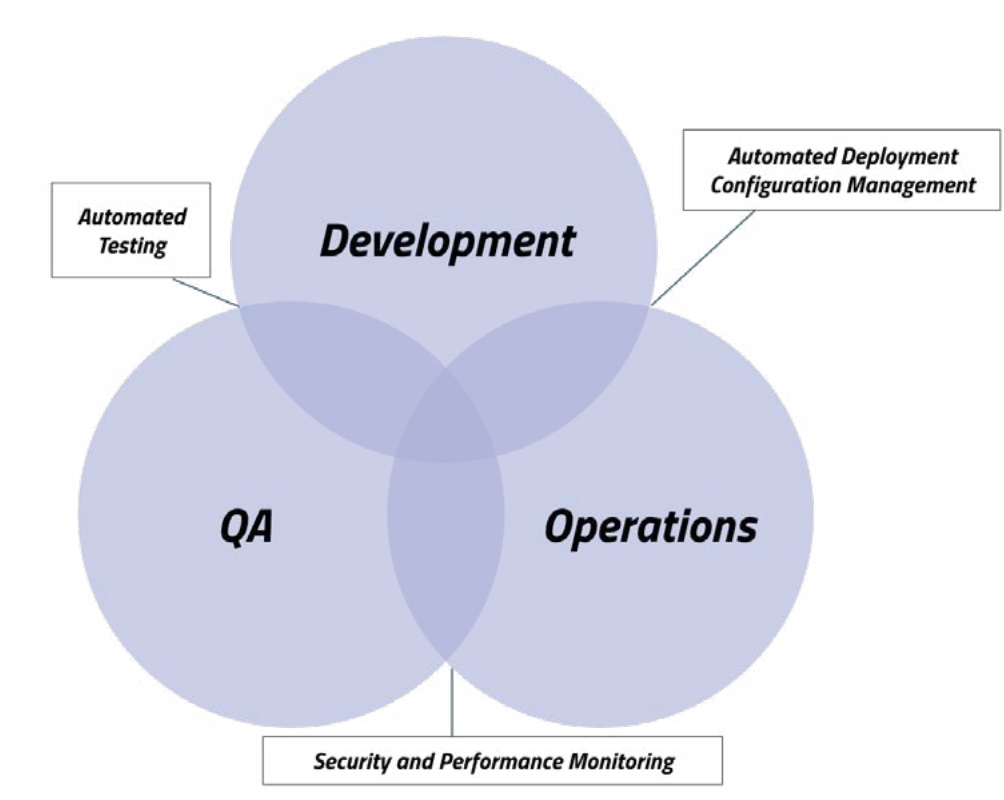
5.2. Equipos de desarrollo DevSecOps

—5.2.1. ¿Qué es DevSecOps? En este apartado abordaremos la definición y origen del término

Empresas de todo el mundo ya han cambiado el enfoque ante las ciberamenazas lo que ha provocado que la seguridad de la información se incorpore a la práctica totalidad de los procesos y consecuentemente al desarrollo; el enfoque DevOps no podía ser una excepción. Fruto de esa incorporación de la Seguridad (SEC) al DevOps nace DevSecOps.

Gracias a DevSecOps, compañías de todo el mundo podrán impulsar que la automatización de la modelación de la amenaza, la valoración del riesgo y la seguridad sean aspectos fundamentales del desarrollo de producto.

Las claves del DevSecOps están reflejadas en el siguiente gráfico, dónde se ve una interacción entre los equipos de pruebas, desarrollo y operaciones, llevando a cabo test automatizados, monitorización de la seguridad y rendimiento y gestión de la configuración automática sobre el desarrollo.



Partiendo de la situación del desarrollo seguro con:

- Procesos manuales repetitivos
- La implementación requiere de días a semanas
- No repetible y propenso a errores
- La intervención humana provoca inconsistencias
- Tiempo de inactividad frecuente
- Más fácil: se requiere menos habilidad técnica
- Los equipos trabajan en silos
- No se realizan pruebas de seguridad sobre el código de forma prematura

Llegamos a DevSecOps con:

- Configuración automatizada e implementación de software
- La implementación toma minutos
- Proceso continuo y repetible
- Consistente
- Tiempo de inactividad mínimo
- Más difícil, se necesita de más habilidad técnica, pero el proceso colaborativo de equipos ayuda en la consecución de los objetivos
- Colaboración continua entre los equipos
- Se realizan pruebas de seguridad tempranas y automatizadas durante la codificación

DevSecOps es una práctica en AppSec (Application Security) que introduce la seguridad en el SDLC (Software development life cycle) de forma temprana evitando relegarla a la fase final. También se trata de una colaboración entre los equipos de desarrollo y operaciones con el área de seguridad para integrar la seguridad en el ciclo de vida del software.

DevSecOps requiere un cambio de cultura, en procesos de una compañía y debe acompañarse de un conjunto de herramientas para habilitar a los equipos en la distribución de la seguridad en una organización. Ante todo, no debe implicar un freno en el ciclo de vida del software ni una merma de la seguridad, por ello es fundamental incorporar pasos como testing y riesgo de mitigaciones antes de un despliegue en producción e integrar dichos pasos en un ciclo continuo de despliegue e integración (CI/CD).

El concepto parte de "shift left" (testing antes del proceso de desarrollo), el cual integra la seguridad en una fase inicial y no final. Ello habilita a los desarrolladores a validar la seguridad de su código durante el desarrollo de el mismo evitando dejarlo al final del SDLC.

La integración de DevSecOps engloba todo el SDLC: desde la planificación y diseño de un software hasta la fase de codificación, construcción, testing y despliegue. Dicho proceso ofrece un continuo feedback en tiempo real del nivel de riesgo de un desarrollo.

DevSecOps se basa en los mismos tres pilares en los que la seguridad se asienta: Organización, Procesos y recursos (tecnológicos y humanos). Mediante la unión de los tres pilares conseguimos equipos más autónomos y con capacidad de entrega mejor y más Agile.

—5.2.2. Organización. Fundamentos de un equipo DevSecOps

Para la correcta integración de una cultura DevSecOps se deben dar los siguientes fundamentos:

- Principio de seguridad operativa compartida: cuando la responsabilidad de la seguridad es compartida por toda la organización, en lugar de aislarla en un solo equipo, los incidentes de seguridad son resueltos antes del despliegue de ellos mismos. Dicho principio tiene como efecto la reducción de costes en resoluciones de incidentes o perdidas de información.

- Incremento de la velocidad y frecuencia en la entrega de nuevos servicios: mediante la integración de actores en todo el desarrollo y la adición de herramientas automáticas obtenemos un proceso automatizado que evita de procesos manuales en aspectos de ciberseguridad.

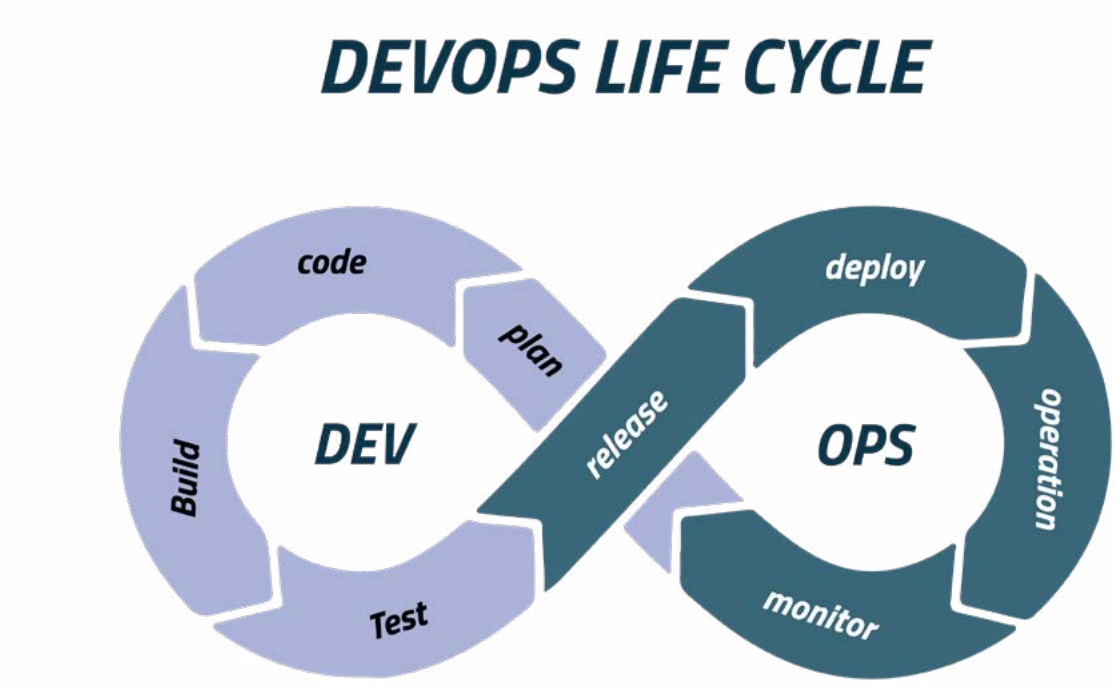
- Formaciones en ciberseguridad a lo largo de toda la organización: no solo debemos hacer foco en añadir vigilantes en todos los procesos del desarrollo de software, además hay que darles el conocimiento actualizado en aspectos de ciberseguridad. Las formaciones pueden ser desde aspectos relacionados con defensa en programación, detección de phishing corporativo, toma de requisitos no funcionales relacionados con aspectos de ciberseguridad, etc.

—5.2.3. Procesos. ¿Cómo aplicar DevSecOps?

Tal y como ya hemos explicado el término DevOps se asocia a metodologías como Continuous Delivery o desarrollo ágil, focalizando la integración entre desarrolladores y administradores de sistemas para poder obtener una mayor calidad del software en un menor tiempo.

Como podemos observar en la siguiente gráfica DevOps elabora un ciclo continuo de entrega de software focalizado en:

- Planificación de los nuevos desarrollos
- Desarrollo de las nuevas funcionalidades
- Construcción del artefacto desarrollado
- Testing del artefacto generado
- Preparación de los cambios a desplegar en un entorno de producción
- Despliegue y configuraciones del entorno
- Monitorización del software para controlar errores o performance.



El desarrollo de aplicaciones está sujeto a vulnerabilidades de seguridad en el ciclo DevOps, por ello integrar la seguridad en todo el ciclo es fundamental para asegurarnos que nuestro desarrollo es seguro. Esa incorporación de la seguridad lo haremos incorporando los siguientes procesos:

- Secure by design: tras tener claro nuestro ecosistema, es importante tener en cuenta que evolucionará y que se verá retada por atacantes cada día que pasa. Por ello la seguridad durante el diseño es fundamental para retar las soluciones técnicas propuestas, este proceso es conocido como Threat Modeling. Modelado de amenazas (Threat Model): es la fase más fundamental cuando no la más importante; se trata de la evaluación de riesgos durante la planificación del desarrollo El objetivo consiste en determinar cuáles son las amenazas que requieren mitigación y los modos de hacerlo. Esta fase anticipa vulnerabilidades antes de existir y permite reducir los tiempos de entrega segura. A alto nivel los pasos de esta fase serían:

    o Paso 1. Recopilar información general (identificar los casos de uso, los flujos de datos, identificar y comprender los componentes que son de confianza y los que no los son, el modelo de administración, configuración y mantenimiento, crear la lista de dependencia internas junto con la auditoría de la infraestructura, crear la lista de dependencias externas, entre otra información a recopilar)

    o Paso 2. Crear y analizar el modelo de amenazas

    o Paso 3. Revisión de amenazas

    o Paso 4. Identificar técnicas y tecnologías de mitigación

    o Paso 5. Documentar el modelo de seguridad y las consideraciones de implementación

    o Paso 6. Implementar y probar mitigaciones

    o Paso 7. Mantener el modelo de amenazas sincronizado con el diseño

- Validación de cumplimiento (Compliance validation): identificación y validación de los estándares en seguridad aplicados a nivel mundial, o los que específicamente se requieran por estar sujetos a una regulación, cuyo cumplimiento asegura que nuestro desarrollo de software verá reducido su riesgo.

- Revisión de código (Code review): un principio básico de la seguridad de software es el de "los dos pares de ojos" ("4 eyes principle") por el cual todo cambio en software ha de pasar por la validación de otra persona para evitar la manipulación de insider (en este caso un trabajador que por iniciativa propia o por acuerdo con ciberdelincuentes decide introducir vulnerabilidades en un software).

- Automatización de tests de seguridad: la actividad anterior no reduce el riesgo a cero, es por ello que es importante la implantación de herramientas automáticas que validan nuestro desarrollo y reportan vulnerabilidades, es momento de integrar herramientas de seguridad automática. Esta integración debe ser parte de los tests unitarios y en caso de fallo ser bloqueante en una entrega nueva. Como veremos más tarde, debemos contar con análisis estáticos y dinámicos dentro de este apartado. Tampoco deberíamos olvidarnos de los test de penetración en las aplicaciones y servicios.

- Monitorización de la Seguridad (Security Monitoring): todos los aspectos anteriores nos permiten tener una cierta seguridad en cada desarrollo nuevo (orientados a un despliegue seguro), pero nos falta una pieza fundamental que es controlar el estado en producción. es por ello que es fundamental vigilar nuestra aplicación y evaluar actuaciones para mantenerlo operativo y seguro. Para ello la monitorización es clave para vigilar que nuestra infraestructura, servicios o redes se mantienen a salvo.

