DevOps

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universidad Rey Juan Carlos

gsyc-profes (arroba) gsyc.urjc.es

Noviembre de 2018

Contenidos

Definición de DevOps

DevOps es un término acuñado por Andrew Shafer y Patrick Debois en la conferencia de desarrollo *Agile* del año 2008
Se origina con la composición de dos palabras:

- *Development*
  Desarrollo, programación de software en sentido amplio, esto es, análisis, diseño, codificación y prueba
- *Operations*
  Operaciones, explotación del software, puesta en producción, uso real

Definición de Bass, Weber y Zhu [4]:
DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality

Definición de Davis y Daniels [2]:
Devops is a cultural movement that changes how individuals think about their work, values the diversity of work done, supports intentional processes that accelerate the rate by which businesses realize value, and measures the effect of social and technical change. It is a way of thinking and a way of working that enables individuals and organizations to develop and maintain sustainable work practices. It is a cultural framework for sharing stories and developing empathy, enabling people and teams to practice their crafts in effective and lasting ways

Definición de Huttermann [1]:
DevOps is a mix of patterns intended to improve collaboration between development and operations. DevOps addresses shared goals and incentives as well as shared processes and tools. Because of the natural conflicts among different groups, shared goals and incentives may not always be achievable. However, they should at least be aligned with one another

Aquí lo definiremos como

`Grupo de técnicas que buscan optimizar el trabajo conjunto de desarrolladores de software y administradores de sistemas`

- Optimizar: que sea rápido, sencillo, barato, de calidad y sin conflictos

Estas *técnicas* se agrupan en dos grandes categorías

- Culturales, políticas, organizativas. Referidas a la interacción entre personas
- Herramientas software

Las segundas son importantes, pero las primeras, más

Hay algunas cosas relativamente claras:

- Qué es *DevOps*
- Qué no es *DevOps*
- Qué problemas se quieren solucionar
- Qué objetivos se buscan
- Lo relativo a herramientas software

Lo que no está tan claro es *cómo*. *DevOps* es una disciplina compleja

- Organizar el trabajo de personas es difícil. No se aprende en un libro. Lo que puede valer en un entorno, puede ser inaplicable en otro

¿Qué NO es DevOps? (1)

- *DevOps* no significa que desaparezca la frontera entre desarrollo y producción
- No significa que los desarrolladores controlen el software en producción
- No significa que los administradores editen el código fuente
- Según la bibliografía especializada, *DevOps* nunca debería ser un departamento en una empresa, ni un cargo. No tiene sentido hablar de *Ingeniero DevOps*
  - Aunque la industria sí usa este término
- No significa que una sola persona desarrolle y opere
  - Excepto tal vez en empresas muy pequeñas
- No significa que una persona trabaje por dos (y cobre por una)

¿Qué NO es DevOps? (2)

- *DevOps* no es una herramienta software. Tienen su utilidad, pero ni son suficientes ni son imprescindibles
- *DevOps* no es una certificación. No es una metodología concreta y única que se pueda enseñar, que se pueda seguir y de la que uno se pueda examinar

Con frecuencia, en una ofertas de empleo donde se solicita un *ingeniero devops*, realmente lo que se está buscando es un desarrollador con conocimientos de integración continua/entrega continua/despliegue continuo.

Word Cloud para DevOps en ofertas de empleo

Fuente: The DevOps Job Market, Scalyr blog

Desarrollo Ágil

DevOps está muy ligado con el desarrollo de software *agile* (*ágil*), proviene de la misma comunidad, tiene objetivos muy similares

- Podemos considerar *DevOps* una extensión del moviento *agil* a la explotación del software, no solo a su desarrollo

Modelo de desarrollo de software en cascada

El desarrollo en cascada (*waterfall*) es el tradicional, generalmente aceptado y prácticamente único hasta los años 1990.
Formado por pasos que se siguen secuencialmente, de forma rígida, uno tras otro, sin vuelta atrás

