VNiVERSiDAD D SALAMANCA

CAMPUS DE EXCELENCIA INTERNACIONAL

MÁSTER UNIVERSITARIO
INGENIERÍA INFORMÁTICA

Trabajo de Fin de Máster

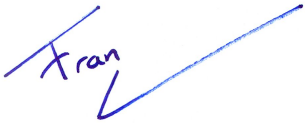
DevOps Factory

Alumno: ***Alejandro Paredero de Dios***

Tutor: Francisco José García Peñalvo

D. Francisco José García Peñalvo certifica que este documento titulado “DevOps Factory” y el código fuente han sido realizados por D. Alejandro Paredero de Dios, con DNI 70885537C, y que estos constituyen la memoria del trabajo completado para superar la asignatura Trabajo de Fin de Máster del Máster Universitario en Ingeniería Informática de la Universidad de Salamanca

En Salamanca, a 1 de Septiembre de 2021



Firmado digitalmente por GARCIA PEÑALVO FRANCISCO JOSE - 07862615L
Fecha: 2021.09.03 09:35:19 +02'00'

Tabla de contenidos

1. Introducción

En la actualidad, el software está en todas partes. En nuestro día a día dependemos del software en todos los aspectos cotidianos: en nuestro trabajo, en casa, en los deportes, en las instituciones educativas, mientras viajamos, y cómo no, cuando nos comunicamos. Gracias al software, estamos más conectados, somos más efectivos, más creativos y más eficientes.

En términos laborales, el software crea millones de empleos directos e indirectos en una variedad amplia de profesionales en los centros de trabajo. Desde desarrolladores de software, a aspectos muy específicos - cómo la seguridad, calidad, arquitectura, oficina de proyectos o administrativos entre otros - , es también un motor económico para otros sectores ajenos al mismo, como el de servicios que han percibido un gran aumento en su facturación debido tanto a los productos de software producidos como la satisfacción de demanda de dichos servicios en los entornos donde se ubican las empresas de software.

Respecto a los sueldos en la industria del software, la media de los sueldos de la industria del software en la UE es un 34% más alta que la media de los sueldos en la UE, y hasta un 80% superior que la media de los sueldos de los trabajadores del sector servicios. El sueldo medio supera la barrera de los 45.000€ brutos anuales, y se estima que en la UE se paga un total superior a 162 mil millones de euros en salarios en la UE.

A diferencia de los productos físicos, el desarrollo de software tiene como objetivo producir un artefacto digital con objetivo de prestar un servicio o utilidad a un usuario o colectivo final. Sin embargo, la producción de estos elementos requieren de una organización y orquestación de trabajo a un nivel máximo de detalle, similar a la producción de un elemento tangible. Es por ello, que el desarrollo de software va fuertemente ligado a metodologías de trabajo y desarrollo.

En la década de los 80 se publicó una forma de desarrollo ámpliamente aceptada por la industria de software denominado “Desarrollo en cascada”, en la cual el software de desarrolla siguiendo una serie de pasos o etapas claramente delimitados, donde hasta que no finalice una etapa no se puede comenzar la siguiente. Estos pasos son: Requisitos, diseño, implementación, verificación y mantenimiento.

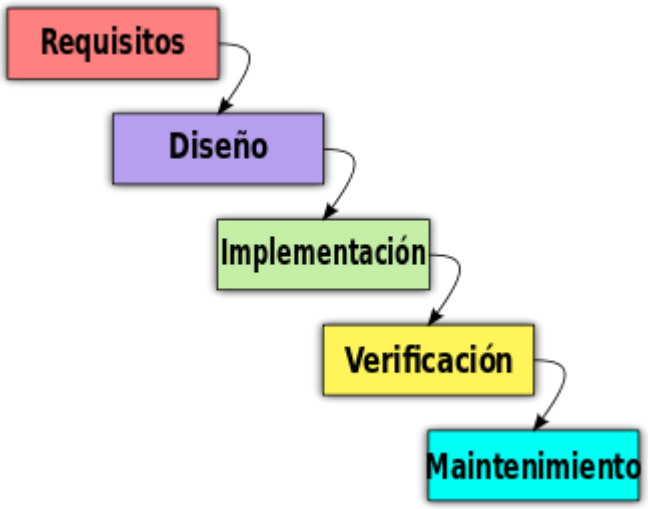


Figura 1. Desarrollo en cascada

En la actualidad hay empresas que mantienen esta metodología de desarrollo, especialmente en productos “legacy” implementados años atrás, donde un cambio de forma de trabajo no es

posible. El uso de esta metodología en proyectos nuevos mantiene una tendencia menor respecto a otras metodologías que a continuación explicaremos, ya que esta presenta una serie de inconvenientes, como que estos proyectos toman mucho tiempo para ser completados, son muy rígidos ante los cambios, y según aumenta de tamaño el proyecto, más difícil es de gestionar todos los aspectos que lo componen.

Por otra parte, en las últimas dos décadas han aparecido nuevas metodologías de desarrollo, enfocadas construir un producto de software no cómo un producto completo, si no como un proceso iterativo, capaz de ofrecer un servicio mínimo que irá evolucionando con el tiempo. Son las conocidas metodologías ágiles, o metodologías de desarrollo ágil.

Las metodologías ágiles ofrecen numerosas ventajas, como una mejora constante de la calidad del producto, al realizar entregas pequeñas de software; mayor satisfacción del cliente, puesto que el ciclo de vida es más flexible y adaptables a cambios; fomenta un trabajo colaborativo de los propios equipos de la empresa de software; permite extraer feedback del producto puesto en producción y adoptar nuevas decisiones en futuros desarrollos, y permite una reducción de costes, tanto en refactorización o desecho de trabajo, como hasta el abandono completo del proyecto, un hecho bastante común en los proyectos de gran tamaño seguidos con una metodología en cascada, por su complejidad y retrasos en el tiempo.

De las diversas metodologías ágiles existentes, la más adoptada por las empresas actualmente es la metodología Scrum. Scrum está basado en la organización de trabajo de un proyecto de desarrollo de software en ciclos temporales cortos y de duración fija

(normalmente dos semanas, aunque también se encuentran tres o cuatro semanas). En cada iteración se tiene que entregar un resultado completo, que se entiende como un incremento del producto final, aportándole nuevas funcionalidades o mejoras sobre las ya existentes.

En la metodología Scrum se identifican unos roles en los equipos los cuales establecen la gobernabilidad y la prioridad de desarrollo de las tareas dentro de los ciclos iterativos antes mencionados. Además, la organización y comunicación es un elemento clave, por lo que existen una serie de eventos o reuniones, con carácter diario, que permiten sincronizar y seguir de cerca la evolución del trabajo, permitiendo desbloquear aquellos problemas que el equipo de desarrollo detecta durante su trabajo.

Si bien Scrum habla mucho sobre la organización del trabajo y de los equipos, el software es un elemento habilitador crucial para gestionar de forma efectiva los recursos de los proyectos. En un contexto donde una empresa puede trabajar al mismo tiempo en distintos proyectos, compuesto por varios equipos de desarrollo los cuales pueden estar formados por grupos de personas distintas, es importante mantener una estandarización en la medida de lo posible.

Es aquí donde el rol del administrador de sistemas tiene una evolución en sus funciones y responsabilidades. El perfil del *SysAdmin* es el de satisfacer las necesidades relacionadas con la infraestructura y las aplicaciones necesarias para el desarrollo de sus usuarios finales, -en este caso, los equipos de desarrollo -.

Los proyectos de software, a pesar de que utilicen distintas tecnologías de desarrollo (como lenguajes de programación) y tengan distintos cometidos finales, siguen unos procedimientos similares en las fases de construcción y prueba de artefactos, y es aquí donde el rol del *SysAdmin* juega un papel importante en la eficiencia y ahorro de costes, que es la de proporcionar unas herramientas que de forma automática faciliten las operaciones a todos los equipos involucrados (desarrollo, calidad, seguridad, e incluídos ellos mismos: operaciones) para que el producto de software llegue lo antes posible a los entornos de producción. Esta adaptación del personal de operaciones o *SysAdmin* a esa nueva forma de trabajar se ha denominado DevOps, y su alcance no se limitará solo a los *SysAdmin.*

1.1. DevOps

DevOps es la combinación de los términos en inglés *Development* y *Operations*, y designa la unión de personas, procesos y tecnologías para ofrecer valor a los clientes de forma continua. En el caso del personal de operaciones, su objetivo es proporcionar una serie de servicios a los equipos que permitan agilizar sus fases de trabajo desde un contexto automatizado y en ocasiones, programático.