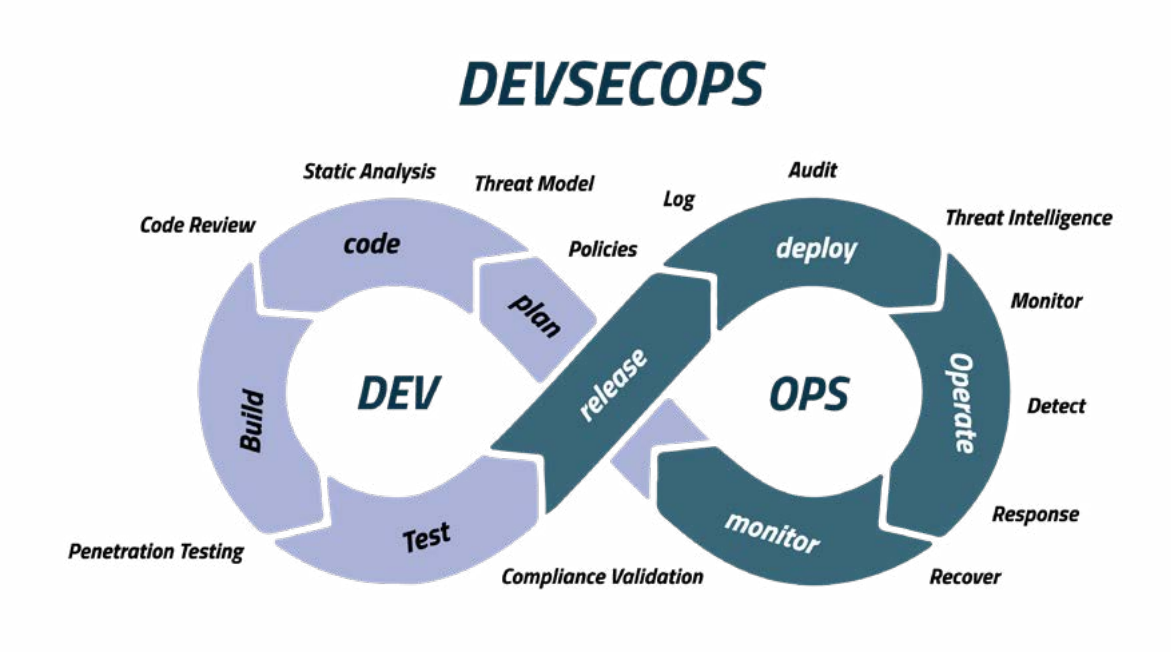
  En este punto debemos señalar que no debemos olvidar la gestión de las vulnerabilidades derivadas del código de los servicios de la infraestructura.

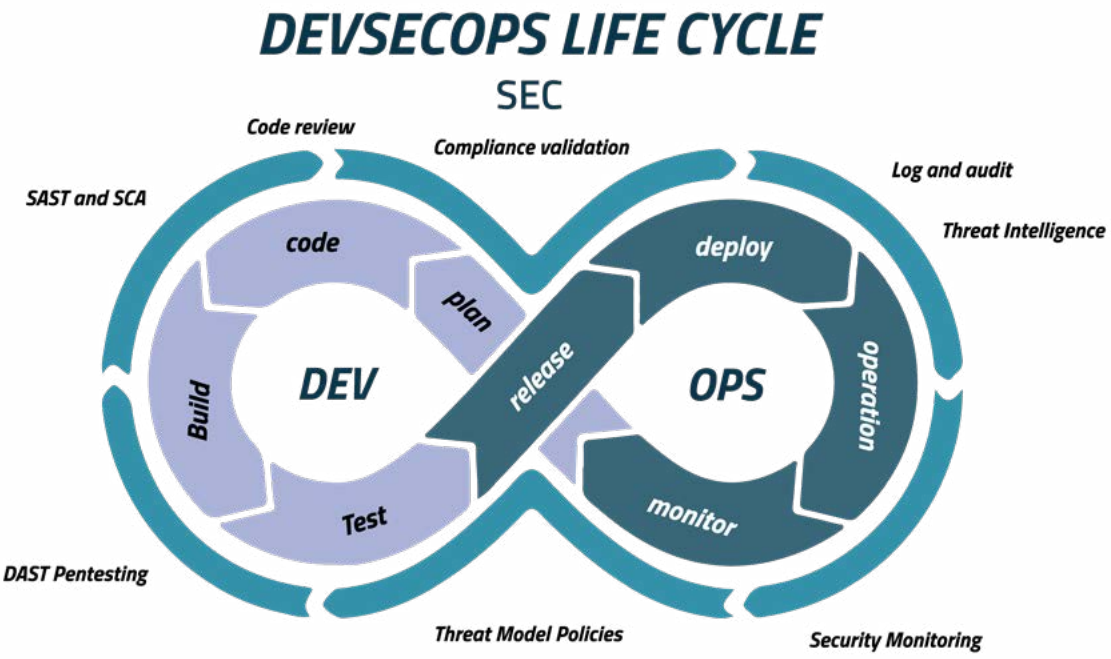
- Registro y auditabilidad (Log & Audit): una fuente fundamental de lo que sucede en nuestra aplicación son los ficheros de log donde queda traza de todas las actividades. Dichos ficheros deben contener la información necesaria para resolver ataques contra nuestro software.

  Debemos auditar completamente llos servicios, procesos e infraestructuras y hacerlo desde el punto de vista de un atacante para así identificar los puntos débiles. Gracias a esto podremos establecer contramedidas frente a potenciales brechas de seguridad y evitar impactos en forma de pérdida o degradación de la operación, sanciones/multas/reclamaciones, pérdidas económicas, perdidas de oportunidad de negocio y /o daños reputacionales.

- Inteligencia (Threat intelligence): en esta fase abordamos el análisis de datos, incluidos los derivados de la identificación del modelo de amenazas (por ejemplo, la técnicas, tácticas y procedimientos de los grupos identificados en el modelado o la información de fuentes abiertas OSINT) para prevenir y atajar ataques de ciberdelincuentes.

Estos procesos se han incorporado a la gráfica DevOps anteriormente descrita, resultando una nueva representación; DevSecOps:





Adicionalmente debemos citar otros procesos relevantes que sin embargo no están representados en el gráfico resumen anterior:

- Elección tanto del marco de desarrollo como del marco de desarrollo seguro (sobre este último marco hablaremos en el capítulo siguiente).
- Concienciación, formación y mutuo entendimiento: fundamental ofrecer a los empleados formaciones en como incorporar las políticas y las prácticas en seguridad en todo proceso que realicen. La transparencia es la llave para la confianza, por ello hacer saber a tus empleados de la necesidad de la seguridad y de cómo pueden ayudar a conseguir el objetivo. Por ejemplo: en desarrollo seguro aplicar citado principio "4 eyes principle" es fundamental para establecer la confianza y la seguridad en cada cambio elaborado en el software.
- Recluta de Security Champions: algunos desarrolladores sobresalen en su entusiasmo en ciberseguridad, reclutarlos como Security Champions añadirá mayor capacidad de seguridad y motivará a los empleados en un plan de carrera en la organización.
- Cambio de cultura "from top to down": para que funcione DevSecOps la seguridad de la información ha de formar parte del ADN de una organización desde la directiva, managers, producto owners, desarrolladores, marketing, seguridad física, etc.
- Ruptura de silos: la práctica de DevSecOps debería fomentar una mejora de comunicación entre distintas áreas en la que no se ocultaran datos sensibles para las partes.

—5.2.4. Beneficios de DevSecOps.

Podemos obtener grandes beneficios de adoptar una cultura DevSecOps, el más destacado es el incremento de velocidad en la entrega de software seguro.

Los beneficios obtenidos de aplicar una cultura DevSecOps en nuestra organización serán:

- Seguridad proactiva: DevSecOps integra las mejores prácticas en ciberseguridad en el proceso del desarrollo de software, como las ya citadas (modelado de amenazas, el principio de los cuatro ojos o las revisiones de código, escaneo de vulnerabilidades de seguridad o auditoria de los logs) que estando orientadas a afrontar los problemas antes de que sucedan proporcionan beneficios tan claros como evitar errores graves en la fase final, un descenso en la tasa de éxito de los ciberataques, procesos más eficientes de entrega reduciendo la carga en resolución de incidentes, mejoras en la eficacia y eficiencia en la investigación de incidentes, etc. Todas ellas, implementadas correctamente redundan en una reducción del riesgo (reduciendo la probabilidad y el impacto).

- Rapidez y ahorro de costes en el desarrollo de productos: los incidentes en seguridad requieren de tiempo de desarrolladores para resolverlos y no focalizarse en mejorar el software. Estos cuellos de botella son minimizados mediante DevSecOps previéndolos y solventándolos antes. Esto permite ahorrar costes y poder enfocar en otros aspectos del producto.

- Evolutivo y protección completa: DevSecOps apoya una cultura donde la seguridad es multicapa y aplica en todo el ciclo de vida del desarrollo. Permite automatizarse y adaptarse a los nuevos requisitos.

- Incremento de la calidad: DevSecOps incrementa la seguridad global a través de la automatización de la cobertura de código. Esto permite identificar, resolver y aplicar los patch de seguridad en vulnerabilidades de forma rápida.

6 MARCOS DE DESARROLLO SEGURO, FORMATIVOS Y BUENAS PRÁCTICAS

6.1. ¿Por qué ayudarse de marcos de desarrollo seguro?

Un marco de desarrollo seguro permite establecer un orden a la hora de adoptar una nueva práctica del que se tiene un total desconocimiento. El establecimiento de una referencia va a permitir a una organización tener un orden de cumplimiento interno, basándose en prácticas de compliance, con el fin de garantizar que se respeten ciertas normas específicas de un sector concreto.

Entendiendo varios marcos SecDevOps con los que alinearse, será de gran ayuda para cumplir con la exigencia del programa de cumplimiento de normas que tiene que adecuarse a las necesidades y la dinámica comercial de cada organización. De esta forma, conociendo estos marcos y viendo las ventajas y desventajas que aportan, vamos a tener ya cierto bagaje para posicionarse de cara a uno u otro marco, creando las bases para establecer unos requerimientos para adoptar esta práctica.

A continuación, se van a mostrar los marcos de desarrollo más destacados, y que van a sentar las bases para el ciclo de desarrollo seguro del software basado en la realización de diferentes chequeos de seguridad continuos durante el proyecto que se encuentra en construcción.

6.2. OWASP

Ya hemos introducido a OWASP (Open Web Application Security Project) dentro de la necesidad de construir desarrollos seguros desde el diseño y por defecto, si bien este proyecto requiere de una mayor atención, en nuestro estudio. OWASP es un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que un software sea inseguro. Este proyecto está gestionado por la FUNDACIÓN OWASP, organización sin ánimo de lucro que apoya y gestiona todos los proyectos que surgen en OWASP. La propia comunidad OWASP está formada por diferentes empresas, organizaciones educativas y particulares de todo el mundo con el fin de crear artículos, metodologías, documentación, herramientas y tecnologías, que una vez publicadas, están a libre disposición de forma gratuita para cualquier persona.

Como la organización sin fines de lucro más grande del mundo para la seguridad del software, OWASP, define su misión: "No más software inseguro". Sus objetivos son los siguientes: Apoyo a la construcción de proyectos impactantes, desarrollo y promoción de comunidades a través de eventos y reuniones de capítulos en todo el mundo, y creación publicaciones y recursos educativos. Todos estos objetivos con el fin de permitir que los desarrolladores escriban un mejor software y que los profesionales de la seguridad consigan un mundo del software más seguro.

Los valores fundamentales del proyecto OWASP van en línea con su misión y objetivos:

- Abierto: Todo en OWASP es radicalmente transparente desde las finanzas hasta el código.
- Innovador: apoyo a la innovación y los experimentos para encontrar soluciones a los desafíos de seguridad del software.
- Global: se promueve que cualquier persona en todo el mundo participe en la comunidad OWASP.
- Integridad: la comunidad es respetuosa, solidaria, veraz y neutral respecto a los proveedores.

—6.2.1. OWASP TOP 10

Uno de los proyectos destacados de OWASP, es el OWASP TOP 10. Este proyecto es un estándar de concienciación para desarrolladores e ingenieros de seguridad de aplicaciones, que representa un amplio consenso sobre los riesgos de seguridad más críticos para las aplicaciones web durante los últimos años.

El proyecto ha sido reconocido mundialmente por los desarrolladores como el primer paso hacia una codificación más segura.

Las organizaciones pueden adoptar este documento como referencia para conocer los mayores riesgos y comenzar el proceso que garantice que sus aplicaciones web logran minimizar y eludir estos riesgos. Se puede decir que el uso de OWASP TOP 10 es el primer paso más efectivo para promover un cambio en la cultura del desarrollo de software dentro de una organización a una que produzca código más seguro.