1. Análisis de requerimientos
2. Diseño
3. Desarrollo (programación)
4. Prueba
5. Despliegue
6. Mantenimiento

Desarrollo Ágil

En los años 1990 empiezan a aparecer diversas metodologías de desarrollo de software que cuestionan el modelo en cascada

- *rapid application development, the unified process, dynamic systems development method (DSDM), scrum, extreme programming (XP), feature-driven development*

En 2001 se publica el *Manifesto for Agile Software Development* que resume y condensa todas estas metodologías

Manifiesto por el desarrollo ágil de software:

Estamos descubriendo formas mejores de desarrollar
software tanto por nuestra propia experiencia como
ayudando a terceros. A través de este trabajo hemos
aprendido a valorar:

Individuos e interacciones sobre procesos y herramientas.
Software funcionando sobre documentación extensiva.
Colaboración con el cliente sobre negociación contractual.
Respuesta ante el cambio sobre seguir un plan.

Aunque valoramos los elementos de la derecha,
valoramos más los de la izquierda.

12 principios del manifiesto ágil

http://agilemanifesto.org/iso/es/principles.html

Scrum

Scrum es una de las metodologías de desarrollo de software ágil más populares. Una idea dentro de la filosofia *DevOps* es incluir las operaciones en los *sprints* de *Scrum*. Esto se puede hacer de varias formas, es una materia abierta

- Es posible integrar personal de operaciones en los equipos *Scrum*, aunque no es una idea muy habitual, va contra el principio de separación desarrollo-operaciones
- Es más natural una integración más débil: p.e. asistencia de personal de operaciones a las reuniones de *Scrum*, como miembro externo
- También se pueden adaptar las técnicas de *Scrum* dentro del equipo de operaciones

Scrum es una metodología de desarrollo ágil de software elaborada por Ken Schwaber y Jeff Sutherland, publicada en 1995

- Se forman equipos de desarrolladores, típicamente entre 5 y 9 (más un *product owner* más un *scrum master*)
- El trabajo se descompone en ciclos denominados *sprints*, que duran entre 1 y 4 semanas. Típicamente 2
- Al final de cada *sprint* se entrega una versión del software

Equipos de Scrum

En los equipos de *scrum* hay tres roles

- *Dueño del producto, Product owner*
  Es una persona, que representa al cliente. Tiene la visión del producto final y poder de decisión sobre cómo debe ser el producto.
- *Scrum master*
  Es el responsable de que se siga la metodología *scrum* .
  Modera las reuniones, dirije al equipo en lo necesario para que el equipo se auto-dirija
- Miembro del equipo
  Son los desarrolladores. Los equipos son multifuncionales, sin distinción de roles entre analista/programador/tester. Todos puedes hacer cualquier función y son responsables de todo (aunque cada uno tenga una especialidad propia)

Los valores en *scrum* son

- Respeto entre las personas
- Responsabilidad y disciplina auto-impuesta
- Compromiso
- Trabajo enfocado en aportar valor al cliente

Las unidades básicas de construcción del producto son las *historias de usuario*

- El usuario de tipo xxxx quiere hacer yyyy. Esto le aporta el valor zzzz
- Las *historias de usuario* las aporta el *product owner*

Reuniones de trabajo en Scrum

Planificación del *sprint*

- Reunión de todo el equipo, típicamente de unas 4 horas, antes de cada *sprint*
- A partir de las *historias de usuario* que propone el *product owner*, el equipo decide cuáles implementar y cómo

Scrum diario

- Reunión de 15 minutos, del equipo al completo, siempre en el mismo sitio, a la misma hora, de pie, con horario inflexible, falte quien falte
- Cada miembro explica qué hizo ayer, qué hará hoy, qué obstáculos cree que pueden impedir el sprint

Evaluación de *sprint*

- Reunión de unas dos horas al final del sprint
- Se presenta lo realizado (solo lo concluido)
- Se evalúa el trabajo.