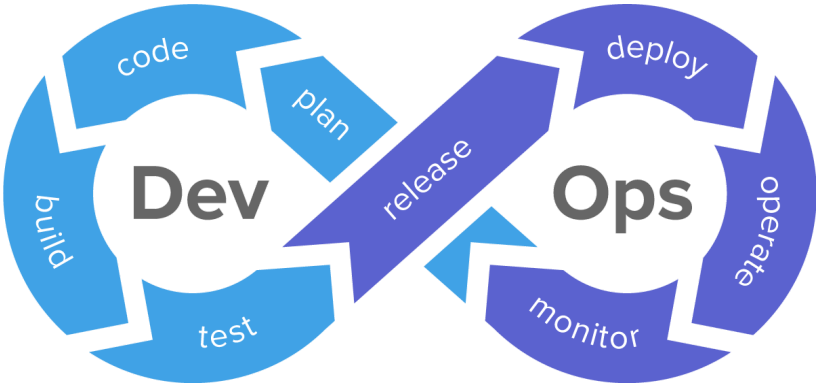


Figura 2. DevOps

Se trata de proporcionar una serie de herramientas que los equipos pueden utilizar para entregar su trabajo, y a través de automatizaciones, realizar las operaciones repetitivas y mundanas, pero requeridas para comprobar y validar su trabajo. En el contexto de desarrollo es por ejemplo, proporcionar un repositorio de código donde puedan realizar sus entregas, y mediante automatizaciones, realizar una primera fase de pruebas sobre el código entregado: Asegurar que compila, validar la documentación técnica, someterlo a una batería de pruebas, realizar comprobaciones de seguridad, etc.

Como mencioné anteriormente, DevOps está basado en cultura, procesos y tecnologías. La cultura hace referencia a que las personas adopten y trabajen de forma colaborativa siguiendo los principios de desarrollo de software acordados, los procesos referencian a la capacidad de automatización de las etapas necesarias para conseguir que el software que se entregue cumpla con las normas de la compañía, y la tecnología menciona las herramientas utilizadas para conseguir el cometido. Si bien las dos primeras acciones dependen principalmente del grupo de personas y del objetivo a construir, el presente proyecto trata de facilitar el tercer y último pilar, el software.

1.2. Factorías de software

La forma de trabajo de una empresa de desarrollo de software conocida como “factorías de software” presenta unas ventajas respecto a una empresa de desarrollo de software de forma

generalista, y es que la especialización en un producto o temática dentro de las factorías de software dan unos resultados generalmente superiores. Por describir algunas características:

- **Productividad y eficiencia:** Al especializarse en una temática en concreto, las acciones y procesos relacionados al desarrollo se pueden estandarizar, simplificar y automatizar.

  Un ejemplo es la reutilización del código de software el cual impacta directamente en la reducción del tiempo necesario del producto final. La automatización reduce el margen de error humano, permite que los recursos se puedan dedicar a otras actividades y aumenta la velocidad de respuesta ante incidencias o cambios en los aspectos iniciales. La suma de estos factores permite que una Software Factory proporcione un servicio formado por profesionales altamente cualificados a precios muy competitivos.

- **Calidad:** La calidad es un elemento clave en el desarrollo de software. Facilita el trabajo de los desarrolladores para comprobar sus piezas construidas, tanto en el sistema en sí como la integración en los terceros.

  La integración de código reutilizable permite focalizar los esfuerzos en características que serán únicas para la parte detallada de cada proyecto. Las aplicaciones desarrolladas utilizando una fábrica de software también se pueden verificar antes de la implementación, asegurando que se sigan las mejores prácticas de fabricación durante su desarrollo. Los tipos de pruebas de calidad que mencionamos son pruebas de funcionamiento, seguridad o arquitectura entre otros.

- **Ahorro:** La principal diferencia entre un equipo de desarrollo de software generalista y una fábrica de software es el precio de los servicios por los servicios prestados. La organización de factorías de software está organizada en centros de desarrollo de aplicaciones de una temática específica, subcontratados o no.

  Industrializar la producción de software abarata los costes y reduce los ciclos de desarrollo de soluciones a medida. Esto, sumado a la combinación de nuevas metodologías de desarrollo, como la mencionada anteriormente Scrum, permite una flexibilidad de producto dentro de unos costes razonables de asumir por el cliente.

Uniendo los dos conceptos anteriores: DevOps y Factoría de software, se fusionan dos objetivos esenciales para una producción eficiente de software: El primero facilita la integración de todos los equipos de trabajo bajo un objetivo común, y el segundo consigue mantener un flujo continuado de mejoras tecnológicas, así como agilizar la relación entre cliente y compañía.

Gracias a lo que llamaremos “devops-factory”, presento a continuación un sistema capaz de construir un entorno con herramientas del ecosistema *DevOps* utilizadas en las distintas etapas de desarrollo de software. Esta construcción se realiza gracias a una factoría de software en la que atendiendo a las distintas opciones nos creará los manifiestos de infraestructura así como de aplicaciones seleccionadas, con la capacidad extra de aplicar dichos manifiestos en los entornos, entregando un entorno completamente funcional.

2. Problema a resolver

El principal reto de las empresas de software es disponer de unos procedimientos estándar a lo largo de toda la empresa .

Cada proyecto IT se confecciona bajo unos requisitos de tecnología apropiados que aseguren su correcto funcionamiento, como un lenguaje de programación, metodología a seguir, etc.

Abstrayéndose de esas especificaciones técnicas, existe en todo proyecto unas etapas de desarrollo que son idénticas en los mismos. De ahí la importancia de crear un

Es por ello que gracias a una factoría podemos crear una plantilla para todas esas necesidades sobre un mismo marco de herramientas, independientemente de las soluciones tecnológicas concretas. En la figura 3, mostramos un pequeño ejemplo del diverso ecosistema de aplicaciones existentes en el mercado.

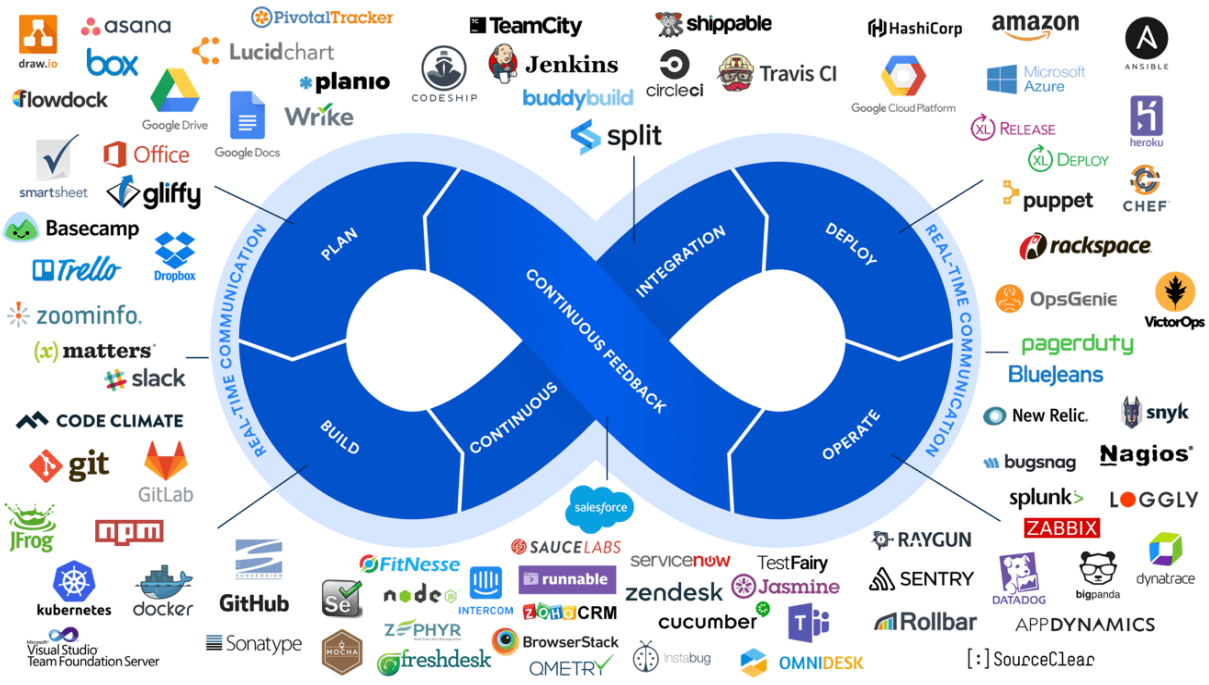


Figura 3. Ecosistema herramientas DevOps.

DevOps es una metodología de trabajo "infinita" que lleva a cabo las etapas de planificación a la puesta en producción de un producto software en una forma lo más automatizada posible. Es muy común encontrar un gráfico explicativo de DevOps como el anterior, en forma de bucle infinito con las etapas en las que está compuesto. A continuación las describimos:

1. **Planificación**: Realizada principalmente por metodologías ágiles, especifica cuál es el objetivo y el trabajo que deben realizar los equipos involucrados en el proyecto.
2. **Código**: desarrollo y primera revisión de código. Se utilizan herramientas de gestión de código fuente
3. **Construcción**: Se utilizan herramientas de integración continua que permiten automatizar la fase de construcción de los artefactos.
4. **Pruebas**: Fase en la que se realizan las pruebas sobre el artefacto construído. Se comienzan con pruebas unitarias, que garantizan el funcionamiento correcto del propio artefacto, y se extiende a pruebas con aplicaciones externas simuladas o ubicadas en los diversos entornos con objeto de detectar posibles fallos en etapas previas a producción.
5. **Release**: Garantizados los resultados de pruebas, se publica el artefacto en los repositorios de la empresa.
6. **Deploy**: Con el artefacto ya disponible, etapa sobre la gestión de cambios, aprobaciones de versiones, automatización de versiones sobre los distintos entornos existentes..
7. **Operate**: Etapa en la que ejecuta la aplicación en el entorno.
8. **Monitor**: Monitorización activa del artefacto, recopilación de información para retroalimentar las siguientes actividades de desarrollo.