La última actualización del proyecto OWASP TOP 10 tuvo lugar en 2021, actualizando la versión anterior de 2017. En el nuevo documento se destacan tres nuevas categorías, 4 categorías con cambios de alcance y otras dónde se han producido consolidaciones de varias categorías anteriores en una única. A continuación, se describen las 10 categorías de la versión 2021:

- A01:2021-Broken Access control. El control de acceso se mueve hacia la primera posición de riesgos desde la quinta. El 94% de las aplicaciones se probaron en busca de algún control de acceso que no estuviera protegido. Las 34 CWEs (Common Weakness Enumerations) fueron las acciones comunes más empleadas para ganar el control de una aplicación y tuvieron el mayor impacto que cualquier otra categoría.
- A02:2021-Cryptographic Failures. Los fallos en criptografía suben una posición al número dos, anteriormente conocido como exposición de datos confidenciales, que era un síntoma general en lugar de una causa raíz. El enfoque renovado está en que los fallos relacionados con la criptografía a menudo conducen a la exposición de datos confidenciales o al compromiso del sistema.
- A03:2021-Injection. La inyección de código cae una posición hasta la tercera categoría. El 94 % de las aplicaciones se probaron para detectar algún tipo de inyección, y los 33 CWEs asignados a esta categoría tienen la segunda mayor cantidad de casos en las aplicaciones. El Cross-site Scripting, vulnerabilidad conocida por permitir la inyección de código en páginas web visitadas por el usuario, ahora es parte de esta categoría en esta edición.
- A04:2021-Insecure Design. El diseño inseguro es una nueva categoría para 2021, con un enfoque en los riesgos relacionados con los fallos de diseño de la aplicación. Si realmente se quiere conseguir evolucionar como industria, se requiere un mayor uso del modelado de amenazas, patrones y principios de diseño seguro y arquitecturas de referencia.

- A05:2021-Security Misconfiguration. La configuración incorrecta de seguridad sube del número 6 en la versión anterior, dónde el 90 % de las aplicaciones se probaron para detectar algún tipo de error de configuración. La tendencia ha sido promover un software altamente configurable, por ello, no sorprende ver que esta categoría sube. La categoría anterior para entidades externas XML (XXE), vulnerabilidad presente en las aplicaciones que analizan entradas XML, ahora forma parte de esta categoría.
- A06:2021-Vulnerable and Outdated Components. Esta categoría sube del puesto 9 en 2017 y es un problema conocido que cuesta probar y evaluar el riesgo. Es la única categoría que no tiene vulnerabilidades y exposiciones comunes (CVE) asignadas a los CWE incluidos.
- A07:2021-Identification and Authentication Failures. Los errores de identificación y autenticación anteriormente eran autenticación rota. Ahora incluyen CWE que están más relacionadas con fallos de identificación. Esta categoría sigue siendo una parte integral del Top 10, pero la mayor disponibilidad de marcos estandarizados parece estar ayudando a crecer esta categoría en importancia.
- A08:2021-Software and Data Integrity Failures. Los fallos de integridad de datos y software es una nueva categoría para la versión 2021, que se enfoca en hacer suposiciones relacionadas con actualizaciones de software, datos críticos y canalizaciones de CI/CD sin verificar la integridad. Uno de los impactos ponderados más altos de los datos de Common Vulnerability and Exposures/ Common Vulnerability Scoring System (CVE/CVSS) asignados a los 10 CWE en esta categoría.
- A09:2021-Security Logging and Monitoring Failures. Los errores de registro y monitoreo de seguridad anteriormente eran "registro y monitoreo insuficientes". Esta categoría se expande para incluir más tipos de fallos, es difícil de probar y no está bien representada en los datos de CVE/CVSS. Sin embargo, los errores en esta categoría pueden afectar directamente la visibilidad, las alertas de incidentes y el análisis forense.
- A10:2021-Server-Side Request Forgery. La falsificación de la solicitud del lado del servidor se agregó nueva a OWASP TOP 10. Esta categoría representa el escenario en el que los miembros de la comunidad de seguridad OWASP indican que tiene importancia, aunque no se muestra esta tendencia de ataque en los datos en este momento.

—6.2.2. OWASP CLASP

CLASP (Comprehensive Lightweight Application Security Process) es un proyecto del OWASP que establece una serie de actividades, roles y buenas prácticas dirigidas a coordinar los procesos de desarrollo seguro de software.

La organización OWASP CLASP se asienta en cinco perspectivas o vistas que abordan los conceptos generales de esta metodología, la distribución de funciones, la valoración de las actividades aplicables, la implementación de estas actividades, y el listado de problemas que pueden dar lugar a la aparición de vulnerabilidades.

CLASP define una vulnerabilidad de seguridad como un fallo en un entorno de software, especialmente en una aplicación, que permite a un atacante asumir privilegios dentro del sistema, utilizar y regular su funcionamiento, comprometer los datos que contiene, o asumir confianza no otorgada al atacante.

Una vulnerabilidad de seguridad ocurre en una aplicación de software cuando cualquier parte de ella permite un incumplimiento de la política de seguridad que la rige.

CLASP identifica 104 tipos de problemas subyacentes que forman la base de la seguridad de vulnerabilidades en el código fuente de la aplicación. Un tipo de problema individual en sí mismo, a menudo no es una vulnerabilidad de seguridad, con frecuencia es una combinación de problemas que crean una falsa condición de seguridad que conduce a una vulnerabilidad en el código fuente.

Los 104 tipos de problemas se clasifican según CLASP en 5 categorías de alto nivel, dónde cada tipo de problema puede tener más de una categoría matriz.

CLASP define una consecuencia de una vulnerabilidad explotada o explotable como un fallo en uno o más de estos servicios básicos de seguridad:

- Autorización (control de acceso a recursos)
- Confidencialidad (de datos u otros recursos)
- Autenticación (establecimiento e integridad de la identidad)
- Disponibilidad (negación de servicio)
- Responsabilidad
- No repudio

Por otro lado, CLASP contempla las fases del ciclo de vida del desarrollo de software, dónde una vulnerabilidad relacionada con la seguridad puede afectar a los sistemas y tener bien definidas estas fases va a permitir estar en alerta:

- Políticas de seguridad
- Especificaciones
- Análisis y diseño
- Implementación
- Test y operación
- Mantenimiento

Otro punto para tener en cuenta son las mejores prácticas que promueve el marco de CLASP. Si las vulnerabilidades de seguridad integradas en el código fuente de las aplicaciones sobreviven a la producción, pueden convertirse en pasivos corporativos con amplios y severos compromisos comerciales con impacto en la organización. Ante las opciones de explotar estas vulnerabilidades, no existe una alternativa razonable al uso de las mejores prácticas de aplicación en seguridad tan pronto como sea posible durante todo el ciclo de desarrollo del software.

Para ser efectivas, las mejores prácticas de seguridad de aplicaciones de software deben tener un proceso para guiar a un equipo de desarrollo en la creación e implementación de una aplicación de software que sea lo más resistente posible a las vulnerabilidades de seguridad.

Dentro de un proyecto de desarrollo de software, las mejores prácticas CLASP son la base de todas las actividades de desarrollo de software relacionadas con la seguridad, ya sea planificación, diseño o implementación, incluido el uso de todas las herramientas y técnicas que respaldan CLASP.

Estas son las Mejores Prácticas CLASP:

- **Instituir programas de concienciación.**

Los conceptos y técnicas esenciales de seguridad pueden ser ajenos a los desarrolladores de software de la organización y otros involucrados en la aplicación del desarrollo y despliegue. Por lo tanto, es imperativo desde el principio educar a todos los involucrados. Es fundamental, que los gerentes de proyecto, como fuerza impulsora detrás de la mayoría de los proyectos de actualización o desarrollo de aplicaciones, tengan en cuenta y promuevan que la seguridad ha de ser un objetivo importante del proyecto, tanto a través de la capacitación como de la rendición de cuentas.

Los programas de concientización se pueden implementar fácilmente, utilizando expertos externos o recursos, según corresponda, y generar un alto rendimiento al ayudar a garantizar que otras actividades que promuevan el software seguro se implementarán de manera efectiva.

- **Realizar evaluaciones de aplicaciones.**

La aplicación de pruebas y evaluaciones deben seguir siendo un componente central de la estrategia de seguridad. Las evaluaciones, en particular las pruebas automatizadas, pueden encontrar problemas de seguridad no detectados durante las revisiones de código o implementación. Es muy importante encontrar riesgos de seguridad introducidos por el entorno operativo, y actuar en consecuencia como un mecanismo de defensa en profundidad al detectar errores en el diseño, especificación o implementación.

- **Capturar requisitos de seguridad.**

Asegurar que los requisitos de seguridad tengan el mismo nivel de importancia que todos

otros "imprescindibles". Es fácil para los arquitectos de aplicaciones y los administradores de proyectos centrarse en la funcionalidad al definir los requisitos, ya que soportan el mayor propósito de la aplicación para entregar valor a la organización. Las consideraciones de seguridad pueden pasar fácilmente por el camino. Por lo tanto, es crucial que la seguridad en los requisitos sea una parte explícita de cualquier esfuerzo de desarrollo de aplicaciones. Entre los factores a considerar:

Una comprensión de cómo se utilizarán las aplicaciones y cómo podrían ser un ataque dirigido o malintencionado.

- Los activos (datos y servicios) a los que accederá o proporcionará la aplicación, y qué nivel de protección es apropiado dadas las necesidades de su organización.
- Apetito por el riesgo, las regulaciones a las que está sujeto y el impacto potencial en la pérdida de reputación en caso de que una aplicación sea explotada.
- La arquitectura de la aplicación y probables vectores de ataque.
- Controles compensatorios potenciales, y su costo y efectividad.

- **Implementar prácticas de desarrollo seguras.**

Actividades de seguridad definidas, artefactos, alineamientos y refuerzo continuo deben convertirse en parte de la cultura general de la organización.

- **Construir procedimientos de remediación de vulnerabilidades.**

Es especialmente importante en el contexto de las actualizaciones de aplicaciones y mejoras para definir qué pasos se tomarán para identificar, evaluar, priorizar y remediar vulnerabilidades.

- **Definir y monitorear métricas.**

No se puede gestionar lo que no se puede medir. Desafortunadamente, implementar un esfuerzo efectivo de monitoreo de métricas puede ser una tarea difícil. A pesar de ello, en este sentido, las métricas son un elemento esencial de su esfuerzo general de seguridad de aplicaciones.

Son cruciales para evaluar la postura de seguridad actual de una organización, para ayudar a centrar la atención en las vulnerabilidades más críticas y revelar qué tan bien o mal están funcionando las inversiones en mejoras de seguridad.

- **Publicar alineamientos de seguridad operacional.**

La seguridad no termina cuando una aplicación se completa y se implementa en un entorno de producción. Hay que aprovechar al máximo la red existente y las inversiones en seguridad operativa que requieren que se informe y se eduque a aquellos encargados de monitorear y administrar la seguridad de los sistemas en ejecución con asesoramiento y orientación sobre los requisitos de seguridad que exige la aplicación, junto a la mejor manera de hacer uso de las capacidades que se han integrado en el aplicativo.

6.3. MITRE – Metodología SecDevOps

The MITRE Corporation, conocida comúnmente como MITRE es una organización estadounidense sin ánimo de lucro, que participa activamente en el desarrollo de prácticas que ayuden a mejorar la ciberseguridad. Es mundialmente conocida por participar en la definición del marco MITRE ATT&CK, para la identificación de las ciberamanezas.

En este caso, ha establecido unas premisas básicas y una metodología de desarrollo seguro para ayudar a los equipos de SecDevOps. El principal propósito es describir claramente cómo la seguridad y las pruebas se pueden integrar en un entorno DevSecOps sin comprometer la velocidad, la seguridad o calidad, mientras se proporciona una referencia de terminología, las metodologías, los procesos, los entornos y las tecnologías de automatización utilizadas en los programas DevSecOps.

Las 4 claves que promueve MITRE

- **Propuesta de valor de DevSecOps**

  Los programas pueden obtener un valor significativo al implementar DevSecOps. Sin embargo, las pruebas y la seguridad no deben sacrificarse en ningún momento.

- **Desplazamiento a la izquierda "Shift Left"**

  Este término hace referencia a los esfuerzos de un equipo de DevOps para garantizar la seguridad de las aplicaciones en las primeras etapas del ciclo de vida del desarrollo. Sin embargo, se puede hacer la prueba al final del proceso de cuánto tiempo y costes se ha ahorrado al equipo, evitando fallos y resolución de errores comunes.

- **Agile y DevSecOps van juntos**

  DevSecOps debe ser alimentado por el desarrollo de software con metodología Agile. La retroalimentación en cada sprint (etapa) de desarrollo es clave y debe ser parte para conseguir un entregable de calidad.

- **La automatización es clave**

  La seguridad y la automatización de pruebas pueden reducir el tiempo de entrega, mejorar la calidad y seguridad y eliminar el error humano.

6.4. Otros marcos de desarrollo seguro

Existen otros marcos de desarrollo seguro que también pueden ser referencia para asentar SecDevOps. A continuación, se explican las líneas generales de otros dos marcos S-SDLC y SSDF.

S-SDLC (Secure Software Development Life Cycle)

Se basa en verificar los requisitos de seguridad a lo largo de las distintas fases de construcción del software: análisis, diseño, desarrollo, pruebas y mantenimiento.

Sobre todo, durante las dos primeras, ya que gran parte de las debilidades de los sistemas se generan incluso antes de comenzar las tareas de programación. Las claves del S-SDLC son la atención al detalle, para favorecer la identificación inmediata de las vulnerabilidades; y la mejora continua.

SSDF (Secure Software Development Framework)

Este marco es una iniciativa del NIST (National Institute of Standards and Technology de Estados Unidos), el cuál provee indicaciones para evangelizar a la organización acerca de la importancia de la seguridad informática; proteger el software de uso habitual ante hipotéticos ataques, orquestar un desarrollo seguro de software, y detectar y solucionar con rapidez cualquier vulnerabilidad.