Kanban

Kanban es una metodología de gestión de procesos

- Tiene su origen en la industria del automóvil: *Toyota Prodution System* y *Lean Manufacturing*
- Se usa, entre otras cosas, para desarrollo de software, como metodología ágil y especialmente ligera
- Es muy adecuada para *DevOps*. Se puede usar de diversas formas, por ejemplo que desarrollo y operaciones compartan la misma pizarra Kanban, aunque cada equipo gestione sus tareas

El elemento principal es el tablero Kanban, también llamado pizarra Kanbam. Es un diagrama que representa el flujo de trabajo

- Tradicionalmente se usaba una pizarra con tarjetas adhesivas o imanes,
- Hay versiones software, típicamente como aplicación web. P.e. `https://trello.com`

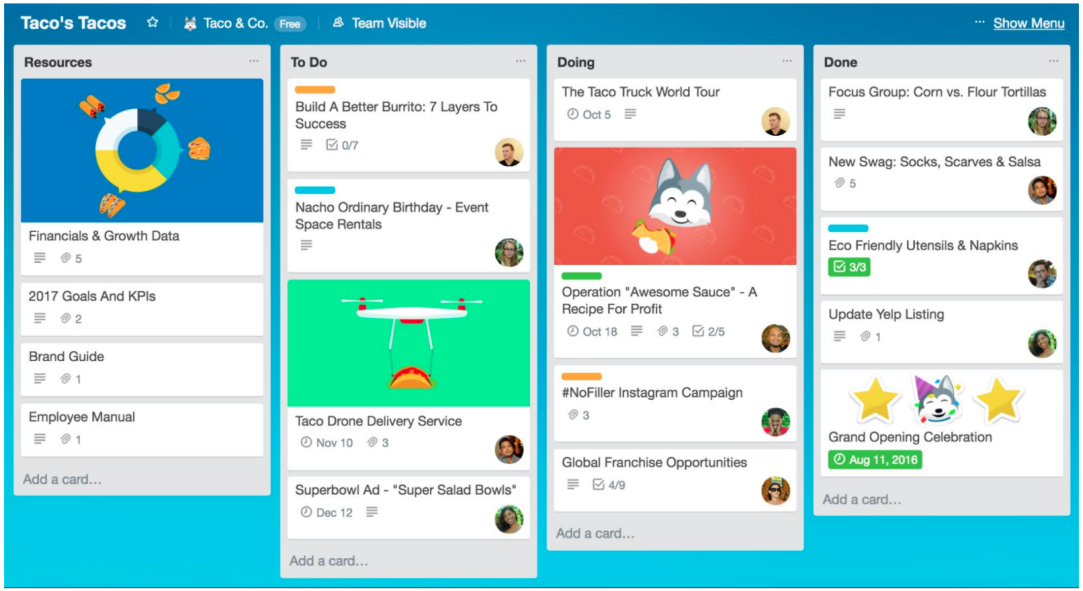


Figura: Tablero Kanban con Trello

- Cada tarea, característica o historia de usuario se anota en una tarjeta o *post-it*, que se va desplazando desde la columna de la izquierda hasta la columna de la derecha
- WIP: *Work in Progress*. Tarjetas que circulan por el tablero. Es importante minimizar el WIP
- El numero de columnas es variable, entre 4 y 7 son valores típicos. La denominación de cada columna se adapta para cada empresa
  Ejemplo:
  `Pendiente | Analizando | En desarrollo | Probando | Aceptado | Producción`

Tarjetas Kanban

Cada tarjeta tiene

- Descripción de la tarea

Puede tener:

- Quién la está haciendo
- Su fecha límite
- Distintos colores
  - Tal vez según la urgencia
  - Más habitualmente, por el tipo de trabajo
    P.e. Verde: mantenimiento. Amarillo: historia de usuario. Rojo: Bug
- Indicador de progreso

Diferencias desarrollo-operaciones

La mayoría de problemas que se procura resolver con *DevOps* parten de que normalmente hay diferencias marcadas entre desarrollo y operaciones. Son equipos muy separados, con diferente lenguaje, culturas, habilidades, objetivos...
Una de las mayores diferencias es que

- Los desarrolladores (analistas, programadores, *testers* y responsables de calidad) buscan el cambio continuamente, para corregir errores y añadir funcionalidad
- Los administradores (administradores de sistemas, de bases de datos y de redes) buscan estabilidad. Ven cualquier cambio como un riesgo potencial. Nadie les agradecerá la nueva funcionalidad. Pero sí les culparán de los problemas de explotación provocados por los cambios