3. Objetivos

El objetivo del presente proyecto es proporcionar una pieza de software capaz de proporcionar toda la infraestructura y aplicaciones necesarias para disponer de un entorno de desarrollo de aplicaciones que permita implementar la forma de trabajo DevOps en un entorno educativo o empresarial con una actualmente pequeña selección de opciones.

El despliegue de una infraestructura completa es un término bastante amplio y complejo puesto que hay que tener en consideración el entorno donde queremos tener nuestro ecosistema de aplicaciones. La factoría DevOps busca eliminar esa parte

Es por ello, que la factoría DevOps permite las siguientes opciones.

- Construye el manifiesto de aplicaciones: Basado en aplicaciones en contenedores Docker, se crea un manifiesto para la tecnología docker-compose, el cual orquesta la
- Construye el manifiesto de Infraestructura.

La factoría DevOps está basada en generación de plantillas. Durante su ejecución construye todos los artefactos y manifiestos necesarios por las tecnologías descritas en la sección anterior, depositandolos en la carpeta de trabajo definida en la configuración.

Además, si se desea y se dispone de los requisitos mínimos, la factoría DevOps permite aplicar dichos manifiestos de forma directa, sin necesidad de intervenir de forma adicional

4. Solución planteada

La solución planteada se basa en una arquitectura de aplicaciones ejecutadas dentro de contenedores Docker orquestada mediante docker-compose. En la figura 4 mostramos una referencia ejemplo de la arquitectura construida con la Factoría DevOps

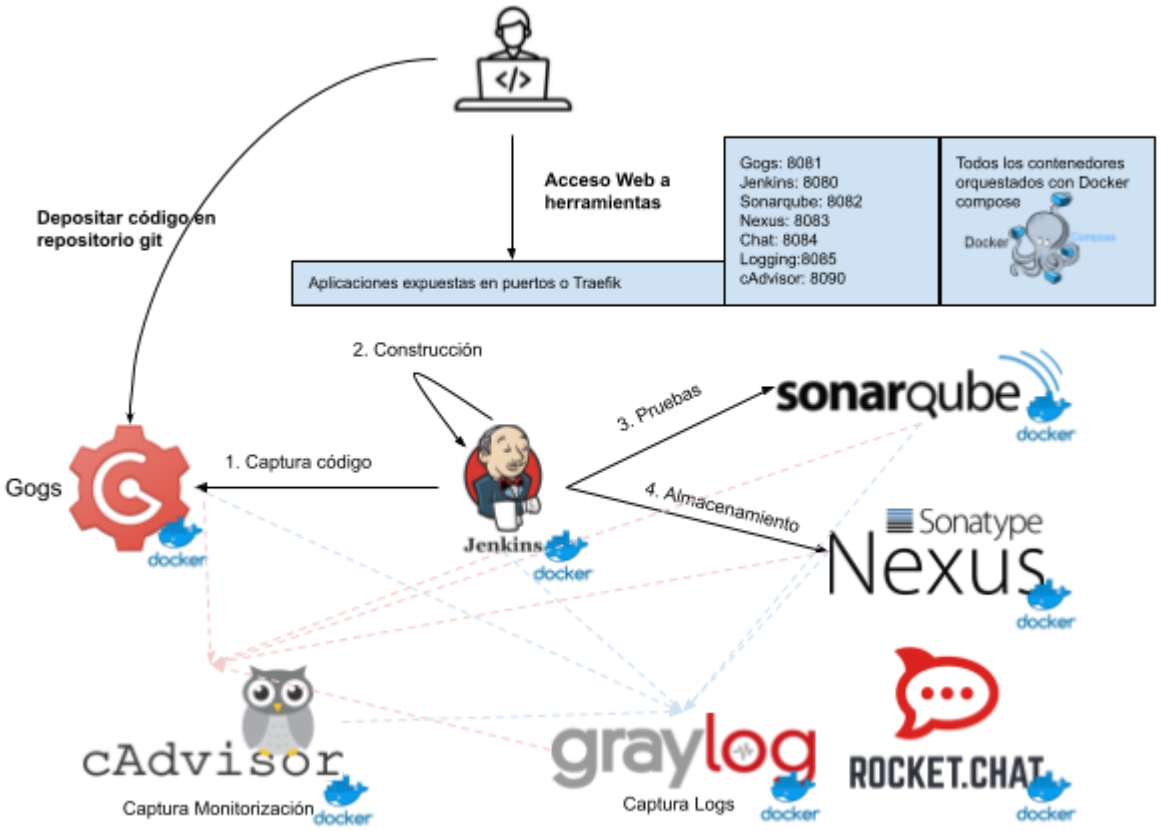


Figura 4. Arquitectura de aplicaciones

Gracias a Docker las aplicaciones son independientes entre sí en relación a dependencia de bibliotecas, binarios y archivos de configuración.

Dentro de las diferentes etapas descritas sobre DevOps, la factoría está compuesta por un ecosistema de aplicaciones agrupadas en distintas categorías. La factura actual permite

desplegar o no una aplicación de cada categoría adaptándose a las necesidades del proyecto que se quiera abordar.

En la imagen anterior se presenta un ejemplo de una arquitectura con todas las aplicaciones disponibles en cada una de sus categorías.

A continuación describimos cada una de las categorías:

- **Servidor Git:** El servidor git es el punto de entrada de cualquier proyecto de software. Todo código fuente necesita de un sistema de gestor de código fuente, y el más utilizado en las últimas décadas es Git, sustituyendo al antiguo SVN por las ventajas que ofrece, como funcionamiento distribuido, gran control en distintos ciclos de desarrollo paralelos denominado ramas o deduplicación de código por deltas entre ramas.

  La factoría DevOps ofrece la posibilidad de utilizar Gitea (https://gitea.io/en-us/) o Gogs (https://gogs.io/)

- **Integración Contínua:** También conocido como sistema de Integración contínua, es el corazón de la arquitectura. Esta aplicación es el orquestador dentro de todo el proceso automatizado, desde la captura del código fuente del repositorio Git y ejecuta los pasos descritos en el proceso de construcción, pruebas y publicación de artefactos. Estos sistemas tienen una fuerte carga de configuración y habitualmente se apoyan en tecnologías como XML o YAML para describir mediante una sintaxis concreta un manifiesto de configuración acorde a cada proyecto. Por último, esta gran capacidad de

configuración también permite que este sistema se haga cargo de la parte relativa al despliegue de los artefactos construidos sobre los diferentes entornos.

La factoría DevOps ofrece la posibilidad de utilizar Jenkins (https://www.jenkins.io/) o Drone (fase beta) (https://www.drone.io/)

- **Calidad de software:** Las herramientas de calidad de software con aplicaciones especializadas en la fase de pruebas unitarias de los artefactos construidos en base al código entregado. Las pruebas se pueden realizar tanto a software compilado como lenguajes de programación interpretados (PHP, ASP, JSP). El uso de este tipo de aplicaciones es la realización de pruebas unitarias sobre el entregable con el objetivo de la detección temprana de errores, tanto en funcionamiento (ej, *stack overflow*) asi como la validación de datos lógicos (ej, operación de suma o resta). Cuanto mayor sea el nivel de automatización en estas herramientas, menor será el trabajo manual requerido por los desarrolladores para verificar su trabajo. Estas herramientas proporcionan un cuadro de mandos donde reciben un informe tras la realización de las pruebas: Si el resultado es positivo, la aplicación de integración continua seguirá con la ejecución de las siguientes aplicaciones. En caso negativo no se avanza, dando al desarrollador la información necesaria para subsanar los errores detectados.

  La factoría DevOps ofrece la posibilidad de utilizar Sonarqube (https://www.sonarqube.org/)

- **Repositorio de artefactos:** El repositorio de artefactos es un software especializado en el almacenamiento de los diversos artefactos construidos o entregados desde una

fuente externa. Generalmente, tras la fase de pruebas, si el artefacto se considera bueno, se almacena para su posterior uso y despliegue en los diferentes entornos. Al igual que una biblioteca física, cuyo objetivo es almacenar y categorizar los libros en base a unos metadatos con el objeto de disponibilizarlo a los usuarios, un repositorio de artefactos gestiona unos metadatos dependiendo de la tecnología a almacenar, sirviendo los paquetes por los interfaces nativos de la tecnología correspondiente. Ejemplos de tecnologías populares gestionadas por repositorios de artefactos son: Maven, APT, YUM, Docker, NPM, Python Pip entre otros.

La factoría DevOps ofrece la posibilidad de utilizar Sonatype Nexus (https://www.sonarqube.org/)

- **Chat:** El chat es un software que permite la comunicación entre dos o más individuos. Tradicionalmente el chat se utilizaba como elemento de mensajería entre personas físicas, pero gracias a la adopción generalizada de estos sistemas en los entornos de trabajo, gracias a la implementación de APIs, algunas herramientas y aplicaciones ofrecen la capacidad de enviar mensajes como si de otra persona se tratase. De esta forma, los sistemas son capaces de informar en tiempo real del estado de sus trabajos, avisar de fallos, o incluso recibir instrucciones sin necesidad de tener que operar de forma directa la aplicación en cuestión.

  La factoría DevOps ofrece la posibilidad de utilizar Rocket.chat (https://rocket.chat/)

- **Logging**: Los sistemas de mensajes, o logging, son las herramientas encargadas de procesar los mensajes de salida de las aplicaciones de nuestra infraestructura.