7 HERRAMIENTAS

7.1. Introducción

Es importante entender que DevSecOps es una práctica que no implica de manera directa el uso de herramientas, las herramientas se tienen que entender como un facilitador para poder aplicar de manera correcta, estandarizada y ágil los controles de seguridad definidos por la organización, pero sin una definición de procedimientos y adopción correcta de la práctica DevSecOps las herramientas no servirán.

No obstante, sería prácticamente imposible aplicar de manera correcta y ágil un ciclo DevSecOps sin disponer de herramientas que nos ayuden a aplicar estos controles, el coste de tiempo y esfuerzo sería muy difícil de asumir por los equipos de seguridad y la probabilidad de fallo humano, así como la adopción a nuevos estándares de seguridad sería prácticamente imposible. En base a esto, en esta sección vamos a realizar una breve introducción a las distintas tipologías de herramientas, así como una clasificación basa en niveles de madurez que nos pueda servir de guía para saber dónde hay que priorizar o cual es el camino a seguir para poder aplicar la práctica paso a paso.

7.2. Herramientas & Clasificación

Existen múltiples herramientas que nos pueden ayudar a aplicar los controles de seguridad en el ciclo de vida de desarrollo seguro, pero el uso de estas también irá acompañado del nivel de madurez que tenga la compañía, es por ello que esta sección pretende explicar que tipos de herramientas existen por tipología y un nivel de agrupación orientativo que nos ayude a entender que sería lo básico y que camino se debería de seguir para alcanzar el nivel más alto de madurez.

Adicionalmente hemos reflejado herramientas complementarias que están relacionadas con el desarrollo seguro, aunque no tienen por qué ser estrictamente de la práctica DevSecOps.

En base a los indicado anteriormente tendríamos las siguientes tipologías de herramientas:

- Threat Modeler (Modelado de Amenazas): Herramientas destinadas al modelado de aplicaciones antes de su desarrollo y la identificación de las potenciales amenazas que podrán existir durante el desarrollo de las mismas. Estas herramientas ayudan a identificar de manera temprana las potenciales amenazas a las que nos enfrentaremos y tener de manera clara los riesgos a los que estaríamos expuestos. Al igual que las herramientas de Defect Manager, son soluciones que nos ayudan a mejorar en los procesos de desarrollo de aplicaciones, permitiendo a la organización tener un control de inicio al fin del desarrollo.

- Plugin del Idle: Herramienta que se disponibilidad como un ad-on al IDE de desarrollo. Esto permite al desarrollador ir conociendo mientras desarrolla las potenciales vulnerabilidades que tendrá su código, facilitándole la resolución en fases tempranas y adicionalmente ayudándole a aprender y mejorar en su conocimiento de desarrollo seguro. No todas las soluciones de mercado disponen de este componente o no son soportados por todos los IDE de desarrollo, pero es de gran utilidad y nos ayuda a movernos hacia el "Shift Left".

▪ SAST (Static Application Security Testing): Herramientas de análisis seguridad que evalúa el de código fuente de nuestro desarrollo informando de vulnerabilidades y calidad del mismo.

Es por tanto un análisis estático son aquellas que analizan el código fuente de las aplicaciones, línea por línea. vVan analizando e identificando potenciales vulnerabilidades en el código para que puedan ser resueltas antes de proceder a un entorno en ejecución. Estas herramientas pueden tender a tener un alto volumen de falsos positivos y es por ello que es importante establecer líneas base que nos permitan agilizar los análisis y descartar de manera rápida los falsos positivos. Algunas soluciones para mejorar en la obtención de falsos positivos analizan también el byte code o los binarios, pudiendo "simplificar" el proceso y sacar falsos positivos de aquello que podrá ser utilizado. En la práctica de DevSecOps es importante que el análisis estático tenga un nivel de eficiencia alto, esto nos permitirá resolver de manera temprana muchas de las potenciales vulnerabilidades.

▪ SCA (Software Composition Analisys): SCA (Software Composition Analysis) u OSA (Open Source Anaylsis) identifica componentes de código abierto y el riesgo de vulnerabilidades, licencias, etc. Son herramientas que analizan las librerías que componen nuestra aplicación con el objetivo de detectar vulnerabilidades en aquellos componentes que no han sido directamente desarrollo de la compañía, sino que son librerías externas, en su mayoría de código abierto, que utilizamos en nuestros desarrollos. Estas herramientas son de una importancia muy alta en el entorno actual de desarrollo, numerosos informes indican un uso muy alto de librerías externas en los desarrollos de las compañías, (debido a la existencia de muchas funcionalidades ya implementadas). estas librerías sino son analizadas de manera correcta pondrán en riesgo nuestra aplicación y será vulnerable ante ataques independientemente que nuestro código (desarrollado internamente) sea seguro.

▪ DAST (Dynamic Application Security Testing): un problema de la solución anterior es el contexto acotado a solo nuestro código, ¿qué sucede cuando una vulnerabilidad afecta a un flujo que conecta varias piezas de software? En este contexto aparecen las pruebas DAST (Dynamic Analysis Security Testing) los cuales engloban flujos de una aplicación y buscan vulnerabilidades de seguridad más complejas.

Las herramientas de análisis dinámico de código son aquellas que realizan un análisis de la aplicación en ejecución, es decir con la aplicación corriendo en un entorno controlado simulan el consumo de la misma y van reproduciendo potenciales ataques, analizando el comportamiento de la aplicación e indicando las vulnerabilidades encontradas. Estas herramientas son de gran interés para realmente validar que seguridad tendremos en la aplicación de manera monolítica, sin capas externas que nos pudieran "ocultar" vulnerabilidades directamente relacionadas con la aplicación. Por norma general las herramientas de análisis de código suelen disponibilidad las capacidades de SAST y DAST de manera conjunta, teniendo un único portal donde puedas ir viendo el conjunto de vulnerabilidades.

▪ IAST (Interactive Application Security Testing): Herramientas que realizan un análisis de la aplicación de forma interactiva, combinando pruebas de seguridad estáticas y dinámicas, permitiendo que la aplicación pueda ir aprendiendo y mejorando en la ejecución de estas pruebas y que cada vez sean pruebas más efectivas basadas en el comportamiento de la aplicación. Estas herramientas funcionan con la adición de una librería al software, y actualmente solo funcionan con aquellos lenguajes instrumentalizados

como podría ser Java, pero no serían herramientas válidas para tecnologías como JavaScript (altamente utilizado en el desarrollo de frontales). En base a esto, si bien son soluciones que realizan un análisis avanzado de seguridad en las aplicaciones actualmente la tecnología no está teniendo una penetración en mercado alta y muchas compañías optan por consolidar primero los controles SAST, DAST y SCA antes de moverse a tecnologías IAST.

▪ RASP (Runtime application Self-Protection): Herramientas que permiten la protección de la aplicación de manera autónoma mediante la inclusión de una librería en el código fuente de la aplicación, al igual que el IAST estas herramientas funcionan con lenguajes instrumentalizados, pero no serían efectivas para aquellas aplicaciones con tecnologías más enfocadas a frontal (JavaScript). Estas herramientas son efectivas para la protección de la aplicación ante ciertas carencias que puedan existir en capas superiores (como por ejemplo el WAF) pero en ningún caso las sustituyen. Por el contrario, requieren de disponibilizar el agente en todas las aplicaciones y mantenerlo en los entornos de producción y aunque el consumo de estos agentes tiende a ser bajo, es importante tenerlo en cuenta.

▪ Defect Manager (Gestión de Defectos): Herramientas destinadas a la gestión centralizada de defectos, permitiendo a los desarrolladores y responsables de seguridad realizar un seguimiento centralizado y común de los defectos y vulnerabilidades de los desarrollos. Estas herramientas son muy útiles y ayudan a generar una cultura asociada enfocada al DevSecOps, unificando equipos y permitiendo realizar tareas de seguimiento y reporting conjunto.

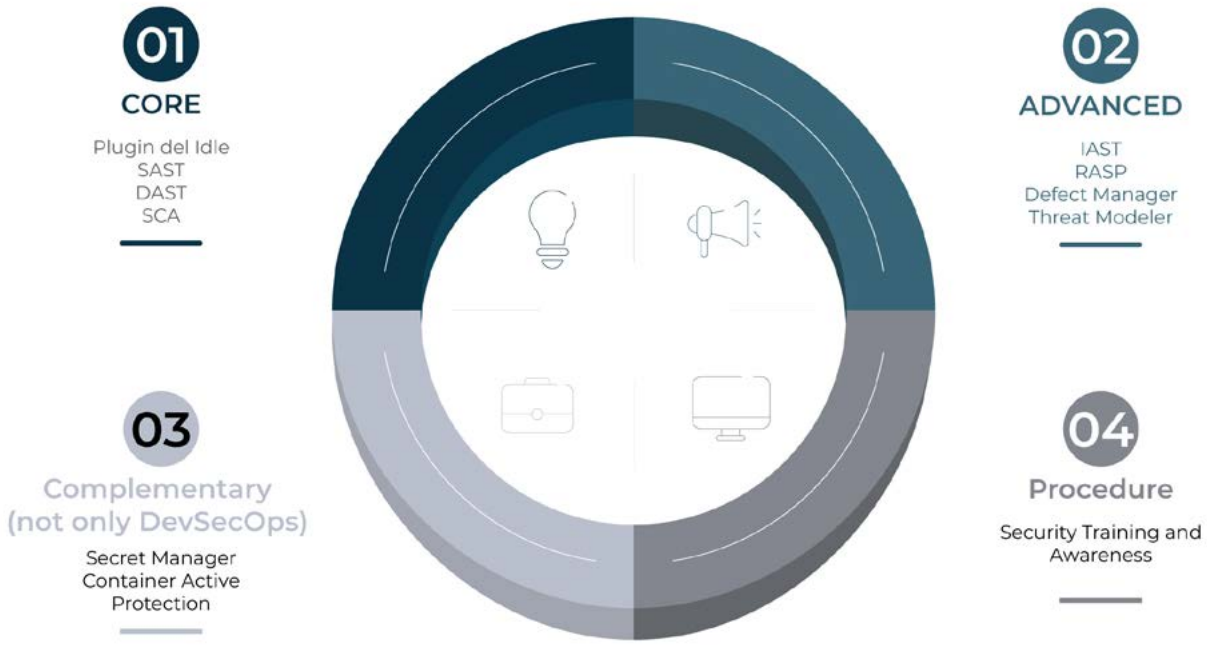
▪ Secret Management (Gestión de Secretos): Herramientas destinadas a la gestión de secretos de manera segura, no son soluciones asociadas directamente con una práctica de desarrollo seguro, pero si nos ayudan a disponibilizar capacidades de almacenamiento de credenciales de manera segura evitando de esta forma que los secretos residan en el código hard-coded.

▪ Container Active Protection (Protección activa de contenedores): Herramientas destinadas a la protección activa de contenedores en ejecución, estas herramientas nos ayudan a proteger las aplicaciones desarrolladas una vez están ejecutándose en entornos productivos, monitorizando el contenedor e identificando potenciales vulnerabilidades en runtime. Al igual que la anterior, no son herramientas únicamente vinculadas a un ciclo de vida del desarrollo seguro, pero nos ayudan a completar el ciclo y analizar nuestros aplicativos una vez están siendo ejecutados. En algunas ocasiones hay soluciones que incluso se integran en el despliegue y realizan controles de SCA.

▪ Security Training and Awareness: Herramientas destinadas a facilitar los programas de formación y concienciación para el desarrollo seguro, existen múltiples soluciones que nos permiten ofrecer este tipo de programas de manera sencilla y divertida para que la aceptación por parte de los desarrolladores sea más efectiva (CTFs, Gamificación, entre otros). El uso de estas herramientas nos permitirá mejorar la concienciación de la compañía y avanzar en el camino para obtener aplicaciones más seguras.

Una vez identificadas las principales tipologías de herramientas que podemos encontrar en el mercado para implementar una práctica DevSecOps, el siguiente gráfico pretende reflejar una clasificación de las mismas en base a los siguientes criterios:

- Core: Conjunto de herramientas necesarias para poder aplicar una práctica DevSecOps dentro de una organización.
- Advanced: Conjunto de herramientas que nos ayudan a mejorar la práctica y nos proporcionan un análisis adicional de seguridad de nuestras aplicaciones.
- Complementary: Herramientas adicionales al ciclo de vida de desarrollo seguro y que nos ayudan dentro del ciclo DevSecOps para disponibilidad aplicaciones más seguras.
- Procedure: Herramientas o procedimientos que ayudan a la mejora de la práctica de desarrollo seguro dentro de la organización.



7.3. Adopción progresiva de las herramientas

En relación con el gráfico anterior la adopción de las herramientas suele darse en cada compañía de una forma progresiva, conforme la organización va dando pasos en la madurez tecnológica del desarrollo seguro. El siguiente gráfico viene a representar ese avance.

8 AUTOMATIZACIÓN "SEC"

8.1. Introducción

En este apartado hablaremos de los métodos de automatización de las herramientas de ciberseguridad en el ciclo de vida de desarrollo más comunes. Diferentes estrategias de automatización de controles y problemas que podemos encontrar en el camino

8.2. Pipelines

Algunos autores de documentación sobre buenas prácticas en automatización de factorías de software explican el concepto de pipelines software haciendo un paralelismo a las líneas clásicas de producción "analógica" tradicionales.