Problemas típicos (1)

Enumeramos a continuación algunos problemas habituales entre el equipo de desarrollo y el equipo de operaciones, que las técnicas *DevOps* buscan solucionar

- El software que funcionaba en los equipos de desarrollo, da errores en producción
  - Desarrollo echa la culpa a operaciones
  - Operaciones echa la culpa a desarrollo
- Aparece algún problema menor en la funcionalidad o el rendimiento. El equipo de desarrollo hace un parche rápido sin pasar todos los controles de calidad
  - Operaciones instala el parche, arregla una cosa pero rompe otra
  - Operaciones no instala el parche, porque sabe que los parches son peligrosos

Problemas típicos (2)

- Desarrollo hace un producto de baja calidad, lo mínimo para ser aceptado
  - El trabajo ya está *hecho*. El diagrama de Gantt está cumplido. Los problemas posteriores no importan, son cosa de *mantenimiento*, de otro contrato, de otra subcontrata, de otro presupuesto...
  - Consumir más recursos (tiempo, esfuerzo) para entregar un producto de más calidad, no reportará beneficios al equipo de desarrollo

Problemas típicos (3)

- Problema contrario al anterior: Desarrollo *anancástico* (demasiado perfeccionista)
  - El equipo de desarrollo prepara un un software con cambios radicales (nuevo lenguajes, nuevas librerías). O preparado para eventualidades poco probables.
  - Todo lo contrario al *pequeño cambio incremental*. No tiene en cuenta las implicaciones para operaciones. Dispara los costes y/o los plazos, peligrando la viabilidad de la empresa. A operaciones o al mercado no llegan las soluciones adecuadas porque la solución *óptima* no está disponible

Problemas típicos (4)

- Para intentar evitar los problemas anteriores, se *mejora* la especificación de los *deliverables* (entregables) que unos proporcionan a otros
  - Negociaciones duras, criterios muy rígidos, contratos complicados, especificaciones a la defensiva. Lo fundamental es, en caso de problema, dejar claro *quién tiene la culpa*
  - Esto aumenta los trámites, la preparación y la frecuencia de las entregas
  - Aumenta la distancia entre los equipos

Problemas típicos (5)

- Cultura del héroe (*rock star*, *ninja*, *crack*)
  - Programador individualista. Con frecuencia hace software con errores y no documentado. Aparentemente es muy valioso porque solo él sabe arreglar esos errores
- Planificación rígida
  - Inicialmente se prepara un diagrama de Gantt. Luego todo se fuerza para que encaje en el diagrama

Valores a promover

- Equipos motivados y productivos
- Compromiso con objetivos y valores comunes
- Respeto al otro
- Colaboración bienintencionada entre las partes/las empresas
- Aceptación de un cierto ratio de errores como inevitables, sin buscar culpables

Todo esto

- Tiene ventajas evidentes
- Es difícil de conseguir

Motivación de equipos

Según Poppendieck, la motivación se puede conseguir con:

- Sensación de pertenencia
- Confianza en una cierta tolerancia a los errores
- Confianza en la capacidad propia y del resto del equipo
- Celebración conjunta de los progresos
  - Teniendo en cuenta que no todo el mundo sale de copas o juega al Paintball

Trabajo en equipo productivo

- Definir objetivos, métodos, pasos, plazos temporales... y reajustarlo cuando sea necesario
- Evitar la microgestión. Que los gestores digan qué hacer, pero sin demasiados detalles del cómo. El equipo se auto-organiza
  - Esto suele ser más eficiente
  - Hace al equipo sentirse más valorado

- Hacer pequeños experimentos. Fallar a menudo pero pronto y con pequeñas cosas
- De vez en cuando (una vez al día, a la semana...) reservar un rato para alguien de operaciones se siente con alguien de desarrollo
- Evitar el *presentismo laboral*. Respetar el equilibrio entre el trabajo y la vida personal. No esperar que los empleados hagan jornadas maratonianas en la oficina y que luego contesten al correo a cualquier hora