Permite centralizar, buscar y analizar la información que generan las aplicaciones, pudiendo correlacionar eventos, acciones o hechos que sucedan en tiempo de ejecución. El avance de las herramientas actuales de logging permiten configurar acciones automatizadas al detectar determinados mensajes producidos por los sistemas que auditan.

La factoría DevOps ofrece la posibilidad de utilizar Graylog (https://www.graylog.org/)

- **Monitorización:** La monitorización es un elemento fundamental para comprobar el estado de salud de los servicios y aplicaciones. Con la monitorización, junto al *logging* producido por los sistemas podemos averiguar cuál es el estado actual del sistema, detectar anomalías o incluso prevenir que estas ocurran antes de tiempo. Estos sistemas funcionan principalmente de dos formas: Instalando un agente en los elementos que desean monitorizar (principalmente servidores o elementos con sistema operativo), o se dota al propio sistema de monitorización de la capacidad de interrogar a los elementos de una forma activa (por ejemplo, a través de SSH, APIs o un endpoint con un DSL [lenguaje específico de dominio]).

  La factoría DevOps ofrece la posibilidad de utilizar cAdvisor, una herramienta desarrollada por Google para la visualización del consumo de recursos de los contenedores Docker. (https://github.com/google/cadvisor). Debido a la naturaleza del software y de la arquitectura Docker, actualmente esta herramienta no permite la monitorización de nodos o contenedores externos al nodo en el que se está ejecutando

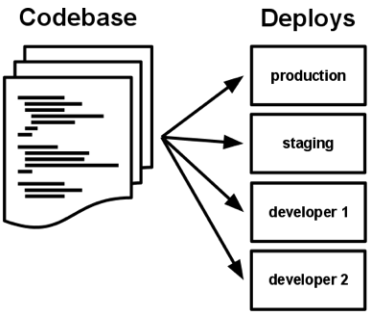
Independientemente de la solución de aplicaciones elegidas para el desarrollo de aplicaciones, la naturaleza de la factoría DevOps es seguir la técnica descrita por *12factor* o *"Twelve-Factor App"* (https://12factor.net/) en la que las aplicaciones actualmente dispuestas ya siguen dicha forma de trabajo.

Twelve-Factor App es un documento consensuado por la comunidad de desarrolladores originando un estándar de desarrollo de aplicaciones de software. Explica, en un contexto avanzado cómo se deben llevar a cabo el desarrollo de aplicaciones de software, abstrayendo la capa de ejecución con la capa de configuraciones y de tratamiento de datos considerado persistencias. El objetivo es crear aplicaciones resilientes, capaces de ser ejecutadas en cualquier entorno, adaptadas a aparecer y desaparecer en los entornos bajo demanda prestando especial atención a que sean escalables, no generen inconsistencia ni incompatibilidad en los datos.

Uno de los principales objetivos de la factoría DevOps es proporcionar las herramientas y técnicas necesarias para seguir esta forma de trabajo. Cabe destacar que las aplicaciones proporcionadas por la factoría siguen esta metodología de trabajo, en la cual nos hemos asegurado que la ejecución de nuestras aplicaciones separa claramente la etapa de ejecución, con la capa de datos que debe administrar, garantizando que podamos volver a conectar los datos de las aplicaciones de la factoría DevOps conectando los ficheros existentes de ejecuciones previas.

Twelve-Factor App está compuesto por 12 premisas a un nivel muy técnico el cual describimos brevemente a continuación.

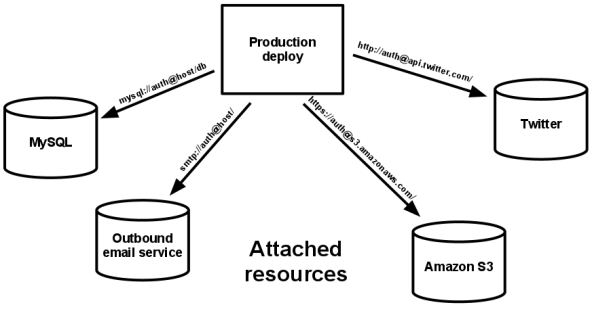
1. **Código base (Codebase):** Almacenar el código fuente de la aplicación en un repositorio de código fuente, permitiendo tener un histórico y trazabilidad de los cambios realizados.

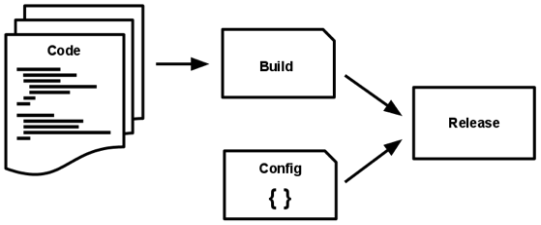


2. **Dependencias:** Declarar y aislar explícitamente las dependencias de la aplicación. Tecnologías como Docker permiten encapsular una aplicación con todas las bibliotecas, configuraciones y requerimientos para una ejecución aislada.

3. **Configuraciones:** Mantener las configuraciones independientes de la ejecución de la aplicación. Similar a las dependencias, permite que las configuraciones sean elementos independientes y "conectables" a la aplicación.

4. **Backing services:** Tratar a los recursos "backing services" (bases de datos, APIs, volúmenes de datos) como recursos conectables.



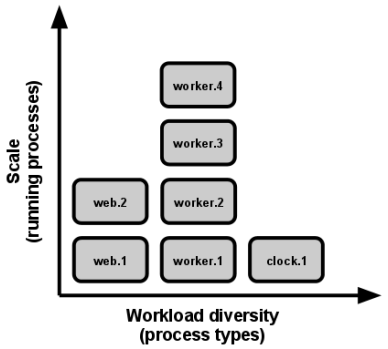


5. **Construir, desplegar, ejecutar:** Separar completamente la etapa de construcción de la etapa de ejecución.

6. **Procesos *stateless*:** Ejecutar la aplicación como un procesos sin estado, sin dependencia de los datos tras una ejecución. Por ejemplo, un motor de base de datos debe ser independiente de los datos que manipula.

7. **Asignación de puertos:** Establecer comunicación con las aplicaciones mediante puertos, evitando en la medida de lo posible técnicas indirectas como sondeo de ficheros.

8. **Concurrencia:** Permitir que la aplicación se pueda ejecutar un número indeterminado de veces sin que su funcionamiento de la misma interfiera entre el número de aplicaciones corriendo.



9. **Desechabilidad:** Permitir que la aplicación pueda ser detenida en cualquier momento sin interferir en la capa de datos. Esto permite hacer el sistema más robusto intentando conseguir inicios rápidos y finalizaciones seguras, asi como garantizar alta disponibilidad.

10. **Paridad en desarrollo y producción:** Mantener desarrollo, preproducción y producción tan parecidos como sea posible. No se debe cambiar el contenido de la aplicación entre entornos, puesto que las pruebas no serían válidas.

11. **Logs:** Tratar los logs de la aplicación como una transmisión de eventos dentro de la aplicación, permitiendo su extracción y análisis.

12. **Administración de procesos:** Si nuestra aplicación necesita tareas de gestión/administración, que puedan ser ejecutados como procesos que solo se ejecutan una vez y cuyos cambios se reflejen en información persistente que pueda ser interpretada por futuras ejecuciones.

4.1. Herramientas

A continuación describimos para la solución planteada las herramientas y tecnologías utilizadas para la creación de la factoría DevOps.

Ansible: Ansible es una herramienta de gestión de la configuración que permite la administración automatizada y remota, centralizando el control del proceso en una máquina. Ansible es una herramienta Open Source con soporte adicional de pago proporcionado por RedHat. Ansible está desarrollado principalmente en Python pero su utilización y la creación de manifiestos es a través del lenguaje de marcas basado en espacios YAML (https://yaml.org) .

Ansible es una herramienta muy potente, capaz de administrar no solo servidores, sino también elementos físicos de hardware, interactuar con múltiples tecnologías y proveedores comerciales, tanto de infraestructura como de aplicaciones. Prácticamente cualquier recurso tecnológico con interfaz de administración (por ejemplo, conexiones SSH, WinRM, APIs) puede ser administrado y gestionado con este software.

Ansible sigue una arquitectura cliente-servidor en la que el nodo que ejecuta ansible se denomina como “nodo maestro” en la que se le proporcionan dos documentos:

- Inventario: Fichero donde se indican los datos de conexión de los nodos a administrar, incluyendo en nuestro caso el propio nodo que ejecuta Ansible. Al encontrarse definido de forma fija en un documento, se considera “inventario estático”. Ansible admite la inclusión de nuevos nodos en tiempo de ejecución (no listados en el documento de inventario), y esa capacidad es aprovechada si en la factoría DevOps se solicita crear infraestructura en AWS o en Vagrant y además ejecutar las aplicaciones sobre ellos.

- Playbook. Un playbook, o guía, es el fichero de entrada que contiene las tareas y acciones que debe realizar Ansible. En nuestro proyecto este documento hace referencia a una estructura compleja de documentos Ansible que contiene toda la lógica de creación de la factoría DevOps.