Estos principios se basan en:

1. Flujo Left-to-right de producción: Se sigue un flujo definido de izquierda a derecha, donde un estado alimenta al estado siguiente para estandarizar la fabricación del producto.

2. Flujo Right-to-left para feedback: Que permite a los ingenieros solucionar problemas yendo directamente al origen de los mismos y evitar que el problema se propague por la línea de producción.

3. Entornos preparados para cambios constantes: Una factoría productiva tiene la capacidad de incluir nueva tecnología o realizar cambios de calado en algunas de sus partes sin afectar al resto y esa resiliencia hace que tenga éxito.

Volviendo al software, un pipeline de desarrollo es, por lo tanto, un conjunto de procesos automatizados que se realizan secuencialmente en el desarrollo de software. El objetivo es optimizar y acelerar el proceso de desarrollo, asegurando que la calidad y los estándares se mantengan a lo largo del ciclo de vida del producto.

Los procesos que forman parte de un pipeline de desarrollo incluyen, como mínimo, de manera automatizada, para tener una mayor eficiencia los siguientes pasos:

1. Integración de código: Los desarrolladores consolidan su código desarrollado en local en un repositorio único de código
2. Compilación: El código de los desarrolladores se convierte en una aplicación
3. Pruebas automatizadas: Se pasan las baterías de pruebas necesarias al software antes de pasar a cualquier entorno productivo
4. Validación de pruebas: Se validan las pruebas pasadas en el punto anterior y, si son satisfactorias, el artefacto está listo para poder ponerse en producción
5. Despliegue en producción

Desde el punto de vista de desarrollo software, y sólo con la definición mínima que se acaba de especificar, el uso de un pipeline de desarrollo permite detectar y corregir errores de manera temprana, en la fase de validación de pruebas lo que reduce el tiempo y los costos totales del desarrollo. Además, facilita la colaboración y la comunicación entre los equipos, ya que todos los procesos están claramente definidos y automatizados.

Durante los siguientes puntos de este apartado se explicarán buenas prácticas de ciberseguridad a incluir en los pipelines para automatizar también los análisis de ciberseguridad y detectar, de manera análoga a fallos en lógica de negocio que se detectan con los tests unitarios comunes, vulnerabilidades de ciberseguridad de una manera temprana.

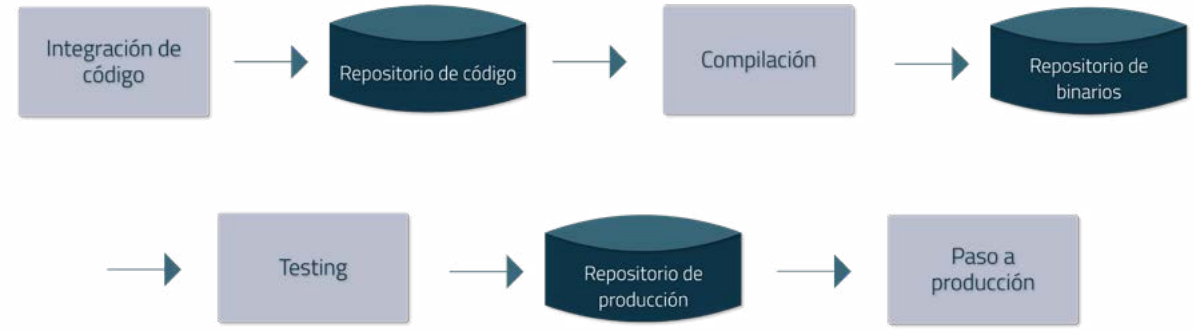
En resumen, un pipeline de desarrollo es una herramienta clave para el desarrollo ágil y eficiente de software, que permite una mayor calidad, un proceso más rápido y una mayor eficiencia en la colaboración y la comunicación del equipo de desarrollo.

8.3. Ciberseguridad en los pipelines

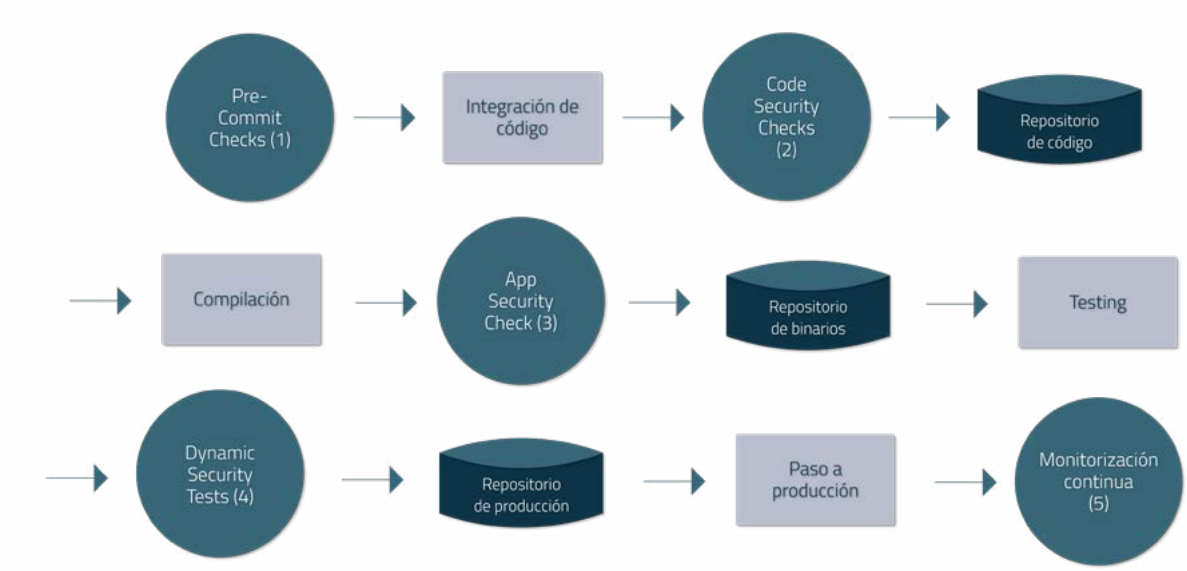
Como se ha explicado, la razón de ser de la automatización de los diferentes estadios del desarrollo de software atiende a la necesidad de agilidad y eficiencia en el proceso de desarrollo. Una parte fundamental de la calidad de un artefacto software es la ciberseguridad.

En este punto se tratará de manera breve en qué puntos de estos pipelines tienen sentido los diferentes "checks" de seguridad.

Partiendo de la definición de pipeline del punto anterior, de manera gráfica:



Observamos cómo un desarrollador, desde que sube su código hasta que pasa a producción, este se integra y pasa por diferentes estados, pero, en principio, con este enfoque, ningún test de seguridad. Aplicando las herramientas que se han especificado en el punto 4.2, la imagen sería la siguiente:



1. **Pre-Commit Checks**: Antes de que un desarrollador integre su código recién desarrollado para la compilación de la aplicación, se incluirán dos tipos de controles de seguridad que ya se han explicado en puntos anteriores del documento:

    a. Modelado de amenazas

    b. Ayudas sobre ciberseguridad en el IDE del desarrollador

2. **Code Security Checks**: Una vez el código del desarrollador ha iniciado la integración de su código hacia el repositorio principal, las herramientas de seguridad, de manera automatizada, ya disponen del código para poder analizarlo de manera preliminar para buscar problemas de seguridad que puedan derivarse de la integración de esa porción de código. En este punto, se configurarán en los pipelines las herramientas SAST y, en algunos casos, SCA si no trabajan directamente sobre el artefacto compilado y lo hacen a través del código.

3. **App Security Checks**: Aquellas herramientas de seguridad que necesitan una aplicación funcional para realizar sus comprobaciones se ejecutarán en el pipeline en este punto. Aquí se encontrará sobre todo el DAST, aunque también algunas herramientas SCA y SAST necesitan el código compilado para poder realizar su función. Dependerá de la herramienta, por tanto, que se esté utilizando, puede estar en el punto anterior, o en este.

4. **Dynamic Security Test**: En este último estado de testing, se realizarán más chequeos dinámicos, pero ya con la aplicación no solo compilada, sino desplegada en un entorno de prueba que, idealmente, será equivalente al entorno de producción. De esta manera, se podrán detectar vulnerabilidades que no dependan únicamente del código o de la lógica de negocio de la propia aplicación, sino de las integraciones con otras piezas del entorno de ejecución final.

5. **Monitorización continua**: Las vulnerabilidades, por ejemplo, en librerías que incluye la aplicación deben ser continuamente evaluadas y llevar una política de escaneos y resolución de vulnerabilidades robusta en aplicaciones en producción.

8.4. Inventariado y monitorización

A medida que los entornos basados en metodología DevSecOps se convierten en el núcleo del desarrollo tecnológico, cobra más importancia la monitorización e inventariado de estos entornos. La contrapartida a la automatización suele ser, comúnmente, la tendencia al descontrol y la agilidad tiende a convertirse en un arma de doble filo, desde el punto de vista de la ciberseguridad.

Para evitar los problemas relacionados con este "descontrol", es importante que desde el principio se tenga claro que:

1. El inventariado automatizado de repositorios y despliegues es la base de cualquier estrategia de ciberseguridad sobre un entorno DevSecOps. No se puede proteger aquello que no se conoce. La potencia de DevSecOps reside, en gran parte, en la agilidad a la hora de poder realizar parcheos de seguridad o modificación de aplicaciones de manera temprana, barata y ágil cuando aparece alguna vulnerabilidad. Para poder hacer esto, es necesario que todas las aplicaciones, su código y componentes estén inventariados de manera correcta, ya que, en cualquier otro caso, no se podrán identificar de manera ágil aquellos desarrollos que requieran acción desde ciberseguridad en caso de incidente.

2. La monitorización de los entornos DevSecOps pasa a ser fundamental. Es necesario tener en cuenta que los entornos de integración y despliegue continuo se convierten de manera automática en "joyas de la corona" de las compañías que adoptan este método de desarrollo y gestión de la infraestructura, pues es el punto donde:

   a. Se almacenan el código

   b. Se pueden almacenar secretos (keys, api-keys, etc) de manera incorrecta

   c. Se cuenta con usuarios de servicio con permisos para despliegue de infraestructura y promoción entre diferentes entornos

   d. Residen las identidades de los desarrolladores

   e. Si se han desplegado herramientas de ciberseguridad en los pipelines, estarán los informes sobre vulnerabilidades

Por lo tanto, la monitorización exhaustiva de estos entornos, de actividades anómalas sobre los pipelines, en usuarios o en infraestructura será un aspecto capital de la seguridad DevSecOps en cuanto a prevención de incidentes de seguridad.

8.5. Bloqueo VS Alerta

Por último y totalmente relacionado con la automatización de controles de seguridad, a la hora de definir el comportamiento de las herramientas de ciberseguridad desplegadas a lo largo de los pipelines, se presenta, de manera habitual, la disyuntiva entre si los reportes desfavorables que identifican vulnerabilidades en alguno de los chequeos deben prohibir el despliegue de manera automática.

Este problema es algo para lo que no hay una respuesta única universal, ya que dependerá completamente del nivel de madurez de la organización en seguridad DevSecOps y su confianza en las herramientas desplegadas, básicamente porque el principal enemigo de los departamentos de ciberseguridad, hablando de seguridad en entornos de desarrollo, son los temidos "falsos positivos".

Un "falso positivo" puede resumirse, en este contexto, en una vulnerabilidad reportada por alguna de las herramientas de análisis de seguridad, que resulta, después de un análisis pormenorizado, no ser una vulnerabilidad real. Por la complejidad de los lenguajes de programación y las diferentes maneras de desarrollar y utilizar, por ejemplo, frameworks de desarrollos en algunos lenguajes de programación, es bastante común que las herramientas de seguridad reporten gran cantidad de "falsos positivos".

Además de las herramientas (tecnológicas) de seguridad, los departamentos de ciberseguridad cuentan con algo necesario para que DevSecOps funcione y esto no es otra cosa que la confianza por parte de todos y todas las profesionales de los equipos de desarrollo. Esta confianza, en el equipo y en la tecnología de análisis de ciberseguridad se degradará de manera crítica y rápida si no se tiene el nivel de madurez suficiente y comienzan a parar despliegues de manera automatizada por "falsos positivos". Por el trabajo adicional que se pone en los equipos de desarrollo para demostrar esos errores de detección, en momentos normalmente tensos como son los pasos a producción. Por lo que la recomendación general es tener ese modo bloqueo como un objetivo, un aspiracional al que llegar cuando la madurez en el afinado de la tecnología y las reglas, y sobre todo, en la confianza mutua en el equipo sea alto.

9 CLOUD INFRAESTRUCTURA COMO CÓDIGO, CONTENEDORES, KUBERNETES Y MICROSERVICIOS

9.1. Breve introducción a la infraestructura como código

La infraestructura como código (IaC) es una metodología que se basa en la definición y gestión de recursos de infraestructura mediante el uso de código. Como si de cualquier pieza software se tratase.

Antes de la normalización de IaC, la configuración y gestión de la infraestructura se debía realizar manualmente, lo que a menudo resultaba en inconsistencias en la configuración y mayor impacto de errores humanos. Además, la administración de la infraestructura manualmente resultaba en un alto costo y una menor eficiencia, ya que se necesitaba mucho tiempo para realizar tareas repetitivas.

Con la infraestructura como código, los recursos de infraestructura se definen, despliegan y gestionan a través del uso de lenguajes de programación.