Comunicación efectiva

Que los individuos y equipos:

- Comprendan las circustancias y dificultades de los demás
- Busquen influenciar en otros de forma positiva.
  No porque *te lo mando* o *me debes una*, sino porque esto es lo mejor para todos
- Reconozcan el trabajo ajeno. Hacerlo en público es mucho más efectivo. Y si hay que hacer algún reproche, con mucha mano izquierda y en privado

Reuniones de calidad

- Todos los convocados llegan puntuales. La reunión acaba puntualmente
- Meta-decisiones claras. Ya sea por jerarquía, por votación, o idealmente, por consenso
- Los participantes hablan de uno en uno
- Lo que solo afecta a unos pocos, no se trata en el tiempo de todos
- ...

Sin olvidar las

- Discusiones retrospectivas. Reuniones con periodicidad predeterminada para tratar las etapas superadas, para analizarlas y extraer conclusiones.
- Reuniones *post morten*. Similares a las retrospectivas, pero provocadas por un problema concreto. Siempre es necesario trabajar de forma constructiva sin echar la culpa a nadie, pero en estos casos, mas que nunca.

Automatización

La automatización es una técnica fundamental en *DevOps*

Tareas que se pueden automatizar:

- Construcción (compilación)
- Pruebas
- Despliegue (puesta en producción)
- Configuración en los distintos entornos
- Monitorización
- Control de incidencias

Ultimamente se ha introducido el término *orquestar*:

- Automatizar
  Usar herramientas (scripts o similares) que permitan realizar una tarea sin intervención de una persona
- Orquestar
  Coordinar diversas automatizaciones de tareas individuales, para que formen procesos / flujos de trabajo

Automatizar (incluyendo orquestar) es, en general, positivo. Con algunas salvedades

- Debemos asegurarnos de que merezca la pena. Que el esfuerzo necesario para preparar y mantener la automatización, sea menor que el esfuerzo de realizar las tareas a mano.
- Paradoja del exceso de automatización
  Inevitablemente, habrá ocasiones en que el sistema requiera intervención humana (errores, cambios no previstos...).
  Cuánto más automatizado esté el sistema:
  - Más complejos serán estos cambios, más especializado tendrá que ser el personal
  - Menos especializado será el personal del día a día.
    Como esto lo puede llevar cualquiera, el resultado es que lo acaba llevando cualquiera

Técnicas de despliegue (1)

- Despliegue frecuente
  Un cambio grande puede ser muy drástico. Por el contrario, el despliegue frecuente pone en producción pequeños cambios, de forma continua
  - Esto familiariza a todo el equipo con el proceso de introducir novedades
  - Los cambios menores implican problemas potenciales menores
  - Hay técnicas más avanzadas (integración continua, entrega continua, despliegue continuo). Pero realizar al menos *despliegue frecuente* es prácticamente imprescindible dentro de la filosofía *DevOps*

Técnicas de despliegue (2)

- Conmutación de funciones
  Ejemplo: Ponemos en producción funcionalidad nueva. Pero si falla y decidimos desactivarla, se puede hacer con un conmutador sencillo desde el código. Sin necesidad de volver a desplegar el código *viejo*
  - Los contenedores pueden hacer innecesaria esta técnica
- *Dark Launching*
  Las nuevas versiones se aplican solo a unos pocos usuarios
  - Esto facilita la corrección de problemas y limita los problemas potenciales
  - Pueden ser los empleados, pueden ser voluntarios, pueden ser usuarios que hemos detectado como *avanzados* o pueden ser aleatorios

Técnicas de despliegue (3)

- *Blue Green Deployment*
  La versión nueva y la versión anterior se preparan para que funcionen en paralelo
  - Para conmutar de la *versión azul* a la *versión verde* no hay que cambiar el código, solo el router/el *balanceador* de carga o algún fichero de configuración

Integración, entrega y despliegue continuo

Las siguiente técnicas son habituales en *DevOps*, aunque

- No son imprescindibles para hacer *DevOps*
- Implementarlas no significa estar haciendo *DevOps*