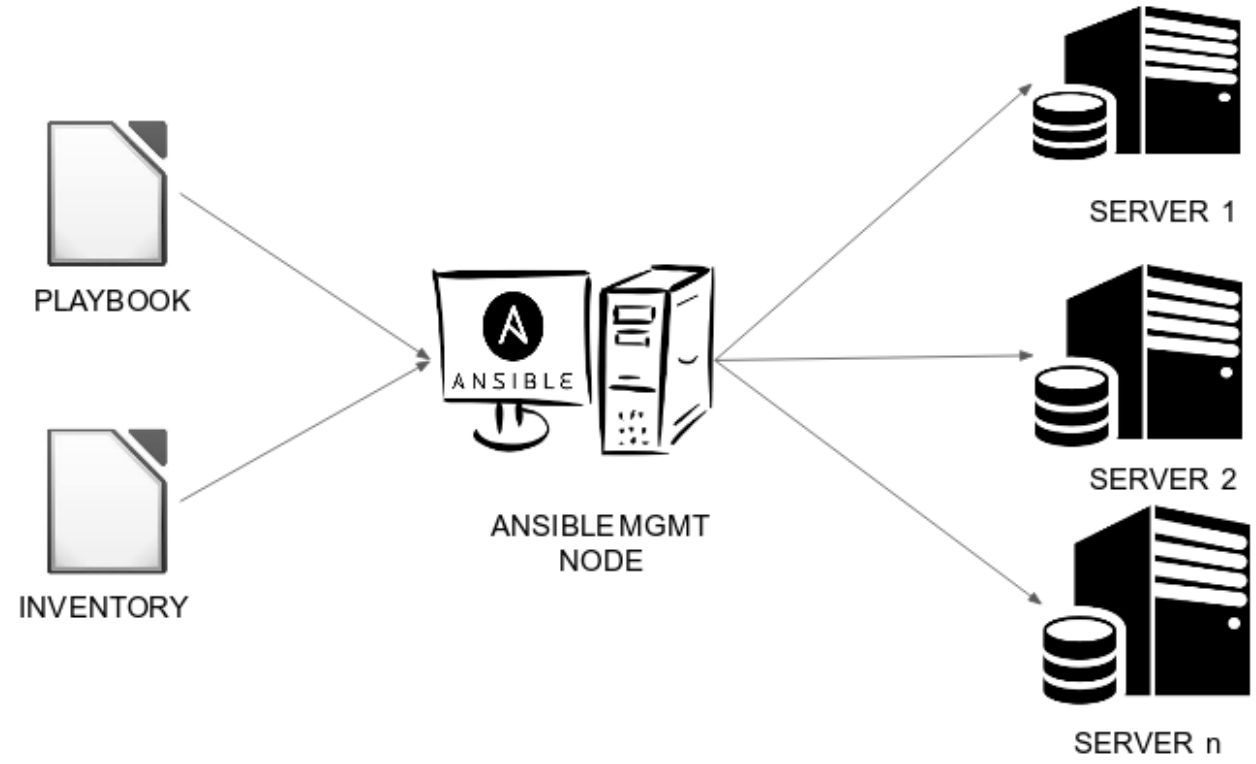


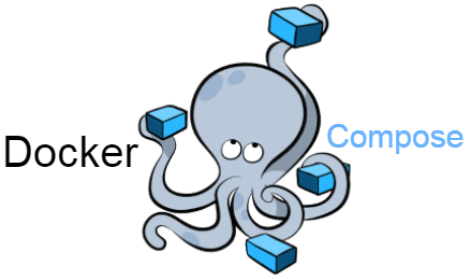
Figura 5. Arquitectura de Ansible

En nuestro proyecto utilizamos Ansible como elemento principal orquestador de la factoría de Software. El documento que el usuario completa es el fichero de Inventario, con el objetivo de que en él encuentre toda la información relativa a la factoría DevOps: Qué conjunto quiere seleccionar de la factoría, así como el entorno donde quiere que se despliegue.

Docker es un software open source que ejecuta aplicaciones dentro de contenedores, proporcionando una capa adicional de abstracción y automatización de virtualización de aplicaciones sobre cualquier sistema operativo con soporte Docker. Docker utiliza características de aislamiento de recursos del kernel Linux para permitir que contenedores independientes se ejecuten sobre una misma instancia.

A diferencia de las máquinas virtuales, las aplicaciones que se ejecuten en contenedores Docker deben diseñarse para que solo se ejecute un proceso. De esta manera los contenedores se consideran como procesos, facilitando su administración.

En nuestro proyecto, para garantizar el despliegue en cualquier tipo de infraestructura, nos aseguramos que todas las aplicaciones de la factoría DevOps corran en contenedores Docker. De esta forma nos abstraemos sobre cuál es el sistema operativo sobre el que deben correr las aplicaciones.



Docker Compose es una herramienta que se apoya sobre Docker y ayuda a definir y compartir aplicaciones compuesta por varios contenedores o sistemas compuestos por diversas aplicaciones. La definición de los documentos de docker-compose es con el lenguaje YAML, y en este se definen todos los requisitos para su ejecución y, con un solo comando, ponerlo todo en marcha o eliminarlo.

La gran ventaja de usar Compose es que puede definir la pila de la aplicación en un archivo, mantenerlo en la raíz del repositorio del proyecto (ahora tendrá control de versiones) y permitir que un tercero contribuya al proyecto. Un usuario solo tendría que clonar el repositorio e iniciar la aplicación Compose. De hecho, es posible que vea bastantes proyectos en GitHub/GitLab en los que se hace exactamente esto.

En nuestro proyecto, toda la capa de aplicaciones DevOps está gestionada por un documento de docker compose construido de forma dinámica desde Ansible, permitiendo el despliegue de las aplicaciones con el comando único antes referenciado.

Terraform es una herramienta Open Source creada por Hashicorp (http://hashicorp.io) dedicada a la administración de Infraestructura como código (IaC). Gracias a Terraform, podemos definir infraestructura mediante documentos escritos en un lenguaje declarativo aplicando estas definiciones en múltiples proveedores de infraestructura. En la actualidad es la herramienta más utilizada en entornos enterprise debido a la gran compatibilidad de entornos que permite, siendo los más conocidos Amazon Web Services, Microsoft Azure, Google Cloud, Oracle Cloud, OpenStack, VMWare vSphere, entre otros.

Terraform permite crear infraestructura híbrida, es decir, puede manipular distintos proveedores de infraestructura sobre un mismo documento, manteniendo un control de estado del mismo sobre unos ficheros auxiliares en formato JSON los cuales podemos manipular con aplicaciones de terceros.

El funcionamiento de Terraform es simple. Consta de un software ejecutable simple, sin dependencias. El software se ejecuta junto a los manifiestos que declaran la infraestructura en el estado que queremos que se configure. Terraform detecta los proveedores de infraestructura sobre los que actuará y descarga un plug-in por cada proveedor en la carpeta de trabajo, obteniendo las instrucciones actualizadas (por ejemplo, instrucciones propias para

orquestar acciones con Amazon Web Services), y acto seguido comprueba la información del documento con la infraestructura existente en el proveedor, y hace una comparativa.

Tras el análisis de Terraform, éste informa al usuario de los cambios que realizaría para construir la infraestructura. Con la afirmación del usuario, realiza las acciones necesarias y crea unos documentos de estado auxiliares que permite a Terraform iterar sobre futuros cambios en el manifiesto. Si en el futuro, se añaden o eliminan recursos, Terraform es capaz de realizar las acciones necesarias para adaptar la infraestructura del proveedor con lo reflejado en el documento. Del mismo modo, si alguien realiza un cambio en la infraestructura sin reflejar en los manifiestos, en la siguiente ejecución de Terraform éste detecta estos cambios no reflejados y vuelve a aplicar la configuración descrita en el manifiesto, deshaciendo dichos cambios.

En nuestro proyecto de DevOps Factory damos la posibilidad al usuario de crear los manifiestos necesarios para desplegar la infraestructura necesaria en Amazon Web Services, por lo que si el usuario selecciona dicho entorno generamos las plantillas necesarias para el despliegue. Además, gracias a Ansible también podemos realizar el despliegue como tal.

Vagrant es una herramienta que nos ayuda a crear y manejar máquinas virtuales en un mismo entorno de trabajo. Vagrant está enfocado en la administración de proveedores de infraestructura local, como VirtualBox, permitiendo una . Si bien Terraform permite recientemente administrar

VirtualBox, Vagrant fue concebida por la misma empresa Hashicorp como una alternativa sencilla de manipulación de máquinas virtuales en entorno local.

En nuestro proyecto, en caso de seleccionar vagrant, creamos un manifiesto de Vagrant con una máquina Centos 7, y en su interior todos los paquetes necesarios para ejecutar el contenido de aplicaciones. De forma adicional se mapean los puertos para conectar a los servicios.

Amazon Web Services es la plataforma en la nube más adoptada y completa del mundo. Permite a particulares y empresas el consumo de recursos tecnológicos bajo demanda, a un precio bajo e inicialmente sin contratos de permanencia. Fundada en 2006, cuenta con más de 200 servicios distintos y con centros de computación de datos distribuidos en todo el mundo. A la fecha de escritura de esta memoria Amazon Web Services está trabajando en la construcción de un centro de procesamiento de datos en Zaragoza, estimándose su finalización a mediados de 2022, siendo el primer proveedor en la nube que ofrece sus servicios en la región ibérica.