Véase el siguiente ejemplo: Existe la necesidad de TI de desplegar, por parte de un equipo de desarrolladores, cierto contenido en un servidor web. En un entorno clásico de infraestructura, las operaciones básicas serían:

1. Proveer un servidor en el datacenter con un sistema operativo. Tendría que llevarla a cabo un equipo de infraestructuras
2. Configurar la red en ese servidor. Implica equipo de redes
3. Instalar y configurar el software necesario para el servidor Web. Podría ser una instalación semi-automatizada por el equipo de infraestructura
4. Desplegar el contenido. Los desarrolladores pueden alojar su contenido en la web

En contrapartida, siguiendo una metodología DevSecOps e infraestructura como código, un único equipo sería el encargado del diseño y despliegue del código necesario que, al ser ejecutado, montaría todos los recursos necesarios de infraestructura, conexiones y paquetes software, así como su configuración, para servir el contenido web. Habitualmente en entornos de Cloud Pública.

Los principales beneficios de la utilización de infraestructura como código son:

- **Consistencia**: Con la infraestructura como código, todos los recursos se definen y gestionan a través del código, lo que garantiza que los recursos estén configurados y desplegados de manera consistente en todos los entornos. Esto ayuda a prevenir errores humanos y reduce la probabilidad de fallas en el sistema.
- **Escalabilidad**: IaC permite la escalabilidad de la infraestructura mediante la definición de plantillas que pueden ser fácilmente replicadas y configuradas para satisfacer las necesidades. Esto significa que la infraestructura puede ser escalada de manera más eficiente y rápida.
- **Eficiencia**: IaC permite la automatización de tareas repetitivas, lo que reduce la cantidad de tiempo y esfuerzo necesarios para configurar y gestionar la infraestructura.
- **Control de versiones:** IaC permite el control de versiones de la infraestructura, lo que significa que los cambios en la infraestructura pueden ser rastreados y revertidos si es necesario.
- **Colaboración**: IaC promueve la colaboración entre los equipos de desarrollo y operaciones, hasta el punto de habilitar equipos multidisciplinares DevSecOps que ayuda a eliminar la brecha entre desarrollo y operaciones y a mejorar la eficiencia en general.

En general, la infraestructura como código es una metodología que ha sido posible gracias a la evolución de herramientas y tecnologías de automatización.

9.2. Seguridad en infraestructura como código

La criticidad de los desarrollos de infraestructura como código, a la hora de definir qué recursos y con qué configuraciones se van a desplegar hacen que sea, igualmente crítica una gestión de la seguridad en los mismos.

Para ello, será necesario, de manera análoga a lo que se espera en una metodología DevSecOps para el resto de código desarrollado:

- Análisis de vulnerabilidades: Es importante realizar un análisis de vulnerabilidades en la infraestructura antes de implementarla. Esto ayuda a identificar los posibles riesgos de seguridad y corregirlos antes de que se implemente la infraestructura.
- Seguridad por defecto y desde el inicio: La ciberseguridad debe ser considerada desde el inicio del proceso de creación de la infraestructura como código. Esto significa que el código debe ser escrito con la seguridad en mente, en lugar de ser una consideración tardía.
- Políticas de seguridad definidas: Es importante tener una política de seguridad clara y definida que se aplique a la infraestructura creada con la metodología de infraestructura como código. Esto ayuda a garantizar que la seguridad se tenga en cuenta en todos los aspectos de la infraestructura. Una estrategia muy habitual en equipos con un nivel de madurez alto en Infraestructura como código es la creación de un "catálogo de servicios", una serie de plantillas con la infraestructura más común configuradas de serie acorde a las políticas y requisitos de seguridad (y validadas por el responsable de ciberseguridad), que los equipos de desarrollo simplemente "importan" en sus entornos y que contribuyen sobremanera tanto a la eficiencia de los equipos de trabajo, como a la consistencia en configuraciones seguras de los servicios de infraestructura.
- Pruebas de seguridad: Las pruebas de seguridad deben realizarse en todas las etapas del desarrollo de la infraestructura como código. Esto ayuda a identificar posibles brechas de seguridad y a corregirlas antes de que la infraestructura se implemente.
- Automatización: Se deben implementar herramientas de seguridad automatizadas, como escáneres de vulnerabilidades, para ayudar a identificar y corregir posibles riesgos de seguridad. Esto también ayuda a garantizar que la infraestructura se mantenga segura a medida que se realiza la implementación y actualización.

En resumen, la ciberseguridad es un aspecto crítico que debe considerarse en todo el proceso de implementación de la infraestructura como código. Se deben implementar medidas de seguridad en todas las etapas, desde la planificación hasta la implementación y mantenimiento, para garantizar la seguridad de la infraestructura.

9.3. Contenedores, clústers y microservicios:

—9.3.1 Contenedores

Además de la infraestructura como código, hay otra serie de avances que pueden hacer que la eficiencia de los desarrollos ágiles en modo DevSecOps se dispare desde el punto de vista tecnológico. Una de estas tecnologías es, sin duda, el despliegue de cargas computacionales en contenedores.

De manera simple, la tecnología de despliegue de cargas en contenedores permite crear y distribuir aplicaciones de manera independiente a su entorno, es decir, la generación de "paquetes" que funcionan en cualquier lugar, independientemente del servidor en el que se ejecutan, ya que todo su entorno de ejecución se desarrolla dentro del llamado "contenedor". En este "contenedor" se encuentran todas las dependencias a nivel sistema operativo y software necesarias para que se realice la ejecución del software. Parcialmente podría ser una analogía a lo que de una manera más tradicional supondrían las máquinas virtuales, solo que con unos despliegues mucho más ligeros.

Habrá ciertos conceptos básicos que debemos controlar a la hora de hablar de contenedores, son los siguientes:

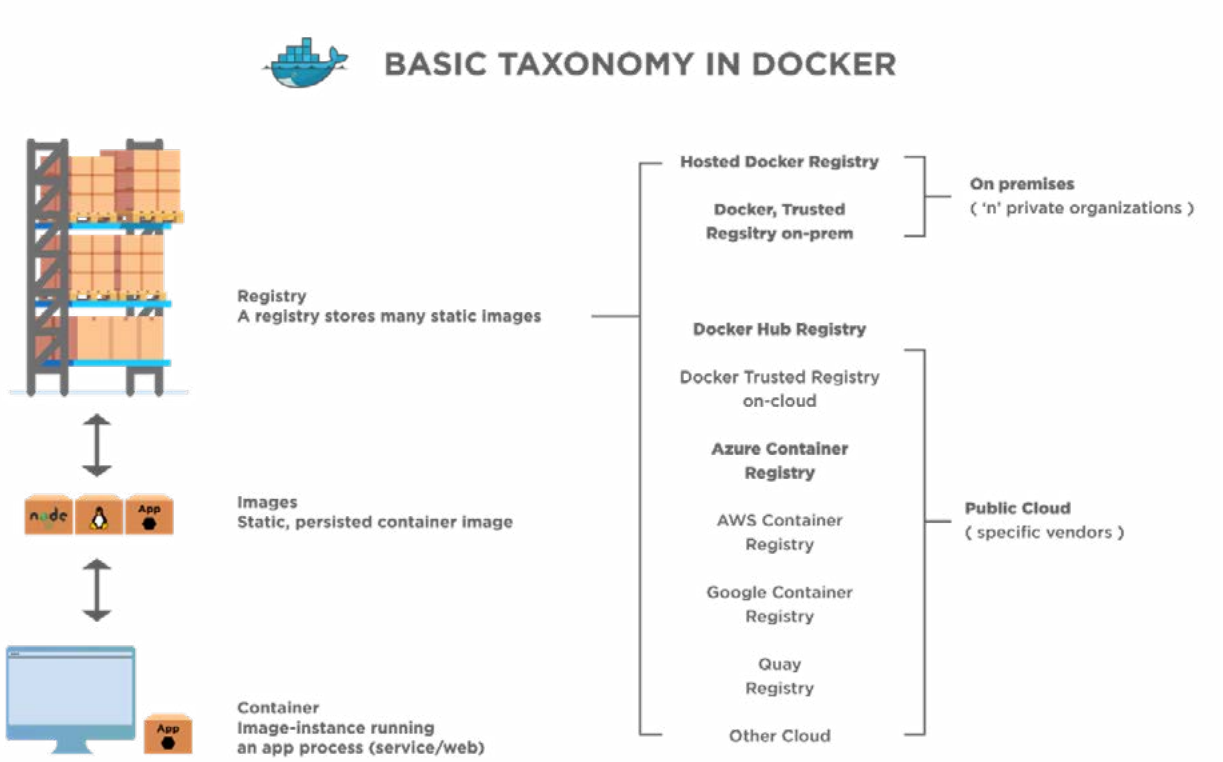
Dockerfile: Será la plantilla que generará nuestro contenedor. En este fichero se define, entre otras cosas, qué sistema operativo tendrá nuestro contenedor y qué software necesitará para ejecutar la tarea que queremos para él.

Imagen: Para que no sea necesaria la ejecución del los comandos incluidos en el dockerfile cada vez que queramos "instanciar" un contenedor, a través de los dockerfiles se generan las llamadas "imágenes", que son instancias de servidor ya preparadas para la ejecución según lo definido.

Registro: Colección de imágenes que podemos utilizar

Contenedor: Llamamos contenedor a la ejecución de una instancia de una imagen

Habría que hacer una imagen como esta, pero añadiendo antes de Image que se crean a partir de un dockerfile, no he encontrado ninguna:



—9.3.2 Clústers (Kubernetes)

Como hemos visto en el punto anterior, los contenedores nos pueden servir para el despliegue de aplicaciones simples de manera autocontenida en entornos fácilmente replicables. Esto es suficiente en algunos casos, pero en grandes despliegues, queda ampliamente escaso. De la misma manera que no pensamos en un servicio medianamente complejo, desplegado sólo en un servidor o máquina virtula, tampoco ocurre así con los contenedores.

Para gestionar infraestructuras más complejas, se utilizan clústers de contenedores, siendo la tecnología más adoptada para ello Kubernetes.

Kubernetes, creada por Google y cedida a la comunidad, convirtiéndose en un estándar de facto a la hora de administrar y orquestar contenedores de manera eficiente y escalable.

Cuando se habla de eficiencia, Kubernetes ayuda a la asignación de recursos necesaria a los diferentes contenedores que gestiona de una manera dinámica y eficaz. Al hablar de recursos, se incluye todo el entorno de ejecución de las cargas, incluyendo, por ejemplo, su capa de red y almacenamiento.

Más interesante aún es su capacidad de escalado, entendiendo por esta, la capacidad del propio clúster para, por ejemplo, auto-replicar contenedores que se utilizan para servicios con picos de demanda para poder absorber ese incremento de tráfico, para después, volver a "apagar" esa infraestructura cuando no sea necesaria.

Por ejemplo, una empresa de retail tiene una oferta puntual que genera mucho tráfico en su servicio de ecommerce, en una infraestructura tradicional, la escalabilidad para absorber ese "pico" solía ser tradicionalmente compleja de conseguir. En cambio, en entornos basados en clústers de Kubernetes, por defecto pueden definirse políticas que gestionen este tipo de casuísticas y se puedan absorber estas contingencias sin problema.

Cuando se hable de Kubernetes, será interesante conocer una serie de conceptos básicos:

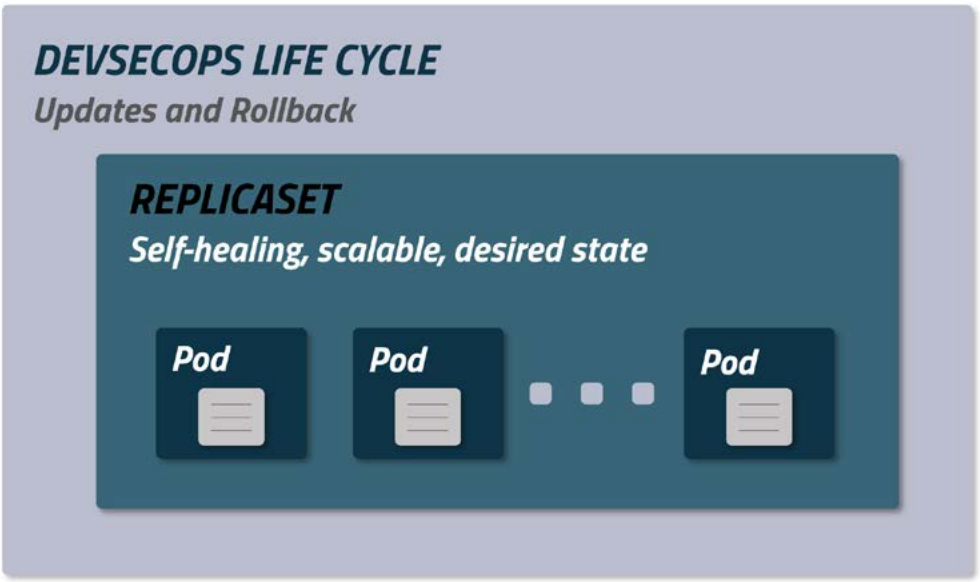
1. **Pod:** Es la unidad más básica de Kubernetes. Es una abstracción que representa uno o varios contenedores y sus recursos asociados que comparten entorno de ejecución. De manera conceptual, representaría el entorno mínimo de una aplicación o servicio que se ejecuta en el clúster.