En cualquier entorno de desarrollo moderno de cierto tamaño, hay varios desarrolladores, utilizando un sistema de control de versiones. Cada uno tiene su copia de trabajo del software, que con cierta periodicidad, integra en el repositorio principal

- *Continuous Integration* (CI)
  Realizar esta integración muy a menudo. Típicamente varias veces al día

- *Continuous Delivery* (CD)
  No solamente hacer *Continuous Integration*, sino dar un paso más allá. Además de integrar el código, asegurarse de que está listo para ponerse en producción muy a menudo. Esto es, pasar los controles de calidad y automatizar la puesta en producción. Tal vez no tan a menudo como la CI, pero sí muy a menudo. P.e. una vez al día.
- *Continuous Deployment*
  No solo hacer *Continuous Delivery*, sino dar un paso más allá. Además de asegurarse de que el código tiene calidad como para ponerse en producción, ponerlo realmente en producción

Herramientas

Las siguientes herramientas son habituales cuando se siguen los principios *DevOps*

- Contenedores Docker

  `https://gsyc.urjc.es/~mortuno/lagrs/02-virtualizacion_I.pdf`

  `https://gsyc.urjc.es/~mortuno/lagrs/02-virtualizacion_III.pdf`

- Ansible
- Jenkins
- Vagrant

  `https://gsyc.urjc.es/~mortuno/lagrs/vagrant.pdf`

Jenkins

Jenkins

`https://jenkins.io`

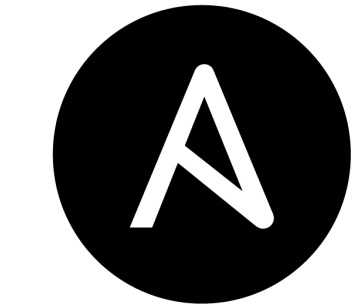
Jenkins es una herramienta para implementar Integración Continua (C.I.)

- Aparece en el año 2011. Es software libre, muy popular
- Aplicación basada en web

- Va un paso más allá de herramientas como Maven, que construyen el fuente pero no hacen C.I.
- Su entorno nativo es Java, pero tiene *plugins* para distintos lenguajes, herramientas de control de versiones, de virtualización, de testing, de comunicación con el personal, etc
- La unidad principal es el *build*: el conjunto de pasos para desplegar una aplicación software
  - Un *build* se pueden disparar manualmente, por un commit, o con planificación periódica similar a cron
  - Un *build* se organiza en *pipelines*, cada una compuesta de *steps*

```
Jenkinsfile (Declarative Pipeline)
pipeline {
    agent any
    stages {
        stage('Build') {
            steps {
                echo 'Building'
            }
        }
        stage('Test') {
            steps {
                echo 'Testing'
            }
        }
        stage('Deploy') {
            steps {
                echo 'Deploying'
            }
        }
    }
}
```

Ansible

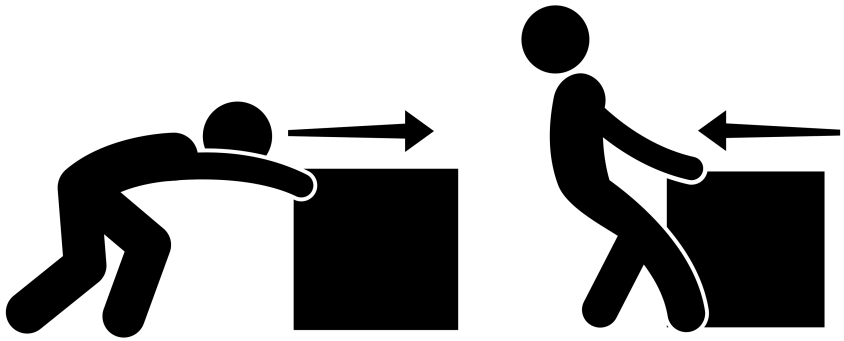


`https://www.ansible.com`

Ansible es un software de gestión de configuraciones

- Creado por Michael DeHaan en 2012. En la actualidad pertenece a RedHat
- software libre, muy popular
- Arquitectura cliente servidor