Amazon Web Services ofrece sus servicios bajo cuatro modelos ampliamente conocidos dentro de la computación elástica que denominamos brevemente a continuación:

- **IaaS**: Infrastructure as a Service o Infraestructura como Servicio, hace referencia a la capacidad de proporcionar recursos de infraestructura a nivel de Hardware, donde el cliente selecciona el sistema operativo y dispone de permisos de administración sobre la misma. Es el cliente quien debe hacerse cargo de desplegar las aplicaciones, hacer

las copias de seguridad y de mantener la seguridad en dicha máquina virtual. El proveedor nos cobra por los recursos de hardware solicitados y de la licencia del sistema operativo si la tuviera.

- **PaaS**: Platform as a Service o Plataforma como Servicio, hace referencia a la capacidad de desplegar una plataforma de software específica sobre la que tendremos una capacidad de uso y administración limitada al funcionamiento propio de dicha plataforma. Todo lo relativo al mantenimiento de la plataforma (Sistemas operativos, versiones de la plataforma) es gestionado por el proveedor. Un ejemplo de plataforma sería la administración de un cluster de Kubernetes, software ámpliamente utilizado en entornos enterprise para la administración distribuida de contenedores Docker. El proveedor cobra por la infraestructura necesaria para ejecutar la plataforma y un pequeño canon por la administración limitada de dicha plataforma.

- **SaaS**: Software as a Service o Software como Servicio, permite hacer uso de aplicaciones específicas sin necesidad de preocuparnos por la administración y mantenimiento del mismo. Un claro ejemplo son las bases de datos: El proveedor nos proporciona unos datos de conexión a una base de datos mantenida por él, el cual hacemos un uso de usuario; mientras todo lo relativo a su administración, alta disponibilidad, redundancia, copias de seguridad, auditoría interna, etc. es realizada por el proveedor. Tiene un coste mayor a los modelos vistos anteriormente puesto que delegamos mayor responsabilidad al proveedor.

- **FaaS**: Function as a Service o Función como Servicio, va a un nivel más allá del Software. Se trata de una forma de ejecutar una pieza de código software de forma "serverless" (sin servidor dedicado), siendo una forma mayor de abstracción ya que no requerimos de un servidor o una aplicación dedicada sobre la que soportar el pago completo. Generalmente el modelo de pago es por número de peticiones.

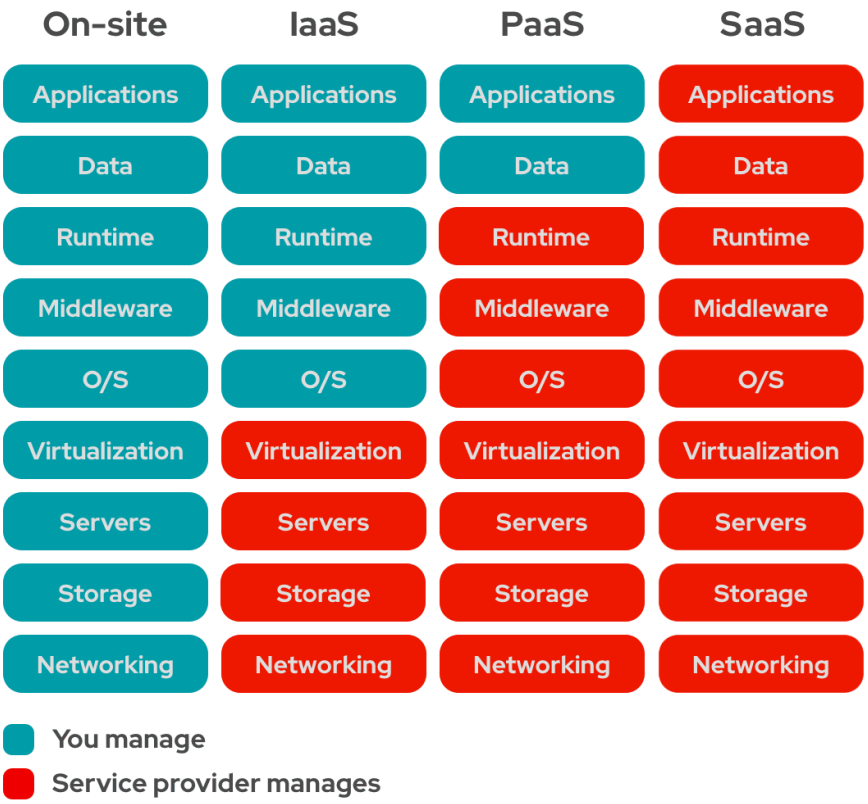
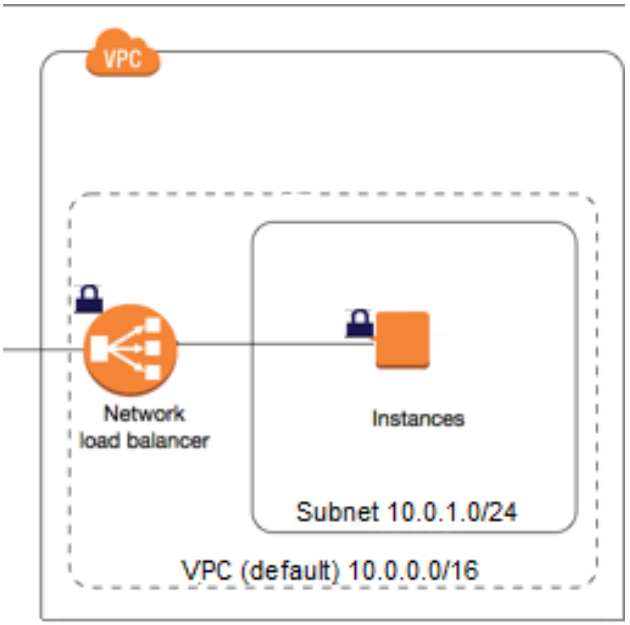


Figura 6. Comparativa On-Site, IaaS, PaaS y SaaS

En nuestro proyecto, si seleccionamos AWS como proveedor de infraestructura, creamos unos manifiestos ejecutables por Terraform que crean una arquitectura en dos capas.

Para garantizar seguridad en el despliegue, creamos un recurso de aislado de Amazon denominado Virtual Private Cloud (VPC) la cual aísla los recursos frente a cualquier otro elemento ya existente en la cuenta de Amazon en cuestión. En su interior, se asignan unas subredes de trabajo para finalmente disponer de una máquina virtual (IaaS) donde se instalará el software necesario para ejecutar la plataforma de contenedores (No desplegamos bajo modelo PaaS puesto que la plataforma que utilizamos, docker-compose, no está disponible).

La máquina virtual está detrás de un balanceador de carga, que nos permite obtener un DNS único de Amazon, además de garantizar una capa de seguridad adicional, puesto que podemos elegir qué puertos exponemos en Internet, dando esa segunda capa de acceso. En iteraciones futuras sobre este proyecto, si se requiere un número mayor de máquinas virtuales podemos crecer detrás del balanceador de carga, dándonos la alta disponibilidad y rotación sobre los recursos proporcionados por las máquinas virtuales



5. Arquitectura

La factoría DevOps se ha construido utilizando la aplicación Ansible y sus características, más concretamente a través de un rol, el cual contiene toda la lógica en la fase de creación de los manifiestos de la factoría, así como el despliegue si se desea. En la figura 7 se muestra una arquitectura de ejecución a alto nivel, donde Ansible procesa la configuración de entrada y produce los documentos resultantes junto con las acciones adicionales si se desea.

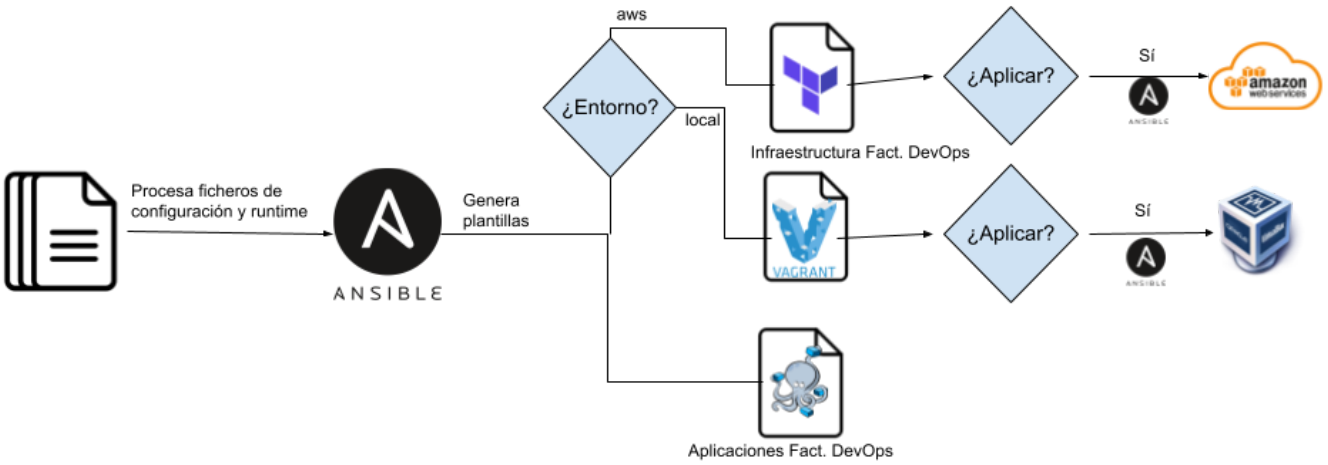


Figura 7. Arquitectura de ejecución de DevOps Factory

La documentación oficial de Ansible dispone de una guía de buenas prácticas respecto al uso de Ansible (https://docs.ansible.com/ansible/2.9/user_guide/playbooks_best_practices.html#directory-layout) , el cual incluye una sección para la organización de los documentos que componen sus scripts. En nuestra factoría tenemos una disposición reducida con los ficheros estrictamente necesarios, como se muestra en la figura 8, y se explica a continuación:

- **README.md:** Fichero “leeme” escrito en sitanxis MarkDown relativo a este proyecto interpretable por el usuario y el sistema de gestión de código GIT

- **ansible.cfg:** Fichero con los parámetros de la propia aplicación de Ansible. No confundir con los parámetros de usuario.
- **devops_factory:** Carpeta que contiene el Rol de Ansible
    - **README.md:** Fichero "leeme" escrito en sitanxis MarkDown relativo a este proyecto interpretable por el usuario y el sistema de gestión de código GIT.
    - **defaults:** Carpeta que contiene los ficheros con variables por defecto para la factoría DevOps.
    - **meta:** Carpeta que contiene metainformación del rol de Ansible.
    - **tasks:** Carpeta que contiene los ficheros con las tareas que ejecuta Ansible. Por defecto, se carga el fichero main.yml, siendo los demás cargados en tiempo de ejecución.
    - **templates:** Carpeta que contienen, en formato plantilla, configuraciones y ficheros necesarios de las aplicaciones que despliega la factoría. Estas plantillas se consolidan en ficheros finales junto con las variables obtenidas y definidas por el usuario, a través del fichero de inventario y/o del contenido de la carpeta defaults.
- **inventory:** Fichero con lista de máquinas para la ejecución de Ansible, donde se indica la máquina principal. Además, lugar donde almacenar las principales variables de la factoría DevOps.
- **playbook.yaml:** Fichero principal para la ejecución de Ansible. Este fichero referencia al contenido de la carpeta anterior "devops_factory".

```
.
├── README.md
├── ansible.cfg
├── devops_factory
│   ├── README.md
│   ├── defaults
│   │   └── main.yml
│   ├── meta
│   │   └── main.yml
│   ├── tasks
│   │   ├── apply_docker_compose.yaml
│   │   ├── env_aws.yaml
│   │   ├── env_local.yaml
│   │   ├── env_vagrant.yaml
│   │   ├── main.yml
│   │   ├── presanity_checks.yaml
│   │   └── template_docker_compose.yaml
│   └── templates
│       ├── 000_envs
│       │   ├── 000_Vagrantfile.j2
│       │   ├── 000_none.yaml
│       │   ├── main.tf.j2
│       │   ├── outputs.tf.j2
│       │   └── variables.tf.j2
│       ├── 100_docker-compose.yaml.j2
│       ├── gitea
│       │   └── app.ini.j2
│       ├── gogs
│       │   └── app.ini.j2
│       ├── jenkins
│       │   └── config.xml.j2
│       ├── sonar_postgres
│       │   └── init-database.sh.j2
│       └── traefik
│           ├── acme.json.j2
│           └── traefik.toml.j2
├── inventory
└── playbook.yaml

11 directories, 26 files
```

Figura 8. Estructura de documentos de la factoría DevOps

En el fichero de Inventario de Ansible se encuentra la configuración principalmente accesible. Bajo una condiciones normales, este es el único punto que el usuario necesita personalizar acorde a las variables descritas y definidas. La estructura de este documento es con el lenguaje YAML, y en la Figura 8 mostramos la estructura del mismo:

```
all:
  hosts:
    master:
      ansible_connection: local
      ansible_python_interpreter: python3
      # Server configuration
      env: local #(vagrant|aws|local)
      apply_env: true #(true|false)
      apply_dockerfile: true #(true|false)

      ## Toolchain
      traefik: false #(true|false)
      cadvisor: true #(true|false)
      git_server: gogs #(gogs|gitea|none)
      continuous_integration: none #(jenkins|drone|none)
      quality_software: none #(sonarqube|none)
      artifact_repository: none #(nexus|none)
      chat: none #(rocketchat|none)
      logging: none #(graylog|none)

      #AWS
      aws_access_key: <AWS Access Key>
      aws_secret_key: <AWS Secret Key>
      aws_instance_type: "t2.large"
      aws_unique_deployment_name: tfm_devops

      aws_ec2_username: ec2-user
                                                aws_region:        us-east-1
#(us-east-1|us-east-2|us-west-1|us-west-2|eu-west-1)
                                                              terraform_url:
https://releases.hashicorp.com/terraform/1.0.2/terraform_1.0.2_linux_amd
64.zip #https://terraform.io/downloads.html

      #Vagrant
      vagrant_machine_gb: 4
```

Figura 9. Configuración en el fichero de inventario

En las siguientes tablas describimos la información relativa a cada una de las variables contenidas en el fichero de inventario y su comportamiento por defecto.

En la tabla 1 describimos las variables relativas a la selección del entorno y su posible aplicación posteriormente

Variable	Parámetro por defecto	Parámetros disponibles	Descripción
`env`	"local"	vagrant | aws | local	Tipo de infraestructura sobre la que crear la factoría DevOps. Con "local", solo crea el manifiesto docker-compose.yaml. Con aws, crea los manifiestos necesarios para desplegar infraestructura con Terraform. con Vagrant crea los manifiestos necesarios para desplegar la infraestructura en Virtualbox a través de Vagrant.
`apply_env`	true	true | false	Con true, el script ejecuta el despliegue de infraestructura de forma automatizada con Ansible. False omite el paso.
`apply_dockerfile`	true	true | false	Con true, se despliega el manifiesto de docker-compose relativo a las aplicaciones con Ansible. False omite el paso.

Tabla 1. Variables funcionamiento entorno

En la tabla 2 describimos las opciones disponibles como aplicaciones que permite la factoría DevOps, donde podemos elegir una aplicación para cada componente, u omitirlo si no estamos interesados en esa parte del proceso.

Variable	Parámetro por defecto	Parámetros disponibles	Descripción
`traefik`	false	true | false	Beta: Uso de proxies inversos para el acceso de las aplicaciones en lugar de acceso por puerto.
`cadvisor`	true	true | false	cadvisor es un servicio de monitorización de contenedores. Útil para observar rendimiento, CPU, disco...
`git_server`	gogs	gogs | gitea | none	Servidor Git. gogs es una aplicación minimalista de Git. Gitea se define como una alternativa a Github. none no despliega servidor git en el laboratorio
`continuous_integration`	jenkins	jenkins|drone|none	Servidor de integración continua. jenkins es el sistema de integración continua más popular en entorno empresarial. drone es una alternativa a jenkins con una ejecución simplificada. none no despliega servidor de integración continua.
`quality_software`	sonarqube	sonarqube | none	sonarqube es el principal software open source para pruebas unitarias y análisis estático de código fuente. none no despliega herramienta de calidad
`artifact_repository`	nexus	nexus | none	nexus es el software open source más conocido para gestión de paquetes. none no despliega repositorio de artefactos.
`chat`	rocketchat	rocketchat | none	rocketchat es una alternativa equivalente al conocido Slack, una herramienta de chat basada en canales con capacidad de conexión de aplicaciones mediante webhooks. none no despliega herramienta de

			chat.
`logging`	graylog	graylog | none	graylog es la principal herramienta open source para gestion de logs, equivalente al producto enterprise Splunk. none no despliega heramienta de recolección de logs.

Tabla 2. Variables de herramientas disponibles

En la tabla 3 describimos las opciones necesarias para desplegar el entorno en la nube Amazon Web Services si hemos elegido esa opción y queremos que Ansible haga las plantillas con esos credenciales. Recordemos que los credenciales se obtienen desde el panel de control de AWS, sección *"My security credentials"*

Variable	Parámetro por defecto	Parámetros disponibles	Descripción
`aws_access_key`	< Tu clave de acceso>	-	Clave de acceso obtenida en Amazon IAM
`aws_secret_key`	<Tu clave secreta>	-	Clabe secreta obtenida en Amazon IAM
`aws_instance_type`	"t2.large"	Cualquier tipo disponible en AWS	AWS identifica a cada máquina con un nombre dependiendo de los recursos solicitados. Se puede consultar la lista completa aquí.
`aws_unique_deployment_name`	"tfm_devops"	Prefijo que se añadirá en los recursos de amazon	Con ello, evitamos posible solapamientos con otros recursos en ejecución, como otra factoría ya desplegada.

Tabla 3. Variables de Amazon Web Services

En la tabla 4 describimos las variables relativas al despliegue en máquina virtual local mediante VirtualBox utilizando Vagrant. A tenor de la simplicidad, actualmente las opciones de personalización y ajuste permiten adaptar el dimensionamiento de memoria RAM.

Variable	Parámetro por defecto	Parámetros disponibles	Descripción
`vagrant_machine_gb`	4	{ integer }	Número de GB de la máquina virtual.

Tabla 4. Variables de Vagrant

Para finalizar, en la tabla 5 describimos cuáles son los puertos generales de acceso de las herramientas. Sea como fuere la infraestructura donde se ejecute el manifiesto docker-compose, éste requiere del número de puertos comprendido entre tcp/8080 y tcp/8090, así como los puertos udp/8086 y udp/8087 para la aplicación de monitorización.