2. **ReplicaSet:** Es uno de los controladores que se pueden configurar en el clúster y donde se define cuántas instancias (copias) de cada Pod se ejecutan y cómo actualizarlas.

3. **Deployment:** Otro de los controladores que se encarga de administrar la implementación de las aplicaciones. Un deployment especifica el número de réplicas que se deben crear y cómo actualizarlas

4. **Servicio:** Es una abstracción que se define como un conjunto de pods y una política de acceso a ellos desde fuera del clúster. Los servicios cuentan con direccionamiento de red y definen la manera de acceder a los pods desplegados en el clúster. Independientemente de réplicas y deployments. Que todo lo que ocurre dentro del clúster sea "invisible" desde fuera y sólo se consuman servicios es lo que otorga a un despliegue de Kubernetes esa flexibilidad en cuanto a escalabilidad, ya que para un cliente es transparente.

Habría que meter un gráfico como este, pero en pod poner varios contenedores y por encima de deployment el servicio:



—9.3.3 Microservicios

Si hay una estrategia desde el punto de vista de arquitectura sofware que encaja a la perfección con entornos de contenedores y metodologías DevSecOps, esta es la basada en microservicios.

Este paradigma de arquitectura software se enfoca en la construcción de aplicaciones, no como un monolito único que contiene toda la funcionalidad, sino como un conjunto de servicios pequeños que trabajan juntos para cumplir una función completa. Cada servicio se enfoca en una tarea específica dentro de la aplicación, y se comunica con otros servicios mediante una interfaz bien definida.

Se dice que encaja perfectamente con entornos basados en contenedores, porque cada microservicio se ejecuta idealmente en su propio proceso y puede ser escalado de forma independiente según sea necesario. Además, los microservicios son independientes en términos de lenguaje de programación, herramientas y tecnologías utilizadas para construirlos, lo que significa que se pueden incluso utilizar diferentes lenguajes de programación y tecnologías para cada microservicio.

Hablando de escalabilidad, en el ejemplo expuesto sobre el retailer que tiene un pico de demanda en el punto anterior, podría escalar todo lo relacionado con la parte de gestión de sesiones y productos cuando se experimenta ese incremento, sin necesidad de, por ejemplo, escalar servicios relacionados con backoffice o administración). Además de las ventajas claras en cuanto a escalabilidad, un enfoque basado en microservicios proporciona:

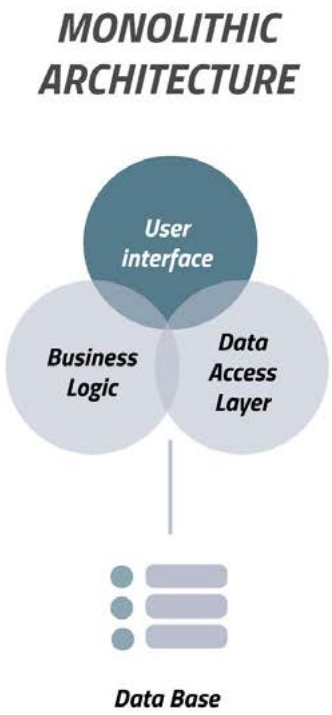
1. Flexibilidad tecnológica: cada microservicio puede utilizar diferentes tecnologías, lo que permite a los desarrolladores utilizar la mejor herramienta para cada tarea. Si, por ejemplo, el equipo piensa que Python para realizar tareas de machine learning es más sólido que Java, puede realizarse en ese lenguaje lo relacionado con esa rama y mantener en Java servicios más básicos como, por ejemplo, autenticación y gestión de sesiones.

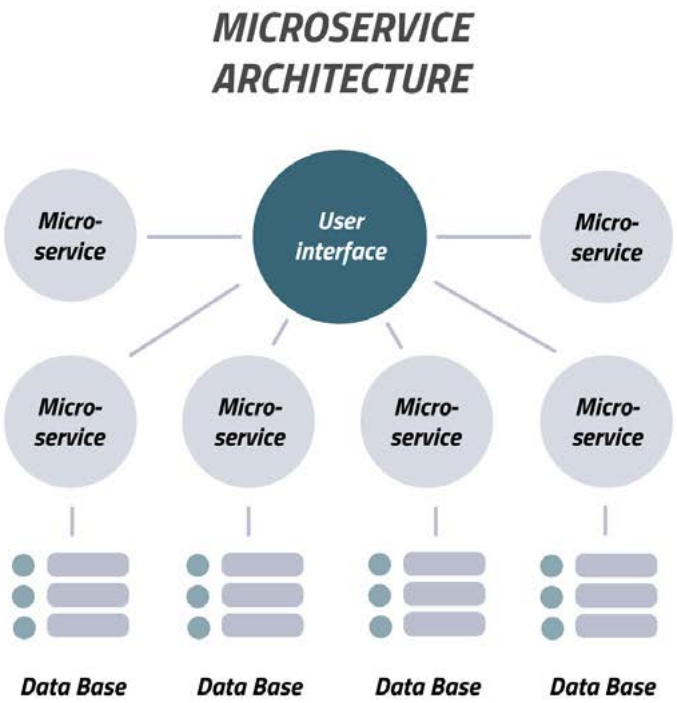
2. Facilidad de mantenimiento: como los microservicios son independientes, es más fácil mantenerlos y actualizarlos sin afectar a otros servicios.

3. Resiliencia: si un servicio falla, otros servicios en la aplicación pueden seguir funcionando, lo que mejora la resiliencia de la aplicación.

4. Mejor escalabilidad de equipo: los microservicios permiten a los equipos de desarrollo trabajar de forma más independiente, lo que acelera el proceso de desarrollo y permite una mejor colaboración.

En resumen, enlazando los conceptos de "Pod", que se ha visto en el punto anterior sobre Kubernetes, como ese conjunto mínimo de microservicios para una funcionalidad y entendiendo la unidad de abstracción superior "Service" como ese conjunto de funcionalidades que conforman un servicio o aplicación, se entenderá la versatilidad y potencia en cuanto a escalado y flexibilidad que puede aportar un entorno basado en clústers y diseñado siguiendo una metodología de arquitectura basada en microservicios a los desarrollos.





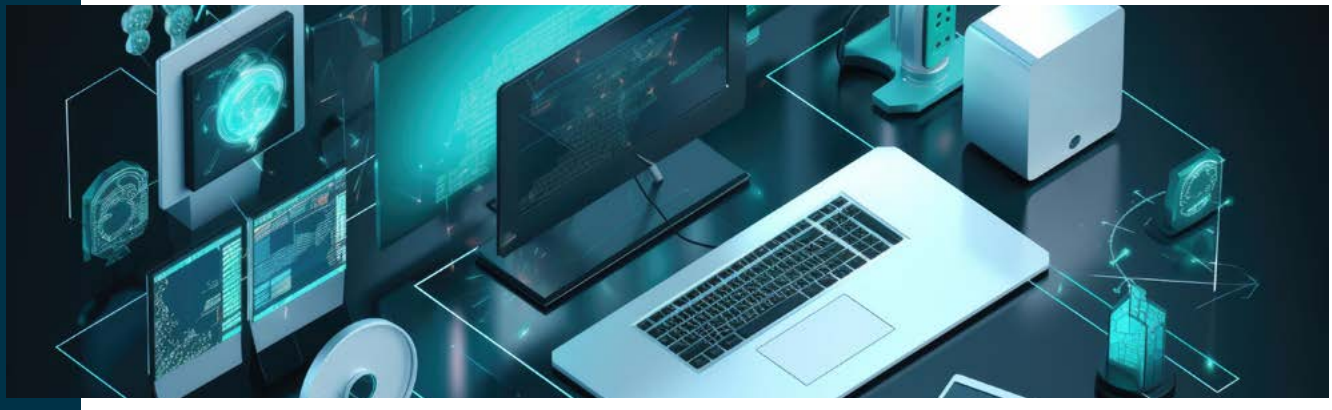
9.4. Seguridad en entornos dockerizados:

Como se ha comprobado, los contenedores son una herramienta eficiente de empaquetar, distribuir y ejecutar aplicaciones, pero también, claro, representan desafíos desde el punto de vista de ciberseguridad.

Los contenedores son entornos aislados que se ejecutan en un sistema operativo host compartido, lo que significa que las vulnerabilidades en un contenedor pueden afectar a otros contenedores del mismo sistema. Además, los contenedores son altamente portátiles y pueden ser implementados en diferentes plataformas y entornos, lo que puede aumentar el riesgo de ataques maliciosos

Según OWASP, en su trabajo OWASP Kubernetes Top 10, los principales problemas, desde el punto de vista de seguridad, que se pueden encontrar en entornos basados en clústers de contenedores, son los siguientes:

- K01 - Configuraciones de workloads inseguras: Configuraciones de pods y servicios en Kubernetes puede permitir la exposición de datos confidenciales o vulnerabilidades de seguridad en la infraestructura de la aplicación. Esto es especialmente común en entornos cloud cuando no se hace una buena gestión de API-Key y secretos
- K02 - Vulnerabilidades en la cadena de suministro: Vulnerabilidades de seguridad que pueden surgir de los componentes y paquetes de software utilizados por una aplicación de Kubernetes que provienen de proveedores externos, como, por ejemplo, paquetes software de librerías o imágenes base de Linux que pueden tener vulnerabilidades o estar obsoletas.
- K03: Configuraciones RBAC excesivamente permisivas: Configuraciones de roles y permisos que pueden permitir a los usuarios acceder a recursos y funciones que no deberían tener acceso.
- K04: Falta de aplicación centralizada de políticas: Fundamentalmente por la falta de una política de seguridad centralizada y consistente que se aplique a través de todos los componentes y cargas de trabajo de Kubernetes.



- K05: Registros y monitoreo inadecuados: La falta de monitorización adecuada para identificar y responder a las amenazas de seguridad en la infraestructura de Kubernetes.
- K06: Mecanismos de autenticación defectuosos: Problemas de autenticación que pueden permitir a los usuarios no autorizados acceder a la infraestructura de Kubernetes.
- K07: Ausencia de controles de segmentación de red: Una segmentación adecuada de la red que pueda permitir a los usuarios no autorizados acceder a recursos y funciones que no deberían tener acceso.
- K08: Fallos en la gestión de secretos: La falta de seguridad adecuada en la gestión de los secretos de Kubernetes, lo que puede exponer información confidencial y privilegios de acceso.
- K09: Componentes de clúster mal configurados: La configuración incorrecta o defectuosa de los componentes de clúster, como puedan ser ReplicaSet o deployments que pueden comprometer la seguridad de la infraestructura de Kubernetes.
- K10: Componentes de Kubernetes desactualizados y vulnerables: Falta de actualizaciones de seguridad y parches en los componentes de Kubernetes, lo que puede exponer la infraestructura a vulnerabilidades conocidas.

Se puede observar por lo tanto que, para poder cubrir estos aspectos sobre la seguridad en entornos basados en contenedores, será necesario realizar de manera continua y automatizada análisis tanto de manera estática en la creación y configuración de las cargas, como de manera dinámica de monitorización de qué está ocurriendo en cada momento en los entornos de ejecución.

10 HOJA DE RUTA PARA ADOPTAR DEVSECOPS

Mientras que la industria del software celebra más de una década de DevOps, hay un creciente impulso hacia la adopción de DevSecOps y hacer de la seguridad una parte del software desde el principio. Crear software seguro y, al mismo tiempo, cumplir los requisitos de velocidad y escala del mercado es una paradoja para las organizaciones de IT modernas. Las organizaciones suelen enfrentarse a una serie de retos comunes durante la adopción de DevSecOps.

Este capítulo trata de resumir las principales recomendaciones y estrategias con el objetivo de ayudar a las organizaciones en la implantación de DevSecOps.

10.1. Niveles de madurez

El concepto de madurez de DevSecOps es una forma útil de explicar las recomendaciones y el valor de cada uno de los pasos durante la adopción. Seguir un modelo de madurez también ayuda a contar una historia que incluya a las personas, los procesos y los cambios tecnológicos que conlleva la transformación a DevSecOps.

DevSecOps es un enfoque para el desarrollo y la operación de software que se centra en integrar la seguridad en todas las etapas del ciclo de vida del software. Esto significa que, en lugar de tratar la seguridad como un problema separado que se aborda al final del proceso de desarrollo, se incluye desde el principio y se mantiene a lo largo de todo el proceso.

Lo primero a tener en cuenta es que DevSecOps no es un destino, sino que se ve como un viaje cuyo éxito se base en los pilares fundamentales.

Con una base sólida en DevSecOps, las organizaciones pueden mejorar gradualmente su nivel de madurez en el uso de esta práctica. Esto implica evolucionar y ampliar sus capacidades para lograr sus objetivos empresariales y aumentar la eficiencia, la calidad y la seguridad. Con el tiempo, los equipos de desarrollo pueden integrar y configurar conjuntos de herramientas automatizadas que les ayudarán a adoptar un enfoque proactivo y predictivo en materia de seguridad, proporcionando información valiosa y comentarios en tiempo real para informar futuros esfuerzos de madurez.

A medida que la práctica DevSecOps madure, las organizaciones pueden esperar aumentar constantemente la velocidad, frecuencia y calidad de sus despliegues de software, al mismo tiempo que fomentan una cultura de equipo centrada en la innovación y la colaboración. Esto las preparará para abordar los desaíos actuales y futuros en un entorno de TI en constante evolución.