- Los clientes se denominan *nodos*. Son las máquinas controladas. Solo necesitan un servidor de ssh, por tanto soporta cualquier Linux, Unix, macOS. También funciona sobre la PowerShell de Microsoft Windows
- El servidor se denomina *controlling machine*. Es la máquina que administra y controla los nodos. El soporte nativo es para Linux. Hay versiones para macOS y otras plataformas, prácticamente cualquiera donde funcione python y pip



Push Pull

Iconos: www.vecteezy.com

- Ansible sigue un paradigma *push*: el controlador envía las órdenes
- Otras herramientas similares como *puppet* tienen un enfoque *pull*, que resulta más complejo: la máquina administrada corre un demonio que, periódicamente, se *trae* las órdenes

Conceptos principales en Ansible

- *Inventory*
  Fichero que contiene el listado de las máquinas administradas, con su nombre y/o dirección IP, puerto, claves ssh, etc
- *Playbook*
  Es una especie de *howto* automatizable, un script de propósito específica (configurar una máquina), de alto nivel y fácilmente legible por humanos
  - Palabra inglesa que significa libro de juego, libro de tácticas o libro de reglas
  - Escrito en formato YAML, similar a JSON pero con sintaxis pensada para que sea cómodo para las personas. Filosofía análoga al formato *markdown*, pero para datos, no para texto
- *Role*
  Estructura de nivel superior al *playbook*. Permite hacer plantillas de *playbooks*. Está formado por *playbooks*, ficheros, dependencias entre *playbooks*...

Ansible Galaxy

Repositorio centralizado de roles. Facilita la instalación de software. Equivalente a tener un repositorio de libros de instrucciones, pero que se autoejecutan

`https://galaxy.ansible.com`

```
# Configurar un servidor web básico con  nginx
- name: Configurar servidor web con nginx
  hosts: miServidor01
  sudo: yes
  tasks:
    - name: install nginx
      apt: name=nginx update_cache=yes

    - name: copy nginx conf file
      copy: src=files/nginx.conf dest=/etc/nginx/nginx.conf

    - name: copy nginx server file
      copy: src=files/server.conf dest=/etc/nginx/sites-available/default

    - name: enable config
      file: >
        dest=/etc/nginx/conf.d/default
        src=/etc/nginx/sites-available/default
        state=link
    - name: copy index.html
      template: src=templates/index.html.j2 dest=/usr/share/nginx/html/index.ht

    - name: restart nginx
      service: name=nginx state=restarted enabled=yes
```

- El guión denota un elemento de una lista (una tarea, (*task*)
- Cada tarea suele estar compuesta por varias líneas.
  La primera suele ser el nombretiene un nombre (opcional)
  A continuación, la orden
  p.e.
  `apt: name=nginx update_cache=yes`
  ejecutará en cada nodo
  `apt update; apt install -y nginx`

Referencias

[1] *DevOps for Developers*
Michael Huttermann. Ed. Apress, 2012

http:

//proquest.safaribooksonline.com/book/software-engineering-and-development/9781430245698

[2] *Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale*
Jennifer Davis, Katherine Daniels. Ed. O'Reilly, 2016

http:

//proquest.safaribooksonline.com/book/software-engineering-and-development/9781491926291

[3] *DevOps for Web Development*
Mitesh Soni. Ed. Pack, 2016.

http://proquest.safaribooksonline.com/book/web-design-and-development/9781786465702

[4] *DevOps: A Software Architect's Perspective*
Len Bass, Ingo Weber, Liming Zhu. Ed. Pearson, 2015

http:

//proquest.safaribooksonline.com/book/software-engineering-and-development/9780134049885

[5] *Kanban in Action*
Marcus Hammarberg, Joakim Sunden. Ed. Manning, 2014

`http://proquest.safaribooksonline.com/book/software-engineering-and-development/agile-development/9781617291050`

[6] *The Elements of Scrum*
Chris Sims, Hillary Louise Johnson. Ed. Dymaxicon, 2011