Servicio	Variable	Puerto
Servidor git	`git_server`	8081 (8089 tcp/22)
Integración Contínua	`continuous_integration`	8080 (tcp/8443) (drone runner 8088)
Calidad de software	`quality_software`	8082
Repositorio de artefactos	`artifact_repository`	8083
Chat	`chat`	8084
Logging	`logging`	8085 (8086 tcp y udp para syslog, 8087 tcp y udp para GELF)
cAdvisor	`cAdvisor`	8090

Tabla 5. Puertos disponibles en aplicaciones seleccionadas

6. Funcionamiento y workflow

A continuación describimos cómo utilizar la factoría DevOps. La pieza principal de software es Ansible, procesa la configuración escrita por el usuario generando las plantillas de la factoría DevOps correspondiente. Además, Ansible puede desplegar la factoría si el usuario lo desea. Para todo ello, la máquina dónde se ejecute debe cumplir unos requisitos mínimos que describiremos a continuación.

Requisitos mínimos:

- Sistema Linux: Debido a cuestiones de compatibilidad de los softwares utilizados, el proyecto se ha realizado sobre un sistema operativo basado en Debian GNU/Linux. Por ejemplo, relativo a Ansible, éste no es compatible sobre sistemas Windows, como podemos leer en su página oficial. [https://docs.ansible.com/ansible/latest/installation_guide/intro_installation.html]
- **Ansible:** Ansible en su versión 2.X. Este proyecto se ha realizado con versión 2.9.0
- **Python 3:** Recomendado versión 3.8 o superior. Las distribuciones principales de Linux traen por defecto una versión actualizada de Python. Las bibliotecas de Python relacionadas con Ansible se instalar de forma automática con su instalación a través de los medios comunes como apt, yum o pip.

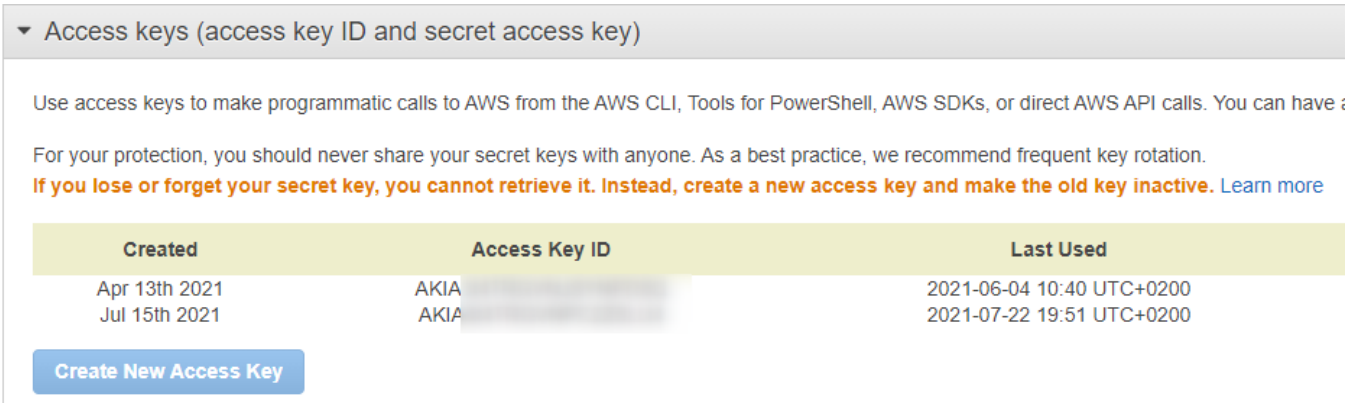
Además, para despliegue de entornos en local:

- **Virtualbox:** Software para emulación de máquinas virtuales sobre el propio ordenador en el que queremos ejecutar la factoría.

- **Vagrant:** Software encargado de procesar el manifiesto de máquina virtual y ejecutarlo sobre el proveedor VirtualBox

En caso de despliegue sobre Amazon web Services:

- **Credenciales de AWS válidos**: Disponibles desde la consola de Amazon Web Services: Click en nombre de cuenta → My security Credentials → Access Keys. Más información en https://docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html



Una vez cubiertos los requisitos mínimos descritos anteriormente, descargamos y descomprimimos el artefacto tfm-devops-factory en una carpeta de trabajo.

Bajo los principios de arquitectura seguidos, la factoría ha sido diseñada para recoger en un único fichero, - el inventario - todas las variables y propiedades principales de la misma.

Una vez configurado, ejecuta el comando

```
$ ansible-playbook playbook.yaml
```

Durante la ejecución se crea una carpeta con el nombre output, el cual contiene todos los manifiestos y ficheros necesarios para el despliegue de la factoría devops con los parámetros solicitados.

Si en la ejecución se solicitó el despliegue de la infraestructura sobre el proveedor, y la aplicación del manifiesto docker-compose.yaml para desplegar la aplicación, los ficheros ubicados en el ordenador actual también se copian en el servidor recién creado.

7. Conclusiones

Este trabajo de fin de máster se ha realizado con la intención de crear una primera versión de un entorno de aplicaciones utilizadas en los proyectos de desarrollo de software utilizando los principios de DevOps completamente funcional para su uso educativo o para comienzo de proyectos pequeños, proporcionando las principales herramientas Open Source que la mayoría de empresas utilizan por su solidez y profesionales en el mercado.

Gracias a esta factoría, cualquier estudiante o interesado en estas tecnologías puede crear un entorno completo y seguro para aprender, probar, y mejorar sus habilidades relacionadas con el desarrollo de software utilizando este conjunto de herramientas sin necesidad de tener que instalar, configurar o adquirir conocimientos adicionales (como por ejemplo habilidades avanzadas de Docker, docker-compose, Ansible, Terraform o Vagrant) que no son necesarios para los desarrolladores de software.

Si bien el enfoque es facilitar el uso de las herramientas a los desarrolladores, la naturaleza de este proyecto también permite aprender, probar y administrar las tecnologías utilizadas para construir la propia factoría DevOps: como Ansible, Terraform, Vagrant, Docker y Docker Compose es otro gran aspecto a tener en cuenta, puesto que son tecnologías que actualmente son muy demandadas por las empresas, lo cual garantiza una buena entrada en el mercado laboral.

Por último, esta factoría admite una gran capacidad de personalización, así como la inclusión de nuevas tecnologías, nuevas etapas en el proceso de desarrollo (como por ejemplo, seguridad, simulación de aplicaciones reales o “mocking”) o nuevas etapas en el despliegue de software; pudiendo gestionar el propio proyecto dentro de una factoría DevOps. Es decir, la propia factoría se puede gestionar como un proyecto de software sobre las propias herramientas que despliega, generando una versión mejorada.

8. Líneas de trabajo futuras

Esta primera versión de factoría DevOps incluye opciones establecidas por defecto:

- Un primer paso es permitir una mayor capa de personalización en las opciones de configuración, tanto de la nube, como por ejemplo no tener que crear una VPC aislada para nuestra factoria, si no que pudiera convivir con productos actuales, así como facilitar la personalización de la capa de aplicaciones.

- Terraform proporciona a la fecha de esta memoria la mayor capacidad de interacción respecto a entorno que con cualquier otra herramienta. Siguiendo unos pasos similares a los manifiestos de relativos a AWS, poder extender la creación de infraestructura a otros proveedores, tanto en On Premise (como por ejemplo OpenStack), como otros proveedores en la nube (Microsoft Azure, Google Cloud, Oracle Cloud).

- La arquitectura actual no incorpora sistema de monitorización avanzado. Actualmente se realiza con cAdvisor, que proporciona información relativa a los contenedores Docker sobre los que tiene visibilidad, pero si se realizan despliegues en otras máquina, éste sistema no es capaz de visualizarlo. Es por ello, que incorporar un sistema de monitorización Open Sauce, como el stack ELK (https://www.elastic.co/es/what-is/elk-stack), es una opción muy interesante para mejorar.

- Incrementar el número de opciones respecto a las aplicaciones. Gracias a la tecnología Docker, podemos incluir aplicaciones complejas en contenedores con funcionamiento muy delimitado. Incluir aplicaciones adicionales incrementando el número de aplicaciones disponibles.

9. Bibliografía

- Software: A €910 Billion Catalyst for the EU Economy: https://softwareimpact.bsa.org/eu/
- Desarrollo en cascada: https://es.wikipedia.org/wiki/Desarrollo_en_cascada
- Paradigma Digital. (2015). *Qué es DevOps (y sobre todo qué no es DevOps)*. https://www.paradigmadigital.com/techbiz/que-es-devops-y-sobre-todo-que-no-es-devops
- Amazon Web Services: https://aws.amazon.com
- Docker Inc., Docker: https://www.docker.com
- Docker Inc., Docker compose: https://docs.docker.com/compose
- Drone, Inc,. Drone: https://www.drone.io
- Google cAdvisor: https://github.com/google/cadvisor
- Graylog Inc., Graylog: https://www.graylog.org
- Hashicorp Vagrant: https://vagrantup.com
- Hashicorp Terraform: https://terraform.io
- Jenkins Inc,. Jenkins: https://www.jenkins.io
- RedHat Ansible: https://www.ansible.com
- Rocket.chat Inc,. Rocket.chat: https://rocket.chat
- SonarSource Sonarqube: https://www.sonarqube.org
- Oracle VirtualBox: https://virtualbox.com
- The Twelve-Factor App: https://12factor.net