Los siguientes niveles de clasificación ofrecen una guía en este viaje de adopción de DevSecOps:

- Nivel 1 – Base: se caracteriza por la ejecución manual de las herramientas de seguridad durante el despliegue y las pruebas del software. No existen métricas claras de seguridad. Se considera la seguridad como un silo. La gestión de equipos, los procesos y la seguridad de las aplicaciones siguen estando a un nivel muy ad hoc.
- Nivel 2 – Principiante: Existen algunas automatizaciones básicas de seguridad durante la construcción de imágenes de software y las pruebas correspondientes que se realizan de forma estandarizada. Se lleva a cabo la revisión del modelo de seguridad regularmente. En este nivel se establecen los pilares fundamentales.
- Nivel 3 – Intermedio: Se ha avanzado mucho en la automatización de la seguridad utilizando herramientas maduras. Esto permite un aprovisionamiento seguro, escalable y coherente. Con el apoyo de toda la organización, toma forma una estrategia de seguridad para el despliegue continuo de aplicaciones.
- Nivel 4 – Avanzado: gracias a la cultura de equipo, la organización sigue en busca de mejoras adoptando tecnologías emergentes. Se desarrolla una monitorización y visibilidad eficiente durante el proceso de desarrollo de software.
- Nivel 5 – Experto: Se dispone de una cultura de colaboración y apoyo entre los distinto equipos que permite realizar despliegues continuos de forma eficiente. Visibilidad en tiempo real de la visión agregada de la seguridad.

Niveles de madurez DevSecOps					
	Nivel 1 Base	**Nivel 2 Principiante**	**Nivel 3 Intermedio**	**Nivel 4 Avanzado**	**Nivel 5 Experto**
Cultura y Organización	El SME de seguridad forma parte de los miembros del equipo central.	El SME de seguridad cuenta con experiencia para aplicar y fomentar las capacidades de automatización de la seguridad. Ayuda a proteger las cargas de trabajo, reduzce su tiempo de comercialización y aumenta sus beneficios.	Los requisitos de cumplimiento de seguridad (CSDL unificada) se rastrean en una herramienta centralizada de gestión de tareas para cumplir los Criterios de Preparación de Seguridad (SRC) de la CSDL.	Las partes interesadas están siempre disponibles para proporcionar comentarios y aclaraciones al SME de seguridad cuando sea necesario para lograr los requisitos de cumplimiento de seguridad (CSDL unificado).	Se dispone de apoyo de liderazgo para desbloquear los equipos interfuncionales en busca de ayuda para la automatización de la seguridad y el cumplimiento (personas, procesos y herramientas).
Diseño y Arquitectura	No se comprenden las necesidades de requisitos de seguridad (requisitos CSDL unificados).	El SME/Arquitecto de Seguridad revisa regularmente la arquitectura y el modelo de amenazas con el equipo. El arquitecto se asegura de que el equipo comprende el modelo de amenazas, las necesidades de seguridad, los requisitos y las limitaciones antes de explorar soluciones técnicas.	El SME / Arquitecto de Seguridad se centra en los requisitos de seguridad de comprobabilidad automatizada. El SME / Arquitecto de Seguridad invita al equipo a ayudar a probar y seleccionar alternativas técnicas que satisfagan las necesidades generales de conformidad de seguridad y arquitectura.	El equipo tiene una clara comprensión del modelo de amenazas y de la arquitectura. El equipo colabora regularmente con el Arquitecto de Sistemas para proporcionar comentarios y recomendaciones para introducir mejoras en la solución de automatización de la seguridad y en la arquitectura.	El SME / Arquitecto de Seguridad no sólo diseña la solución de automatización de la seguridad, sino que también diseña la visibilidad de los resultados / métricas de seguridad.
Construcción	Ejecución o configuración manual de las herramientas de seguridad	Habilitación de una tarea de automatización de la seguridad independiente de la herramienta CI para proteger las cargas de trabajo.	El repositorio de código está integrado con una herramienta de análisis estático de código para validar las vulnerabilidades de seguridad.	El proceso "Software Quality Gates" es usado: la construcción fracasa si no se cumple la calidad (vulnerabilidades de seguridad)	Despliegues continuos «sin intervención» basados en parámetros de seguridad del producto
Pruebas y verificaciones	Pruebas de seguridad manuales	Comprobación automatizada de terceros, escaneado de contenedores, análisis estático de código, comprobación de harden, escaneado de aplicaciones web, cargas de trabajo seguras en entornos de nube pública.	Herramienta automatizada de postura de seguridad en la nube habilitada para medir la postura de seguridad del inquilino de la nube frente a los requisitos de seguridad.	Vulnerabilidades de seguridad detectadas y subsanadas de inmediato	Análisis de seguridad automatizado > %80 de los requisitos de seguridad (CSDL unificado)
Información y reporte	Métricas de seguridad del producto incoherentes y parciales	La notificación del estado de la automatización de la seguridad se envía al equipo del proyecto utilizando herramientas colaborativas.	Existe un proceso de visibilidad de la automatización de la seguridad para recopilar, analizar y publicar los resultados/pruebas de seguridad de los productos de autovalidación en una ubicación central para la CSDL unificada.	Se proporciona automáticamente información sobre el cumplimiento de la seguridad de la CSDL.	Visibilidad en tiempo real de la visión agregada de la seguridad del producto y la información de cumplimiento CSDL para proyectos, soluciones y servicios en desarrollo.

Las organizaciones deberían aspirar a alcanzar un nivel de madurez superior, por ejemplo, el nivel 4 o 5, ya que estos conllevarán más beneficios para su organización y se alinearán con las tendencias actuales del sector.

10. 2. Desafíos

Los cambios necesarios para llevar a cabo la adopción de DevSecOps puede estresar a los programas de seguridad existentes. La adopción de nuevas tecnologías y procesos impulsada por DevOps puede dejar la seguridad en un segundo plano o, en algunos casos, dejar al descubierto lagunas en la seguridad y la gestión de riesgos.

Algunos obstáculos comunes en la adopción de DevSecOps son:

- Falta de métricas: La transformación de DevOps y DevSecOps debe medirse en función de objetivos establecidos. Las métricas pueden incluir el rendimiento de la entrega de software o el rendimiento general de la organización a lo largo del tiempo.
- Alcance: cuando las organizaciones empiezan a integrar actividades y escáneres de seguridad en DevSecOps, se tiende a desarrollar un alcance excesivo de conjuntos de reglas y configuraciones de escáneres. En muchos casos, esta estrategia genera un número de vulnerabilidades de seguridad que hace imposible gestionarlas de manera eficiente durante la transición. Esto a su vez provoca reticencia a la hora de solucionar los problemas, y falta de apoyo y aceptación por parte de los equipos de desarrollo, que puede llegar a ralentizar los esfuerzos evolucionar la cultura DevOps.
- Falta de conocimientos: uno de los principales retos es la falta de conocimientos y de herramientas o procesos que ayuden a incorporar la seguridad en el despliegue de software. Permitir que los equipos logren este objetivo es vital para garantizar que puedan crear software seguro.
- Cultura: DevSecOps se basa en una cultura y una mentalidad en la que varios equipos multifuncionales comparten el objetivo común de crear software y productos seguros desde el inicio. Sin embargo, existe una reticencia natural al cambio, en especial por lo equipos de seguridad cuya misión principal es minimizar los riesgos. Adoptar nuevas formas de trabajo y procesos puede suponer un reto para estos equipos.

10. 3. ¿Cómo empezar?

La implantación de un marco DevSecOps puede beneficiar enormemente a las organizaciones al integrar las consideraciones de seguridad en el proceso de desarrollo de software desde el principio. Esto permite identificar y abordar de forma proactiva las posibles vulnerabilidades de seguridad, en lugar de intentar solucionarlas a posteriori. Construir una base sólida en DevSecOps es crucial para mejorar gradualmente las prácticas y maximizar el valor. Sin embargo, es importante reconocer que se trata de un viaje que requiere tiempo, esfuerzo y la voluntad de experimentar con diferentes enfoques.

Uno de los componentes clave de este viaje es educar a los miembros del equipo en prácticas de seguridad que pueden no haber formado parte de sus funciones anteriores. En un proceso de desarrollo tradicional, la seguridad es a menudo una ocurrencia tardía, con responsabilidades divididas entre diferentes equipos o departamentos. Con DevSecOps, la seguridad se integra en cada paso del proceso, y es importante que todos los miembros del equipo comprendan su papel en el mantenimiento de la seguridad. Esto puede requerir formación y otros recursos para garantizar que todos conozcan las mejores prácticas y cómo aplicarlas.

Otro aspecto a tener en cuenta es el hecho de que la curva de madurez de DevSecOps no sigue una hoja de ruta estricta y única. Cada organización tiene necesidades y retos únicos, y puede ser necesario probar y equivocarse para encontrar la combinación adecuada de herramientas, procesos y estrategias de comunicación que mejor funcionen. Esto es especialmente cierto cuando se trata de equilibrar la necesidad de seguridad con la necesidad de velocidad y agilidad en el proceso de desarrollo.

Una vez en marcha, se recomienda empezar por un caso de uso de DevSecOps. Este caso de uso debe de ser lo suficientemente pequeño, pero con un alto potencial de éxito. Esto permite al equipo aprender, fracasar y tener éxito sin afectar a la actividad principal de la organización.

Es clave, por tanto, empezar poco a poco. Las pruebas de seguridad deben comenzar lo más a la izquierda posible en el SDLC y deben realizarse con un alcance gradualmente creciente. Por ejemplo, en lugar de habilitar escaneos completos o escaneos con todo el conjunto de reglas para un punto de contacto de seguridad previo al compromiso, se debe considerar mantener el conjunto de reglas limitado a las vulnerabilidades principales. Las actividades de seguridad que tienen lugar más tarde en el SDLC pueden incluir exploraciones y revisiones más profundas para garantizar la seguridad antes del despliegue.

10. 4. Medir el éxito de la implantación

Medir los resultados de la práctica DevSecOps recién establecida puede ser un reto. Dependiendo de los objetivos de negocio específicos, las organizaciones utilizan una variedad de métricas para medir e identificar las fortalezas y debilidades de su práctica DevSecOps a través de cada uno de los pilares y ayudar a determinar los próximos pasos en el viaje DevSecOps. Estas métricas también proporcionan un mecanismo que puede presentar el resultado de invertir en la adopción de DevSecOps.

Afortunadamente, muchas de estas métricas pueden aprovecharse directamente de los conjuntos de herramientas iniciales. Una vez implementadas, las herramientas DevSecOps pueden generar cantidades masivas de datos significativos (por ejemplo, métricas de calidad y seguridad, frecuencia de despliegue, rendimiento de las aplicaciones e informes de cumplimiento) para demostrar la alineación con los objetivos empresariales y ayudar a impulsar mejoras operativas.

Para maximizar este beneficio, estas herramientas deben adaptarse continuamente, integrarse entre sí y gestionarse como parte de la práctica DevSecOps. Una cadena de herramientas altamente integrada permite a la organización recopilar datos significativos que informan las mejoras operativas para impulsar la madurez de DevSecOps y maximizar el valor de sus inversiones.

La supervisión de las métricas y la respuesta a las mismas proporcionan una buena perspectiva de cómo se está actuando en relación con la estrategia DevSecOps. Ejemplos de métricas de alto valor de seguridad relevantes incluyen las siguientes:

- Media (tiempo) para detectar incidentes de seguridad
- Media (tiempo) para investigar incidentes de seguridad
- Media (tiempo) para contener incidentes de seguridad
- Nivel de cumplimiento de seguridad (en %)
- (Número) de incidentes de seguridad relacionados con controles de seguridad DevSecOps mal configurados
- (Número) de formaciones de seguridad a las que han asistido miembros del equipo DevSecOps
- Integración de los sistemas DevSecOps con el SOC de la organización (en %) para la recopilación de eventos, (Número) de casos de uso de monitoreo de seguridad desplegados en relación con las amenazas de seguridad que se aplican a el pipeline CI/CD
- (Número) de vulnerabilidades reportadas en las fases de desarrollo, liberación y tiempo de ejecución
- (Número) de vulnerabilidades recurrentes durante las fases de desarrollo, liberación o tiempo de ejecución
- (Número) de despliegues de Pods que eluden el pipeline CI

Un programa de métricas completo es aquel que abarca de manera integral los aspectos de personas, procesos y tecnología, y proporciona información sobre el rendimiento y el éxito. Por ejemplo, las métricas deben mostrar problemas que surgen cuando las personas no siguen procesos adecuados, así como problemas que surgen cuando las herramientas no se utilizan de manera efectiva debido a la falta de procesos adecuados. Para lograr esto, es esencial medir y recopilar los datos relevantes en cada etapa del proceso de desarrollo y seguridad.

Estos datos permiten a los equipos medir y optimizar diferentes aspectos de su proceso de desarrollo de software para mejorar la calidad y la eficiencia de este. Los datos se utilizan para ayudar a los equipos a establecer metas y objetivos concretos para mejorar el rendimiento y la calidad del software.

11 REFERENCIAS

- WILSON, G. (2020). DevSecOps. Great Britain: Rethink Press.
- KIM, G., BEHR K. y SPAFFORD, G. (2013). The Phoenix Project. Portland: IT Revolution.

DevSecOps

Guía de iniciación en la Seguridad aplicada al DevOps

www.ismsforum.es
info@ismsforum.es
(+34) 915 63 50 62

@ISMSForum

ISMS Forum

Una iniciativa de

ismsforum